nextnano

Software for semiconductor nanodevices

nextnano++ Documentation

Release 1.25.13

nextnano GmbH

Jul 01, 2025

GETTING STARTED

The *nextnano*++ tool is a Schrödinger-Poisson-current solver and simulates quantum wells, quantum wires, quantum dots, . . .

Features of *nextnano*++ include:

- includes group IV materials (Si, Ge, SiGe) and all III-V materials, its ternaries and quaternaries;
- the nitrides are available in the zinc blende and wurtzite crystal structure
- flexible structures and geometries (1D, 2D and 3D)
- fully quantum mechanical electronic structure, based on the 8-band $\mathbf{k} \cdot \mathbf{p}$ model
- strain, piezo- and pyroelectric charges
- growth directions along [001], [011], [111], [211], . . . in short along any crystallographic direction
- equilibrium and non-equilibrium, calculation of current close to equilibrium (semi-classical)
- magnetic fields

This tool is documented in following sections:

CHAPTER

ONE

OVERVIEW

1.1 Running

The *nextnano++* tool is a console application that is run from within *nextnanomat*. Alternatively, it can be executed from the command line (*Command Line*). The input file specifies the device that shall be simulated.

1.2 Input file

The input file specifies all properties of the device, such as geometry, material composition, grid, contacts,… Furthermore, it sets all parameters that are needed to define the program flow of *nextnano++*. The keywords that can be used for this purpose are defined in the syntax (*Input Syntax*) of the input file.

1.3 Output

The *nextnano++* tool exports its results to a directory and in a certain format that have to be specified in the section (*Simulation Output*) of the input file.

1.4 Examples

The *nextnano++* installation provides some example input files (*Tutorials*) (`C:\Program Files\nextnano\2020_12_09\Sample files\nextnano++ sample files`) that can be run with *nextnanomat*, to get familiar with the program.

1.5 Material database

All material properties that are needed for simulation are specified as material parameters in database files (*database{ }*), which are provided with the *nextnano++* installation. The database covers a large amount of *Zincblende-related …zb{} groups in database{ }* (all III-V and diamond-type like Si, Ge, …), *Wurtzite-related …wz{} groups in database{ }* (GaN, AlN, InN, …) materials, and their alloys.

CHAPTER

TWO

MODELS

Some of models implemented in *nextnano*++.

2.1 Crystal coordinate systems

For zinc-blende materials there are three-digit Miller indices. The Miller indices define a **plane**. There exists a vector that is perpendicular to this plane, e.g. in zinc blende blende materials, the [hkl] **vector** is always perpendicular to the (hkl) **plane**. However, for wurtzite, this is not necessarily true. For instance, although the [0001] vector is perpendicular to the (0001) plane, in general is does not hold that the vector that is perpendicular to the (hkil) plane is defined by [hkil]. Note: For a 1D simulation, the heterostructure is always grown along the x axis. For a 2D simulation, always the (x,y) plane is used.

Zinc blende

```
crystal_zb{
   x_hkl = [1, 0, 0]    # Specify (hkl) plane perpendicular to x axis
   y_hkl = [0, 1, 0]    # Specify (hkl) plane perpendicular to y axis
```

The x axis of the simulation coordinate system is perpendicular to this (hkl) plane of the crystal, here: (1 0 0). The y axis of the simulation coordinate system is perpendicular to this (hkl) plane of the crystal, here: (0 1 0). The Miller indices (here: (0 0 1)) for the z axis are determined automatically. For zinc blende it holds: The vector [hkl] is perpendicular to the (hkl) plane.

Another example:

```
crystal_zb{
   x_hkl = [3,  1, 1]   #
   y_hkl = [0, -1, 1]   #
```

x axis of simulation coordinate system is perpendicular to (3 1 1) plane of crystal coordinate system, i.e. the x axis is along [311] direction. y axis of simulation coordinate system is perpendicular to (0 -1 1) plane of crystal coordinate system, i.e. the y axis is along [0-11] direction. The Miller indices (here: [2, -3, -3]) for the z axis are determined automatically, i.e. (2 -3 -3) plane, i.e. the z axis is along [311] direction.

Wurtzite

Usually for wurtzite, the four-digit Miller-Bravais indices (h k i l) are used. We also use this notation but omit the ‘i’ because i = - h - k. The three integer values (Miller indices) that are given for `x_hkl` refer to a plane and not to a direction. The x direction is then the one that is perpendicular to this plane.

This vector along the x axis has indices that are in general not identical to the Miller indices in wurtzite.

```
crystal_wz{                 # e.g. hexagonal [0001] axis along x axis
   x_hkl = [ 0, 0, 1]    # Specify (hkil) plane perpendicular to x axis: (␣
↪0, 0,  0, 1)
   y_hkl = [ 1, 0, 0]    # Specify (hkil) plane perpendicular to y axis: (␣
↪1, 0, -1, 0)
```

This corresponds to the four-digit Miller-Bravais indices hkil = (0, 0, 0, 1) that define the (hkil)=(0001) plane. Coincidently, the vector [0001] is pependicular to it. This corresponds to the four-digit Miller-Bravais indices hkil = (1, 0, -1, 0) that define the (hkil)=(10-10) plane. The Miller-Bravais indices for the (hkil) plane perpendicular to the z axis are determined automatically inside the code (here: (-1 2 -1 0)).

Another example:

```
crystal_wz{                # e.g. (10-10) plane is perpendicular to x axis
   x_hkl = [ 1, 0, 0]      # hkil = ( 1, 0, -1, 0)
   y_hkl = [-1, 2, 0]      # hkil = (-1, 2, -1, 0)
```

This corresponds to the four-digit Miller-Bravais indices hkil = (1, 0, -1, 0) that define the (hkil)=(10-10) plane. The x axis of the simulation coordinate system is perpendicular to this plane. This corresponds to the four-digit Miller-Bravais indices hkil = (-1, 2, -1, 0) that define the (hkil)=(-12-10) plane. The y axis of the simulation coordinate system is perpendicular to this plane. The Miller-Bravais indices of the (hkil) plane perpendicular to the z axis are determined automatically inside the code (here: (0001)). Coincidently, the vector [0001] is pependicular to it. In this particular case, no rotation has to be applied to the crystal (rotation matrix = identity matrix).

```
crystal_wz{
   ...
   rotation_c_a_ratio_use_substrate = yes  # (default: yes)
   rotation_c_a_ratio = 1.63299            # c/a ratio
```

In wurtzite, the c/a ratio of the two lattice constants ‘c’ and ‘a’ is important. The ideal one, $c/a = \sqrt{8/3} = 1.63299...$, is not the one present in GaN, AlN or InN, i.e. in real materials. For the rotation of the crystal coordinate system to the simulation coordinate system, a specific c/a ratio has to be assumed. By default, we use the one of the substrate material. If you want to use the ideal c/a ratio, you have to specify `rotation_c_a_ratio_use_substrate = no`.

Additionally, one can specify a custom value for the c/a ratio. If no customized value is specified, `rotation_c_a_ratio = sqrt(8/3)` (default). The actually used rotation matrix is written to the log file.

```
x_hkl = [ ., ., .]
y_hkl = [ ., ., .]
z_hkl = [ ., ., .]
```

Exactly two of these three axes have to be specified, the third one is calculated internally.

2.2 Introduction to strain calculation

Here we introduce the theoretical background of the strain and stress calculation in *nextnano++*. At first we will describe the definition of a strain tensor ε and stress tensor σ and then describe the basis of strain tensor calculation in *nextnano++*. A strain tensor is used to calculate the shifts and splittings of band-edge energies and piezoelectric charges.

The detailed explanation for the syntax in `strain{ }` is here: *strain{ }*.

Table of contents

2.2.1 Strain tensor ε

The calculation of strain effects in *nextnano++* is based on linear continuum elasticity theory, in which a crystal can be described by a field of material points with coordinates $\mathbf{x}$. A distortion of the crystal shifts any point to a new position $\mathbf{x}' = \mathbf{x}'(\mathbf{x})$. A field of displacement vectors $\mathbf{u}$ is defined as the devision between the new position and the original position:

$$\mathbf{u}(\mathbf{x}) := \mathbf{x}'(\mathbf{x}) - \mathbf{x}$$

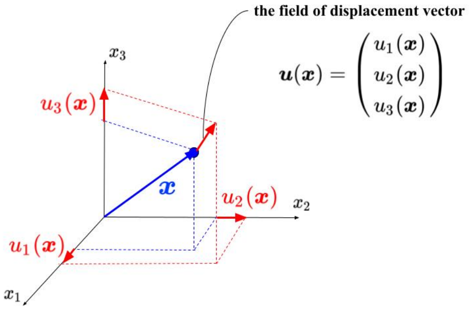


Figure 2.2.1.1: The field of displacement vector $\mathbf{u}$ at $\mathbf{x}$. This is the vector along which the point that was at the position $\mathbf{x}$ moved through the displacement.

A strain tensor ε is defined using this displacement vector:

$$\varepsilon_{ij} := \frac{1}{2}\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right]; \quad (i, j = 1, 2, 3)$$

Strain is dimensionless. The diagonal elements of this strain tensor ε_{ii} represents the length changes per unit length in x_i-direction as described in Figure 2.2.1.2.

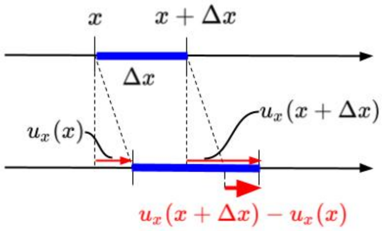


Figure 2.2.1.2: Deformation of a dilatable string in an unstrained (top) and strained state (bottom). We can see the diagonal element $\varepsilon_{ii} = \frac{\partial u_i}{\partial x_i}$ represents the length changes per unit length in x_i-direction.

The off-diagonal elements $\varepsilon_{ij(i \neq j)}$ arise due to shear deformations of the crystal. Figure 2.2.1.3 shows the deformation of an infinitesimal rectangle in $x_1 x_2$ plane. We can see $\frac{\partial u_2}{\partial x_1} = \frac{u_2(x_1+\Delta x_1, x_2) - u_2(x_1, x_2)}{\Delta x} = \sin\alpha \simeq \alpha$ and $\frac{\partial u_1}{\partial x_2} = \frac{u_1(x_1, x_2+\Delta x_2) - u_1(x_1, x_2)}{\Delta x_2} = \sin\beta \simeq \beta$. In these angle changes, $\frac{\alpha-\beta}{2}$ corresponds to a pure solid-body rotation and $\frac{\alpha+\beta}{2} = \frac{1}{2}\left[\frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\right] = \varepsilon_{12}$ measures the shear strain.

By definition strain tensor ε is symmetric (i.e. $\varepsilon_{ij} = \varepsilon_{ji}$) so the number of components that must be specified is actually 6. Voigt notation is the useful convention in which these 6 independent components are written in form of a 6×1 matrix for short. This notation reads:

$$11 \to 1, 22 \to 2, 33 \to 3, 23 \to 4, 31 \to 5, 12 \to 6$$

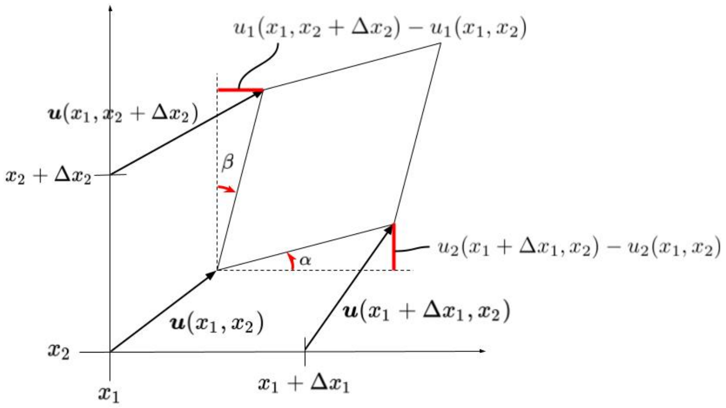


Figure 2.2.1.3: Deformation of an infinitesimal rectangle in a strained state.

and

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{bmatrix} = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{23} \\ 2\varepsilon_{13} \\ 2\varepsilon_{12} \end{bmatrix}$$

2.2.2 Stress tensor σ

A stress tensor component σ_{ij} represents the force towards x_j-direction acting on infinitsimal area that is perpendicular to x_i-direction. Its unit is the same with pressure ($[\mathrm{Pa}] = [\mathrm{N}/\mathrm{m}^2]$).

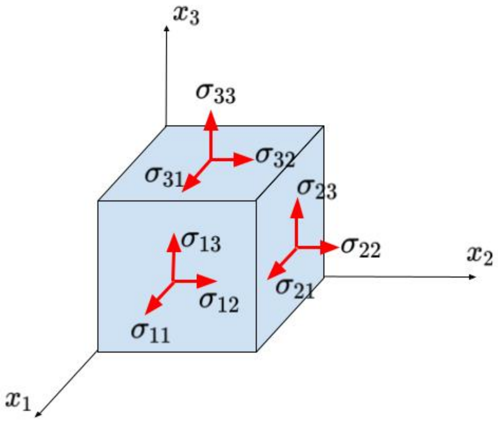


Figure 2.2.2.1: The components of stress tensor σ.

In linear approximation, this stress tensor is related to the strain tensor ε by means of Hook's law:

$$\sigma_{ij} = \sum_{kl} C_{ijkl} \varepsilon_{kl}$$

where C_{ijkl} is the component of eleasticity stiffness tensor, which is the forth-order tensor comprising $3^4 = 81$ components. It's dimension is the same with stress tensor components and defined as $[\mathrm{GPa}]$ in *nextnano++*. In

Voigt notation, C is the form of a 6×6 matrix by putting $C_{ijkl} = C_{mn}$ $(i, j, k, l = 1, 2, 3,\ m, n = 1, ...6)$. Then the Hook's law reads

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{bmatrix}$$

For many crystal structures with high symmetry, many of these coefficients are 0 and some are related to others. The elasticity tensor of zincblende and wurtzite crystals are given by

$$C_{zb} = \begin{bmatrix} C_{11} & C_{12} & C_{12} & & & \\ C_{12} & C_{11} & C_{12} & & & \\ C_{12} & C_{12} & C_{11} & & & \\ & & & C_{44} & & \\ & & & & C_{44} & \\ & & & & & C_{44} \end{bmatrix}$$

$$C_{wz} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & & & \\ C_{12} & C_{11} & C_{13} & & & \\ C_{13} & C_{13} & C_{33} & & & \\ & & & C_{44} & & \\ & & & & C_{44} & \\ & & & & & C_{66} \end{bmatrix}$$

with $C_{66} = \frac{1}{2}[C_{11} - C_{22}]$ in wurtzite.

These constants are defined in *database_nnp.in*. You can also overwrite these values in your input file.

- For zinc-blend materials, for example:

```
database{
    binary_zb{
        name = GaAs
        valence = III_V

        elastic_consts{
            c11 = 122.1                     # [GPa] elastic constants
            c12 = 56.6                      # 1 * 1011 dyn/cm2 = 10 GPa  ->␣
↪ 12.21 * 1011 dyn/cm2 = 122.1 GPa
            c44 = 60.0                      # The elastic constants are␣
↪needed for the calculation of the strain in heterostructures.
        }
    }
}
```

- For wurtzite materials, for example:

```
database{
    binary_zb{
        name = GaN
        valence = III_V

        elastic_consts{
            c11 = 390                       # [GPa] elastic constants
            c12 = 145                       # 1 * 1011 dyn/cm2 = 10 GPa  ->␣
↪ 39.0 * 1011 dyn/cm2 = 390 GPa
            c13 = 106                       #
```

(continues on next page)

(continued from previous page)

```
            c33 = 398                    #
            c44 = 105                    # The elastic constants are␣
↪needed for the calculation of the strain in heterostructures.
        }
    }
}
```

2.2.3 Strain and stress calculation

Next we will describe how the strain tensor ε and stress tensor σ are determined in general. Then the two types of calculation implemented in *nextnano*++ are introduced briefly.

In general

The principle of conservation of linear momentum results in the following equations of stress tensor components for $i = 1, 2, 3$:

$$\sum_{j=1}^{3} \frac{\partial \sigma_{ji}}{\partial x_j} + f_i = 0$$

where $\mathbf{f}$ is the body force such as gravity. When the boundary conditions are specified, the field of displacement vector $\mathbf{u}$, by which the stress tensor components σ_{ij} are eventually written, is determined according to these simultaneous differential equations. Then the strain tensor ε and stress tensor σ are also determined from $\mathbf{u}$.

Note

The principle of conservation of angular momentum, on the other hand, results in the symmetricity of stress tensor: $\sigma_{ij} = \sigma_{ji}$

The field of displacement vector which satisfies the above balance equations and boundary conditions also minimizes the total potential energy $U + V_E$ where U is the elastic strain energy and V_E is the potential energy associated with the body force $\mathbf{f}$. This is so called minimum total potential energy principle.

In the linear approximation regime, the elastic energy stored in the whole body is:

$$U = \frac{1}{2} \int_V C_{ijkl} \varepsilon_{ij} \varepsilon_{kl} \, dV$$

When the body force $\mathbf{f}$ is assumed to be zero throughout the system, solving the above differential equations is equivalent to find the strain tensor that minimizes this elastic energy U.

In nextnano++

There are two kinds of calculation of strain, `pseudomorphic_strain{ }` and `minimized_strain{ }`, in *nextnano*++. In both of implementations pseudomorphic layer is assumed as the boundary condition between the substrate and the layer grown on this substrate. The substrate is assumed to be so thick that the in-plane lattice constants of the layer is matched to that of substrate. Also, the body force $\mathbf{f}$ is assumed to be 0 throughout the structure.

In this assumption, the analytic expressions for strain tensor that satisfies the aforementioned stress balance equations (i.e. that minimizes the elastic energy) can be found for 1D structures. This analytic solution is implemented on `pseudomorphic_strain{ }`. This feature also works in 2D or 3D but the user must be sure that the model makes sense from a physical point of view (i.e. the 2D/3D structure should consist of different layers along the growth direction whereas the layers must be homogenous along the two perpendicular directions).

On the other hand, `minimized_strain{ }` calculates the strain tensor by minimizing the elastic energy mentioned before. This can also be used for 1D simulations. In this case, the results will be equivalent to the analytical model `pseudomorphic_strain{ }`.

The detailed explanation for the syntax in `strain{ }` is here: *strain{ }*. Please refer to *[AndlauerPhD2009]* for more details about these topics.

Last update: nnnn/nn/nn

2.3 Piezoelectricity in wurtzite

The *nextnano*++ tool can simulate growth orientation dependence of the piezoelectric effect in heterostructures. Following A.E. Romanov et al., Journal of Applied Physics 100, 023522 (2006), we consider $In_xGa_{1-x}N$ and $Al_xGa_{1-x}N$ thin layers pseudomorphically grown on GaN substrates. The c-axis of the substrate GaN is inclined by an angle θ with respect to the interface of the heterostructure.

The layer is assumed to be very thin compared to substrate so that the strain is approximately homogeneous in all direction (pseudomorphic), and the ternary alloys mimic the orientation of crystallography direction. The layer material deforms such that the lattice translation vector of each layer has a common projection onto the interface.

The strain in a crystal induces piezoelectric polarization, which contributes as an additional component to the total charge density profile. The important consequence of their analysis is that the piezoelectric polarization normal to the interface becomes zero at a nontrivial angle. The piezoelectric charge in a heterostructure in general results in an additional offset between electron and hole spatial probability distribution, thereby reducing the overlap of their wave functions in real space. The small overlap of electron and hole leads to an inefficient radiative recombination, i.e. lower efficiency of optoelectronic devices. The work by Romanov et al. paved the way to device optimization by the growth direction of the crystal.

An introduction for the strain calculation is described here: *Introduction to strain calculation*

Table of contents

References

- A.E. Romanov, T.J. Baker, S. Nakamura, and J.S. Speck, Journal of Applied Physics **100**, 023522 (2006)
- S. Schulz and O. Marquardt, Phys. Rev. Appl. **3**, 064020 (2015)
- S.K. Patra and S. Schulz, Phys. Rev. B **96**, 155307 (2017)

The corresponding input files are located in the *nextnano*++ sample files folder:

- *Romanov_InGaN_theta_nnp.in*
- *Romanov_AlGaN_theta_nnp.in*
- *Romanov_InGaN_theta_nnp_2nd.in*
- *Romanov_InGaN_theta_nn3.in*

- *Romanov_InGaN_theta_nn3_2nd.in*

2.3.1 Specify crystal orientation

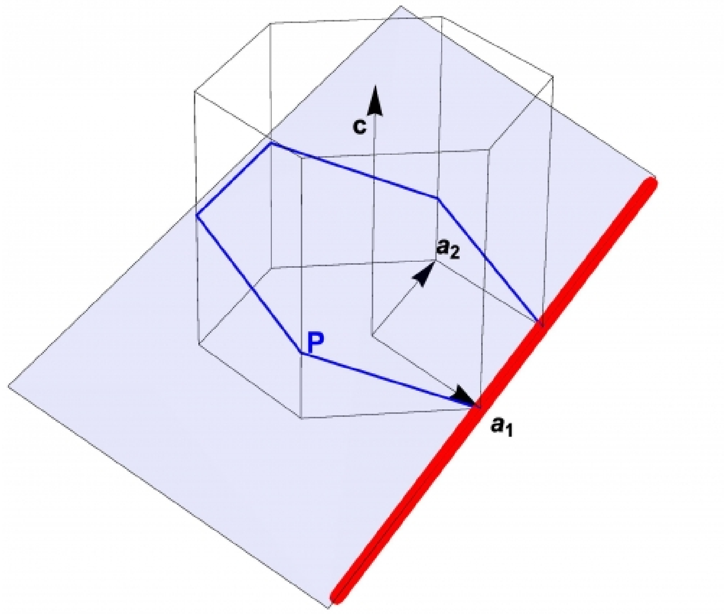


Figure 2.3.1.1: Rotation of a wurtzite structure. The blue plane is parallel to the interface.

The nextnano software treats the rotation of crystal orientation by the **Miller-Bravais indices** in the input file. The setup of our system is as follows: the x-axis of the simulation coordinate system (hereafter **x'**-axis) is taken to the normal vector of the interface. The z-axis of the simulation system (**z'**) is normal to the (-1 2 -1 0) plane of the crystal, i.e. it is along $\mathbf{a}_2$ direction in Figure 2.3.1.1. The rotation axis indicated with red line is along **z'**-axis, and the interface is shown as the blue plane. The inclination angle θ is defined as the angle between the c-axis [0001] and the normal vector of the blue plane, which is **x'**-axis.

Then the crystal orientation is specified in *nextnano++* input file as

```
crystal_wz{
    x_hkl = [ 1, 0, l(theta)] # x axis perpendicular to (hkl) plane = (hkil) plane
    z_hkl = [-1, 2, 0]        # z axis perpendicular to (hkl) plane = (hkil) plane
}
```

where $l(\theta)$ is an integer determined by the inclination angle. This statement means *the x'-axis is normal to the (1 0 -1* $l(\theta)$*) plane of the crystal, whereas z'-axis is normal to the (-1 2 -1 0) plane*. (Note that *nextnano++* does not require the third entry, i.e. the letter `i`, in Miller-Bravais notation (hkil) because `i=-(h+k)`.)

The index $l(\theta)$ is deduced from a simple geometry consideration. Figure 2.3.1.2 shows the cross-section of a wurtzite lattice that is perpendicular to the rotation axis in Figure 2.3.1.1.

- When $\theta = 0$, the interface is normal to the (0001) plane, i.e. **x'**-axis is normal to the (0001) plane.
- When $\theta = 90$ degree, the **x'**-axis should be normal to the (1 0 -1 0) plane of the crystal.
- When $0 < \theta < 90$ degree, definition of the index is $l(\theta) := \frac{c}{d}$ and the following relation holds

$$d = \frac{\sqrt{3}}{2} a \tan\theta.$$

From these equations we find

$$l(\theta) = \frac{2c}{\sqrt{3} a \tan\theta}.$$

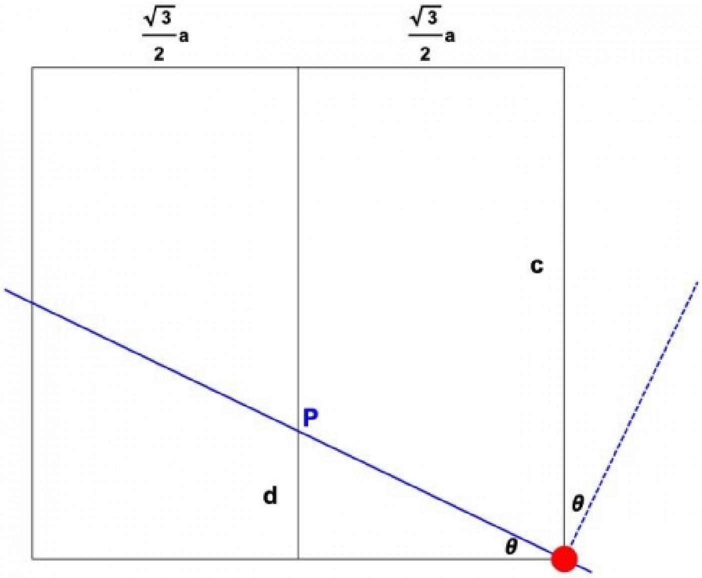


Figure 2.3.1.2: Cross-section of the wurtzite lattice. The dashed blue line indicates the **x'**-direction, which is normal to the interface (solid blue line).

The plane to be determined can be then taken as

$$(hkil) = (\sin\theta \; 0 \; \; -\sin\theta \; \frac{2c}{\sqrt{3}a}\cos\theta)$$

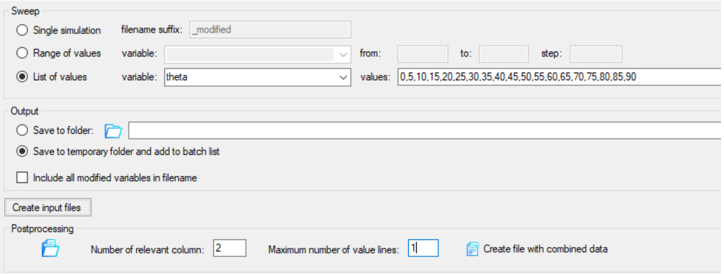
We note that the expression in the third case includes the other two special cases. To approximate the direction with integer entries, we multiply 100 and take the floor function:

```
$gamma = $c_InGaN / $a_InGaN # c/a ratio
# ideal c/a ratio in wurtzite is SQRT(8/3)=1.63299
$h = floor(100*sin(theta))
$l = floor(100*2*gamma*cos(theta)/sqrt(3))
x_hkl = [$h, 0, $l]  # x axis perpendicular to (hkl) plane = (hkil) plane
```

2.3.2 Parameter sweep of the angle using Template: Sweep over the variable theta

- Input file: *Romanov_InGaN_theta_nnp.in*

One can make use of **'Template'** feature of *nextnanomat* to sweep the angle θ and obtain crystal orientation dependence of several physical quantities. Here, calculation is performed for every 5 degrees.



Sweep
○ Single simulation filename suffix: _modified
○ Range of values variable: from: to: step:
● List of values variable: theta values: 0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90

Output
○ Save to folder:
● Save to temporary folder and add to batch list
☐ Include all modified variables in filename

Create input files

Postprocessing
Number of relevant column: 2 Maximum number of value lines: 1 Create file with combined data

We obtain the angle dependence using **'post-processing'** feature. Here, we collect the strain tensor components ε_{xx}, ε_{yy}, ε_{zz}, ε_{xy}, ε_{xz} and ε_{yz} that are in columns 2, 3, 4, 5, 6, 7 of the file `strain_simulation.dat`.

- Select file containing values for the strain tensor components `strain_simulation.dat` by clicking on the folder icon below *post-processing*.

- Select 1 for the *Maximum number of values lines.*
- Select 2 for the *Number of relevant column. (to do: Improve nextnanomat to include all columns.)*
- Click on *Create file with combined data* to generate file `theta_strain_simulation_Column2.dat`.
- Select 3 for the *Number of relevant column.*
- Click on *Create file with combined data* to generate file `theta_strain_simulation_Column3.dat`.
- Select 4 for the *Number of relevant column.*
- Click on *Create file with combined data* to generate file `theta_strain_simulation_Column4.dat`.
- Select 5 for the *Number of relevant column.*
- Click on *Create file with combined data* to generate file `theta_strain_simulation_Column5.dat`.
- Select 6 for the *Number of relevant column.*
- Click on *Create file with combined data* to generate file `theta_strain_simulation_Column6.dat`.
- The post-processing results are contained in the folder `<name_of_input_file>_postprocessing`.
- Finally, the plotted results of the post-processing file can be exported to gnuplot. Add all columns to the Overlay, and then click on: *Create and Open Gnuplot (*.plt) from Items of Overlay*

2.3.3 Strain

Figure 2.3.3.1 and Figure 2.3.3.2 are the strain tensor elements as a function of inclination angle θ, with respect to **simulation** and **crystal** coordinate systems, respectively. One can confirm that they reproduce correctly Figure 5 and 6 in [Romanov2006]. Please note that Romanov takes **z'**-axis as growth direction, while we take **x'**-axis. Therefore **x'**- and **z'**-axes are interchanged from [Romanov2006].

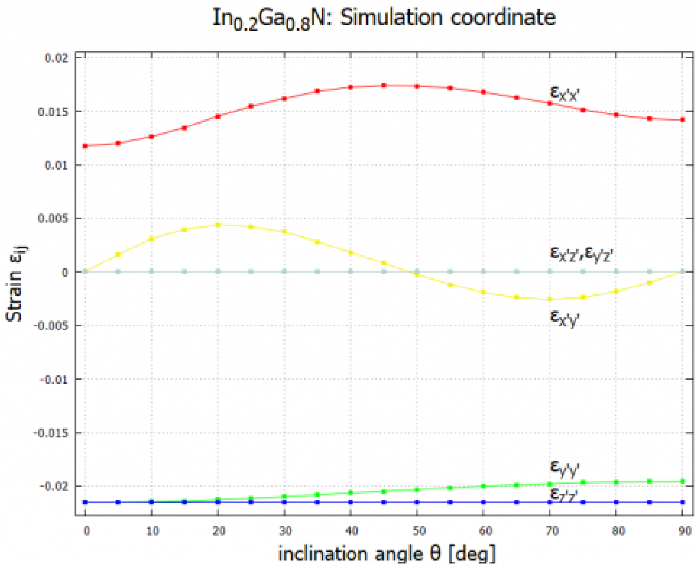


Figure 2.3.3.1: Elastic strain tensor components as a function of c-axis inclination angle θ in **simulation** coordinate system.

2.3.4 Piezoelectric effect (first-order)

The piezoelectric effect is at first instance described by a linear response against strain. In crystal coordinate system,

$$P_{\mu}^{(1)} = \sum_{j=1}^{6} e_{\mu j} \epsilon_j,$$

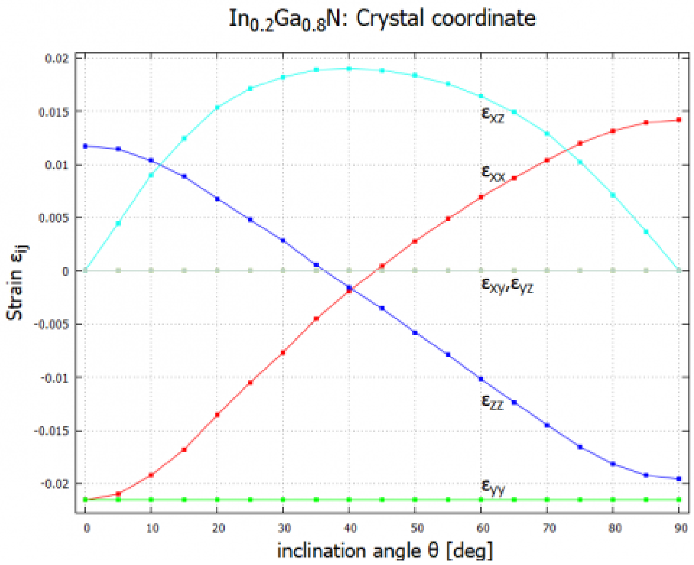


Figure 2.3.3.2: Elastic strain tensor components as a function of c-axis inclination angle θ in **crystal** coordinate system.

where $\mu = 1, 2, 3$ and the strain tensor is expressed in six-dimensional Voigt notation

$$\begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{pmatrix} = \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ 2\epsilon_{yz} \\ 2\epsilon_{xz} \\ 2\epsilon_{xy} \end{pmatrix}.$$

Please note that the indices x, y, z without prime refer to the axes of the crystal coordinate system. The superscript $^{(1)}$ indicates first-order piezoeffect. For the symmetry of the wurtzite structure, only three parameters remain in the piezoelectric coefficient tensor e_{ij}

$$\begin{pmatrix} P_x^{(1)} \\ P_y^{(1)} \\ P_z^{(1)} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & e_{15} & 0 \\ 0 & 0 & 0 & e_{15} & 0 & 0 \\ e_{31} & e_{31} & e_{33} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ 2\epsilon_{yz} \\ 2\epsilon_{xz} \\ 2\epsilon_{xy} \end{pmatrix} = \begin{pmatrix} 2e_{15}\epsilon_{xz} \\ 2e_{15}\epsilon_{yz} \\ e_{31}(\epsilon_{xx} + \epsilon_{yy}) + e_{33}\epsilon_{zz} \end{pmatrix},$$

cf. Eq. (4) in [Schulz2015]. **Note that Eq. (14) in [Romanov2006] misses the factor 2 for off-diagonal elements of the strain tensor.** These equations are implemented with corresponding material parameters in the database. The following flags export the strain tensor components and piezoelectric polarization vector in **crystal** and **simulation** coordinate systems (see *nextnano++*). The piezoelectric polarization vector with respect to the simulation coordinate system can be found in the file `Strain\piezoelectric_polarization_vector_simulation.dat`.

```
strain{
    output_strain_tensor{
        crystal_system    = yes
        simulation_system = yes
    }

    output_polarization_vector{
        crystal_system    = yes
        simulation_system = yes
    }

```

(continues on next page)

(continued from previous page)

```
    output_polarization_vector_components{
        crystal_system    = yes
        simulation_system = yes
    }
}
```

For consistency, we have used the same material parameters as [Romanov2006], i.e. we have overwritten our default material parameters of the database with the values specified in the input file.

Analytical expression is derived as follows [Schulz2015]. Since we are interested in the polarization normal to the interface, it is useful to switch to the simulation coordinate system (x', y', z'). This can be done by transforming the polarization vector and the strain tensor to the simulation system,

$$P^{(1)}_{\mu'} = \left(RP^{(1)}\right)_{\mu'} = \sum_{\mu=1}^{3} R_{\mu'\mu} P^{(1)}_{\mu}, \;\; \epsilon_{\mu'\nu'} = \left(R\epsilon R^{-1}\right)_{\mu'\nu'} = \sum_{\mu,\nu=1}^{3} R_{\mu'\mu} R_{\nu'\nu} \epsilon_{\mu\nu},$$

where the 3×3 rotation matrix R accounts for a rotation of angle θ

and we have used the fact that the rotation matrix is orthogonal: $(R^{-1})_{\mu\nu} = R_{\nu\mu}$. Prime denotes the axes in simulation coordinate system. These equations can be expressed in vector form as

$$\begin{pmatrix} P^{(1)}_x \\ P^{(1)}_y \\ P^{(1)}_z \end{pmatrix} = R^{-1}(\theta) \begin{pmatrix} P^{(1)}_{x'} \\ P^{(1)}_{y'} \\ P^{(1)}_{z'} \end{pmatrix}, \quad \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ 2\epsilon_{yz} \\ 2\epsilon_{xz} \\ 2\epsilon_{xy} \end{pmatrix} = S^{-1}(\theta) \begin{pmatrix} \epsilon_{x'x'} \\ \epsilon_{y'y'} \\ \epsilon_{z'z'} \\ 2\epsilon_{y'z'} \\ 2\epsilon_{x'z'} \\ 2\epsilon_{x'y'} \end{pmatrix}$$

where $S(\theta)$ is a 6×6 matrix. The second transformation is given in Eq. (13) in [Romanov2006]. From equations above, we obtain the first-order piezoelectric effect in the simulation coordinate system

$$\begin{pmatrix} P^{(1)}_{x'} \\ P^{(1)}_{y'} \\ P^{(1)}_{z'} \end{pmatrix} = R(\theta) \begin{pmatrix} 0 & 0 & 0 & 0 & e_{15} & 0 \\ 0 & 0 & 0 & e_{15} & 0 & 0 \\ e_{31} & e_{31} & e_{33} & 0 & 0 & 0 \end{pmatrix} S^{-1}(\theta) \begin{pmatrix} \epsilon_{x'x'} \\ \epsilon_{y'y'} \\ \epsilon_{z'z'} \\ 2\epsilon_{y'z'} \\ 2\epsilon_{x'z'} \\ 2\epsilon_{x'y'} \end{pmatrix}.$$

The **z'**-component is explicitly

$$\begin{aligned} P^{(1)}_{z'} =& e_{31} \cos\theta \epsilon_{x'x'} \\ &+ \left(e_{31} \cos^3\theta + \frac{e_{33} - 2e_{15}}{2} \sin\theta \sin 2\theta \right) \epsilon_{y'y'} \\ &+ \left(\frac{e_{31} + 2e_{15}}{2} \sin\theta \sin 2\theta + e_{33} \cos^3\theta \right) \epsilon_{z'z'} \\ &+ \left[(e_{31} - e_{33}) \cos\theta \sin 2\theta + 2e_{15} \sin\theta \cos 2\theta \right] \epsilon_{y'z'}. \end{aligned}$$

Note that the corresponding analytical expression Eq. (18) in [Romanov2006] misses the factor 2 in front of e_{15} in the 2nd, 3rd and 4th line, and contains a typo in the 3rd line, i.e. e_{33} has to be e_{31} in the first term. Our expression is consistent to eq. (5) in [Schulz2015]. Figure 2.3.4.1 compares the results of the nextnano software with the results of [Romanov2006] and [Schulz2015], respectively. The analytical results in Figure 2.3.4.1 are the plot of the equation above, with an interchange of **x'**- and **z'**-axes.

From the results in Figure 2.3.4.1 we can see that the piezoelectric polarization vanishes at an intermediate angle around 38 degree and that it is maximized when the inclination angle is zero.

2.3.5 Post-Processing for polarization

We obtain the angle dependence using **'post-processing'** feature. Here, we collect the polarization components P_x that is in column 1 of the file `polarization_vector_piezoelectric_simulation.dat`.

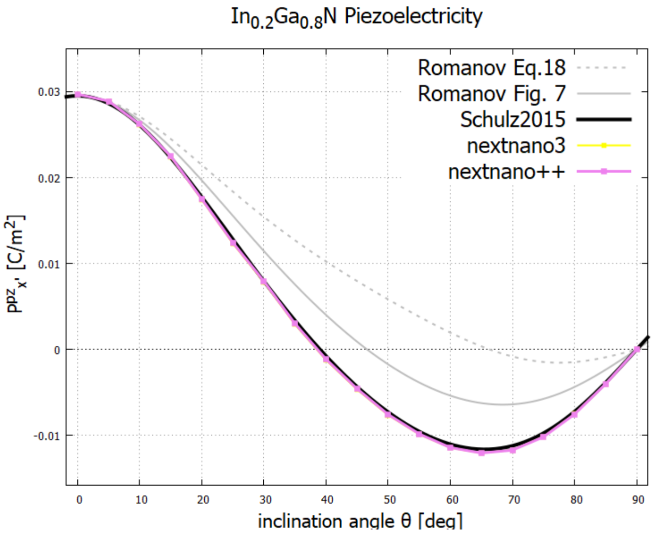


Figure 2.3.4.1: Piezoelectric polarization as a function of inclination angle. The gray dotted curve contains a typo $e_{33} \leftrightarrow e_{31}$ and misses the factor 2. When the first typo is fixed, the gray solid curve is obtained and looks to be consistent with Figure 7(a) in [Romanov2006]. With the factor 2 the result becomes the black curve, consistent to [Schulz2015].

- Select file containing values for the piezoelectric components `polarization_vector_piezoelectric_simulation.dat` by clicking on the folder icon below *post-processing*.
- Select `2` for the *Number of relevant column*.
- Select `1` for the *Maximum number of values lines*.
- Click on *Create file with combined data* to generate file `theta_polarization_vector_piezoelectric_simulation_Colu` `dat`.
- The post-processing results are contained in the folder `<name_of_input_file>_postprocessing`.
- Finally, the plotted results of the post-processing file can be exported to gnuplot. Add all columns to the Overlay, and then click on: *Create and Open Gnuplot (*.plt) from Items of Overlay*

2.3.6 Alloy content dependence

One can also sweep the alloy content x. The following results correspond to Figure 7(a) in [Romanov2006]. One can see that the zero point is universal for different alloy contents. The zero point is different compared to [Romanov2006] as he misses the factor of 2 for the strain tensor component. As can be seen in Figure 2.3.4.1 shown above, this mistake is not relevant for 0 and 90 degrees.

2.3.7 AlGaN

- Input file: *Romanov_AlGaN_theta_nnp.in*

Similarly, piezoelectric polarization of Al$_x$Ga$_{1-x}$N/GaN structure is calculated and shown in Figure 2.3.7.1. This result corresponds to Figure 8(a) in [Romanov2006]. The piezoelectric effect vanishes at around 38 degree in this case as well. Again, the zero point is different compared to [Romanov2006] as he misses the factor of 2 for the strain tensor component. As can be seen in Figure 2.3.4.1 shown above, this mistake is not relevant for 0 and 90 degrees.

The sign of the piezoelectric polarization in Figure 2.3.7.1 is opposite to the case of InGaN/GaN composition (Figure 2.3.6.1). This is due to the fact that the lattice constants of InN, GaN and AlN obey the following relation

$$a_{\mathrm{InN}} > a_{\mathrm{GaN}} > a_{\mathrm{AlN}}$$

(also for c). Since we take GaN as a substitute, In$_x$Ga$_{1-x}$N layer is subject to compressive strain, whereas Al$_x$Ga$_{1-x}$N is under tensile strain [Romanov2006].

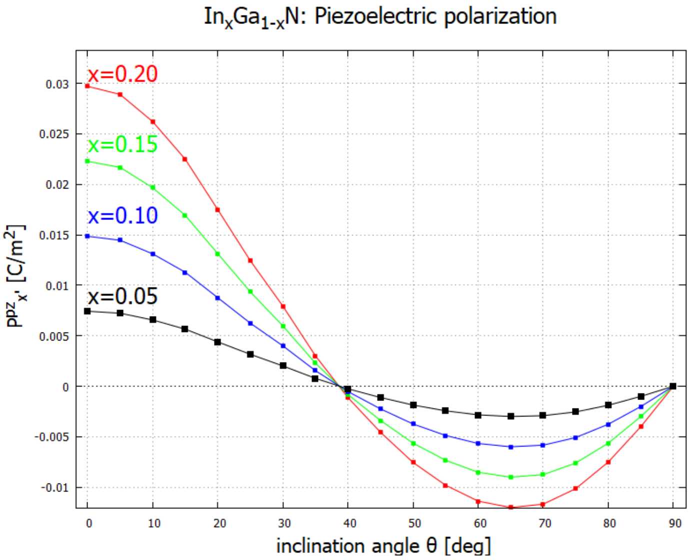


Figure 2.3.6.1: Alloy content dependence of the piezoelectric polarization for $In_xGa_{1-x}N$/GaN structure. $In_xGa_{1-x}N$ is under biaxial compressive strain with respect to GaN.

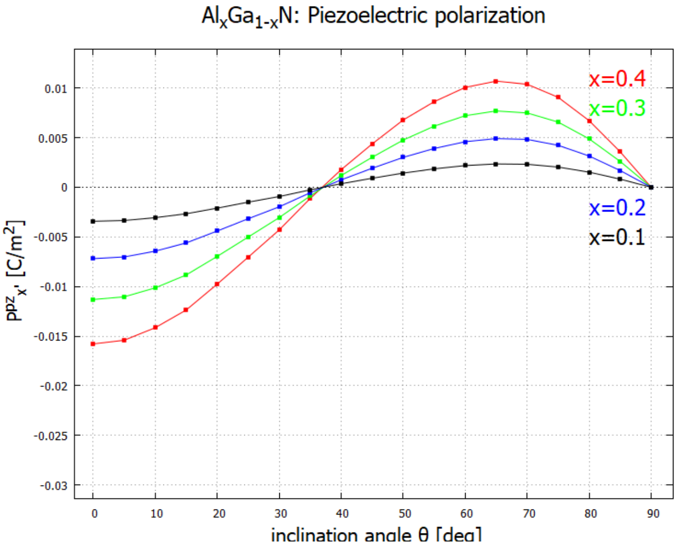


Figure 2.3.7.1: Alloy content dependence of the piezoelectric polarization for $Al_xGa_{1-x}N$/GaN structure. $Al_xGa_{1-x}N$ is under biaxial tensile strain with respect to GaN.

2.3.8 Piezoelectric effect (second-order)

- Input file: *Romanov_InGaN_theta_nnp_2nd.in*

Optimization of optoelectronic device design requires an accurate and detailed knowledge of the growth-direction dependence of the built-in electric field. Recently, the second order piezoelectric effect has been reported to be relevant for wurtzite III-N materials, namely GaN, AlN and InN. This potentially affects the electronic and optical properties of the devices. The piezoelectric polarization is generalized in crystal coordinate as [Patra2017]

$$P_\mu^{\mathrm{pz}} = \sum_{j=1}^{6} e_{\mu j}\epsilon_j + \frac{1}{2}\sum_{j,k=1}^{6} B_{\mu jk}\epsilon_j\epsilon_k + \cdots ,$$

where $e_{\mu j}$ and $B_{\mu jk}$ are first- and second-order piezoelectric coefficients, respectively. For binary wurtzite structure, one can show that $B_{\mu jk}$ has 8 independent components $B_{311}, B_{312}, B_{313}, B_{333}, B_{115}, B_{125}, B_{135}, B_{344}$. The explicit expression of the second-order term is given in Eq. (3) in [Patra2017].

One can turn on the second-order contribution in *nextnano++* as

```
# nextnano++
strain{
    ...
    second_order_piezo = yes        # default: no
}
```

Figure 2.3.8.1 shows the results of the nextnano software. While the second-order contribution becomes negligible between the orientation $(10\bar{1}3)$ and $(10\bar{1}2)$, and also between 85 and 95 degrees, it enhances the piezo effect up to 14% in other directions. This figure can be qualitatively compared to Figure 1(c) in [Patra2017], but note that they consider binary InN/GaN structure there while we are using $\mathrm{In}_{0.2}\mathrm{Ga}_{0.8}\mathrm{N}$/GaN. The pink curve is different from the one in Figure 2.3.4.1 because we employed the material parameters used in [Patra2017].

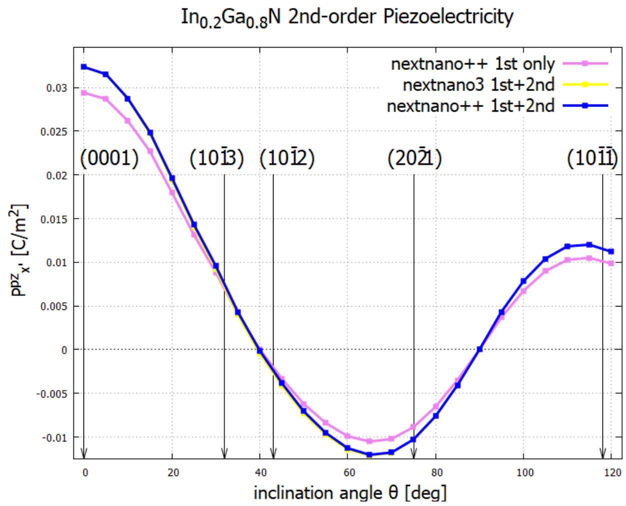


Figure 2.3.8.1: Second-order piezoelectricity. The second-order term enhances the piezoelectric polarization. The *nextnano³* result (yellow) is consistent to the *nextnano++* result (blue). Interface planes are indicated at corresponding angles.

Last update: nnnn/nn/nn

2.4 Electrostatic potential

2.4.1 Poisson Equation

This equation governs the relation between the **electrostatic potential** $\phi(\mathbf{x})$ and **total charge density distribution** $\rho(\mathbf{x},\phi)$ as follows:

$$-\nabla\cdot[\varepsilon_0\varepsilon_r(\mathbf{x})\nabla\cdot\phi(\mathbf{x})]=\rho(\mathbf{x},\phi) \tag{2.4.1.1}$$

where ε_0 is the vacuum permittivity, ε_r is the material dependent static dielectric constant. And the total charge density distribution consists of the **densities of ionized donors** N_D^+, **ionized acceptors** N_D^-, **piezoelectric and pyroelectric charge** ρ_{pz} and ρ_{py}, besides the **carrier densities** $n(\mathbf{x},\phi)$ and $p(\mathbf{x},\phi)$, which are calculated either classically or quantum mechanically:

$$\rho(\mathbf{x},\phi)=e[-n(\mathbf{x},\phi)+p(\mathbf{x},\phi)+N_D^+(x)-N_A^-(\mathbf{x})+\rho_{\mathrm{pz}}(\mathbf{x})+\rho_{\mathrm{py}}(\mathbf{x})] \tag{2.4.1.2}$$

When the Schrödinger-Poisson equation is solved, i.e. `quantum_poisson{ }` is specified in *run{ }* section, the carrier densities defined in either multi-band model or single-band model are substituted into this $\rho(\mathbf{x},\phi)$ and the Poisson equation is solved accordingly. Then the resulting $\phi(\mathbf{x})$ is returned into the Schrödinger equation and the carrier densities are calculated once again.

This cycle is continued until the carrier densities satisfies the convergence criteria, which can be tuned by the users from *run{ poisson{ } }*. The final result of $n(\mathbf{x},\phi)$, $p(\mathbf{x},\phi)$ and $\phi(\mathbf{x})$ must satisfy both Schrödinger and Poisson equations, or we can say that **the Schrödinger equation and Poisson equation are self-consistent with respect to the resulting carrier densities and electrostatic potential**.

On the other hand, when only the Poisson equation is solved, i.e. only `poisson{ }` is specified *run{ }* section, the carrier densities are calculated according to (2.5.1.1) and (2.5.1.2) instead. We can say in other words that the **carrier density calculation in the context of Thomas-Fermi approximation and the Poisson equation are self-consistent with respect to the resulting carrier densities and electrostatic potential**.

Last update: 04/12/2024

2.5 Charge densities

2.5.1 Classical

Things are much more simpler.

When any kind of Schrödinger equation is not solved, the charge carrier densities are estimated from the position-dependent conduction and valence band edges $E_{\mathrm{c}}^i(\mathbf{x})$ and $E_{\mathrm{v}}^i(\mathbf{x})$, quasi-Fermi levels, and the electrostatic potential $\phi(\mathbf{x})$ in the context of Thomas-Fermi approximation.

These **classical charge carrier densities** are calculated as

$$n(\mathbf{x})=\sum_{i\in\mathrm{CB}}N_{\mathrm{c}}^i(T)\,\mathcal{F}_{1/2}\Big([-E_{\mathrm{c}}^i(\mathbf{x})+e\phi(\mathbf{x})+E_{\mathrm{F},n}(\mathbf{x})]/kT\Big) \tag{2.5.1.1}$$

$$p(\mathbf{x})=\sum_{i\in\mathrm{VB}}N_{\mathrm{v}}^i(T)\,\mathcal{F}_{1/2}\Big([E_{\mathrm{v}}^i(\mathbf{x})-e\phi(\mathbf{x})-E_{\mathrm{F},p}(\mathbf{x})]/kT\Big). \tag{2.5.1.2}$$

Here $N_{\mathrm{v}}^i(T)$ and $N_{\mathrm{v}}^i(T)$ are the equivalent density of states at the conduction and valence band edges, which are given by

$$N_l^i(T)=g_l^i\left(\frac{m_{\mathrm{dos},\lambda}^i kT}{2\pi\hbar}\right)^{2/3}\qquad(l,\lambda)=(\mathrm{v,h}),\ \text{or }(\mathrm{c,e}). \tag{2.5.1.3}$$

Here $m_{\mathrm{dos},\lambda}^i$ is the density-of-mass for $d=3$ defined in (2.5.2.10).

This calculation of carrier densities is much faster than the quantm mechanical calculation, but the quantum effect such as energy quantization, carrier leackage into the barrier, etc. cannot be taken into account.

Also in this case, the carrier densities can be written as $n(\mathbf{x},\phi)$ and $p(\mathbf{x},\phi)$, which enters into the non-linear Poisson equation introduced next.

Moreover, when the current equation is included in the calculation scheme, seeing the carrier densities as $n(\mathbf{x},\phi,E_{\mathrm{F},n})$ and $p(\mathbf{x},\phi,E_{\mathrm{F},p})$ makes it easy to understand what the self-consistent calculation is actually doing.

2.5.2 Quantum mechanical

Multi-band model ($\mathbf{k}\cdot\mathbf{p}$ model)

Once the μ-th component envelope function of the j-th eigenstate of electron ($l=\mathrm{c}$) or hole ($l=\mathrm{v}$) in the i-th band is obtained as $(F_\mu)^i_{l,j}(\mathbf{x})$ from the **multi-band Schrödinger equation**, the probability distribution of this j-th eigenstate reads

$$p^i_{l,j}(\mathbf{x})=\sum_\mu\left|(F_\mu)^i_{l,j}(\mathbf{x})\right|^2. \tag{2.5.2.1}$$

where we are assuming 3D structure so far.

Then the **quantum mechanical carrier densities for 3D structure** are defined from these probability densities, energy eigenvalues $E_{\mathrm{c},j}$ and $E_{\mathrm{v},j}$, position-dependent quasi-Fermi levels $E_{\mathrm{F},n}(\mathbf{x})$ and $E_{\mathrm{F},p}(\mathbf{x})$ as

$$n(\mathbf{x})=\sum_{i\in\mathrm{CB}}g^i_\mathrm{c}\sum_j p^i_{\mathrm{c},j}(\mathbf{x})\ f\Big([E^i_{\mathrm{c},j}-E_{\mathrm{F},n}(\mathbf{x})]/kT\Big) \tag{2.5.2.2}$$

$$p(\mathbf{x})=\sum_{i\in\mathrm{VB}}g^i_\mathrm{v}\sum_j p^i_{\mathrm{v},j}(\mathbf{x})\ f\Big([-E^i_{\mathrm{v},j}+E_{\mathrm{F},n}(\mathbf{x})]/kT\Big) \tag{2.5.2.3}$$

where $f(E)$ is the Fermi-Dirac distribution at temperature T, g^i_c and g^i_v represent the possible spin and valley degeneracies.

When the simulation is over **1D structure**, the wave function can be separated into the plane wave specified with the lattice wave vector $\mathbf{k}_\|$ in the lateral 2D direction and the quantized wave function in the growth direction, which has the $\mathbf{k}_\|$-dependency. Then the charge carrier densitiy is obtained by the following integral over $\mathbf{k}_\|$:

$$n(x)=\sum_{i\in\mathrm{CB}}g^i_\mathrm{c}\sum_j\frac{1}{(2\pi)^2}\int_{\Omega_{BZ}}d^2\mathbf{k}_\|\ p^i_{\mathrm{c},j}(x,\mathbf{k}_\|)\ f\Big([E^i_{\mathrm{c},j}(\mathbf{k}_\|)-E_{\mathrm{F},n}(x)]/kT\Big) \tag{2.5.2.4}$$

$$p(x)=\sum_{i\in\mathrm{VB}}g^i_\mathrm{v}\sum_j\frac{1}{(2\pi)^2}\int_{\Omega_{BZ}}d^2\mathbf{k}_\|\ p^i_{\mathrm{v},j}(x,\mathbf{k}_\|)\ f\Big([-E^i_{\mathrm{v},j}(\mathbf{k}_\|)+E_{\mathrm{F},n}(x)]/kT\Big) \tag{2.5.2.5}$$

Here the integration is over the two-dimensional Brillouin zone Ω_{BZ}.

Similarly, the charge carrier densities for **2D structure** is calculated by the integral over the 1-dimensional Brillouin zone as

$$n(\mathbf{x})=\sum_{i\in\mathrm{CB}}g^i_\mathrm{c}\sum_j\frac{1}{2\pi}\int_{\Omega_{BZ}}\mathrm{dk}\ p^i_{\mathrm{c},j}(\mathbf{x},\mathrm{k})\ f\Big([E^i_{\mathrm{c},j}(k)-E_{\mathrm{F},n}(\mathbf{x})]/kT\Big) \tag{2.5.2.6}$$

$$p(\mathbf{x})=\sum_{i\in\mathrm{VB}}g^i_\mathrm{v}\sum_j\frac{1}{2\pi}\int_{\Omega_{BZ}}\mathrm{dk}\ p^i_{\mathrm{v},j}(\mathbf{x},\mathrm{k})\ f\Big([-E^i_{\mathrm{v},j}(k)+E_{\mathrm{F},p}(\mathbf{x})]/kT\Big) \tag{2.5.2.7}$$

Single-band model

Things are simpler.

When the **single-band Schrödinger equation** is set to be solved, the envelope function of the j-th eigenstate has only one component $F^i_{l,j}(\mathbf{x})$. Also, the k-integration in (2.5.2.4) to (2.5.2.7) can be done analytically due to the parabolic dispersion according to the effective mass tensor $\underline{m}^{*i}_e$ and $\underline{m}^{*i}_h$.

Thanks to this simpicity the **quantum mechanical charge carrier densities** for d-dimensional simulation can be written up by the following expression:

$$n(\mathbf{x}) = \sum_{i\in\text{CB}} g^i_\text{c} \left(\frac{m_{\text{dos,e}}kT}{2\pi\hbar^2}\right)^{(3-d)/2} \sum_j p^i_{\text{c},j}(\mathbf{x})\, \mathcal{F}_{(1-d)/2}\Big([E^i_{\text{c},j} - E_{\text{F},n}(\mathbf{x})]/kT\Big) \tag{2.5.2.8}$$

$$p(\mathbf{x}) = \sum_{i\in\text{VB}} g^i_\text{v} \left(\frac{m_{\text{dos,h}}kT}{2\pi\hbar^2}\right)^{(3-d)/2} \sum_j p^i_{\text{v},j}(\mathbf{x})\, \mathcal{F}_{(1-d)/2}\Big([-E^i_{\text{v},j} + E_{\text{F},p}(\mathbf{x})]/kT\Big) \tag{2.5.2.9}$$

TODO: The sign in the fermi-dirac integral might be opposite. check the source code.

Here $\mathcal{F}_n(E)$ denotes the Fermi-Dirac integral of order n and $m^i_{\text{dos},\lambda}$ is so-called density-of-states mass defined as

$$m^i_{\text{dos},\lambda} = \left(\det \bar{m}^{*i}_\lambda\right) \quad \lambda = \text{e,h} \tag{2.5.2.10}$$

where $\bar{m}^{*i}_\lambda$ describes the 2×2 or 1×1 submatrix of the effective mass tensor $\underline{m}^{*i}_\lambda$ in the direction of $\mathbf{k}_\parallel$.

In any cases, the carrier densities are dependent on the electrostatic potential $\phi(\mathbf{x})$ through the wave function, which is obtained from the ϕ-dependent Hamiltonian $H(\phi)$. Thus we can also write them as $n(\mathbf{x},\phi)$ and $p(\mathbf{x},\phi)$, which enters into the non-linear Poisson equation introduced later.

Moreover, when the current equation is included in the calculation scheme, seeing the carrier densities as $n(\mathbf{x},\phi,E_{\text{F},n})$ and $p(\mathbf{x},\phi,E_{\text{F},p})$ makes it easy to understand what the self-consistent calculation is actually doing.

Last update: 04/12/2024

2.6 Doping

2.6.1 Activation Energies

Table 2.6.1.1: Donor levels (n-type) in units of eV relative to conduction band edge

Donor Name	Energy	Source
n-As-in-Si	0.054	
n-As-in-Si	0.049	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
n-P-in-Si	0.045	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
n-Sb-in-Si	0.039	
n-N-in-Si	0.045	
n-As-in-Ge	0.013	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
n-P-in-Ge	0.012	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
n-N-in-SiC	0.10	
n-Si-in-GaAs	0.0058	
n-Si-in-AlAs	0.007	300 K, Landolt-Boernstein
n-Si-in-Al0.27Ga0.73As	0.006	Landolt-Boernstein

More parameters can be found here

Table 2.6.1.2: Acceptor levels (p-type) in units of eV relative to valence band edge

Acceptor Name	Energy	Source
p-In-in-Si	0.16	
p-B-in-Si	0.045	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
p-Al-in-Si	0.057	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
p-B-in-Ge	0.010	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
p-Al-in-Ge	0.010	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
p-Al-in-SiC	0.20	
p-C-in-GaAs	0.027	Landolt-Boernstein 1982

More parameters can be found here

2.7 Incomplete ionization

The densities of ionized impurities are calculated in the context of Thomas-Fermi approximation with these formulas:

$$N_{\mathrm{D}}^{+}(\mathbf{x}) = \sum_{i\in\text{Donors}} \frac{N_{\mathrm{D},i}(\mathbf{x})}{1+g_{\mathrm{D},i}\exp((E_{\mathrm{F},n}(\mathbf{x})-E_{\mathrm{D},i}(\mathbf{x}))/k_{\mathrm{B}}T)} \tag{2.7.1}$$

$$N_{\mathrm{A}}^{-}(\mathbf{x}) = \sum_{i\in\text{Acceptors}} \frac{N_{\mathrm{A},i}(\mathbf{x})}{1+g_{\mathrm{A},i}\exp((E_{\mathrm{A},i}(\mathbf{x})-E_{\mathrm{F},p}(\mathbf{x}))/k_{\mathrm{B}}T)} \tag{2.7.2}$$

where the summation is over all different donor or acceptors, $N_{\mathrm{D}}, N_{\mathrm{A}}$ are the doping concentrations, $g_{\mathrm{D}}, g_{\mathrm{A}}$ are the degeneracy factors ($g_{\mathrm{D}} = 2$ and $g_{\mathrm{A}} = 4$ for shallow impurities), and E_D, E_A are the energies of the neutral donor and acceptor impurities, respectively.

These energies of neutral impurities $E_{\mathrm{D},i}, E_{\mathrm{A},i}$ are determined by the ionization energies $E_{\mathrm{D},i}^{\mathrm{ion}}, E_{\mathrm{A},i}^{\mathrm{ion}}$, the bulk conduction and valence band edges (including shifts due to strain) and the electrostatic potential.

$$E_{\mathrm{D},i}(\mathbf{x}) = E_{\mathrm{c}}(\mathbf{x}) - e\phi(\mathbf{x}) - E_{\mathrm{D},i}^{\mathrm{ion}}(\mathbf{x}) \tag{2.7.3}$$

$$E_{\mathrm{A},i}(\mathbf{x}) = E_{\mathrm{v}}(\mathbf{x}) - e\phi(\mathbf{x}) + E_{\mathrm{A}}^{\mathrm{ion}}(\mathbf{x}) \tag{2.7.4}$$

Last update: 04/12/2024

2.8 Carrier transport

2.8.1 Drift-Diffusion Model

The continuity equations in the presence of generation G recombination R of electron-hole pairs read

$$\begin{gathered} -e\frac{\partial n}{\partial t} + \nabla\cdot\big(-e\mathbf{j}_n(\mathbf{x})\big) = -e\big(G(\mathbf{x})-R(\mathbf{x})\big), \\ e\frac{\partial p}{\partial t} + \nabla\cdot e\mathbf{j}_p(\mathbf{x}) = e\big(G(\mathbf{x})-R(\mathbf{x})\big), \end{gathered} \tag{2.8.1.1}$$

where the current is proportional to the gradient of quasi Fermi levels $E_{\mathrm{F},n/p}(\mathbf{x})$

$$\begin{aligned}\mathbf{j}_n(\mathbf{x}) &= -\mu_n(\mathbf{x})n(\mathbf{x})\nabla E_{\mathrm{F},n}(\mathbf{x}),\\ \mathbf{j}_p(\mathbf{x}) &= \mu_p(\mathbf{x})p(\mathbf{x})\nabla E_{\mathrm{F},p}(\mathbf{x}).\end{aligned} \tag{2.8.1.2}$$

Here the charge current has the unit of $(\text{area})^{-1}(\text{time})^{-1}$. $\mu_{n/p}$ are the mobilities of each carrier. In *nextnano*++, $\mu_{n/p}$ are determined using the mobility model specified in the input file under *currents{ }*.

Hereafter we consider stationary solutions and set $\dot{n} = \dot{p} = 0$. The governing equations then reduce to

$$\begin{aligned}\nabla\cdot\mu_n(\mathbf{x})n(\mathbf{x})\nabla E_{\mathrm{F},n}(\mathbf{x}) &= -(G(\mathbf{x}) - R(\mathbf{x})),\\ \nabla\cdot\mu_p(\mathbf{x})p(\mathbf{x})\nabla E_{\mathrm{F},p}(\mathbf{x}) &= G(\mathbf{x}) - R(\mathbf{x}),\end{aligned} \tag{2.8.1.3}$$

which we call **current equation**.

We can also say that the current equation governs the relationship between the **carrier densities** $n(\mathbf{x})$, $p(\mathbf{x})$ and **quasi Fermi levels** $E_{\mathrm{F},n/p}(\mathbf{x})$.

The *nextnano*++ tool solves this equation and Poisson equation (and also Schrödinger equation) self-consistently.

In their solution, the corresponding calculation of the carrier densities $\left(n(\mathbf{x},\phi,E_{\mathrm{F},n}),p(\mathbf{x},\phi,E_{\mathrm{F},p})\right)$ and Poisson equation are firstly iterated for a given quasi-Fermi levels until the carreir densities converge. Then the resulting carrier densities are substituted into the current equation and the quasi-Fermi levels are updated. This whole cycle is iterated until the quasi-Fermi levels satisfies the convergence criteria, which can be tuned by the users from *run{ current_poisson{ } }* or *run{ quantum_current_poisson{ } }*.

Last update: 04/12/2024

2.9 Generation and recombination rates

The recombination mechanisms that *nextnano*++ takes into account for the right-hand-side of (2.8.1.1) are

- Shockley-Read-Hall (SRH) recombination
- Auger recombination
- Radiative recombination
- "fixed (applied)"

The equations and parameters used for the three recombination mechanisms on the top are explained here: *recombination_model{ }*.

The last one "fixed (applied)" is the contribution defined from *structure{region{generation{}}}* and *optics{ photogeneration{ } }*. These typically represent **generation** instead of recombination and used for the simulation of the devices under irradiation such as solar cells or CCDs. (For example, see *nextnano*++ tutorial *GaAs solar cell*.)

According to the specification in the section *classical{ }*, *nextnano*++ can calculate optoelectronic characteristics of the arbitrary structure by means of the so-called semi-classical model.

In this model, various quantities are calculated from the **spontaneous emission rate**, which is calculated at each position $\mathbf{x}$ for the photons with each energy E based on the energy-resolved carrier densities $n(\mathbf{x},E)$ and $p(\mathbf{x},E)$ obtained in the forgoing simulation.

2.9.1 Spontaneous emission rate

$$R_{\mathrm{rad}}^{spon}(\mathbf{x},E) = C(\mathbf{x})\int dE_{\mathrm{h}}\int dE_{\mathrm{e}}\; n(\mathbf{x},E_{\mathrm{e}})p(\mathbf{x},E_{\mathrm{h}})\delta(E_{\mathrm{e}} - E_{\mathrm{h}} - E). \tag{2.9.1.1}$$

Here $C(x)$ $[\mathrm{cm}^3\mathrm{s}^{-1}]$ is the (material-dependent) radiative recombination parameter which is proportional to the one specified in the database (*Radiative recombination*)

Then the other optical characteristics like stimulated emission rate, absorption/gain spectrum, and the imaginary part of the dielectric constant are calculated according to this $R_{\mathrm{rad}}^{spon}(\mathbf{x},E)$.

2.9.2 Generation by the irradiation (fixed(applied))

There is another radiative recombination rate output on *recombination.dat* called **“fixed(applied)”**, which should be always negative. This is the contribution of the generation specified from *structure{region{generation{}}}* and *optics{ photogeneration{ } }*. When we do not specify either of them, this recombination rate is always 0.

$$\begin{aligned} R_{\text{fixed}}(\mathbf{x}) = &- \big(G(\mathbf{x}) \quad \text{specified from structure}\big) \\ &- \Big(\int dE\ G(E,\mathbf{x}) \text{ calculated according to the configuration in classical}\Big). \end{aligned} \tag{2.9.2.1}$$

This is mostly used for the analysis of the absorbing devices such as solar cells or CCDs.

Last update: 04/12/2024

2.10 Mobility

This section describes all mobility models implemented in the nextnano software. Related syntax can be found *here*.

Note

If you need more mobility models implemented in *nextnano*++, contact us

2.10.1 Low-field mobility models

Four low-field following mobility models are supported in *nextnano*++.

Constant

The constant mobility model is due to lattice scattering (phonon scattering) and leads to a constant mobility that depends only on the temperature T. The lattice atoms oscillate about their equilibrium sites at finite temperature leading to a scattering of carriers which results in a temperature dependent mobility $\mu^{n,p}_{const}$. $\mu^{n,p}_{max}$ is the mobility due to bulk phonon (lattice) scattering. For all semiconductors the temperature dependent lattice mobility is modeled by a power law:

$$\mu^{n,p}_{const}(T) = \mu^{n,p}_{max} \cdot \left(\frac{T}{T_0}\right)^{-exponent}, \tag{2.10.1.1}$$

with temperature T and reference temperature $T_0 = 300K$.

The parameter values used in this model for electrons and holes, respectively, are taken from the PhD thesis of V. Palankovski Simulation of Heterojunction Bipolar Transistors (TU Vienna). (Note: The exponent has opposite sign in his PhD thesis.)

Masetti

The Masetti bulk mobility model is used to simulate the doping dependent mobility in Si and takes into account the scattering of the carriers by charged impurity ions which leads to a degradation of the carrier mobility (ionized impurity scattering). It is a model that combines lattice and impurity scattering. This model is temperature independent and the parameters are given for 300 K. Thus it is only valid for 300 K.

Following *[Masetti1983]*, the equation for mobility is :

$$\mu^{n,p} = \mu^{n,p}_{min1} \cdot e^{-\frac{P_c^{n,p}}{N_D+N_A}} + \frac{\mu^{n,p}_{const} - \mu^{n,p}_{min2}}{1 + \left(\frac{N_D+N_A}{C_r^{n,p}}\right)^{\alpha^{n,p}}} - \frac{\mu_1^{n,p}}{1 + \left(\frac{C_s^{n,p}}{N_D+N_A}\right)^{\beta^{n,p}}} \tag{2.10.1.2}$$

with the reference mobility parameters $\mu^{n,p}_{min1}$, $\mu^{n,p}_{min2}$ and $\mu_1^{n,p}$, the reference doping concentration parameters $P_c^{n,p}$, $C_r^{n,p}$, $C_s^{n,p}$, $\alpha^{n,p}$ and $\beta^{n,p}$, and the concentration of ionized donors N_D and acceptors N_A. The total concentration of ionized impurities is given by $N_D + N_A$. The low-doping reference mobility $\mu^{n,p}_{const}$ is determined by equation (2.10.1.1) (constant mobility-model), i.e. the values in the database under keyword *mobility_constant{}* are the same as under this keyword.

Arora

The Arora mobility model is used to simulate the doping dependent mobility in Si and takes into account the scattering of the carriers by charged impurity ions which leads to a degradation of the carrier mobility (ionized impurity scattering). This model is temperature dependent.

Following *[Arora1982]*, the equation for mobility is:

$$\mu^{n,p} = \mu^{n,p}_{min} \cdot \left(\frac{T}{T_0}\right)^{\alpha_m^{n,p}} + \frac{\mu_d^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_d^{n,p}}}{1 + \left(\frac{N_D+N_A}{N_0^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_N^{n,p}}}\right)^{A_a^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_a^{n,p}}}}, \tag{2.10.1.3}$$

with the reference mobility parameter $\mu^{n,p}_{min}(T_0)$, reference mobility parameter $\mu_d^{n,p}$, lattice temperature T, reference temperature $T_0 = 300K$, reference exponent parameter $A_a^{n,p}$, exponents $\alpha_N^{n,p}$ and $\alpha_a^{n,p}$,reference impurity parameter $N_0^{n,p}$, and concentration of ionized donors N_D and acceptors N_A. The total concentration of ionized impurities is given by $N_A + N_D$.

MINIMOS 6

The mobility model used in MINIMOS 6 is used to simulate the doping dependent mobility in Si and takes into account the scattering of the carriers by charged impurity ions which leads to a degradation of the carrier mobility (ionized impurity scattering). This model is temperature dependent and takes into account the reduced mobility due to lattice scattering (i.e. the values in the database under keyword *mobility_constant{}* are the same as under this keyword apart from the sign of the exponent). The formula of Caughey and Thomas *[CaugheyThomas1967]* is used together with temperature dependent coefficients. This model is well suited for Si. The equation for mobility is:

$$\mu^{n,p} = \mu^{n,p}_{min} + \frac{\mu^{n,p}_{const} - \mu^{n,p}_{min}}{1 + \left(\frac{N_D+N_A}{N_0^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_N^{n,p}}}\right)^{A_a^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_a^{n,p}}}}, \tag{2.10.1.4}$$

with lattice temperature T, reference temperature $T_0 = 300K$, reference exponent parameter $A_a^{n,p}$, exponents $\alpha_N^{n,p}$ and $\alpha_a^{n,p}$, reference impurity parameter $N_0^{n,p}$, and concentration of ionized donors N_D and acceptors N_A.

The total concentration of ionized impurities is given by $N_D + N_A$. The $\mu^{n,p}_{const}$ is determined by the constant mobility-model: equation (2.10.1.1). The formulas for the reference mobility parameter $\mu^{n,p}_{const}$ are

$$\mu^{n,p}_{min}(T) = \mu^{n,p}_{min}(T_0)\left(\frac{T}{T_0}\right)^{\alpha^{n,p}_m} \tag{2.10.1.5}$$

$$\mu^{n,p}_{min}(T) = \mu^{n,p}_{min}(T_0)\cdot\left(\frac{2}{3}\right)^{\alpha^{n,p}_m}\left(\frac{T}{200K}\right)^{\alpha^{n,p}_{m2}}, \tag{2.10.1.6}$$

where (2.10.1.5) applies to temperatures $T \geq 200K$ and (2.10.1.6) to temperatures $T < 200K$. The value $T = 200K$ can be changed by T_{Switch}. By setting $\alpha^{n,p}_m = \alpha^{n,p}_{m2}$ and $\alpha^{n,p}_a = 0$, (2.10.1.6) reduces to (2.10.1.5) and this model can also be applied to other basic materials.

It is a model that combines lattice and impurity scattering.

The parameter values used in this model for electrons and holes, respectively, are taken from the PhD thesis of V. Palankovski Simulation of Heterojunction Bipolar Transistors (TU Vienna). (Note: The exponent has opposite sign in his PhD thesis.)

2.10.2 High-Field Mobility Models

Four high-field mobility models are currently implemented in *nextnano*++. In our implementation, each of them uses results obtained from selected low-field model passed via μ_{low}.

Hänsch

As mentioned above, this model is a special case of the Extended Canali model in the limit of strong surface scattering defined by *W. Hänch and M. Miura-Mattausch*

$$\mu(F) = \frac{2\mu_{\mathrm{low}}}{1+\left(1+\left(2\frac{\mu_{\mathrm{low}}F}{v_{\mathrm{sat}}}\right)^2\right)^{1/2}}$$

where μ_{low} is low-field mobility, v_{sat} is saturation velocity, and F is the driving force.

Extended Canali

The Extended Canali model is an extended version of Jacoboni-Canali model, originally applied to electron and hole drift-velocity measurements in silicon by *Canali, et al.*.

$$\mu(F) = \frac{(\alpha+1)\mu_{\mathrm{low}}}{\alpha+\left(1+\left((\alpha+1)\frac{\mu_{\mathrm{low}}F}{v_{\mathrm{sat}}}\right)^\beta\right)^{1/\beta}}$$

where μ_{low} is low-field mobility, v_{sat} is saturation velocity, and F is the driving force. Parameters α, β and v_{sat} are defined independently for holes and electrons . The driving force F of the respective carriers is evaluated as the gradient of the respective quasi-Fermi level. The α parameter should be set to zero, if one aims at using the Extended Canali model. One can transform it into Hänch model by setting $\alpha = 1$ and $\beta = 2$.

Transferred-Electron

The transferred electron model below bases on Monte Carlo simulation of transport in the III-nitride wurtzite materials done by *M. Farahmand, et al.*.

$$\mu(F) = \frac{\mu_{\mathrm{low}} + \frac{v_{\mathrm{sat}}}{F}\left(\frac{F}{E_0}\right)^\beta}{1+\gamma\left(\frac{F}{E_0}\right)^\alpha+\left(\frac{F}{E_0}\right)^\beta}$$

where μ_{low} is low-field mobility, v_{sat} is saturation velocity, F is the driving force, and E_0 is critical field. Parameters α, β, γ and v_{sat} are defined independently for holes and electrons.

Eastman-Tiwari-Shur

A model based on a modified theory of the high-field domains which takes into account the field dependent diffusion by *L. F. Eastman, et al.* for GaAs MESFETs. Where $E_s \equiv \frac{v_{sat}}{\mu_{low}}$ after work of *J. Chillieri, et al.*.

$$\mu(F) = \frac{\mu_{\text{low}} + \frac{v_{\text{sat}}}{F}\alpha\left(\frac{\mu_{\text{low}}F}{v_{\text{sat}}}\right)^{\beta}}{1 + \alpha\left(\frac{\mu_{\text{low}}F}{v_{\text{sat}}}\right)^{\beta}}$$

where μ_{low} is low-field mobility, v_{sat} is saturation velocity, and F is the driving force. Parameters α, β and v_{sat} are defined independently for holes and electrons. The driving force F of the respective carriers is evaluated as the gradient of the respective quasi-Fermi level.

Parameters α and β can be replaced introducing four other parameters E_{peak}, E_{mid}, v_{peak}, and v_{mid}, all related to the shape of the drift velocity function of the driving force. See *J. Chillieri, et al.* for reference.

$$\beta = \frac{log\left(\frac{E_{\text{mid}}\mu_{\text{low}} - v_{\text{mid}}}{E_{\text{peak}}\mu_{\text{low}} - v_{\text{peak}}} \cdot \frac{v_{\text{peak}} - v_{\text{sat}}}{v_{\text{mid}} - v_{\text{sat}}}\right)}{log\left(\frac{E_{\text{mid}}}{E_{\text{peak}}}\right)}$$

$$\alpha = \frac{E_{\text{peak}}\mu_{\text{low}} - v_{\text{peak}}}{v_{\text{peak}} - v_{\text{sat}}}\left(\frac{v_{\text{sat}}}{E_{\text{peak}}\mu_{\text{low}}}\right)^{\beta}$$

Last update: nnnn/nn/nn

2.11 Hamiltonian: 8-band model for zincblende

2.11.1 The Model

Hint

This model can be triggered for any point of the simulation using `classical{ bulk_dispersion{KP8{}}}`. See the *bulk_dispersion{ }* section for reference on syntax.

Our implementation of the 8-band $\mathbf{k}\cdot\mathbf{p}$ model for bulk crystals is a simplified version of the matrix Hamiltonian described in a PhD thesis *[AndlauerPhD2009]* obtained from the one-particle Hamiltonian

$$\hat{H} = \frac{\hat{\mathbf{p}}^2}{2m} + V_0\left(\mathbf{r}\right) + \frac{\hbar}{4m^2c^2}\left[\hat{\sigma} \times \nabla V_0\left(\mathbf{r}\right)\right] \circ \hat{\mathbf{p}} \tag{2.11.1.1}$$

The description below contains also definitions and relations that can be found in *[BirnerPhD2011]* and *[BahderPRB1990]*.

Warning

The Hamiltonian below does not contain terms related to the presence of the magnetic field. Therefore, **proper operator ordering** is neglected to keep formulas as simple as possible. Also, parameters N^+, N^-, κ, and g are not included here. Comprehensive documentation will be published elsewhere.

Our model is expressed in a basis of class $\mathcal{A}$ functions:

$$\left\{\left|s\uparrow\right\rangle,\ \left|s\downarrow\right\rangle,\ \left|x_1\uparrow\right\rangle,\ \left|x_2\uparrow\right\rangle,\ \left|x_3\uparrow\right\rangle,\ \left|x_1\downarrow\right\rangle,\ \left|x_2\downarrow\right\rangle,\ \left|x_3\downarrow\right\rangle\ \right\}$$

The Hamiltonian can be concisely written in a block form as follows.

$$\hat{\mathcal{H}}_{\mathbf{k}\cdot\mathbf{p}} = \begin{bmatrix} \hat{\mathcal{H}}_{\mathrm{cc}}\left(\mathbf{k},\,\hat{\epsilon}\right) & 0 & \hat{\mathcal{H}}_{\mathrm{cv}}\left(\mathbf{k}\right) & 0 \\ 0 & \hat{\mathcal{H}}_{\mathrm{cc}}\left(\mathbf{k},\,\hat{\epsilon}\right) & 0 & \hat{\mathcal{H}}_{\mathrm{cv}}\left(\mathbf{k}\right) \\ \hat{\mathcal{H}}_{\mathrm{vc}}\left(\mathbf{k}\right) & 0 & \hat{\mathcal{H}}_{\mathrm{vv}}\left(\mathbf{k}\right) + \hat{\mathcal{H}}_{\mathrm{vv}}\left(\hat{\epsilon}\right) + \hat{\mathcal{H}}_{\mathrm{so}\uparrow\uparrow} & \hat{\mathcal{H}}_{\mathrm{so}\uparrow\downarrow} \\ 0 & \hat{\mathcal{H}}_{\mathrm{vc}}\left(\mathbf{k}\right) & \hat{\mathcal{H}}_{\mathrm{so}\downarrow\uparrow} & \hat{\mathcal{H}}_{\mathrm{vv}}\left(\mathbf{k}\right) + \hat{\mathcal{H}}_{\mathrm{vv}}\left(\hat{\epsilon}\right) + \hat{\mathcal{H}}_{\mathrm{so}\downarrow\downarrow} \end{bmatrix}$$

where $\mathbf{k}$ is a wave vector and $\hat{\epsilon}$ is a strain tensor.

Diagonal elements for the conduction band are defined as

$$\hat{\mathcal{H}}_{\mathrm{cc}}\left(\mathbf{k},\,\hat{\epsilon}\right) = E_{\mathrm{c}} + A_{\mathrm{c}}k^2 + a_{\mathrm{c}}\mathrm{Tr}\{\hat{\epsilon}\},$$

where k is length of the wave vector, E_{c} is conduction-band edge, a_{c} is absolute hydrostatic deformation potential for the conduction band, $\mathrm{Tr}\{\hat{\epsilon}\}$ is trace of the strain tensor, A_{c} is defined as

$$A_{\mathrm{c}} = A' + \frac{\hbar^2}{2m_0}.$$

A' is one of Kane parameters. It contains interactions between the conduction band and the remote bands $\mathcal{B}$ with Γ_5 symmetry

$$A' = \frac{\hbar^2}{m_0^2}\sum_{nj}^{\mathcal{B}}\frac{\left|\left\langle s\right|\hat{p}_1\left|n\Gamma_5 j\right\rangle\right|^2}{E_{\mathrm{c}} - E_{n,\Gamma_5}}.$$

Blocks introducing interaction between conduction and valence bands are given by

$$\hat{\mathcal{H}}_{\mathrm{cv}}\left(\mathbf{k}\right) = \begin{bmatrix} \imath P_0 k_1 + B k_2 k_3 & \imath P_0 k_2 + B k_1 k_3 & \imath P_0 k_3 + B k_1 k_2 \end{bmatrix}$$

and

$$\hat{\mathcal{H}}_{\mathrm{vc}}\left(\mathbf{k}\right) = \begin{bmatrix} -\imath P_0 k_1 + B k_2 k_3 \\ -\imath P_0 k_2 + B k_1 k_3 \\ -\imath P_0 k_3 + B k_1 k_2 \end{bmatrix},$$

where $k_1,\ k_2,\ k_3$ are three components of the wave vector of interest, P_0 is a Kane parameter describing interactions between conduction band and valence bands within the $\mathcal{A}$ basis

$$P_0 = -\imath\frac{\hbar}{m_0}\left\langle s\right|\hat{p}_1\left|x_1\right\rangle,$$

and B is a Kane parameter including interaction between the all the bands in class $\mathcal{A}$ and remote bands $\mathcal{B}$ of Γ_5 symmetry

$$B = 2\frac{\hbar^2}{m_0^2}\sum_{nj}^{\mathcal{B}} \frac{\langle s|\,\hat{p}_1\,|n\Gamma_5 j\rangle\,\langle n\Gamma_5 j|\,\hat{p}_1\,|x_3\rangle}{\left[E_\mathrm{c}+E_\mathrm{v}\right]/2 - E_{n,\Gamma_5}}.$$

with top valence band energy $E_\mathrm{v} = E_\mathrm{v,av} + \Delta_0$.

Blocks for the valence bands without the strain included are defined as

$$\begin{aligned}\hat{\mathcal{H}}_\mathrm{vv}\left(\mathbf{k}\right) = &\begin{bmatrix} E_\mathrm{v,av}+\frac{\hbar^2}{2m_0}k^2 & 0 & 0 \\ 0 & E_\mathrm{v,av}+\frac{\hbar^2}{2m_0}k^2 & 0 \\ 0 & 0 & E_\mathrm{v,av}+\frac{\hbar^2}{2m_0}k^2 \end{bmatrix} \\ &+ \begin{bmatrix} L'k_1^2+Mk_2^2+Mk_3^2 & N'k_1k_2 & N'k_1k_3 \\ N'k_1k_2 & Mk_1^2+L'k_2^2+Mk_3^2 & N'k_2k_3 \\ N'k_1k_3 & N'k_2k_3 & Mk_1^2+Mk_2^2+L'k_3^2 \end{bmatrix},\end{aligned}$$

where $E_\mathrm{v,av}$ is average energy of valence bands at Γ point, M, N', and L' are Kane parameters introducing interactions between the valence bands in $\mathcal{A}$ and remote bands $\mathcal{B}$ of Γ_1, Γ_3, Γ_4, Γ_5 symmetries

$$\begin{aligned} M &= H_1 + H_2 \\ N' &= F' - G + H_1 - H_2 \\ L' &= F' + 2G \end{aligned}$$

where

$$\begin{aligned} G &= \frac{\hbar^2}{2m_0^2}\sum_{nj}^{\mathcal{B}}\frac{\left|\langle x_1|\,\hat{p}_1\,|n\Gamma_3 j\rangle\right|^2}{E_\mathrm{v}-E_{n,\Gamma_3}} \\ F' &= \frac{\hbar^2}{2m_0^2}\sum_{nj}^{\mathcal{B}}\frac{\left|\langle x_1|\,\hat{p}_1\,|n\Gamma_1 j\rangle\right|^2}{E_\mathrm{v}-E_{n,\Gamma_1}} \\ H_1 &= \frac{\hbar^2}{2m_0^2}\sum_{nj}^{\mathcal{B}}\frac{\left|\langle x_1|\,\hat{p}_1\,|n\Gamma_5 j\rangle\right|^2}{E_\mathrm{v}-E_{n,\Gamma_5}} \\ H_2 &= \frac{\hbar^2}{2m_0^2}\sum_{nj}^{\mathcal{B}}\frac{\left|\langle x_1|\,\hat{p}_1\,|n\Gamma_4 j\rangle\right|^2}{E_\mathrm{v}-E_{n,\Gamma_4}} \end{aligned}$$

Spin-orbit interaction within the valence bands is introduced by

$$\hat{\mathcal{H}}_{\mathrm{so}\uparrow\uparrow} = \frac{\Delta_0}{3}\begin{bmatrix} 0 & -\imath & 0 \\ \imath & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \left[\hat{\mathcal{H}}_{\mathrm{so}\downarrow\downarrow}\right]^\dagger \quad \text{and} \quad \hat{\mathcal{H}}_{\mathrm{so}\uparrow\downarrow} = \frac{\Delta_0}{3}\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -\imath \\ -1 & \imath & 0 \end{bmatrix} = \left[\hat{\mathcal{H}}_{\mathrm{so}\downarrow\uparrow}\right]^\dagger,$$

where spin-orbit interaction energy Δ_0 is defined by

$$\frac{\Delta_0}{3} = -\imath\frac{\hbar}{4m_0^2c^2}\,\langle x_1|\left[\nabla V_0\left(\mathbf{r}\right)\times\hat{\mathbf{p}}\right]_2|x_3\rangle\,.$$

The strain is introduced to the valence bands by

$$\hat{\mathcal{H}}_\mathrm{vv}\left(\hat{\epsilon}\right) = \begin{bmatrix} l\epsilon_{11}+m\epsilon_{22}+m\epsilon_{33} & n\epsilon_{21} & n\epsilon_{31} \\ n\epsilon_{21} & m\epsilon_{11}+l\epsilon_{22}+m\epsilon_{33} & n\epsilon_{32} \\ n\epsilon_{31} & n\epsilon_{32} & m\epsilon_{11}+m\epsilon_{22}+l\epsilon_{33} \end{bmatrix}$$

where ϵ_{ij} are elements of the strain tensor $\hat{\epsilon}$ and m, n, l are matrix elements of a strain-dependent interaction operator, further defining deformation potentails for the valence bands.

Note

All sections below may be moved elswhere in near future

2.11.2 Offsets

$$E_{\rm c} = E_{\rm g}^{\rm (db)} + E_{\rm v,av}^{\rm (db)} + \frac{1}{3}\Delta_0^{\rm (db)} \quad , \quad E_{\rm v,av} = E_{\rm v,av}^{\rm (db)} \quad , \quad \Delta_0 = \Delta_0^{\rm (db)}$$

Where the following mapping to our database is applied.

Table 2.11.2.1: Mapping of offsets to the database

parameter	value in the database
$E_{\rm g}^{\rm (db)}$	`database{ ..._zb{ conduction_bands{ Gamma{ bandgap } } } }`
$E_{\rm v,av}^{\rm (db)}$	`database{ ..._zb{ valence_bands{ bandoffset } } }`
$\Delta_0^{\rm (db)}$	`database{ ..._zb{ valence_bands{ delta_SO } } }`

Attention

If temperature dependence is triggered then the *Varshni* formula is applied to the energy gap such that $E_{\rm c} \to E_{\rm c}(T)$ and $E_{\rm v,av} \to E_{\rm v,av}(T)$.

2.11.3 Deformation potentials

$$\begin{aligned} a_{\rm c} &= a_{\rm c}^{\rm (db)}, \\ m &= a_{\rm v}^{\rm (db)} - b^{\rm (db)}, \\ n &= \sqrt{3}\, d^{\rm (db)}, \\ l &= a_{\rm v}^{\rm (db)} + 2b^{\rm (db)}, \end{aligned}$$

Where the following mapping to our database is applied.

Table 2.11.3.1: Mapping of deformation potentials to the database

parameter	value in the database
$a_{\rm c}^{\rm (db)}$	`database{ ..._zb{ Gamma{ defpot_absolute } } }`
$a_{\rm v}^{\rm (db)}$	`database{ ..._zb{ valence_bands{ defpot_absolute } }`
$b^{\rm (db)}$	`database{ ..._zb{ valence_bands{ defpot_uniaxial_b } }`
$d^{\rm (db)}$	`database{ ..._zb{ valence_bands{ defpot_uniaxial_d } }`

2.11.4 k.p parameters

Attention

In this section we assume that `rescale_S_to` is not defined in the input file at all, like in the examples below. The topic of rescaling S parameter and it's influence on the Hamiltonian will be discussed elsewhere.

As the $\mathbf{k} \cdot \mathbf{p}$ models have been derived in the literature on numerous ways, there are couple of parameterisation standards available of which preference is not clear. Also, depending on the method applied to obtaining parameters some of them are easier accessible that the others. Therefore, depending on the source and the material of interest different schemes of parametrisation may be preffered by the user. For this purpose multiple possibilities of connecting our database to this model are available.

Default settings

The default settings are equivalent to setting all the attrubutes `use_Luttinger_parameters`, `from_6band_parameters`, `approximate_kappa`, `evaluate_S` to no.

Examples

1. Controlling parameters of the Hamiltonian for computation of electronic energy dispersion for a bulk crystal

```
classical{
    bulk_dispersion{
        KP8{
            from_6band_parameters = no
            use_Luttinger_parameters = no
            approximate_kappa = no
            evaluate_S = no
        }
    }
}
```

2. Controlling parameters of the Hamiltonian for which h Schrödinger equation is solved

```
quantum {
    region{
        kp_8band{
            kp_parameters{
                from_6band_parameters = no
                use_Luttinger_parameters = no
                approximate_kappa = no
                evaluate_S = no
            }
        }
    }
}
```

Then the Kane parameters are defined by

$$M = \frac{\hbar^2}{2m_0} M^{(\mathrm{db})} \quad , \quad N' = \frac{\hbar^2}{2m_0} N'^{(\mathrm{db})} \quad , \quad L' = \frac{\hbar^2}{2m_0} L'^{(\mathrm{db})}$$

$$A_\mathrm{c} = \frac{\hbar^2}{2m_0} S^{(\mathrm{db})} \quad , \quad B = \frac{\hbar^2}{2m_0} B^{(\mathrm{db})} \quad , \quad P = \sqrt{\frac{\hbar^2}{2m_0} E_\mathrm{p}^{(\mathrm{db})}}$$

where the following mapping to our database is applied.

Table 2.11.4.1: Mapping of Kane parameters to the database

parameter	value in the database
$M^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ M } } }`
$L'^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ L } } }`
$N'^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ N } } }`
$S^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ S } } }`
$B^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ B } } }`
$E_\mathrm{p}^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ E_P } } }`

Luttinger parameters and electron effective mass

One needs to set all three parameters `from_6band_parameters`, `use_Luttinger_parameters`, `evaluate_S` to `yes` to use the Luttinger parameters (as defined for 6-band $\mathbf{k} \cdot \mathbf{p}$ model) and the effective mass of electrons.

Examples

1. Controlling parameters of the Hamiltonian for computation of electronic energy dispersion for a bulk crystal

```
classical{
    bulk_dispersion{
        KP8{
            from_6band_parameters = yes
            use_Luttinger_parameters = yes
            approximate_kappa = no
            evaluate_S = yes
        }
    }
}
```

2. Controlling parameters of the Hamiltonian for which h Schrödinger equation is solved

```
quantum {
    region{
        kp_8band{
            kp_parameters{
                from_6band_parameters = yes
                use_Luttinger_parameters = yes
                approximate_kappa = no
                evaluate_S = yes
            }
        }
    }
}
```

Then the Kane parameters are defined by

$$
\begin{aligned}
M &= \frac{\hbar^2}{2m_0}\left[-\gamma_1^{(\mathrm{db})} + 2\gamma_2^{(\mathrm{db})} - 1\right] \\
N' &= \frac{\hbar^2}{2m_0}\left[-6\gamma_3^{(\mathrm{db})}\right] + \frac{E_\mathrm{p}^{(\mathrm{db})}}{E_\mathrm{g}} \\
L' &= \frac{\hbar^2}{2m_0}\left[-\gamma_1^{(\mathrm{db})} - 4\gamma_2^{(\mathrm{db})} - 1\right] + \frac{E_\mathrm{p}^{(\mathrm{db})}}{E_\mathrm{g}} \\
A_c &= \frac{\hbar^2}{2m_0}\left[\frac{1}{m_\mathrm{e}^{(\mathrm{db})}} - \frac{2E_\mathrm{p}^{(\mathrm{db})}}{3E_\mathrm{g}} - \frac{E_\mathrm{p}^{(\mathrm{db})}}{3\left[E_\mathrm{g} + \Delta_0^{(\mathrm{db})}\right]}\right] \\
B &= \frac{\hbar^2}{2m_0} B^{(\mathrm{db})} \\
P &= \sqrt{\frac{\hbar^2}{2m_0} E_\mathrm{p}^{(\mathrm{db})}},
\end{aligned}
$$

where the following mapping to our database is applied.

Table 2.11.4.2: Mapping to the database

parameter	value in the database
$\gamma_1^{(\mathrm{db})}$	`database{ ..._zb{ kp_6_bands{ gamma_1 } } }`
$\gamma_2^{(\mathrm{db})}$	`database{ ..._zb{ kp_6_bands{ gamma_2 } } }`
$\gamma_3^{(\mathrm{db})}$	`database{ ..._zb{ kp_6_bands{ gamma_3 } } }`
$m_e^{(\mathrm{db})}$	`database{ ..._zb{ conduction_bands{ Gamma{ mass } } } }`
$\Delta_0^{(\mathrm{db})}$	`database{ ..._zb{ valence_bands{ delta_SO } } }`
$E_\mathrm{p}^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ E_P } } }`
$B^{(\mathrm{db})}$	`database{ ..._zb{ kp_8_bands{ B } } }`

Rescaling S

One of ways to get rid of spurious solutions in quantum structures is to rescale S parameter to 0 or 1. The S defines A_c as

$$A_c = \frac{\hbar^2}{2m_0} S$$

Examples

1. Rescaling S in the Hamiltonian for computation of electronic energy dispersion for a bulk crystal

```
classical{
    bulk_dispersion{
        KP8{
            rescale_S_to = 1
        }
    }
}
```

2. Rescaling S in the Hamiltonian for which the Schrödinger equation is solved

```
quantum {
    region{
        kp_8band{
            kp_parameters{
                rescale_S_to = 1
            }
        }
    }
}
```

The initial value of S is determined according to choices described before. If one chose `evaluate_S = no` then

$$S = S^{(db)},$$

otherwise, if one chose `evaluate_S = yes` then

$$S = \frac{1}{m_\mathrm{e}^{(\mathrm{db})}} - \frac{2E_\mathrm{p}^{(\mathrm{db})}}{3E_\mathrm{g}} - \frac{E_\mathrm{p}^{(\mathrm{db})}}{3\left[E_\mathrm{g} + \Delta_0^{(\mathrm{db})}\right]}.$$

In the input file, one can request consistent rescaling the model such that $S \to S^{(\mathrm{new})}$ resulting in

$$A_c = \frac{\hbar^2}{2m_0} S^{(\mathrm{new})}.$$

A rescaled Kane energy $E_{\mathrm{p}}^{(\mathrm{new})}$ is evaluated to ensure that the model gives the same electronic band structure (ideally) as before the rescaling, but without spurious solutions. It is done directly from the assumption the $S = S^{(\mathrm{new})}$.

$$E_{\mathrm{p}}^{(\mathrm{new})} = E_{\mathrm{p}}^{(\mathrm{db})} + \left[S - S^{(\mathrm{new})}\right] \frac{E_{\mathrm{g}}\left[E_{\mathrm{g}} + \Delta_0^{(\mathrm{db})}\right]}{E_{\mathrm{g}} + \frac{2}{3}\Delta_0^{(\mathrm{db})}}$$

After the rescaled Kane energy is evaluated, it is used to update or redefine other relevant Kane parameters entering the model.

$$L' \to L' + \frac{E_{\mathrm{p}}^{(\mathrm{new})} - E_{\mathrm{p}}^{(\mathrm{db})}}{E_{\mathrm{g}}},$$
$$N' \to N' + \frac{E_{\mathrm{p}}^{(\mathrm{new})} - E_{\mathrm{p}}^{(\mathrm{db})}}{E_{\mathrm{g}}},$$
$$P = \sqrt{\frac{\hbar^2}{2m_0} E_{\mathrm{p}}^{(\mathrm{new})}}.$$

Where L' and N' are initially evaluated according to the choices in the `kp_parameters{ }` group as described in previous sections.

2.11.5 Zeeman Term

Magnetic effects are included via the Zeeman term

$$\hat{H}_n = \frac{\mu_B}{2} \cdot \hat{\sigma} \cdot \hat{g}_n \cdot \mathbf{B}$$

for both electrons and holes. The $\hat{g}_n$ for electrons is diagonal and can be replaced with a scalar

$$g_c = \frac{\left(E_c^{\uparrow} - E_c^{\downarrow}\right)}{\mu_B B}$$

Free electron has the g factor equal 2, while in the crystal, due to spin-orbit splitting is different and can be approximated by so-called Roth's formula.

$$g_c = 2 - \frac{2E_p\Delta_0}{3E_g\left(E_g + \Delta_0\right)}$$

Parameter *KP8{ correct_electron_gfactor }* controls evaluation of the g_c for the bulk model as follows.

If `correct_electron_gfactor` is negative, then

$$g_c = 2$$

and

$$\hat{H}_n = \frac{\mu_B}{2} \cdot \hat{\sigma} \cdot 2 \cdot \mathbf{B}.$$

If `correct_electron_gfactor = 0`, then

$$m_{eff} = S + P^2 * \frac{E_g + \frac{2}{3}\Delta_0}{E_g\left(E_g + \Delta_0\right)}$$

where, S, P, E_g, and Δ_0 are taken from the database or estimated as defined in `kp_parameters{ }` group. A ne parameter P_{new} is defined

$$P_{new} = \sqrt{\left(m_{eff} - \frac{\hbar^2}{2m_0}\right) \frac{E_g\left(E_g + \Delta_0\right)}{E_g + \frac{2}{3}\Delta_0}}$$

and used to compute corrected g-factor as

$$g_s = \left(P_{new}^2 - P^2\right) \frac{2m_0}{\hbar^2} \frac{2\Delta_0}{3E_g\left(E_g + \Delta_0\right)}.$$

This g-factor is then used in the Zeeman term.

$$\hat{H}_n = \frac{\mu_B}{2} \cdot \hat{\sigma} \cdot (2 + g_s) \cdot \mathbf{B}$$

If *KP8{ rescale_kp_everywhere }* is set to `yes`, then the Kane parameters L and N^+ are rescaled

$$L \to L + \frac{P_{new}^2 - P^2}{E_g}$$

$$N^+ \to N^+ + \frac{P_{new}^2 - P^2}{E_g}$$

which results in replacing the P with a new one

$$P \to P = P_{new}$$

to be used in the remaining part of the Hamiltonian.

If `correct_electron_gfactor` is positive, then the algorithm is the same as in the case of having it zero, with a change that the band gap has assigned teh value of `correct_electron_gfactor`.

$$E_g \to E_g = G$$

at the very beginning. This assignment influences only the band gap used within this algorithm. The band gap in other parts of the model is included normally, based on the parameters from the database.

Last update: 27/05/2025

2.12 Interface Hamiltonian 8-band Zinc-Blende

2.12.1 The Model

The interface Hamiltonian is used to model the heterostructures with atomic sharp interfaces. This is an extension of the standard 8-band $\mathbf{k} \cdot \mathbf{p}$ model (described *here*). The model, originally derived in *[Kilpstein2010]*, includes $l = 2$ perturbation terms, which have been omitted in our implementaion, consistent with the approach in *[Livneh-PRB2012]*.

The interface Hamiltonian in the block form is given by:

$$\hat{\mathcal{H}}_{interface} = \delta(z - z_i) \begin{bmatrix} D_S & 0 & 0 & \pi\beta \\ 0 & D_X & \pi\alpha & 0 \\ 0 & \pi\alpha & D_X & 0 \\ \pi\beta & 0 & 0 & D_Z \end{bmatrix}$$

where δ is the Dirac delta function, z_i is the position of the interface, D_S, D_X, D_Z, α, and β are the interface Hamiltonian parameters, π is the parameter that can take values of ± 1. Parameter π is used to distinguish different ordering of materials at the interface, for example InaAs/GaSb and GaSb/InAs interfaces would have same interface paramters, but opposite π values.

2.12.2 Implementation and usage

The interface Hamiltonian is available only for 1D simulations. For the best results, ensure that there is a grid point at the interface position, for example like so:

```
grid{
    xgrid{
        ...
        line{ pos = $x_1          spacing = $x_interface }   # material1
        line{ pos = $x_interface  spacing = $x_2 }           # material2
        ...
    }
}
```

To add the interface Hamiltonian to the simulation, use the `interface` keyword in `quantum{ region{ kp8_band{} } }` section of input file.

Example:

```
quantum{
    region{
        kp_8band{
            ...
            interface{
                position = $x_interface
                D_S = 1.0
                D_X = 2.0
                D_Z = 1.5
                alpha = 0.1
                beta = 0.1
                reverse = no
            }
            ...
        }
    }
}
```

`reverse=no` corresponds to $\pi = 1$, and `reverse=yes` corresponds to $\pi = -1$. Other interface parameters are expressed in $eV \cdot nm$.

One can add multiple interfaces to the simulation by adding multiple `interface` blocks to the input file. Example below has 2 interfaces with the same paramters, but different order of materials (common case for modelling of quantum well).

```
quantum{
    region{
        kp_8band{
            ...
            interface{
                position = $x_interface1
                D_S = 1.0
                D_X = 2.0
                D_Z = 1.5
                alpha = 0.1
                beta = 0.1
                reverse = no
            }
            interface{
                position = $x_interface2
```

(continues on next page)

(continued from previous page)

```
                D_S = 1.0
                D_X = 2.0
                D_Z = 1.5
                alpha = 0.1
                beta = 0.1
                reverse = yes
            }
        }
    }
}
```

To set up repeating interfaces, one can use the `array_x` keyword in the `interface` block.

```
quantum{
    region{
        kp_8band{
            ...
            interface{
                position = 10.0
                ... # parameters here
                array_x{
                    shift = 2.0
                    min = -2
                    max = 3
                {
            }
        }
    }
}
```

The above example will create 6 interfaces with the same parameters, at positions 6.0, 8.0, 10.0, 12.0, 14.0 and 16.0 nm.

Last update: 23/01/2025

2.13 Excitons

The exciton states are computed using the approach from *[ChuangOpto1995]*. The explanation below only covers the most important aspects of the model, for detailed derivation please refer to the *[ChuangOpto1995]* book. The exciton computation is only available for 1D systems.

2.13.1 Model

Assuming the effective mass approximation for the electron and hole, the Schrödinger equation for the exciton can be written as:

$$(H_e(\bar{r}_e) + H_h(\bar{r}_h) - \frac{e^2}{4\pi\varepsilon|\bar{r}_e - \bar{r}_h|})\Phi(r_e, r_h) = E\Phi(r_e, r_h) \tag{2.13.1.1}$$

where H_e and H_e are the Hamiltonians for the electron and hole, respectively, $\bar{r}_e$ and $\bar{r}_h$ are the coordinates of the electron and hole, respectively, ε is the dielectric constant, and E is the energy of the exciton.

The wave function of the exciton, formed by electron n and hole m, will be calculated in the form:

$$\Phi(r_e, r_h) = exp(i\bar{K}_t\bar{R}_t)F(\rho, z_e, z_h) = exp(i\bar{K}_t\bar{R}_t)\phi_{nm}(\rho)f_n(z_e)g_m(z_h) \tag{2.13.1.2}$$

where $\bar{K}_t$ is the in-plane wave vector of the exciton, $\bar{R}_t$ is the in-plane coordinate of the exciton, $F(\rho, z_e, z_h)$ is the exciton envelope function, $f_n(z_e)$ and $g_m(z_h)$ are the single particle envelope wave functions of electron and hole in the growth direction. Then, the equation for the unknown $\phi_{nm}(\rho)$ is given by:

$$\left(-\frac{\hbar^2}{2m_r}\nabla_\rho^2 - V_{nm}(\rho)\right)\phi_{nm}(\rho) = E_{binding}\phi_{nm}(\rho) \tag{2.13.1.3}$$

where m_r is the reduced mass of the exciton, $E_{binding}$ is the binding energy of the exciton, and $V_{nm}(\rho)$ is expressed as:

$$V_{nm} = \int dz_e |f_n(z_e)|^2 \int dz_h |g_m(z_h)|^2 \frac{e^2}{4\pi\varepsilon_s(\rho^2 + |z_e - z_h|^2)} \tag{2.13.1.4}$$

The solution of the equation for $\phi_{nm}(\rho)$ can be found variationally by minimizing the binding energy of the exciton. The form of the solution is assumed to be similar to 1S state of 2D hydrogen atom:

$$\phi(\rho) = \sqrt{\frac{2}{\pi}\frac{1}{\lambda}} exp(-\rho/\lambda) \tag{2.13.1.5}$$

where λ is the variational parameter, which has an interpretation of exciton inplane Bohr radius. The variational parameter λ is determined by minimizing the binding energy of the exciton from equation (2.13.1.3).

2.13.2 Averaging model parameters

The model depends on dielectric constant ε, effective masses of the electron and hole m_e and m_h, which are not constant in heterostructures. If not given in the input file, the volume averaged values of these parameters are used. For effective masses, density weighted average is also possible.

2.13.3 Excitons in multiband Hamiltonians

The computation of the exciton in the case of 8-band $\mathbf{k}\cdot\mathbf{p}$ Hamiltonian is complicated by the fact, that the electron and hole Hamltonians are no longer separable. In that case the equations derived from effective mass Hamiltonians are used, using wave functions computed with the 8-band Hamiltonian. As the effective masses are not longer parameters of the Hamiltonian, the effective masses used are computed from the parameters for the 8-band Hamiltonian: L, M, N, E_P, S, E_{gap}. The same approach is used for 6-band $\mathbf{k}\cdot\mathbf{p}$ Hamiltonian, where the effective masses are computed from the parameters L, M, N.

2.14 Optical spectra

2.14.1 Fermi's golden rule

The *nextnano*++ tool has another important calculation scheme of optical properties, which is specified in the section *optics{ quantum_spectra{ } }*. Here *nextnano*++ calculates them using the Fermi's golden rule (time-dependent perturbation theory) with 8-band k.p model.

- Optical absorption coefficient
- Real/imaginary part of the dielectric constant
- Refractive index
- Optical gain as a negative part of optical absorption coefficient
- Spontaneous emission rate
- Transition intensity (optical matrix element)

For further detail about this section, please see *Optical absorption for interband and intersubband transitions*.

This page will summarize theory, that is currently distributed on the following pages:

- *Intersubband transitions in InGaAs/AlInAs multiple quantum well systems*
- *Optical intraband transitions in a quantum well - Momentum matrix elements and selection rules*
- *Optical absorption for interband and intersubband transitions*

2.15 Optoelectronic characterization

2.15.1 Current and Power

^Photocurrent

Then the **photocurrent** I_{photo} is calculated as the summation of the integration of these "radiative" and "fixed":

$$I_{\text{photo}} = e \cdot \left(\int d\mathbf{x}\; R^{stim}_{\text{rad,net}}(\mathbf{x}) + \int d\mathbf{x}\; R_{\text{fixed}}(\mathbf{x}) \right) \tag{2.15.1.1}$$

Power

$$\sum_i V_{\text{i-th contact}} \cdot I_{\text{i-th contact}} \tag{2.15.1.2}$$

Absorbed-power

$$\int dE d\mathbf{x}\; E \cdot G(E, \mathbf{x}) \tag{2.15.1.3}$$

where $G(E, x)$ is the generation rate calculated according to the configuration in *classical{ }*.

Emitted-power

$$\int dEdx\; E \cdot R^{spon}_{\text{rad}}(E, x) \tag{2.15.1.4}$$

2.15.2 Efficiencies

IQE - internal quantum efficiency

is calculated as

$$\eta_{IQE} = \frac{I_{\text{photo}}}{I_{\text{total}}} \tag{2.15.2.1}$$

where I_{total} is the total injected current consisted of both electron and hole currents.

The electrical power and optical power are calculated and output in *power.dat*:

RQE - volume quantum efficiency

, which is also called as **radiative quantum efficiency**, is calculated as

$$\eta_{VQE} = \frac{R^{stim}_{\text{rad,net}} + R_{\text{fixed}}}{R_{\text{total}}} \tag{2.15.2.2}$$

where $R_{\text{total}} = R^{stim}_{\text{rad,net}} + R_{\text{fixed}} + R_{\text{Auger}} + R_{\text{SRH}}$ is the total recombination rate including both radiative and non-radiative recombination.

Both η_{IQE} and η_{VQE} agree if the electrons and holes injected into the active region are fully consumed up by the recombination there. However, if they are not consumed up, $e \cdot R_{\text{total}} < I_{\text{charge}}$ and this results in $\eta_{\text{IQE1}} > \eta_{\text{IQE2}}$

Last update: 09/12/2024

CHAPTER
THREE

COURSES

This site will soon contain sets of tutorials targeting selected semiconductor devices.

CHAPTER FOUR

TUTORIALS

4.1 Introduction

This page lists all tutorials for *nextnano*++. The following labels are used to distinguish selected tutorials.

— DEV — Tutorials under development. The input files are not present in any release yet, and it is not clear when they will be added.

— SOON — Tutorials that are finished or almost finished. Their input files are not present in any release yet. They will be added to the next release.

— NEW — Tutorials for which input files are available since the last release (most likely alpha).

— EDU — Tutorials written aiming at teaching.

— FREE — Tutorials that can be run using free distributions of *nextnano*++

Attention

Links to the tutorials and names of exemplary input files may change.

4.2 Basics

Below you can find basic tutorials introducing the most important elements of *nextnano*++ syntax as well as fundamental concepts hidden behind them. We are continuously working on including new tutorials here sou you can learn *nextnano*++ easier.

4.2.1 Defining Structures

The set of tutorials below is the most basic one aiming at teaching you how to define structures for your simulations. The most relevant elements of *nextnano*++ syntax is presented here.

— FREE — Hello World

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *basics_1D_hello_world.in*

Scope of the tutorial:

- The general structure of the input files
- Running the input file with *nextnanomat*
- Basic content of the simulation output

- Defining 1D structures
- Computing basic band profiles

Introduced Keywords:

- `global{ temperature simulate1D{} substrate{ name } crystal_zb{ x_hkl y_hkl } }`
- `grid{ xgrid{ line{ pos spacing } }`
- `structure{ region{ binary{ name } contact{ name } everywhere{} line{ x } } }`
- `contacts{ fermi{ name bias } }`
- `classical{ Gamma{} HH{} LH{} output_bandedges{ averaged } }`

Relevant output Files:

- *bias_00000\bandedges.dat*

Introduction

The input file *basics_1D_hello_world.in* is prepared to compute a band profile of a simple 1D structure consisting of an InAs layer sandwiched between two GaAs layers without strain, see Figure 4.2.1.1.

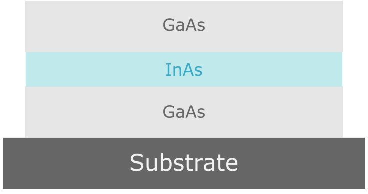


Figure 4.2.1.1: A schematic of a GaAs/InAs/GaAs heterostructure

Global Settings of the Simulation

The group *global{ }* is required to define multiple general aspects of whole simulation. The temperature of the crystal and carriers is set to 300 K by setting `temperature = 300`. The band gap is temperature dependent by default. Choosing that the simulation is held in 1D space is done by calling `simulate1D{}`. The substrate is chosen by a nested group `substrate{ name = "GaAs"}`, where `name` is an attribute to which you can assign any of the available *material names*. In this case the choice of substrate material is arbitrary, because strain calculations are not triggered. Crystal orientation in the simulation coordinate system is defined inside a nested group `crystal_zb{}` setting values of two attributes: `x_hkl = [100]` and `y_hkl = [010]`, which assigns [100] direction to the x-axis of the simulation (the axis of the 1D simulation) and [010] direction to the y-axis of the simulation (still existing).

```
global{ # this group is required in every input file
    temperature = 300           # set temperature (required)
    simulate1D{}                # choose between 1D, 2D or 3D simulation
    substrate{ name = "GaAs" }  # substrate material (required)
    crystal_zb{                 # crystal orientation
        x_hkl = [1, 0, 0]       # x-axis is perp. to lattice plane (100)
        y_hkl = [0, 1, 0]       # y-axis is perp. to lattice plane (010)
                                # z-axis is determined from x-axis and y-axis
    }
}
```

Numerical Grid

The group *grid{ }* is used to define the numerical grid of the simulation. As there is only x-axis in the 1D simulations, only `xgrid{ }` group is used to define the grid. Each group `line{}` defines a “line” (a point in 1D, a line in 2D, and a plane in 3D) at a position `pos` forcing a grid spacing `spacing` in its vicinity and assuring that there is a grid point at the specified coordinate `pos`.

```
grid{ # this group is required in every input file
    xgrid{                      # grid in x direction
        line{
            pos = 0.0           # start of device at x=0.0 nm
            spacing = 4.0       # grid spacing 4.0 nm
        }
                                # from x=0.0 nm to x=20.0 nm further grid points
                                # are created according to the interpolated spacing (4.0 -> 0.5)
                                # (no equidistant grid spacing)
        line{
            pos = 20.0          # grid point at GaAs/InAs interface
            spacing = 0.5       # grid spacing 0.5 nm
        }
                                # from x=20.0 nm to x=30.0 nm further grid points
                                # are created according to the interpolated spacing (0.5 -> 0.5)
                                # (equidistant grid spacing)
        line{
            pos = 30.0          # grid point at InAs/GaAs interface
            spacing = 0.5       # grid spacing 0.5 nm
        }
                                # from x=30.0 nm to x=50.0 nm further grid points
                                # are created according to the interpolated spacing (0.5 -> 4.0)
                                # (no equidistant grid spacing)
        line{
            pos = 50.0          # end of device at x=50.0 nm
            spacing = 4.0       # grid spacing 4.0 nm
        }
    }
}
```

There are 4 "lines" specified in the input file. The two of them with `pos = 0.0` and `pos = 50.0`, as the most outer ones, define the span of the entire grid. The remaining two, with `pos = 20.0` and `pos = 30.0`, are defined at the positions of material interfaces defined in the next group, to assure stable representation of the design in the discrete grid space. The figure Figure 4.2.1.2 shows schematically the process of defining the grid.

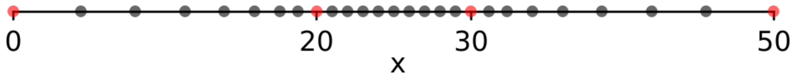


Figure 4.2.1.2: Schematics of the simulation grid with four "lines" defined (red circles). Interpolated grid points between lines are depicted with black circles.

One can also view the grid spacing using *nextnanomat*, see Figure 4.2.1.3.

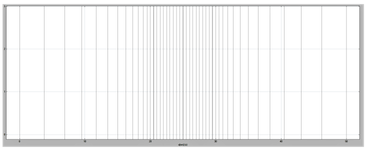


Figure 4.2.1.3: The numerical grid in the simulation.

Defining the Structure

The definition of specific structure is kept in the group *structure{ }*. Here groups `region{}` are used to assign binary materials (using `binary{}`) and boundary conditions for Poisson and current equations (using `contact{}`) to specific regions within the earlier defined space. First, material **GaAs** and boundary condition named "whatever" are assigned to entire space by specifying `binary{ name = GaAs }`, `contact{ name = whatever }`, and `everywhere{}` inside one `region{}`. Next, material **InAs** is assigned to a region spanning from $x = 20.0$ to $x = 50.0$, by defining another `region{}` group, containing ``binary{ name = InAs } ` and :code:`line{ x = [ 20.0, 30.0 ] }``. In that case, **InAs** is overwriting **GaAs** in the selected region, while the boundary conditions specified by `contact{}` remain.

```
structure{ # this group is required in every input file
    region{
        binary{ name = GaAs }               # material GaAs
        contact{ name = hello_world }       # contact definition
        everywhere{}                        # ranging over the complete device, from␣
↪x=0.0 nm to x=50.0 nm
    }
    region{
        binary{ name = InAs }               # material InAs
        line{ x = [ 20.0, 30.0 ] }          # overwriting previously defined GaAs in␣
↪the interval x=20.0 nm to x=30.0 nm
}
```

Bondary conditions

The boundary conditions for Poisson and current equations are specified in the group *contacts{ }*. They have to be specified even if the equations are not solved. Here, the boundary condition for quasi-Fermi levels only is chosen by calling `fermi{}`. The contact is named "hello_world" by setting `name = hello_world`. This name is used for referencing to this specific contact in the definition of the structure. The energy of Fermi level is set to 0 eV by setting `bias = 0.0`.

```
contacts{ # this group is required in every input file
    fermi{                                  # type of contact
        name = hello_world                  # refer to regions with contact name
↪'hello_world'
        bias = 0.0                          # region with contact name 'hello_world'␣
↪is set to 0 V
    }
}
```

Choice of Bands

The *classical{ }* group is called to choose which bands should be taken into account in the semiclassical simulations, here only computing the profile. The first conduction band at Γ point, heavy-, and light-hole valence bands are selected by calling groups: `Gamma{}`, `HH{}`, and `LH{}`, respectively. The group `output_bandedges{}` allows to output the band profile, while its attribute `averaged = no` ensures that the profile is not going to be averaged over neighboring grid points in the output file.

```
classical{ # this group is required in every input file
    Gamma{}                                 # include conduction band at gamma point␣
↪in the calculation
    HH{}                                    # include heavy hole band in the␣
↪calculation
    LH{}                                    # include light hole band in the␣
↪calculation
    output_bandedges{ averaged = no }       # necessary to see a energy profile
}
```

Running the Simulation and Viewing the Results

The simulation can be started in *nextnanomat* by pressing **F8** on the keyboard or by clicking the icon . A folder with simulation results is created in the output directory.

The output of the simulation can be viewed under the "Output" tab at the top of *nextnanomat*. Within the tab, navigate to the folder `bias_00000` and click on `bandedges.dat`. A plot of the Gamma, LH and HH energy profiles should be visible.

The grid used in the simulation can be shown by checking the box "Show grid" in the menu on the left of *nextnanomat*. To export the figure as a `.plt` file, click on the icon in the top right corner.

Then click on `bandedges.dat`. Hold down `shift` on the keyboard and click the plots of your interest. In this tutorial, *Gamma[eV]*, *HH[eV]* and *LH[eV]* are chosen from the bottom right panel. Press `shift + a` on the keyboard or the icon in the top right corner of *nextnanomat*.

Next, select icon at the top and choose the option "Create and Open Gnuplot File (*.plt) from Items of Overlay". A Gnuplot window should pop up. Click the icon and name the file, and save it.

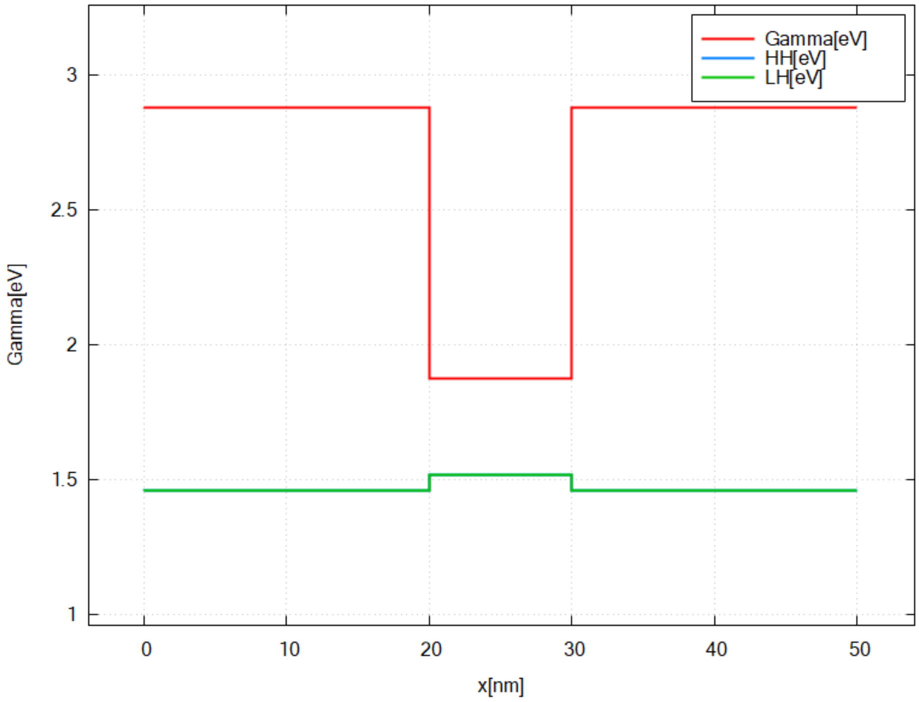


Figure 4.2.1.4: Energy profile of GaAs/InAs/GaAs heterostructure without considering strain.

Last update: 16/07/2024

— FREE — Finite Periodic Structures

- *Header*

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *basics_1D_finite_periodic_simple.in*
- *basics_1D_finite_periodic_double.in*

Introduction

We will now concentrate on two particular features inside the `structure{ }` group which enable you to create periodic structures conveniently. We will discuss their application at the example of a finite superlattice structure. After completing this tutorial, you will know more about

- creating periodic structures with `array_x{}`
- duplicating periodic structures with `array2{}`

Keywords: `array_x{}`, `array2{}`

Main

In the first part, we want to show how to create the structure in Figure 4.2.1.5.

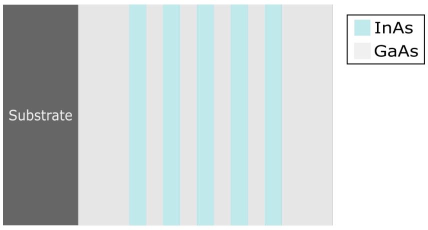


Figure 4.2.1.5: shows multiple GaAs/InAs quantum wells, which forms a finite superlattice

In the second part, we extend the input file of part one, and create the structure shown in Figure 4.2.1.6.

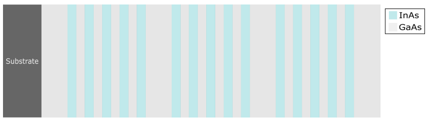


Figure 4.2.1.6: shows a sequence of three GaAs/InAs superlattices

Based on what we learned in tutorial 1, we should have the basic knowledge to create these structure without using *arrays*. It would be quite an effort to create layer by layer. *arrays* offer a convenient alternative to that approach. The idea is to duplicate an existing **sample structure** multiple times in a specific direction. This takes just a few lines of code and gives in addition much more flexibility for your simulations. The **sample structure** in our case will be the GaAs/InAs/GaAs heterostructure from tutorial 1.

Input file 1: Repeated regions

Specifying the structure of the device

```
structure{ # this group is required in every input file
    region{
        binary{ name = GaAs }           # material: GaAs
        contact{ name = whatever }      # contact definition
        everywhere{}                    # ranging over the complete device
    }
    region{
        binary{ name = InAs }           # material: InAs
        line{ x = [ 20.0, 30.0 ] }      # ranging from x=20.0 nm to x=30.0 nm,␣
↪overwrites the previously defined GaAs

        array_x{                        # line{ x = [ 20.0, 30.0 ] } is duplicated in␣
↪the x direction
            shift = 20.0                # the interval  x = [ 20.0, 30.0 ] inside line
↪{} is shifted by an integer multiple of 20.0 nm
            max = 2                     # 2 duplicates in +x direction
            min = -2                    # 2 duplicates in -x direction
        }

        # In short, we are creating 5 InAs regions (overwriting GaAs) in the␣
↪intervals:
        # line{ x = [20.0+i*shift, 30.0 nm+i*shift]} (min<=i<=max)
    }
}
```

As in tutorial 1, we create an InAs layer, which ranges from $x = 20$nm to $x = 30$nm. By introducing `array_x{}` this layer is duplicated along x. The position of the duplicates is determined by the `shift` value. The shift direction ($+x$ or $-x$) and the number of duplicates in each direction is set by `max` and `min`.

Here, `max=2` creates two duplicates in the $+x$ direction every 20nm. The first "copy" ranges from $x = 20\text{nm} + \mathbf{1} \cdot 20\text{nm} = 40\text{nm}$ to $x = 30\text{nm} + \mathbf{1} \cdot 20\text{nm} = 50\text{nm}$ and the second ranges from $x = 20\text{nm} + \mathbf{2} \cdot 20\text{nm} = 60\text{nm}$ to $x = 30\text{nm} + \mathbf{2} \cdot 20\text{nm} = 70\text{nm}$. Analogous, `min=-2` creates two duplicates in the $-x$ direction every 20nm. Mind the negative sign!

After defining the structures, we have to adapt the grid to our newly constructed device.

Specifying the grid

```
grid{ # this group is required in every input file
    xgrid{                    # grid in x direction
        line{
            pos = -40.0       # start device at x=-40.0 nm
            spacing = 4.0     # grid spacing 4.0 nm
        }
                              # from x=0.0 nm to x=20.0 nm further grid points
                              # are created according to the interpolated spacing (4.0 -
↪> 0.5)
                              # (no equidistant grid spacing)
        line{
            pos = -20.0       # bottom GaAs/InAs interface at x=-20.0 nm
            spacing = 0.5     # grid spacing 0.5 nm
        }
                              # from x=-20.0 nm to x=70.0 nm further grid points
                              # are created according to the interpolated spacing (0.5 -
↪> 0.5)
```

(continues on next page)

(continued from previous page)

```
                            # (equidistant grid spacing)
        line{
            pos = 70.0      # top InAs/GaAs interface at x=70.0 nm
            spacing = 0.5   # grid spacing 0.5 nm
        }
                            # from x=70.0 nm to x=90.0 nm further grid points
                            # are created according to the interpolated spacing (0.5 -
↪> 4.0)
                            # (no equidistant grid spacing)
        line{
            pos = 90.0      # start device at x=90.0 nm
            spacing = 4.0   # grid spacing 4.0 nm
        }
    }
}
```

We first extend the device, since we created new material regions: the bottom of the lowest InAs layer is located at $x_min = -2 \cdot 20\text{nm} + 20\text{nm} = -20.0\text{nm}$ and the top of the highest InAs layer is located at $x_min = 30\text{nm} + \mathbf{2} \cdot 20\text{nm} = 70\text{nm}$. We have chosen $x = -40\text{nm}$ and $x = 70\text{nm}$ as our start and end points, in order to include all new material layers. In tutorial 1 we have learned that we also have to take care about interfaces. To keep things simple, we use an equidistant grid spacing inside the superlattice.

Output

We simulate the device by clicking `F8` on the keyboard. In the related output file (⇒ `bias_00000` ⇒ `bandedges.dat`) you should find a plot of band edges as shown in Figure 4.2.1.7.

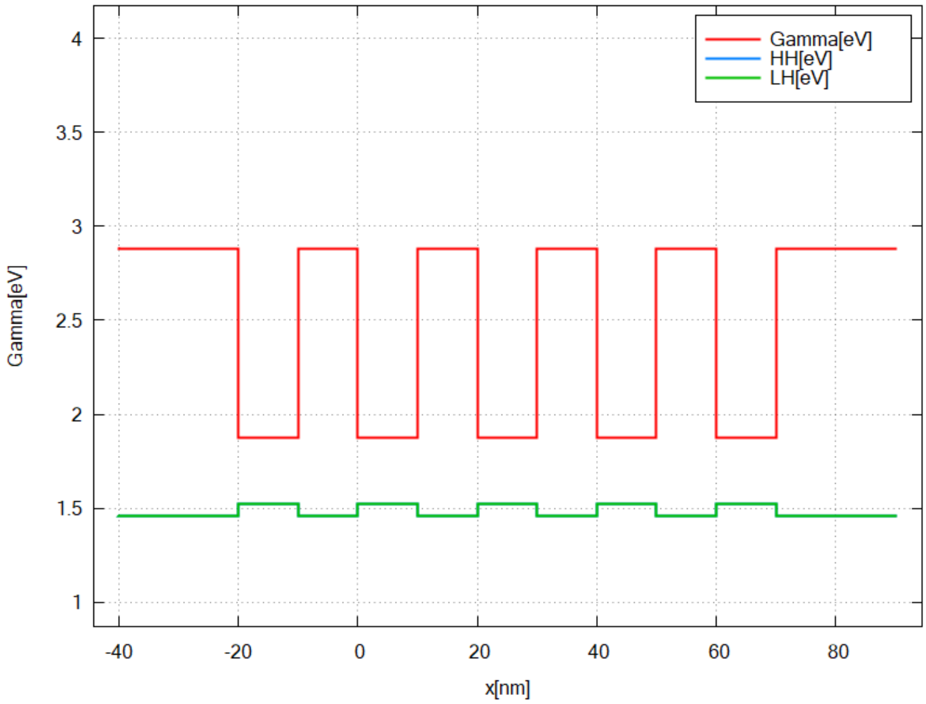


Figure 4.2.1.7: shows energy profile of multiple quantum well structure

Input file 2: Repeated structres

Specifying the structure of the device

```
structure{ # this group is required in every input file
    region{
        binary{ name = GaAs }           # material: GaAs
        contact{ name = whatever }      # contact definition
        everywhere{}                    # ranging over the complete device
    }
    region{
        binary{ name = InAs }           # material: InAs
        line{ x = [ 20.0, 30.0 ] }      # ranging from x=20.0 nm to x=30.0 nm,␣
↪overwrites the previously defined GaAs

        array_x{                        # line{x=[20.0,30.0]} is duplicated in the x␣
↪direction
            shift = 20.0                # the interval  x = [ 20.0, 30.0 ] inside line
↪{} is shifted by an integer multiple of 20.0 nm
            max = 2                     # 2 duplicates in +x direction
            min = -2                    # 2 duplicates in -x direction
        }
        # In short, we are creating 5 InAs regions at positons:
        # line{ x = [20.0+i*shift, 30.0 nm+i*shift]} (min<=i<=max)

        array2_x{
            shift = 120.0               # the structure previously defined inside␣
↪this region
            max = 2                     # is duplicated and shifted by i*120 nm (1<=i
↪<=max) in +x.
        }
    }
}
```

We add the group `array2_x{}` which is used to duplicate the structure defined by `array_x{}` within the same `region{}`. We get a sequence of periodic structures. The usage is analogous to `array_x{}`, thus it follows the same logic with `shift`, `max` and `min`.

Specifying the grid

```
grid{ # this group is required in every input file
    xgrid{                          # grid in x direction
        line{
            pos = -50.0             # start device at x=-50.0 nm
            spacing = 4.0           # grid spacing 4.0 nm
        }
                                    # from x=-50.0 nm to x=-20.0 nm further grid␣
↪points
                                    # are created according to the interpolated␣
↪spacing (4.0 -> 0.5)
                                    # (no equidistant grid spacing)

        line{                       # fixed grid points are created at the bottom␣
↪GaAs/InAs interfaces of every multiple QW structure
            pos = -20.0             # bottom GaAs/InAs interface at x=-20.0 nm
            spacing = 0.5           # grid spacing 0.5 nm

            array{                  # fixed grid point at x=-20 nm is duplicated␣
```

(continues on next page)

(continued from previous page)

```
↪(including spacing)
                shift = 120.0       # shifted by 120.0 nm
                max = 2             # two copies are created at x=-20.0 nm+i*shift (1
↪<=i<=max)
            }
        }

        line{                       # fixed grid points are created in the middle of␣
↪two multiple QW structures to change grid spacing
            pos = 85.0              # position: x=85.0 nm
            spacing = 4.0           # grid spacing 4.0 nm

            array{                  # fixed grid point at x=85.0 nm is duplicated␣
↪(including spacing)
                shift = 120.0       # shifted by 150.0 nm
                max = 1             # one copy is created at x=85.0 nm+max*shift
            }
        }

        line{                       # fixed grid points are created at the top GaAs/
↪InAs interfaces of every multiple QW structure
            pos = 70.0              # top InAs/GaAs interface at x=70.0 nm
            spacing = 0.5           # grid spacing 0.5 nm

            array{                  # fixed grid point at x=70.0 nm is duplicated␣
↪(including spacing)
                shift = 120.0       # shifted by 120.0 nm
                max = 2             # two copies are created at x=70.0 nm+i*shift (1
↪<=i<=max)
            }
        }
                                    # from x=310.0 nm to x=340.0 nm further grid␣
↪points
                                    # are created according to the spacings (0.5 -> 4.
↪0),
                                    # which is interpolated (no equidistant spacing)
        line{
            pos = 340.0             # end device at x=340.0 nm
            spacing = 4.0           # grid spacing 4.0 nm
        }
    }
}
```

In this example, we show that method of arrays also exist for the `grid{ }`. Here, they are called `array{}`, but used equivalently to `array_x{}`. They create copies of one *fixed* grid point, including the related spacing value.

Output

We simulate the device by clicking `F8` on the keyboard. In the related output file you should find a plot of band edges (⇒ `bias_00000` ⇒ `bandedges.dat`) similar to Figure 4.2.1.8.

Just for demonstration, Figure 4.2.1.9 shows a screenshot of the employed grid.

Important things to remember

- Creating periodic structures works as follows: A special array of one template regions (here: one layer) is constructed
- Position and number of new regions are determined by `shift`, `max` and `min`

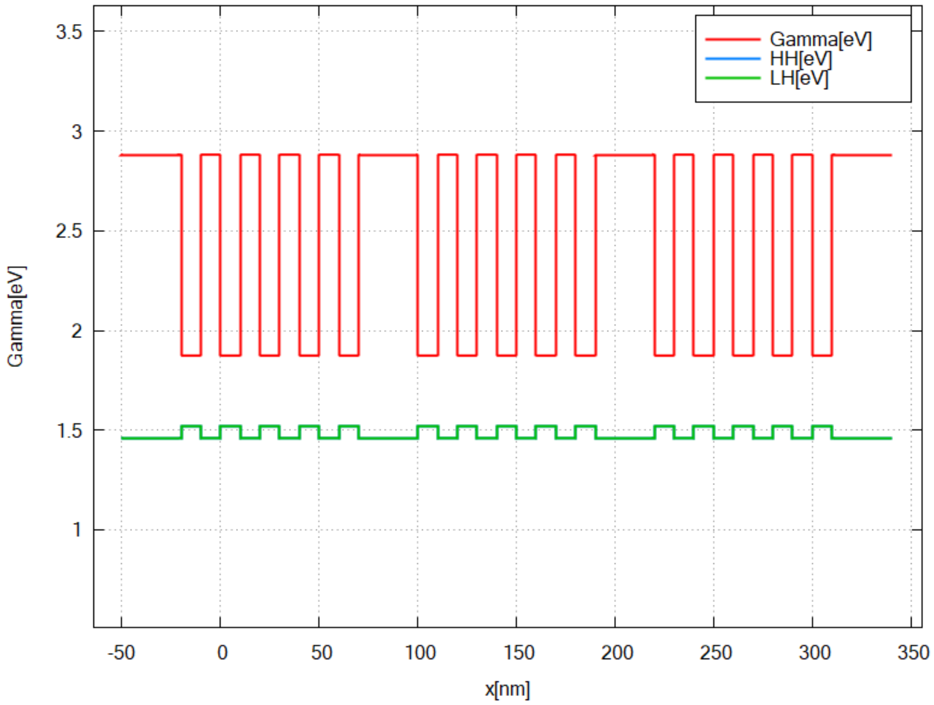


Figure 4.2.1.8: shows the band edges of conduction band at gamma point (Gamma), heavy hole (HH) and light hole (LH) of the complete structure

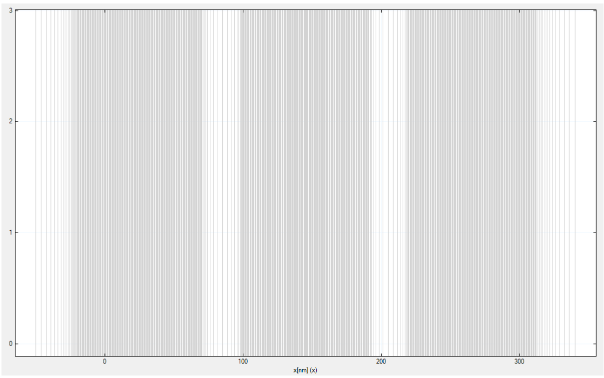


Figure 4.2.1.9: Numerical grid (gray).

- Creating a sequence of periodic structures with `array2{}` works equivalently to `array{}`
- Do not forget to adapt the grid to the complete structure. It is also possible to create an array of grid points.

Last update: 16/07/2024

— FREE — Constant Doping

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *basics_1D_doping_constant_p.in*
- *basics_1D_doping_constant_np.in*

Introduction

This tutorial is the third in our introductory series. We want to show the general framework of adding doping to material regions in *nextnano++*. After completing this tutorial, you will know more about

- adding doping to material regions
- specify the species (donor/ acceptor)

Keywords: `doping{}`, `impurities{ }`, `donor{}`, `acceptor{}`

Overview

As an overview, Figure 4.2.1.10 shows the two structures that will be created in this tutorial.

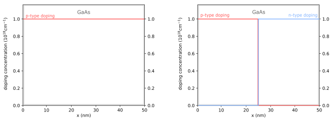


Figure 4.2.1.10: shows p-doped GaAs (left) and p-doped/ n-doped GaAs (right).

The Basics I: Adding doping to bulk material

As an introductory example to doping, we want to n-dope a single GaAs layer as shown on the left of Figure 4.2.1.10. You can use the template input file *basics_1D_doping_constant_p.in*.

Specifying regions with dopants

```
structure{ # this group is required in every input file
    output_impurities{ boxes = yes}             # output doping concentration [10^18␣
↪cm-3]

    region{
        binary{ name = GaAs }                   # material: GaAs
        contact{ name = whatever }              # contact definition
        everywhere{}                            # rangeing over the complete device,␣
↪from x=0.0 nm to x=50.0 nm

        doping{                                 # add doping to the region
            constant{                           # constant doping concentration␣
↪profile
                name = "Custom_impurity_name"   # name of impurity
                conc = 1.0e18                   # doping concentration [cm-3]
            }
        }
    }
}
```

First of all, we create just one thick GaAs layer. Then we add doping to the exact same region by the specifier `doping{}`. Inside `doping{}`, we have to set the doping profile. Here we choose to have constant doping concentration over the whole region. Inside `constant{}` we specify `name` and doping concentration (`conc`) for this region. The name is arbitrary, and you can choose whatever name you like. By giving the doping a reference name, we can select the species and electronic properties for this doping later inside the group `impurities{ }`.

Since we want to inspect the doping concentration distribution for every grid point in the output, the flag `boxes = yes` inside `output_impurities{ }` is active.

Specify impurity species

```
impurities{ # if doped regions exist, this group is required
    acceptor{                           # select the species of dopants
        name = "Custom_impurity_name"   # select doping regions with name = "Custom_
↪impurity_name"
        energy = 0.045                  # ionization energy of dopants [eV]
        degeneracy = 4                  # degeneracy of dopants
    }
}
```

If dopants are added to any region, the group `impurity{}` has to be included in the input file. `acceptor{}` sets the species for regions with name "Custom_impurity_name". We further refine the properties by setting ionization energy (`energy`) and degeneracy level (`degeneracy`).

Output

We simulate the device by clicking `F8` on the keyboard. In the related output folder you should find a plot of the concentration profile (⇒ `Structure` ⇒ `density_acceptor.dat`) as shown in Figure 4.2.1.11.

The Basics II: Adding different doping to bulk material (p-n junction)

As another introductory example, we n-dope the first half and p-dope the second half of the single GaAs layer as in Figure 4.2.1.10 (right). Now, the doping regions do not coincide with the material regions. We have to define material and doping regions separately. You can use the template input file *basics_1D_doping_constant_np.in*.

Specifying regions with dopants

```
structure{ # this group is required in every input file
    output_impurities{ boxes = yes}     # output doping concentration [10^18 cm-3]

```

(continues on next page)

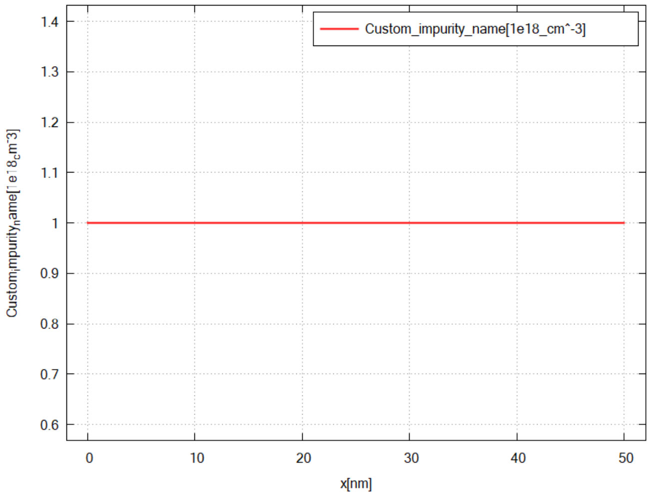


Figure 4.2.1.11: shows the doping concentration of acceptors along the x direction.

(continued from previous page)

```
    region{
        binary{ name = GaAs }         # material: GaAs
        contact{ name = whatever }    # contact definition
        everywhere{}                  # ranging over the complete device, from x=0.
↪0 nm to x=50.0 nm
    }

    region{                           # separate region for adding doping only (no␣
↪material is specified)
        line{ x = [ 0.0, 25.0 ] }     # position: x=0.0 nm to x=25.0 nm
        doping{                       # add doping to the region
            constant{                 # constant doping concentration profile
                name = "p-type"       # name of impurity
                conc = 1.0e18         # doping concentration [cm-3]
            }
        }
    }

    region{                           # separate region for adding doping only (no␣
↪material is specified)
        line{ x = [ 25.0, 50.0 ] }    # position: x=25.0 nm to x=50.0 nm
        doping{                       # add doping to the region
            constant{                 # constant doping concentration profile
                name = "n-type"       # name of impurity
                conc = 1.0e18         # doping concentration [cm-3]
            }
        }
    }
}
```

In the code above, we first create a bulk GaAs layer and then add two *doping regions* for n-type and p-type dopants. The doping regions do not include a material specification. Inside these regions, the position (`line{}`) and the doping (`doping{}`) is specified. The dopants are added to the previously defined material region. In fact, this example illustrates that, as far as the initialization is concerned, *nextnano*++ treats doping and materials separately.

Specify impurity species

```
impurities{ # if doped regions exist, this group is required
    donor{                  # select the species of dopants
        name = "n-type"     # select doping regions with name = "n-type"
        energy = 0.045      # ionization energy of dopants
        degeneracy = 2      # degeneracy of dopants
    }
    acceptor{               # select the species of dopants
        name = "p-type"     # select doping regions with name = "p-type"
        energy = 0.045      # ionization energy of dopants [eV]
        degeneracy = 4      # degeneracy of dopants
    }
}
```

As we already know if dopants are added, the group `impurity{}` has to be included in the input file. Apart from `acceptor{}`, we introduce `donor{}` as another doping species. For both species we refine the properties here.

Output

We simulate the device by clicking `F8` on the keyboard. In the related output folder you should find a plot of the concentration profiles (⇒ `Structure` ⇒ `density_acceptor.dat` / `density_donor.dat`) as shown in Figure 4.2.1.12 and Figure 4.2.1.13.

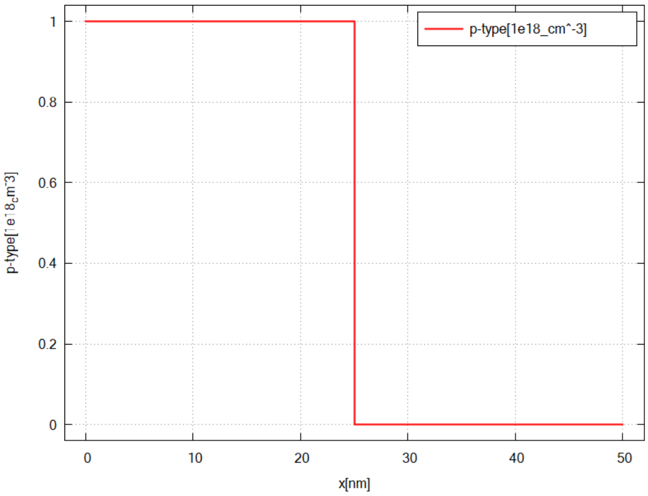


Figure 4.2.1.12: shows the doping concentration of acceptors along the x direction (p-doped region).

Important things to remember

- dopants are part of a region, i.e. structure{...region{...doping{}...}...}. Here you determine the concentration of one impurity type for each grid point.
- The impurity type (species and properties) are defined inside the group impurity{}

Last update: 16/07/2024

— FREE — Adding and Replacing Doping

- *Header*

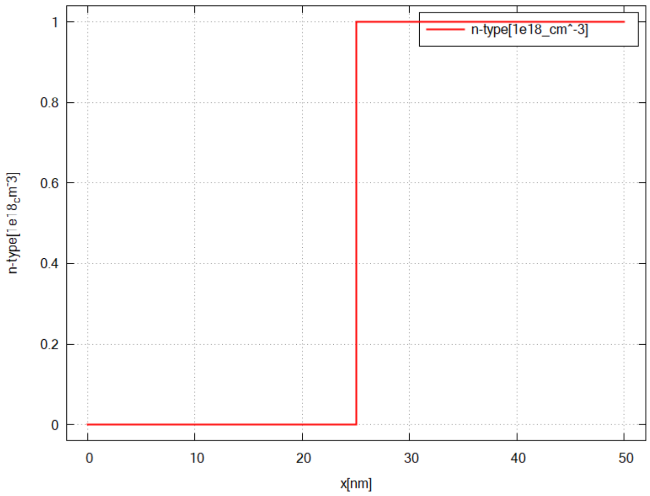


Figure 4.2.1.13: shows the doping concentration of donors along the x direction (n-doped region).

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *basics_1D_doping_adding.in*
- *basics_1D_doping_replacing.in*

Introduction

This tutorial continues our discussion about doping, and extend our basic knowledge gained from *previous tutorial*. After completing this tutorial, you will know more about

- replacing impurities by impurities of the same type
- removing doping
- adding different impurity species to the same region

Overview

The device structures for this tutorial are shown in Figure 4.2.1.14.

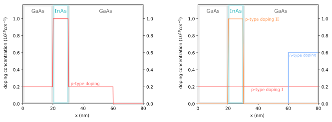


Figure 4.2.1.14: GaAs/InAs/GaAs heterostructure with p-type doping (left) and with different doping (right)

1. Replace and remove doping

We will now consider the structure in Figure 4.2.1.14 (left). You can use the template input file *basics_1D_doping_replacing.in*.

Specifying regions with dopants

```
structure{ # this group is required in every input file
    output_impurities{ boxes = yes}     # output doping concentration [10^18 cm-3]

    region{
        binary{ name = GaAs }           # material: GaAs
        contact{ name = whatever }      # contact definition
        everywhere{}                    # ranging over the complete device, from x=0.0 nm to x=80.0 nm

        doping{                         # add doping to the region
            constant{                   # constant doping concentration profile
                name = "p-type"         # name of impurity
                conc = 2.0e17           # doping concentration [cm-3]
            }
        }
    }

    region{
        binary{ name = InAs }           # region InAs
        line{ x = [ 20.0, 30.0 ] }      # overwriting GaAs at position: x=20.0 nm to x=30.0 nm

        doping{                         # add doping to the region
            constant{                   # constant doping concentration profile
                name = "p-type"         # name of impurity
                conc = 1.0e18           # doping concentration [cm-3]
                add = no                # overwrites previously defined doping with label "p-type"
                                        # Note: the default value is add=yes, which adds
                                        #       dopants to existing dopants
            }
        }
    }

    region{                             # region for deleting dopants
        line{ x = [ 60.0, 80.0 ] }      # position: x=60.0 nm to 80.0 nm
            doping{
                remove{}                # removing all dopants from this region
            }
        }
    }
}
```

In this example, we apply the idea of overwriting previous regions to doping. We first define an p-doped GaAs region with impurity concentration $1.0e18cm^{-3}$ ranging over the whole device. Then, we want to overwrite GaAs in the interval between $x = 20nm$ and $x = 30nm$ with p-doped InAs, with different impurity concentration. However, we have to be careful when applying the idea of overwriting previous regions to doping. By default, the doping is added and not overwritten. To replace the existing doping, it is necessary to use the specifier `add = no`.

If we want to remove all dopants from an interval, as it is the case in the region ranging from $x = 60nm$ to $x = 80nm$, we have to use `remove{}`.

Specify impurity species

```
impurities{ # required if doping exists
    donor{                      # select the species of dopants
        name = "p-type"         # select doping regions with name = "p-type"
        energy = 0.045          # ionization energy of dopants
        degeneracy = 2          # degeneracy of dopants
}
```

Here, we specify to have only p-type impurities in our device.

Output

We simulate the device by clicking `F8` on the keyboard. In the related output folder you should find a plot of the concentration profiles (⇒ `Structure` ⇒ `density_donor.dat`) as shown in Figure 4.2.1.15

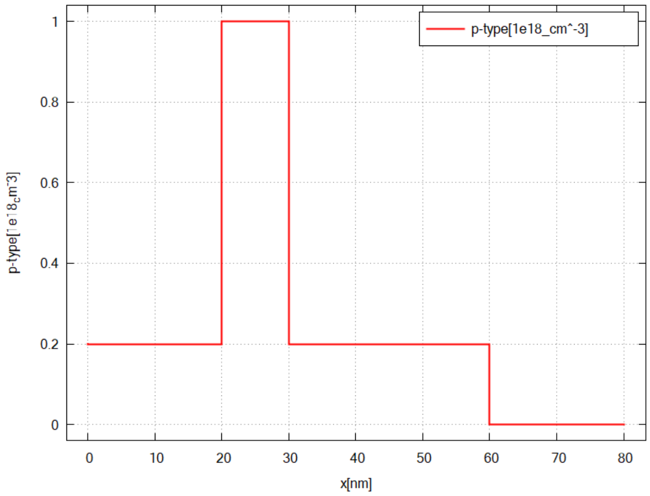


Figure 4.2.1.15: Doping concentration of donors along the x direction.

2. Add different dopants

We will now consider the structure in Figure 4.2.1.14 (right). You can use the template input file *basics_1D_doping_adding.in*.

Specifying regions with dopants

```
structure{ # this group is required in every input file
    output_impurities{ boxes = yes}     # output doping concentration [10^18 cm-3]

    region{
        binary{ name = GaAs }           # material: GaAs
        contact{ name = whatever }      # contact definition
        everywhere{}                    # ranging over the complete device, from x=0.
↪0 nm to x=80.0 nm

        doping{                         # add doping to the region
            constant{                   # constant doping concentration profile
                name = "p-type-I"       # name of impurity
                conc = 2.0e17           # doping concentration [cm-3]
            }
        }
    }
```

(continues on next page)

(continued from previous page)

```
    region{
        binary{ name = InAs }           # region InAs
        line{ x = [ 20.0, 30.0 ] }      # overwriting GaAs at position: x=20.0 nm to␣
↪x=30.0 nm

        doping{                         # add p-doping to the region: the existing "p-
↪type-I" doping is not overwritten
            constant{                   # constant doping concentration profile
                name = "p-type-II"      # name of impurity
                conc = 1.0e18           # doping concentration [cm-3]
            }
        }
    }

    region{                             # region for adding doping
        line{ x = [ 60.0, 80.0 ] }      # position: x=60.0 nm to 80.0 nm

        doping{                         # add n-doping to the region: the existing "p-
↪type-II" doping is not overwritten
            constant{                   # constant doping concentration profile
                name = "n-type"         # name of impurity
                conc = 4.0e17           # doping concentration [cm-3]
            }
        }
    }
}
```

Here, we crete GaAs and InAs each with specific doping. Note that InAs replaces GaAs on the interval x = [20.0, 30.0], while the doping definitions do not influence each other. Also, on the interval x = [60.0, 80.0], n-type doping is simlpy added.

It should be emphasized that the option `doping{...add=no..}` is only applicable to dopants of the same **dopant type**. Remember: a doping type, i.e. chemical element, is associated with one particular `name`. If we wish to replace dopants by a different dopant type, we would need to remove the existing dopants first and then add the new ones.

Specify impurity species

```
impurities{ # required if doping exists
    acceptor{                   # select the species of dopants
        name = "p-type-I"       # select doping regions with name = "p-type-I"
        energy = 0.045          # ionization energy of dopants
        degeneracy = 4          # degeneracy of dopants
    }

    acceptor{                   # select the species of dopants
        name = "p-type-II"      # select doping regions with name = "p-type-II"
        energy = 0.045          # ionization energy of dopants
        degeneracy = 4          # degeneracy of dopants
    }

    donor{                      # select the species of dopants
        name = "n-type"         # select doping regions with name = "n-type"
        energy = 0.045          # ionization energy of dopants
        degeneracy = 2          # degeneracy of dopants
    }
}
```

For every impurity type, we have to add a new `accceptor{}`/ `donor{}` group.

Output

We simulate the device by clicking `F8` on the keyboard. In the related output folder you should find a plot of the concentration profiles (⇒ `Structure` ⇒ `density_donor.dat`) as shown in Figure 4.2.1.16

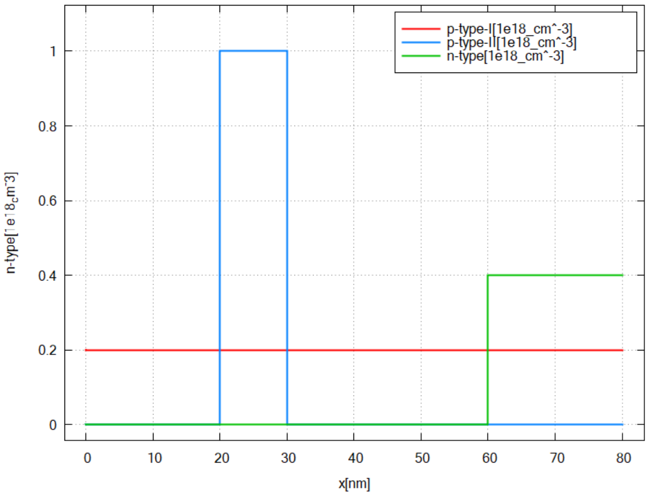


Figure 4.2.1.16: Doping concentration of donors/ acceptors along the x direction.

Important things to remember

- The *nextnano++* tool treats each doping type associated with a particular `name` separately, thus they do not overwrite each other.
- only doping associated with the same `name` can overwrite each other (`add = no`)

Last update: 16/07/2024

— FREE — Doping Functions

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *basics_1D_doping_predefined.in*
- *basics_1D_doping_analytic.in* (not compatible with the free version)

Introduction

This tutorial is the fifth in our introductory series. In the previous tutorials, we've already encountered one predefined doping profile - the constant one. In the following, we will see more possibilities to create doping profiles. After completing this tutorial, you will know more about:

- different doping profiles, namely linear and Gaussian
- crating custom doping profiles

Keywords: `Gaussian1D{}`, `linear{}`, `import{ }`

Overview

As an overview, Figure 4.2.1.17 shows all the structures that will be created in this tutorial.

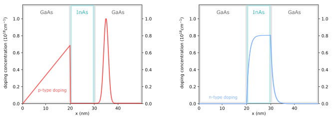


Figure 4.2.1.17: shows doping profiles including linear and Gaussian functions (left) and user defined functions (right).

Using pre-defined doping profiles

In this example we demonstrate two pre-defined doping profiles, namely Gaussian and linear profiles. For that we consider the setup in Figure 4.2.1.17 (left). The associated input file is *basics_1D_doping_predefined.in*.

Specifying regions with dopants

```
structure{ # this group is required in every input file
    output_impurities{ boxes = yes}     # output doping concentration [10^18 cm-3]

    #---------
    # material
    #---------

    region{
        binary{ name = GaAs }           # material: GaAs
        contact{ name = whatever }      # contact definition
        everywhere{}                    # region spreads over the complete device
    }

    region{
        binary{ name = InAs }           # region: InAs
        line{ x = [ 20.0, 30.0 ] }      # position: x=20.0 nm to x=30.0 nm
    }

    #-------
    # doping
    #-------

    region{
        line{ x = [ 30.0, 40.0 ] }      # position: x = 30.0 nm to 40.0 nm
        doping{                         # add doping to the region
            gaussian1D{                 # Gaussian doping concentration profile
                name = "p-type"         # name of impurity
                conc = 1.0E18           # maximum of doping concentration [cm-3]
                x = 35                  # x coordinate of Gauss center
```

(continues on next page)

(continued from previous page)

```
                sigma_x = 1.0            # standard deviation in x direction
            }
        }
    }

    region{
        line{ x = [ 0.0, 20.0 ] }        # position: x = 0.0 nm to 20.0 nm
        doping{                          # add doping to the region
            linear{                      # linear doping concentration profile
                name = "p-type"          # impurity name
                conc = [0, 6.0e17]       # start and end value of doping concentration [cm-3]
                x    = [0.0, 20.0]       # position: x=0.0 nm to x=20.0 nm
            }
        }
    }
}
```

We separated the structural set up in two sections: 1) material and 2) doping. In the doping section we use `linear{}` and `gaussian1D{}` to specify the doping profiles. For defining the Gaussian profile

$$C_{gaussian}(x) = C_{conc}\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}(\frac{x-x_0}{\sigma})^2}$$

with the total doping concentration C_{conc}, coordinate of the maximum x_0 and standard deviation σ, three parameters has to be specified. For defining the linear profile

$$C_{linear}(x) = \frac{C_{end} - C_{start}}{x_{end} - x_{start}} \cdot x + C_{start},$$

we specify start and end value of doping concentration $[y_{start}, y_{end}]$ with the corresponding x coordinates $[x_{start}, x_{end}]$, both as vectors.

Specify impurity species

```
impurities{ # required if doping exists
    acceptor{                    # select the species of dopants
        name = "p-type"          # select doping regions with name = "p-type"
        energy = 0.045           # ionization energy of dopants
        degeneracy = 4           # degeneracy of dopants
    }
}
```

Output

We simulate the device by clicking `F8` on the keyboard. In the related output folder you should find a plot of the concentration profile (⇒ `Structure` ⇒ `density_acceptor.dat`) as shown in Figure 4.2.1.18.

2. Using custom doping profiles

In this example we introduce custom defined doping profiles. For that we consider the set up in Figure 4.2.1.17 (right). The associated input file is *basics_1D_doping_analytic.in*

Defining custom functions

```
import{ # this group is optional
    analytic_function{                   # definition of analytic function
        name = "custom_exp_fun_I"        # name of function
        function = "1e18 *(1-exp(-x+20))"    # define the function
```

(continues on next page)

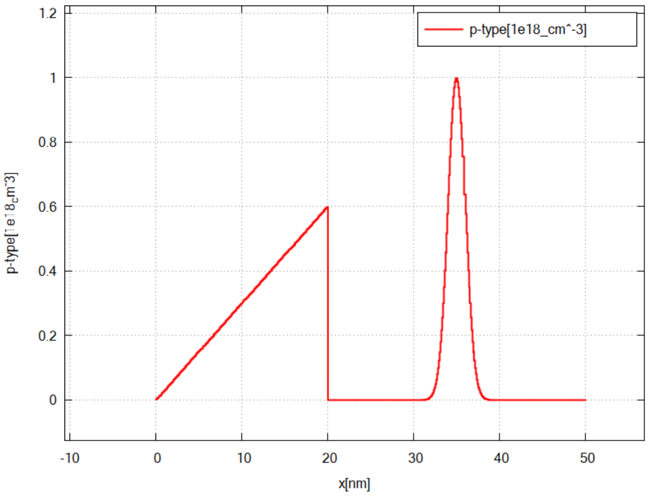


Figure 4.2.1.18: shows the doping concentration of donors along x.

(continued from previous page)

```
    }
    analytic_function{                      # definition of analytic function
        name = "custom_exp_fun_II"          # name of fucntion
        function = "1e18*exp(-x+30)"        # define the function
    }
}
```

In order to create custom doping profiles, we have to define analytical functions in the group `import{ }` first. The analytical expression is given by a string. Later, we can incorporate these functions for adding doping by referring to the corresponding `name`.

Specifying regions with dopants

```
structure{ # this group is required in every input file
    output_impurities{ boxes = yes}                 # output doping concentration [10^
↪18 cm-3]

    #---------
    # material
    #---------

    region{
        binary{ name = GaAs }                       # material: GaAs
        contact{ name = whatever }                  # contact definition
        everywhere{}                                # region spreads over the␣
↪complete device
    }

    region{
        binary{ name = InAs }                       # region: InAs
        line{ x = [ 20.0, 30.0 ] }                  # position: x=20.0 nm to x=30.0 nm
                                                    # overwrites the previously␣
↪defined GaAs region
    }

    #-------
    # doping
```

(continues on next page)

(continued from previous page)

```
    #-------

    region{                                         # region: adds doping
        line{ x = [ 20.0, 30.0 ] }                  # position: x=20.0 nm to x=30.0 nm
        doping{
            import{                                 # reference to import{ } group,␣
↪where custom functions are defined
                name = "n-type"                     # name of impurity
                import_from = "custom_exp_fun_I"    # import doping profile: custom_
↪exp_fun_I
            }
        }
    }

    region{                                         # region: adds doping
        line{ x = [ 30.0, 50.0 ] }                  # position: x=30.0 nm to x=50.0 nm
        doping{
            import{                                 # reference to import{ } group,␣
↪where custom functions are defined
                name = "n-type"                     # name of impurity
                import_from = "custom_exp_fun_II"   # import doping profile: custom_
↪exp_fun_II
            }
        }
    }
}
```

Inside `doping{}`, the previously defined functions are used to create custom doping profiles. We import each function (`import_from`) from the group `import{ }` by referring to the name that we had assigned. The function is then evaluated on the interval specified inside `line{}` yielding the final doping profile.

Besides the shape of the doping profile we also specify the `name`, as usually.

Specify impurity species

```
impurities{ # required if doping exists
    acceptor{                   # select the species of dopants
        name = "p-type"         # select doping regions with name = "p-type"
        energy = 0.045          # ionization energy of dopants
        degeneracy = 4          # degeneracy of dopants
    }
}
```

Output

We simulate the device by clicking `F8` on the keyboard. In the related output folder you should find a plot of the concentration profile (⇒ `Structure` ⇒ `density_donor.dat`) as shown in Figure 4.2.1.19.

Important things to remember

- before importing and using our own functions, we first have to define them in the `import{ }` group

---

Last update: 16/07/2024

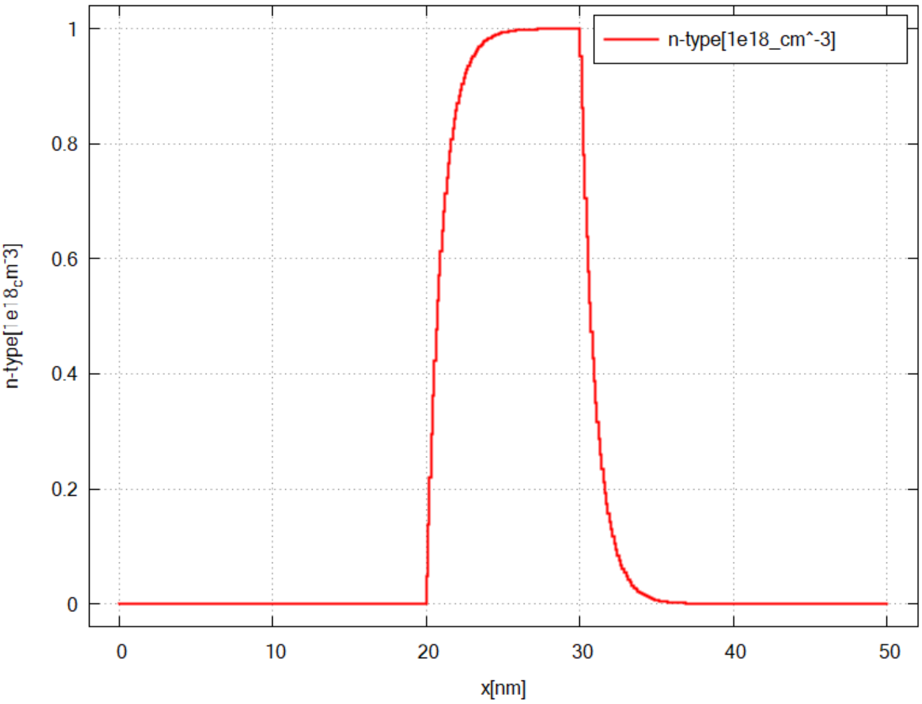


Figure 4.2.1.19: The doping concentration of donors along the x direction.

— FREE — Doping in Heterostructure

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *basics_1D_doping_heterostructure.in*

Introduction

This tutorial is an example of defining a heterostructure with multiple doping regions (Figure 4.2.1.20). The device structure is shown in Figure 4.2.1.20.

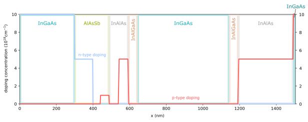


Figure 4.2.1.20: shows structure with doping profile

Specifying the structure

Inside the group `structure{ }`, we've separated the code into two blocks. In the first we defined material regions, and in the second we added doping.

```
    # ---------
    # materials
    # ---------

    region{
        ternary_constant{                   # constant alloy composition
            name = "In(x)Ga(1-x)As"         # material: InGaAs
            alloy_x = 0.5                   # alloy composition
        }
        contact{ name = whatever }          # contact definition
        everywhere{}                        # ranging over the complete device, from␣
↪x=0.0 nm to x=1503.0 nm
    }

    region{
        ternary_constant{                   # constant alloy composition
            name = "AlAs(x)Sb(1-x)"         # material: AlAsSb
            alloy_x = 0.85                  # alloy composition
        }
        line{ x = [ 300.0, 488.0] }         # overwriting InGaAs in the interval from␣
↪x = 300.0 nm to  x=488.0 nm
    }

    region{
        ternary_constant{                   # constant alloy composition
            name = "Al(x)In(1-x)As"         # material: AlInAs
            alloy_x = 0.5                   # alloy composition
        }
        line{ x = [ 488.0, 1493.0] }        # overwriting InGaAs in the interval from␣
↪x = 388.0 nm to x=1493.0 nm
    }

    region{
        quaternary_constant{                # constant alloy composition
            name = "Al(x)Ga(y)In(1-x-y)As"  # material: AlGaInAs
            alloy_x = 0.4                   # alloy composition
            alloy_y = 0.2                   # alloy composition
        }
        line{ x = [ 593.0, 1193.0] }        # overwriting AlInAs in the interval from␣
↪x = 593.0 nm to x=1193.0 nm
    }

    region{
        ternary_constant{                   # constant alloy composition
            name = "In(x)Ga(1-x)As"         # material: InGaAs
            alloy_x = 0.5                   # alloy composition
        }
        line{ x = [ 643.0, 1143.0] }        # overwriting AlInAs in the interval from␣
↪x = 643.0 nm to x=1143.0 nm
    }
```

There are often many ways to create a desired structure. However, utilizing the symmetry of a structure can sometimes simplify things. In the code above we for example, we try to omit defining each material layer separately.

Instead, we defined the outer material layers as one region and then overwriting it inside by the next inner layers. Thus, we do not have to define the two InGaAs or InAlGaAs regions separately.

```
    # ------
    # doping
    # ------

    region{                             # region for adding doping
        line{ x = [0.0, 300.0]}         # position: x=0.0 nm to 300.0 nm

        doping{
            constant{                   # constant doping concentration profile
                name = "n-type-doping"  # name of impurity
                conc = 1.0e19           # doping concentration [cm-3]
            }
        }
    }

    region{                             # region for adding doping
        line{ x = [300.0, 400.0]}       # position: x=300.0 nm to 400.0 nm
        doping{
            constant{                   # constant doping concentration profile
                name = "n-type-doping"  # name of impurity
                conc = 5.0e18           # doping concentration [cm-3]
            }
        }
    }

    region{                             # region for adding doping
        line{ x = [440.0, 484.0]}       # position: x=440.0 nm to 484.0 nm
        doping{
            constant{                   # constant doping concentration profile
                name = "p-type-doping"  # name of impurity
                conc = 1.0e18           # doping concentration [cm-3]
            }
        }
    }

    region{                             # region for adding doping
        line{ x = [534.0, 589.0]}       # position: x=534.0 nm to 589.0 nm
        doping{
            constant{                   # constant doping concentration profile
                name = "p-type-doping"  # name of impurity
                conc = 5.0e18           # doping concentration [cm-3]
            }
        }
    }

    region{                             # region for adding doping
        line{ x = [1193.0, 1493.0]}     # position: x=1193.0 nm to 1493.0 nm
        doping{
            constant{                   # constant doping concentration profile
                name = "p-type-doping"  # name of impurity
                conc = 5.0e18           # doping concentration [cm-3]
            }
        }
    }

```

(continues on next page)

(continued from previous page)

```
    region{                                 # region for adding doping
        line{ x = [1493.0, 1503.0]}         # position: x=1493.0 nm to 1503.0 nm
        doping{
            constant{                       # constant doping concentration profile
                name = "p-type-doping"      # name of impurity
                conc = 1.0e19               # doping concentration [cm-3]
            }
        }
    }
```

We define each doping region one at a time: first n-type regions and then p-type regions.

Specify impurity species

```
impurities{ # required if doping exists
    donor{                          # select the species of dopants
        name = "n-type-doping"      # select doping regions with name = "n-type-doping"
        energy = 0.045              # ionization energy of dopants
        degeneracy = 2              # degeneracy of dopants
    }
    acceptor{                       # select the species of dopants
        name = "p-type-doping"      # select doping regions with name = "p-type-doping"
        energy = 0.045              # ionization energy of dopants
        degeneracy = 4              # degeneracy of dopants
    }
}
```

Last update: 16/07/2024

— FREE — Variables

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *basics_1D_variables.in*

Introduction

This tutorial teaches how to use variables in the input file. Besides their advantages for the code, e.g. enhance flexibility, creating dependencies between parameters, etc., variables enable performing parameter sweeps in *nextnano*++. After completing this tutorial, you will know more about:

- defining variables
- common usage of variables in *nextnano*++

In this tutorial we want to create a GaAs/InAs/GaAs single quantum from tutorial 1 well once again, this time using variables.

Defining variables

```
# Independant variables
#----------------------

$device_start = 0.0              # device starts at x = 0.0 nm (DisplayUnit:nm)
$device_length = 50.0            # device ranges from  $device_start to $device_start + $device_length (DisplayUnit:nm)
$InAs_width = 20.0               # thickness of InAs layer (DisplayUnit:nm) (ListOfValues:5.0, 10.0, 20.0)

$grid_spacing_fine = 0.5         # fine grid spacing value (DisplayUnit:nm)
$grid_spacing_course = 2.0       # coarse grid spacing value (DisplayUnit:nm)

# Derived variables
#------------------

$InAs_start = $device_start  + ( $device_length - $InAs_width )/2    # calculating start position of InAs layer (InAs layer should be centered around the middle of the device) (DisplayUnit:nm) (DoNotShowInUserInterface)
$InAs_end = $device_start  + ( $device_length + $InAs_width )/2      # calculating end position of InAs layer (DisplayUnit:nm) (DoNotShowInUserInterface)
```

Variables start with the character “$” followed by their name. A good practice is place the variables at the beginning of the input file. In the example we see one major application for variables in *nextnano*++, namely the structural design. Since we are now able to define dependencies between parameters explicitly, three variables - `$xmin`, `$device_length` and `$InAs_width` - set up the complete device structure.

The comments (DisplayUnit: …), (ListOfValues: …) and (DoNotShowInUserInterface) are important for parameter sweeps which we will discuss later. The purpose of these three specifiers in particular are to display the unit of the variable in the sweep interface, to give a list of sweep values and to exclude a variable from the sweep interface.

Specifying the grid

```
grid{ # this group is required in every input file
    xgrid{                                      # grid in x direction
        line{
            pos = $device_start                 # assign start position of device (x=0.0 nm)
            spacing = $grid_spacing_fine        # assign course grid spacing (4.0 nm)
        }
        line{
            pos = $InAs_start                   # assign grid point at GaAs/InAs interface (20.0 nm)
            spacing = $grid_spacing_course      # assign fine grid spacing (0.5 nm)
        }
        line{
            pos = $InAs_start                   # assign grid point at InAs/GaAs interface (30.0 nm)
            spacing = $grid_spacing_course      # assign fine grid spacing (0.5 nm)
        }
        line{
            pos = $device_start+$device_length  # assign end position of device (x=50.0 nm)
            spacing = $grid_spacing_fine        # assign course grid spacing (4.0 nm)
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

The grid is now completely derived from the variables. Now, if some variables are changed, we ensure that the grid is adapted to the structure of the device.

Specifying the structure

```
structure{
    region{
        binary{ name = GaAs }                   # GaAs region
        contact{ name = whatever }              # contact definition
        everywhere{}                            # region spreads over the complete␣
↪device (from $device_start to $device_start+$device_length)
    }
    region{
        binary{ name = InAs }                   # InAs region
        line{ x = [ $InAs_start , $InAs_end ] } # derived position of InAs layer
                                                # overwrites the previously defined␣
↪GaAs region
    }
}
```

We assign the previously derived variables for the position of the InAs layer to the corresponding region.

Application: Performing a parameter sweep

For performing a parameter sweep, it was necessary to introduce variables. Now, we want to show how to sweep through the InAs layer thickness and then output the simulated energy profiles.

The first step is to initialize the sweep. Under the tab *Template* in *nextnanomat* we load the currently opened input file by clicking ref: icon (Figure 4.2.1.21). Then we select *list of values* and the variable `$InAs_width` which should be swept. Since we specified a list of values for `$InAs_width` in the input file, the list is automatically inserted. Then we have to create the input files for each value in the list. By clicking `create input file` they are added to the batch list. The second step is to run all files from the batch list by pressing `F10`.

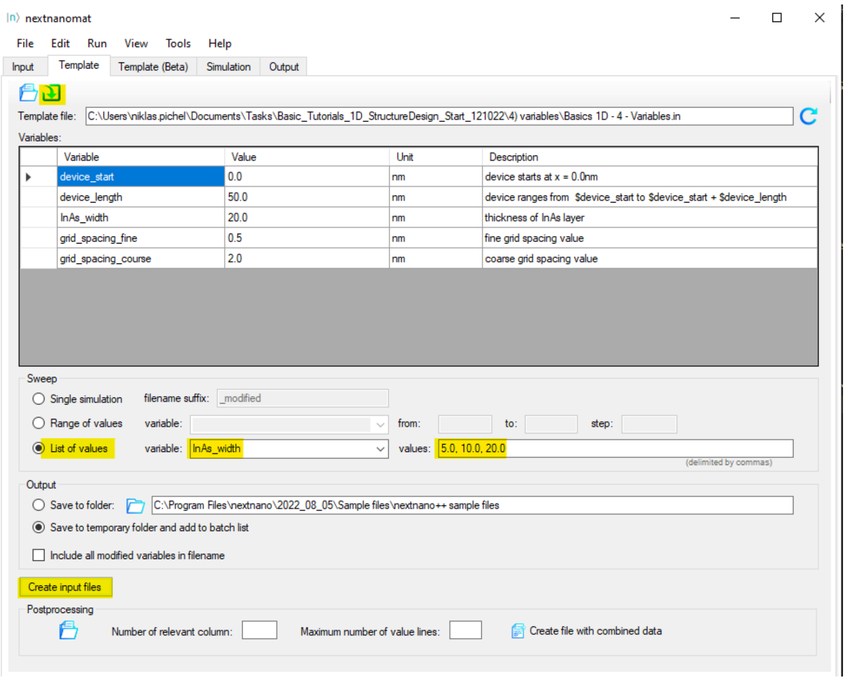
After running the simulation you should find an output folder for every sweep value: `basics_1D_variables_InAs_width_<SweepValue>`. Figure 4.2.1.22 shows the overlay of energy profiles from every sweep.

Important things to remember

- Variables are defined by “*$*” + “Name of variable”

Last update: 16/07/2024

Importing files



Template file: C:\Users\niklas.pichel\Documents\Tasks\Basic_Tutorials_1D_StructureDesign_Start_121022\4) variables\Basics 1D - 4 - Variables.in

Variables:

Variable	Value	Unit	Description
device_start	0.0	nm	device starts at x = 0.0nm
device_length	50.0	nm	device ranges from $device_start to $device_start + $device_length
InAs_width	20.0	nm	thickness of InAs layer
grid_spacing_fine	0.5	nm	fine grid spacing value
grid_spacing_course	2.0	nm	coarse grid spacing value

Sweep

○ Single simulation filename suffix: _modified

○ Range of values variable: from: to: step:

● List of values variable: InAs_width values: 5.0, 10.0, 20.0 (delimited by commas)

Output

○ Save to folder: C:\Program Files\nextnano\2022_08_05\Sample files\nextnano++ sample files

● Save to temporary folder and add to batch list

☐ Include all modified variables in filename

Postprocessing

Number of relevant column: Maximum number of value lines: Create file with combined data

Figure 4.2.1.21: Screenshot showing *nextnanomat* interface to initialize the sweep:

1. load input file,
2. select variable and list of values for the sweep,
3. create new input files (saved to temporary folder)

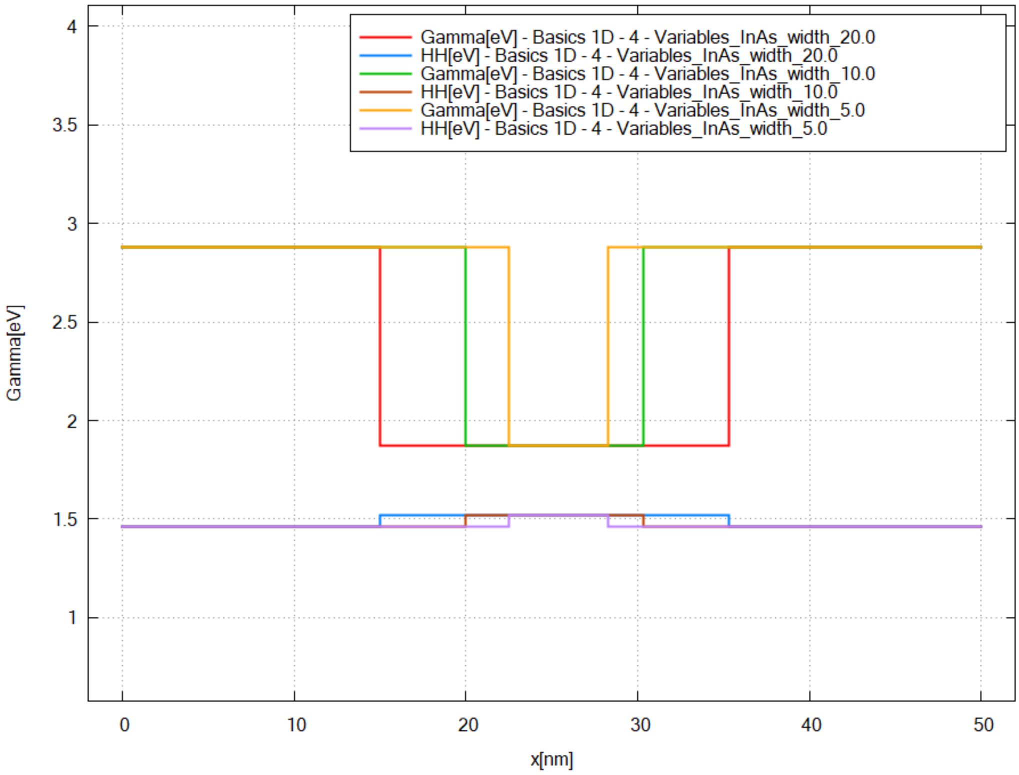


Figure 4.2.1.22: Overlay of energy profiles (conduction band at Γ and heavy hole valence band) corresponding to different InAs layer widths

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *import-dat_1D_nnp.in* - importing *.dat files to 1D simulation
- *import-dat_2D_nnp.in* - importing *.dat files to 2D simulation
- *import-dat_3D_nnp.in* - importing *.dat files to 3D simulation

Scope of the tutorial:

This tutorial is presenting how to import various files to *nextnano*++ simulations. The examples cover importing electric potential, alloy contents, and strain for 1D, 2D, and 3D simulations from *.dat files.

Relevant output Files:

- *bias_00000\bandedges.dat*
- *Imports\Ternary_alloy.dat*
- *Imports\Strain_Tensor.dat*
- *Imports\Potential.dat*

Introduced Keywords:

- `import{ directory file{ filename format } }`
- `region{ ternary_import{ } }`
- `strain{ import_strain{ } }`
- `poisson{ import_potential{} }`

Importing data

Reading external files

The pivotal group responsible for importing files for simulations with *nextnano*++ is the group *import{ }*. Its purpose for this tutorial is to inform *nextnano*++ about:

- the location of a selected file,
- format of the file,
- name of the file,
- how to refer to the file,
- whether the data should be rescaled.

EXAMPLE 1. Importing a file from the location of the input file

Let us say that one has an input file `C:\input_files\my_input_file.in`. Having the following script in the file

```
import{
    file{
        name = "some_imported_data"
        filename = "my_alloy_from_XRF"
        format = DAT
    }
}
```

results in *nextnano++* trying to access and read a file `C:\input_files\my_alloy_from_XRF.dat`. The file can be used in the input file under the name `some_imported_data`.

EXAMPLE 2. Importing a file from an arbitrary location

Giving that the data to import is stored elsewhere, one just needs to define `import{ directory }` attribute to navigate *nextnano++* to the location of the file to import. The script

```
import{
    directory = "D:\\my_precious_measurements\\"
    file{
        name = "some_imported_data"
        filename = "my_alloy_from_XRF"
        format = DAT
    }
}
```

instructs *nextnano++* to access and read `D:\my_precious_measurements\my_alloy_from_XRF.dat`.

Hint

It is also allowed to write `directory = "D:\my_precious_measurements\"` instead of `directory = "D:\\my_precious_measurements\\"`

Note

The input files prepared for this tutorial have `import{ directory = "./"}` specified which is equivalent to not specifying `directory` attribute at all.

EXAMPLE 3. Importing a file and rescaling

Let us assume that somebody has prepared a file containing alloy content defined in the range from 0 to 100. The *nextnano++* tool requires the content to be imported as mole fraction, therefore, defined in the range from 0 to 1. To import data from such a file one can use a scaling factor `0.01` which will be used while reading the file. Running the following script

```
import{
    directory = "D:\\my_precious_measurements\\"
    file{
        name = "some_imported_data"
        filename = "my_alloy_from_XRF"
        format = DAT
        scale = 0.01
    }
}
```

results in *nextnano*++ rescaling all the imported values (except the domain, coordinates) by multiplying them by `0.01`. Therefore, a data for 2D simulation

coord-x	coord-y	alloy-x
0	3	10
5	5	25
20	6	70

will be read as

coord-x	coord-y	alloy-x
0	3	0.1
5	5	0.25
20	6	0.7

To use imported file in the simulation, one needs to use the reference name specified by `import{ file{ name } }` in other proper places in the input file.

Electric potential

For this tutorial we provide you with three files containing electric potential for importing *import-dat_1D_nnp_potential.dat*, *import-dat_2D_nnp_potential.dat**, and *import-dat_3D_nnp_potential.dat* for 1D, 2D, and 3D simulations, respectively. Let us consider 1D simulation for simplicity; 2D and 3D cases are similar.

EXAMPLE 4. Importing electric potential from a `*.dat` file

The :*import_1D_dat_nnp_potential.dat* is imported in the input file :*import-dat_1D_nnp.in* as follows.

```
import{
    file{
        name = "imported_potential"
        filename = "import-dat_1D_nnp_potential.dat"
        format = DAT
    }
}
```

It allows *nextnano*++ to use the data through the name `"imported_potential"` elsewhere. As the electric potential is related to the Poisson equation, one needs to use the name inside a nested group `poisson{ import{ } }` in order to inform the tool that these data should be used as an electric potential. The relevant piece of script in the :*import_1D_dat_nnp.in* is:

```
poisson{
    import_potential{
        import_from = "imported_potential"
    }
}
```

Strain tensor

You can apply these manners to the other parameters, such as, strain and potential.

```
import{

    file{
        name = "imported_strain"                    # name for referencing the␣
↪imported data in the input file
```

(continues on next page)

(continued from previous page)

```
        filename = "import-dat_1D_nnp_strain.dat"   # name of file which is imported
        format = DAT                                # format of the file to be␣
↪imported. At the moment only AVS format and a simple .dat format is supported.
    }
}
```

```
strain{
    import_strain{
        import_from = "imported_strain"             # reference to imported data in␣
↪import{ }. The file being imported must have exactly six data components
                                                    # expected order of tensor␣
↪components is: e_11, e_22, e_33, e_12, e_13, e_23.
    }
    output_strain_tensor{
        simulation_system = yes
        crystal_system    = yes
    }
}
```

In the case, the import file has only one column (x) of a coordinate. Number of required columns of coordinate depends on dimensionality of the simulation, 2 columns (x and y) are necessary for 2D simulation and 3 columns (x, y, and z) for 3D simulation. Additionally, the file contains 6 tensor components, ε_{11}, ε_{22}, ε_{33}, ε_{12}, ε_{13}, and ε_{23}, each in separate collumn.

Alloy compositions

```
structure{
    output_alloy_composition{}
    region{
        line{
            x = [0, 16]
        }
        ternary_import{
            name = "Al(x)Ga(1-x)As"             # ternary material name for this␣
↪region which uses imported alloy profile
            import_from = "imported_ternary"    # reference to imported data in import
↪{ }. The file being imported must have exactly one data component (x)
        }
    }
}
```

As `ternary_import{ }` is used to import alloy profile, *imported_ternary* file contains information about alloy profile. The file has the following data.

x-coord	alloy_parameter
4	15
12	30

The "alloy_parameter" should be ≤ 1, therefore, `import{ file{ scale } }` is necessary to be consistent with that.

Once you import a file, you can use it multiple times.

```
import{

```

(continues on next page)

(continued from previous page)

```
    file{
        name = "imported_quaternary"                    # name for referencing the␣
↪imported data in the input file
        filename = "import-dat_1D_nnp_quaternary.dat"   # name of file which is␣
↪imported
        format = DAT                                    # format of the file to be␣
↪imported.
                                                        # At the moment only AVS␣
↪format and a simple .dat format is supported.
    }
}
```

```
structure{

    region{
        line{
            x = [17, 33]
        }
        quaternary_import{
            name = "Al(x)Ga(y)In(1-x-y)As"          # quaternary material name for␣
↪this region which uses imported alloy profile
            import_from = "imported_quaternary"     # reference to imported data in␣
↪import{ }.
                                                    # sThe file being imported must␣
↪have exactly two data components (x,y).
        }
    }
    region{
        line{
            x = [34, 50]
        }
        quaternary_import{
            name = "Al(x)Ga(1-x)As(y)Sb(1-y)"   # quaternary material name for this␣
↪region which uses imported alloy profile
            import_from = "imported_quaternary"    # reference to imported data in␣
↪import{ }.
                                                # The file being imported must have␣
↪exactly two data components (x,y).
        }
    }
}
```

In the code, you are using *import-dat_1D_nnp_quaternary.dat* file twice to specify those alloy compositions.

Imported data in the simulation

`import{ output_imports{} }` outputs all imported data including scale factors. The filenames of the outputs correspond to the ones defined `import{ file{ name } }`.

Electric potential

Attention

Prepared input files are **not** solving the Poisson equation.

The Figure 4.2.1.23 shows the potential defined in the import files.

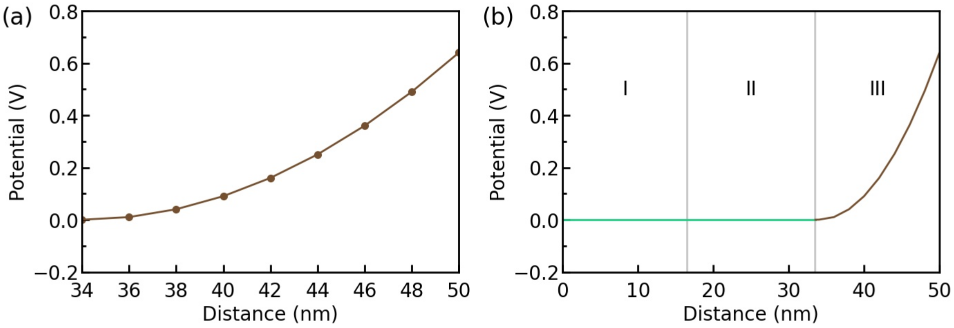


Figure 4.2.1.23: The potential introduced from the import file. The resulting potential in the entire structure.

Strain tensor

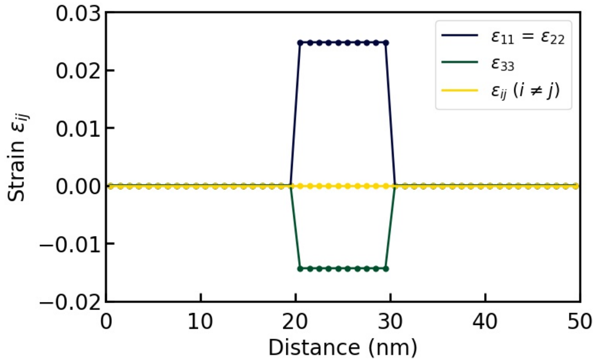


Figure 4.2.1.24: Imported strain tensor.

Alloy compositions

Figure 4.2.1.25 shows the alloy compositions in each region defined in the import files (a), (b) and the input file (c).

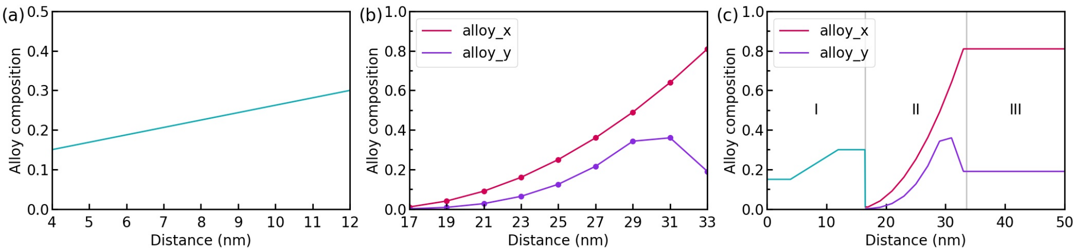


Figure 4.2.1.25: The alloy composition of Al(x)Ga(1-x)As is shown in (a). The alloy composition of Al(x)Ga(y)In(1-x-y)As is shown in (b) (The violet line: x, The purple line: y). (c) shows the alloy compositions in the whole structure. Region I is Al(x)Ga(y)In(1-x-y)As, region II is Al(x)Ga(y)In(1-x-y)As, and region III is Al(x)Ga(1-x)As(y)Sb(1-y), respectively.

The grid points in Figure 4.2.1.25 are originated from the import files.

There are some important points you can see from Figure 4.2.1.25 (c). First, you should be aware that the values between grid points are interpolated linearly. Therefore, the composition between the region I and the region II steeply drops. Second, the regions in which any date is not specified in import files are interpolated by constants. As the composition of the region III is not specified in the import files, it has continuously taken over the value at the boundary between the region II and the region III.

Resulting bandedges

At last, we briefly check the band edges of the structure (Figure 4.2.1.26).

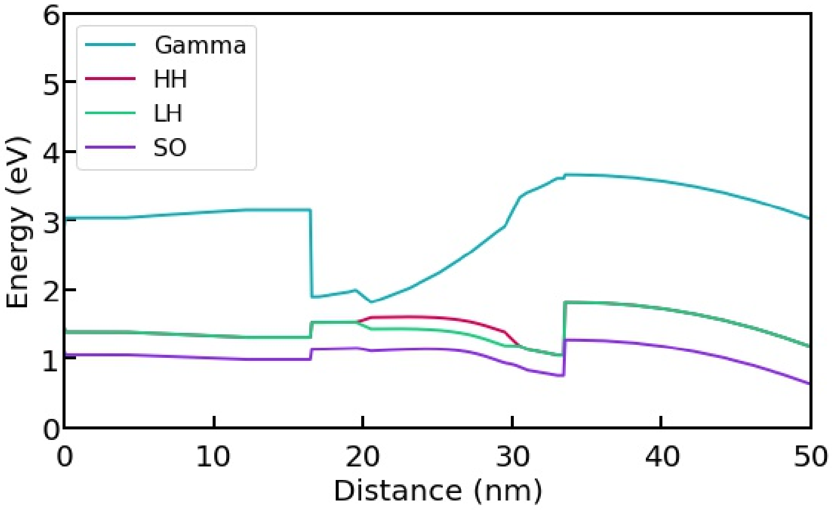


Figure 4.2.1.26: The band edges of the structure. The HH band and the LH band are degenerated in the region where there is no strain.

HH, LH, and SO band mean heavy hole, light hole, and split off band, respectively. The Fermi level is set to 0 eV You can refer to *Definition of Band Offsets (zincblende)* for further knowledge about band offsets.

2D and 3D simulations

In the 2D simulation, you can import files in the same manners as in the 1D simulation. Of course, the import files have to be 2 dimensional.

Figure 4.2.1.27 shows the geometry of the materials used in this simulation.

Here, we look at the alloy compositions of the materials as an example of a 2D import file. *2D_ternary_alloy.dat*, *2D_quaternary_alloy.dat* are imported for specifying the alloy compositions for the materials above.

Attention

In this tutorial we are assuming always that the imported data is defined on a domain or subspace of the simulation domain. Therefore, the number of dimensions of the domain of the imported data is always assumed to be the same as of the simulation, e.g., **2D** simulation imports data with **two** first columns standing for **x** and **y** coordinates. If you need a tutorial covering such case, let us know here.

Last update: 16/07/2024

4.2.2 Contacts and Boundary Conditions

This will be a set of tutorials teaching basics on how to define and choose boundary conditions for your simulations to represent various physical scenarios at the boundaries of your simulation. Currently, you can find here only one tutorial, for the Schottky contact, which will be later split and expanded into multiple more specific tutorials.

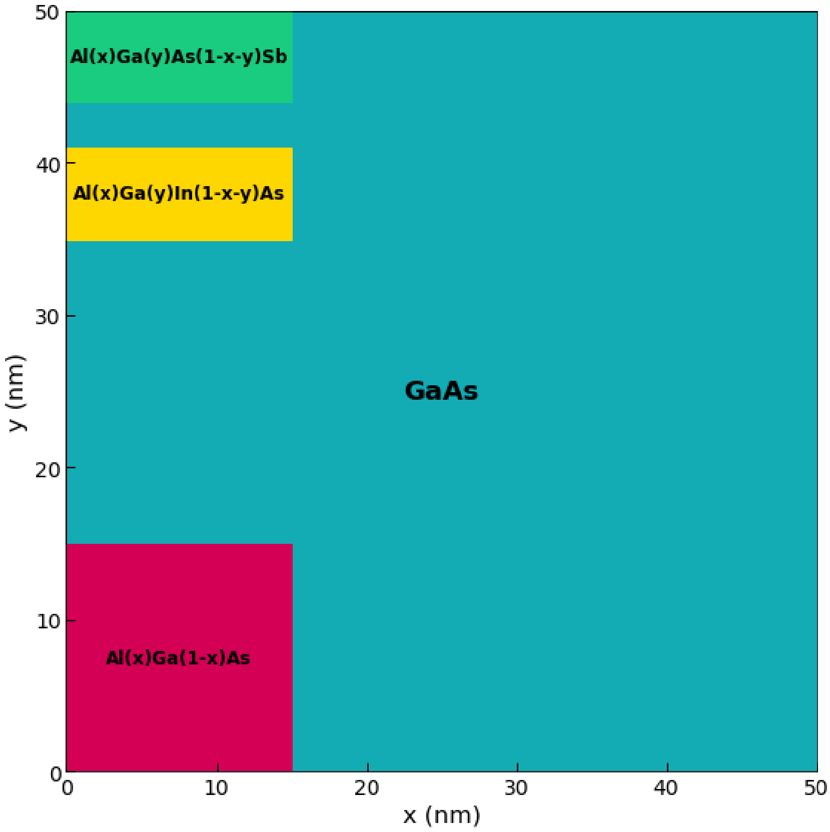


Figure 4.2.1.27: The geometry of the materials used in the 2D simulation. The dashed line is along x = 7.5 nm.

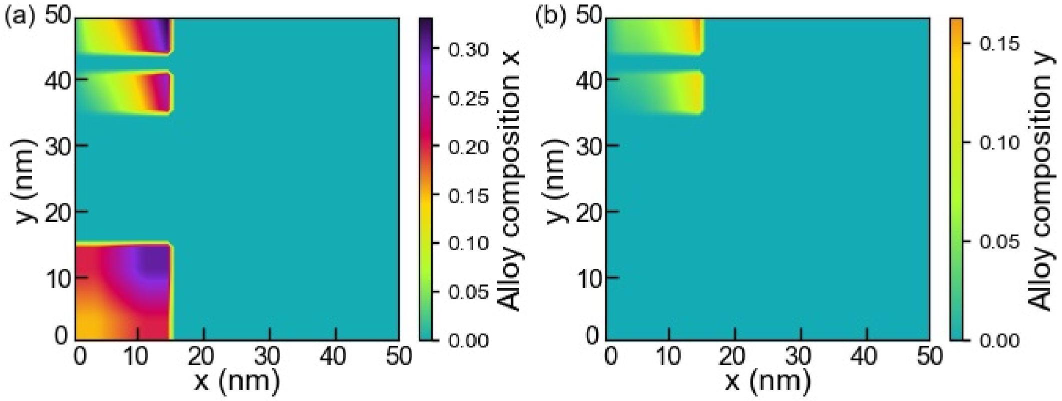


Figure 4.2.1.28: The alloy compositions of (a) x and (b) y of the materials used in the 2D simulation.

— FREE — Schottky Barrier

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *contacts_1D_ohmic_charge_neutral_GaAs_nnp.in*
- *contacts_1D_schottky_barrier_GaAs_nnp.in*
- *contacts_2D_schottky_barrier_GaAs_nnp.in* (not compatible with the free version)

Scope:

The Schottky barrier at the boundary of simulation domain.

Introduction

When a metal is in contact with a semiconductor, a potential barrier is formed at the metal-semiconductor interface. In 1938, Walter Schottky suggested that this potential barrier arises due to stable space charges in the semiconductor. At thermal equilibrium, the Fermi levels of the metal and the semiconductor must coincide. There are two limiting cases:

a) **Ideal Schottky barrier:**

- metal/n-type semiconductor: The barrier height ϕ_B is the difference of the metal work function ϕ_M and the electron affinity (χ) in the semiconductor.

$$\mathrm{e}\phi_B = \mathrm{e}(\phi_M - \chi_s)$$

- metal/p-type semiconductor: The barrier height $\phi_{B,p}$ is given by:

$$\mathrm{e}\phi_{B,p} = \mathrm{e}(\phi_M - \chi_s) - E_{\mathrm{gap}}$$

b) **Fermi level pinning:**

If surface states on the semiconductor surface are present: The barrier height is determined by the property of the semiconductor surface and is independent of the metal work function

Attention

Note that this approach have physical sense only for structures that are not biased, in global equilibrium.

Consequently, the Schottky barrier corresponds a (Dirichlet) boundary condition for the electrostatic potential, i.e. the solution of the Poisson equation in the semiconductor, because the conduction and valence band edge energies are in a definite energy relationship with the Fermi level of the metal.

```
contacts{
    schottky{                   # Schottky barrier
        name = contact
        bias = 0.0              # [V] apply voltage
```

(continues on next page)

(continued from previous page)

```
        barrier = 0.53          # [V] GaAs, S.M. Sze, "Physics of Semiconductor␣
↪Devices", p. 275 (2nd ed.)
    }
}
```

Schottky Barrier

All input files in this tutorial assume n-type donor concentration in $GaAs$ has to be $1 \cdot 10^{19}$ cm^{-3} with realistic *activation energies*. With the temperature set to 300 K this effects in having the Fermi level in the conduction band of the n-doped GaAs, see *bandedges.dat* output by *contacts_1D_ohmic_charge_neutral_GaAs_nnp.in*.

Running the *contacts_1D_ohmic_charge_neutral_GaAs_nnp.in* and *contacts_1D_schottky_barrier_GaAs_nnp.in* with the `$barrier=0.53` and `$barrier=0.00` one can obtain a comparison of band profiles as presented in the Figure 4.2.2.1, which shows the conduction band edge profile for n-type $GaAs$ in equilibrium with

- the Schottky barrier of 0.53 V, i.e. the conduction band edge is pinned 0.53 eV above the Fermi level set at 0 eV
- the Schottky barrier of 0.00 V
- no barrier within “ohmic” contact

at position of 10 nm. The contact regions in these simulations are defined in the range from 0 nm to 10 nm but no equations are solved inside this region as both Fermi levels and electric potential are already assumed there as boundary conditions.

Note that in equilibrium the Fermi level is constant and equal to 0 eV in the whole device. If the semiconductor is doped, the conduction and valence band edges are shifted with respect to this Fermi level, i.e. relative to 0 eV and are thus dependent on doping. This is a bulk property and independent of surface effects, like ohmic contacts or Schottky barrier height, see right end of the Figure 4.2.2.1. At the left boundary, however, the band profile is affected by the type of contact.

Note

A Schottky barrier of 0 V is not equivalent to an ohmic contact.

Both `contacts{ schottky{} }` and `contacts{ ohmic{} }` used in the input files poses Dirichlet boundary conditions for the Poisson and Current equations. Within the `contacts{ ohmic{} }`, the electrostatic potential is set to the value satisfying requirement of charge neutrality in the region of this contact, $\phi = 0$. The `contacts{ schottky{} }` in the input files sets this value by the Schottky barrier, ϕ_B, being the value of the conduction band edge at the boundary with respect to the Fermi level:

$$E_c - E_F = \mathrm{e}\, \phi_B$$

In this particular example, an artificial Schottky barrier of -0.0365 V would be an equivalent to results obtained using an `contacts{ ohmic{} }`, (i.e. flat band condition), but only for the same temperature and the same doping concentration.

The input file *contacts_2D_schottky_barrier_GaAs_nnp.in* shows how to obtain the same results within 2D simulation.

Last update: 16/07/2024

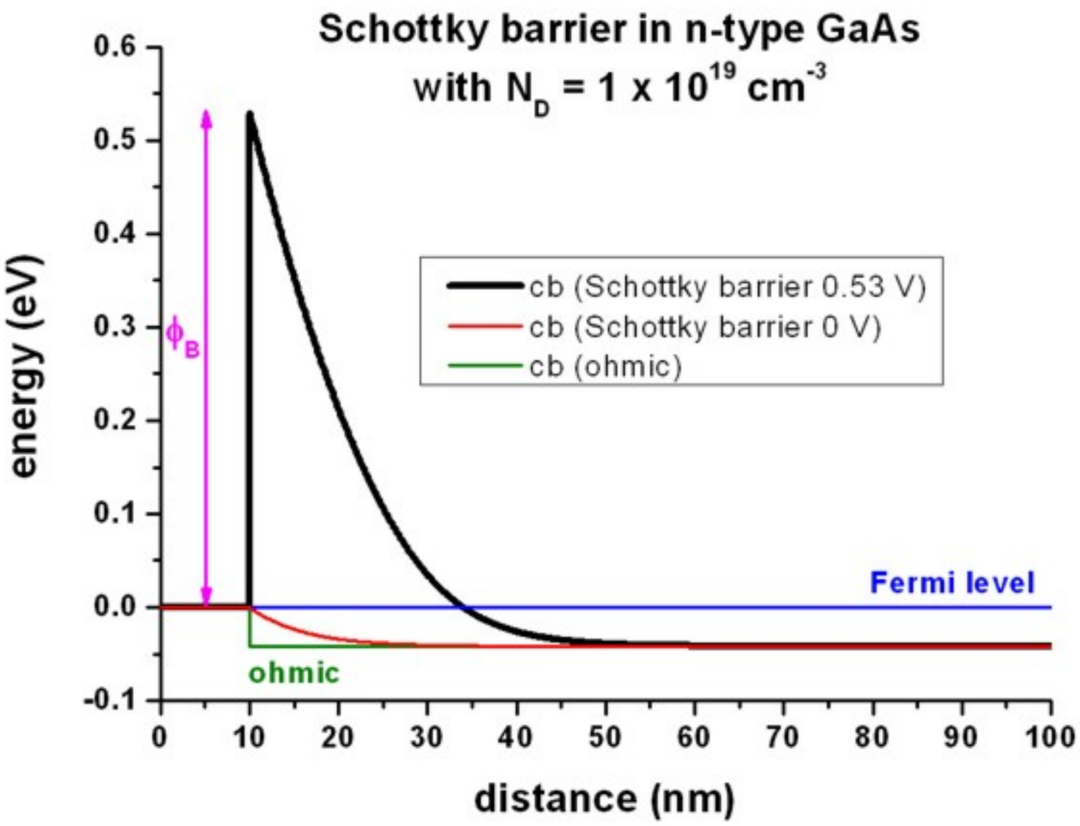


Figure 4.2.2.1: Calculated conduction band profile

— FREE — Surface Charges

- *Header*
- *Interface charges (surface states)*
- *Surface states - Acceptors*

Header

Files for the tutorial located in *nextnano++\examples\basics*

- *contacts_1D_zero_field_surface_charges_GaAs_nnp.in*
- *contacts_1D_zero_field_surface_acceptors_GaAs_nnp.in*

Scope:

Surface charges on boundaries - comparison to the Schottky barrier

Interface charges (surface states)

Instead of specifying a Schottky barrier, the user can alternatively specify a fixed surface charge density as presented in *contacts_1D_zero_field_surface_charges_GaAs_nnp.in*. The use of charges is similar as of dopants. One needs to define them for a specific region

```
structure{
    ...
```

(continues on next page)

(continued from previous page)

```
    region{ # interface charges (surface states)
        line{ x = [10 , 10 + $Width] }
        doping{
        constant{
            name = "negative-interface-charge"   # name of impurity
            conc = $VolumeDensity                # doping concentration [cm-3]
        }
    }
}
```

and define with some name and given sign.

```
impurities{
    ...
    charge{
        name = "negative-interface-charge"          # refer to region with name␣
↪negative-interface-charge
        type = negative
    }
}
```

Figure 4.2.2.2 shows that the red curve (= "ohmic" contact with interface charge density σ (surface states) of -8.4796 · 10^{12} $|e|$ /cm^2 = -1.3586 · 10^{-2} C/m^2) is equivalent to the black curve (Schottky barrier of 0.53 eV).

A sheet charge density of -8.4796 · 10^{12} cm^{-2} corresponds to a volume charge of -8.4796 · 10^{20} cm^{-3} if one assumes this charge to be distributed over a grid spacing of 0.1 nm. In this case, the interface charge density corresponds to a Neumann boundary condition for the derivative of the electrostatic potential ϕ:

$$\frac{d\phi}{dx} = -E_x = \text{const},$$

where E_x is the electric field component along the x direction. E_x is related to the interface charge as follows:

$$E_x = \frac{\sigma}{\epsilon_r \epsilon_0}$$

where ϵ_0 is the permittivity of vacuum and ϵ_r is the dielectric constant of the semiconductor. In this example:

- ϵ_r = 12.93 for GaAs
- E_x = 1049.7 kV/cm

The output for the electric field (in units of [kV/cm]) can be found in this file: *electric_field.dat*

The output for the interface densities can be found in this file: *material\density_fixed_charge.dat*.

Surface states - Acceptors

Input file: *1DSchottky_barrier_GaAs_surface_states_acceptor_nnp.in*

Instead of specifying a Schottky barrier, the user can alternatively specify a density of acceptor surface states (p-type doping). Essentially, this can be done by specifying a p-type doping region that is very thin, i.e. the doping is specified only on one grid point.

In this example, we use a doping area of 0.1 nm at the surface that we dope p-type with a volume density of 847.96 · 10^{18} cm^{-3}. This corresponds to a sheet charge density of 8.4796 · 10^{12} cm^{-2} where we assume the states to have realistic *activation energies*.

```
impurities{
    ...
    acceptor{ # p-type
        name = "impurity_p"
```

(continues on next page)

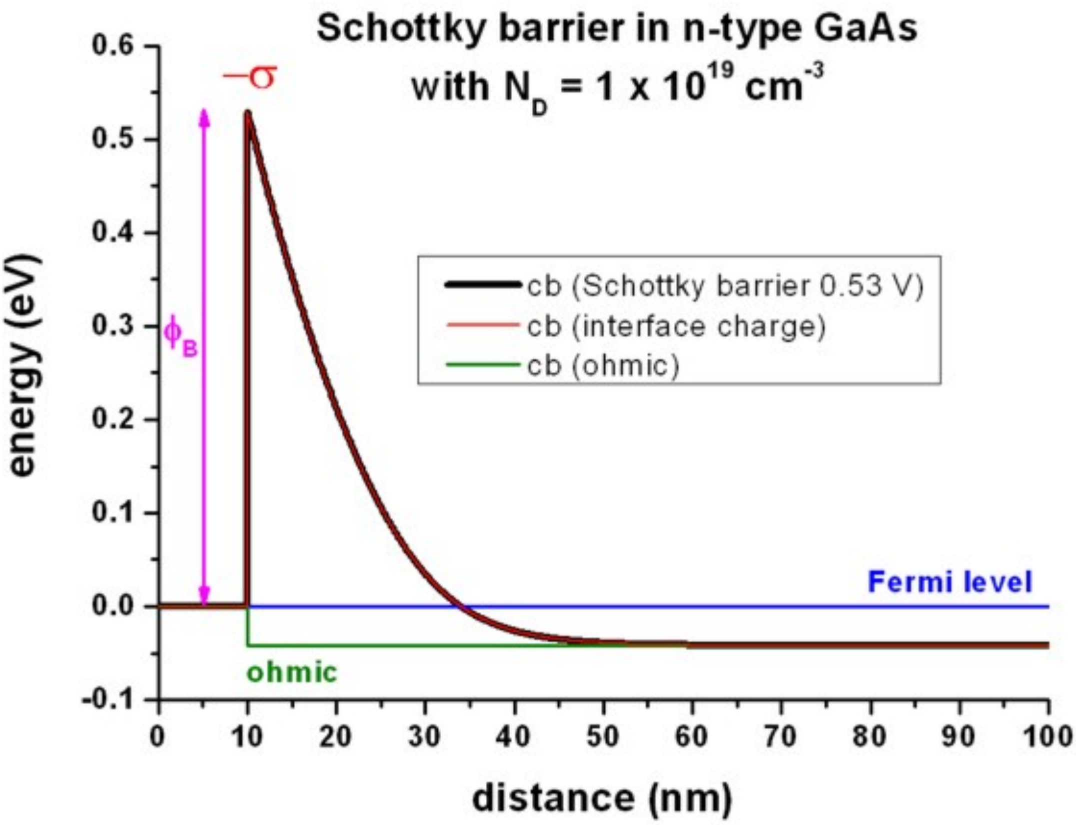


Figure 4.2.2.2: Calculated conduction band profile

(continued from previous page)

```
        energy = 0.027 # p-C-in-GaAs (Landolt-Boernstein 1982)
        degeneracy = 4 # degeneracy of energy levels, 2 for n-type, 4 for p-type
    }
}
```

The results are the same as shown in Figure 4.2.2.2 for the interface charges.

Last update: 16/07/2024

4.2.3 Electrostatics and Strain

— DEV — Solution of the Poisson equation for different charge density profiles

Input Files:

- *1D_Poisson_dipole_nnpp.in*
- *1DPoisson_linear_nnp.in*
- *1D_Poisson_delta_nnpp.in*

> **Note**
>
> If you want to obtain the input files that are used within this tutorial, please check if you can find them in the installation directory. If you cannot find them, please submit a Support Ticket.

Scope:
In this tutorial we show solution of Poisson equation for constant, linear and delta-function like charge density profile of positive and negative charges.

Output files:

- *bias_00000\density_electron.dat*, *bias_00000\density_hole.dat*
- *bias_00000\electric_field.dat*
- *bias_00000\potential.dat*

1) Dipole: Constant charge density profile of positive and negative charge

Input file: *1D_Poisson_dipole_nnpp.in*

The following figures (Figure 4.2.3.1 and Figure 4.2.3.2) show a dipole charge density distribution where

- the left region (from x = 0 nm to x = 10 nm) carries a constant positive charge density (resulting from ionized donors N_D^+) and
- the right region (from x = 10 nm to x = 20 nm) carries a constant negative charge density (resulting from ionized acceptors N_A^-).

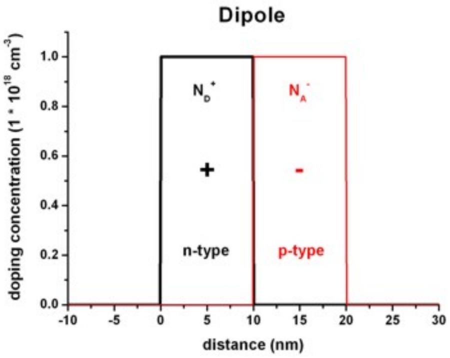


Figure 4.2.3.1: Doping distribution

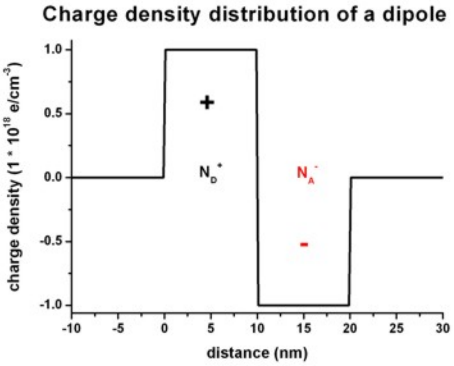


Figure 4.2.3.2: Charge density distribution

We have to solve the Poisson equation:

$$\frac{d^2\phi}{dx^2} = -\frac{\rho}{\epsilon_r\epsilon_0}$$

Figure 4.2.3.3 shows the corresponding electric field distribution and Figure 4.2.3.4 shows the electrostatic potential profile

The electric field is given by

$$E(x) = -\frac{d\phi}{dx}$$

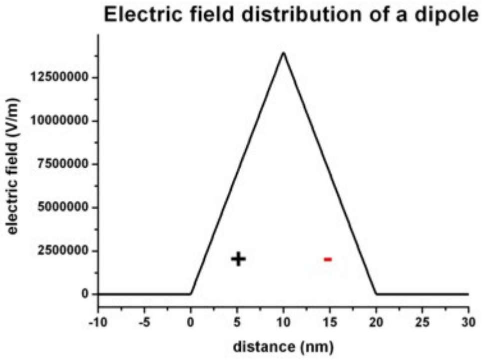


Figure 4.2.3.3: Electric field distribution

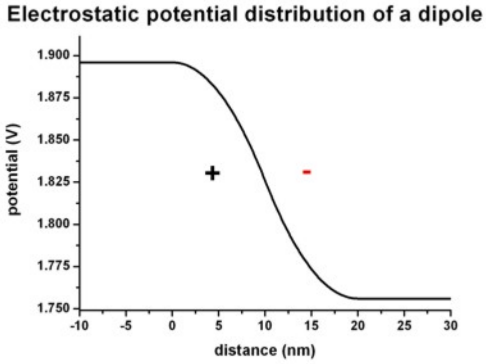


Figure 4.2.3.4: Electrostatic potential distribution

and has a linear dependence (~ -x) because the electrostatic potential has a quadratic dependence (~ x^2). The maximum value of the electric field is given by:

$$E_{\max} = \frac{\rho}{\epsilon_r \epsilon_0} \cdot x_0 = \frac{e \cdot 1 \cdot 10^{18}\mathrm{cm}^{-3}}{12.93 \cdot 8.8542 \cdot 10^{-12}\mathrm{As/Vm}} \cdot 10\mathrm{nm} = 139.95\mathrm{kV/cm}$$

where x_0 is the width of the positive (or negative) charge density region, and ϵ_r = 12.93 is the static dielectric constant of $GaAs$.

The drop of the electrostatic potential between 0 nm and 20 nm is simply given by the area that is below the graph of the electric field:

$$\Delta\phi = \frac{1}{2} E_{\max} \cdot 20\mathrm{nm} = 139.95\mathrm{mV}$$

2) Linear charge density profile of positive and negative charge

Input file: *1D_Poisson_linear_nnpp.in*

The following figures (Figure 4.2.3.5 and Figure 4.2.3.6) show a linearly varying charge density distribution where

- the left region (from x = 0 nm to x = 10 nm) carries a linearly decreasing positive charge density (resulting from ionized donors N_D^+) and
- the right region (from x = 10 nm to x = 20 nm) carries a linearly increasing negative charge density (resulting from ionized acceptors N_A^-).

Figure 4.2.3.7 shows the corresponding electric field distribution and Figure 4.2.3.8 shows the electrostatic potential profile

The electric field shows a quadratic dependence (~ $-x^2$) whereas the electrostatic potential shows a cubic dependence (~ x^3).

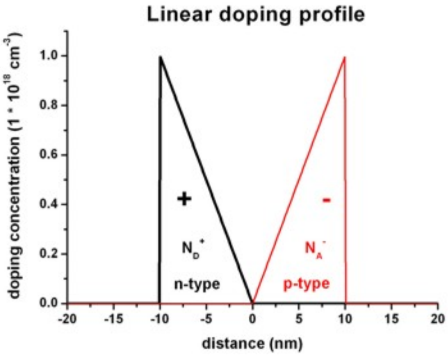


Figure 4.2.3.5: Doping profile

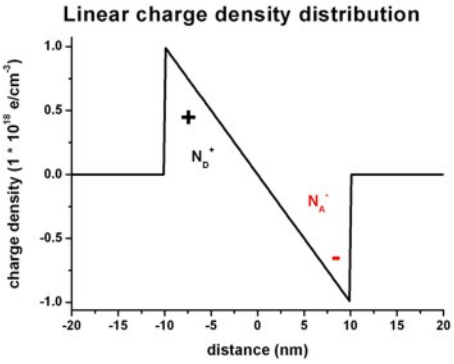


Figure 4.2.3.6: Charge density distribution

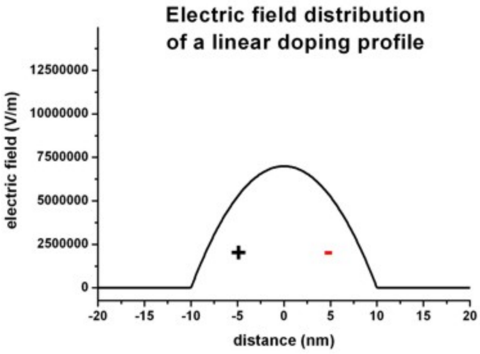


Figure 4.2.3.7: Electric field distribution

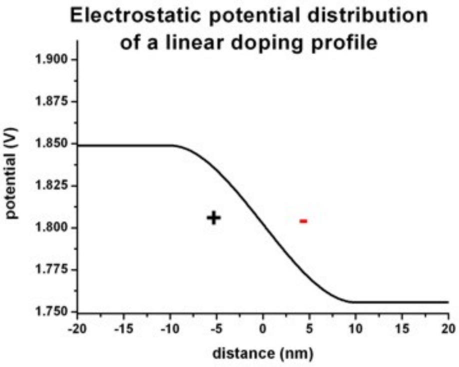


Figure 4.2.3.8: Electrostatic potential

3) Delta-function like charge density profile of positive and negative charges

Input file: *1D_Poisson_delta_nnpp.in*

The following figures (Figure 4.2.3.9 and Figure 4.2.3.10) show a delta-function like charge density distribution where

- in the middle of the structure (x = 0 nm) there is a constant positive charge density of width 1 nm (resulting from ionized donors N_D^+) and
- at the boundaries of the structure there are constant negative charge densities of width 1 nm each (resulting from ionized acceptors N_A^-).

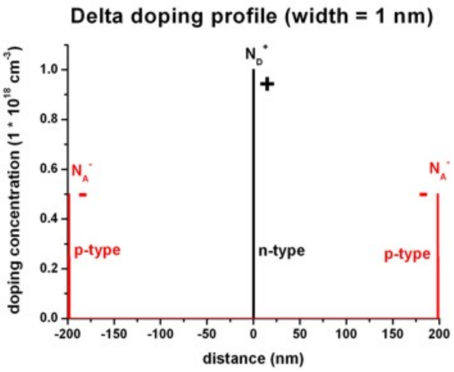


Figure 4.2.3.9: Doping profile

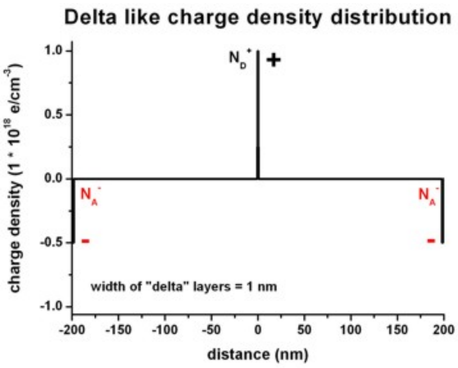


Figure 4.2.3.10: Charge density distribution

Figure 4.2.3.11 shows the corresponding electric field distribution and Figure 4.2.3.12 shows the electrostatic potential profile

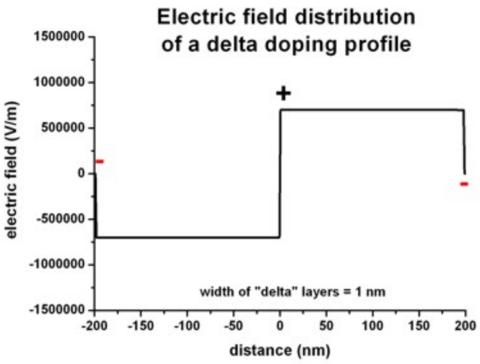


Figure 4.2.3.11: Electric field distribution

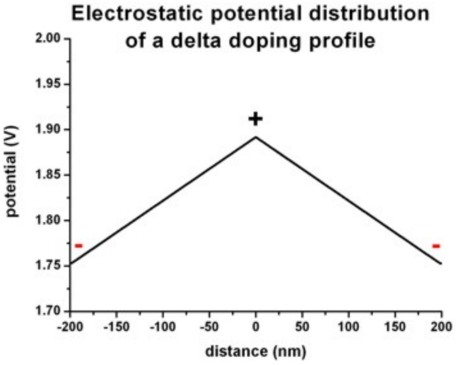


Figure 4.2.3.12: Electrostatic potential

Last update: nnnn/nn/nn

Band gap of strained AlGaInP on GaAs substrate

Input Files:

- *AlGaInP_on_GaAs_1D_nnp.in*

Scope:

In this tutorial we study the band gaps of strained $Al_xGa_yIn_{1-x-y}P$ on a $GaAs$ substrate. The material parameters are taken from *[VurgaftmanJAP2001]*.

Output Files:

- *strain\strain_simualtion.dat*
- *strain\hydrostatic_strain.dat*
- *bias_00000\bandedges.dat*

Strain

To understand the effect of strain on the band gap on the individual components of the quaternary $Al_xGa_yIn_{1-x-y}P$, we first examine the effects on

1) AlP strained tensely with respect to $GaAs$
2) GaP strained tensely with respect to $GaAs$
3) InP strained compressively with respect to $GaAs$
4) $Al_xGa_{1-x}P$ strained tensely with respect to $GaAs$
5) $Ga_xIn_{1-x}P$ strained with respect to $GaAs$
6) $Al_xIn_{1-x}P$ strained with respect to $GaAs$
7) $Al_{0.4}Ga_{0.6}P$ strained tensely with respect to $GaAs$
8) $Ga_{0.4}In_{0.6}P$ strained compressively with respect to $GaAs$
9) $Al_{0.4}In_{0.6}P$ strained compressively with respect to $GaAs$

Each material layer has a length of 10 nm in the simulation. The material layers 4), 5) and 6) vary their alloy contents linearly, i.e.

4) $Al_xGa_{1-x}P$: x = 0.0 to x = 1.0 (from 10 nm to 20 nm)
5) $Ga_xIn_{1-x}P$: x = 0.0 to x = 1.0 (from 30 nm to 40 nm)

6) $Al_xIn_{1-x}P$: x = 1.0 to x = 0.0 (from 50 nm to 60 nm)

There is no external stress applied to the structure, so Poisson's ratio holds. All layers are strained pseudomorphically with respect to a $GaAs$ substrate (i.e. the layers are biaxially strained in the plane perpendicular to the growth direction to match the lattice constant of $GaAs$).

The biaxial strain in the layers can be calculated with this formula:

$$e_{yy} = e_{zz} = \frac{a_{\text{substrate}} - a}{a}$$

where a is the lattice constant. The output of the strain tensor can be found in this file: *strain\strain_simualtion.dat*

The hydrostatic strain is the trace of the strain tensor and corresponds to the volume deformation:

$$e_{\text{hydro}} = Tr(e_{ij}) = e_{xx} + e_{yy} + e_{zz}$$

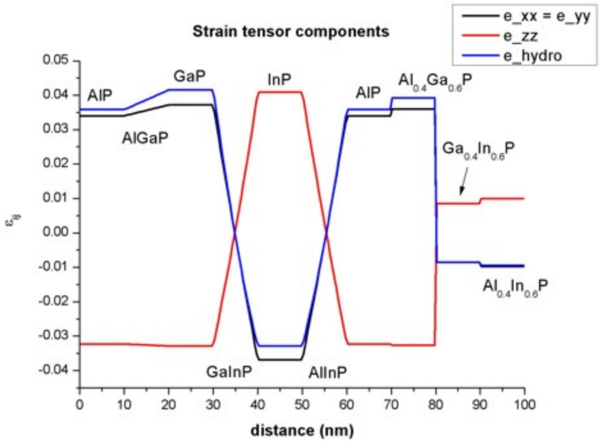


Figure 4.2.3.13: Strain tensor components

band gaps

Figure 4.2.3.14 shows the conduction band edges at the Gamma, L and X points and the heavy hole, light hole and split-off hole valence bands. The red line shows that band gap, i.e. the difference between the lowest conduction band minimum and the valence band maximum. The band gap maximum occurs at $Al_{0.55}In_{0.45}P$ (2.355 eV).

The conduction and valence band edges have been obtained taking into account the shifts and splittings of the bands due to strain and deformation potentials.

Note that conduction and valence band offsets are not taken into account in this plot. The zero of energy was taken to be the unstrained heavy hole / light hole band edge.

Due to strain, the degeneracy of the heavy and light hole is lifted. Also, the X band splits into two X bands (2-fold and 4-fold degeneracy).

In the case of tensile (compressive) strain, the light (heavy) hole band is the valence band maximum.

Note that the material parameters include band gap bowing.

Figure 4.2.3.15 compares the overall band gap to the case where band gap bowing has been neglected.

The *nextnano++* tool supports quaternaries:

```
quaternary_constant{
    name = "Al(x)Ga(y)In(1-x-y)P"
    alloy_x = 0.255
    alloy_y = 0.255
}
```

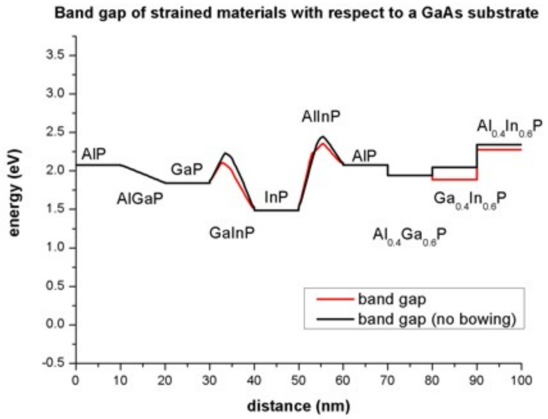


Figure 4.2.3.14: Band edge and band gap profile

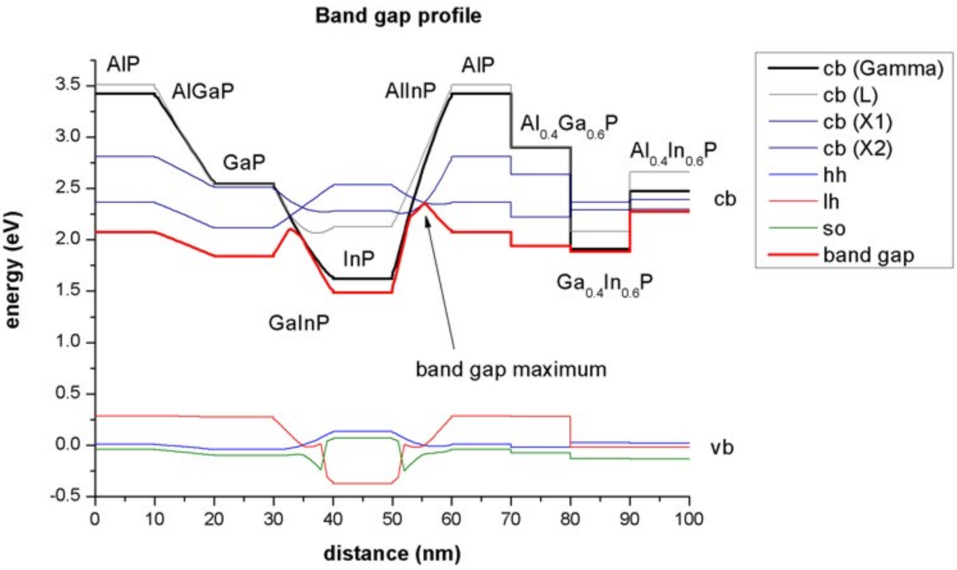


Figure 4.2.3.15: Ban dgap profile

Appendix E of the PhD thesis of T. Zibold (*[ZiboldPhD2007]*) is related to the *nextnano++* implementation of quaternaries.

Last update: nnnn/nn/nn

— NEW/EDU — Piezo- and Pyroelectric charges in GaN/AlN/GaN wurtzite heterostructure

Header

Files for the tutorial located in *nextnano++\examples\education*

- *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_offsets.in*
- *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain.in*
- *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_pyro.in*
- *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-pyro.in*
- *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-piezo.in*
- *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-piezo-pyro.in*
- *piezo-pyro-charges_wz_GaN-AlGaN-AlN_1D_nnp_strain-piezo-pyro.in* - additional example with polarization doping

Scope of the tutorial:

- defining wurtzite heterostructure
- piezo- and pyroelectricity in wurtzite

Main adjustable parameters in the input file:

- parameter `$Strain`
- parameter `$Polarity`

Relevant output files:

- *bias_00000\bandedges.dat*
- *bias_00000\potential.dat*

Introduction

This tutorial presents how to define wurtzite heterostructure and explains how piezo- and pyroelectric polarization constants influence respective charges on interfaces on a n example of GaN/AlAn/GaN heterostructure bringing insight into piezoelectricity and pyroelectricity in wurtzite. More detailed explanation of piezoelectricity in wurtzite can be also found in *Piezoelectricity in wurtzite*.

Crystallographic orientation

Input files for this tutorial simulate a GaN/AlN/GaN wurtzite structure grown pseudomorphically on GaN, i.e., the AlN is tensely strained, whereas the GaN is unstrained. The growth direction [0001] is set along which corresponds to the growth on Ga-polar GaN (0001) surface (Ga-face polarity).

As the wurtzite structure belongs to the hexagonal crystal system, one should take additional care about defining Miller indices of the growth plane.

```
global{ }
    simulate1D{}

    ## This is along [0001] direction: Ga-face polarity
    crystal_wz{
        x_hkl = [ 0, 0, 1 ]   # hkil = (0, 0,  0, 1) Miller-Bravais indices
        y_hkl = [ 1, 0, 0 ]   # hkil = (1, 0, -1, 0) Miller-Bravais indices

    substrate{
        name = "GaN"
    }
}
```

Although the four-digit Miller-Bravais indices ($hkil$) are usually used in a wurtzite structure, you have to omit i in *nextnano++* because $i = h - k$ holds. `x_hkl` refers to a plane and perpendicular to the crystal growing direction. See *Crystal coordinate systems* for more details. As the wurtzite structure lacks symmetry plane perpendicular to the c-axis, the c-plane is polarized. The 0001 plane in GaN is the *Ga-polar* plane, while the opposite $000\overline{1}$ plane is the *N-polar* plane. All the examples in this tutorials are prepared for the growth on the Ga-polar plane. The N-polar polarity is discussed at the end.

Strain-induced energy shift

Energy profiles without the strain effects

Figure 4.2.3.16 shows the energy band offsets (conduction and valence band edges) of the heterostructure. It is done by neglecting all polarization and strain effects. Poisson equation is solved to bring the offsets already near the Fermi level set to zero. Clearly AlN forms the barrier for both electrons and holes.

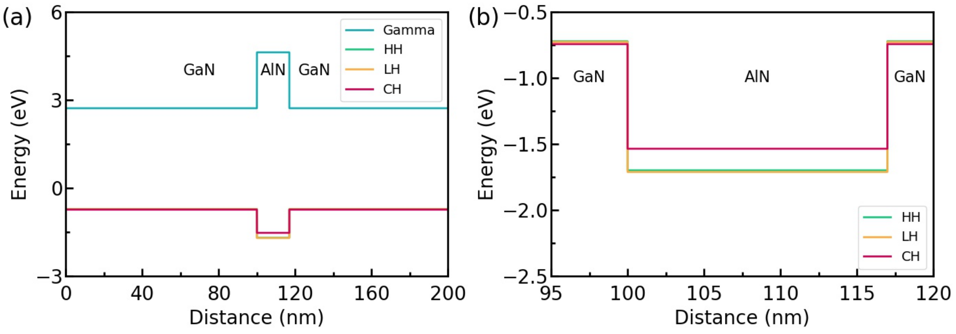


Figure 4.2.3.16: Calculated conduction and valence band structures without strain effects. (a) Full energy profile. (b) Valence band edges of AlN. (Run *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_offsets.in* to reproduce.)

It is visible that without strain the CH (crystal hole) band lies above the HH (heavy hole) and LH (light hole) bands in AlN while the situation is different for GaN, where HH band has the highest energy. This mechanism is explained in *[Chuang1996]*. Note that heavy and light hole are not degenerate under no-strain condition, unlike in zincblende crystals.

Including energy shift due to pseudomorphic strain

As the substrate in the simulation is set to GaN, the GaN remains unstrained also when the strain model is turned on. Since AlN has the lattice constant, $a_{\text{AlN}} = 0.3112$ nm, smaller than the one of GaN, $a_{\text{GaN}} = 0.3189$ nm, it becomes strained as follows.

The biaxial (in-plane) strain is tensile.

$$\varepsilon_{||} = (a_{\text{substrate}} - a)/a = 0.0247429$$

The uniaxial (growth direction) strain is compressive.

$$\varepsilon_{\perp} = -2(c_{13}/c_{33})\varepsilon_{||} = -0.0143283$$

The hydrostatic strain is positive, which corresponds to an increase in volume for AlN.

$$\varepsilon_{\text{hy}} = Tr(\varepsilon_{\text{ij}}) = (2\varepsilon_{||} + \varepsilon_{\perp}) = 0.0351575$$

Introduction of the strain leads to an energy shift of both conduction and valence band edges.

The crystal anisotropy leads to two distinct conduction band deformation potentials for the Γ point in wurtzite. The one is parallel, `defpot_absolute_l`, and the other one is perpendicular, `defpot_absolute_t`, to the c axis. These values are taken from the *database_nnp.in*.

```
binary_wz{
    name = AlN

    ...

    conduction_bands{
        Gamma{
            defpot_absolute_l = -20.5       # Vurgaftman2 (a1) along c axis
            defpot_absolute_t = -3.9        # Vurgaftman2 (a2) perpendicular to c axis
        }
    }
}
```

Denoting `defpot_absolute_l` as $a_{\text{c,caxis}}$ and `defpot-absolute_t` as $a_{\text{c,aaxis}}$, the conduction band minimum energy including the hydrostatic shift is given by

$$\begin{aligned} E_{\text{c}}' &= E_{\text{c}} + a_{\text{c,caxis}}\varepsilon_{\perp} + 2a_{\text{c,aaxis}}\varepsilon_{||} \\ &= 4.712 + (-20.5 \times (-0.0143283)) + 2(-3.9) \times 0.0247429 \\ &= 4.712 + 0.10073553 \\ &= 4.81274 \text{ eV} \end{aligned}$$

Therefore, the barrier for electrons is increased in this particular example.

Note

Data for uniaxial deformation potentials of other minima than Γ are not available yet. The uniaxial deformation potential is zero for the conduction band at the Γ point.

There are six valence band deformation potentials (D_1, D_2, D_3, D_4, D_5, and D_6) which arise from a full treatment of the effect of strain on the six-band Hamiltonian. These values are also specified in *database_nnp.in*.

```
binary_wz{
    name = AlN

    ...

    valence_bands{
        defpotentials = [ -17.1, 7.9, 8.8, -3.9, -3.4, -3.4 ] # D_1, D_2, D_3, D_4, D_5, D_6, respectively, Vurgaftman2
    }
}
```

In contrast to zincblende, an absolute deformation potential for the valence band is not needed. The shifts of the valence bands are obtained by diagonalizing the Bir-Pikus strain Hamiltonian, which is a general approach giving correct shifts for arbitrary crystallographic orientations. Note that this holds only for the valence bands.

In our example, the tensile strain in AlN shifts all holes upwards, - the heavy hole by 0.32847 eV, - the light hole by 0.32877 eV and - the crystal field split-off hole by 0.64726 eV, thus strongly reducing the barrier for holes.

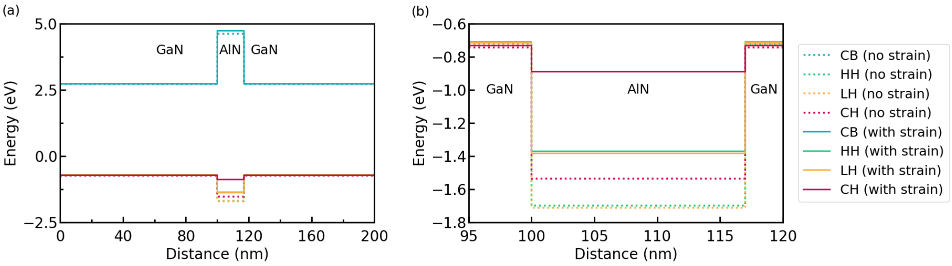


Figure 4.2.3.17: Calculated conduction and valence band structures with strain effects. (a) Full energy profile. (b) Valence band edges of AlN. (Run *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain.in* to reproduce.)

Polarization Effects

Polarization charges are simply computed basic formula from classical electrodynamics once proper Polarization fields are defined.

$$\nabla \circ \mathbf{P} = -\rho$$

Note that polarization effects are addittive, i.e., if $P = P_1 + P_2$ then

$$\nabla \circ \mathbf{P} = \nabla \circ [\mathbf{P}_1 + \mathbf{P}_2] = \nabla \circ \mathbf{P}_1 + \nabla \circ \mathbf{P}_2 = -\rho_1 - \rho_2$$

Pyroelectric polarization (spontaneous polarization)

The wurtzite material GaN, AlN, and InN are pyroelectric materials and thus show the pyroelectric polarization. The pyroelectric polarization field $\mathbf{P}_{\mathrm{py}}(\mathbf{x})$ is antiparallel to the c-axis, [0001], of the hexagonal unit cell (x-direction of exemplary simulations). Therefore, only non-zero component of the pyroelectric polarization vectors is parallel to the x-axis of the exemplary simulation: -0.034 C/m^2 for GaN and -0.090 C/m^2 for AlN.

Once the pyroelectric polarization is defined, the pyroelectric charge density can be computed as.

$$\rho_{\mathrm{py}}(\mathbf{x}) = -\nabla \circ \mathbf{P}_{\mathrm{py}}(\mathbf{x})$$

If the c-axis is oriented along the x-axis as in our example, this equation reduces to

$$\rho_{\mathrm{py}}(x) = -\frac{\partial}{\partial x} P_{\mathrm{py}}(x)\,.$$

As the derivative is non-zero only at the discontinuity of the polarization at the interfaces, all polarization charges will be located at these interfaces for this example. The surface densities of the polarization charges can be determined based on the Polarizations of GaN, $P_{\mathrm{py,x}}(\mathrm{GaN})$, and AlN, $P_{\mathrm{py,x}}(\mathrm{AlN})$, as follows:

The 1$^{\text{st}}$ interface (GaN/AlN) at 100 nm:

$$-\left[P_{\text{py,x}}\left(\text{AlN}\right)-P_{\text{py,x}}\left(\text{GaN}\right)\right]=P_{\text{py,x}}\left(\text{GaN}\right)-P_{\text{py,x}}\left(\text{AlN}\right)=-0.034+0.090=0.056\ \text{C}/\text{m}^{2}$$

2$^{\text{nd}}$ interface (AlN/GaN) at 117 nm:

$$-\left[P_{\text{py,x}}\left(\text{GaN}\right)-P_{\text{py,x}}\left(\text{AlN}\right)\right]=P_{\text{py,x}}\left(\text{AlN}\right)-P_{\text{py,x}}\left(\text{GaN}\right)=-0.090+0.034=-0.056\ \text{C}/\text{m}^{2}$$

The interface charge of $-0.056\ \text{C}/\text{m}^2$ corresponds to $34.952\times10^{12}\ \text{electrons}/\text{cm}^2$.

Piezoelectric polarization

Piezoelectric polarization appears due to presence of strain. In the exemplary simulation the AlN layer is strained, while GaN is not. Therefore, the piezoelectric polarization is non-zero only in the AlN layer.

$$P_{\text{pz,x}}\left(\text{AlN}\right)=e33\,\varepsilon_{\perp}+e31\left[\varepsilon_{\|}+\varepsilon_{\|}\right]=1.79\times\left[-0.0143283\right]-0.50\times2\cdot0.0247429=-0.050390\ \text{C}/\text{m}^{2}$$

The piezoelectric constants are specified in *database_nnp.in*.

```
binary_wz{
    name = AlN

    piezoelectric_consts{
        e31 = -0.50   e33 = 1.79   # Vurgaftman1 (Vurgaftman2 lists d_ij (/= e_ij !)␣
↪parameters.)
        e15 = -0.48                # [Tsubouchi1985] (experiment) and [Momida2016] and␣
↪O. Ambacher
    }
}
```

Note

The `e15` is not relevant for [0001] growth direction.

Similarly as for the pyroelectric polarization the piezoelectric charge density can be computed as

$$\rho_{\text{pz}}\left(\mathbf{x}\right)=-\nabla\circ\mathbf{P}_{\text{pz}}\left(\mathbf{x}\right)$$

and

$$\rho_{\text{pz}}\left(x\right)=-\frac{\partial}{\partial x}P_{\text{pz}}\left(x\right),$$

if the c-axis is oriented along the x-axis as in our example.

In this case as well, the derivative is non-zero only at the interfaces yielding the surface densities of the polarization charges based on the Polarizations of GaN, $P_{\text{pz,x}}\left(\text{GaN}\right)$, and AlN, $P_{\text{pz,x}}\left(\text{AlN}\right)$.

The 1$^{\text{st}}$ interface (GaN/AlN) at 100 nm:

$$-\left[P_{\text{pz,x}}\left(\text{AlN}\right)-P_{\text{pz,x}}\left(\text{GaN}\right)\right]=P_{\text{pz,x}}\left(\text{GaN}\right)-P_{\text{pz,x}}\left(\text{AlN}\right)=0+0.050390=0.050390\ \text{C}/\text{m}^{2}$$

2$^{\text{nd}}$ interface (AlN/GaN) at 117 nm:

$$-\left[P_{\text{pz,x}}\left(\text{GaN}\right)-P_{\text{pz,x}}\left(\text{AlN}\right)\right]=P_{\text{pz,x}}\left(\text{AlN}\right)-P_{\text{pz,x}}\left(\text{GaN}\right)=-0.050390-0=-0.050390\ \text{C}/\text{m}^{2}$$

The interface charge of $-0.050390\ \text{C}/\text{m}^2$ corresponds to $31.451\times10^{12}\ \text{electrons}/\text{cm}^2$.

Electrostatic potential of piezo- and pyroelectric charges

The electrostatic potential $\phi\left(\mathbf{r}\right)$ is the solution of the nonlinear Poisson equation.

$$\nabla \circ \left[\epsilon\left(\mathbf{r}\right)\nabla\phi\left(\mathbf{r}\right)\right] = -\rho\left(\mathbf{r},\phi\left(\mathbf{r}\right)\right)$$

The charge density ρ contains the (static) **piezo** and **pyroelectric** charge densities as well as the **electron** and **hole** charge densities and ionized **donors** and **acceptors**.

While the ionization of the impurities and free carriers depend on the electrostatic potential ϕ, the piezo- and pyroelectric charge densities do not.

The figure Figure 4.2.3.18 (a) shows electrostatic potential calculated for the heterostructure including:

1. both pyro- and piezoelectric charges (black)
2. only piezoelectric charges (turquoise)
3. only pyroelectric charges (purple)

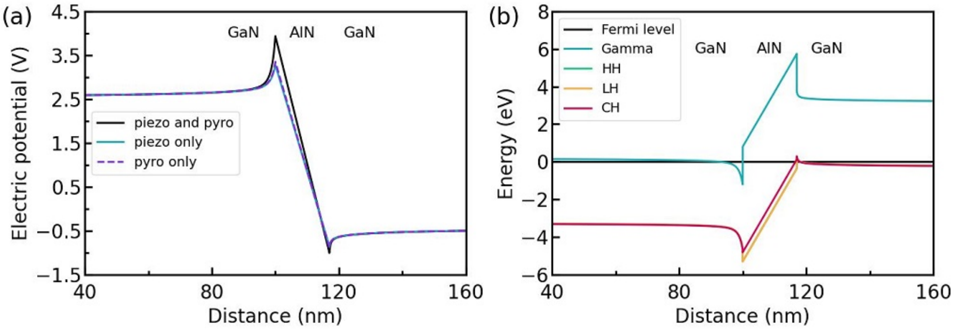


Figure 4.2.3.18: Electrostatic potential and energy profiles for Ga-face polarity. (a) The electrostatic potential with pyroelectric (py) and piezoelectric (pz) charges. (b) Conduction and valence band energy profiles under strain with all polarization charges included. (Run *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-pyro.in*, *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-piezo.in*, and *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-piezo-pyro.in* to reproduce.)

The pyro and piezoelectric contributions are comparable in this example. The band structure including the electrostatic potential is plotted in Figure 4.2.3.18 (b). Note that the conduction band is pulled below and the valence band above the Fermi level near the interfaces.

N-face polarity versus Ga-face polarity

The exactly same simulation of the GaN/AlN/GaN wurtzite structure can be performed also for the N-face polarity. The only difference from the previous simulations is implemented in the crystallographic orientation of the system.

Figure 4.2.3.19 shows again the electrostatic potential and the energy profiles, as before, but for both, Ga-face and N-face polarities.

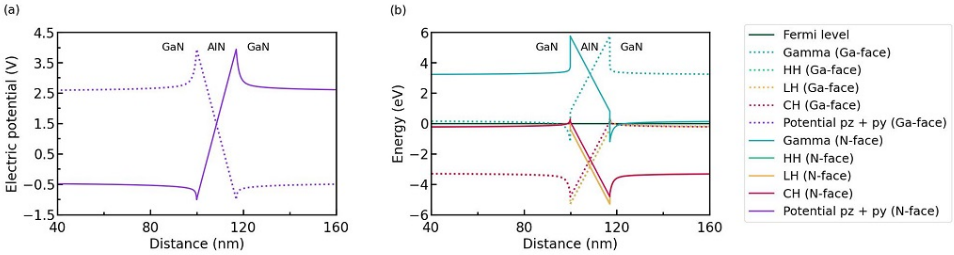


Figure 4.2.3.19: Electrostatic potential and energy profiles for Ga-face (dotted) and N-face polarities (solid). (a) The electrostatic potential with pyroelectric (py) and piezoelectric (pz) charges. (b) Conduction and valence band energy profiles under strain with all polarization charges included.

Note that the positions of 2D electron gas (2DEG) and 2D hole gas (2DHG) are reversed.

Exercises

1. Repeat all simulations for N-face polarity-
2. Explain why the built-in electric field is comparable in all simulations: *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-pyro.in*, *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-piezo.in*, and *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-piezo-pyro.in*

Last update: 07/08/2024

4.2.4 Currents

— EDU — Electron transport in n-type Silicon

Header

Files for the tutorial located in *nextnano++\examples\education*
1D_el_transport_Si_n_dop_nnp.in

Scope:

- mobility
- drift velocity
- mean free path
- scattering time
- resistance
- conductivity

Important output files:

- *bias_xxxxx/IV_characteristics.dat*
- *bias_xxxxx/velocity_electron.dat*
- *bias_xxxxx/mobility_electron.dat*

Problem

An n-type silicon layer of thickness $d = 1\ \mu\mathrm{m}$ is grown on a $1 \times 1\,\mathrm{cm}^2$ insulating substrate. It is doped with phosphorous (P) donors with a doping concentration of $N_\mathrm{D} = 1 \cdot 10^{16}\,\mathrm{cm}^{-3}$. Two ohmic contacts are located on the opposite sides of the sample, therefore, distanced by $l = 1\,\mathrm{cm}$ from each other.

Calculate:

a. mean drift velocity of charge carriers in the sample,
b. mean free path for the charge carriers in the sample by considering the effective scattering time and the mean drift velocity,
c. resistance and conductivity

at room temperature when 1 V of bias is applied to the contacts. Assume electron mobility $\mu_\mathrm{e} = 1222.58\,\mathrm{cm}^2/\mathrm{Vs}$ and hole mobility $\mu_\mathrm{h} = 425.54\,\mathrm{cm}^2/\mathrm{Vs}$.

Input file

The input file *1D_el_transport_Si_n_dop_nnp.in* contains a 1D definition of 1 cm long n-doped Si at 300 K as stated in the problem. Assumed mobilities of carriers in Si are overwritten in the group `database{ }`.

```
database{
    binary_zb{
        name = "Si"
        mobility_constant{
            electrons{ mumax = 1222.58 } # (cm2/Vs)
            holes{ mumax = 425.54} # (cm2/Vs)
        }
    }
}
```

The complete structure is n-doped with an impurity concentration of $N_\mathrm{D} = 10^{16}\ \mathrm{cm}^{-3}$. Activation energy of the dopants is taken from *this table*. Degeneracy is chosen 2 as typical for donors.

```
$doping_concentration = 1e16  # (cm^3)
$width = 1e7                  # (nm)


structure{
    region{ # Doping layer
        line{ x = [ -1.0, $width + 1.0 ] }
        doping{
            constant{
                name = "Phosphorus"
                conc = $doping_concentration
            }
        }
    }
}

impurities{
    donor{
        name = "Phosphorus"
        energy = 0.045      # (eV)
        degeneracy = 2
    }
}
```

The structure is biased with a voltage of 1 V and 0 V applied to the left and right contact, respectively.

```
contacts{ # this group is required in every input file
     ohmic{
         name = contact_right
         bias = 0.0 # (V)
     }
     ohmic{
         name = contact_left
         bias = 1.0 # (V)
     }
 }
```

The simulation of current inside the material is done based on the Drift-Diffusion model solved self-consistently with the Poisson equation. Therefore `poisson{ }`, `currents{ }`, and `run{ current_poisson{ } }` groups are present in the input file. *Constant mobility model* is chosen for this simulation. Among multiple interesting outputs, the ones useful for solving the problem are also added: electron velocity, mobility and currents.

```
$mobility_model = constant
currents{
    mobility_model = $mobility_model
    recombination_model{}

    output_mobilities{}
    output_currents{ }
    output_velocities{}
}
```

These can be found in output files: *IV_characteristics.dat*, *velocity_electron.dat*, and *mobility_electron.dat*. Computed values are used later in the tutorial to determine the scattering time, mean free path and resistance of the material.

Note

Scattering time of bulk crystal, mean free path and resistance cannot be outputted by *nextnano*++.

Solutions

Mean drift velocity

The mean drift velocity $v_{\mathrm{d,e}}$ of the electrons at an applied electric field $E = \frac{U}{d} = \frac{1\ \mathrm{V}}{1\ \mathrm{cm}} = 1\ \mathrm{V/cm}$ is given as follows:

$$v_{\mathrm{d,e}} = \mu \cdot E = \mu \cdot \frac{U}{d} = 1222.58\ \mathrm{cm}^2/\mathrm{Vs} \cdot \frac{1\mathrm{V}}{1\ \mathrm{cm}} = 1222.58\ \mathrm{cm/s} = 12.23\ \mathrm{m/s}$$

The drift velocities of electrons and holes at each grid point (in units of cm/s) can be found in the files *bias_XXXXX/velocity_electron* and *bias_XXXXX/velocity_hole*, respectively. From the simulation *1D_el_transport_Si_n_dop_nnp.in* one can read the drift velocity for electrons $v_{\mathrm{d,e}} = 1222.5797\,\mathrm{cm/s}$.

Mean free path

The mean free path can be calculated by the simple formula $l_{\mathrm{mfp}} = v_{\mathrm{d,e}} \cdot t_{\mathrm{eff,e}}$. We already determined the drift velocity $v_{\mathrm{d,e}}$. We only have to find the effective scattering time $t_{\mathrm{eff,e}}$. The effective scattering time of the electrons $t_{\mathrm{eff,e}}$ can be calculated as follows:

$$t_{\mathrm{eff,e}} = \mu \cdot \frac{m_{\mathrm{e,cond}}}{e} = 1222.58\ \mathrm{cm}^2/\mathrm{Vs} \cdot 0.258 \frac{m_0}{e} = 1.79 \cdot 10^{-13}\ s = 0.18\ \mathrm{ps}$$

where the conduction electron mass is given by

$$m_{\mathrm{e,cond}} = \frac{1}{1/0.916 + 2/0.19} m_0 = 0.258\ m_0.$$

Therefore, the mean free path for bulk Si is given by

$$l_{\mathrm{mfp}} = v_{\mathrm{d},e} \cdot t_{\mathrm{eff},e} = 0.0022 \text{ nm}.$$

Resistance and conductivity

The calculated current density j (in units of [A/cm^2] for a 1D simulation) can be found in the file: *bias_xxxxx/IV_characteristics.dat*. For an applied voltage of 1 V the calculated value reads

$$j = 19507 \text{ A/m}^2 = 1.9507 \text{ A/cm}^2.$$

Taking into account the dimensions of the Si sample ($A = 1 \text{ cm}^2$), this corresponds to a total current I of

$$I = 19507 \text{ A/m}^2 \cdot 1 \text{ cm} \cdot 1 \text{ }\mu\text{m} = 1.9507 \cdot 10^{-4} \text{ A} = 0.2 \text{ mA}.$$

The ohmic resistance is thus given by

$$R = \frac{U}{I} = \frac{1\,V}{1.9507 \cdot 10^{-4} \text{ A}} = 5105.2\ \Omega = 5.1 \text{ k}\Omega.$$

The conductivity σ is given by

$$\sigma = \frac{j}{E} = \mu_e\, n\, e = \frac{19507 \text{ A/m}^2}{1 \text{ V/cm}} = 195\ \Omega\text{m}.$$

and is related to the resistance as follows:

$$\sigma = \frac{j}{E} = \frac{I/A}{U/d} = \frac{1}{w\,R},$$

where w is the width of the sample. Here, $w = 1\ \mu$m.

Further Exercises

1. Repeat the calculations for InSb assuming electron mobility $\mu_{e,\mathrm{InSb}} = 4 \cdot 10^5 \text{ cm}^2/\text{Vs}$ and compare your findings with the results you have obtained for Si.
2. Repeat the calculations for Two-dimensional electron gases (2DEGs) in AlGaAs/GaAs heterostructures assuming electron mobility $\mu_{e,\mathrm{2DEG}} = 10^7 \text{ cm}^2/\text{Vs}$ and compare your findings with the results you have obtained for Si.

Hint

You can change the material to, e.g., InSb by altering the variable `$material`. Custom material parameters, which should not be taken from the default, should be specified in the group `database{ }`.

Answers

Drift velocity

- Electrons in InSb in a field of 1 V/cm have mean drift velocities of 4 · 10^5 cm/s = 4 km/s.
- Two-dimensional electron gases (2DEGs) in a field of 1 V/cm in AlGaAs/GaAs heterostructures have mean drift velocities of the order ~100 km/s.

Scattering time

- An effective scattering time for electrons in InSb (m_e = 0.0135 · m_0) is 3.1 ps.
- An effective scattering time for two-dimensional electron gases (2DEGs) in AlGaAs/GaAs heterostructures (m_e = 0.2 m_0) is of the order 1.1 ns.

Mean free path

- $l_{\mathrm{mfp}} = 12.4\,\mathrm{nm}$ for InSb.
- $l_{\mathrm{mfp}} = 110\,\mu\mathrm{m}$ for AlGaAs/GaAs (2DEG).

Last update: nnnn/nn/nn

— DEV — I–V characteristics of n-doped Si structure

Input files:

- *I-V_n-doped-Si_1D_nnp.in*
- *I-V_n-doped-Si_2D_nnp.in*
- *I-V_n-doped-Si_3D_nnp.in*
- *I-V_nin-doped-Si_1D_nnp.in*
- *I-V_nin-doped-Si_2D_nnp.in*
- *I-V_nin-doped-Si_3D_nnp.in*

Scope:

This tutorial aims to simulate the I-V characteristics of n-doped and n-i-n doped Si structures.

Output files:

- *IV_characteristics.dat*
- *bias_xxxxx/bandedges.dat*

I-V characteristics of an n-doped Si structure

Structure

The structure we are dealing with consists of bulk Si that is sandwiched between two contacts. The whole structure has the following dimensions (see also):

- along x-axis: $20\,\mathrm{nm}$ ($1\,\mathrm{nm}$ contact, $18\,\mathrm{nm}$ Si, $1\,\mathrm{nm}$ contact)
- along y-axis: $5\,\mathrm{nm}$

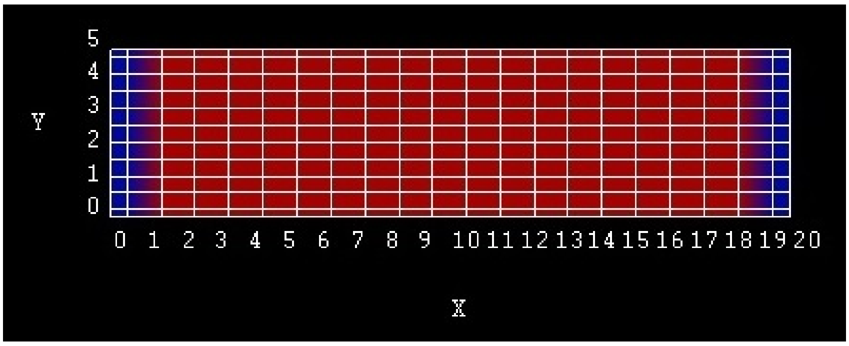


Figure 4.2.4.1: Simulated structure consisting of a left and right contact (blue) and n-doped Si layer (red).

The Si is n-type doped with a donor concentration of $N_{\mathrm{D}} = 1 \cdot 10^{20}\,\mathrm{cm}^{-3}$. The energy level is 0.044 eV below the conduction band edge. This leads to an electron density of $n = 13.48 \cdot 10^{18}\,\mathrm{cm}^{-3}$, which corresponds to the concentration of the ionized donors. The Fermi level E_{F} is taken to be at 0 eV in an equilibrium simulation, i.e. $V = 0\,\mathrm{V}$. The distance of the conduction band from the Fermi level can be calculated in the following way:

- For the effective electron mass at the Δ-point we have:

$$m_e = m^*_{e,\mathrm{DOS}} = (m_\mathrm{l} \cdot m_\mathrm{t} \cdot m_\mathrm{t})^{\frac{1}{3}} = (0.916 \cdot 0.19^2)^{\frac{1}{3}} m_0 = 0.321 m_0,$$

where m_l is the longitudinal and m_t is the transversal mass of the effective mass tensor.

- The effective density of states reads:

$$N_\mathrm{c} = 12 \cdot \left(\frac{2\pi m_e k_\mathrm{B} T}{h^2}\right)^{\frac{3}{2}} = 12 \cdot (0.321 \cdot 0.026 \cdot 2.0886 \cdot 10^{14})^{\frac{3}{2}} = 12 \cdot 2.282 \cdot 10^{18}\,\mathrm{cm}^{-3} = 2.738 \cdot 10^{19}\,\mathrm{cm}^{-3},$$

where the factor of 12 arises due to the six-fold degeneracy of Si at Δ and the two-fold spin degeneracy. Similarly, we obtain the effective density of states for holes:

$$N_\mathrm{v,\,hh} = 9.875 \cdot 10^{18}\,\mathrm{cm}^{-3},$$
$$N_\mathrm{v,\,lh} = 1.502 \cdot 10^{18}\,\mathrm{cm}^{-3}.$$

Note that heavy and light holes are degenerate for $k = 0$, i.e. $N_\mathrm{v} = N_\mathrm{v,\,hh} + N_\mathrm{v,\,lh} = 1.1377 \cdot 10^{19}\,\mathrm{cm}^{-3}$.

- The Semiconductor equation is given by

$$np = n_\mathrm{i}^2 = N_\mathrm{c} N_\mathrm{v} \exp\left(-\frac{E_\mathrm{gap}}{k_\mathrm{B} T}\right) = N_\mathrm{c} \cdot 1.138 \cdot 10^{19}\,\mathrm{cm}^{-3} \exp\left(-\frac{1.095}{0.026}\right) = 1.238 \cdot 10^{20}\,\mathrm{cm}^{-6},$$

with $E_\mathrm{gap} = 1.095\,\mathrm{eV}$, $n_\mathrm{i} = 1.113 \cdot 10^{10}\mathrm{cm}^{-3}$ and $p = n_\mathrm{i}^2/n = 9.185\,\mathrm{cm}^{-3}$.

- The occupation of the different energy states can either be described by Maxwell-Boltzmann statistics:

$$n(T) = N_\mathrm{c}(T) \exp\left(\frac{E_\mathrm{F} - E_\mathrm{c}}{k_\mathrm{B} T}\right),$$
$$p(T) = N_\mathrm{v}(T) \exp\left(\frac{E_\mathrm{v} - E_\mathrm{F}}{k_\mathrm{B} T}\right),$$

or Fermi-Dirac statistics:

$$n(T) = N_\mathrm{c}(T) \mathcal{F}_{1/2}\left(\frac{E_\mathrm{F} - E_\mathrm{c}}{k_\mathrm{B} T}\right),$$
$$p(T) = N_\mathrm{v}(T) \mathcal{F}_{1/2}\left(\frac{E_\mathrm{v} - E_\mathrm{F}}{k_\mathrm{B} T}\right),$$

where $\mathcal{F}_{1/2}$ is the Fermi-Dirac integral of order $1/2$ multiplied by the factor $2/\sqrt{\pi}$ (i.e. $\mathcal{F}_{1/2}$ includes the Gamma pre-factor)

When using the Maxwell-Boltzmann statistics as an approximation, we obtain:

$$E_\mathrm{c} = k_\mathrm{B} T \ln\left(\frac{N_\mathrm{c}}{n}\right) = 0.026\,\mathrm{eV} \cdot \ln\left(\frac{2.738 \cdot 10^{19}\,\mathrm{cm}^{-3}}{13.478 \cdot 10^{18}\,\mathrm{cm}^{-3}}\right) = 0.026\,\mathrm{eV} \cdot \ln(2.031) = 18.3\,\mathrm{meV},$$
$$E_\mathrm{v} = -k_\mathrm{B} T \ln\left(\frac{N_\mathrm{v}}{p}\right) = -0.026\,\mathrm{eV} \cdot 42.538 = -1.099\,\mathrm{eV}.$$

Note that *nextnano++* uses the Fermi-Dirac integrals (Fermi-Dirac statistics), where the following results are obtained: $E_\mathrm{c} = 13.85\,\mathrm{meV}$ and $E_\mathrm{v} = -1.0815\,\mathrm{eV}$.

Results

We sweep the voltage at the right contact from $0.0\,\mathrm{V}$ to $0.2\,\mathrm{V}$ in 10 steps. The input files used for the simulations are *I-V_n-doped-Si_1D_nnp.in*, *I-V_n-doped-Si_2D_nnp.in* *I-V_n-doped-Si_3D_nnp.in*. The calculated current density for each bias point can be found in *IV_characteristics.dat*. The resulting I-V characteristics is depicted in Figure 4.2.4.2.

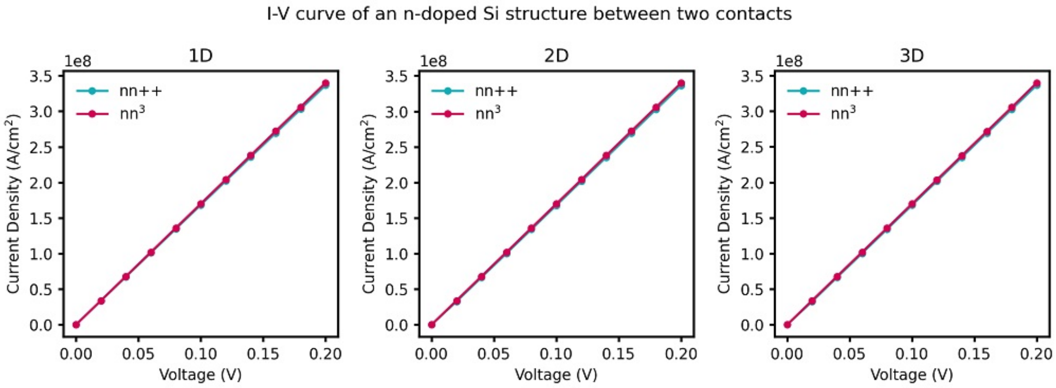


Figure 4.2.4.2: Simulated I-V characteristics of an n-doped Si structure using constant mobility model.

The units for the current in a 2D simulation are $[\mathrm{A/m}]$. Dividing this two-dimensional current value by the width of the device (in our case 5 nm) we obtain the current in units of $[\mathrm{A/cm^2}]$, which is the usual unit of a 1D simulation. As our simple 2D example structure is basically equivalent to a 1D structure we can easily compare our 2D results with the 1D results to check for consistency. It is also possible to perform a 3D simulation. In this case, the units for the three-dimensional current are $[\mathrm{A}]$. Dividing by the area of the device of $25\,\mathrm{nm}^2$, we obtain the 1D units of $[\mathrm{A/cm^2}]$.

I-V characteristics of an n-i-n-doped Si structure

Structure

The second example is an n-i-n (n-doped, intrinsic, n-doped) Si structure, which is shown in Figure 4.2.4.3. The width of the intrinsic region is 14 nm, and the n-doped regions are both 2 nm wide.

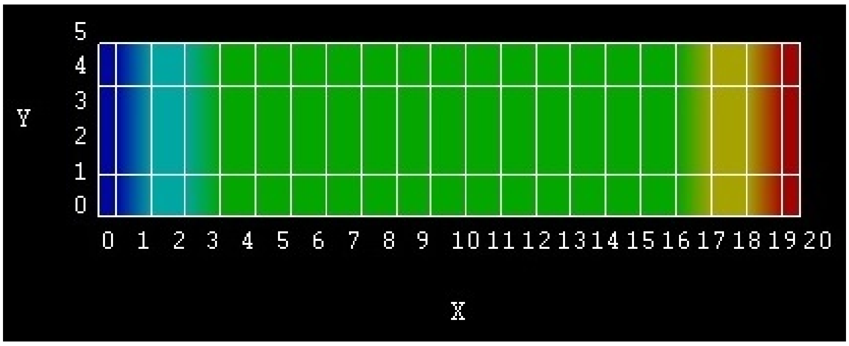


Figure 4.2.4.3: Simulated n-i-n structure consisting of a left contact (dark blue), n-doped Si (light blue), intrinsic Si (green), n-doped Si (yellow) and right contact (red).

Results

In Figure 4.2.4.4 the current-voltage (I-V) characteristic is shown. The input files used for the simulations are *I-V_nin-doped-Si_1D_nnp.in*, *I-V_nin-doped-Si_2D_nnp.in* *I-V_nin-doped-Si_3D_nnp.in*. The data of the I-V curve can be found in the corresponding file *IV_characteristics.dat*.

In order to compare the results from 1D, 2D and 3D simulations, we have divided the 2D current by the width of the device (in our case 5 nm) and the 3D current by the cross-section area of the device of (in our case $25\,\mathrm{nm}^2$), to get the current density in units of $[\mathrm{A/cm^2}]$. The obtained results are in perfect agreement.

Figure 4.2.4.5 shows the conduction band profile (*bias_xxxxx/bandedges.dat*) for different voltages.

This tutorial also exists for *nextnano³*.

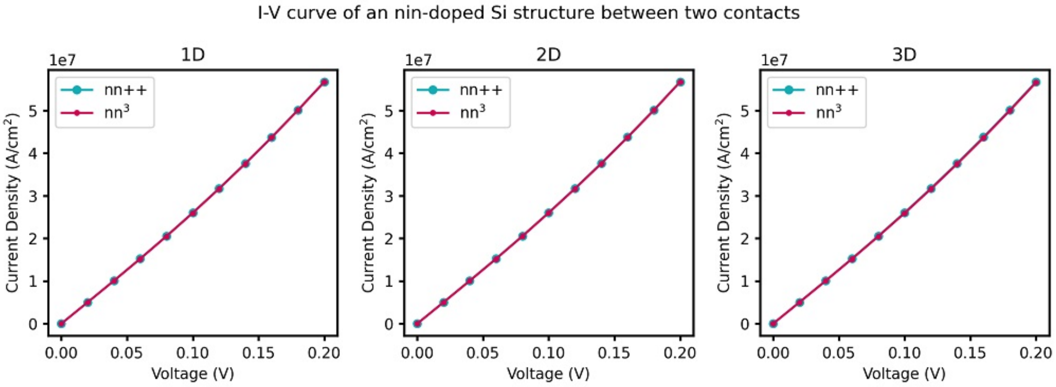


Figure 4.2.4.4: Simulated I-V characteristics of the n-i-n doped Si structure using constant mobility model.

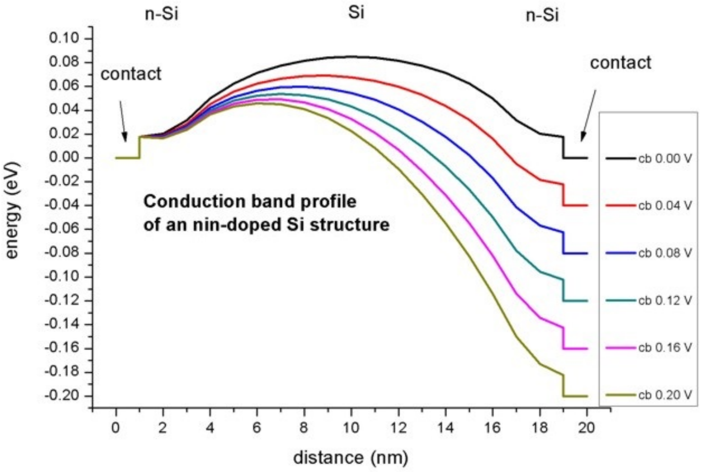


Figure 4.2.4.5: Simulated conduction band profile of the n-i-n Si structure for different voltages.

Last update: 17/07/2024

— DEV — I–V characteristics of n-doped GaN single layer

Header

Input Files:

- *IV_GaN_n_doped_1D_nnp.in*
- *IV_GaN_n_doped_2D_nnp.in*
- *IV_GaN_n_doped_3D_nnp.in*

Scope of the tutorial:

- currents
- wurtzite

Main adjustable parameters in the input file:

- parameter

Relevant output files:

- *IV_characteristics.dat*

Introduction

This tutorial shows the accuracy of drifft-diffusion model implemented in *nextnano*++ on a simple example: a single layer of an n-doped GaN. We compare the I–V characteristics obtained by *nextnano*++ with analytical solutions.

IV characteristics of an n-doped GaN single layer

The conductivity σ and the resistivity ρ of an n-type doped GaN sample can be calculated analytically, following formulas:

$$\sigma = q\mu_n n,$$
$$\rho = d/\sigma,$$

where q is electron charge, n is concentration of electron carriers, μ_n is mobility of electrons, and d is thickness of the material.

This is a good check for the results obtained with *nextnano*++ simulations. The thickness of the GaN layer is $d = 100$ nm.

The structure we are dealing with consists of bulk GaN that is sandwiched between two contacts. The whole structure has the following dimensions:

material	width (nm)	doping
contact	10	
n-GaN	100	$1 \times 10^{18}\ \mathrm{cm}^{-3}$
contact	10	

As you see, the GaN is n-type doped with a donor concentration of $N_D = 1 \times 10^{18}\ \mathrm{cm}^{-3}$. The energy level is chosen to be $0.01507\ \mathrm{eV}$ below the conduction band edge.

```
impurities{
    donor{ name = "Si_donor" degeneracy = 2 energy = 0.01507 }
}
```

This leads to the electron density of $5.2846 \times 10^{17}\ \mathrm{cm}^{-3}$. This is also equivalent to the concentration of the ionized donors. The result obtained by another commercial software is $5.355 \times 10^{17}\ \mathrm{cm}^{-3}$.

```
contacts{
    ohmic{ name = "left_contact" bias = 0.0 }
    ohmic{
        name = "right_contact"
        !WHEN $biassweep bias = [ $biasstart, $biasend ]
        !WHEN $biassweep steps = $biassteps
        !WHEN $nosweep   bias = $biasstart
    }
}
```

If `$biassweep = 1`, sweeping bias takes place. Otherwise, if `$biassweep = 0` and `$nosweep (= 1 - $biassweep) = 1`, sweeping bias is not applied. Since the bias is swept from 0.00 V to 0.10 V, `$biasstart` is set to `0.0` and `$biasend` is set to `0.1`. In addition, `$biassteps` is equal to `10`.

We take the GaN mobility to be constant: $\mu_n = 100\ \mathrm{cm}^2/\mathrm{Vs}$. The mobility model that is applied is called `constant` and described as below.

```
currents{
    mobility_model = constant
    recombination_model{
        SRH = no
        Auger = no
        radiative = no
    }
    output_currents{ }
}
```

We sweep the voltage at the right contact and calculate the current density for 0.00 V, 0.01 V, 0.02 V, …, 0.10 V (10 steps).

Results

1D

The current-voltage (IV) characteristic can be found in the following file: *IV_characteristics.dat*. Figure 4.2.4.6 shows the IV curve obtained by *nextnano++*.

The figure shows that the GaN layer is an ohmic resistor. From Figure 4.2.4.6, you can obtain a resistivity of the n-GaN layer of $1.1819 \times 10^{-6}\ \Omega\mathrm{cm}^2$. Another commercial software results in $1.43 \times 10^{-6}\ \Omega\mathrm{cm}^2$.

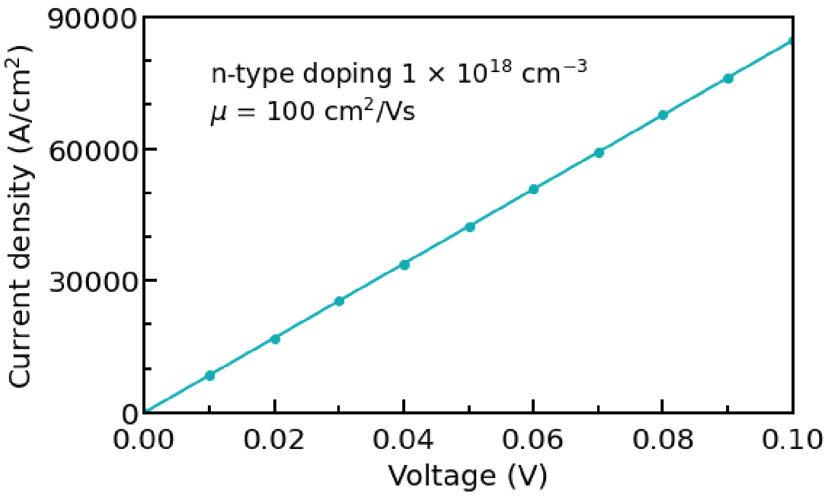


Figure 4.2.4.6: IV curve of an n-doped GaN single layer.

A good check is the analytic formula given above. From this, you can obtain:

$$\sigma_n = e\mu_n n = 1.6022 \times 10^{-19}\ \mathrm{As} \times 100\ \mathrm{cm}^2/\mathrm{Vs} \times 5.2846 \times 10^{17}\ \mathrm{cm}^{-3} = 8.4670\ \mathrm{A/Vcm}$$

$$\rho = d/\sigma = 100\ \mathrm{nm}/(8.46700\ \mathrm{A/Vcm}) = 1.1811 \times 10^{-6}\ \Omega\mathrm{cm}^2$$

Another analytical result with the other commercial software is $1.168 \times 10^{-6}\ \Omega\mathrm{cm}^2$.

Thus, you can see that the *nextnano++* result agrees better with the analytical result than the result by the other commercial software.

2D

Now, we try the same structure in a 2D *nextnano++* simulation to check if the 2D result agrees with the 1D one. The input file *IV_GaN_n_doped_2D_nnp.in* is used for this section. The width of the sample along the y direction is 200 nm. The x direction is the same as in 1D.

Note that the unit for the current in a 2D simulation is $[\mathrm{A/cm}]$. Dividing this two-dimensional current value by the width of the device (in our case 200 nm), we obtain the current density in units of $[\mathrm{A/cm}^2]$ which is the usual unit of a 1D simulation. As our simple 2D example structure is basically equivalent to a 1D structure, we can easily compare our 2D results with the 1D results to check for consistency.

voltage	current $(\mathrm{A/cm})$ (*nextnano++* 2D)	current density $(\mathrm{A/cm}^2)$ (*nextnano++* 2D*)	current density $(\mathrm{A/cm}^2)$ (*nextnano++* 1D)
0	0	0	0
0.02	0.33845	16922.4	16922.4
0.04	0.67689	33844.7	33844.7
0.06	1.0153	50767.0	50767.0
0.08	1.3538	67689.2	67689.2
0.10	1.6922	84611.2	84611.3

$*$ Here, the current density of the 2D simulation is obtained by dividing the current $[\mathrm{A/cm}]$ by the width 200 nm.

From the IV characteristics obtained from the 2D simulation, you can obtain a resistivity of the n-GaN layer of $1.1819 \times 10^{-6}\ \Omega\mathrm{cm}^2$ which agrees very well with the 1D result (1D: $1.1819 \times 10^{-6}\ \Omega\mathrm{cm}^2$).

3D

Of course, it is also possible to simulate this structure in 3D. In this case, the unit of the current is $[\mathrm{A}]$ and have to be divided by the area of the device perpendicular to the current flow direction to obtain the units of $[\mathrm{A}/\mathrm{cm}^2]$.

Last update: 17/07/2024

— DEV — n-i-n Si resistor

> **Attention**
>
> This tutorial is under construction

Input files:

- *nin-resistor_Si_Sabathil_JCE_2002_1D_nnp.in*

Scope:

This tutorial aims to simulate the current through n-i-n Si transistors. We illustrate our method for calculating the current by studying simple one-dimensional examples that we can compare to full Pauli master equation results. Our method is capable of calculating the electronic structure of a device fully quantum mechanically, yet employing a semi-classical scheme for the evaluation of the current. As we shall see, the results are close to those obtained by the full Pauli master equation provided we limit ourselves to situations not too far from equilibrium. The tutorial is based on the example presented on p. 43 in Stefan Hackenbuchner's PhD thesis *[HackenbuchnerPhD2002]* and on the following paper: *[Sabathil2002]*.

Output files:

- *bias_xxxxx\density_electron.dat*
- *bias_xxxxx\bandeges.dat*
- *IV_characteristics.dat*

Structure

We consider a one-dimensional 300 nm Si-based n-i-n resistor at room temperature where "n-i-n" stands for "n-doped / intrinsic / n-doped" (see Figure 4.2.4.7). The intrinsic region and the n-doped regions are each 100 nm wide. At both ends of the device there are ohmic contacts.

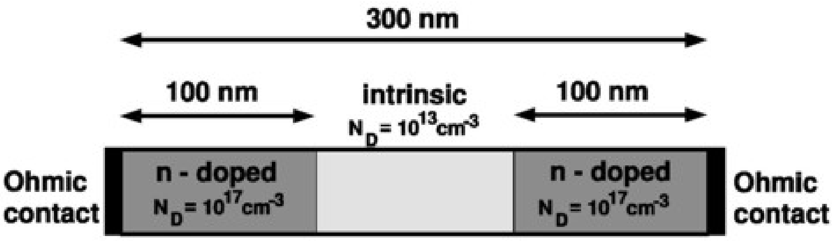


Figure 4.2.4.7: Geometry of the n-i-n Si resistor

The n-doped regions at the left and right sides are doped with a doping concentration of $N_\mathrm{D} = 1 \cdot 10^{17}\,\mathrm{cm}^{-3}$. The intrinsic region in the center of the device has a background concentration of $n_\mathrm{i} = 1 \cdot 10^{13}\,\mathrm{cm}^{-3}$ (see p. 43 in *[HackenbuchnerPhD2002]*). This value is calculated by *nextnano++* automatically and does not have to be entered in the input file. Assuming Maxwell-Boltzmann statistics, the intrinsic carrier concentration n_i is given by

$$n_\mathrm{i} = (N_\mathrm{c} N_\mathrm{v})^{\frac{1}{2}} \exp\left(-\frac{E_\mathrm{gap}}{2k_\mathrm{B}T}\right), \tag{4.2.4.1}$$

where $T = 300\,\mathrm{K}$ is the temperature, $E_{\mathrm{gap}} = 1.095\,\mathrm{eV}$ is the band gap energy of Si at $T = 300\,\mathrm{K}$, $N_{\mathrm{c}} = 2.738 \cdot 10^{19}\,\mathrm{cm}^{-3}$, $N_{\mathrm{v}} = 1.138 \cdot 10^{19}\,\mathrm{cm}^{-3}$. Using (4.2.4.1), one obtains $n_{\mathrm{i}} = 1.12 \cdot 10^{10}\,\mathrm{cm}^{-3}$. For a more detailed discussion of this equation (including Fermi-Dirac statistics), please read the description in Tutorial *I-V characteristics of an n-doped Si structure*.

The conductivity electron mass is given by

$$m^*_{\mathrm{e,cond}} = \frac{2}{1/0.916 + 2/0.19}\,\mathrm{m}_0 = 0.258 m_0,$$

whereas the DOS electron effective mass is given by

$$m^*_{\mathrm{e,DOS}} = (0.916 \cdot 0.19^2)^{\frac{1}{3}}\,m_0 = 0.321\,m_0.$$

The static dielectric constant is given by $\epsilon = 11.7$. For the donors we assumed an ionization energy of 0.015 eV and a degeneracy factor of 2.

Simulation

The electron density in *nextnano*++ can be calculated in two different ways:

- classical density (Thomas-Fermi approximation)
- quantum mechanical density (local quasi-Fermi levels).

The charge density is calculated for a given applied voltage by assuming the carriers to be in local equilibrium that is characterized by energy-band dependent local quasi-Fermi levels $E_{\mathrm{F}}(x)$ (i.e. in the simplest case, one for holes and one for electrons). These local quasi-Fermi levels are determined by global current conservation $\nabla \mathbf{j} = 0$, where the current is assumed to be given by the semi-classical relation $\mathbf{j} = \mu(x) n(x) \nabla E_{\mathrm{F}}(x)$, where $\mu(x)$ is the electron mobility determined by the chosen *mobility model*. The carrier wave functions and energies are calculated by solving the single-band Schrödinger-Poisson equation self-consistently. The Schrödinger, Poisson and current continuity equations are solved iteratively. As a preparatory step, the built-in potential is calculated for zero applied bias by solving the Schrödinger-Poisson equation self-consistently employing a predictor-corrector approach. The ohmic contacts impose the boundary conditions $E = 0\,\mathrm{kV/cm}$ on the electric field. For applied bias, the Fermi level and the potential at the contacts are then shifted according to the applied potential which fixes the boundary conditions. The main iteration scheme itself consists of two parts:

- In the first part, the wave functions and potential are kept fixed and the quasi-Fermi are calculated self-consistently from the current continuity equation.
- In the second part, the quasi-Fermi levels are kept constant, and the density and the potential are calculated self-consistently from the Schrödinger and Poisson equations.

In the input file *nin-resistor_Si_Sabathil_JCE_2002_1D_nnp.in* the variable `$QM` at the top of the file can be used for conveniently switching between classical `$QM = 0` and quantum mechanical `$QM = 1` calculations.

Electron densities

Now let us first have a look at the electron densities at equilibrium (i.e. applied bias $V = 0\,\mathrm{V}$) for the cases of classical and quantum mechanical calculations. The electron density is the sum over all three valleys (Γ-point, L-point and X-point (or Δ for Si) in the Brillouin zone), whereas for Si the dominant valley is the X valley which is sixfold degenerate (or twelvefold degenerate including spin degeneracy). Thus, we solve Schrödinger's equation only in the X valley and take for the other valleys the classical density only. For the quantum mechanical calculation we have to choose appropriate boundary conditions, which are to be specified by the variable `$BC_QM` at the top of the input file *nin-resistor_Si_Sabathil_JCE_2002_1D_nnp.in*.

In Figure 4.2.4.8 we compare the classical and the quantum mechanical electron densities for 0 V applied bias. The figure shows quantum mechanical calculating using Dirichlet and von Neumann boundary conditions. Dirichlet boundary conditions force the wave function to be zero at the boundaries and thus the electron density is zero there as well.

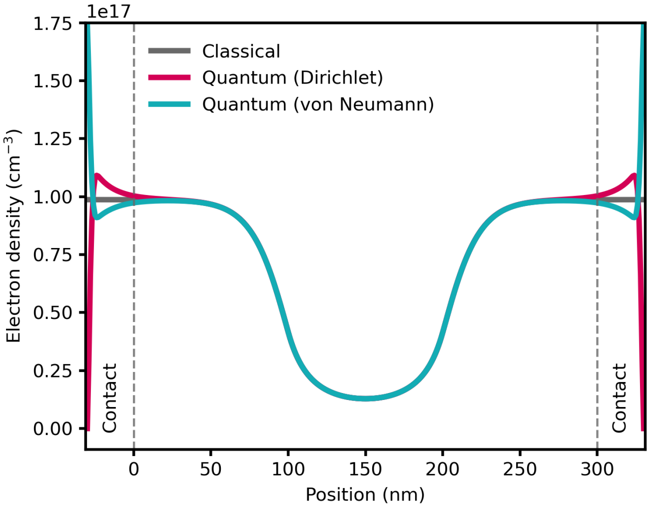


Figure 4.2.4.8: Comparison between classical and quantum mechanical electron densities for the n-i-n resistor. Quantum mechanical simulations using Dirichlet and von Neumann boundary conditions are shown.

I-V characteristics

Now we vary the applied bias from $0\,\mathrm{V}$ to $0.25\,\mathrm{V}$ in steps of $0.05\,\mathrm{V}$ and solve the drift-diffusion equations without taking quantum mechanical densities into account (classical simulation). Here, we compare two different models for calculating the mobility μ, namely, the constant mobility model ($\mu = 1417\,\mathrm{cm}^2/\mathrm{Vs}$) and the Hänsch mobility model. The Hänsch model is a high field mobility model, which includes the dependency of μ on the electric field.

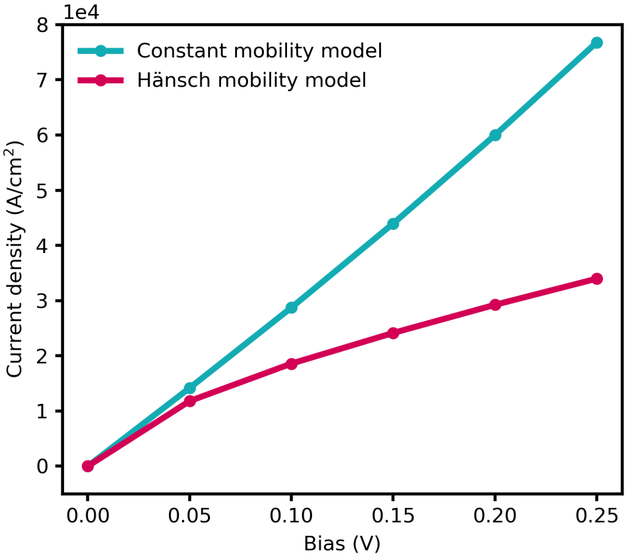


Figure 4.2.4.9: IV characteristics of the 300 nm Si n-i-n resistor for the constant mobility model and high field mobility model Hänsch (classical simulations).

The conduction band edges E_c and Fermi levels $E_{\mathrm{F,e}}$ (i.e. chemical potentials) for the electrons at different applied voltages are plotted in Figure 4.2.4.10.

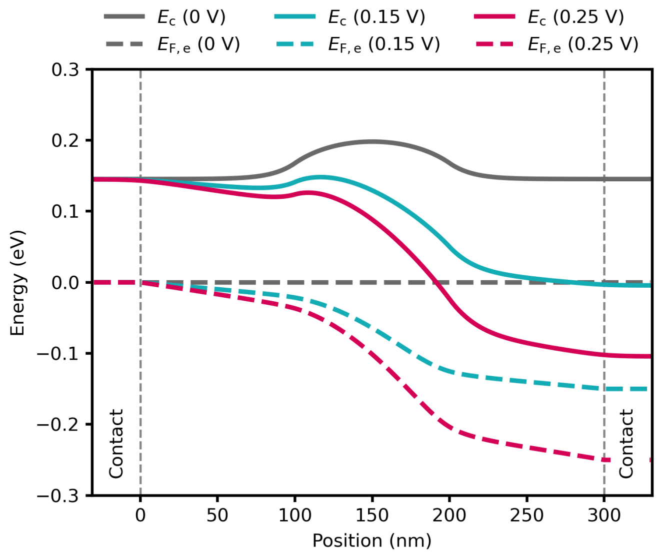


Figure 4.2.4.10: Conduction band edge profile E_c and electron quasi-Fermi levels $E_{\mathrm{F,e}}$ at bias points of $0\,\mathrm{V}$, $0.15\,\mathrm{V}$ and $0.25\,\mathrm{V}$.

Quantum mechanical calculations

As one may expect, true quantum mechanical effects play little role in this case and both the *nextnano++* (i.e. the semi-classical drift-diffusion) and the Pauli master equation approach yield practically identical results for the density and conduction band edge energies (i.e. for the electrostatic potential). We would like to point out that this good agreement is a nontrivial finding, as we calculate the density quantum mechanically with self-consistently computed local quasi-Fermi levels rather than semi-classically.

Figure 4.2.4.11 shows the conduction band edge energies and the electron densities for an applied bias of $0.25\,\mathrm{V}$. One can see that our results agree very well with the solution of the Pauli master equation *[Fischetti1998]*. Fischetti obtains for the current density $6.8 \cdot 10^4\ \mathrm{A/cm^2}$, whereas we obtain $3.65 \cdot 10^4\ \mathrm{A/cm^2}$ by using a (semi-)classical drift-diffusion model. However, we note that the current is directly proportional to the mobility in our model, i.e. changing the mobility therefore changes the value of the current. If we had chosen a constant mobility of $\mu = 1417\,\mathrm{cm^2/Vs}$, then the current at $0.25\,\mathrm{V}$ applied bias had been $7.67 \cdot 10^4\ \mathrm{A/cm^2}$ (compare with I-V characteristics above).

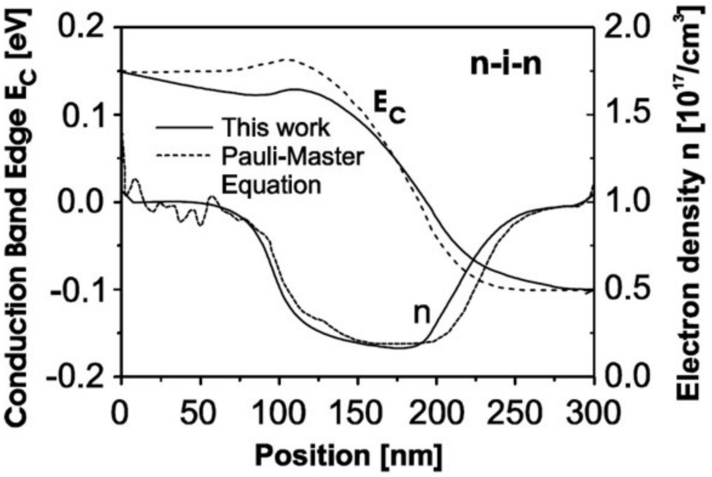


Figure 4.2.4.11: Calculated conduction band edges E_c and the electron densities n of the n-i-n structure as a function of position inside the structure. The results obtained from the Pauli master equation *[Fischetti1998]* are compared to our quantum mechanical results (full lines).

Conclusion

Here, we demonstrated our approach to calculate the electronic structure in non-equilibrium, where we combine the stationary solutions of the Schrödinger equation with a semi-classical drift-diffusion model. For the electrostatic potential and the charge carrier density, the method leads to a very good agreement with the more rigorous Pauli master equation approach. In addition, the current can also be described accurately.

Last update: nnnn/nn/nn

4.2.5 Other

— EDU — Interpolation of 2-component alloys

Header

Files for the tutorial located in *nextnano++\examples\education*

- *Interpolation_In(x)Ga(1-x)As_1D_linear_quadratic_nnp.in*
- *Interpolation_In(x)Ga(1-x)As_1D_cubic_nnp.in*
- *Interpolation_A(x)B(1-x)C_1D_linear_quadratic_nnp.in* - for arbitrary materials
- *Interpolation_A(x)B(1-x)C_1D_cubic_nnp.in* - for arbitrary materials

Scope of the tutorial:

-

Main adjustable parameters in the input file:

- parameter `$linear`
- parameter `$STRAIN`

Relevant output files:

- *bias_00000\bandedges.dat*

Introduction

In *Interpolation schemes*, you can see how to introduce interpolations in your simulation system. This tutorial helps you understand that more through plotting band offsets of a ternary compound In(x)Ga(1-x)As with the different interpolation schemes.

Band offsets also provides with some insights into how to define band offsets, which is related to this tutorial.

How to set up simulations and why

First, we define *structure{ }* to build our simulation system.

```
structure{
    region{
        ternary_linear{   # the composition x of In(x)Ga(1-x)As varies linearly
            name      = $material
            alloy_x   = [ 0.0, 1.0 ]       # vary x from 0.0 to 1.0 in In(x)Ga(1-x)As
            x         = [ $xmin, $xmax ]   # x coordinate of start and end point (nm)
        }
        line{ x = [ $xmin, $xmax ] }        # In(x)Ga(1-x)As exists from 0.0 to 1.0␣
↪along the x direction
        contact{ name = "fermi_contact" }   # This region will be defined as a␣
↪contact. In this case, the contact is called "fermi_contact"
    }
}
```

As a result, pure GaAs exists at $x = 0$ (nm) and pure InAs exists at $x = 1$ (nm). The composition varies linealy respect to x coordinate (nm).

Next, we consider what outputs to obtain from the simulation. We want to know the band offsets of In(x)Ga(1-x)As, therefore, we need the syntax *classical{ }*.

```
classical{
    Gamma{}              # a conduction band with a minimum at Gamma point
    HH{}                 # a heavy-hole valence band with a minimum at Gamma point
    LH{}                 # a light-hole valence band with a minimum at Gamma point
    SO{}                 # a split-off valence band with a minimum at Gamma point
    output_bandedges{}   # obtain band edges above
    output_bandgap{}     # obtain a band gap energy (optional)
}
```

The result is folded inside *bias_00000\bandedges.dat*.

We also have to initialize the poisson condition in *poisson{ }*. We do not want to apply an electric field to the simulation because it affects the band offsets. Therefore, we explicitly define no electric field in the simulation.

```
poisson{
    electric_field { strength = 0 }
}
```

If you use `charge_neutral{}` instead, it causes an electric field to require charge neutrality at all grid points. You can get more information in *poisson{ }*

Lastly, we introduce strain effects into the system. The strain is caused by the mismatch of lattice constants between the substrate InP and In(x)Ga(1-x)As. We assume that the strain is homogeneous.

Thus, we use `pseudomorphic_strain{ }` here.

```
$STRAIN = 0 # Choose strain option: 1: include strain, 0: do not include strain ␣
↪(ListOfValues: 0, 1)
```

```
strain{
    pseudomorphic_strain{ }
}
```

To ignore the strain, we use `$STRAIN`. If `$STRAIN = 1`, we take account into strain. If `$STRAIN = 0`, we do not.

```
run{
!IF($STRAIN)
    strain{ }
!ENDIF
}
```

This is necessary to calculate strain effects. We will see the strain effects to the band offsets of In(x)Ga(1-x)As at the end of this tutorial. Refer to *strain{ }* for further information.

Interpolations

We have three interpolation schemes, according to *Interpolation schemes*. Note that material parameters $P_{ABC}(x)$, P_{AC}, and P_{BC} correspond to the ones of In(x)Ga(1-x)As, pure InAs and pure GaAs, respectively.

Linear - no bowing

In this scheme, the material parameter $P_{ABC}(x)$ is represented as follows,

$$P_{InGaAs}(x) = x \times P_{InAs} + [1 - x] \times P_{GaAs}$$

This formula means that all material parameters of In(x)Ga(1-x)As are independent of a bowing parameter. There are three necessary material parameters (the energy gap E_g^Γ, the average energy of three top valence bands $E_{v,av}$, and the spin-orbit splitting energy Δ_{so}) to obtain band offsets of In(x)Ga(1-x)As (*Band offsets*).

Therefore, for example, in terms of the energy gap ($E^\Gamma_{g,InGaAs}$), the following formula holds.

$$E^\Gamma_{g,InGaAs}(x) = x \times E^\Gamma_{g,InAs} + [1 - x] \times E^\Gamma_{g,GaAs}$$

This is also true for the other two parameters ($E_{v,av}$ and Δ_{so}).

We need to define those parameters of InAs and GaAs with `database{ }`.

```
database{
    # All the material parameters of InAs here (equivalent to P_InAs)
    binary_zb{
        name = InAs
        conduction_bands{
            Gamma{
                bandgap         = 0.417      # E_{g,InAs}^{Gamma}, Vurgaftman1 (0 K)
                bandgap_alpha   = 0.276e-3   # Vurgaftman1
                bandgap_beta    = 93         # Vurgaftman1
            }
        }
        valence_bands{
            bandoffset   = 1.390   # E_{v,av,InAs}, A. Zunger
            delta_SO     = 0.39    # Delta_{so,InAs}, Vurgaftman1
        }
    }

    # All the material parameters of InAs here (equivalent to P_GaAs)
    binary_zb{
        name = GaAs
        conduction_bands{
            Gamma{
                bandgap         = 1.519       # E_{g,GaAs}^{Gamma}, Vurgaftman1 (0 K)
                bandgap_alpha   = 0.5405e-3   # Vurgaftman1
                bandgap_beta    = 204         # Vurgaftman1
            }
        }
```

(continues on next page)

(continued from previous page)

```
        valence_bands{
            bandoffset   = 1.346   # E_{v,av,GaAs}, A. Zunger
            delta_SO     = 0.341   # Delta_{so,GaAs}, Vurgaftman1
        }
    }
```

Then, we further define bowing parameters, which are all `0` in linear interpolation, inside `database{ }` as well.

```
    # All bowing parameters are set to 0 in linear interpolation
    ternary_zb{
        name        = "In(x)Ga(1-x)As"
        valence     = III_V
        binary_x    = InAs
        binary_1_x  = GaAs

        conduction_bands{
            Gamma{ bandgap = 0.0 }   # set to 0 deliberately
        }
        valence_bands{
            bandoffset   = 0.0   # set to 0 deliberately
            delta_SO     = 0.0   # set to 0 deliberately
        }
    }
```

The original database file (default: *database_nnp.in*) that *nextnanomat* refers to has data about In(x)Ga(1-x)As, thus, it is automatically adopted and overwrites your database unless you explicitly define that they are equivalent to `0`. Therefore, **you have to check the original database and how the bowing parameters of materials are defined before you define them by your own.**

Quadratic - constant bowing

In this scheme, the material parameter $P_{ABC}(x)$ is represented as follows,

$$P_{InGaAs}(x) = x \times P_{InAs} + [1-x] \times P_{GaAs} - x[1-x] \times b_{InGaAs}$$

b_{InGaAs} is a constant bowing parameter and we have to define it inside `database{ }` in this case. We also have to define parameters P_{InAs} and P_{GaAs} as well as in the linear scheme.

```
database{
    # All the material parameters of InAs here (equivalent to P_InAs) as well as in␣
↪the linear scheme
    binary_zb{
        name = InAs
    }

    # All the material parameters of InAs here (equivalent to P_GaAs) as well as in␣
↪the linear scheme
    binary_zb{
        name = GaAs
    }
```

Then, we define constant bowing parameters b_{InGaAs} as follows.

```
    # All bowing parameters are constant in quadratic interpolation
    ternary_zb{
        name          = "In(x)Ga(1-x)As"
        valence       = III_V
```

(continues on next page)

(continued from previous page)

```
        binary_x     = InAs
        binary_1_x   = GaAs

        conduction_bands{
            Gamma{ bandgap = 0.477 }   # Vurgaftman1
        }
        valence_bands{
            bandoffset   = -0.43   # the band offset ( = average valence band edge␣
↪energy)
            delta_SO     = 0.15    # Vurgaftman1
        }
    }
```

Here, some necessary parameters to describe band offsets, for example $E^{\Gamma}_{g,InGaAs}$, is represented as follows,

$$E^{\Gamma}_{g,InGaAs} = x \times E^{\Gamma}_{g,InAs} + [1-x] \times E^{\Gamma}_{g,GaAs} - x[1-x] \times b_{InGaAs}$$

b_{InGaAs} is the bowing parameter for the band gap and defined in the code as `Gamma{ bandgap = 0.477}`.

This is true for the other two parameters ($E_{v,av}$ and Δ_{so}) as well.

Cubic - composition-dependent bowing

In this scheme, the material parameter $P_{ABC}(x)$ is represented as follows,

$$P_{InGaAs}(x) = x \times P_{InAs} + [1-x] \times P_{GaAs} - x[1-x] \times b_{InGaAs}(x)$$

$$b_{InGaAs}(x) = x \times b_{In(x)Ga(1-x)As \to InAs} + [1-x] \times b_{In(x)Ga(1-x)As \to GaAs}$$

$b_{InGaAs}(x)$ is a composition-dependent bowing parameter. The $b_{In(x)Ga(1-x)As \to InAs}$ is a constant bowing parameter for nearly pure InAs ($x = 1$), while the $b_{In(x)Ga(1-x)As \to GaAs}$ is also a constant bowing parameter for nearly pure GaAs ($x = 0$).

To define $b_{In(x)Ga(1-x)As \to InAs}$, and $b_{In(x)Ga(1-x)As \to GaAs}$, we need `bowing_zb{}`. Moreover, `ternary2_zb{}` should be used to relate all the bowing parameters and the component materials (InAs and GaAs) for the alloy (In(x)Ga(1-x)As). Again, note that we also have to define parameters P_{InAs} and P_{GaAs} as well as in the linear scheme.

```
database{
    # All the material parameters of InAs here (equivalent to P_InAs) as well as in␣
↪the linear scheme
    binary_zb{
        name = InAs
    }

    # All the material parameters of InAs here (equivalent to P_GaAs) as well as in␣
↪the linear scheme
    binary_zb{
        name = GaAs
    }
```

Then, we define composition-dependent bowing parameters as follows. As explained before, the original database has data about In(x)Ga(1-x)As. Therefore, we need `ternary2_zb{}` to have a different name from the one in `ternary_zb{}` to avoid duplication between them.

```
    bowing_zb{
        name      = "InGaAs_Bowing_InAs"
        valence   = III_V
```

(continues on next page)

(continued from previous page)

```
        conduction_bands{
            Gamma{ bandgap = 0.359 }   # b_In(x)Ga(1-x)As ---> b_InAs (x = 1)
        }
        valence_bands{
            bandoffset   = -0.43   # the band offset ( = average valence band edge␣
↪energy)
            delta_SO     = 0.15    # Vurgaftman1
        }
    }

    bowing_zb{
        name      = "InGaAs_Bowing_GaAs"
        valence   = III_V
        conduction_bands{
            Gamma{ bandgap = 1.43 }   # b_In(x)Ga(1-x)As ---> b_GaAs (x = 0)
        }
        valence_bands{
            bandoffset   = -0.43   # the band offset ( = average valence band edge␣
↪energy)
            delta_SO     = 0.15    # Vurgaftman1
        }
    }

    ternary2_zb{
        name         = "In(x)Ga(1-x)As_cubic"   # rename to avoid duplication with␣
↪data on the original database
        valence      = III_V
        binary_x     = InAs
        binary_1_x   = GaAs
        bowing_x     = InGaAs_Bowing_InAs       # b_In(x)Ga(1-x)As ---> b_InAs (x = 1)
        bowing_1_x   = InGaAs_Bowing_GaAs       # b_In(x)Ga(1-x)As ---> b_GaAs (x = 0)
    }
```

Here, some necessary parameters to describe band offsets, for example $E^{\Gamma}_{g,InGaAs}$, is represented as follows, As explained before,

$$E^{\Gamma}_{g,InGaAs} = x \times E^{\Gamma}_{g,InAs} + [1-x] \times E^{\Gamma}_{g,GaAs} - x[1-x] \times b_{InGaAs}(x)$$

$b_{InGaAs}(x)$ is the bowing parameter for the bang gap and defined as the formula below on the Table 6.14 in *[Adachi2009]*.

$$b_{InGaAs}(x) = 0.359 + 0.491 \cdot (1-x) + 0.580 \cdot (1-x)^2$$

Therefore,

$$b_{In(x)Ga(1-x)As \rightarrow InAs} = b_{InGaAs}(1) = 0.359 + 0.491 \cdot (1-1) + 0.580 \cdot (1-1)^2 = 0.359$$

$$b_{In(x)Ga(1-x)As \rightarrow GaAs} = b_{InGaAs}(0) = 0.359 + 0.491 \cdot (1-0) + 0.580 \cdot (1-0)^2 = 1.43$$

Because we do not have formulas for the bowing parameters for $E_{v,av}$ and Δ_{so}, we define them as the same values between InAs and GaAs in the code above. This means that the two bowing parameters are constant and have the quadratic scheme for the valence bands.

Band offsets with the different schemes

According to the three schemes, which is explained above, we plot band offsets of In(x)Ga(1-x)As (Figure 4.2.5.1).

Note that we define the bowing parameters for $E_{v,av,InGaAs}(x)$ and $\Delta_{so,InGaAs}(x)$ as constant in the cubic scheme, therefore valence bands in the scheme are plotted with the quadratic scheme instead. Without strain, E_{HH}

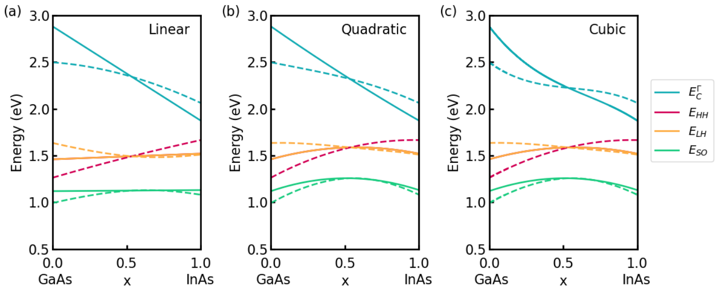


Figure 4.2.5.1: Band edges of In(x)Ga(1-x)As with a linear scheme in (a). (b) is with a quadratic scheme. (c) is with a cubic scheme. The band edges without strain are plotted with solid lines. The ones with strain are plotted with dotted lines.

and E_{LH} are degenerated in the all schemes. When strain is introduced due to the mismatch of lattice constants between the substrate InP and In(x)Ga(1-x)As, band edges are bent. This is because interpolations are executed first and then the strain is introduced to shift band energies.

Exercises

Plot band offsets of Al(x)Ga(1-x)As with the following steps:

- check the original database and how it is defined in it
- plot them with the linear scheme
- plot them with the quadratic scheme
- plot them with the cubic scheme
- introduce strain into the simulations and check the effects

You can get some clues to solve them in *Interpolation schemes* and *Band offsets*.

Last update: 08/03/2024

4.3 p-n Junctions & Solar Cells

4.3.1 — FREE — GaAs p–n junction

Author Stefan Birner

Note

See a *tutorial on IV curves* for pn junctions described here

Input Files:

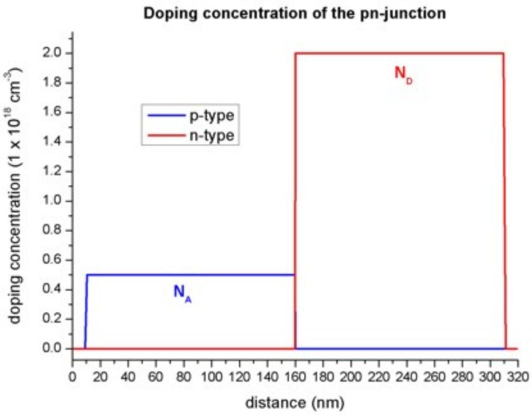
- *pn_junction_GaAs_1D_nnp.in*
- *pn_junction_GaAs_2D_nnp.in*
- *pn_junction_GaAs_3D_nnp.in*

This tutorial aims to reproduce Figure 3.1 (p. 51) of Joachim Piprek's book *"Semiconductor Optoelectronic Devices - Introduction to Physics and Simulation"* (Section 3.2 "pn-junctions")

Doping concentration

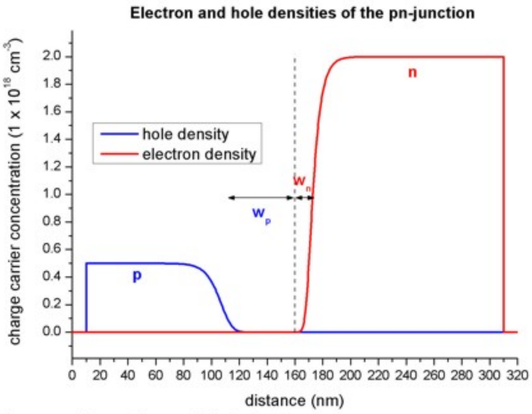
The structure consists of 300 nm GaAs. At the left and right boundaries, metal contacts are connected to the GaAs semiconductor (i.e. from 0 nm to 10 nm, and from 310 nm to 320 nm). The structure is p-type doped from 10 nm to 160 nm and n-type doped from 160 nm and 310 nm.

The following figure shows the concentration of donors and acceptors of the p-n junction. In the p-type region between 10 nm and 160 nm, the number of acceptors, N_A is 0.5×10^{18} cm^{-3} In the n-type region between 160 nm and 310 nm, the number of donors, N_D is 2.0×10^{18} cm^{-3}



Carrier concentrations

The equilibrium condition for a p-n junction is achieved by a small transfer of electrons from the n region to the p region, where they recombine with holes. This leads to a **depletion region** (depletion width = $w_p + w_n$), i.e. the region around the p-n junction only has very few free carriers left. The following figure shows the electron and hole densities and the depletion region around the p-n junction at 160 nm. Here, we assumed that all donors and acceptors are fully ionized.



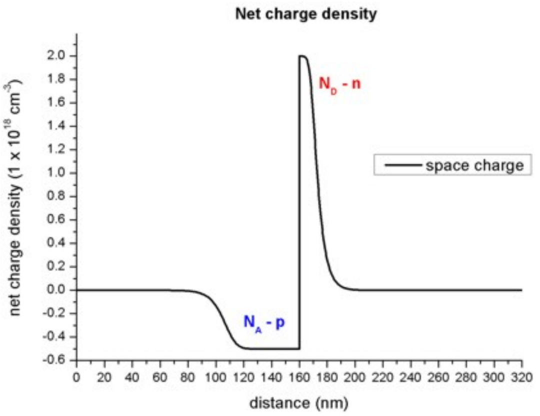
Note: Donors and acceptors are fully ionized!

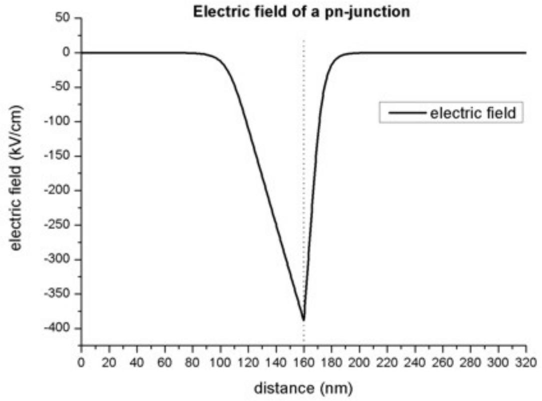
Net charges (space charge)

In the depletion region, a net charge results from the ionized donors N_D and ionized acceptors N_A. The following figure shows the net charge density of the p-n junction.

Electric field

The slope of the electric field is proportional to the net charge (Poisson equation), thus the extremum of the electric field is expected to be at the p-n junction. In regions without charges, the electric field is zero. The following figure shows the electric field of the p-n junction.





The extremum of the electric field F_{max} (at 160 nm) can be approximated as follows:

$$\begin{aligned} F_{max} &= \frac{-eN_A w_p}{\epsilon\epsilon_0} = -6.997 \times 10^{14} \text{V/m}^2 w_p = 387 \text{kV/cm} \\ &= \frac{-eN_D w_n}{\epsilon\epsilon_0} = -2.799 \times 10^{15} \text{V/m}^2 w_n = 386 \text{kV/cm} \end{aligned}$$

Symbol	Value
e	1.6022×10^{-19}As
ϵ	12.93 (Dielectric constant of GaAs)
ϵ_0	8.854×10^{12}As/(Vm)
N_A	0.5×10^{18}cm^{-3}
N_D	2.0×10^{18}cm^{-3}
w_p	55.3 nm
w_n	13.8 nm

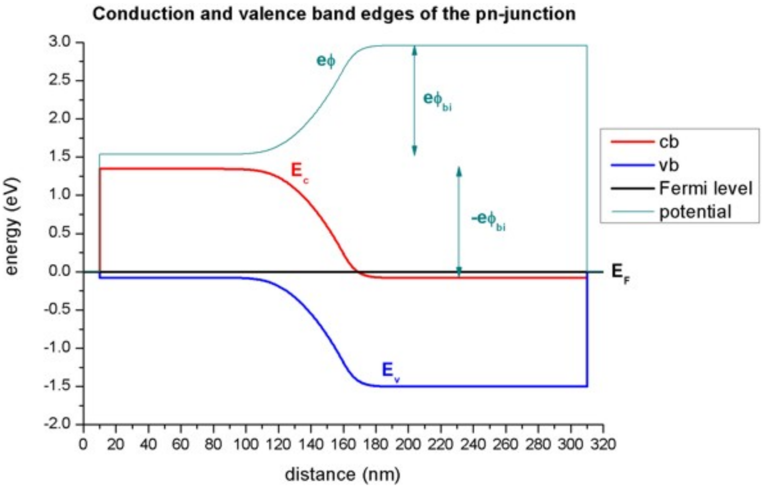
Electrostatic potential, conduction and valence band edges

In regions, where the electric field is zero, the electrostatic potential is constant. The electrostatic potential phi determines the conduction and valence band edges:

- $E_c = E_{c0} - e\phi$
- $E_v = E_{v0} - e\phi$

The following figure shows the conduction and valence band edges, the electrostatic potential and the Fermi level of the p-n junction.

Without external bias (i.e. equilibrium), the Fermi level E_F is constant ($E_F = 0$eV).



The built-in potential ϕ_{bi} was calculated by *nextnano++* to be equal to 1.426 V It can be approximated as follows:

$$\phi_{bi} = F_{\max}(w_p + w_n)/2$$

Assuming $F_{\max} = 387\text{kV/cm}$, this would result in a depletion width: $w_p + w_n = 73.7\text{nm}$

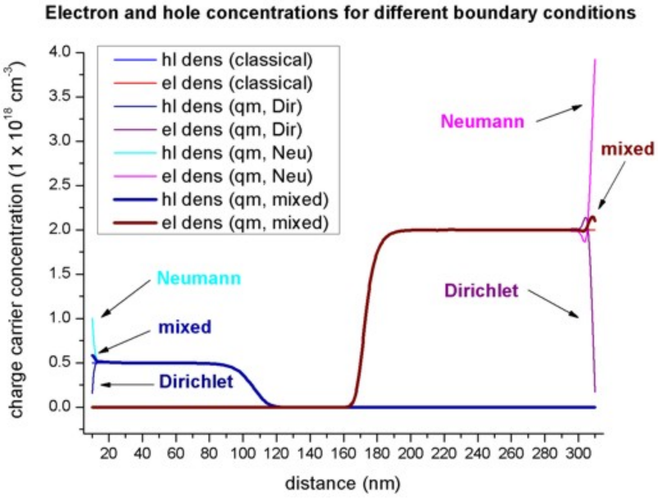
To allow for a constant chemical potential (i.e. constant Fermi level E_F), a total potential difference of $-e\phi_{bi}$ is required.

Quantum mechanical solution

Using the *nextnano³* input file *pn_junction_GaAs_1D_QM_nn3.in*, we can solve the Schrödinger equation for the electrons, light and heavy holes in the single-band approximation over the whole device, rather than classically. We calculate up to 300 eigenvalues for each band. Thus the electron and hole densities are calculated **purely quantum mechanically**. The following figure shows the electron and hole concentrations for the classical and quantum mechanical calculations. For the QM calculations, different boundary conditions were used.

- **Dirichlet** boundary conditions force the wave functions to be zero at the boundaries, thus the density goes to zero at the boundaries which is unphysically.
- **Neumann** boundary conditions lead to unphysically large values at the boundaries.

For the classical calculation, the densities at the boundaries are constant. Nevertheless, in the interesting region around the p-n junction, all four options lead to identical densities.

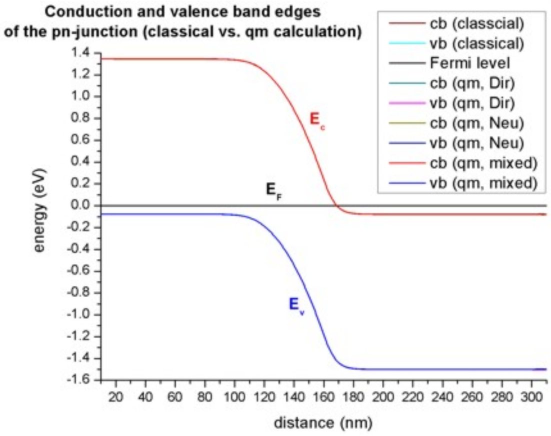


The following figure shows the band edges of the p-n junction for the four cases:

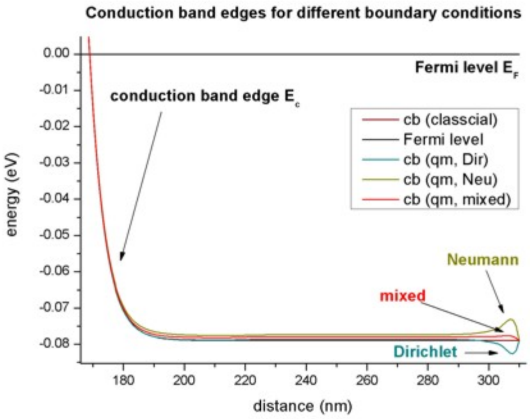
- Classical calculation

- Quantum mechanical calculation with **Dirchlet** boundary conditions
- Quantum mechanical calculation with **Neumann** boundary conditions
- Quantum mechanical calculation with mixed boundary conditions (this feature is no longer supported)

For all cases the band edges are identical in the area around the p-n junction. Tiny deviations exist at the boundaries of the device.



This figure is a zoom into the right boundary of the conduction band edge. On this scale, the tiny deviations for the different boundary conditions can be clearly seen.



Non-equilibrium

So-called "quasi-Fermi levels" which are different for electrons (E_F, n) and holes (E_F, p) are used to describe nonequilibrium carrier concentrations.

In equilibrium the quasi-Fermi levels are constant and have the same value for both electrons and holes ($E_{Fn} = E_{Fp} = 0\text{eV}$). The current is proportional to the mobility and the gradient of the quasi-Fermi level E_F.

2D/3D Simulations

- *pn_junction_GaAs_2D_nnp.in*
- *pn_junction_GaAs_3D_nnp.in*

These input files are for the same p-n junction structure as in the 1D case, but extended into 2D and 3D.

- 2D: rectangle of dimension 320 nm x 200 nm
- 3D: cuboid of dimension 320 nm x 200 nm x 100 nm

Complete input file for nextnano++

```
# pn_junction_GaAs_1D_nnp.in                                                ␣
↪!

global{
simulate1D{}

temperature = 300.0                                                      #␣
↪Kelvin

substrate{ name = "GaAs" }

crystal_zb{
        x_hkl = [1, 0, 0]
        y_hkl = [0, 1, 0]
}
}

grid{
#
# For consistency reasons, we use the same nonuniform grid spacing as the␣
↪nextnano3 input file.
# However, using jumps in the grid spacing (e.g. at x=100.0 where the grid␣
↪spacing changes abruptly)
# is not a good practice, as numerical errors increase.
#
xgrid{
        line{ pos = 0.0      spacing = 2.0  }
        line{ pos = 10.0     spacing = 2.0  }
        line{ pos = 10.0     spacing = 1.0  }
        line{ pos = 100.0    spacing = 1.0  }
        line{ pos = 100.0    spacing = 0.5  }
        line{ pos = 140.0    spacing = 0.5  }
        line{ pos = 140.0    spacing = 0.25 }
        line{ pos = 180.0    spacing = 0.25 }
        line{ pos = 180.0    spacing = 0.5  }
        line{ pos = 220.0    spacing = 0.5  }
        line{ pos = 220.0    spacing = 1.0  }
        line{ pos = 310.0    spacing = 1.0  }
        line{ pos = 310.0    spacing = 2.0  }
        line{ pos = 320.0    spacing = 2.0  }
}
}

structure{
output_region_index{ boxes = no }
output_material_index{ boxes = no }
output_alloy_composition{ boxes = no }
output_impurities{ boxes = no }

region{
        everywhere{}
        binary{ name = "GaAs" }
}
region{
        line{
```

(continues on next page)

(continued from previous page)

```
                x = [0.0, 10.0]
        }
        binary{
                name = "GaAs"
        }
        contact { name = source }
}
region{
        line{
                x = [10.0, 310.0]
        }
        binary{
                name = "GaAs"
        }
}
region{
        line{
                x = [310.0, 320.0]
        }
        binary{
                name = "GaAs"
        }
        contact { name = drain }
}

region{
        line{
                x = [ 0.0, 160.0]
        # x = [10.0, 160.0]     # doping must not start at 10.0
        }
        doping{
                constant{
                        name = "p-type"
                        conc =  0.5e18
                }
        }
}

region{
        line{
        # x = [160.0, 310.0]    # doping must not end at 310.0
                x = [160.0, 320.0]
        }
        doping{
                constant{
                        name = "n-type"
                        conc = 2.0e18
                }
        }
}
}

impurities{
#    donor{ name = "n-type" energy = 0.027   degeneracy = 2 }
# acceptor{ name = "p-type" energy = 0.0058  degeneracy = 4 }
        donor{ name = "n-type" energy = -1000.0 degeneracy = 2 }    # '-
```

(continues on next page)

(continued from previous page)

```
↪1000.0' eV = all ionized
acceptor{ name = "p-type" energy = -1000.0 degeneracy = 4 }    # '-1000.0'␣
↪eV = all ionized
}

contacts{
ohmic{ name = "source" bias = 0.0 }
ohmic{ name = "drain"  bias = 0.0 }
}

classical{
Gamma{}
HH{}
LH{}
SO{}

output_bandedges{ averaged = no}
output_carrier_densities{}
output_ionized_dopant_densities{}
output_intrinsic_density{}
}

poisson{
output_potential{}
output_electric_field{}
}

run{
solve_poisson{ }
}
```

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.3.2 I–V characteristic of GaAs p–n junction | 1D/2D/3D

Warning

This tutorial is under construction

Input Files:

- *pn_junction_GaAs_ForwardBias_1D_nnp.in*
- *pn_junction_GaAs_ForwardBias_2D_nnp.in*
- *pn_junction_GaAs_ForwardBias_3D_nnp.in*

Scope:

This tutorial shows how to perform bias sweeps to compute IV curves.

Most relevant keywords:

- `contacts{ ohmic{ bias } }`
- `contacts{ ohmic{ steps } }`

Output Files:
IV_characteristics.dat

Introduction

In the present tutorial we are concerned with the question of how to determine the I-V characteristics of a device. For this purpose, one side of the device is biased, and the simulation is repeatedly executed for a range of different voltages. The *nextnano++* tool offers a convenient way to perform this bias sweep. The computed current and voltage values are automatically collected in one file. In what follows, we simulate a simple p-n junction (see also *p-n junction tutorial*), to demonstrate the usage of the keywords which are relevant to trigger the bias sweep.

Input File

First, two contact regions at both ends of the structure are needed: one as source and the other as drain channel. The contact regions will allow us to bias the structure by applying an explicit voltage to either side of the device.

```
structure{

    ...

    region{
        line{ x = [ -$BOUNDARY, -$SIZE] }   # contact on left device boundary
        contact{ name = leftgate }          # contact name
    }
    region{
        line{ x =[ $SIZE, $BOUNDARY] }      # contact on right device boundary
        contact{ name = rightgate }         # contact name
    }

    ...
}
```

The actual properties of the contacts are specified inside the group `contacts{ }`. There are several contact types available (e.g. `ohmic{}`, `schottky{}`, `fermi{}`, . . .), each imply different boundary conditions which are applied to the electrostatic potential $\phi(x)$. In our case we choose `ohmic{}` contacts.

The voltage on the right side is set to zero (bias = 0 V) and the left contact is biased. In order to sweep over different voltages automatically, the bias for the left contact is to be specified as a vector with start and end value (bias = [$\mathrm{V_{start}}$, $\mathrm{V_{end}}$]). The attribute steps specifies the total number of voltage values.

```
contacts{
    ohmic{ # left contact
        name = leftgate                 # refer to region labeled 'leftgate'
        bias = [ 0 , 1.0]               # [V] start and end value of bias sweep
        steps = 20                      # number of sweep values
    }
    ohmic{ # right contact
        name = rightgate                # refer to region labeled 'rightgate'
        bias = 0.0                      # [V] unbiased
    }
}
```

For simulating charge carrier transport the Poisson and Current equation are solved self consistently. It is important to use proper convergence parameter inside the group `run{ }`.

Note

It is important to be aware that applying different voltages change the physical properties of the system, e.g. the electric field, and therefore it is not guaranteed that one set of convergence parameters are applicable to all voltages of the sweep.

```
poisson{
    charge_neutral{}   # initialize Fermi levels in the contacts that charge␣
↪neutrality is obtained

    # output settings
    output_potential{}
    output_electric_field{}
}

currents{
    electron_mobility{
        low_field_model = minimos       # mobility model
    }
    hole_mobility{
        low_field_model = minimos       # mobility model
    }

    recombination_model{   # recombination models
        SRH       = yes
        Auger     = yes
        radiative = yes
    }

    insulator_bandgap = 0.5         # initialization parameter

    minimum_density_electrons  = 1e-10      # convergence parameter
    minimum_density_holes      = 1e-10      # convergence parameter
    maximum_density_holes      = 1e15       # convergence parameter

    # output settings
    output_fermi_levels{}
    output_currents{ }
    output_mobilities{}
    output_recombination{}
}

run{
    current_poisson{
        iterations           = 1000      # max iteration
        current_repetitions  = 10        # current repetition
        alpha_fermi          = 0.7       # under-relaxation parameter
        residual_fermi       = 1e-12     # desired residual of Fermi levels
        output_log           = yes       # information about convergence behavior
    }
}
```

Results

When the input file is executed, simulation results for each bias value are written in separate folders. These are located in the output folder of the simulation under *\bias_xxxxx* and contain e.g. band edges, electric fields, convergence behaviors, etc.

The output folder also contains a file with the combined current-voltage values. The corresponding file is labeled *IV_characteristics.dat*. The I-V curve, as presented in Figure 4.3.2.1, can be directly visualized in *nextnanomat*.

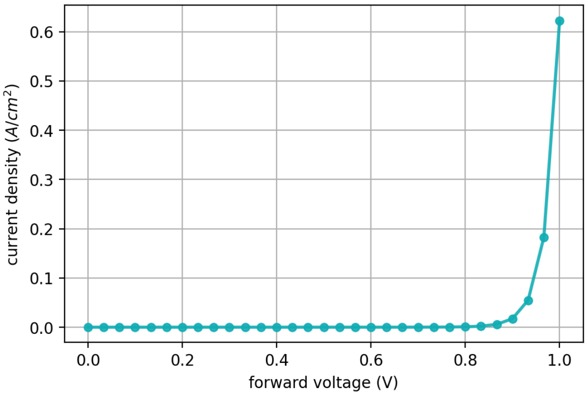


Figure 4.3.2.1: Current density as function of applied bias (1d simulation)

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.3.3 — NEW/EDU — p-n junction in the dark

Attention

This tutorial is under construction

Header

Files for the tutorial located in *nextnano++\examples\education*

- *pn-junction-dark_GaAs_Nelson_2003_1D_nnp.in*

Scope of the tutorial:

-

Main adjustable parameters in the input file:

- parameter `$min_density`
- parameter `$max_density`

Relevant output files:

- *bias_XXXXX\bandedges.dat*
- *bias_XXXXX\density_electon.dat*
- *bias_XXXXX\density_hole.dat*
- *bias_XXXXX\electric_field.dat*
- *bias_XXXXX\potential.dat*
- *IV_characteristics.dat*

Introduction

In this tutorial, you can learn fundamentals of p-n junction. We refer to §6 in [*NelsonPSC2003*] and §2 in [*Sze_Kwok_2007*] to make this tutorial. We look into the physical properties of the GaAs p-n junction at equilibrium first. Then, we apply forward bias and investigate the current-voltage characteristics. We apply the p-n junction to a solar cell and explain the basic principles of the solar cell in *— EDU — p-n junction under illumination*. If you are interested in simulation of solar cells, we recommend that you read it too.

At equilibrium

Figure 4.3.3.1 shows the schematic illustration of the p-n junction.

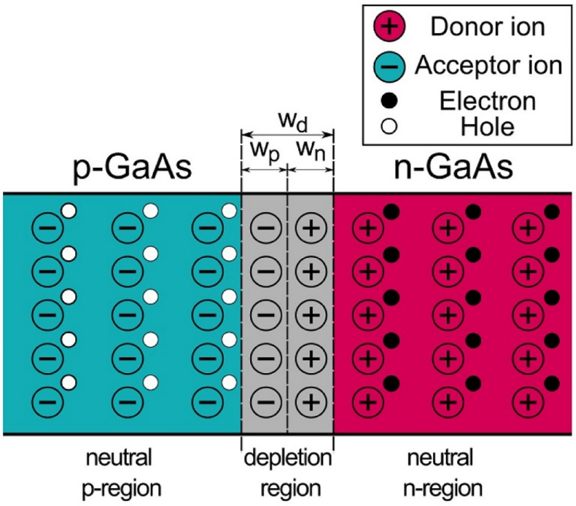


Figure 4.3.3.1: The schematic illustration of the p-n junction.

At equilibrium, the built-in-potential V_{bi} is formed across the space charge region. The process of forming the built-in-potential is explained below. First, the carrier density gradients arise across the junction when p-doped GaAs and n-doped GaAs are joined. Then, the free electrons in n-doped GaAs diffuse and combine with holes in p-doped GaAs. Similarly, the free holes in p-doped GaAs diffuse and combine with electrons in n-doped GaAs. On the other hand, the ionized dopants, such as negatively charged acceptor and positively charged donor, cannot move and are fixed at their initial positions. Therefore, **the ionized dopants in the region where the carriers are depleted form the electric field and the built-in-potential V_{bi} that impede the diffusion of majority carriers**. The space charge region (the width: w_d) denotes the region that is charged and loses the mobile carriers.

Figure 4.3.3.2 shows the basic characteristics of the diode at equilibrium.

Note that we assume the all dopants are ionized in the result to be consistent with **Fig. 6.3.** in *[NelsonPSC2003]*. You can see that the electric field is formed within the space change region and the voltage is equivalent to V_{bi} from Figure 4.3.3.2 (b) and (c).

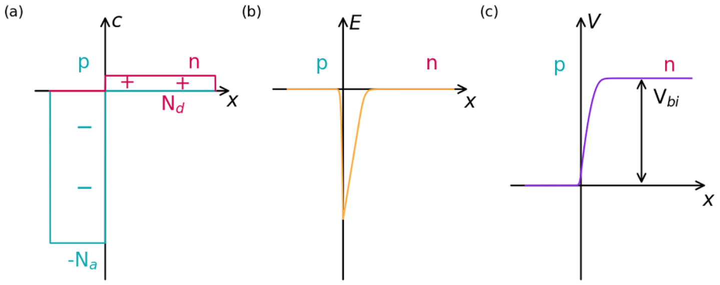


Figure 4.3.3.2: Some characteristics are shown across a p-n junction. (a) shows the dopant profile. (b) and (c) are the electric field and the potential across the space charge region, respectively.

Figure 4.3.3.3 shows (a) the band profiles and (b) the carrier densities at equilibrium. *bandedges.dat*, *density_electon.dat*, and *density_hole.dat* are used to produce this figure.

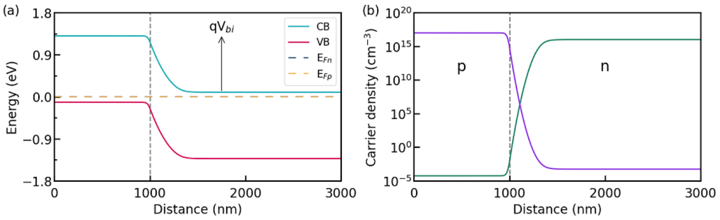


Figure 4.3.3.3: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

In (a), CB and VB represent conduction and valence band, respectively. E_{Fn} and E_{Fp} are the electron quasi Fermi level and the hole quasi Fermi level. The results are in a good agreement with **Fig. 6.5.** in *[NelsonPSC2003]*.

V_{bi} can be calculated at *potential.dat* and it is $2.7848 - 1.5779 = 1.207$ V in this case. The width of the space charge region w_{scr} can be acquired by the following procedures.

Since

$$w_p = \frac{1}{N_a}\sqrt{\frac{2\varepsilon_0\varepsilon V_{bi}}{q\left(\frac{1}{N_a}+\frac{1}{N_d}\right)}} \tag{4.3.3.1}$$

and

$$w_n = \frac{1}{N_d}\sqrt{\frac{2\varepsilon_0\varepsilon V_{bi}}{q\left(\frac{1}{N_a}+\frac{1}{N_d}\right)}}, \tag{4.3.3.2}$$

Thus,

$$w_{scr} = w_p + w_n = \sqrt{\frac{2\varepsilon_0\varepsilon}{q}\left(\frac{1}{N_a}+\frac{1}{N_d}\right)V_{bi}} \tag{4.3.3.3}$$

Each parameter corresponds to a value in the table below.

Parameter	Value
q (Elementary charge)	1.6022×10^{-19} C
ε_0 (Vacuum permittivity)	8.854×10^{-12} C/(Vm)
ε (Relative permittivity of GaAs)	12.93
N_a	1.0×10^{17} cm^{-3}
N_d	1.0×10^{16} cm^{-3}

Thus, w_{scr} is:

$$w_{scr} = \sqrt{\frac{2 \cdot 8.854 \times 10^{-14}\ \mathrm{C/(Vcm)} \cdot 12.93 \cdot 1.207\ \mathrm{V}}{1.6022 \times 10^{-19}\ \mathrm{C}} \frac{1.0 \times 10^{17}\ \mathrm{cm}^{-3} + 1.0 \times 10^{16}\ \mathrm{cm}^{-3}}{1.0 \times 10^{17}\ \mathrm{cm}^{-3} \cdot 1.0 \times 10^{16}\ \mathrm{cm}^{-3}}} = 4.356 \times 10^{-5}\ \mathrm{cm} = 435.6\ \mathrm{nm}$$

From the equation (4.3.3.3), you can see that the higher the dopant concentration is, the thinner w_{scr} becomes.

The derivation of the equations is explained in §6 in *[NelsonPSC2003]*.

Under applied bias

We look into the case of the diode under forward bias. Figure 4.3.3.4 shows animation of (a) the band profiles, (b) the electric field, and (c) the space charge, respect to the applied bias.

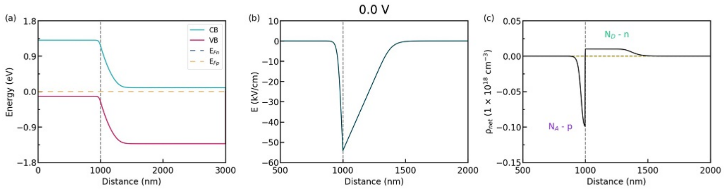


Figure 4.3.3.4: Some characters, (a) the band profiles, (b) the electric field, and (c) the space charge, respect to the applied bias.

As you see in Figure 4.3.3.4, the width w_{scr} decreases as the forward bias is applied. The width w_{scr} under the forward bias can be represented as follows.

$$w_{scr} = w_p + w_n = \sqrt{\frac{2\varepsilon_0\varepsilon}{q}\left(\frac{1}{N_a} + \frac{1}{N_d}\right)(V_{bi} - V)} \tag{4.3.3.4}$$

As the width w_{scr} decreases, the electric field that prevents the diffusion of majority carriers also decreases.

Whereas the current density across the diode is 0 at equilirium, applied bias enables majority carriers to diffuse across the junction. This means that a net current of electrons flow from **n** to **p**, and a net current of holes from **p** to **n**.

To see the effects of applied bias more clearly, let us look at the band profiles and carrier densities at 0.5 V.

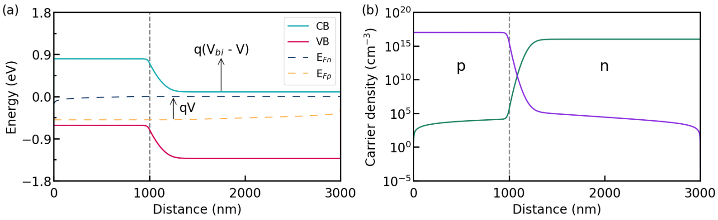


Figure 4.3.3.5: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

The results are consistent with **Fig. 6.6.** in *[NelsonPSC2003]* with high accuracy. The built-in-potential is reduced to $V_{bi} - V = 1.207 - 0.5 = 0.707$ V. Here, the difference between the quasi Fermi levels within the space charge region is equivalent to qV.

Thus,

$$qV = E_{Fn} - E_{Fp} \tag{4.3.3.5}$$

This relation can be seen from Figure 4.3.3.5 (a).

J-V curve

In this section, we sweep forward bias to acquire J-V curve. You can refer to *I–V characteristic of GaAs p–n junction | 1D/2D/3D* to understand how to apply bias in *nextnano*++.

Figure 4.3.3.6 shows the J-V curve of the diode. *IV_characteristics.dat* is used to produce this figure.

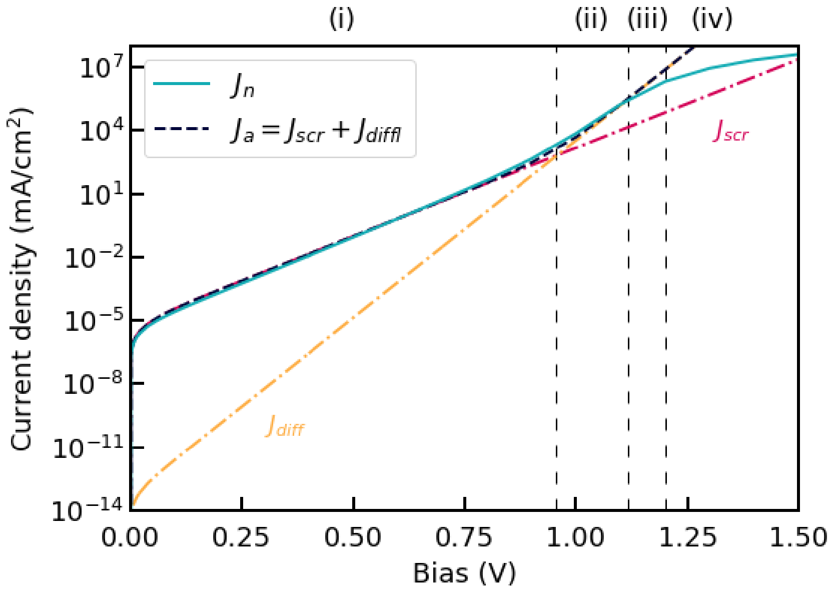


Figure 4.3.3.6: J-V curve of the diode. (i) space charge recombination current region, (ii) diffusion current region, (iii) high-injection region, (iv) series-resistence effect region.

The light-blue curve shows the numerical result in *nextnano*++. The violet and orange dashed-dotted curves are acquired analytically. They correspond to J_{scr} and J_{diff} in **Fig. 6.7.** in *[NelsonPSC2003]*, respectively.

J_{scr} is called the recombination current density and expressed in the following equation:

$$J_{scr}(V) = J_{scr,0}(\exp(qV/2k_BT) - 1), \tag{4.3.3.6}$$

where

$$J_{scr,0} = \frac{qn_i(w_p + w_n)}{\sqrt{\tau_n \tau_p}} \tag{4.3.3.7}$$

J_{diff} is called the diffusion current density and expressed in the following equation:

$$J_{diff}(V) = J_{diff,0}(\exp(qV/k_BT) - 1), \tag{4.3.3.8}$$

where

$$J_{diff,0} = q{n_i}^2 \left(\frac{D_n}{N_a L_n} + \frac{D_p}{N_d L_p} \right) \tag{4.3.3.9}$$

The parameters used in the expressions above are in the table.

Parameters	Description (unit)	Value used for the analytical J-V curve
k_B	Boltzmann constant (J/K)	1.3806E-23
T	The temperature (K)	300
n_i	The intrinsic carrier density (cm^{-3})	2.318E+6
$\tau_{n/p}$	The lifetimes of electrons/holes (s)	3.333 × 10^{-9} for J_{scr} and 1.0 × 10^{-10} for J_{diff} (*)
$D_{n/p}$	The diffusion coefficients of electrons/holes (cm^2/Vs)	219.73 / 20.681
$L_{n/p}$	The diffusion lengths of electrons/holes (cm)	1.4823 × 10^{-4} / 4.5476 × 10^{-5}

Attention

(*) There seems to be some errors related to the units in Fig. 6.7. in *[NelsonPSC2003]*
Therefore we used the lifetimes as fittig parameters.

The derivation of those equations above are described in §6 in *[NelsonPSC2003]*. J_a is the sum of J_{scr} and J_{diff} ($J_a = J_{scr} + J_{diff}$). Our result (the light-blue curve) is in a good agreement with J_a until $V \approx 1.2\ (V)$.

Our result shows the four distinct regions as marked Figure 4.3.3.6 (region (i), (ii), (iii), (iv)). In the next section, we identify the origins of the appearance of the regions.

Recombination current region

The region (i) is attributed to the recombination current region, where the contribution of J_{scr} is dominant. In this region, electrons and holes recombine within the space charge region since the region still exists. Therefore, the recombination current flows to compensate externally for the disappearance of the recombined carriers. As you can see from (4.3.3.6), in the semi-log plot $log(J)\ vs\ V$, the slope in the region (i) is $qV/2k_BT$.

Diffusion current region

The region (ii) is the diffusion current region. The contribution of J_{diff} is large in this region. Since the space change region almost disappears, a large amount of carriers starts to diffuse. This means that electrons are injected into p-doped GaAs and holes are injected into n-doped GaAs (minority carriers injection). As you can see from (4.3.3.8), in the semi-log plot $log(J)\ vs\ V$, the slope in the region (ii) qV/k_BT.

High-injection region

With increasing the forward bias towards V_{bi}, the injected hole density becomes comparable to the electron density at the n-side of the junction. You can see it in Figure 4.3.3.7 (b), where 1.2 V is applied to the diode.

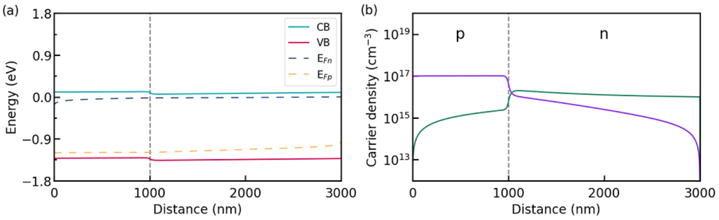


Figure 4.3.3.7: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

Then, the electron density must increase to maintain the neutrality. As a result, $n \approx p$ holds.

Because of the law of the junction,

$$np = n_i^2 \exp(qV/k_BT), \tag{4.3.3.10}$$

we acquire the equation as follows.

$$n = p = n_i \exp(qV/2k_BT) \tag{4.3.3.11}$$

Therefore, the current density becomes roughly proportional to $\exp(qV/2k_BT)$.

Series-resistance effect

At large currents, the voltage drop outside the space charge region becomes too large to ignore. This is equivalent to considering a single resistance (R) added in series to the ideal diode and corresponds to the region (iv). In this region, the diffusion current density becomes proportional to the applied voltage to the diode (V^*).

$$J_{diff,region(iv)} \approx J_{diff,0}\frac{qV^*}{k_BT}, \tag{4.3.3.12}$$

where

$$V^* = V - IR \tag{4.3.3.13}$$

Numerical control

Since we solve the current equation and the poisson equation (explanation: *Optoelectronic characterization*) self-consistently, we need some techniques to make the calculations more stable.

In this section, we introduce the effects of *minimum_density_electrons*, *minimum_density_holes*, *maximum_density_electrons*, and *maximum_density_holes*.

You should also check *Convergence*.

In Figure 4.3.3.6, we divide the simulation scheme into 3, depending on the magnitudes of minimum and maximum carrier densities (scheme (A), (B), and (C)). Scheme (A): $0 \sim 0.4$ V Scheme (B): $0.4 \sim 0.7$ V Scheme (C): $0.7 \sim 1.5$ V

First, the code below defines the magnitudes of the minimum and maximum carrier densities. Note that we use the variables `$min_density` and `$max_density` for convenience.

```
currents{
    minimum_density_electrons  = $min_density
    minimum_density_holes      = $min_density
    maximum_density_electrons  = $max_density
    maximum_density_holes      = $max_density

}
```

Usually, you can set the values of `$min_density_*` and `$max_density_*` by referring to *bias_XXXXX\density_electon.dat* and *bias_XXXXX\density_hole.dat*. In the scheme (C), the maximum electron and hole densities are about 1.0×10^{18} (cm^{-3}). Therefore, it is set to `1.0E+20`. Similarly, you can set `$minimum_density_*`. Since the minimum electron and hole densities are about 1.0×10^{0} (cm^{-3}), `$minimum_density_electrons = 1.0E-2` and `$minimum_density_holes = 1.0E-2` are enough low to evaluate the current density accurately. Note that $x = 0$ (nm) and $x = 3000$ (nm) correspond to the positions of the interfaces of diode/contact. Therefore, we do not include the carrier densities at the positions into the procedures.

In the scheme (B), `$minimum_density_electrons = 1.0E-2` and `$minimum_density_holes = 1.0E-2` is enough low as well. However, you have to take care of the magnitude of `$minimum_density_*`.

Figure 4.3.3.8 (a) shows the effect of the magnitude of `$minimum_density_*` on the current density under 0.5 V in the scheme (B).

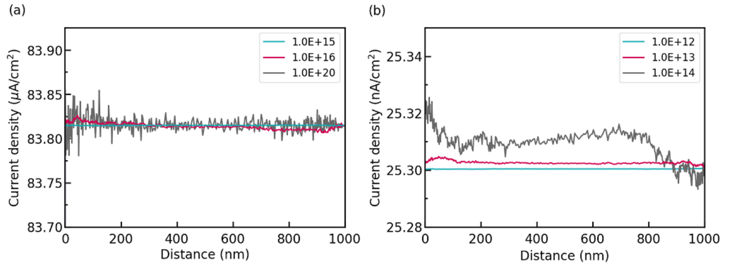


Figure 4.3.3.8: The effect of the magnitude of `$minimum_density_*` on the current density. (a) is under 0.5 V in the scheme (B). (b) is under 0.1 V in the scheme (A).

Although the current density has to be constant through the diode, it becomes unstable at `minimum_density_*` set to `1.0E+16` and `minimum_density*_` set to `1.0E+20`. Thus, you should set `$minimum_density_*` to `1.0E+15`, which shows the constant current density.

In the scheme (A), the same techniques should be applied. Figure 4.3.3.8 (b) shows the effect of the magnitude of `$maximum_density_*` on the current density under 0.1 V in the scheme (A). As you can see, `maximum_density_*` should be set to `1.0E+12` to keep the current density constant through the diode.

Exercises

under construction

Last update: 16/07/2024

4.3.4 — EDU — p-n junction under illumination

Attention

This tutorial is under construction

Header

Files for the tutorial located in *nextnano++\examples\education*

- *pn-junction-illuminated_GaAs_Nelson_2003_1D_nnp.in*

Main adjustable parameters in the input file:

- parameter `$sun`

Relevant output files:

- *bias_XXXXX\bandedges.dat*
- *bias_XXXXX\density_electon.dat*
- *bias_XXXXX\density_hole.dat*
- *bias_XXXXX\electric_field.dat*
- *bias_XXXXX\potential.dat*
- *IV_characteristics.dat*

Introduction

In this tutorial, we introduce simulation of a solar cell with *nextnano*++. This tutorial is based on §6 in *[NelsonPSC2003]* and §13 in *[Sze_Kwok_2007]*. Solar cells work based on p-n junction, which is explained in detail in *— NEW/EDU — p-n junction in the dark*. Therefore, we recommend that you read it before going through this tutorial. In addition, *GaAs solar cell* will help you understand the simulation scheme for solar cells used in *nextnano*++.

How to illuminate in nextnano++

To control the concentration of the irradiated light, you have to adjust some variables in *nextnano*++.

```
$sun    = 10       # concentration of the sun, 10 is used for this tutorial

optics{
    irradiation{
        min_energy          = 0.01
        max_energy          = 5
        energy_resolution   = 1e-4

        global_illumination{
            direction_x = 1
            database_spectrum{
                name = "Solar-ASTM-G173-global"
                concentration = $sun
            }
        }
        global_reflectivity{
            database_spectrum{ name = "GaAs" }
        }
        global_absorption_coeff{
            database_spectrum{ name = "GaAs" }
        }
    }
}
```

`min_energy` and `max_energy` correspond to the minimum and maximum energy of irradiated photons. `energy_resolution` is the energy step which is used to calculate optical properties. `$sun` controls the concentration of the incident light as it can be defined at `global_illumination{ database_spectrum{ concentration = $sun } }`. In this tutorial, `Solar-ASTM-G173-global`, which is equivalent to the solar spectrum, is also used as in *GaAs solar cell*. The data of reflectivity and absorption coefficient of GaAs is written at `database{ }` at the end of the input file. You can refer to *GaAs solar cell* for further information.

Short circuit

Let us investigate the behavior of p-n junction when it is illuminated by the sun light. First, we consider when the voltage across the diode is zero. We call the condition **short circuit**. The junction before the illumination is at equilibrium, having the space charge region and the electric field as shown in Figure 4.3.4.1 (a). The electric field impedes the diffusion of majority carriers as explained in *— NEW/EDU — p-n junction in the dark*.

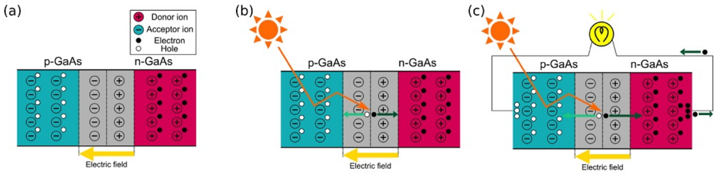


Figure 4.3.4.1: The schematic images showing the principles of a solar cell. (a) is the p-n junction at equilibrium. When it is illuminated, an electron-hole pair is generated at the junction (b). The current runs as long as the diode is illuminated (c). We assume the resistance of the light bulb is zero because of the short circuit.

When the light is illuminated, it excites an electron in the valence band if the energy of the light is bigger than the band gap. The excited electron goes to the conduction band and becomes a conduction electron. On the other hand, a hole is generated at the valence band, instead of the excited electron. **The electric field drifts the electron-hole pair** and the electron goes to the n-doped GaAs whereas the hole goes to the p-doped GaAs as the result (Figure 4.3.4.1 (b)). As long as the junction is illuminated, the electron-hole pair is generated and constitutes the current (Figure 4.3.4.1 (c)).

Figure 4.3.4.2 shows the band profile and the carrier densities at short circuit. *bandedges.dat*, *density_electon.dat*, and *density_hole.dat* are used to produce this figure.

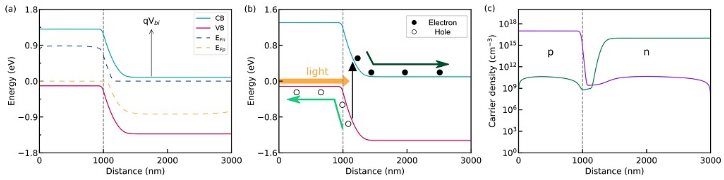


Figure 4.3.4.2: The band profiles are plotted in (a). When the light reaches the junction and the energy is bigger than the band gap, the electron-hole pair is generated as shown in (b). The carrier densities are plotted in (c). The hole density is shown in violet, whereas the electron density is in green.

In (a), CB and VB represent conduction and valence band, respectively. E_{Fn} and E_{Fp} are the electron quasi Fermi level and the hole quasi Fermi level. The results are in a good agreement with Fig. 6.8. in *[NelsonPSC2003]*.

As you can see from Figure 4.3.4.2 (a) and (b), the built-in-potential V_{bi} is formed across the junction and the carriers generated by the illuminated light are drifted by the electric field. In addition, the quasi Fermi levels are split since the carriers are drifted and the carrier densities n and p increase above their equilibrium values.

In the short circuit, the photocurrent density J_{ph} is called the short-circuit current density J_{sc}. **The short-circuit current density is the maximum current density that the solar cell can produce**.

The Photovoltatic effect

When the circuit is connected to a resistive load, **the negative charges accumulated at n-doped GaAs and the positive charges accumulated at p-doped GaAs form a voltage (photovoltage)**. The current flows through the diode due to the voltage and is analogy to the current which flows across the diode under applied bias in the dark. Therefore, this current is called the dark current. The dark current density (J_{dark}) is in the opposite direction to the photocurrent density (J_{ph}) as shown in Figure 4.3.4.3.

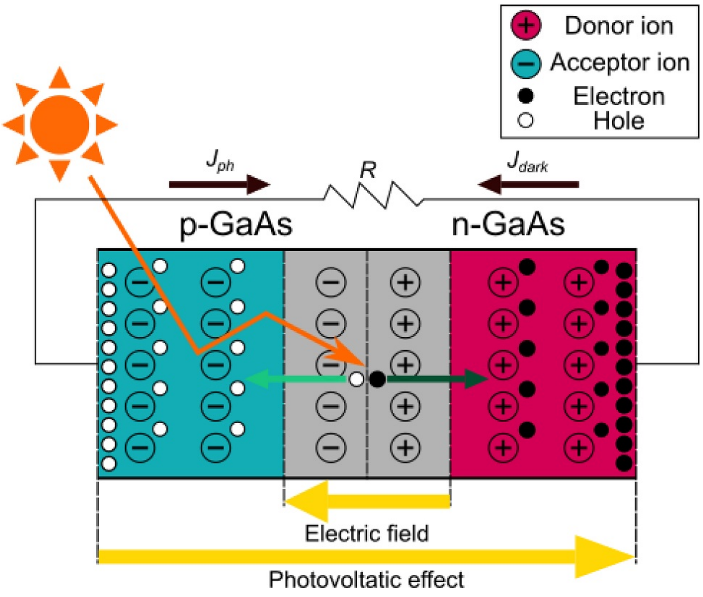


Figure 4.3.4.3: The circuit is connected to a resistive load. Note that J_{dark} flows in the opposite direction to J_{ph}.

Generally speaking, when the photovoltage V is across the diode, the current density J through the diode can be expressed with the **superposition approximation** as below.

$$J(V) = J_{dark}(V) - J_{ph}(V) = J_{dark}(V) - J_{sc}, \tag{4.3.4.1}$$

The photovoltage V is defied so that the forward bias is applied to the diode, where $V > 0$. In the superposition approximation, the photocurrent density is independent of the applied voltage ($J_{ph}(V) = J_{sc}$). Note that we do not take into account the intensity of the irradiated light and the temperature of the diode here for the sake of simplicity. The effects will be explained in the last section of this tutorial.

$J_{dark}(V)$ can be expressed by the equation described in *— NEW/EDU — p-n junction in the dark*.

$$J_{dark}(V) = J_{m,0}(\exp(qV/mk_BT) - 1) \tag{4.3.4.2}$$

where m is the ideality factor and $J_{m,0}$ is a constant. The recombination current density J_{scr} is dominant and $J_{m,0}$ becomes $J_{scr,0}$ when $m = 2$. On the other hand, the diffusion current density J_{diff} is much bigger than J_{scr} and $J_{m,0}$ becomes $J_{diff,0}$ when $m = 1$.

As a result,

$$J(V) = J_{m,0}(\exp(qV/mk_BT) - 1) - J_{sc} \tag{4.3.4.3}$$

Now, let us look into the situation $V = 0.5\ (V)$. In *nextnano*++, we can set the situation by applying forward bias externally to the diode. Thus, the applied bias is equivalent to the photovoltage across the diode. Figure 4.3.4.4 shows the band profiles and the carrier densities of the diode under $0.5\ (V)$.

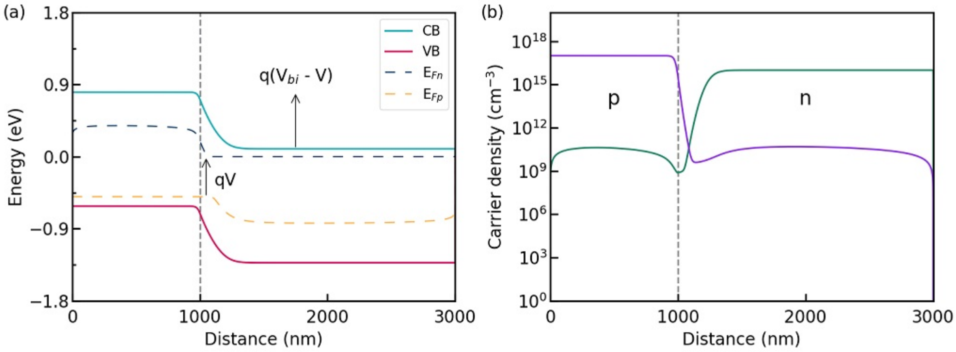


Figure 4.3.4.4: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

The results are very similar to Fig. 6.8. in *[NelsonPSC2003]*. As in the diode with forward bias, the built-in-potential is reduced to $V_{bi} - V$. Applying the bias splits the quasi Fermi levels within the space charge region and the difference of the quasi Fermi levels is equivalent to qV as shown in Figure 4.3.4.4 (a).

Open circuit

When the circuit is open (**open circuit**), the photovoltage V across the diode is called the open-circuit voltage V_{oc}.

Since $J(V) = 0$ in (4.3.4.3) in this case,

$$V_{oc} = \frac{mk_BT}{q} \ln\left(\frac{J_{sc}}{J_0} + 1\right) \tag{4.3.4.4}$$

The open-circuit voltage is the maximum voltage that the solar cell can produce.

J-V curve

We look into the output characteristics of the solar cell in this section. Figure 4.3.4.6 shows J-V curves of the solar cell under the illumination and under the dark condition.

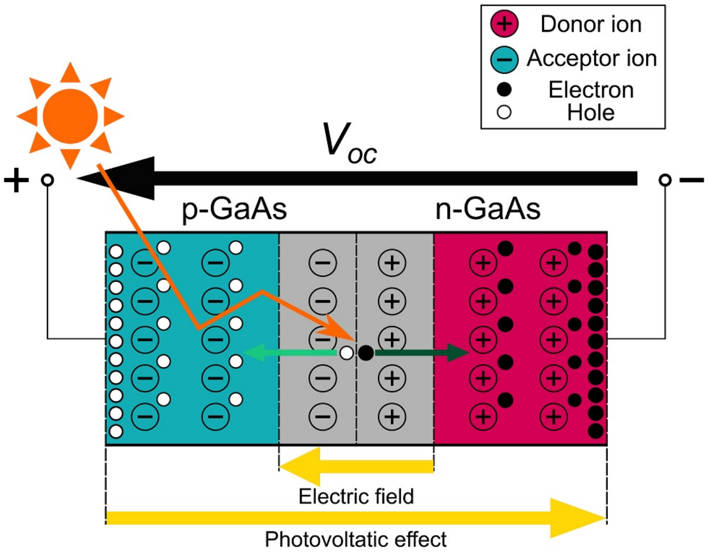


Figure 4.3.4.5: The circuit is open and the voltage V_{oc} is applied to the diode.

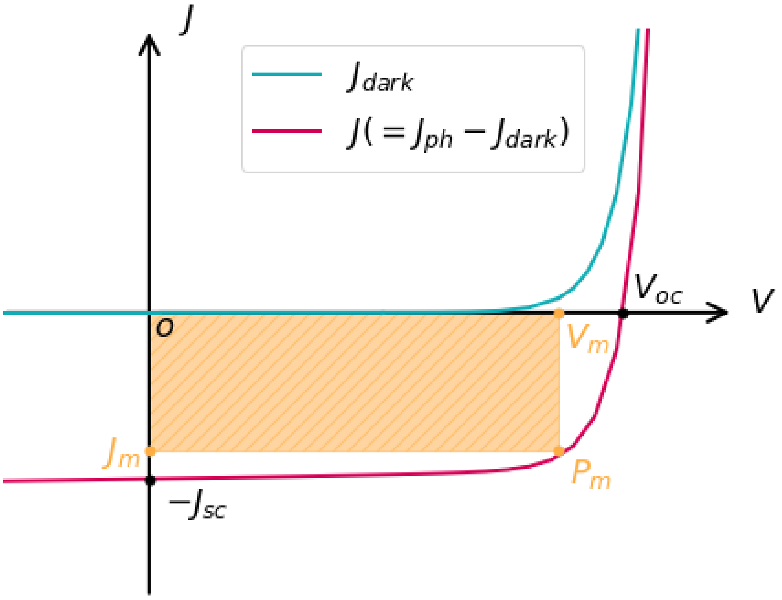


Figure 4.3.4.6: The J-V curves of the solar cell. The J-V curve under the illumination is shown in violet, whereas the J-V curve under the dark condition is in light-blue. The orange-filled area indicates the output of the maximum power density of the solar cell.

Again, the maximum current density that the solar cell can produce is the short-circuit current density, and the maximum voltage of the cell is the open-circuit voltage. However, the output of the maximum power density P_m is not equal to the product of them. It is represented by the intersection of J_m and V_m.

This arises from the parasitic resistances which are connected in series and parallel to the solar cell. The series resistance consist of the electrical resistance present on the carrier transport path, such as the semiconductors and the contacts of the solar cell. The parallel resistance is attributed to leakage of the current due to defects in the solar cell.

We can derive P_m using the equations described in the sections above.

First, the power density of the solar cell is given by

$$P = JV = J_0 V(\exp(qV/mk_B T) - 1) - J_{sc} V \tag{4.3.4.5}$$

The condition for the maximum power density is achieved when $dP/dV = 0$.

Thus,

$$V_m = \frac{1}{\beta} \ln\left[(J_{sc}/J_0) + 1\right] - \frac{1}{\beta} \ln\left(1 + \beta V_m\right) = V_{oc} - \frac{1}{\beta} \ln\left(1 + \beta V_m\right), \tag{4.3.4.6}$$

$$J_m = J_0 \beta V_m \exp(\beta V_m), \tag{4.3.4.7}$$

where $\beta = q/k_B T$.

Therefore,

$$P_m = J_m V_m = J_0 \beta V_m^2 \exp(\beta V_m) = FF J_{sc} V_{oc}, \tag{4.3.4.8}$$

where FF is called the fill factor and the ratio to measure the sharpness of the J-V curve.

$$FF = \frac{J_m V_m}{J_{sc} V_{oc}} \tag{4.3.4.9}$$

In addition, the energy conversion efficiency of the solar cell (η) is derived by dividing P_m by the incident power of the sun P_{in}.

$$\eta = \frac{P_m}{P_{in}} = \frac{I_m V_m}{P_{in}} = \frac{J_{sc} V_{oc} FF}{P_{in}} \tag{4.3.4.10}$$

Effects of irradiation intensity and temperature

So far, the effects of the intensity of the incident light and temperature of the system on the behavior of the solar cell have not been considered. In this section, we briefy investigate the effects on J-V characteristics.

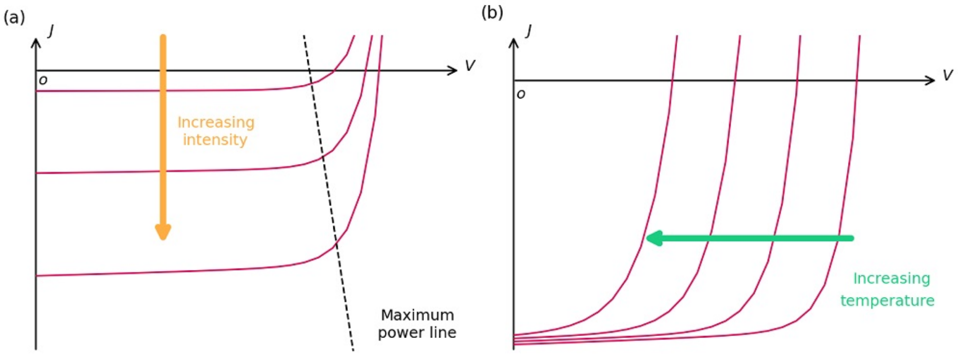


Figure 4.3.4.7: The J-V curves of the solar cell under incident light of various intensities is shown in (a). The J-V curves of the solar cell under different temperatures are shown in (b). The arrows indicate the direction of increasing intensity of the incident sunlight or the temperature.

Effect of irradiation intensity

Figure 4.3.4.7 (a) illustrates the effect of the light intensity on the J-V curve. Since the generation rate for electron-hole pairs is proportional to the light intensity, the photocurrent increases as the light intensity gets bigger. From (4.3.4.4), V_{oc} also increases logarithmically with the irradiation intensity. Thus, the more intensive light enables to obtain a bigger output of the maximum power density. However, increasing the light intensity is not always good as the light also raises the temperature and increases the series resistance of the solar cell, which degrade the cell performance.

Effect of temperature

The effect of the temperature on the J-V curve is shown in (b) in Figure 4.3.4.7. As the temperature is increased, the intrinsic carrier density n_i increases exponentially. When the diffusion current is dominant in the dark current across the solar cell, J_0 becomes $J_{diff,0}$. Thus, $J_{diff,0}$ is also increased as n_i increased. According to (4.3.4.4), V_{oc} decreases logarithmically with increasing $J_{diff,0}$ under a given J_{sc}.

This occurs more noticeably than when the recombination current is dominant ($J_0 = J_{scr,0}$) since $J_{diff,0}$ is proportional to n_i^2, whereas $J_{scr,0}$ is proportional to n_i. The detailed equations of $J_{diff,0}$ and $J_{scr,0}$ are in *— NEW/EDU — p-n junction in the dark*.

Actually, J_{sc} becomes larger since the band gap is reduced as the temperature increases and lower energy photons can be absorbed. However, the gain in $J_{diff,0}$ is more significant than the gain in J_{sc}, which eventually leads to the decrease of V_{oc}. Therefore, increasing temperature reduces the performance of the solar cell.

Exercises

under construction

Last update: 16/07/2024

4.3.5 GaAs solar cell

Header

Files for the tutorial located in *nextnano++\examples*

- *1DGaAs_SolarCell_nnp.in*
- *1DGaAs_SolarCell_nnp_import_generation.in*

- *1DGaAs_SolarCell_nnp_local_absorption.in*
- *1DGaAs_SolarCell_nnp_complex_refractive_index.in*

Here we demonstrate that solar cells can be simulated using nextnano. The self-consistent solutions to the Poisson equation coupled with current (drift-diffusion) equation give the figure of merit of solar cells that consists of arbitrary materials. Current-Voltage (I-V) curves and corresponding power and solar cell efficiency as a function of bias voltage are exported to the output folder.

Input files

Here the numerics parameters are optimized for convergence of the calculation in the bias range of interest. Please pay attention to the convergence of the calculation when you change device geometry etc.

In the simulation of input files *1DGaAs_SolarCell_nnp.in* and *1DGaAs_SolarCell_nn3.in*, the following data are used to calculate generation rate $G(E, x)$ internally:

- Absorption spectrum $\alpha(E)$
- Reflectivity $R(E)$
- Solar spectral irradiance

In *1DGaAs_SolarCell_nnp.in* (*nextnano*++), these data are already specified in *database_optional.in* for some materials. For example, you can use these by specifying `irradiation{}` as follows:

```
classical{
    ...

    irradiation{
        ...

        global_illumination{
            direction_x = 1

            database_spectrum{
                name = "Solar-ASTM-G173-global"
                concentration = 1.0
            }
        }

        global_reflectivity{
            database_spectrum{
                name = "Al0.80Ga0.20As"
            }
        }

        global_absorption_coeff{
            database_spectrum{
                name = "GaAs"
            }
        }
    }
}
```

If you want to use the materials that are not in the database or rewrite the database, you can specify the new data in `database{ }` as you want.

You can also import the data of generation rate itself. In the simulation *1DGaAs_SolarCell_nnp_import_generation.in*, the following file must be read in.

- */optics/GenerationRateLight_vs_Position_sun1.dat*

This data file is also in the sample file folder.

Reference

- J. Nelson, *The Physics of Solar Cells* (Imperial College Press, 2003)
- S.M. Sze and Kwok K. Ng, *Physics of Semiconductor Devices* (Wiley, 2007)

Structure

Figure 4.3.5.1 shows the band edges and quasi Fermi levels of the device. The device structure is as follows:

- 0-30 nm $\mathrm{Al_{0.8}Ga_{0.2}As}$ Window layer
- 30-530 nm p-doped GaAs
- 530-3530 nm n-doped GaAs
- 3530-3630 nm n-doped GaAs back surface field layer

Strain is not calculated in this example.

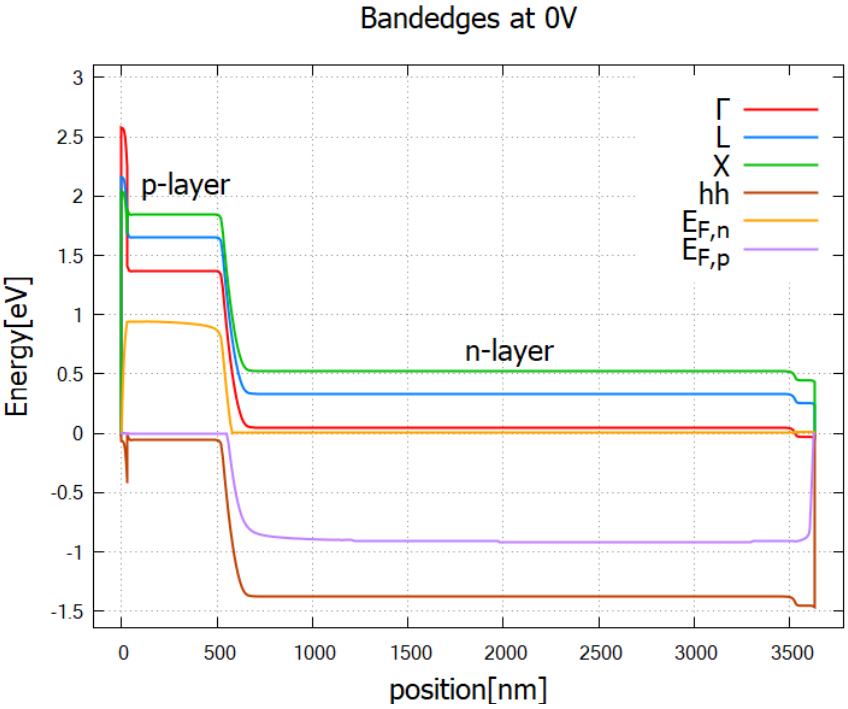


Figure 4.3.5.1: Band edges and quasi-Fermi levels of the solar cell at zero bias *bias_000000/bandedges.dat*

The left side of the device (x=0 nm) is illuminated by the sun. As shown in Figure 4.3.5.6, mobile electrons and holes are created mainly in the p-layer. Electrons then flow to the right because of the AlGaAs ternary barrier (0-30 nm), and holes to the left. The back of the cell (3530-3630 nm) is doped with 10 times larger concentration, so that it prevents the minority carrier (hole) from leaking to the right contact. Since the current from p-layer to n-layer is defined to be positive, the photo-induced current has negative sign.

Simulation procedure

The workflow of the simulation is summarized in Figure 4.3.5.2. To obtain the figures shown in this tutorial,

1. Specify in the input file the three data, namely (1) spectral irradiance (solar spectrum), (2) reflectivity at the front surface and (3) absorption spectrum. (Referring the database or rewriting the database)

2. Run *nextnano*++, and all of your *nextnano*++ results are in your output folder! Generation rate $G(E, x)$ is internally calculated before the current-Poisson iteration starts. The efficiency-voltage curve is generated as a final result.

3. If you already have generation rate profile as a *.dat* file, you can import it.

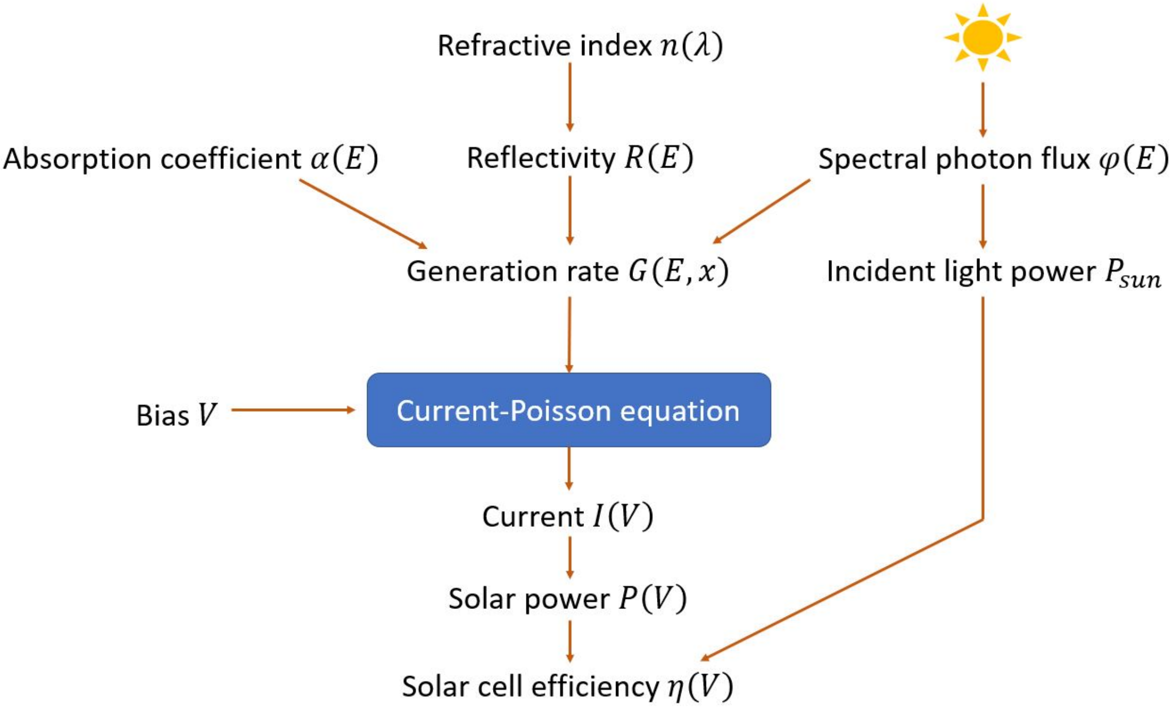


Figure 4.3.5.2: Workflow of solar cell simulation. Each quantity is explained in the following section.

How does a solar cell work? & How do we simulate it?

1. Solar spectrum

The sun emits light with a range of wavelengths ranging from the ultraviolet, visible to infrared region. The extraterrestrial solar spectrum resembles the spectrum of a black body at $T_{\text{sun}} = 5760\text{K}$ [Nelson Chapter 2]:

$$\frac{2\pi \sin^2 \theta_{\text{sun}}}{h^3 c^2} \frac{E^2}{e^{E/k_{\text{B}} T_{\text{sun}}} - 1},$$

where E is the photon energy and $\theta_{\text{sun}} = 1.44 \times 10^{-3}\pi[\text{rad}]$ when measured from the earth. The solar light travels from the sun to the earth, and then from the outer space to our solar cell devices, during which the spectrum attenuates and changes its shape. The standard solar spectrum assumed in solar cell analysis is called AM1.5G (AM = air mass), which takes into account the attenuation of the intensity and illumination from all angles (rather than direct from the sun) due to scattering in the atmosphere. The spectral photon flux, i.e. the spectrum of the number of incident photons per area per time, is denoted by $\phi(E)$ $[\text{m}^{-2}\text{s}^{-1}\text{eV}^{-1}]$. The spectral irradiance, namely the spectrum of the amount of energy supplied per area per time, is given by $L(E) = E\phi(E)$ with the unit of $[\text{Wm}^{-2}\text{eV}^{-1}]$. We have taken the AM1.5G spectral irradiance data from this website (Figure 4.3.5.3). If you have space applications in mind, please use the extraterrestrial spectrum, namely air mass zero (AM0).

The power of incident light

$$P_{\text{sun}} = \int_0^\infty L(E)\text{d}E = 1000\ \text{Wm}^{-2},$$

is solely determined by the condition of the sun and the atmosphere of the earth (for AM0 $P_{\text{sun}} = 1353\ \text{Wm}^{-2}$). The ultimate challenge of solar cell research is to achieve the most efficient conversion of this energy input into electric power P_{out} $[\text{Wm}^{-2}]$. The figure of merit is therefore defined as $\eta = \frac{P_{\text{out}}}{P_{\text{sun}}}$.

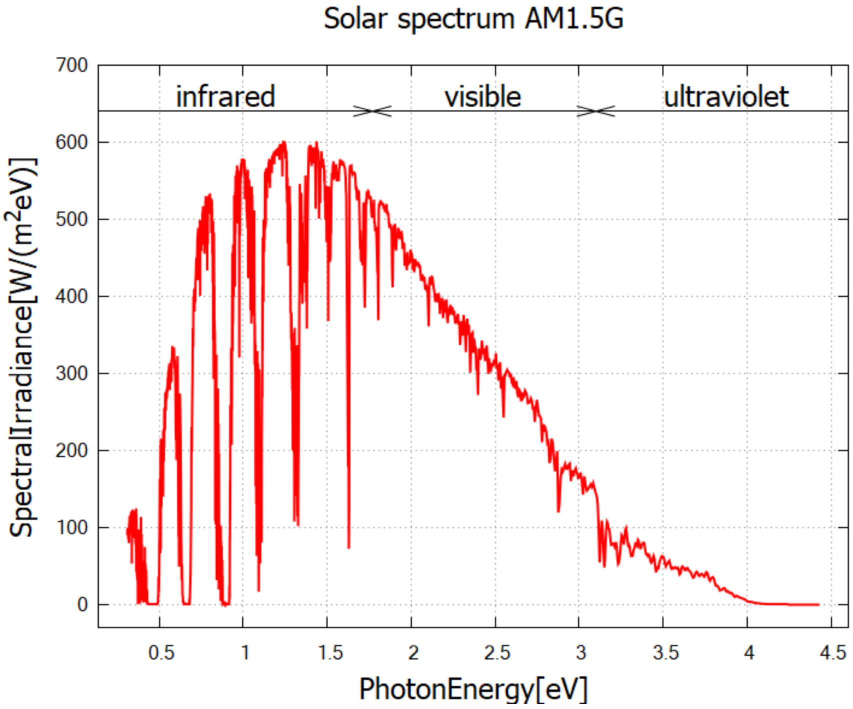


Figure 4.3.5.3: The AM1.5G spectral irradiance $L(E)$, that is, the solar spectrum measured on the earth. The *nextnano++* tool reads from the predifined data `Solar-ASTM-G173-global` and stores it in the output file *Irradiation/illumination_spectrum_power_eV.dat*.

1. Generation rate (internal calculation)

(*If you already have available data for generation rate, you can skip this section.*)

When the sunlight illuminates the device, some photons are reflected at the front surface (air-semiconductor interface) and the rest enters the device. This effect is taken into account by considering the reflectivity of $Al_{0.8}Ga_{0.2}As$. Through the absorption of one photon, a pair of mobile electron and hole is created, while the photon flux attenuates exponentially with respect to the penetration depth. The generation rate thus depends not only on the incident photon flux $\phi(E)$ but also on the absorption coefficient $\alpha(E)$ of the material and the reflectivity $R(E)$ at the surface (Figure 4.3.5.2):

$$G(E,x) = N\phi(E) \cdot (1 - R(E)) \cdot \alpha(E)e^{-\alpha(E)x},$$

where "the number of suns" N is multiplied to the photon flux $\phi(E)$ to take into account the concentration of sunlight. The corresponding keyword is `concentration`.

In the sample input file for *nextnano++*, predefined value is used for $L(\lambda)$. $\alpha(\lambda)$ and $R(\lambda)$ are defined in `database{ }`. In the group `optics{ }`, one can specify which data to use as those variables. These spectra are translated into $\alpha(E)$, $R(E)$, $N\phi(E)$ and substituted into the generation rate formula.

```
# nextnano++
optics{
    ...

    global_illumination{
        direction_x = 1

        database_spectrum{
            name = "Solar-ASTM-G173-global"
            concentration = 1.0             # e.g. 1 sun
        }
```

(continues on next page)

(continued from previous page)

```
    }

    global_reflectivity{
        database_spectrum{
            name = "Al0.80Ga0.20As"
        }
    }

    global_absorption_coeff{
        database_spectrum{
            name = "GaAs"
        }
    }
}
```

If no reflectivity data is specified, perfect interface (zero reflection) is assumed. The reflectivity data of $\mathrm{Al_{0.8}Ga_{0.2}As}$ used in this simulation (`optical_reflectivity` in *database_nnp_optional.in*) have been generated through the Fresnel formula for perpendicular incident light

$$R(\lambda) = |r(\lambda)|^2 = \left| \frac{1 - [n(\lambda) + i\kappa(\lambda)]}{1 + [n(\lambda) + i\kappa(\lambda)]} \right|^2$$

where the refractive index n and extinction coefficient κ of GaAs and AlAs are taken from here. To obtain the values of ternary $\mathrm{Al_{0.8}Ga_{0.2}As}$, we performed linear interpolation.

If you consider a textured surface to reduce surface light reflection, please prepare the corresponding reflectivity data and import to the nextnano simulation. For *nextnano++*, $\alpha(E), R(E)$ and $L(E)$ are stored in the output folder `Irradiation` with file names *absorption_spectrum_eV.dat*, *reflectivity_spectrum_eV.dat* and *illumination_spectrum_eV.dat*, respectively.

The resulting generation rate is shown in Figure 4.3.5.4, Figure 4.3.5.5 and Figure 4.3.5.6.

1. Generation rate (import)

If the generation rate data $G(x) = \int G(E,x)\mathrm{d}E$ (Figure 4.3.5.6) is available from literature or publications, you can import the *.dat* file without worrying about the above-mentioned calculation. The data must contain position $[\mathrm{nm}]$ in the first column and generation rate $[10^{18}\mathrm{cm}^{-3}\mathrm{s}^{-1}]$ in the second. In the sample file *1DGaAs_SolarCell_nnp_import_generation.in*, we import the data generated elsewhere.

```
# nextnano++
structure{
    region{
        everywhere{}
        generation{
            import{ import_from = "GenImportProfile" }
        }
    }
}

import{
    file{
        name     = "GenImportProfile"
        filename = "(directory path)\GenerationRateLight_vs_Position_sun1.dat"
        format   = DAT
        scale    = 1e18  # import data is multiplied by this scaling factor (optional,
↪ default value is 1.0)
    }
}
```

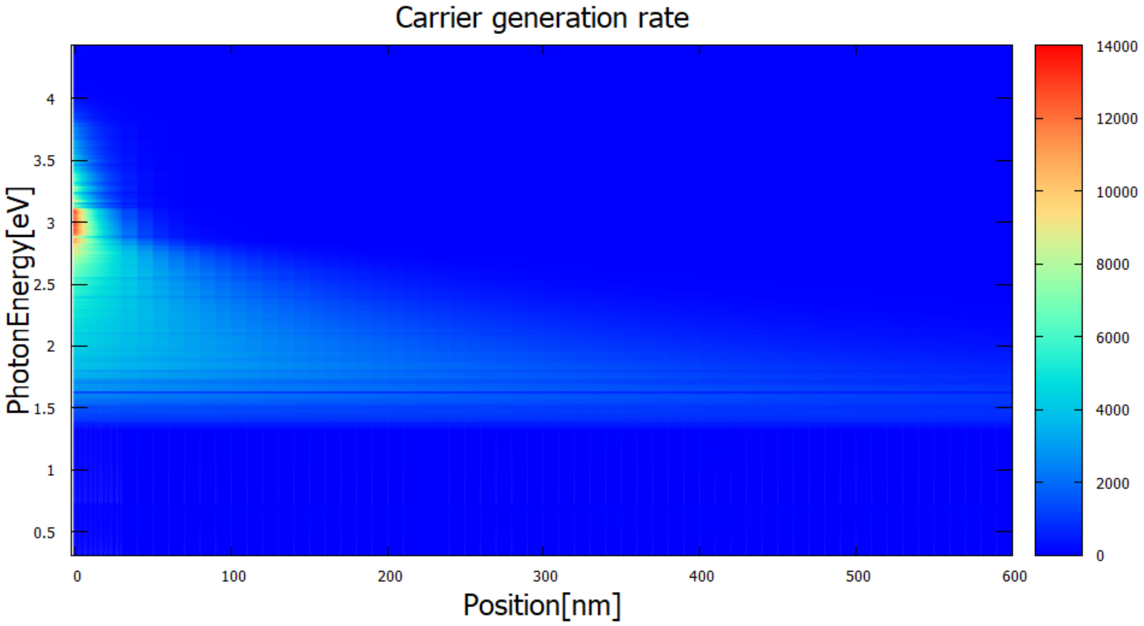


Figure 4.3.5.4: Generation rate as a function of position and energy *Irradiation/photo_generation_energy_resolved.fld* in units of $10^{18}\mathrm{cm}^{-3}\mathrm{eV}^{-1}\mathrm{s}^{-1}$. This quantity is internally calculated using the absorption coefficient, reflectivity of the front surface and solar spectrum AM1.5G (Figure 4.3.5.3). Photons at around 3V are largely absorbed near the front surface due to a large absorption coefficient, which can be seen in the output *optics/AbsorptionCoefficient_eV.dat*/*Irradiation/absorption_spectrum_eV.dat* (not shown). Photons with lower energy, in contrast, travel a longer distance in the device.

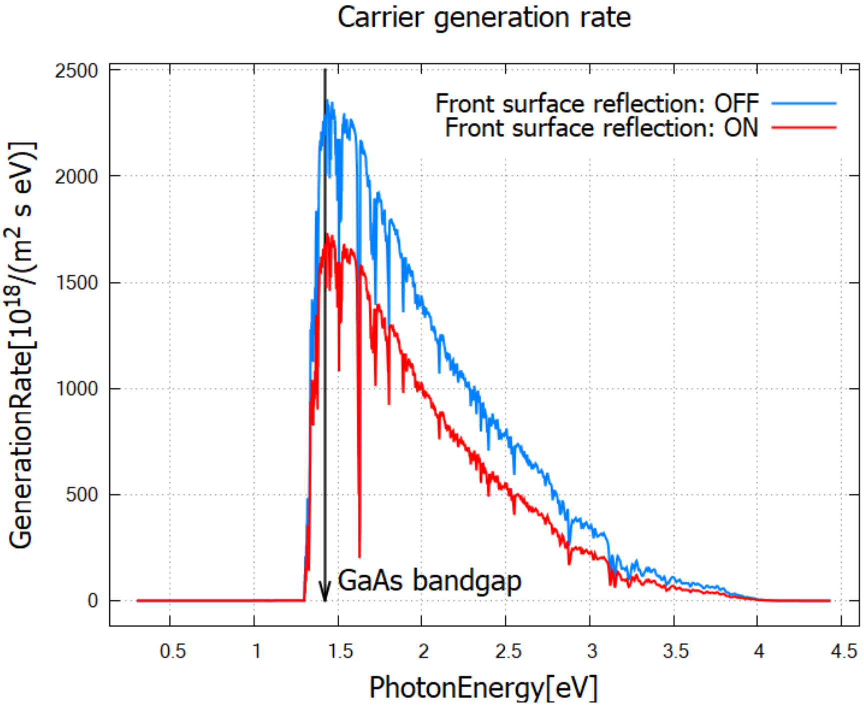


Figure 4.3.5.5: Generation rate as a function of energy *Irradiation/photo_generation_integrated.dat*. Obviously, the generation rate becomes larger when the reflection at the front surface is neglected. One can also clearly see, by comparing with Figure 4.3.5.4, that the low e nergy photons below the band gap cannot contribute to the carrier generation. For this reason the band gap of semiconductors affects the solar cell efficiency and is discussed in the context of the Shockley-Queisser efficiency limit.

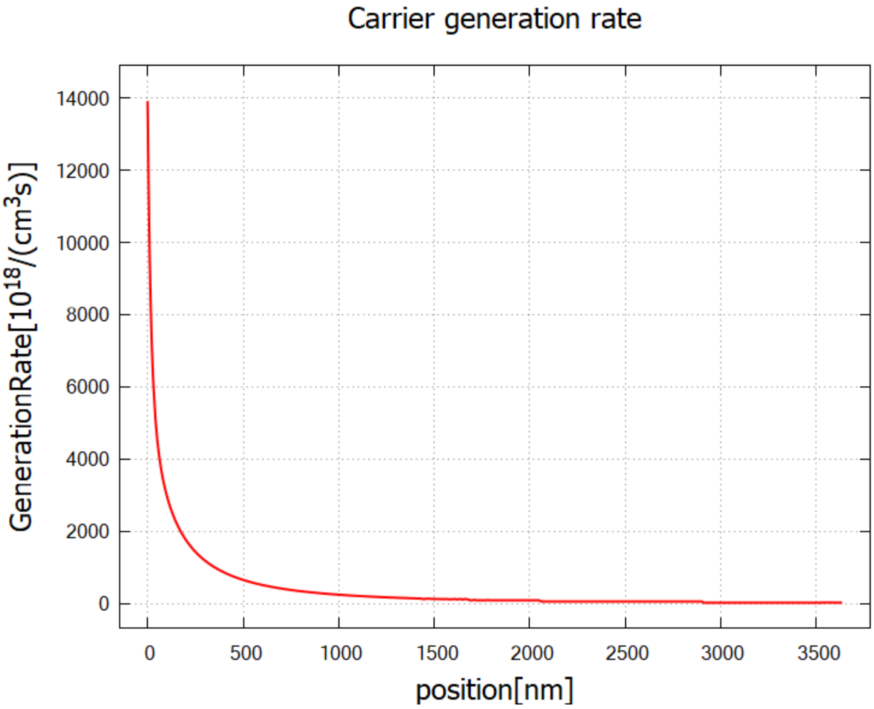


Figure 4.3.5.6: Generation rate as a function of position *Irradiation/photogeneration.dat*. This data is obtained by integrating Figure 4.3.5.4 over energy E. When the photon flux travels through the device, the intensity diminishes exponentially, leading to the exponential decrease in generation rate. Most of the carrier generation occurs within 500 nm from the front surface, i.e. within the p-layer (30–530 nm).

4. Current-Voltage characteristics

The calculated or imported generation rate contributes to the right-hand side of the coupled current equations for electrons and holes,

$$-e\frac{\partial n}{\partial t} + \nabla \cdot \mathbf{j}_n = -e(G-R),$$
$$e\frac{\partial p}{\partial t} + \nabla \cdot \mathbf{j}_p = e(G-R),$$

where G and R are the (position-dependent) generation and recombination rates for electron-hole pairs. Here the charge current density $\mathbf{j}_{n,p}$ has a dimension of (charge)(area)$^{-1}$(time)$^{-1}$ and the generation rate has (volume)$^{-1}$(time)$^{-1}$. The recombination rate is the sum of three different processes $R = R_{\mathrm{rad}} + R_{\mathrm{Auger}} + R_{\mathrm{SRH}}$. See our *Laser diode tutorial*, [Nelson] or other literature for details.

By solving this current equation and the Poisson equation self-consistently, the program obtains the current density at each bias step. The resulting I-V curve is shown in Figure 4.3.5.7. For comparison, the dark current has been simulated by setting

```
# nextnano++
structure{
    region{
        generation{
            constant{ rate = 0.0 }
        }
    }
}
```

The dark current in the present device behaves like in a diode under forward bias. When the sun illuminates the device, electrons and holes are created and current flows in the reverse direction.

If you change the device geometry or materials and the I-V curve is no longer reasonable, it is likely that the numerical calculation did not converge. Please check the *.log* file. For the convergence of the current-Poisson equation, you might need to change the settings under *run{ }*.

5. Solar efficiency

From the I-V curve the solar cell power density $P_{\mathrm{out}} = -IV$ and the efficiency $\eta = \frac{P_{\mathrm{out}}}{P_{\mathrm{sun}}}$ are calculated. For the present device under 1 sun, the maximum efficiency of **15.8%** is achieved at the bias 0.9 V (Figure 4.3.5.8 red). The theoretical limit for GaAs (band gap 1.42 eV at $T = 300\ K$) is around 30% under the AM1.5 condition without concentration [Sze].

The maximum efficiency of the present device increases to **21.6% (nextnano++) for 100-sun concentration**, mainly due to the increase in open circuit voltage (Figure 4.3.5.8 blue). This means one cell operating under 100 suns can produce the same power output as $\frac{100P_{sun}\times 0.216}{P_{sun}\times 0.158} = 137$ cells under 1 sun. Optical concentration reduces the total cost of solar cells since concentrator materials are usually less expensive than the ones for solar cells [Sze].

The *.log* file contains additional properties of the solar cell.

The convergence of the simulation is sensitive to the device settings such as the number of suns. If the convergence fails in your original device, please consider changing the settings in run{ }.

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

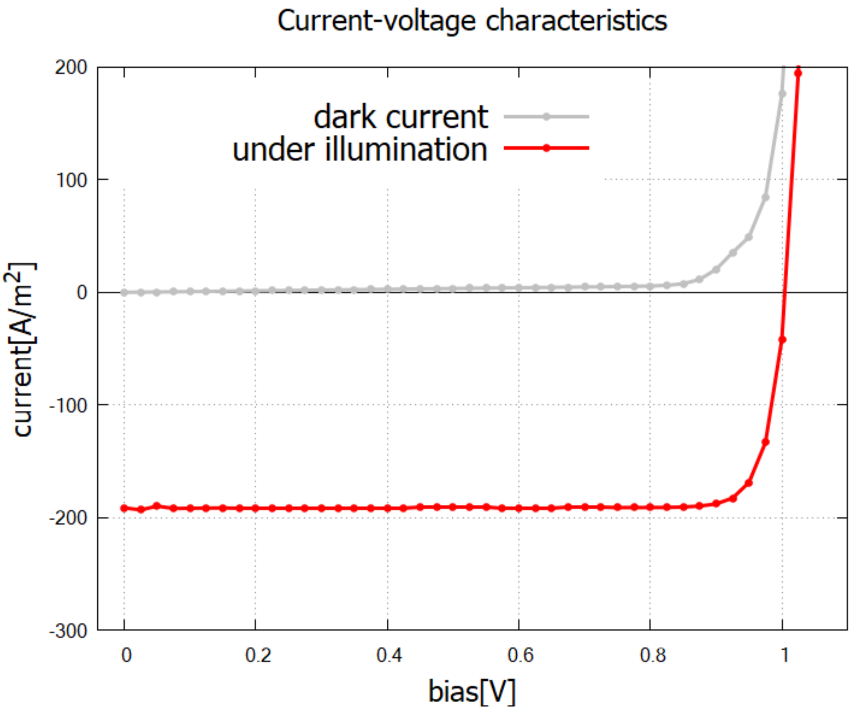


Figure 4.3.5.7: I-V characteristics of the solar cell *IV_characteristics.dat* (*nextnano++*). In the bias regime 0-1 V the system works as a solar cell.

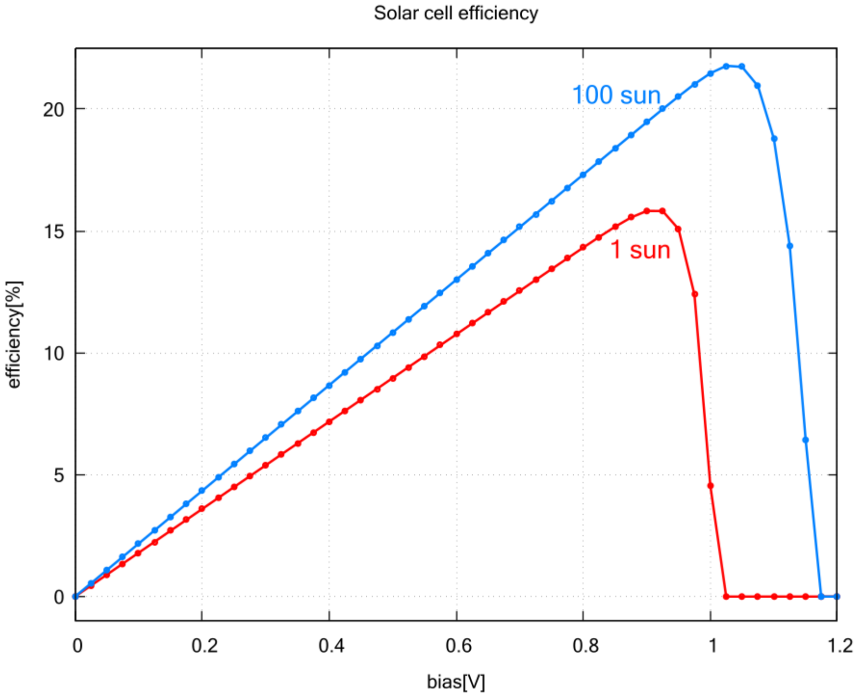


Figure 4.3.5.8: Solar cell efficiency η for no sunlight concentration (red) and 100-sun concentration (blue) by *nextnano++*. The data is contained in *solar_cell_efficiency.dat*.

4.3.6 Cascade solar cell (Tandem solar cell)

Input Files:

- *1DCascadeSolarCell_nnp.in*

In this tutorial, we solve the Poisson equation in an AlGaAs/InGaAs monolithic cascade solar cell (tandem solar cell).

The layout is based on US patent 4179702 (1979): Cascade solar cells by Michael F. Lamorte.

See also the following publication for more details

Computer Modeling of a Two-Junction, Monolithic Cascade Solar Cell
M.F. Lamorte, D.H. Abbott
IEEE Transactions on Electron Devices 27 (1), 231 (1980)

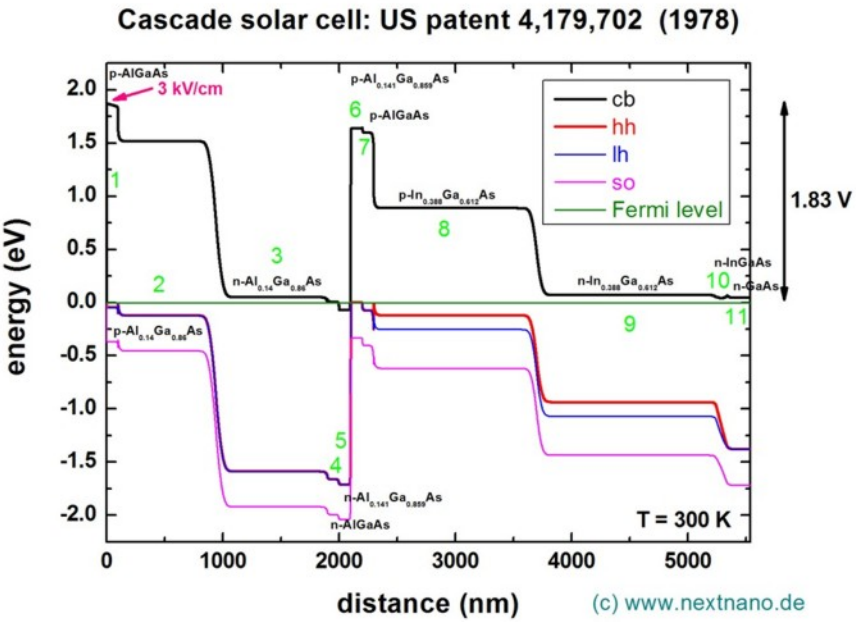
Input files used in this tutorial are followings:

Outputs

Band profile

The following figure shows the conduction band edge and the valence band edges (heavy hole, light hole and split-off hole) of this solar cell at zero bias. The built-in potential has been calculated to be 1.83 V.

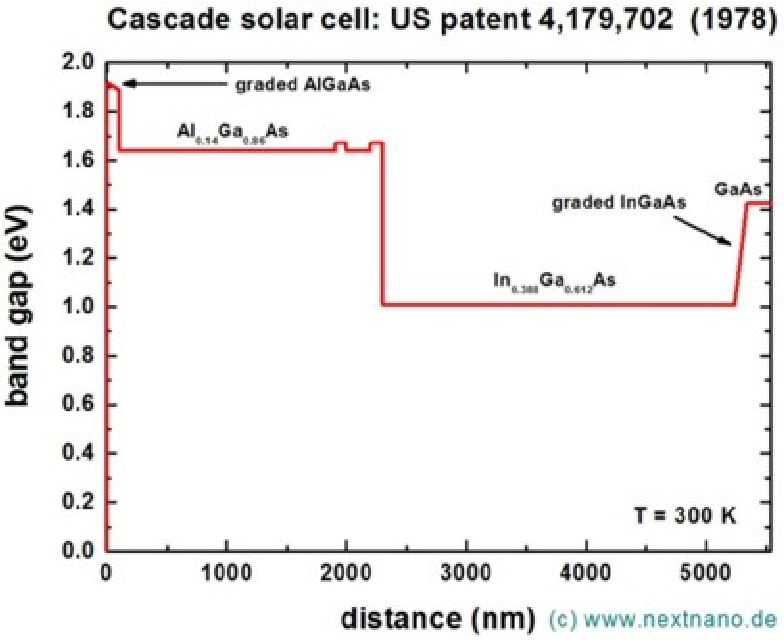
On the left side (region 1), a graded p-type AlGaAs layer has been used to generate an electric field of 3 kV / cm (= 30 meV / 100 nm). We assumed that all materials are strained with respect to the GaAs substrate, thus the degeneracy of heavy and light hole valence band edges is lifted, especially inside the InGaAs regions.



Band gap

The band gap as a function of distance is shown in the following figure. This data can be found in these files. For *nextnano++*, we need to add *classical{ output_bandgap{} }* in the sample file.

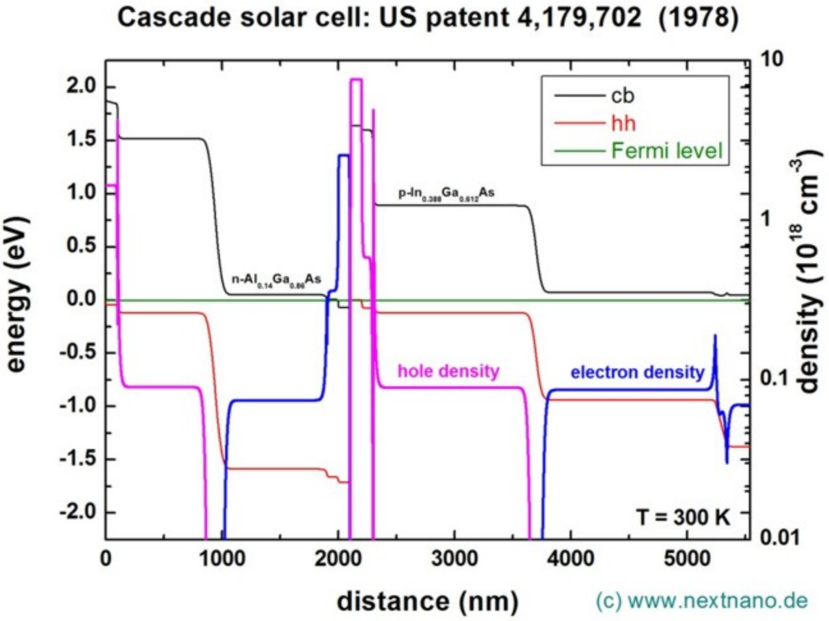
- *bias_00000/bandgap.dat* (*nextnano++*)



Electron and hole densities

Here, the electron and hole densities are plotted. This data can be found in these files.

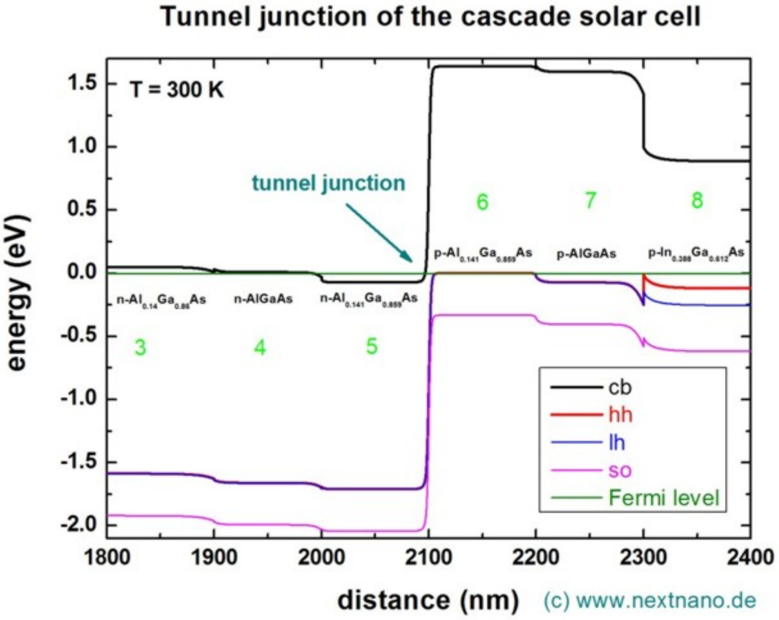
- *bias_00000/density_electron.dat*, *bias_00000/dentity_hole.dat* (*nextnano++*)

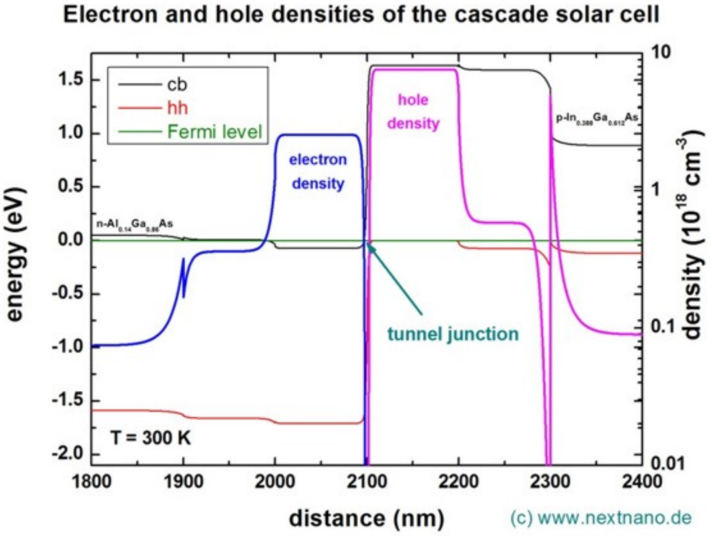


Tunnel junction

The area around the tunnel junction which is in the middle of the device at ~2100 nm is shown in this plot:

The electron and hole densities in the vicinity of the tunnel junction are shown in this graph. Note that the density has been calculated classically (without solving the Schrödinger equation, i.e. without quantum mechanics).


Tunnel junction of the cascade solar cell
T = 300 K
tunnel junction
energy (eV)
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
1800
1900
2000
2100
2200
2300
2400
distance (nm)
(c) www.nextnano.de
3
4
5
6
7
8
n-Al$_{0.14}$Ga$_{0.86}$As
n-AlGaAs
n-Al$_{0.141}$Ga$_{0.859}$As
p-Al$_{0.141}$Ga$_{0.859}$As
p-AlGaAs
p-In$_{0.388}$Ga$_{0.612}$As
cb
hh
lh
so
Fermi level


Electron and hole densities of the cascade solar cell
cb
hh
Fermi level
electron density
hole density
tunnel junction
T = 300 K
n-Al$_{0.14}$Ga$_{0.86}$As
p-In$_{0.388}$Ga$_{0.612}$As
energy (eV)
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
density (10^{18} cm^{-3})
10
1
0.1
0.01
1800
1900
2000
2100
2200
2300
2400
distance (nm)
(c) www.nextnano.de

What we can do on a solar cell using nextnano

We have the demonstration of the simulation for GaAs solar cell using nextnano here: *GaAs Solar Cell*.

As we can see in this demonstration, we can calculate the following characteristics by solving the Poisson equation and current equation self-consistently.

- **Current-Voltage characteristics**
  - The dark current can also be calculated.
- **Solar efficiency**
  - We can also see the effect of optical concentration quantitatively.

The data we need to prepare independently for this calculation is:

1. spectral irradiance (solar spectrum)
2. reflectivity at the front surface
3. absorption spectrum

Both *nextnano*++ can calculate the generation rate $G(x)$. We can also import the data of $G(x)$ directly instead of 2 and 3 above.

The links for all the used data is also specified in this tutorial: *GaAs Solar Cell*.

This tutorial also exists for *nextnano*^{3}.

Last update: 2024/06/10

4.4 Light-Emitting Diodes

4.4.1 InGaAs Multi-quantum well laser diode

Header

Files for the tutorial located in *nextnano++\examples*

- *LaserDiode_InGaAs_1D_cl_nnp.in*
- *LaserDiode_InGaAs_1D_qm_nnp.in*

Introduction

In this tutorial, we simulate optical emission of a 1D InGaAs multi-quantum well laser diode grown on InP substrate. The blue region, seen in Figure 4.4.1.1, is the separate confinement heterostructure (SCH), which forms an optical waveguide in the transverse direction to confine the emitted light (red arrow). The multi-quantum wells and SCH are clad by InP on both sides. A voltage bias is applied to the gray edges.

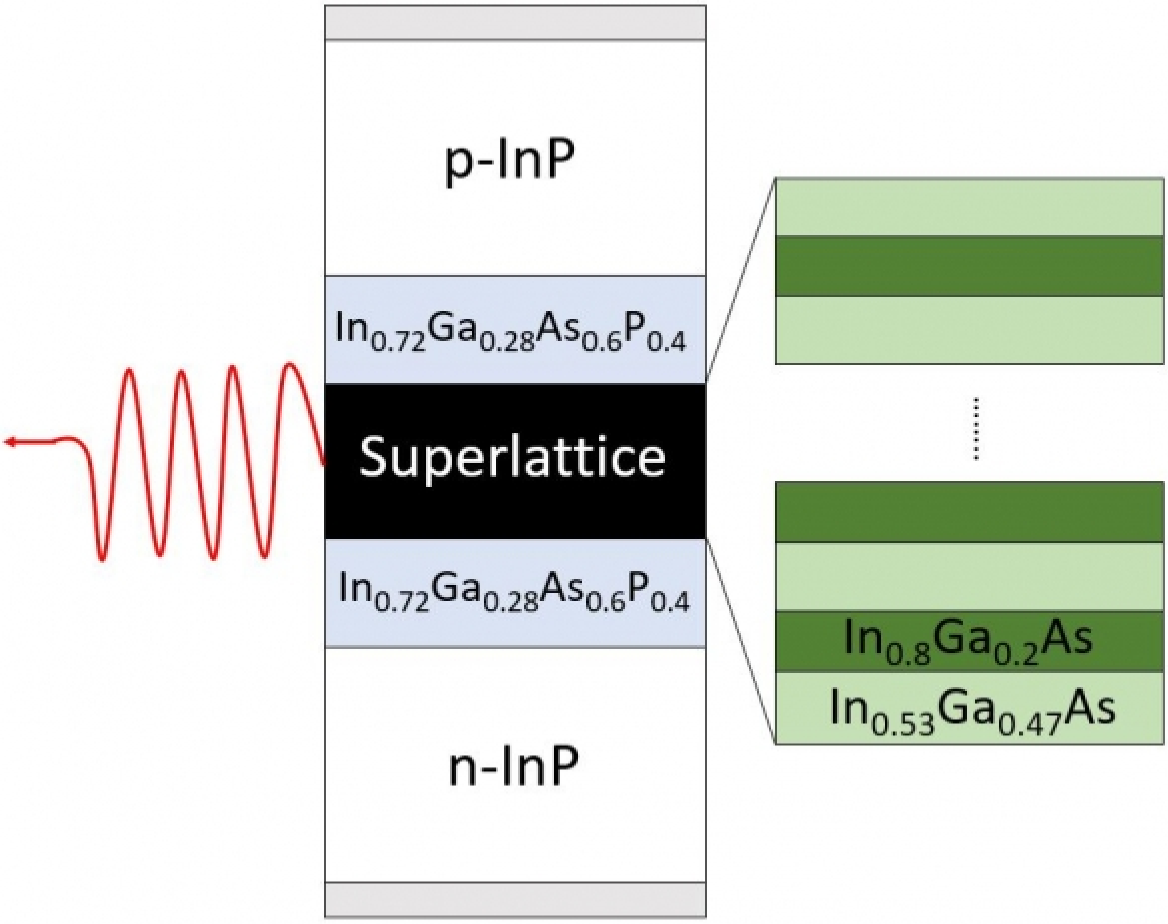


Figure 4.4.1.1: Structure overview

Current equation

The properties of optoelectronic devices are governed by Poisson equation, Schrödinger equation, drift-diffusion and continuity equations. We denote by n and p the carrier number density per unit volume. The continuity equations in the presence of creation (generation, G) or annihilation (recombination, R) of electron-hole pairs read

$$\begin{aligned} -e\frac{\partial n}{\partial t} + \nabla\cdot(-e\mathbf{j}_n(\mathbf{x})) =& -e(G(\mathbf{x})-R(\mathbf{x})), \\ e\frac{\partial p}{\partial t} + \nabla\cdot e\mathbf{j}_p(\mathbf{x}) =& e(G(\mathbf{x})-R(\mathbf{x})), \end{aligned} \tag{4.4.1.1}$$

where the current is proportional to the gradient of quasi Fermi levels $E_{F,n/p}(\mathbf{x})$

$$\begin{aligned} \mathbf{j}_n(\mathbf{x}) &= -\mu_n(\mathbf{x})n(\mathbf{x})\nabla E_{F,n}(\mathbf{x}), \\ \mathbf{j}_p(\mathbf{x}) &= \mu_p(\mathbf{x})p(\mathbf{x})\nabla E_{F,p}(\mathbf{x}). \end{aligned} \tag{4.4.1.2}$$

Here the charge current has the unit of (area)$^{-1}$(time)$^{-1}$. $\mu_{n/p}$ are the mobilities of each carrier. In *nextnano++*, $\mu_{n/p}$ are determined using the mobility model specified in the input file under *currents{ }*. Hereafter we consider stationary solutions and set $\dot{n} = \dot{p} = 0$. The governing equations then reduce to

$$\begin{aligned} \nabla \cdot \mu_n(\mathbf{x}) n(\mathbf{x}) \nabla E_{F,n}(\mathbf{x}) &= -(G(\mathbf{x}) - R(\mathbf{x})), \\ \nabla \cdot \mu_p(\mathbf{x}) p(\mathbf{x}) \nabla E_{F,p}(\mathbf{x}) &= G(\mathbf{x}) - R(\mathbf{x}), \end{aligned} \tag{4.4.1.3}$$

which we call **current equation** (generation $G = 0$ in the present case). The *nextnano*++ tool solves this equation and Poisson equation self-consistently when one specifies it in the input file as:

```
run{
   current_poisson{ }
}
```

Recombination of carriers and emission spectrum

The generation/recombination rate, $R(\mathbf{x})$, originates from several physical processes. In *nextnano*++, the following mechanisms are implemented (cf. *recombination_model{ }*)

- **Schockley-Read-Hall recombination** R_{SRH} – carrier trapping by impurities.
- **Auger recombination** R_{Auger} – a collision between two carriers results in the excitation of one and the recombination of the other with a carrier of opposite charge.
- **radiative recombination** R_{rad} – emission/absorption of a **photon**.

Each mechanism can be turned on and off in the input file.

Radiative recombination describes the recombination of electron-hole pairs at a position $\mathbf{x}$ by emitting a photon and is given by

$$R_{\mathrm{rad}}^{spon}(\mathbf{x}, E) = C(\mathbf{x}) \int dE_h \int dE_e \; n(\mathbf{x}, E_e) p(\mathbf{x}, E_h) \delta(E_e - E_h - E), \tag{4.4.1.4}$$

where $C(x)$ [$\mathrm{cm}^3\mathrm{s}^{-1}$] is the (material-dependent) radiative recombination parameter which is proportional to the one specified in the database (*Radiative recombination*) and $n(\mathbf{x}, E), p(\mathbf{x}, E)$ [$\mathrm{cm}^{-3}\mathrm{eV}^{-1}$] are the charge densities as a function of energy and position.

In *nextnano*++, this radiative recombination whose rate is calculated as above is regarded as **spontaneous emission**. On the other hand, **the net amount of the stimulated emission rate** is given by:

$$R_{rad,net}^{stim}(\mathbf{x}, E) = \left(1 - e^{\frac{E-(E_{Fn}-E_{Fp})}{k_B T}}\right) R_{rad}^{spon}(\mathbf{x}, E) \tag{4.4.1.5}$$

This is consistent with eq.(9.2.39) in *[ChuangOpto1995]*. We note that here it is assumed that photon modes occupied by one photon each, i.e. takes into account neither energy-dependent photon density of states nor Bose-Einstein distribution.

Since the radiative recombination process involves no phonons, this transition is vertical and therefore this contribution is only relevant for semiconductors with a direct band gap such as GaAs.

Absorption coefficient is calculated from $R_{rad,net}^{stim}(E)$ as

$$\alpha(E) = \frac{\pi^2 \hbar^3 c^2}{n_r^2 E^2} \frac{R_{rad,net}^{stim}(E)}{V} \tag{4.4.1.6}$$

where n_r is the refractive index and V is the total volume of the device. The unit is [cm^{-1}]. In case of 1D simulation, calculated $R_{rad,net}^{stim}(E)$ has the unit [$\mathrm{cm}^{-2}\mathrm{s}^{-1}\mathrm{eV}^{-1}$] and is divided by the total length instead of the volume. This formula is consistent with eq (9.2.25) in *[ChuangOpto1995]*.

Input file

In the beginning of the input file, we define several variables for the structure and parameters for simulation. The variable definitions are shown in Figure 4.4.1.2.

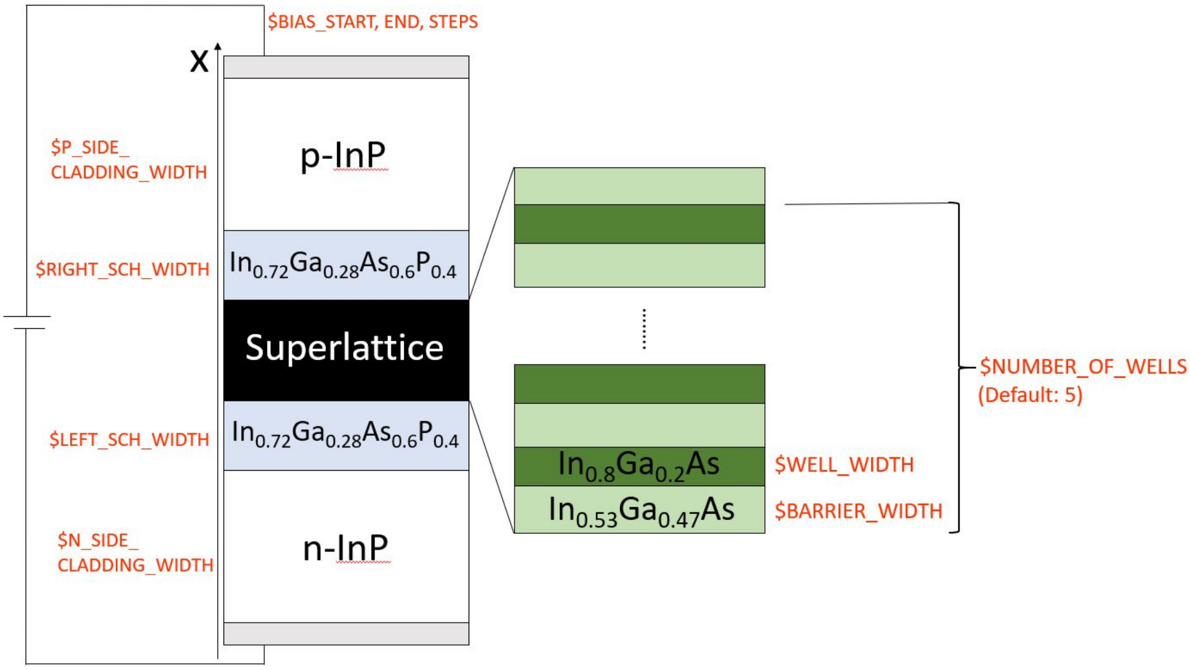


Figure 4.4.1.2: The definition of variables. The gray regions are contacts of 1nm thickness. `$NUMBER_OF_WELLS` determines the repetition of quantum wells. The program automatically sweeps the bias voltage starting from `$BIAS_START` until `$BIAS_END`, at intervals of `$BIAS_STEPS`.

Charge density as a function of position $n(\mathbf{x})$ is always calculated by default. On the other hand, charge density as a function of energy $n(E), p(E)$, charge density as a function of both position and energy $n(\mathbf{x}, E), p(\mathbf{x}, E)$ and emission spectrum are calculated when the followings are specified (see *classical{ }* for details):

```
grid{
    ...
    energy_grid{
        min_energy = -1.5                                  # Integrate from
        max_energy =  0.5                                  # Integrate to
        energy_resolution = 0.005                          # Integration resolution
    }
}


classical{
    ...
    energy_distribution{                                  # Calculation of carrier␣
↪densities as a function of energy
        min_energy = -1.5                                  # Integrate from
        max_energy =  0.5                                  # Integrate to
        energy_resolution = 0.005                          # Integration resolution
        only_density_quantum_regions = yes
    }

    energy_resolved_density{
        only_density_quantum_regions = yes
        output_energy_resolved_densities{}
```

(continues on next page)

(continued from previous page)

```
    }


    semiclassical_spectra{
        output_spectra{

            emission = yes
            gain = yes
            absorption = yes

            spectra_over_energy = yes
            spectra_over_wavelength = yes
            spectra_over_frequency = yes
            spectra_over_wavenumber = yes

            photon_spectra = yes
            power_spectra = yes

        }
        output_photon_density = yes
        output_power_density = yes
    }
}
```

The mobility model and recombination models for the current equation are specified in *currents{ }* as

```
currents{
    mobility_model = constant
#   mobility_model = minimos
    recombination_model{
        SRH         = yes           # 'yes' or 'no'
        Auger       = yes           # 'yes' or 'no'
        radiative   = yes           # 'yes' or 'no'
    }
}
```

The *run{ }* flag specifies which equations to solve. This is the main difference between `LaserDiode_*_qm_nnp.in` and `LaserDiode_*_cl_nnp.in`.

```
# qm
run{
    strain{ }                                        # solves the strain equation
    current_poisson{                                # solves the coupled current and␣
↪Poisson equations self-consistently
        output_log     = yes
        iterations     = 1000
        alpha_fermi = 0.5
    }
    quantum_current_poisson{                        # solves the Schrödinger, Poisson␣
↪(and current) equations self-consistently
        iterations     = 1000
        alpha_fermi    = 0.9
        residual       = 1e6
        residual_fermi = 1e-8
        output_log     = yes
    }
}
```

(continues on next page)

(continued from previous page)

```
# cl
run{
    strain{ }                                              # solves the strain equation
    current_poisson{                                      # solves the coupled current and␣
↪Poisson equations self-consistently
        output_log     = yes
        iterations     = 1000
        alpha_fermi    = 0.7
        residual_fermi = 1e-10
    }
}
```

In this case *nextnano++* first solves the strain equation from the crystal orientation to decide the polarization charges (piezoelectric effect) and shifted band edges. Then the program solves the coupled current-Poisson-Schrödinger equations in a self-consistent way (input file: `LaserDiode_InGaAs_1D_qm_nnp.in`). For the classical calculation (`LaserDiode_InGaAs_1D_cl_nnp.in`), `quantum_current_poisson{ }` is commented out to restrict the calculation to the current-Poisson equations only.

Results

Band structure

The band structure and emission power spectrum of the system are stored in `bandedges.dat`. Figure 4.4.1.3 shows the case for the bias 0.2 V. Here the quasi Fermi level of electrons is lower than the quantum wells.

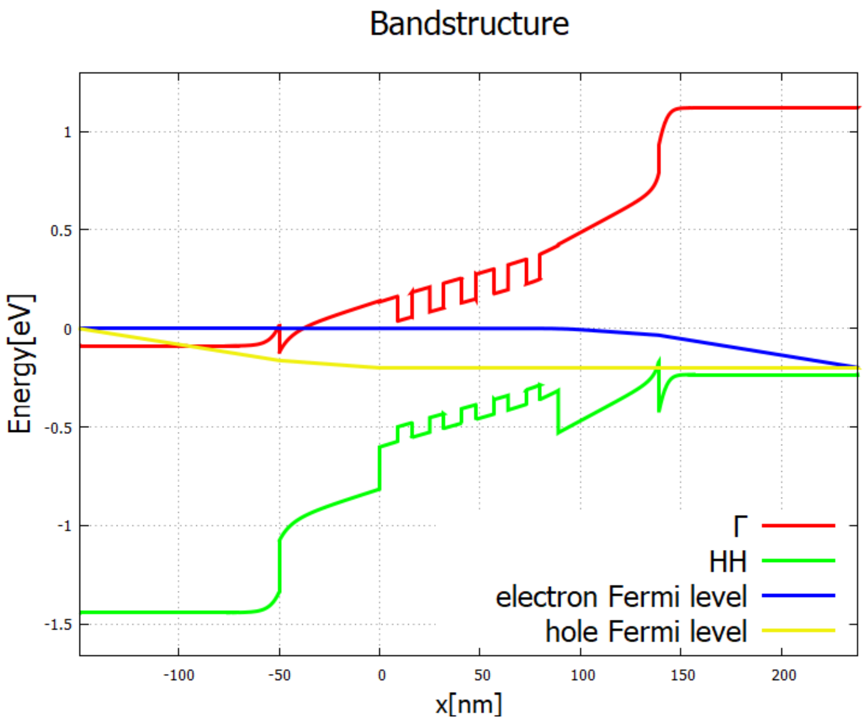


Figure 4.4.1.3: Band structure of the laser diode system for a low bias of 0.2 V.

For the bias 0.8 V (Figure 4.4.1.4), in contrast, it lies above the red line, allowing electrons to flow into the quantum wells. An electron trapped in the quantum wells is likely to recombine with a hole in the valence band, emitting a photon. In the input file `\Optical\emission_photon_density.dat`, one can see that the photons are emitted from this active region (not shown). Figure 4.4.1.12 shows the emission spectrum in this case. When the bias is too small, e.g. Figure 4.4.1.3, the intensity is much smaller, as can be seen in Figure 4.4.1.16.

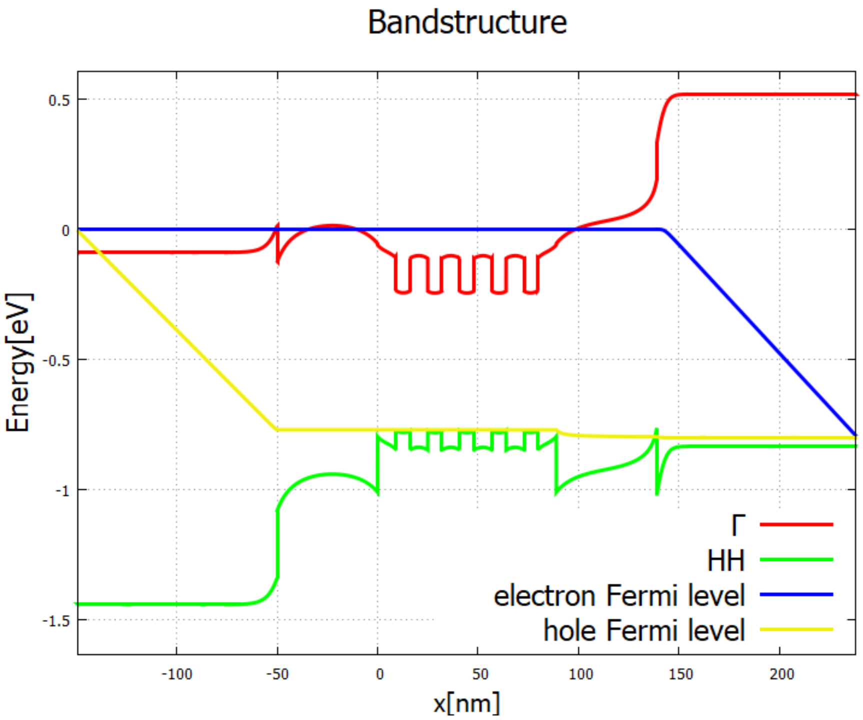


Figure 4.4.1.4: Band structure for a high bias 0.8 V. Electrons flowing from the left and holes from the right recombine in the active zone (multi-quantum well structure).

Energy eigenstates and eigenvalues

In the input file `LaserDiode_InGaAs_1D_qm_nnp.in`, the single-band Schrödinger equation is coupled to the current-Poisson equation and solved self-consistently. The wave functions of electrons and holes along with eigenvalues are written in `\Quantum\probabilities_shift_quantum_region_Gamma_0000.dat` and `\Quantum\probabilities_shift_quantum_region_HH_0000.dat` (Figure 4.4.1.5 and Figure 4.4.1.6). The light hole and split-off states are out of the quantum wells and not of our interest here.

Charge densities

We can find the energy-resolved charge density $n(x,E)$ and $p(x,E)$ in the output `electron_density_vs_energy.fld` and `hole_density_vs_energy.fld`. The following figures represent $n(x,E)$ and $p(x,E)$ $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ with respect to the band edges and quasi-Fermi levels at bias 0.2, 0.4, 0.6, 0.8 and 1.0 V. We can see that the carrier densities around the quantum wells increase as the bias increases.

We also have the charge densities integrated over the device $n(E), p(E)$ $[\mathrm{cm}^{-2}\mathrm{eV}^{-1}]$ and energy $n(x), p(x)$ $[\mathrm{cm}^{-3}]$.

$n(E)$ and $p(E)$ with and without quantum calculation shows different features due to the discretization of energy levels in quantum wells. This is shown in `integrated_densities_vs_energy.dat`.

Figure 4.4.1.12 illustrates the population inversion in stationary (quasi-equilibrium) state of the device under bias. Solid and dashed lines are for quantum and classical calculations, respectively. The black arrows mark the relevant energies of the structure 4 at bias of 0.8 V. The hole density is shown in Figure 4.4.1.13 with higher resolution.

The energy resolution in Figure 4.4.1.13 has been increased by a factor of 10 from Figure 4.4.1.12.

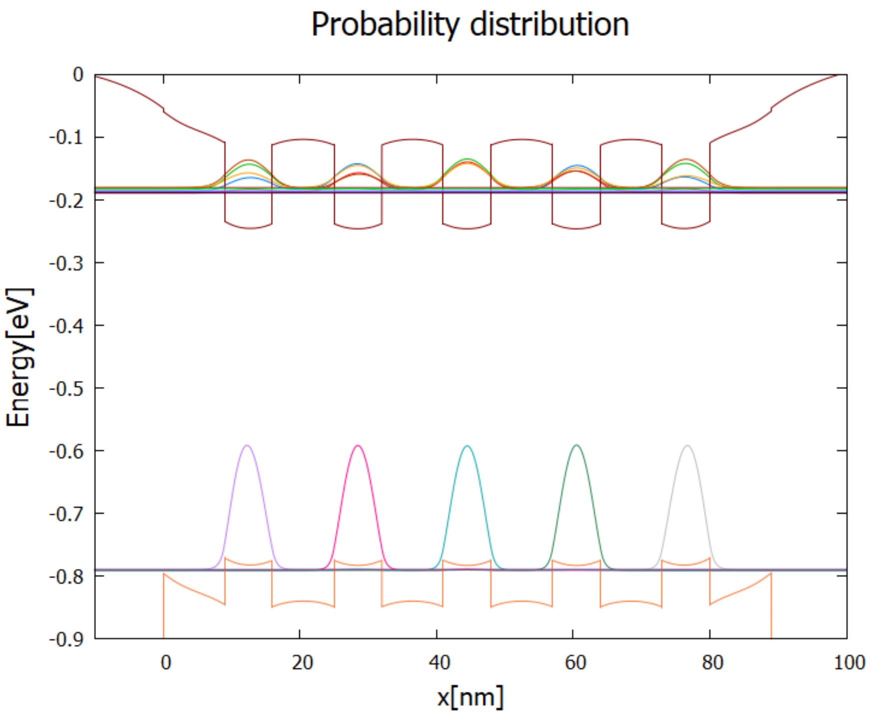


Figure 4.4.1.5: Probability distribution $|\psi(x)|^2$ of the lowest localized modes of electrons and holes for the band structure Figure 4.4.1.3. Horizontal lines are the corresponding eigenenergies.

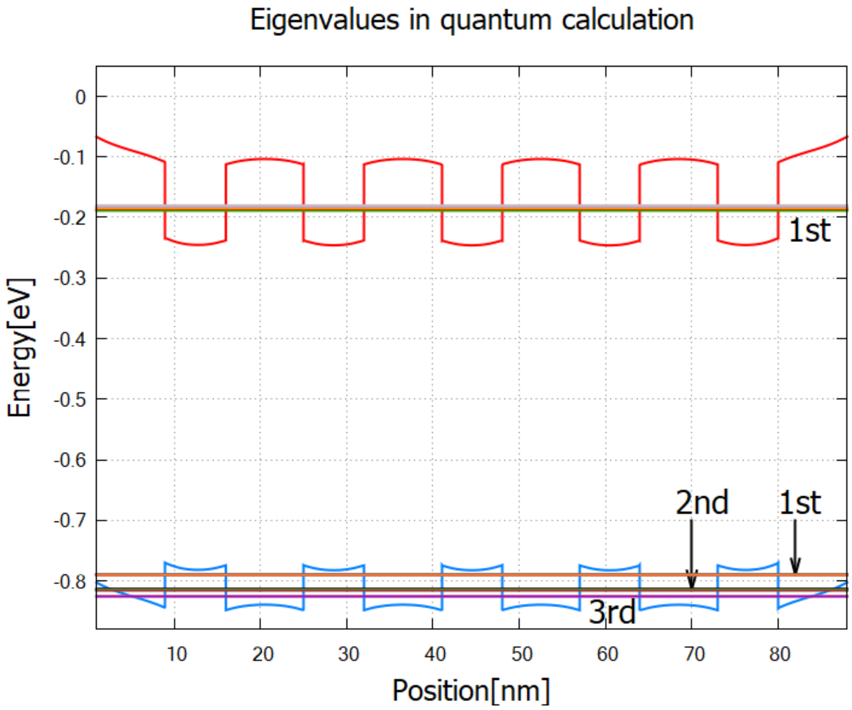


Figure 4.4.1.6: Eigenvalues of the Gamma-band up to 5th and heavy-hole-band states up to 13th in relation to band edges. The Eigenvalues above these are higher than the barrier energy of the quantum wells. The Gamma band has single "miniband", whereas the heavy-hole band has three. The 1st heavy-hole miniband consists of the 1st~5th eigenvalues, the 2nd heavy-hole miniband consists of the 6th~11th eigenvalues and the 3rd consists of the 12th and 13th eigenvalues.

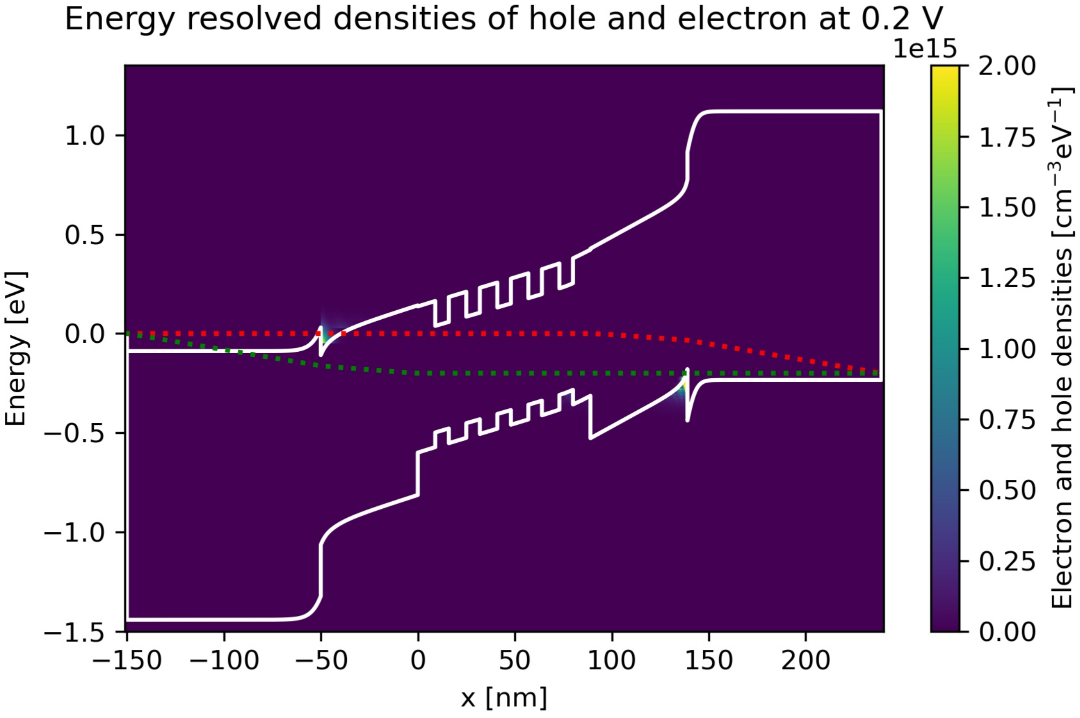


Figure 4.4.1.7: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.2 V in quantum calculation.

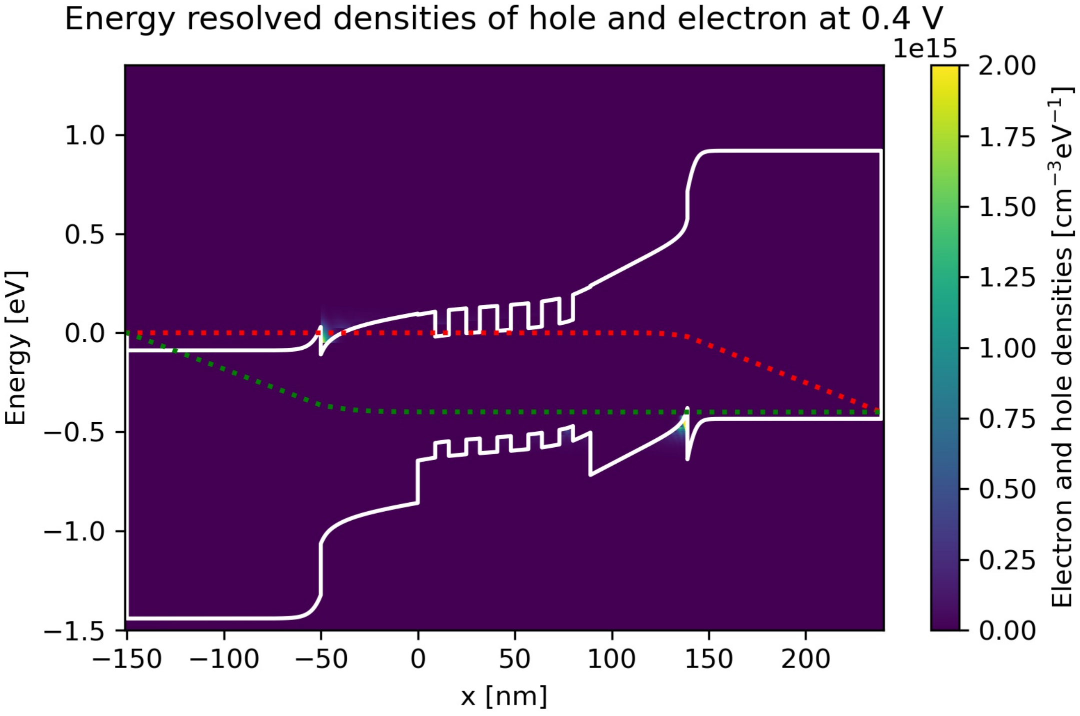


Figure 4.4.1.8: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.4 V in quantum calculation.

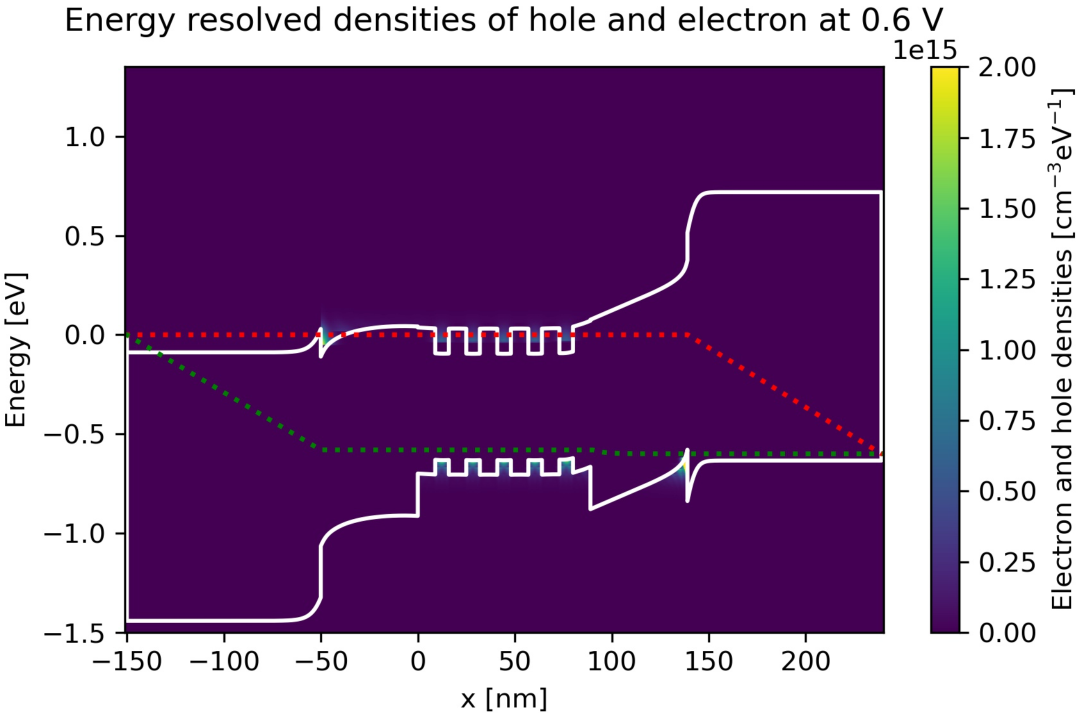


Figure 4.4.1.9: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.6 V in quantum calculation.

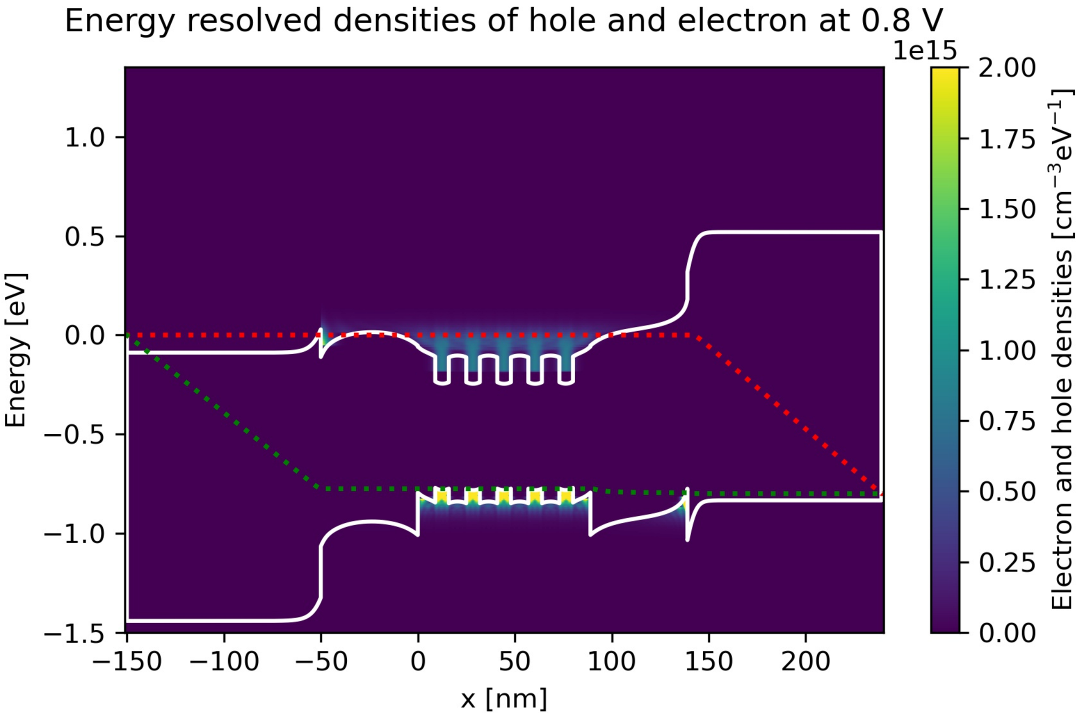


Figure 4.4.1.10: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.8 V in quantum calculation.

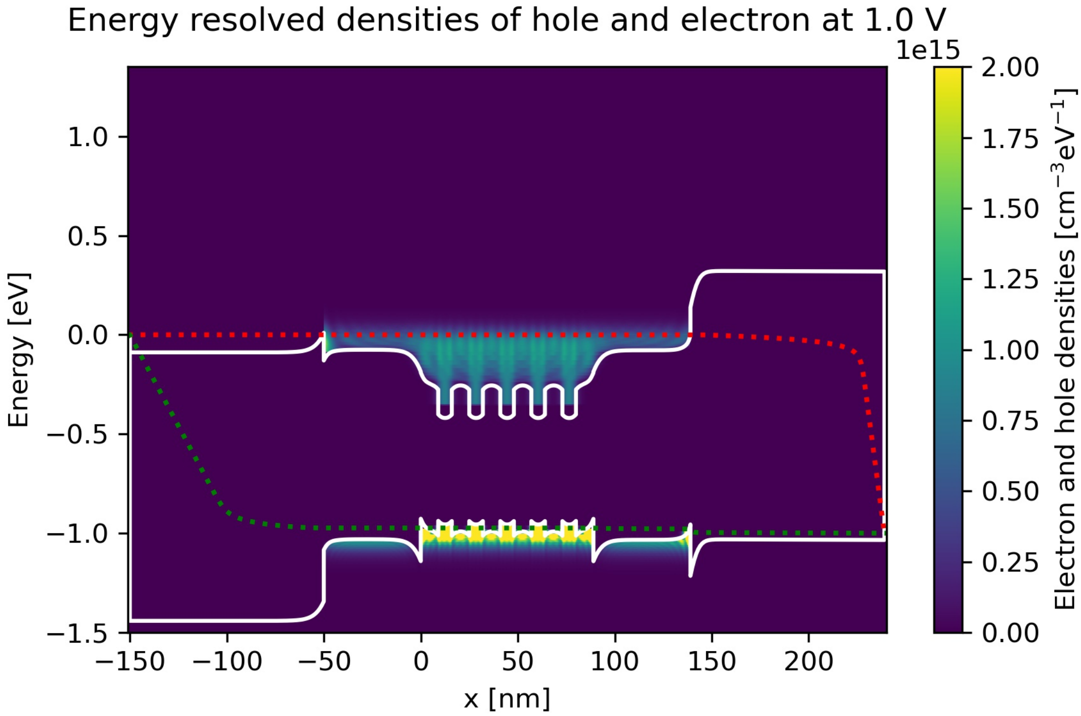


Figure 4.4.1.11: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 1.0 V in quantum calculation.

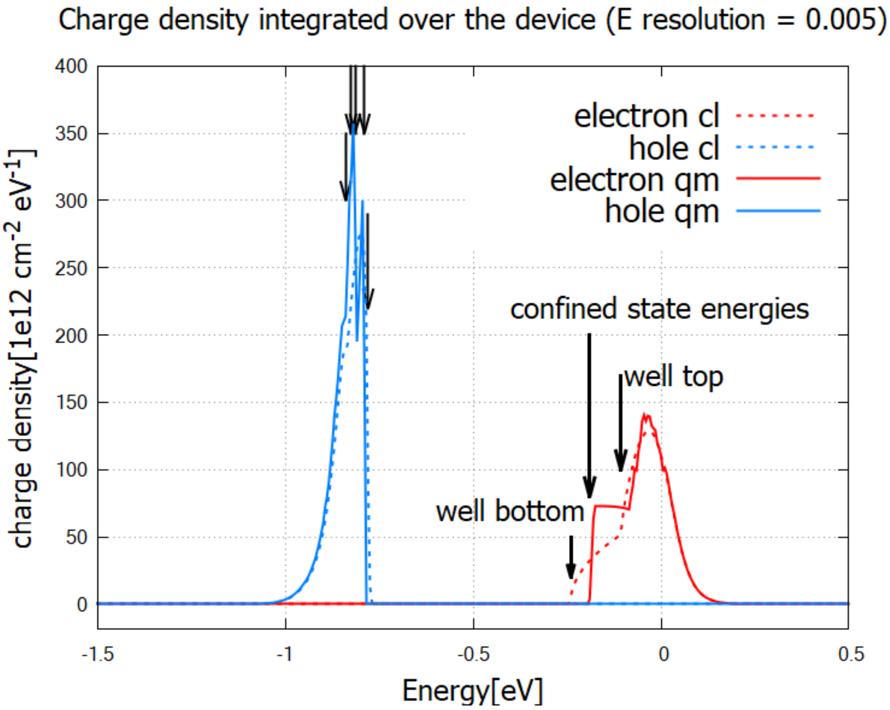


Figure 4.4.1.12: Electron (red) and hole (blue) densities integrated over the device as a function of energy.

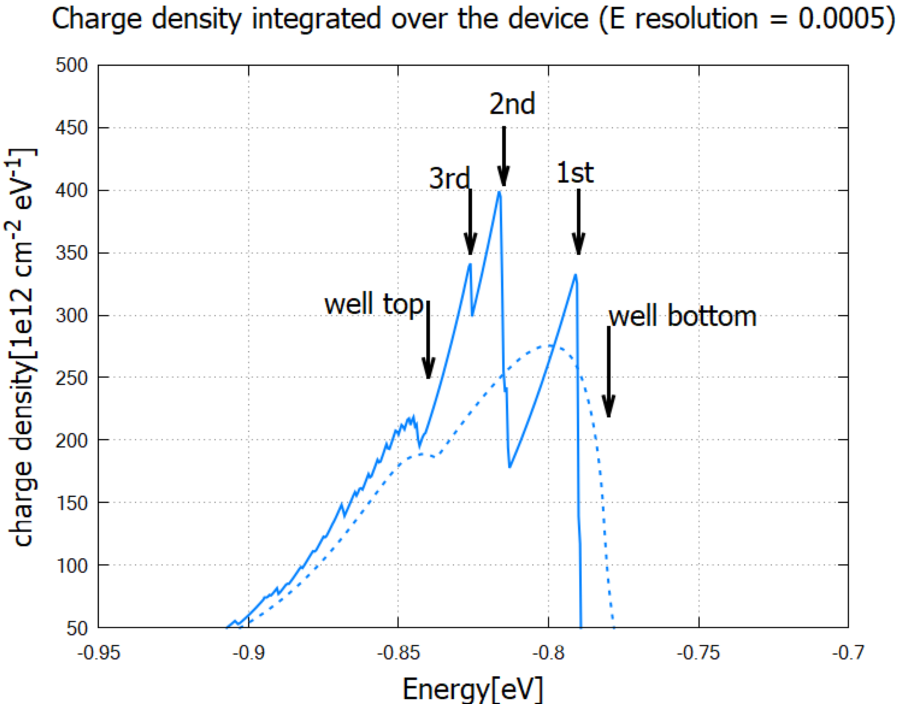


Figure 4.4.1.13: Hole density integrated over the device from classical (dashed) and quantum (solid) calculation.

Note

Although these charge densities either with variable E or x are both obtained by integrating $n(x, E)$ and $p(x, E)$ over the corresponding variable, these are independently calculated in nn++ simulation. Hence it is possible to turn off the calculation only for $n(x, E)$ and $p(x, E)$ calculating the integrated charge densities. In this case it runs much faster and needs much less memory.

Emission and absorption spectra

The spontaneous and stimulated emission spectra are written in `\Optical\semiclassical_spectra_photons.dat` and `\Optical\stim_semiclassical_spectra_photons.dat`, respectively (Figure 4.4.1.14). The peak is at around 0.7-0.8eV, which is consistent with the charge distribution in Figure 4.4.1.12. The stimulated emission does not occur above the quasi Fermi level separation, $E_{Fn} - E_{Fp}$.

The formulas used for the calculation in the source code are specified above: *Recombination of carriers and emission spectrum*.

The absorption spectra are calculated as

$$\alpha(E) = \frac{\pi^2 \hbar^3 c^2}{n_r^2 E^2} \frac{R^{stim}_{rad,net}(E)}{V}$$

where n_r is the refractive index and V is the total volume of the device. The unit is [cm^{-1}]. In case of 1D simulation, calculated $R^{stim}_{rad,net}(E)$ has the unit [cm^{-2}s^{-1}eV^{-1}] and is divided by the total length instead of the volume. This formula is consistent with eq (9.2.25) in *[ChuangOpto1995]*.

The absorption spectra $\alpha(E)$ and gain spectra $g(E)$ are essentially the same quantity with opposite signs,

$$\alpha(E) = -g(E)$$

These are by definition independent of the initial photon population. **Please note that the gain spectrum in nextnano++ is cut off where it is negative**. For details, see *classical{ }*.

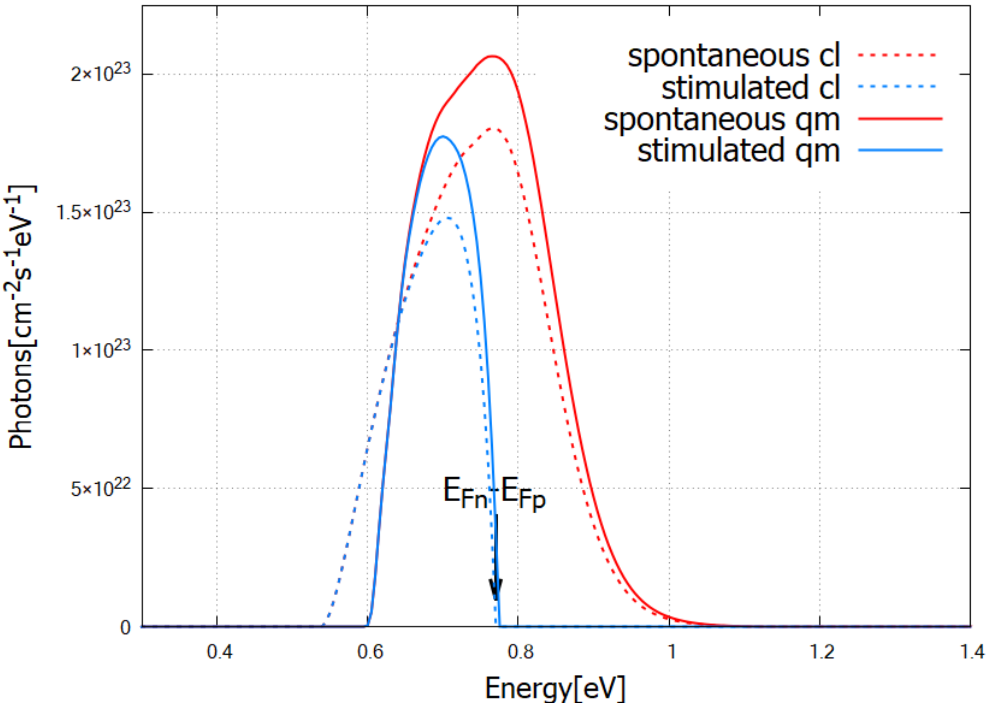


Figure 4.4.1.14: Emission spectrum of the laser diode for the bias 0.8 V.

The spectrum changes its sign at the energy $E_{Fn} - E_{Fp}$, that is, the separation of the quasi Fermi levels. According to the output `bandedges.dat`, this value is -0.0001-(-0.7702)=0.7701eV. The following result has been calculated classically. We also get qualitatively consistent results from quantum mechanical simulation.

Current and internal quantum efficiency

The output file `IV_characteristics.dat` contains right- and left-contact current in unit of [Acm^{-2}]. In the present case, the right-contact current is hole current, whereas the left-contact current is electron current. In Figure 4.4.1.15, we compare the hole current and photocurrent.

Figure 4.4.1.16 clearly shows the consequence of the difference in band structures Figure 4.4.1.3 and Figure 4.4.1.4. The holes and electrons recombine in the multi-quantum well layers, emitting one photon per electron-hole pair. The efficiency of conversion from charge current into photocurrent is called the *internal quantum efficiency*

$$\eta = \frac{I_\text{photon}}{I_\text{charge}}. \tag{4.4.1.7}$$

This quantity is written in `internal_quantum_efficiency.dat` and shown in Figure 16.

Last update: 16/07/2024

4.4.2 UV LED: Quantitative evaluation of the effectiveness of EBL

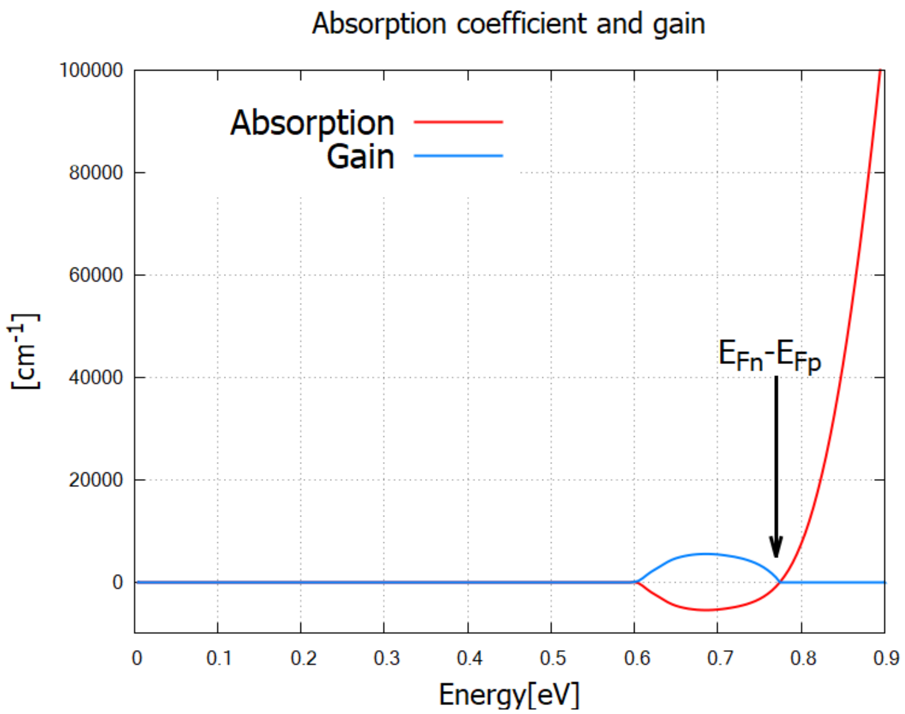


Figure 4.4.1.15: Classically calculated absorption and gain spectra. The sign of the spectrum switches at the energy corresponding to the quasi Fermi-level separation in the active region.

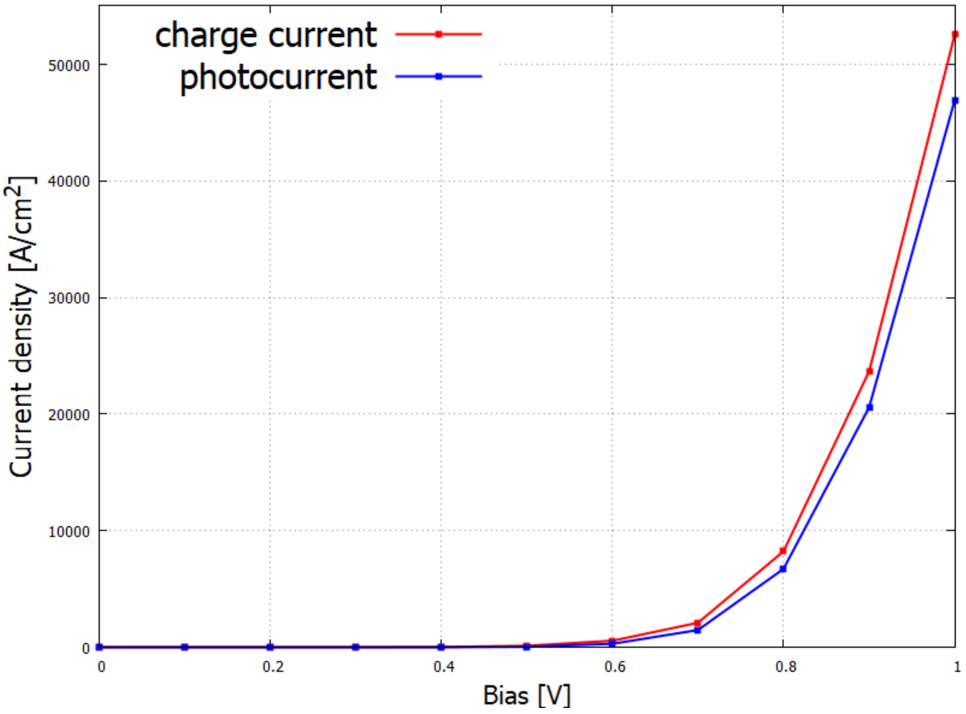


Figure 4.4.1.16: Charge current and photocurrent as a function of bias voltage (IV characteristics).

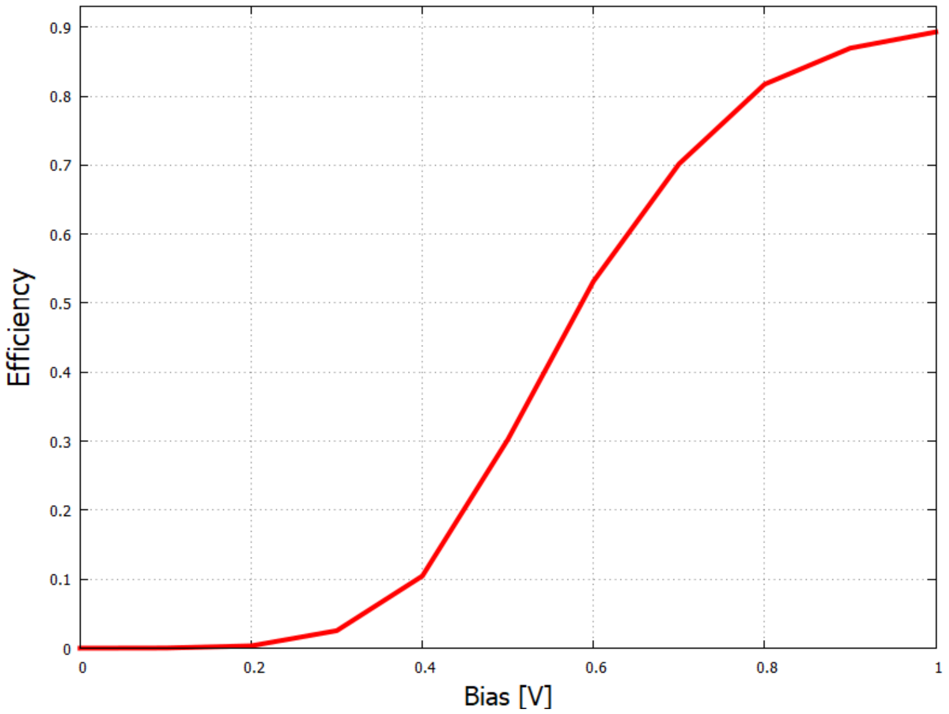


Figure 4.4.1.17: Conversion efficiency of the InGaAs laser diode.

Header

Files for the tutorial located in *nextnano++\examples*

- *1D_DUV_LED_HirayamaJAP2005_EBL_nnp.in*

We investigate how the electron blocking layer (EBL) improves the characteristics of UV LEDs using *nextnano*++. Current-Poisson equation and semi-classical calculation of optical properties (*classical{ }*) in *nextnano*++ enables us to quantitatively analyze the effect of this strucutre.

We refer to the structure used to obtain Fig. 28 in the *[HirayamaJAP2005]*:

Structure

The simulation region consists of the following structure:

- n-$Al_{0.18}Ga_{0.82}N$ layer
- 3-layer MQW based on InAlGaN
- $Al_xGa_{1-x}N$ EBL (Al content = 0.18, 0.24, 0.28)

- p-$Al_{0.18}Ga_{0.82}N$ layer

Each layer has the following thickness and doping concentration:

Material	Thickness	Doping
n-$Al_{0.18}Ga_{0.82}N$	100 nm	8 × 10^{18} [cm^{-3}] (donor)
$In_{0.02}Al_{0.09}Ga_{0.89}N$ - $In_{0.02}Al_{0.22}Ga_{0.76}N$ 3-layer MQW	well: 2.5 nm, barrier: 15 nm	0 [cm^{-3}]
$Al_xGa_{1-x}N$ EBL with x=0.28, 0.24, 0.18	10 nm	0 [cm^{-3}] for x=0.28, 0.24, 2 × 10^{19} [cm^{-3}] for x=0.18 (acceptor)
p-$Al_{0.18}Ga_{0.82}N$	100 nm	2 × 10^{19} [cm^{-3}] (acceptor)

Al content x=0.18 in the EBL is used for the structure without EBL, while x=0.24 and 0.28 are for the structure with EBL in different barrier height.

Donor and acceptor ionization energies are defined as 0.030 eV and 0.158 eV where Si and Mg are in mind, respectively.

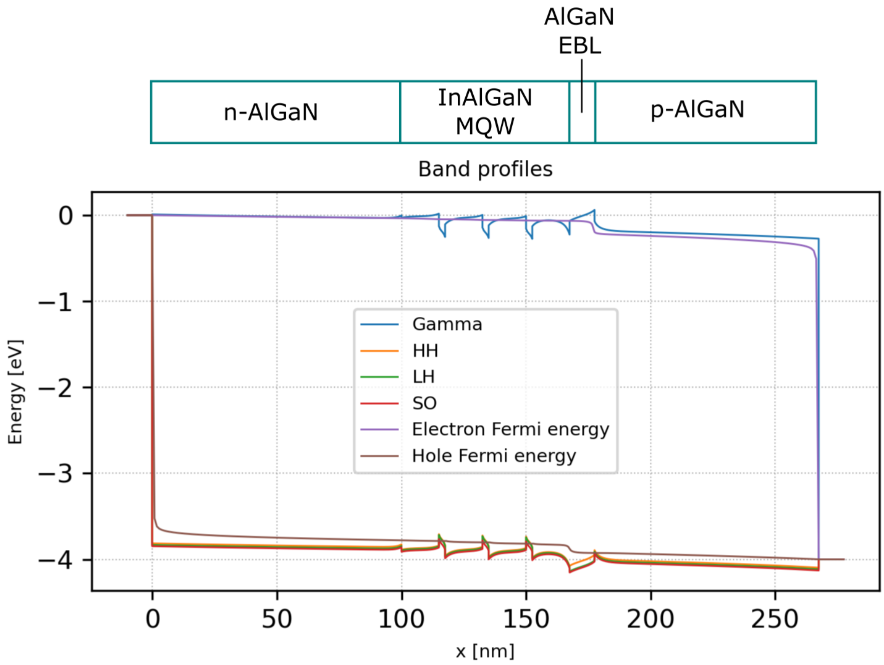


Figure 4.4.2.1: The band edges and Fermi levels for the structure with higher EBL (x=0.28, bias=4.00V, total current density=1.67×10^5 A/cm^2)

Scheme

We can specify which simulation or equations would be solved on *run{ }* section in your input file.

In *1D_DUV_LED_HirayamaJAP2005_EBL_nnp.in* it is described as

```
run{
    strain{ }
    current_poisson{ }
}
```

Then *nextnano*++ solves the current equation and Poisson equation self-consistently after solving strain equation.

After the Current-Poisson equation is converged, optoelectronic characteristics are calculated according to the specification in the section *classical{ }*.

For further details, please see *Optoelectronic characterization*.

Results

Current-voltage characteristics

Here we show the current-voltage characteristics for the total current density I_{total} measured at p-contact and photocurrent density I_{photo}, which is defined as (2.15.1.1). I_{photo} represents the amount of electrical current consumed by the radiative recombination in the total current I_{total}. Please note that the scales of the y-axis in these graphs are different in 10 times.

We can observe that the smaller I_{total} is , the higher the EBL barrier is. On the other hand, at the applied bias of 4.0V, the bigger I_{photo} is, the higher the EBL barrier is. We can say that the larger proportion of the total current consists of the photocurrent in the higher EBL structure, which results in the larger IQE.

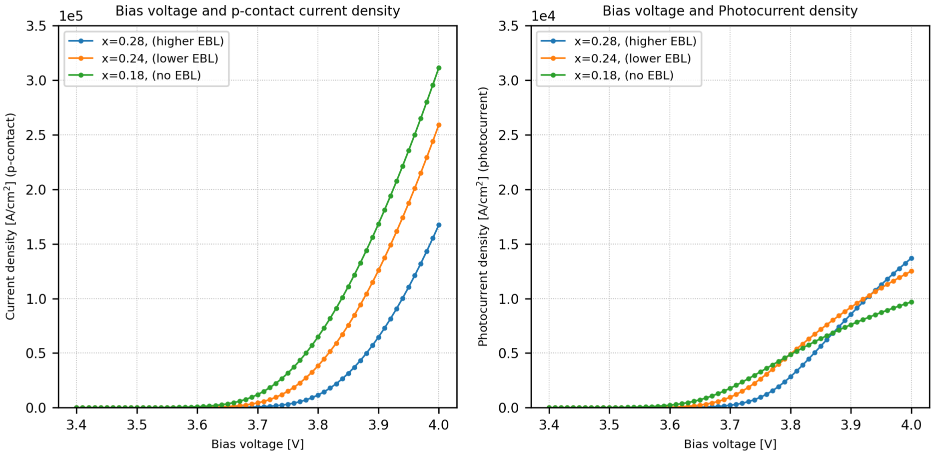


Figure 4.4.2.2: (*Left*:) The relationship between the p-contact current density and bias voltage. (*Right*:) The relationship between the photocurrent I_{photo} and bias voltage.

Bandedges

The following figures show the band edge profiles and the quasi-Fermi levels for the higher EBL (top) and no EBL (bottom) structure where the total current densities are almost the same around 1.70×10^5 A/cm^2. The applied bias is 4.00 V for the left graph and is 3.90 V for the right graph.

Current Density

The following figure show the current density profiles for the higher EBL (top, x=0.28), lower EBL (middle, x=0.24), and no EBL (bottom, x=0.18) structure where the total current densities are almost the same around 1.70×10^5 A/cm^2.

We can see that the amount of electron current and hole current becomes closer as the EBL height is increased, while the electron current is dominant without EBL. It can be also confirmed that the current overflow is suppressed by the EBL.

Charge carrier densities

The figures showed below are the electron and hole densities around the MQW region for the structure with higher EBL and without EBL (left, x=0.28 and right, x=0.18) for almost the same current density around 1.70×10^5 A/cm^2. The introduction of EBL at 167 nm-177 nm reduces the electron densitiy in the p-AlGaN region.

Power of light emission

Here we show the relationship between optical power defined in (2.15.1.4) and current density of p-contact for each structure.

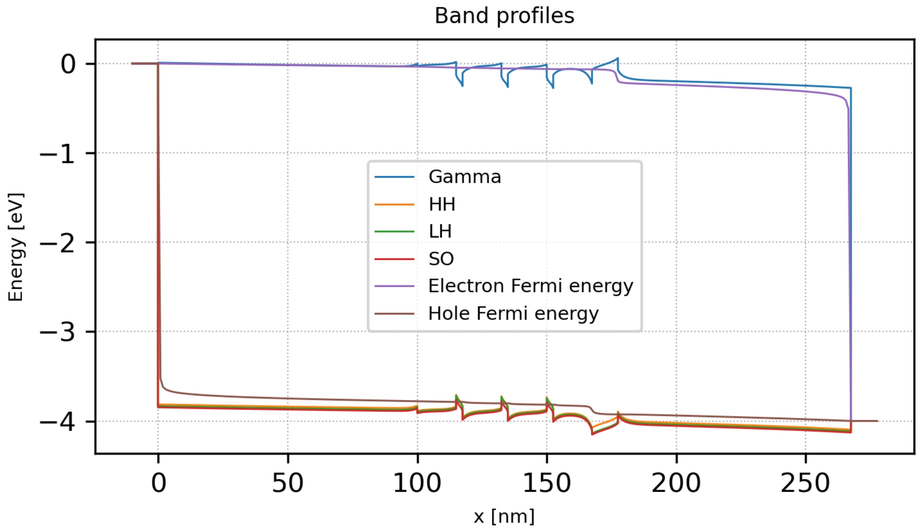


Figure 4.4.2.3: The band edges and Fermi levels for the structure with EBL (x=0.28, bias=4.00V, total current density=1.67×10^5 A/cm^2)

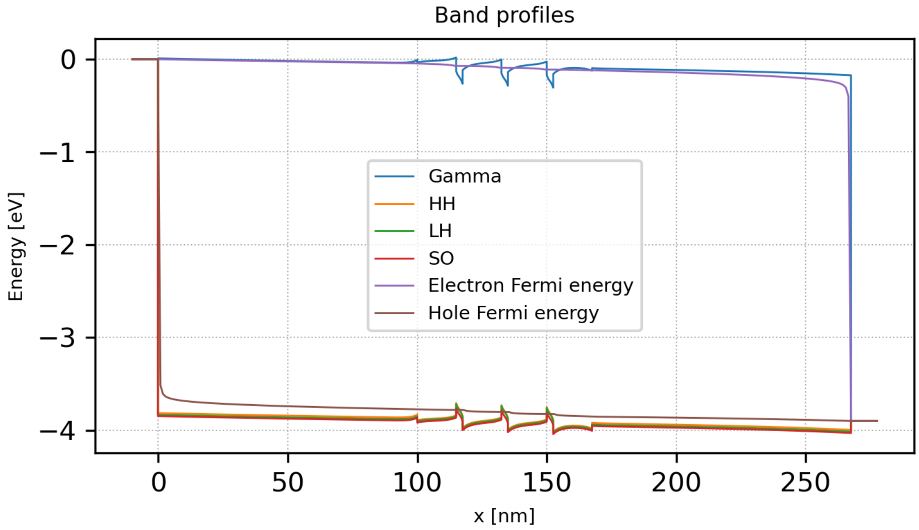


Figure 4.4.2.4: The band edges and Fermi levels for the structure without EBL (x=0.18, bias=3.90V, total current density=1.68×10^5 A/cm^2)

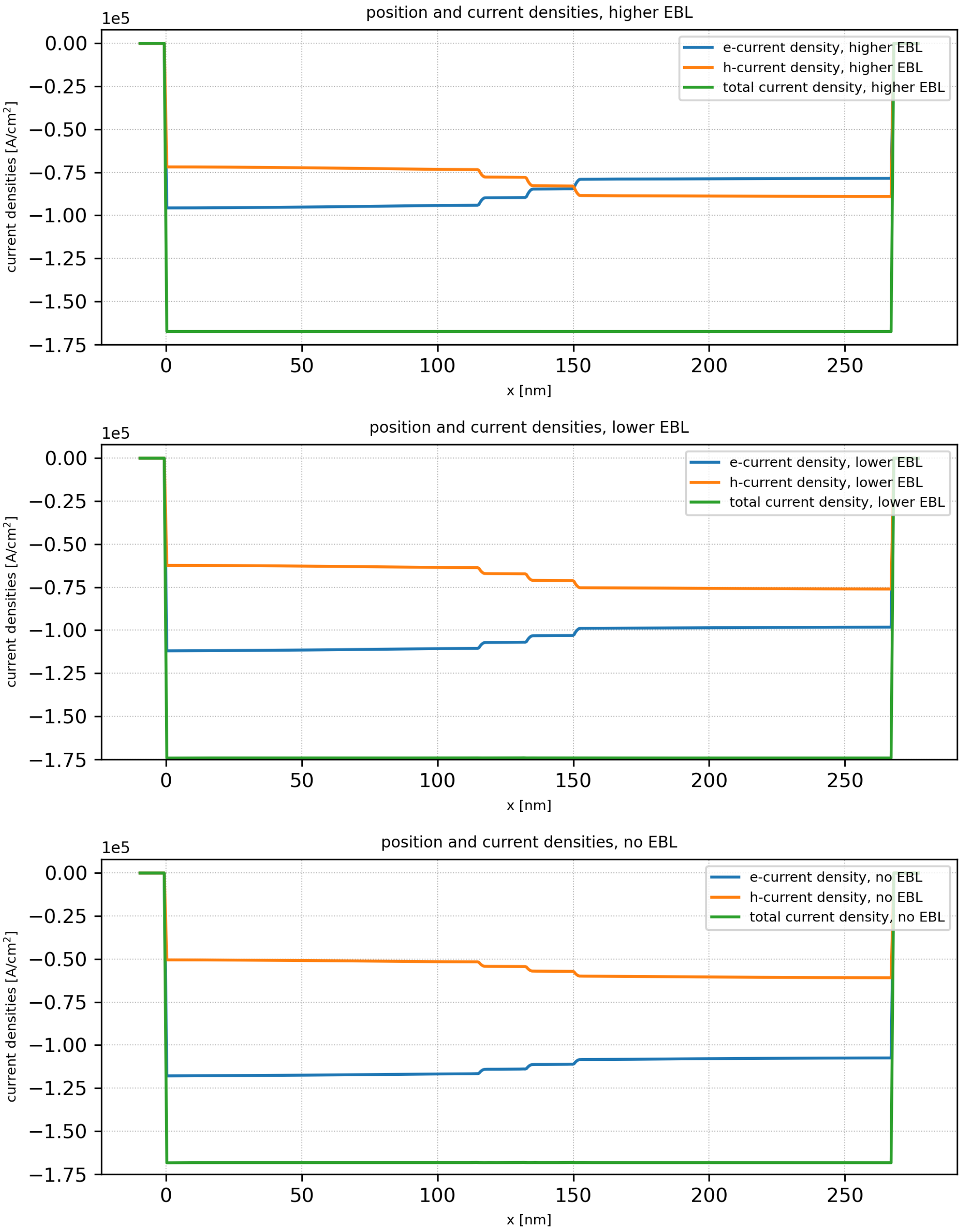


Figure 4.4.2.5: The current density profile for the the structures with higher EBL (*top*, 4.00 V, 1.67× 10^5 A/cm^2), lower EBL (*middle*, 3.94 V, 1.74× 10^5 A/cm^2), and no EBL (*bottom*, 3.90 V, 1.68× 10^5 A/cm^2).

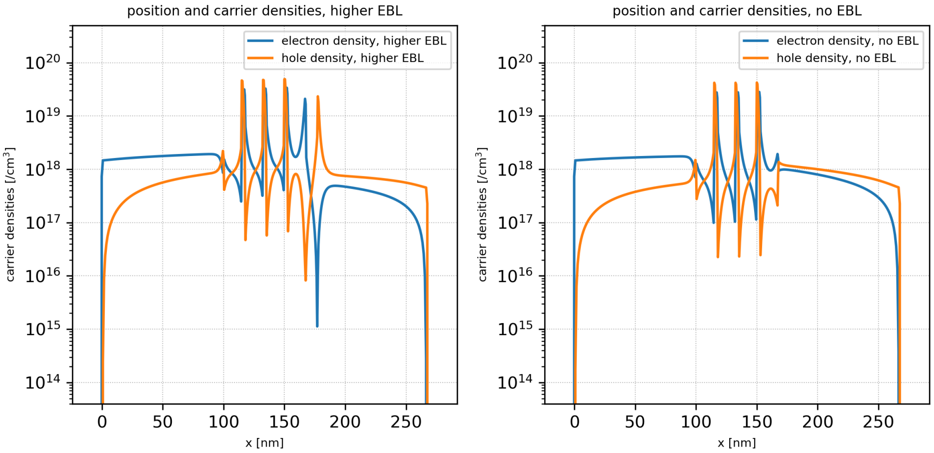


Figure 4.4.2.6: The electron and hole densities calculated in the structures with higher EBL (*left*, 4.00 V, 1.67×10^5 A/cm^2) and no EBL (*right*, 3.90 V, 1.68×10^5 A/cm^2).

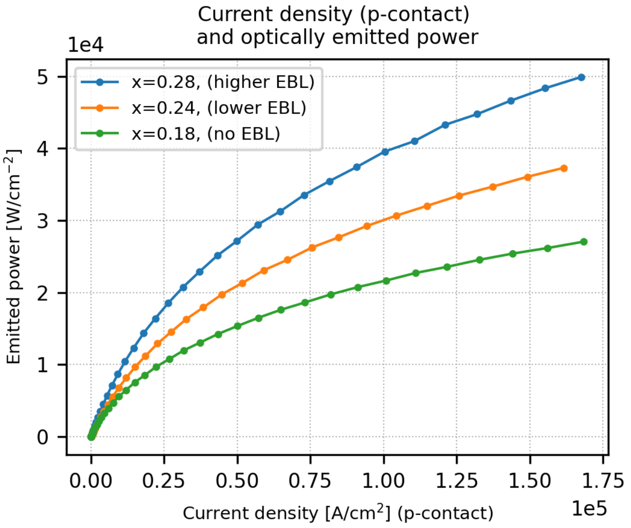


Figure 4.4.2.7: Current vs. power of light emission

Internal quantum efficiency

In *nextnano++*, the **internal quantum efficiency** is calculated as

$$\eta_{\text{IQE}} = \eta_{\text{VQE}} \cdot \eta_{\text{IE}} = \frac{I_{\text{photo}}}{I_{\text{total}}} \tag{4.4.2.1}$$

where I_{photo} is the photo-urrent consumed by the radiative recombination and I_{total} is the current injected in total.

This quantity shows the improvement by the introduction of higher EBL as follows:

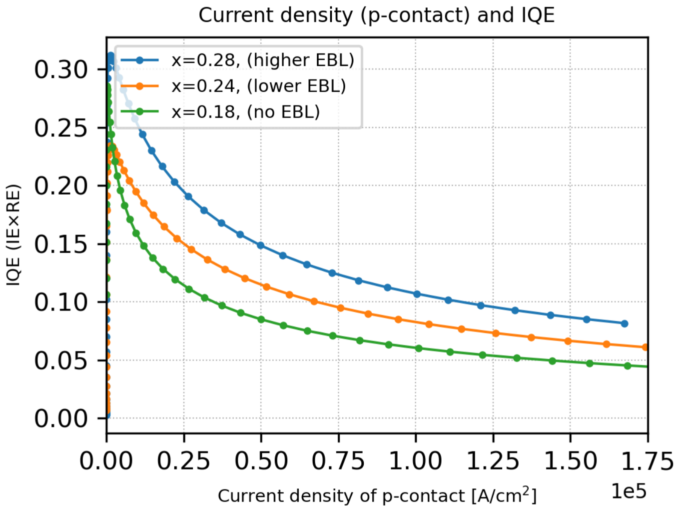


Figure 4.4.2.8: Current and internal quantum efficiency (IQE).

The *nextnano++* tool also outputs the **volume quantum efficiency** η_{VQE}, also known as **radiative efficiency**, which represents the proportion of the radiative recombination rate to the total recombination rate. This quantity is calculated as

$$\eta_{\text{VQE}} = \frac{R^{stim}_{\text{rad,net}} + R_{\text{fixed}}}{R_{\text{total}}} \tag{4.4.2.2}$$

and also shows the improvement by the introduction of EBL:

The IQE can be decomposed like (4.4.2.1) into this volume QE and the **injection efficiency** η_{IE}, which represents the proportion of the current consumed by the total recombination (radiative + nonradiative) to the total injected current.

Thus using the results of η_{IQE} and η_{VQE} above, we can also get this η_{IE} :

From the above results, we can see that the improvement of IQE due to the introduction of EBL comes from the imrovement of mainly IE rather than volume QE.

What can we do further?

The effect of EBL on the optoelectronic characteristics has been estimated quantitatively using the semiclassical calculation in *nextnano++*.

We can also optimize the Al content of EBL or the thickness by sweeping the corresponding parameters, for example. Our open source python package *nextnanopy* is a strong tool for this purpose.

The graphs shown in this tutorial are also generated by a python script using *nextnanopy*.

Last update: 16/07/2024

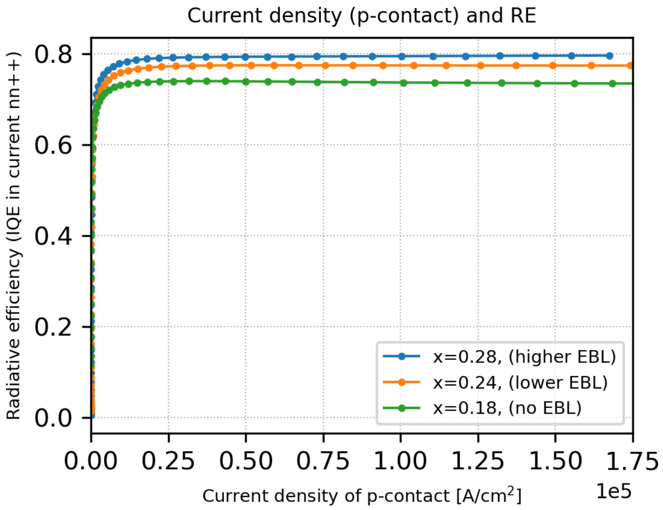


Figure 4.4.2.9: Current and volume quantum effciency (radiative efficiency).

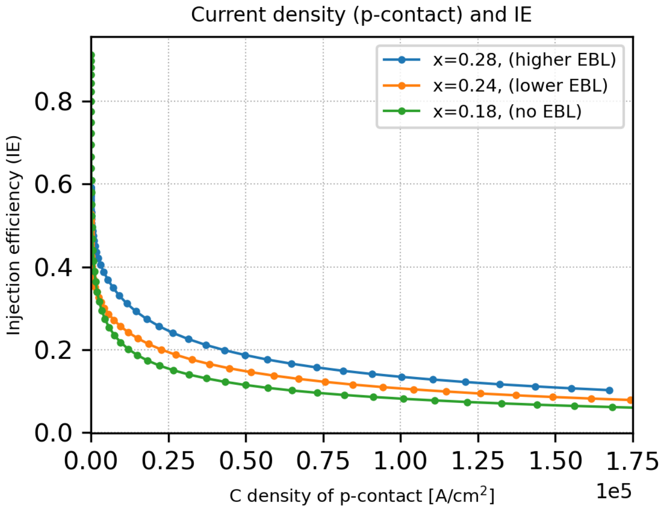


Figure 4.4.2.10: Current and injection efficiency (IE).

4.4.3 UV LED: Quantitative evaluation of the effectiveness of superlattice structure in p-region

Header

Files for the tutorial located in *nextnano++\examples*

- *1D_UV_LED_KozodoyAPL1999_nnp.in*
- *1D_DUV_LED_HirayamaJAP2005_SL_nnp.in*

In the recent UV-LEDs based on AlGaN, the superlattice (SL) structure is introduced into the p-type layer in order to enhance the acceptor ionization, which results in the improvement of the hole conductivity. We investigate how this structure improves the characteristics of UV LEDs using *nextnano*++.

First, the hole concentration in a p-type AlGaN/GaN SL structure is calculated using Schrödinger-Poisson solver and the enhancement of the acceptor ionization is quantitatively examined. This part is based on *[SchubertAPL1996]* and *[KozodoyAPL1999]*.

Second, the SL structure is introduced into the p-region of LED structure with InAlGaN MQW and Current-Poisson equation is solved. Then the IQE result is compared to the LED structure with the bulk p-region. The structure used in this part is based on *[HirayamaJAP2005]*.

Hole density estimation

Structure

The simulation region consists of the following structure:

Material	Thickness	Doping
$Al_{0.2}Ga_{0.8}N$/GaN 8-layer MQW	$L = L_{\mathrm{well}} = L_{\mathrm{barrier}}$	5.0×10^{19} [cm^{-3}]

The simulation is sweeped over the well and barrier thickness L from 1 nm to 10 nm.

Bandedges

The following figure shows the band edge profile and the Fermi energy for the SL structure with $L = 4.0$ nm.

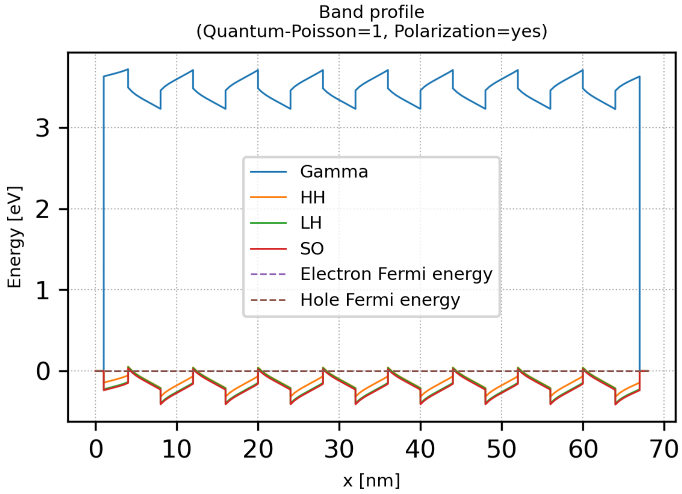


Figure 4.4.3.1: The band edge profile and the Fermi level

The band edge tilting is due to the piezo- and pyro-electricity, which actually enhances the acceptor ionization as can be seen later.

Scheme

Schrödinger-Poisson equation

We can specify which simulation or equations would be solved on `run{ }` section in your input file.

In *1D_UV_LED_KozodoyAPL1999_nnp.in* it is described as

```
run{
    strain{ }
    poisson{ }
    quantum_poisson{ }
}
```

Then *nextnano++* solves the strain equation and self-consistent Schrödinger-Poisson equation.

The resulting electrostatic potential $\phi(x)$, electron density $n(x)$, and hole density $p(x)$ should satisfy both Poisson equation and the carrier density calculation based on Schrödinger equation. For further detailed discussion, please refer to *Optoelectronic characterization*.

Ionization of dopant

The ionized donor and acceptor densities, N_D^+, N_A^- are calculated as

$$N_D^+(\mathbf{x}) = \sum_{i\in\text{Donors}} \frac{N_{D,i}(\mathbf{x})}{1 + g_{D,i}\exp((E_{F,n}(\mathbf{x}) - E_{D,i}(\mathbf{x}))/k_BT)} \tag{4.4.3.1}$$

$$N_A^-(\mathbf{x}) = \sum_{i\in\text{Acceptors}} \frac{N_{A,i}(\mathbf{x})}{1 + g_{A,i}\exp((E_{A,i}(\mathbf{x}) - E_{F,p}(\mathbf{x}))/k_BT)} \tag{4.4.3.2}$$

where the summation is over all different donor or acceptors, N_D, N_A are the doping concentrations, g_D, g_A are the degeneracy factors ($g_D = 2$ and $g_A = 4$ for shallow impurities), and E_D, E_A are the energies of the neutral donor and acceptor impurities, respectively.

These energies E_D, E_A are determined by the ionization energies $E_{D,i}^{ion}, E_{A,i}^{ion}$, the bulk conduction and valence band edges (including shifts due to strain) and the electrostatic potential as

$$E_D(\mathbf{x}) = E_{c,0}(\mathbf{x}) - e\phi(\mathbf{x}) - E_D^{ion}(\mathbf{x}), \tag{4.4.3.3}$$

$$E_A(\mathbf{x}) = E_{v,0}(\mathbf{x}) - e\phi(\mathbf{x}) + E_A^{ion}(\mathbf{x}). \tag{4.4.3.4}$$

The parameters can be specified in the input file as follows:

- Doping concentrations N_D, N_A are specified at `structure{ region{ doping{} } }` like

```
structure{
    ...
    region{
        ...
        doping{
            #constant{
            #    name = "donor_impurity"
            #    conc = 2.0e18               # cm^-3
            #}
            constant{
                name = "acceptor_impurity"
                conc = 5.0e19               # cm^-3
            }
        }
    }
}
```

- The degeneracy factors g_D, g_A and ionization energies $E_{D,i}^{ion}, E_{A,i}^{ion}$ are specified at *impurities{ }* like

```
impurities{
    donor{
        name = "donor_impurity"    # Si
        energy = 0.030             # ionization energy measured from the␣
↪conduction band edge. (fully ionized when -1000)
        degeneracy = 2             # degeneracy: 2 for n-type
    }
    acceptor{
        name = "acceptor_impurity" # Mg
        energy = 0.23              # ionization energy measured from the valence␣
↪band edge. 0.23 eV is taken from Kozodoy1999. (fully ionized when -1000)
        degeneracy = 4             # degeneracy: 4 for p-type
    }
}
```

Results

Spatially averaged hole density

Here we show the relation between $L = L_{\text{well}} = L_{\text{barrier}}$ and the spatially averaged hole densities.

The orange line is the result of Poisson equation ignoring the polarization fields, the blue line is the result of Poisson equation including the polarization fields, and the green line is the result of Schrödinger-Poisson equation including the polarization fields.

The corresponding hole density for **bulk** $Al_{0.2}Ga_{0.8}N$ with the same acceptor concentration 5.0×10^{19} [cm^{-3}] has been calculated as around $\mathbf{0.43} \times \mathbf{10^{18}}$ [cm^{-3}], so the hole density is improved in any case.

What we can also see is that the polarization field further enhances the acceptor ionization, while the quantization effect reduces it as L becomes smaller.

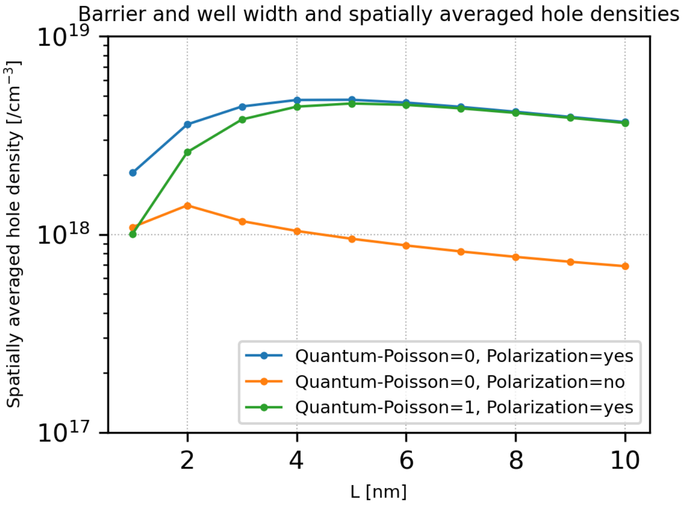


Figure 4.4.3.2: Barrier and well width L and spatially averaged hole densities.

Hole density/Ionized acceptor density distribution

Here we see the spatial distribution of the hole density and ionized acceptor density. We can confirm that the holes generated by the ionization of the acceptors in the barrier layers are accumulated into the well layers.

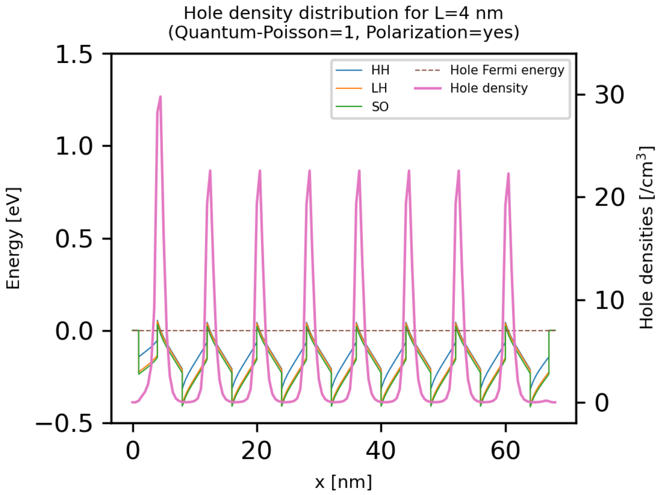


Figure 4.4.3.3: Hole density distribution calculated at L =4.0 nm by Schrödinger-Poisson equation including the polarization fields. The valence band edges are also displayed.

IQE estimation

Structure

The simulation region consists of the following structure:

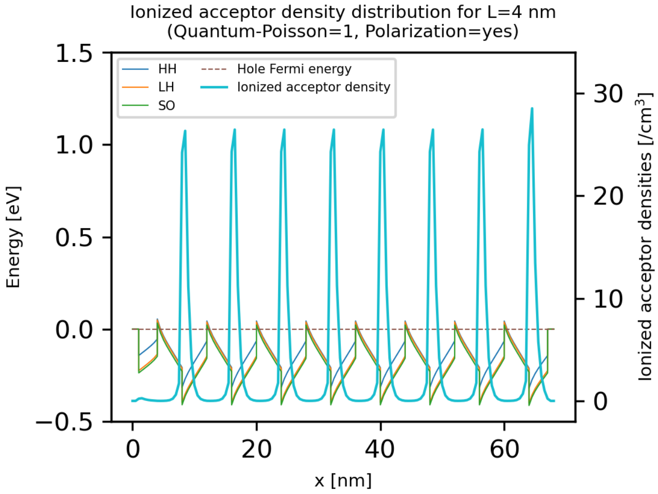


Figure 4.4.3.4: Ionized acceptor density distribution calculated at L =4.0 nm by Schrödinger-Poisson equation including the polarization fields. The valence band edges are also displayed.

Material	Thickness	Doping
n-Al$_{0.18}$Ga$_{0.82}$N	100 nm	8×10^{18} [cm^{-3}] (donor)
In$_{0.02}$Al$_{0.09}$Ga$_{0.89}$N - In$_{0.02}$Al$_{0.22}$Ga$_{0.76}$N 3-layer MQW	well: 2.5 nm, barrier: 15 nm	0 [cm^{-3}]
Al$_{0.24}$Ga$_{0.76}$N/Al$_{0.17}$Ga$_{0.83}$N 8-layer MQW	well: 4.0 nm, barrier: 4.0 nm	2×10^{19} [cm^{-3}] (acceptor)
p-Al$_{0.17}$Ga$_{0.83}$N as a p-contact layer | 20 nm		2×10^{19} [cm^{-3}] (acceptor)

The simulation result of this structure is compared with the structure where the p-region consists of bulk Al$_{0.20}$Ga$_{0.80}$N.

The electron blocking layer is not included here.

Bandedges

The following figures show the band edge profiles and the Fermi energies for the structures with (top) and without (bottom) SL. The width of the SL wells and barriers is set to $L = 4.0$ nm.

Scheme

The corresponding *run{ }* section is described as

```
run{
    strain{ }
    current_poisson{ }
}
```

Then *nextnano*++ solves the current equation and Poisson equation self-consistently after solving strain equation.

After the Current-Poisson equation has been converged, optoelectronic characteristics are calculated according to the specification in the section *classical{ }*.

For further details, please see *Optoelectronic characterization*.

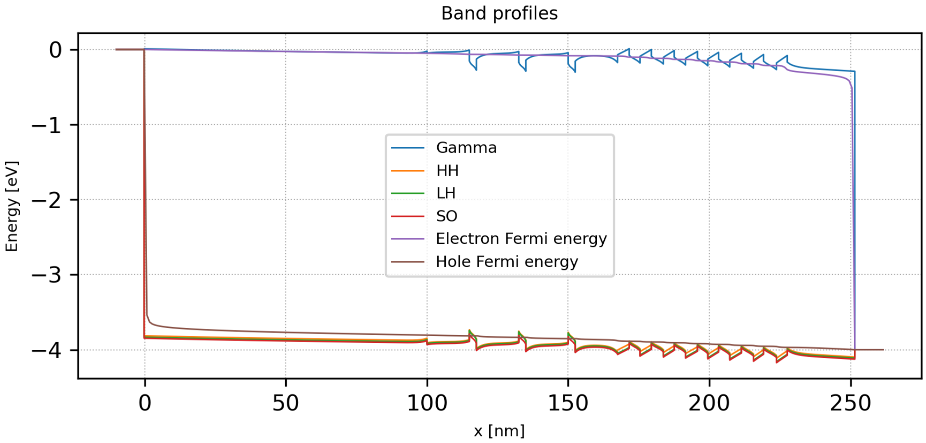


Figure 4.4.3.5: The band edges and Fermi levels for the structure with SL (bias=4.00 V, total current density=2.67×10^5 A/cm^2)

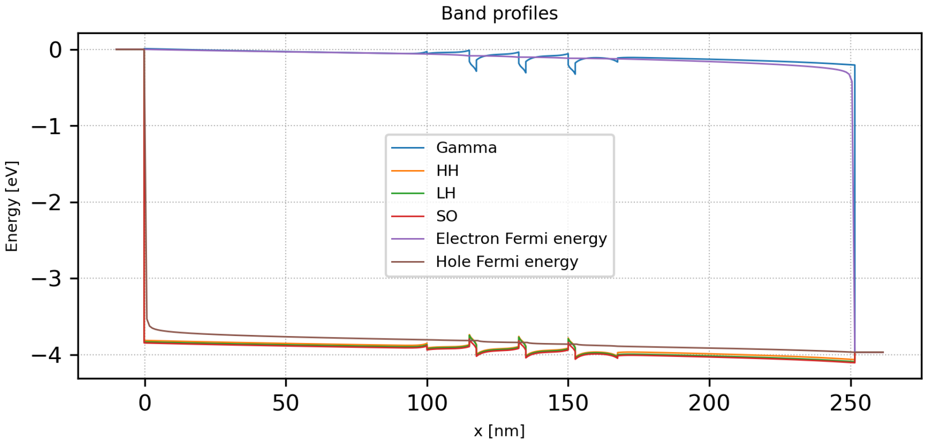


Figure 4.4.3.6: The band edges and Fermi levels for the structure with bulk p-region (bias=3.97 V, total current density=2.71×10^5 A/cm^2)

IQE result

The calculated IQEs with respect to the applied bias (left) and current density (right) are shown here. We can see that the IQE for the structure with SL, which is slightly smaller than that of bulk at the bias around 3.4 V, becomes superior to bulk for larger biases.

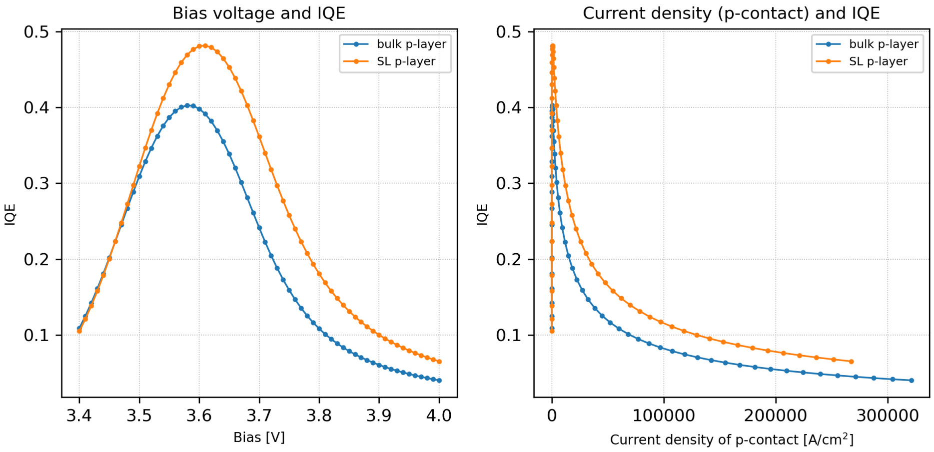


Figure 4.4.3.7: *Left*: Applied bias and IQE. *Right*: Current density at p-contact and IQE.

What can we do further?

By sweeping the simulation over the corresponding parameters, we can optimize the device strucutres such as L, number of SLs, or the Al content of the SL region, for example. Our open source python package *nextnanopy* is a powerful tool for this purpose.

Last update: nnnn/nn/nn

4.5 Quantum Mechanics

4.5.1 Parabolic Quantum Well (GaAs / AlAs)

Input files:

- *parabola_half-parabola_nnpp.in*

Scope:

This tutorial aims to reproduce figures 3.11 and 3.12 (pp. 83-84) of *[HarrisonQWWD2005]*, thus the following description is based on the explanations made therein.

General comments on the solutions of a parabolic potential

An ideal parabolic potential represents a “harmonic oscillator” which is described in nearly every beginner’s textbook on quantum mechanics. The eigenstates can be calculated analytically and are given by the following relationship:

$$E_n = \left(n - \frac{1}{2}\right) \hbar\omega_0 \tag{4.5.1.1}$$

where n = 1, 2, 3, . . .

One feature of a particle that is confined in such a well is that the energy levels are equally spaced by $\hbar\omega_0$ above the zero point energy of 1/2 $\hbar\omega_0$.

The eigenfunctions show an **even-odd** alternation which is also the case in symmetric, square quantum wells.

The eigenenergies can be measured experimentally by analyzing the optical transitions between the conduction and the valence band states, taking into account the selection rules (both states must have the same parity, see *tutorial on interband transitions*). For intersubband transitions, different selection rules apply (see *tutorial on intersubband transitions*). Such an experiment can be used to measure the conduction and valence band offsets because the curvature of the conduction and valence band edges (and thus the eigenstates) depends on the offsets.

More information on this can be found in *[Davies1998]*.

Parabolic quantum well: 10 nm $AlAs$ / 10 nm $AlGaAs$ / 10 nm $AlAs$

It is possible to grow parabolic quantum wells by continuously varying the composition of an alloy. Our structure consists of a 10 nm $Al_xGa_{1-x}As$ parabolic quantum well (the x alloy content varies parabolically) that is surrounded by 10 nm $AlAs$ barriers on each side. We thus have the following layer sequence: 10 nm $AlAs$ / 10 nm $Al_Ga_{1-x}As$ / 10 nm $AlAs$.

Bandeges

Figure 4.5.1.1 shows the conduction band edge and the three lowest electron wave functions (ψ) that are confined inside the parabolic quantum well. All other states are not confined anymore.

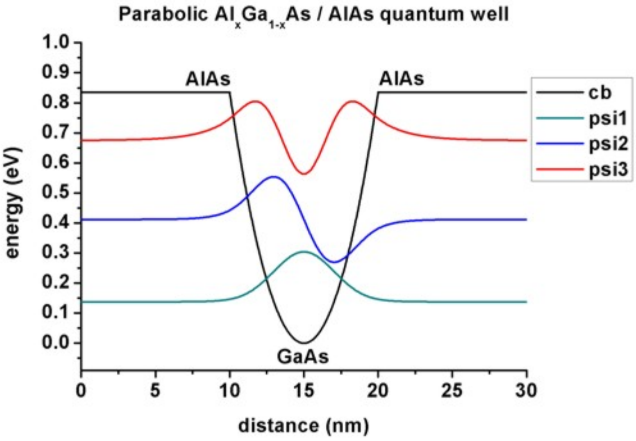


Figure 4.5.1.1: Calculated conduction band edge and the three lowest electron wave functions that are confined inside the parabolic QW. The energies were shifted so that the conduction band edge of $GaAs$ equals 0 eV.

The figure is in perfect agreement with Fig. 3.11 (p. 83) of *[HarrisonQWWD2005]*.

Technical details

The parabolic potential is specified by using a parabolic alloy profile.

```
structure{
    ...
    region{
        line{ x = [ -5.0 , 5.0 ] }
    }
    ternary_linear{
        name = "In(x)Ga(1-x)As"
        alloy_x = [0.0, 1.0]
        x = [ -5.0, 5.0]
    }
}
```

In agreement with Paul Harrison, we assumed a constant effective mass of 0.067 m_0 throughout the whole sample and further assumed the conduction band offset between $GaAs$ and $AlAs$ to be 0.83549 eV.

Output

The conduction band edge of the Gamma conduction band can be found here *bias_00000\bandedge_Gamma.dat*. The 1st column contains the position in units of [nm] and the 2nd column contains the conduction band edge in units of [eV].

The file *probabilities_shift_quantum_region_Gamma.dat* contains the eigenenergies and the squared wave functions (Psi^2). The 1st column contains the position in units of [nm]. Note that the Ψ_n^2 are shifted with respect to their energy E_n so that they can be nicely plotted into the conduction band profile.

amplitudes_shift_quantum_region_Gamma.dat contains the eigenenergies and the wave functions (Psi). The 1st column contains the position in units of [nm]. Note that Psin is shifted with respect to its energy En so that they can be nicely plotted into the conduction band profile.

Both *probabilities_shift_quantum_region_Gamma.dat* and *amplitudes_shift_quantum_region_Gamma.dat* contain the eigenenergies of the electron states in units of [eV]. Paul Harrison uses a 0.01 nm grid whereas we use the 0.01 nm grid only in the middle of the device (or 0.02 nm), but at the boundaries (i.e. from 0 nm to 5 nm and from 25 nm to 30 nm) we use a 0.1 nm grid to avoid long CPU times. The eigenvalues read:

n	E_n (*nextnano++*)	E_n (*[HarrisonQWWD2005]*)
1	0.13777630889948	0.1377751623
2	0.41211073419019	0.4121058503
2	0.67581828697139	0.6755025905

Making use of equation (4.5.1.1) with $\omega_0 = \sqrt{C/m^*}$ (m^* = effective mass, C = constant which is related to the parabolic potential $V(x) = 1/2Kx^2$), one can calculate E_n = $\hbar\omega_0$:

- $\hbar\omega_0$ = $2E_1$ - 0 eV = 0.276 eV
- $\hbar\omega_0$ = E_2 - E_1 = 0.274 eV
- $\hbar\omega_0$ = E_3 - E_2 = 0.264 eV

Obviously, due to the finite $AlAs$ barrier that we have employed, the higher lying states deviate slightly from the analytical results where infinite barriers have been assumed.

Figure 4.5.1.2 shows the eigenenergies for the confined states E_1, E_2 and E_3. As expected they are lying on a straight line because they are separated by $\hbar\omega_0$. The figure is in perfect agreement with Fig. 3.12 (p. 84) of *[HarrisonQWWD2005]*.

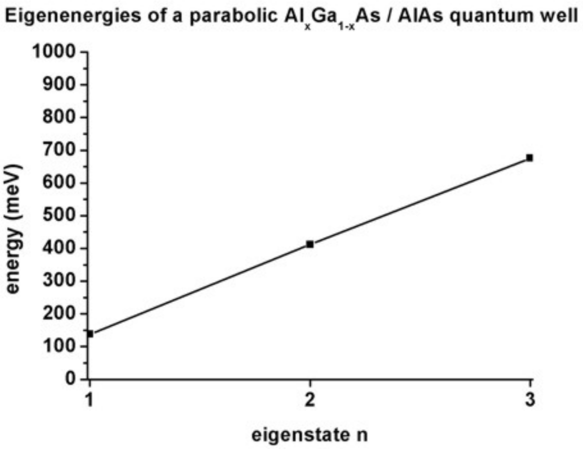


Figure 4.5.1.2: Eigenenergies for the three lowest states.

Matrix elements

The following matrix elements have been calculated:

- `overlap_integrals{}`: This spatial overlap matrix elements $\langle\psi_f|\psi_i\rangle$ simply returns the Kronecker delta as expected, because the wave functions are orthogonal.
- `momentum_matrix_elements{}`: $\langle\psi_f|p_x|\psi_i\rangle$ (see *Tutorial on intraband transition*)
- `dipole_moment_matrix_elements{}`: $\langle\psi_f|x|\psi_i\rangle$ (see *Tutorial on intraband transition*)

"Infinite" (30 eV) parabolic QW confinement for GaAs

Inputfile: *1DGaAs_ParabolicQW_infinite.in*

Figure 4.5.1.3 shows the eigenstates of a parabolic quantum well ($GaAs$) where the confinement is assumed to be 30 eV. Now up to 37 eigenstates are confined in the quantum well (grid resolution: 0.025 nm inside the well, 0.05 nm inside the barrier).

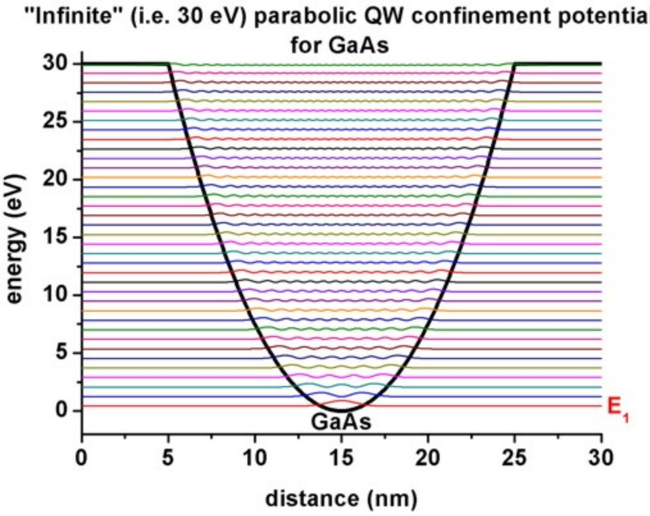


Figure 4.5.1.3: Calculated conduction band profile and probability densities (Ψ^2) for eigenstate n (n = 1, 2, ..., 37).

Figure 4.5.1.4 shows the energies of the 37 confined electron states as a function of eigenstate n. As expected, the curve shows a linear dependence because the eigenstates are equally spaced by $\hbar\omega_0$ = 0.826 eV (where we used E_n = (n - 1/2) $\hbar\omega_0$).

$\hbar\omega_0$ = 2 E_1 - 0 eV = 0.8261 eV	$E_1/(2E_1)$ = 0.5000
$\hbar\omega_0$ = E_2 - E_1 = 0.8260 eV	$E_2/(2E_1)$ = 1.4999
$\hbar\omega_0$ = E_3 - E_2 = 0.8260 eV	$E_2/(2E_1)$ = 2.4997
$\hbar\omega_0$ = E_4 - E_3 = 0.8259 eV	$E_4/(2E_1)$ = 3.4994
$\hbar\omega_0$ = E_5 - E_4 = 0.8259 eV	$E_5/(2E_1)$ = 4.4991
$\hbar\omega_0$ = E_6 - E_5 = 0.8258 eV	$E_6/(2E_1)$ = 5.4987
$\hbar\omega_0$ = E_7 - E_6 = 0.8258 eV	$E_7/(2E_1)$ = 6.4982
$\hbar\omega_0$ = E_8 - E_7 = 0.8257 eV	$E_8/(2E_1)$ = 7.4978

Still, due to the "infinite" barrier of 30 eV (which is still a finite barrier) that we have employed, the higher lying states deviate slightly from the analytical results where infinite barriers have been assumed.

One should bear in mind that the energy level spacing of such parabolic quantum wells is inversely proportional to both the well width and the square root of the effective mass.

It is also interesting to look at the momentum matrix elements, i.e. to investigate the probability for intersubband transitions. The relevant output is contained in these two files:

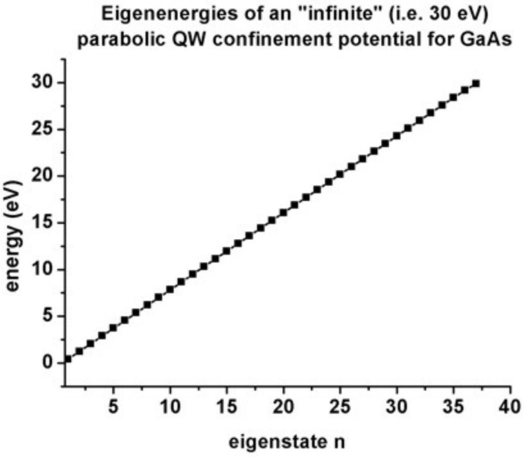


Figure 4.5.1.4: Eigenenergies of 37 eigenstates for an infinite parabolic QW.

- *\bias_00000\Quantum\dipole_moment_matrix_elements_quantum_region_Gamma_100.txt* - p_x
- *\bias_00000\Quantum\momentum_matrix_elements_quantum_region_Gamma_100.txt* - x

From the calculated oscillator strengths it can be seen that only transitions from one level to the neighboring levels (+1 and -1) are **allowed**. Because in the case of a harmonic oscillator the momentum operator is proportional to the sum of the creation and the annihilation operators, thus only states can couple that have different occupation numbers with the difference equal to 1.

"Infinite" (30 eV) half-parabolic QW confinement for GaAs

Input file: *1DGaAs_ParabolicQW_infinite_half_nnpp.in*

Figure 4.5.1.5 shows the eigenstates when taking only the right half of the parabolic quantum well ($GaAs$) that has been calculated above. The confinement is 30 eV on the right and infinite confinement on the left (Dirichlet boundary conditions). Now only 18 eigenstates are confined in the quantum well, i.e. half the number of the eigenvalues compared with the full parabolic QW (grid resolution: 0.025 nm inside the well, 0.05 nm inside the barrier). The figure shows the conduction band profile and the square of the wave functions (Ψ_n^2) for eigenstate n (n = 1, 2, . . . , 18).

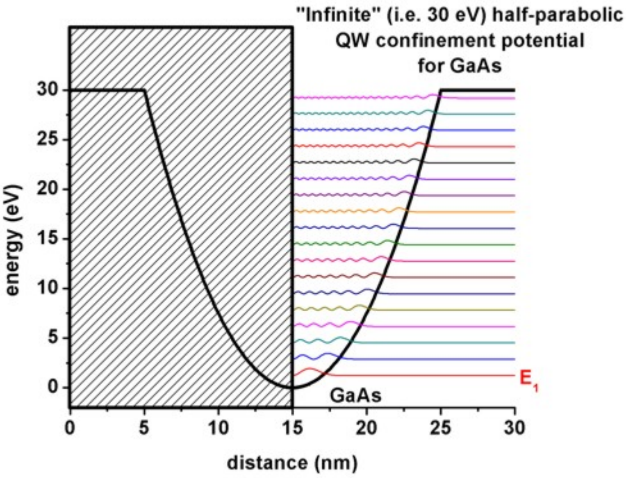


Figure 4.5.1.5: Calculated conduction band profile and probability densities (Ψ^2) for all confined eigenstates in an half-parabolic potential.

Again, the eigenstates are equally spaced. However, the separation energy is now twice as large as before, i.e. $\hbar\omega_0$ = 2 · 0.826 eV = 1.65 eV.

The ground state energy this time is given by: E_1 = 3/2 $\hbar\omega_0$/2.

- $\hbar\omega_0$ = 4/3 E_1 - 0 eV = 1.639 eV
- $\hbar\omega_0$ = E_2 - E_1 = 1.647 eV
- $\hbar\omega_0$ = E_3 - E_2 = 1.648 eV
- $\hbar\omega_0$ = E_4 - E_3 = 1.648 eV

It is also interesting to look at the momentum matrix elements, i.e. to investigate the probability for intersubband transitions. The relevant output is contained in these two files:

- *\bias_00000\Quantum\dipole_moment_matrix_elements_quantum_region_Gamma_100.txt* - p_x
- *\bias_00000\Quantum\momentum_matrix_elements_quantum_region_Gamma_100.txt* - x

Conclusion

We note that also more realistic parabolic quantum wells can be calculated with *nextnano*++. Assuming that the alloy profile is parabolic,

- strain can be included (the strain tensor depends on the alloy profile),
- as well as effective masses that depend on the alloy profile,
- an 8-band k.p model (necessary to get correct intersubband transition energies)
- and bowing parameters (especially important for $AlGaAs$).

All these features are automatically included in the *nextnano*++ code.

Last update: 27/05/2025

4.5.2 Triangular well

In this tutorial we calculate the Schrödinger equation for a triangular well and compare the results with the analytic solution.

The related input files are followings:

- *1DGaAs_triangular_well_nn*.in*

Structure

A triangular well consists of a potential with a constant slope that is bound at one side by an infinite barrier.

For $x < 0$ nm we have an infinite barrier. In our case it is represented by a huge conduction band offset of 100 eV to avoid any wave function penetration into the barrier.

For $x > 0$ nm we have a linear potential of $V(x) = eFx$.

$V(x)$ describes a charge e in an electric field F where the product eF is assumed to be positive.

Comparison of nextnano++ and the analytic solution

The Schrödinger equation for the transverse component of the electronic wave function has the following form inside the well:

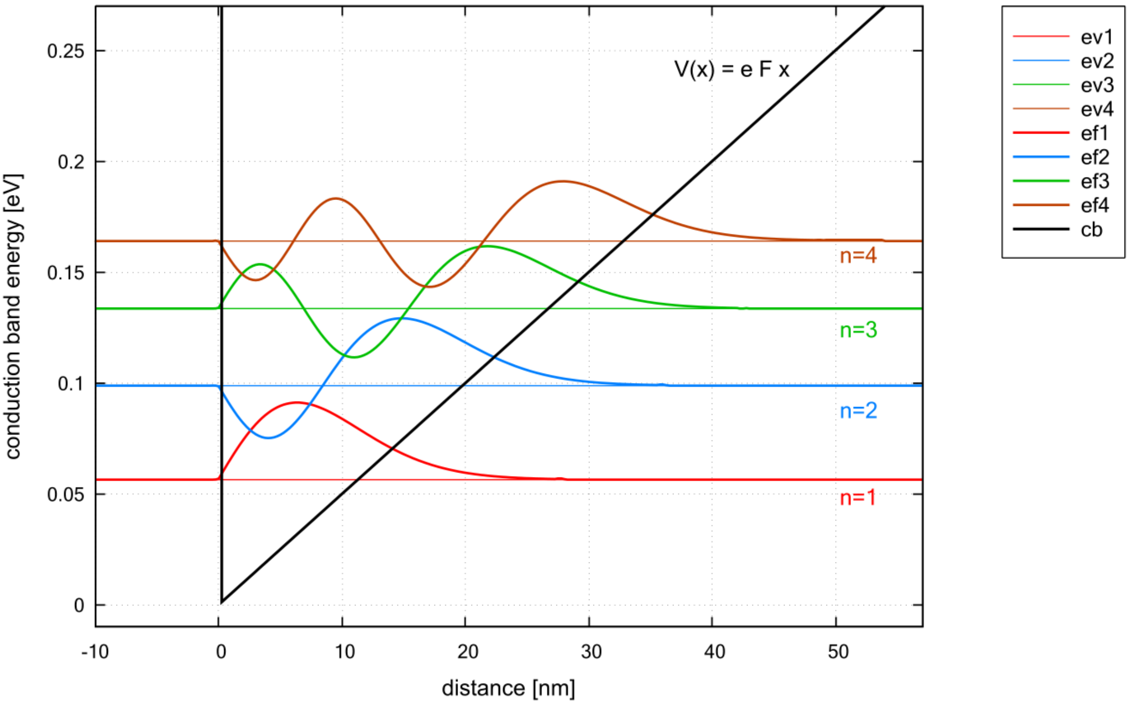
$$\left[-\frac{\hbar^2}{2m^*}\frac{d^2}{dx^2} + eFx \right] \psi(x) = E\psi(x)$$

Usually one applies Dirichlet boundary conditions at $x = 0$ nm so that $\psi(x = 0) = 0$ in order to represent an infinite barrier, i.e. the high barrier prevents significant penetration of electrons into the barrier region.

In our case, we apply Neumann (or Dirichlet) boundary conditions at $x = -10$ nm and $x = 150$ nm and let the infinite barrier be represented by the huge conduction band offset of 100 eV. Then, both boundary conditions lead to the same eigenenergies for the lowest eigenstates.

The Schrödinger equation can be simplified by introducing suitable new variables and thus reduces to the Stokes or Airy equation. Its solutions, the so-called Airy functions, are discussed in most textbooks, see for example:

- *The Physics of Low-Dimensional Semiconductors - An Introduction*, John H. Davies, Cambridge University Press (1998)



The figure shows the conduction band edge (black line) which is represented by a triangular potential well $V(x) = eFx$. Also shown are the four lowest energy levels and corresponding wave functions. The electric field that has been applied is $F = 5$ [MV/m], i.e. $0.05V/10$ nm. The effective electron mass has the value $0.067m_0$ (GaAs).

One can see that the distance between the energy levels decreases with increasing n because the quantum well width gets larger for higher energies. Note that in a parabolic well, the energy levels are equally spaced whereas in an infinitely deep square well, the energy level separation increases with increasing energy.

The eigenvalues of the Airy equation can be calculated using the formula:

$$E_n = c_n \left[\frac{eF\hbar^2}{2m^*} \right]^{1/3}$$

(The units of E_n in this equation are [J].)

The lowest eigenvalue has the value $c_1 = 2.338$.

For large n, c_n can be approximated by the following equation which can be derived from WKB theory (named after Wentzel, Kramers and Brillouin):

$$c_n \simeq \left[\frac{3}{2}\pi \big(n - \frac{1}{4} \big) \right]^{2/3}$$

The eigenvalues for the lowest four eigenstates are in very good agreement with the analytic results:

	nextnano++ eigenvalue	calculated eigenvalue	c_n (exact)	c_n (approximated)
n = 1	0.05647	0.05664	$c_1 = 2.338$	$c_1 = 2.320251$
n = 2	0.09887	0.09889	c_2	$c_2 = 4.081810$
n = 3	0.13358	0.13365	c_3	$c_3 = 5.517164$
n = 4	0.16426	0.16435	c_4	$c_4 = 6.784455$

The triangular potential is not symmetric in x, thus the wave functions lack the even or odd symmetry that one obtains for the infinitely deep square well.

The triangular well model is useful because it can be used to approximate the (idealized) triangularlike shape near a heterojunction formed by the discontinuity of the conduction band and an electrostatic field of electrons or remote ionized impurities.

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.5.3 — FREE — Double Quantum Well

Input files:

- *DoubleQuantumWell_6_nm_nnpp.in*

This tutorial calculates the energy eigenstates of a double quantum well. This aims to reproduce two figures (Figs. 3.16, 3.17, p. 92) of Paul Harrison's excellent book "Quantum Wells, Wires and Dots" (Section 3.9 "The Double Quantum Well"), thus the following description is based on the explanations made therein. *We are grateful that the book comes along with a CD so that we were able to look up the relevant material parameters and to check the results for consistency.*

To generate the input files for various thicknesses and some of the plots the following *nextnanomat* features are used:

- Template tab
- Postprocessing feature

It is recommended to read the documentation about these features of the graphical user interface *nextnanomat* **before** starting this tutorial.

Structure: AlGaAs / 6 nm GaAs / AlGaAs / 6 nm AlGaAs / AlGaAs

Our symmetric double quantum well consists of two 6 nm GaAs quantum wells, separated by a $Al_{0.2}$ $Ga_{0.8}$ As barrier and surrounded by 20 nm $Al_{0.2}$ $Ga_{0.8}$ As barriers on each side. We thus have the following layer sequence: **20 nm $Al_{0.2}$ $Ga_{0.8}$ As** / 6 nm GaAs / **$Al_{0.2}$ $Ga_{0.8}$ As** / 6 nm GaAs / **20 nm $Al_{0.2}$ $Ga_{0.8}$ As**. (The barriers are printed in bold.)

In this tutorial, we demonstrate the following two examples:

1. we set the thickness of the $Al_{0.2}$ $Ga_{0.8}$ As barrier that separates the two quantum wells **4 nm** and calculate **the lowest two eigenstates**.
2. we vary the thickness of the barrier layer **from 1 nm to 14 nm** fixing the width of the quantum well (6 nm). Then we calculate **the lowest two eigenstates** for each case and see the barrier-width dependency of their eigenenergies.

We also explain where the relevant output files are in.

Material Parameters

The material parameters are given in `database_nn*.in` but we can also redefine them manually in input files. In this tutorial, we redefine parameters so that they are the same as the section 3.9 of Paul Harrison's book "Quantum Wells, Wires and Dots".

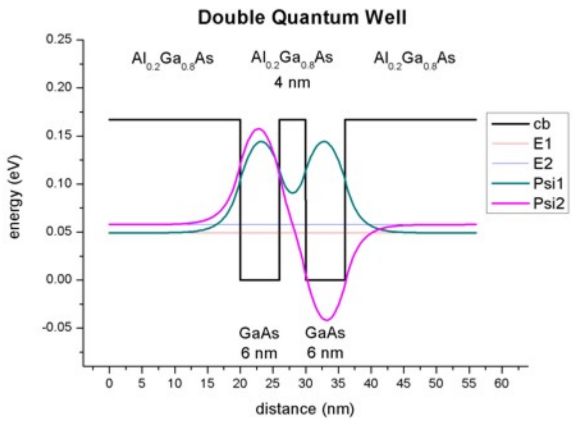
conduction band offset	$Al_{0.2}$ $Ga_{0.52}$ As / GaAs	0.167 eV
conduction band effective mass	$Al_{0.2}$ $Ga_{0.52}$ As	0.084 m_0
conduction band effective mass	GaAs	0.067 m_0

Results

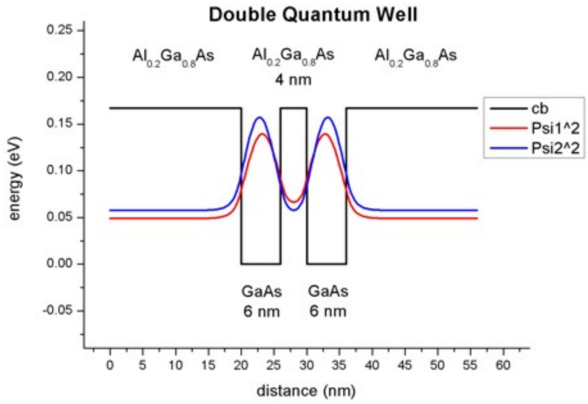
1. barrier width = 4 nm

- The following figure shows the **conduction band edge** and **wave functions** that are confined inside the wells with barrier width = 4 nm.

(Note that the energies were shifted so that the conduction band edge of GaAs equals 0 eV.)



- The wave functions form a symmetric and an anti-symmetric pair. The symmetric one is lower in energy than the anti-symmetric one. The plot is in excellent agreement with Fig. 3.17 (page 92) of Paul Harrison's book "Quantum Wells, Wires and Dots".
- For comparison, the following figure shows for the same structure as above, the square of the wave function rather than ψ only.



Output

a. The conduction band edge of the Gamma conduction band can be found here:

bias_00000/bandedge_Gamma.dat

b. This file contains the eigenenergies of the two lowest eigenstates. The units are [eV].

bias_00000/Quantum/wf_energy_spectrum_quantum_region_Gamma_0000.dat

These are the comparison of eigenvalues:

	nextnano++	Harrison's book
ground state energy [eV]	0.04920	0.04912
first excited state energy [eV]	0.05779	0.05770

c. This file contains the eigenenergies and the wave functions (ψ):

bias_00000/Quantum/wf_amplitudes_shift_quantum_region_Gamma_0000.dat

This file contains the eigenenergies and the squared wave functions (ψ^2):

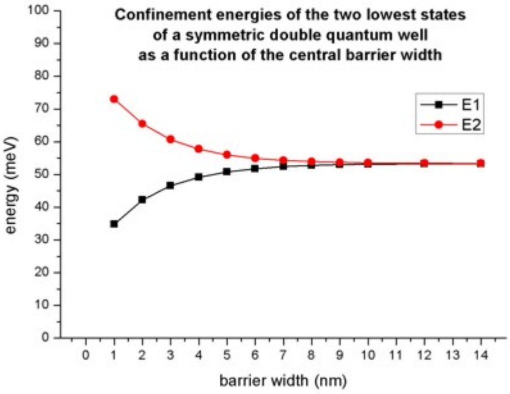
bias_00000/Quantum/wf_probabilities_shift_quantum_region_Gamma_0000.dat

The subscript `_shift` indicates that ψ^2 and ψ are shifted by the corresponding energy levels.

a. and c. can be used to plot the data as shown in the figures above.

2. barrier width = 1 ~ 14 nm

- Here, we varied the thickness of the $Al_{0.2}$ $Ga_{0.8}$ As barrier layer from 1 nm to 14 nm fixing the width of the quantum well (6 nm). We calculated the lowest two eigenstates and show their eigenvalues for each barrier width in the following figure (generated with the :Postprocessing feature of *nextnanomat*).



- If the separation between the two quantum wells is large, the wells behave as two independent single quantum wells having the identical ground state energies. The interaction between the energy levels localized within each well increases once the distance between the two wells decreases below 10 nm. One state is forced to higher energies and the other to lower energies. (Here, the electron spins align in an "anti-parallel" arrangement in order to satisfy the Pauli exclusion principle.)
- This is analogous to the hydrogen molecule where the formation of a pair of bonding and anti-bonding orbitals occurs once the two hydrogen atoms A and B are brought together.

$$\psi_{\text{bonding}} = \frac{1}{\sqrt{2}}\psi_A + \psi_B \quad \text{(lower energy)}$$
$$\psi_{\text{antibonding}} = \frac{1}{\sqrt{2}}\psi_A - \psi_B \quad \text{(higher energy)}$$

- Again, the plot is in excellent agreement with Fig. 3.16 (page 92) of Paul Harrison's book "Quantum Wells, Wires and Dots".

Output

The energy values were taken from the same file as before:

> *bias_00000/Quantum/wf_energy_spectrum_quantum_region_Gamma_0000.dat*

For example, the values for the 1 nm barrier read:

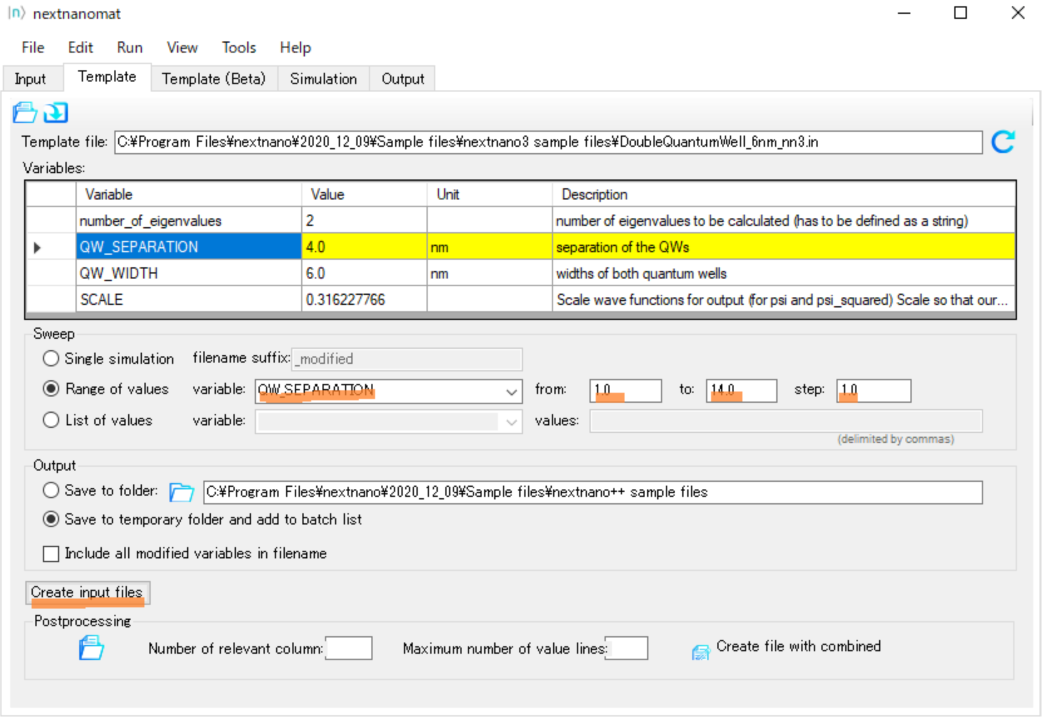
	nextnano++	Harrison's book
ground state energy [eV]	0.03476	0.03470
first excited state energy [eV]	0.07298	0.07290

The values for the 14 nm barrier read:

	nextnano++	Harrison's book
ground state energy [eV]	0.05332	0.05323
first excited state energy [eV]	0.05338	0.05329

Tip: Sweeping

A sweep over the **thickness** of the $Al_{0.2}$ $Ga_{0.8}$ As barrier layer, i.e. the variable `%QW_SEPARATION`, can easily be done by using *nextnanomat*'s Template feature. The following screenshot shows how this can be done. Go to "Template", open input file, select "Range of values", select "QW_SEPARATION", click on "Create input files", go to "Run and start your simulations.



Template file: C:¥Program Files¥nextnano¥2020_12_09¥Sample files¥nextnano3 sample files¥DoubleQuantumWell_6nm_nn3.in

Variables:

Variable	Value	Unit	Description
number_of_eigenvalues	2		number of eigenvalues to be calculated (has to be defined as a string)
QW_SEPARATION	4.0	nm	separation of the QWs
QW_WIDTH	6.0	nm	widths of both quantum wells
SCALE	0.316227766		Scale wave functions for output (for psi and psi_squared) Scale so that our...

Sweep

○ Single simulation filename suffix: _modified

● Range of values variable: QW_SEPARATION from: 1.0 to: 14.0 step: 1.0

○ List of values variable: values:

(delimited by commas)

Output

○ Save to folder: C:¥Program Files¥nextnano¥2020_12_09¥Sample files¥nextnano++ sample files

● Save to temporary folder and add to batch list

☐ Include all modified variables in filename

This tutorial also exists for *nextnano*^{3}.

Last update: nnnn/nn/nn

4.5.4 — EDU — Orbitals of the Hydrogen Atom

Header

Files for the tutorial located in *nextnano++\examples\education*

- *Orbitals_Hydrogen_3D_nnp.in*

Scope of the tutorial:

- Schrödinger equation
- Coulomb potential
- Numerical accuracy

Main adjustable parameters in the input file:

- regularizing parameter `$eta`
- radius of the simulation domain `$pos_end`
- positions of the grid definitions: `$pos_fine`, `$pos_medium`, and `$pos_coarse`
- grid spacings: `$grid_coarse`, `$grid_medium`, and `$grid_fine`

Relevant output files:

- *bias_00000\bandedge_Gamma_1d_z.dat*
- *bias_00000\bandedge_Gamma_2d_yz.dat*
- *bias_00000\Quantum\energy_spectrum_quantum_region_Gamma_00000.dat*
- *bias_00000\Quantum\amplitude_quantum_region_Gamma_XXXX.fld*

Introduction

This tutorial demonstrates use of *nextnano++* in computing orbitals of a Hydrogen atom. As orbitals and their energies can be obtained analytically for the Hydrogen atom (see. *[LeviAQM2006]*), this tutorial serves also as a playground for exploration of numerical limits of 3D simulations on small computers.

In this tutorial we assume that the proton is set in the origin of the coordinate system and the electron is confined by the Coulomb potential arising from the presence of the proton

$$\phi_C(r) = \frac{1}{4\pi\varepsilon_0}\frac{q}{r}, \tag{4.5.4.1}$$

where q is the elementary charge, r is a distance from the proton, and ε_0 is the permittivity of vacuum. This potential can be defined directly in the input file using coordinates as

$$r = \sqrt{x^2 + y^2 + z^2}$$

Note that the length is given in (nm) in the input file. The Schrödinger equation for the system is given by

$$\left[-\frac{\hbar^2}{2m_0}\nabla^2 - q\,\phi_C(x,y,z)\right]\Psi(x,y,z) = E\,\Psi(x,y,z), \tag{4.5.4.2}$$

where m_0 is the mass of a free electron and $\hbar$ is Dirac's constant.

This equation can be solved by *nextnano++* within *1-band model*. To do so one needs to take care about:

- definition of a convenient vacuum "material",
- grid spacing and size of the simulation domain,
- infinity of the potential at the origin of the coordinate system.

Preparing the simulation

Convenient vacuum "material"

Let us define vacuum material *modifying existing material*, e.g., GaAs. As energy dispersion of electron in vacuum is isotropic parabola, 1-band model for conduction band for zincblende crystals can be parametrized to describe vacuum. The effective mass corresponding to the free electron is equal to 1. Assuming that total energy of stationary electron is equal 0 eV, we set the minimum of the band to be zero. We do it safely by setting all: band gap, band offset, and spin-orbit splitting to zero.

```
database{ # gallium arsenide turned into vacuum at 0 eV
    binary_zb{
        name = GaAs # same as the substrate to neglect strain
        conduction_bands{
            Gamma{
                bandgap = 0
                mass    = 1
            }
        }
        valence_bands{
            bandoffset = 0
            delta_SO = 0
        }
    }
}
```

Note that we also turn off temperature dependence of the band gap so that Varshi formula is not applied and does not shift the minimum. Choice of the crystal orientation and substrate are arbitrary in this simulation. We set some, because the solver requires them. All strain effects are ignored as `strain{ }` is not called in the `run{ }` section - there is no strain in the vacuum.

```
global{
    simulate3D{}
    crystal_zb{
        x_hkl = [1, 0, 0]
        y_hkl = [0, 1, 0]
    }
    substrate{ name = "GaAs" }

    temperature_dependent_bandgap = no
    temperature = 4.0 # Kelvin
}
```

The vacuum is ready!

The grid and simulation domain

Keeping in mind that these computations are meant to be held on desktop computers or laptops, the biggest limitation comes from the number of grid points that one can include in the simulation, as it directly impacts RAM needed for the simulation. The simulation grid should be defined to have possibly low number of grid points while keeping most of them the center of the atom to properly represent the potential and orbitals of interest.

In the input file for this tutorial we defined such a grid to keep it fine nearby the center of the atom and gradually coarser while going outwards (basic example on how to define such grids can be found *here*). For that purpose we use 6 *variables* to have quite flexible control over the grid spacing (`$grid_coarse`, `$grid_medium`, and `$grid_fine`) and positions where these spacings begin to apply (`$pos_fine`, `$pos_medium`, and `$pos_coarse`). The last parameter of the grid, `$pos_end`, is defining the size of the entire simulation domain.

All of these parameters together are determining number of grid points, hence, how much memory the simulation will require and how much time it will take to have the Schrödinger equation solved.

```
#spacing
$grid_coarse = 0.1
$grid_medium = 0.05
$grid_fine = 0.005
```

```
grid{
    xgrid{
       line{ pos =-$pos_end       spacing = $grid_coarse }
       line{ pos =-$pos_coarse    spacing = $grid_coarse }
       line{ pos =-$pos_medium    spacing = $grid_medium }
       line{ pos =-$pos_fine      spacing = $grid_fine   }
       line{ pos = 0              spacing = $grid_fine   }
       line{ pos = $pos_fine      spacing = $grid_fine   }
       line{ pos = $pos_medium    spacing = $grid_medium }
       line{ pos = $pos_coarse    spacing = $grid_coarse }
       line{ pos = $pos_end       spacing = $grid_coarse }
      }
    ygrid{
        line{ pos =-$pos_end      spacing = $grid_coarse }
       line{ pos =-$pos_coarse    spacing = $grid_coarse }
       line{ pos =-$pos_medium    spacing = $grid_medium }
       line{ pos =-$pos_fine      spacing = $grid_fine   }
       line{ pos = 0              spacing = $grid_fine   }
       line{ pos = $pos_fine      spacing = $grid_fine   }
       line{ pos = $pos_medium    spacing = $grid_medium }
       line{ pos = $pos_coarse    spacing = $grid_coarse }
       line{ pos = $pos_end       spacing = $grid_coarse }
      }
    zgrid{
       line{ pos =-$pos_end       spacing = $grid_coarse }
       line{ pos =-$pos_coarse    spacing = $grid_coarse }
       line{ pos =-$pos_medium    spacing = $grid_medium }
       line{ pos =-$pos_fine      spacing = $grid_fine   }
       line{ pos = 0              spacing = $grid_fine   }
       line{ pos = $pos_fine      spacing = $grid_fine   }
       line{ pos = $pos_medium    spacing = $grid_medium }
       line{ pos = $pos_coarse    spacing = $grid_coarse }
       line{ pos = $pos_end       spacing = $grid_coarse }
      }
  }
```

Note

In general, the accuracy increases with reduction of the grid, unless machine precision begins to limit accuracy of derivatives.

Regularized Coulomb potential

The Coulomb potential itself is posing a problem in this simulation as it introduces infinity, which gets more and more severe when the grid gets finer around it. One way to remove this infinity is to regularize the potential (4.5.4.1) introducing a regularizing parameter η aiming at removing the infinity.

$$\phi_C(r) \rightarrow \phi_\eta(r) = \frac{1}{4\pi\varepsilon_0}\frac{q}{\sqrt{x^2+y^2+z^2+\eta^2}} \tag{4.5.4.3}$$

Assuming that one cares about accuracy for the ground state in the Hydrogen atom, the η should not modify the potential much outside a volume that is negligibly small in comparison to the orbital 1s. Otherwise the regularization will notably affect eigenenergy and shape of this orbital. We chose $\eta = 3.5 \cdot 10^{-3}$ nm, which is around one order of magnitude smaller than the Bohr radius, $a_B \approx 5.3 \cdot 10^{-2}$ nm.

This potential we define inside the *import{ }* group using `$eta` (see top of the input file) as a variable corresponding to η.

```
$e = 1 #eV
$eps = 55.263E-3 #e^2eV^(-1)nm^(-1)
$pi = 3.1415

import{
    analytic_function{
        name = "Potential"
        function = "(1/(4*$eps*$pi))*( $e/(sqrt((x)^2 + (y)^2 + (z)^2 + $eta^2)) )"
        label = potential_label
    }
output_imports{}   # output all imported data including scale factor.
}
```

The potential is included as an *initialization of Poisson equation*, which further **is not** solved.

```
   poisson{
      import_potential{ import_from = "Potential" }
      output_potential{}
    }
```

Results and Discussion

Orbitals s1 and s2

Let us have a look at s orbitals that are expected to be the most affected by regularization of the Coulomb potential. One can easily compare these orbitals with literature *[LeviAQM2006]* as their amplitudes have symmetry of a sphere. To do so, one can define 1D sections through the center of the atom in the input file using *section{ }* nested group and plot numerical amplitudes together with the ones derived analytically

$$\begin{aligned}\Psi_{1s}(r) &= 2\left(\frac{1}{a_B}\right)^{3/2}\exp\left(-\frac{r}{a_B}\right)\left(\frac{1}{4\pi}\right)^{1/2},\\ \Psi_{2s}(r) &= 2\left(\frac{1}{2a_B}\right)^{3/2}\left(1-\frac{r}{2a_B}\right)\exp\left(-\frac{r}{2a_B}\right)\left(\frac{1}{4\pi}\right)^{1/2}.\end{aligned} \tag{4.5.4.4}$$

Such comparison of s1 and s2 orbitals obtained with both methods is shown in Figure 4.5.4.1 a) and b), respectively.

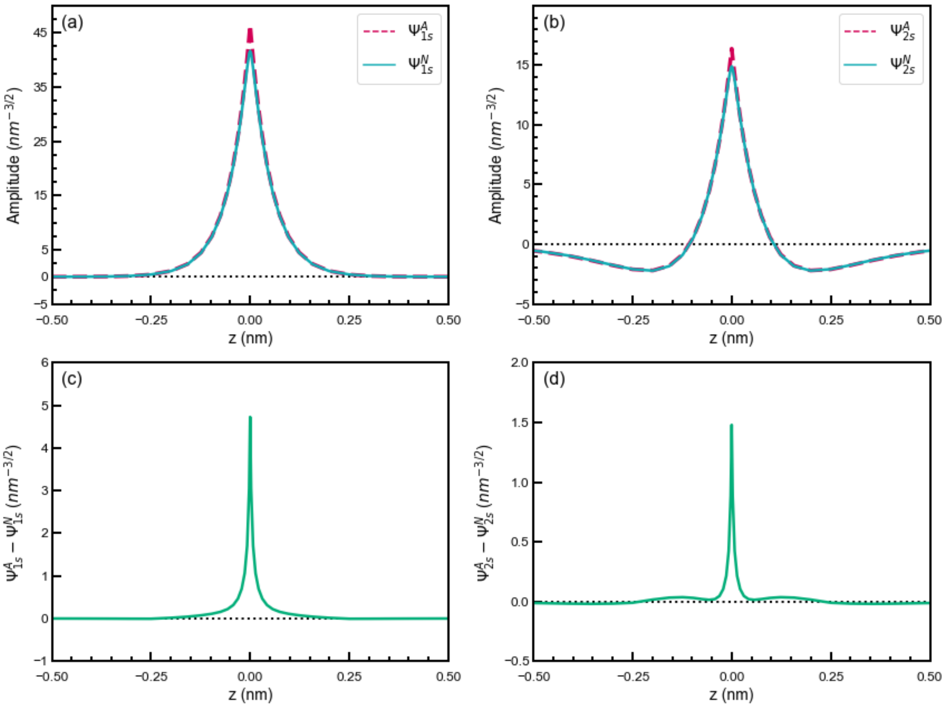


Figure 4.5.4.1: a), b) Comparison of 1s and 2s orbitals obtained from analytical formulas and numerical simulation. c), d) Difference between the analytical and numerical wave functions of s1 and s2 orbitals, respectively.

As seen in the Figure 4.5.4.1 c) and d), the most significant loss of accuracy is present near the center of the atom, where regularization has the biggest effect. It reaches approximately 10% of the maximum amplitude at the zero coordinate, and falls below 1% at radius smaller than 0.05 nm.

Regularized potential

Investigating the potential (see Figure 4.5.4.2) one can see that regularization impacts the potential in the order of magnitude 10^{-2} - 10^{-1} V at the distance near the Bohr radius.

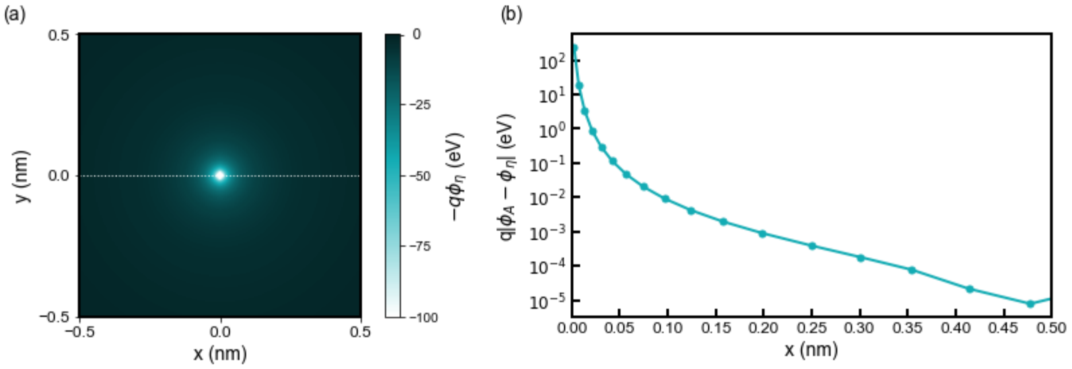


Figure 4.5.4.2: (a) shows the Coulomb energy distribution. (b) is the difference between the numerical Coulomb potential (V_η) calculated by *nextnano++* and the analytical potential (V_A) at $y = 0$, $z = 0$ along the white dash line in (a).

For that reason, a well-computed first orbital can be expected to have the eigenenergy deviating from the analytical value by approximately 10^{-2} - 10^{-1} eV.

Energies

Accordingly, the effect can be best seen by comparing analytical energies of orbitals *[LeviAQM2006]*

$$E_a = \frac{-m_0 q^4}{2(4\pi\varepsilon_0)^2\hbar^2}\frac{1}{n^2}, \tag{4.5.4.5}$$

where n is the principal quantum number, with computation using a fine grid (`$grid_medium=0.01`); see columns 4-6 in Table 4.5.4.1. Here the difference of energies is overestimated by approximately 0.19 eV for the ground state, which corresponds to additional potential energy introduced by the regularization.

Table 4.5.4.1: Eigenenergies obtained using analytical formula (E_a), from the simulation with fine grid (E_{fine}) and coarser grid (E_{fine}) to reduce simulation time.

$Orbital$	n	l	E_a (eV) (*)	E_{fine} (eV)	$\lvert E_{fine} - E_a\rvert$ (eV)	E_{fast} (eV)	$\lvert E_{fast} - E_a\rvert$ (eV)
$1s$	1	0	-13.606	-13.420	1.86×10^{-2}	-13.605	1.26×10^{-3}
$2s$	2	0	-3.401	-3.381	1.96×10^{-2}	-3.424	2.25×10^{-3}
$2p$	2	1	-3.401	-3.402	1.44×10^{-3}	-3.450	4.85×10^{-2}
$2p$	2	1	-3.401	-3.402	1.44×10^{-3}	-3.450	4.85×10^{-2}
$2p$	2	1	-3.401	-3.402	1.44×10^{-3}	-3.450	4.85×10^{-2}
$3s$	3	0	-1.512	-1.506	5.81×10^{-3}	-1.525	1.34×10^{-2}
$3p$	3	1	-1.512	-1.512	4.40×10^{-4}	-1.532	1.96×10^{-2}
$3p$	3	1	-1.512	-1.512	4.40×10^{-4}	-1.532	1.96×10^{-2}
$3p$	3	1	-1.512	-1.512	4.40×10^{-4}	-1.532	1.96×10^{-2}
$3d$	3	2	-1.512	-1.513	5.49×10^{-4}	-1.528	1.63×10^{-2}
$3d$	3	2	-1.512	-1.513	5.49×10^{-4}	-1.528	1.63×10^{-2}
$3d$	3	2	-1.512	-1.512	3.92×10^{-5}	-1.521	8.91×10^{-3}
$3d$	3	2	-1.512	-1.512	3.92×10^{-5}	-1.521	8.91×10^{-3}
$3d$	3	2	-1.512	-1.512	3.92×10^{-5}	-1.521	8.91×10^{-3}

Such fine simulation, however, can take more than half a day to finish. Interesting results can be also obtained using coarser grid, therefore, within shorter simulation runs (couple of minutes). Columns 4 and 7-8 of Table 4.5.4.1 show that it is possible to match energy of the first orbital with the analytical results. However, this is just a luck arising from lowering of numerical accuracy due to coarser grid. The proof are energies of all further orbitals, which deviate from analytical solutions much more than for the fine simulation, moreover, being reduced instead of increased despite additional energy introduced by regularization. As expected, the discrepancy is further gradually reducing as orbitals are localized further away from the center of the potential; amplitudes are less varying in space. The choice of the grid, therefore, depends on the goal of the simulation and must be performed carefully.

Degeneracy of orbitals

Finally, let us have a look at selected amplitudes of orbitals in the Hydrogen atom shown in Figure 4.5.4.3.

As the energy of the orbital without presence of magnetic field is given only by the principal quantum number n one should expect that all computed orbitals within one shell will be randomly superposed. For those, who do not look for such effects, fortunately, symmetry of numerical grid and regularization are partly breaking this degeneracy and the orbitals are distinguishable to some degree. All three orbitals 1s, 2s, 3s may have additionally overestimated energy due to regularization which makes them always separated from superposing with other orbitals; grid may have its own effect here as well. Orbitals p seems to have different energies from orbitals d due to symmetry of the grid influencing their energies as these orbitals have different value of the azimuthal quantum number l. The three orbitals 2p do not look exactly like in the books; they are tilted but seem to have proper relative orientation. The orbitals 3d seems to be notably superposed, however, they remain recognizable and similar to the orbitals shown in the literature. Because the numerical results tend to be rotated and superposed to some degree, the magnetic quantum number is not easy to be indicated and ommited in the Figure 4.5.4.3 and the Table 4.5.4.1.

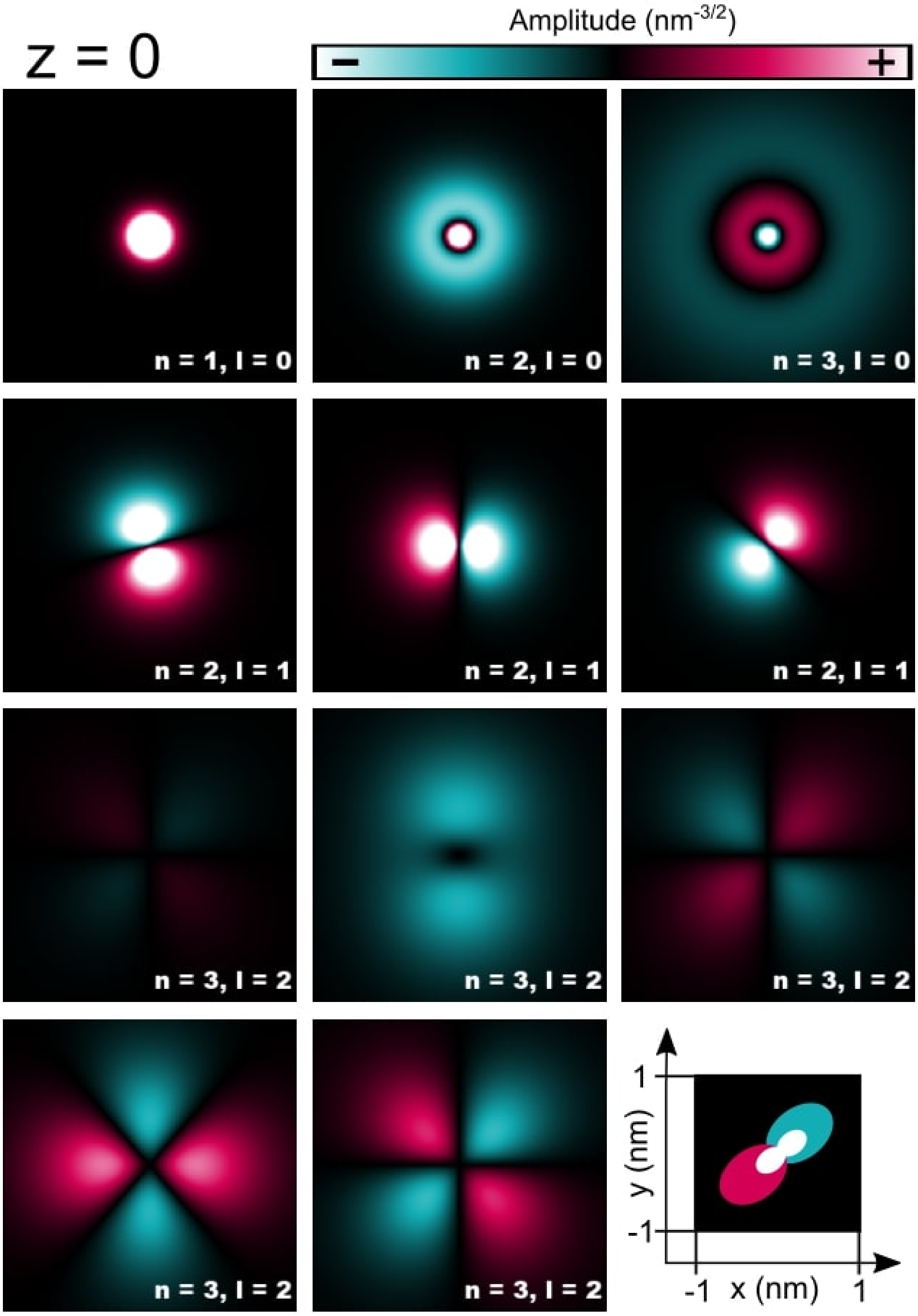


Figure 4.5.4.3: Cross-sections of 1s, 2s, 3s, 2p, and 3d orbitals computed for the Hydrogen atom.

Exercises

Compute lowest s, p, and d orbitals of a hydrogen atom and answer following questions:

- Are computed wave-functions of s orbitals in agreement with analytical solutions?
- Are all energies of orbitals the same as obtained analytically? If not, why do they deviate from analytical solutions?
- Is proper degeneracy present in the numerical solutions?

Additional question on numerics:

- What is the biggest regularizing parameter that can be used for the electrostatic potential and grid spacing if one aims at 1 meV accuracy for the energy of the ground state?

Last update: 27/10/2023

4.6 Quantum Wells

4.6.1 InAs / GaSb broken gap quantum well (BGQW) (type-II band alignment)

Author: Stefan Birner

Input files required:

- *1DInAs_GaSb_BGQW_k_zero_nnp.in*
- *1DInAs_GaSb_BGQW_k_parallel_nnp.in*
- *1DInAs_GaSb_BGQW_k_parallel_nnp_01.in*
- *1DInAs_GaSb_BGQW_k_parallel__nnp_11.in*

This tutorial aims to reproduce Figs. 1, 2(a), 2(b) and 3 of *Hybridization of electron, light-hole, and heavy-hole states in InAs/GaSb quantum wells*

Material parameters used are taken from *Optical transitions in broken gap heterostructures*.

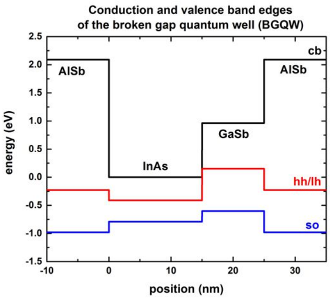
The heterostructure is a **broken gap quantum well (BGQW)** with 15 nm **InAs** and 10 nm **GaSb**, sandwiched between two 10 nm **AlSb** layers. Note that this heterostructure is asymmetric.

To be consistent with the above cited papers, strain is not included into the calculations although this would be possible. The structure has a **type-II** band alignment, i.e. the electrons are confined in the InAs layer, whereas the holes are confined in the GaSb layer. Depending on the width of the InAs and/or GaSb layers, things can be even more complicated because the hole states can hybridize with the electron states, making it difficult to distinguish between electron-like and hole-like states. Another difficulty arises because the lowest electron states might be located below the highest hole states. This requires a new algorithm to occupy the states according to a suitable Fermi level.

The following figure shows the electron and hole band edges of the BGQW structure.

- `band_structure/cb1D_001.dat` (Gamma conduction band edge) in units of [eV]
- `band_structure/vb1D_001.dat` (heavy hole valence band edge) in units of [eV]
- `band_structure/vb1D_002.dat` (light hole valence band edge) in units of [eV]
- `band_structure/vb1D_003.dat` (split-off hole valence band edge) in units of [eV]

The origin of the energy scale is set to the InAs conduction band edge energy. The heavy hole and light hole band edges are degenerate because we neglect the effects of strain to be consistent with the above cited papers.



Results

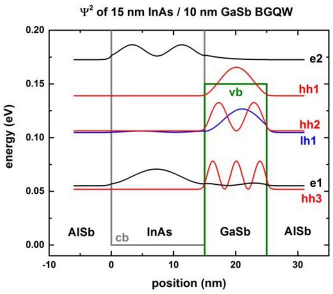
The input file used here is *1DInAs_GaSb_BGQW_k_zero_nnp.in*. The following figure shows the conduction band edge and the heavy/light hole valence band edges in this BGQW structure together with the electron (**e1**, **e2**), heavy hole (hh1, hh2, hh3) and light hole (lh1) energies and wave functions (ψ^2), calculated within 8-band $\mathbf{k} \cdot \mathbf{p}$ theory at the zone center, i.e. at $\mathbf{k}_{||}$:math:` = 0.`

One can clearly see that the electron state (**e1**, **e2**) are confined in the InAs layer (left part of the figure), whereas the heavy (hh1, hh2, hh3) and light hole (lh1) states are confined in the GaSb layer (right part of the figure). One can see a slight hybridization of the **e1** and lh1 states, i.e. these states are mixed states whereas the heavy hole states (hh1, h2, hh3) are not mixed and thus confined in the GaSb layer.

We use the data files

- `Schroedinger_kp/kp_8x8psi_squared_qc001_el_kpar0001_1D_dir.dat`, which contains ψ^2
- `Schroedinger_kp/kp_8x8psi_squared_qc001_el_kpar0001_1D_dir_shift.dat`, which contains $\psi^2 + \mathrm{E_i}$

The latter file contains the square of the wave functions (for `par0001`, i.e $k_{||} = 0$, i.e. $k_x = k_y = 0$), shifted by their energies, so that one can nicely plot the conduction and valence band edges together with the square of the wave functions.



The energies of the eigenstates are in units of [eV] and are contained in the file `Schroedinger_kp/kp_8x8eigenvalues_qc001_el_kpar0001_1D_dir.dat`

The input file *1DInAs_GaSb_BGQW_k_parallel.in* was used for the following results. The following figure shows the E($\mathbf{k}_{||}$) dispersion of the electron and hole states along the [10] direction and along the [11] direction in (k_x, k_y) space. The [01] direction has the same dispersion due to symmetry arguments.

In this input file, the energy levels and wave functions for 24 $\mathbf{k}_{||}$ points along a line from (k_x, k_y) = (0,0) to (k_x, k_y) = (0, k_y) have been calculated.

`Schroedinger_kp/kpar1D_disp_01_00el_8x8kp_ev_min001_ev_max020.dat` contains the $k_{||}$ dispersion from [00] to [01] because in the input file, it is specified that

```
dispersion{
    path{
        name = "kpar_01_00_10"
        point{ k = [0.0,  0.0,  1.0] }
```

(continues on next page)

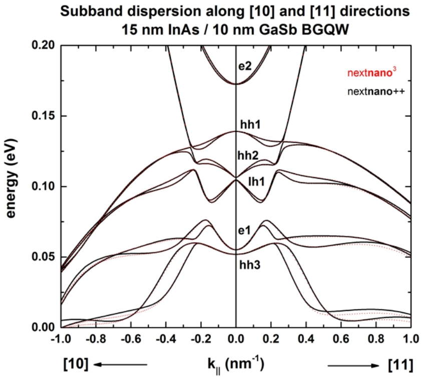
(continued from previous page)

```
        point{ k = [0.0,  0.0,  0.0] }
        point{ k = [0.0,  1.0,  0.0] }
        spacing = 1 / $number_k_parallel
    }
    path{
        name = "kpar_10_00_11"
        point{ k = [0.0, 1.0, 0.0] }
        point{ k = [0.0, 0.0, 0.0] }
        point{ k = [0.0, 1.0, 1.0] }
        spacing = 1 / $number_k_parallel
    }

    output_dispersions{}
    output_masses{}
}
```

The first column contains the $k_{||}$ value, the other columns contain the eigenvalues for each $k_{||}$ value: $\mathrm{E_n}(k_{||}) = \mathrm{E_n}(k_x, k_y) = \mathrm{E_n}(0, k_y)$. Here, n = 1,…,20. (…ev_min 001**_ev_max **020…) Note that for this particular example, the eigenvalues have to be sorted manually if you want to connect the energy values, i.e. to include lines ("lines are a guide to the eye").

The **black lines** are the results of *nextnano*++, the red dots are the results of *nextnano*^{3}.



At an in-plane wave vector of 0.014 1/Å, strong intermixing between the e1 and the lh1 states occurs. In contrast to the wave functions at $k_{||} = 0$, where the **e1** and **lh1** wave functions are nearly purely electron- or hole like, the wave functions at $k_{||}$ = (0, 0.014) = (0.014, 0) are a mixture of electron and light hole wave functions. Compare with Fig. 4 of the A. Zakharova et al.

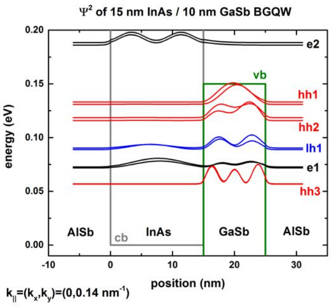
In asymmetric quantum wells, the double spin degeneracy is lifted at finite values of $\mathbf{k}_{||}$ because of spin-orbit interaction. This is the reason why we have two different dispersions $\mathrm{E}(\mathbf{k}_{||})$ for "spin up" and "spin down" states. This also means that the wave functions at finite $\mathbf{k}_{||}$ are different for "spin up" and "spin down" states.

The file `Schroedinger_kp/kp_8x8k_parallel_qc001_el1D_dir.dat` tells us which number of $\mathbf{k}_{||}$ vector corresponds to (k_x, k_y).

```
 k_par_number   k_x [1/nm]      k_y [1/nm]
  1            0.000000E+000   0.000000E+000     ==>   k|| = (kx,ky) = (0,0) [1/nm]
  ...
  29           0.000000E+000   1.400000E+000     ==>   k|| = (kx,ky) = (0,0.14) [1/nm]
1326           1.00000E+000    1.000000E+000     ==>   k|| = (kx,ky) = (1.0,1.0) [1/
↪nm]
```

In the following figure, we plot the square of the wave functions for $\mathbf{k}_{||}$ = (0,0.14) nm^{-1}. The corresponding label of our $\mathbf{k}_{||}$ numbering is **29**. Note that this labeling depends on the $\mathbf{k}_{||}$ space resolution, i.e. the number of $\mathbf{k}_{||}$ points that have been specified in the input file: `num-kp-parallel = 10000`

The wave functions (ψ^2 + $\mathrm{E_i}$)are contained in the file `Schroedinger_kp/kp_8x8psi_squared_qc001_hl_kpar00029_1D_dir_shift.dat`



The electron states (**e1**) couple strongly with the light hole states (lh1). This is expected from the energy dispersion plot because at 0.14 nm^{-1} a strong anticrossing is present for these states. One can also clearly see that for spin up and spin down states, different energy levels and different probability densities exist. This is in contrast to the states at $\mathbf{k}_{||} = 0$ which are two-fold spin degenerate as shown in the figure further above. Our results are similar to Fig. 4 of Zakharova's paper.

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.6.2 Exciton Binding Energy in an Infinite Quantum Well

Input File:

1D_exciton_binding_energy_infinite_QW_nnp.in

Content:

In this tutorial we study the *exciton* binding energy between the electron ground state and heavy hole ground state ($\mathrm{e}_1 - \mathrm{hh}_1$) in a single quantum well (ZnSe/CdTe/ZnSe). This energy correction is crucial, for example, when correlating computed optical transition energies in quantum wells with experimental results.

We aim to reproduce figures 6.4 (p. 196) and 6.5 (p. 197) of Paul Harrison's excellent book "*Quantum Wells, Wires and Dots*" *[HarrisonQWWD2005]*, thus the following description is based on the explanations made therein. We are grateful that the book comes along with a CD so that we were able to look up the relevant material parameters and to check the results for consistency.

Output Files:

\bias00000\Quantum\exciton_spectrum_QuantumRegion_Gamma_HH.dat

Description of analytical formulas

We present briefly the analytical formulas for the exciton binding energy in 1) bulk material and 2) quantum well structure (type-I). A full derivation can be found in *[HarrisonQWWD2005]*.

1) Bulk

The 3D bulk exciton binding energy can be calculated analytically

$$E_{\text{ex,b}} = -\frac{\mu e^4}{32\pi^2\hbar^2 e_\mathrm{r}^2 e_0^2} = -\frac{\mu}{m_0 e_\mathrm{r}^2} \cdot 13.61\ \mathrm{eV},$$

where

- μ is the reduced mass of the electron–hole pair

- $\hbar$ is Planck's constant divided by 2π
- e is the electron charge
- e_r is the dielectric constant ($e_{\mathrm{r,GaAs}}$ = 12.93, $e_{\mathrm{r,CdTe}}$ = 10.6)
- e_0 is the vacuum permittivity
- m_0 is the rest mass of the electron and
- 13.61 eV is the Rydberg energy.

The reduced mass of the electron–hole pair μ is calculated by

$$\frac{1}{\mu} = \frac{1}{m_{\mathrm{e}}} + \frac{1}{m_{\mathrm{h}}},$$

with the effective masses of electrons and holes: $\mathrm{m_e}$ and $\mathrm{m_h}$.

Example

The *reduced mass* of GaAs and CdTe are

$$\frac{1}{\mu_{\mathrm{GaAs}}} = \frac{1}{0.067} + \frac{1}{0.5} \qquad \Rightarrow \qquad \mu_{\mathrm{GaAs}} = 0.0591$$

$$\frac{1}{\mu_{\mathrm{CdTe}}} = \frac{1}{0.096} + \frac{1}{0.6} \qquad \Rightarrow \qquad \mu_{\mathrm{CdTe}} = 0.0828$$

with respective *Bohr radius*

$$\lambda_{\mathrm{GaAs}} = 11.6 \text{ nm}$$

$$\lambda_{\mathrm{CdTe}} = 6.8 \text{ nm}.$$

From the *3D bulk exciton binding energies*

$$E_{\text{ex,b}}(\mathrm{GaAs}) = -4.8 \text{ meV}$$

$$E_{\text{ex,b}}(\mathrm{CdTe}) = -10.0 \text{ meV}$$

the energy of the band gap transition including excitonic effects reads:

$$E_{\text{ex, GaAs}} = E_{\text{gap}} + E_{\text{ex,b}} = 1.519 \text{ eV} - 0.005 \text{ eV} = 1.514 \text{ eV}$$

$$E_{\text{ex, CdTe}} = E_{\text{gap}} + E_{\text{ex,b}} = 1.606 \text{ eV} - 0.010 \text{ eV} = 1.596 \text{ eV}$$

2) Quantum well (type-I)

Analytical results for the exciton ground state transition ($\mathrm{e_1}$–$\mathrm{hh_1}$) of a 1D quantum well (type-I) are only obtainable in the following two limits:

- infinitely thin quantum well (*2D limit*)

$$E_{\text{ex,QW}} = 4 \cdot E_{\text{ex}}$$
$$\lambda_{\text{ex,QW}} = \frac{\lambda_{\text{ex}}}{2}$$

- infinitely thick quantum well (*3D bulk exciton limit*)

$$E_{\text{ex,QW}} = E_{\text{ex}}$$

$$\lambda_{\text{ex,QW}} = \lambda_{\text{ex}}$$

Between these limits, the exciton correction, which depends on the well width, has to be calculated numerically, not only for the ground state but also for excited states (e.g. $\mathrm{e_2}$–$\mathrm{hh_2}$, $\mathrm{e_1}$–$\mathrm{lh_1}$).

Numerical calculation

Our numerical approach is based on a variational principle. We use the separable wave function

$$\psi(r) = \sqrt{\frac{2}{\pi}\frac{1}{\lambda}} \exp\left(-\frac{r}{\lambda}\right), \tag{4.6.2.1}$$

see e.g. p. 562, Eq. (13.4.27), Section 13.4.3 *Variational Method for Exciton Problem* in *[ChuangOpto1995]* or *[BastardPRB1982]*. The excitonic binding energy is then minimized with respect to the variational parameter λ (= Bohr radius).

Simulation

We study the exciton binding energy of a CdTe quantum well (with infinite barriers) as a function of well width. We chose infinite barriers, in order to be able to compare the *nextnano*++ calculations with standard textbook results, originally published by *[BastardPRB1982]*.

Input file

The material parameters used for CdTe are the following:

```
database{
    binary_zb{
        name = "CdTe"
        conduction_bands{
            Gamma{ mass = 0.096 }
            ...
        }
        valence_bands{
            HH{ mass = 0.6 }
            ...
        }
        dielectric_consts{
            static_a = 10.6
        }
        ...
    }
```

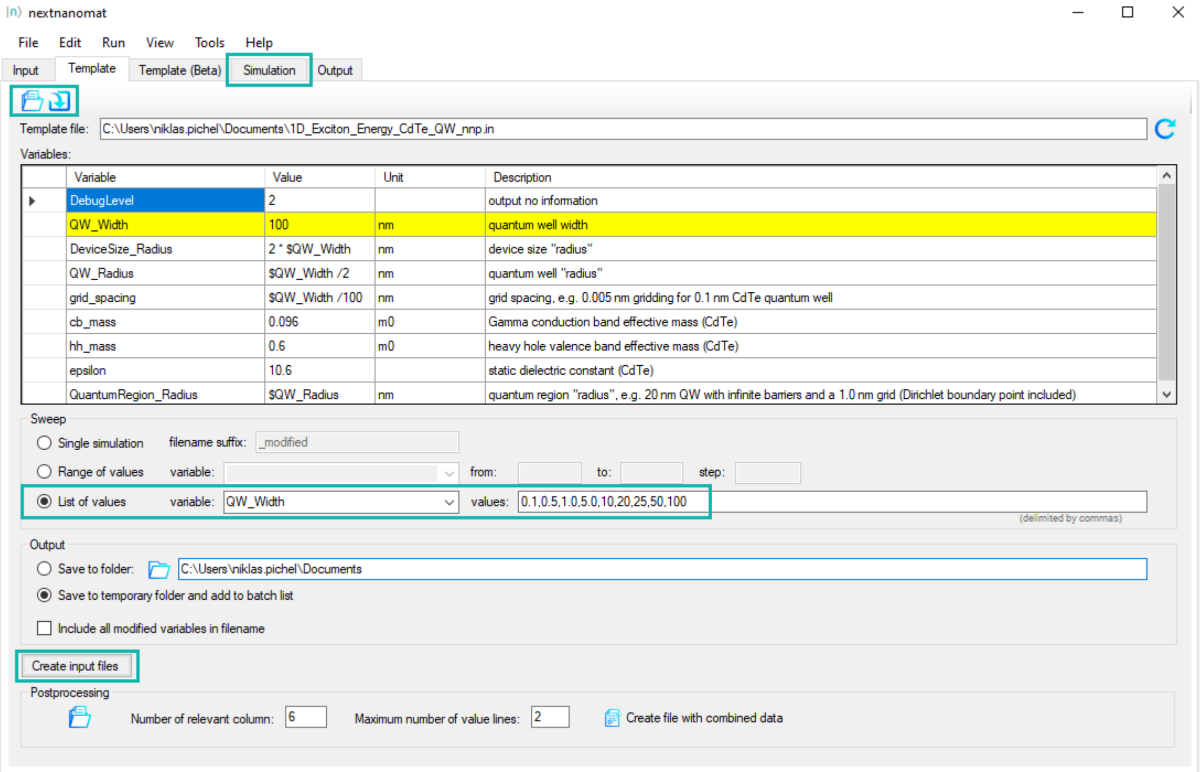
In order to calculate the exciton correction energy, the following group inside `quantum{ }` has to be used:

```
quantum{
    ...
    region{
        ...
        excitons{
            dielectric_const = 10.6
            electron_mass = 0.096
            hole_mass = 0.6
            energy_cutoff = 1000
            accuracy = 1e-10
        }
        ...
    }
}
```

Parameter Sweep

The following screenshot (Figure 4.6.2.1) shows how to use the *Template* feature of *nextnanomat* in order to calculate the exciton binding energy as a function of the quantum well width.

Initialization and execution of parameter sweep:



Template file: C:\Users\niklas.pichel\Documents\1D_Exciton_Energy_CdTe_QW_nnp.in

Variables:

Variable	Value	Unit	Description
DebugLevel	2		output no information
QW_Width	100	nm	quantum well width
DeviceSize_Radius	2 * $QW_Width	nm	device size "radius"
QW_Radius	$QW_Width /2	nm	quantum well "radius"
grid_spacing	$QW_Width /100	nm	grid spacing, e.g. 0.005 nm gridding for 0.1 nm CdTe quantum well
cb_mass	0.096	m0	Gamma conduction band effective mass (CdTe)
hh_mass	0.6	m0	heavy hole valence band effective mass (CdTe)
epsilon	10.6		static dielectric constant (CdTe)
QuantumRegion_Radius	$QW_Radius	nm	quantum region "radius", e.g. 20 nm QW with infinite barriers and a 1.0 nm grid (Dirichlet boundary point included)

Sweep

○ Single simulation filename suffix: _modified

○ Range of values variable: from: to: step:

● List of values variable: QW_Width values: 0.1,0.5,1.0,5.0,10,20,25,50,100 (delimited by commas)

Output

○ Save to folder: C:\Users\niklas.pichel\Documents

● Save to temporary folder and add to batch list

☐ Include all modified variables in filename

Postprocessing

Number of relevant column: 6 Maximum number of value lines: 2

Figure 4.6.2.1: Initializing parameter sweep for QW_Width in tab Template

- Open input file in Template tab.
- Select *List of values*, select variable `QW_Width`. The corresponding list of values are loaded from the template input file.
- Click on *Create input files* to create an input file for each quantum well width.
- Switch to *Simulation* tab and start the batch list of jobs.

Results

Figure 4.6.2.2 shows the exciton binding energy in an infinitely deep quantum well as a function of well width. The quantities are given in units of the 3D bulk exciton energy E_{ex} (also called *effective Rydberg energy*) and in units of the 3D bulk exciton Bohr radius λ_{ex} respectively. We see that the 3D limit is not reproduced correctly in our approach. To obtain the 3D limit, a nonseparable wave function $\psi(r, z_e, z_h)$ has to be used instead of (4.6.2.1).

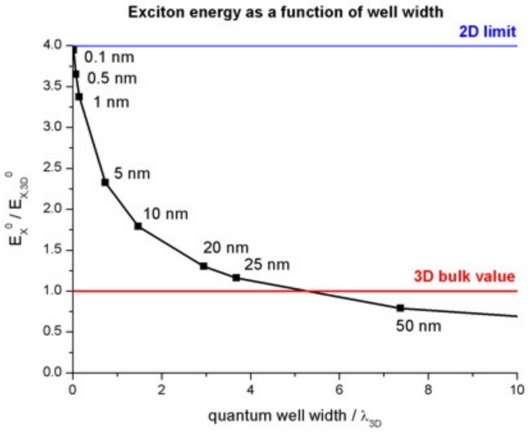


Figure 4.6.2.2: Exciton energy as a function of quantum well width

Figure 4.6.2.3 shows the exciton binding energy in an infinitely deep CdTe quantum well as a function of well width. The *nextnano++* results are in agreement with fig. 6.4 of *[HarrisonQWWD2005]*, although we use a simpler trial wave function with only one variational parameter.

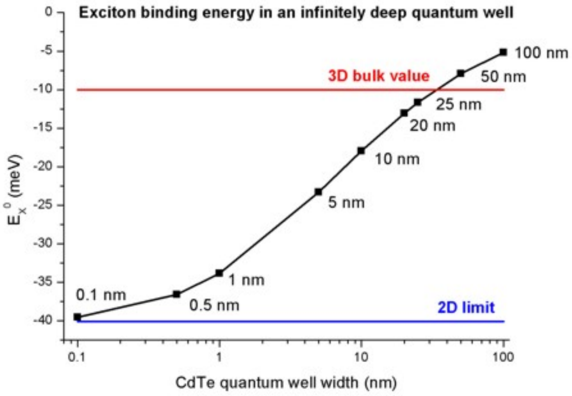


Figure 4.6.2.3: Exciton binding energy in an infinitely deep quantum well

Figure 4.6.2.4 shows the exciton Bohr radius a function of well width

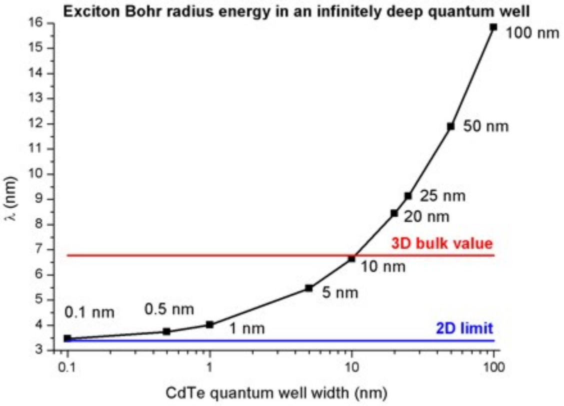


Figure 4.6.2.4: Exciton Bohr radius energy in an infinitely deep quantum well

This tutorial also exists for *nextnano³*.

Last update: 2025/06/26

4.6.3 Scattering times for electrons in unbiased and biased single and multiple quantum wells

Input files:

- *1DGaAs_AlGaAs_10nmQW_Lifetime.in*
- *1DGaAs_AlGaAs_12nmQW_LifetimeFig5_field.in*
- *1DGaAs_AlGaAs_SingleQW_7nm.in*
- *1DGaAs_AlGaAs_DoubleQW_7nm_nonsymmetric.in*
- *1DGaAs_AlGaAs_DoubleQW_LifetimeFig12_field.in*

Note

If you want to obtain the input files that are used within this tutorial, please check if you can find them in the installation directory. If you cannot find them, please submit a Support Ticket.

Scope:

This tutorial tries to reproduce the results of *[FerreiraBastard1989]*.

Scattering time as a function of quantum well width

Input file: *1DGaAs_AlGaAs_10nmQW_Lifetime.in*

First, we want to study the electron lifetimes (scattering rates) of a single quantum well as a function of quantum well width `$QW_width`. (Note: Use *nextnanomat*'s Template feature to automatically sweep over the quantum well width.)

Our quantum well consists of $GaAs$ that is sandwiched between two $Al_{0.3}Ga_{0.7}As$ barriers. The material parameters that we are using for this tutorial are identical to the ones used in *[FerreiraBastard1989]*, namely:

- electron mass: m_e = 0.07 m_0
- conduction band offset: CBO = 0.2138 eV
- static dielectric constant: ϵ = 12.5
- LO phonon energy: $\hbar\omega_0$ = 0.036 eV (longitudinal optical phonon)

For the calculations, a grid resolution of 0.1 nm has been used.

```
quantum{
    region{
        ...
        momentum_matrix_elements{ # calculate dipole moment elements <i|p|j> for␣
↪intraband transitions
            Gamma{}
        }
        dipole_moment_matrix_elements{ # calculate dipole moment elements <i|x|j> for␣
↪intraband transitions
            Gamma{}
        }
        transition_energies{ # calculate transition energies
            Gamma{}
        }
        lifetimes{ # calculate lifetimes
            Gamma{}
            phonon_energy = 0.036  # [eV]
        }
    }
}
```

The following two figures (Figure 4.6.3.1, Figure 4.6.3.2) show the conduction band edges and the lowest confined eigenstates (including the square of the wave functions) for a 6 nm and an 18 nm $AlGaAs/GaAs$ quantum well.

The quantum well width can be varied easily by making use of the variable

```
$QW_width = 10  # (DisplayUnit:nm) (ListOfValues:5.2,5.4,5.6,5.8,6,7,8,10,12,14,15,16,
↪17,18,19,20)
```

which can be swept automatically using the *nextnanomat*'s Template feature. Open the input file and select "List of values" and variable "QW_width".

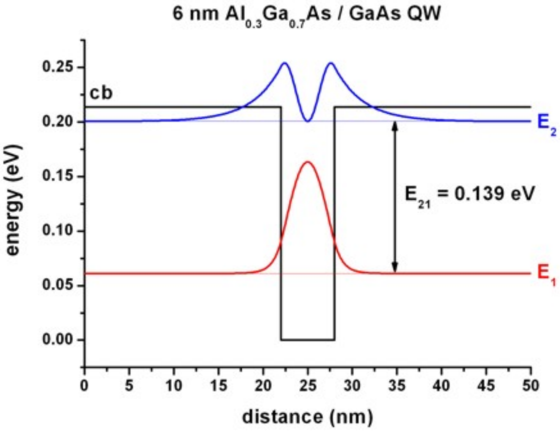


Figure 4.6.3.1: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1 and E_2)

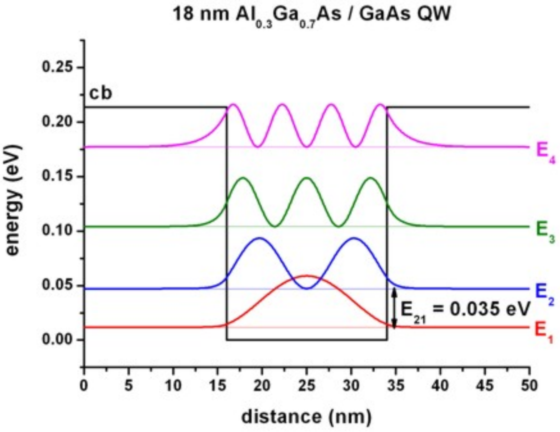


Figure 4.6.3.2: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1, E_2, E_3 and E_4).

Figure 4.6.3.3 shows the electron lifetime of the second eigenstate (E_2 = initial state) to the ground state (E_1 = final state), i.e. the intersubband transition with energy E_{21} for different quantum well widths. The temperature is set to 0 K.

For quantum well widths smaller than 5.4 nm (*[FerreiraBastard1989]*: 5.5 nm), only the ground state is confined and E_2 is unbound. For quantum well widths larger than 18 nm (*[FerreiraBastard1989]*: 17.8 nm), the transition energy E_{21} is smaller than the LO phonon energy of 36 meV, thus scattering through the emission of an LO phonon is not possible anymore. The calculations are in good agreement with Fig. 3 of *[FerreiraBastard1989]*.

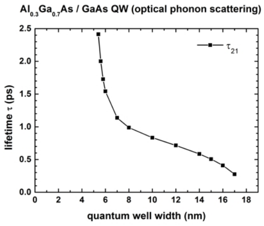


Figure 4.6.3.3: Calculated lifetimes τ as a function of quantum well width

The output of the electron lifetime can be found in this file: *bias_00000\Quantum\lifetimes_quantum_region_Gamma.dat*.

```
...
                  Intersubband dipole moment  | < psi_f* | pz | psi_i > |  [h_bar /␣
↪nm]
------------------|-------------------------------------------------------------------
↪---
                                 Oscillator strength []
------------------|--------------|----------------------------------------------------
↪---
                                               Energy of transition [eV]
------------------|--------------|-------------|--------------------------------------
↪---
                                                             m* [m_0]      lifetime␣
↪[ps]
------------------|--------------|-------------|-------------|-------------|----------
↪---
...
<psi001*|pz|psi002> 0.19717291    0.985747159   0.085864536   0.070000000   0.
↪833765805
...
```

Here, the shown values for the intersubband transitions correspond to a 10 nm QW.

Scattering times as a function of electric field magnitude

Input file: *1DGaAs_AlGaAs_12nmQW_LifetimeFig5_field.in*

This input file will perform a sweep over the electric field strength. Figure 4.6.3.4 shows the lowest eigenstates of a 12 nm $AlGaAs$/ $GaAs$ QW at an applied electric field of -50 kV/cm. This time the conduction band edge is not flat anymore. It is tilted because of the electric field.

The sweep over the electric field magnitude can be done automatically. For these calculations, a grid resolution of 0.10 nm had been used. The calculations presented in Figure 4.6.3.5 are in reasonable agreement with Fig. 5 in *[FerreiraBastard1989]*.

Single quantum wells

Input file: *1DGaAs_AlGaAs_SingleQW_7nm.in*

The two confined energy levels and wave functions of the 7 nm single quantum well are shown in Figure 4.6.3.6. The energy of the ground state is 50.4 meV.

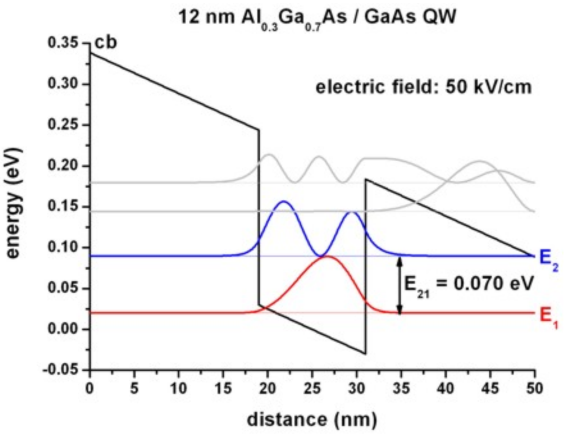


Figure 4.6.3.4: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1 and E_2), when electric field of -50 kV/cm is applied.

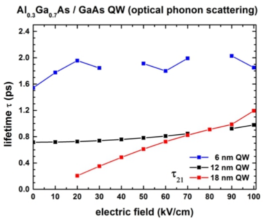


Figure 4.6.3.5: Calculated lifetimes τ in a single quantum well as a function of applied electric field.

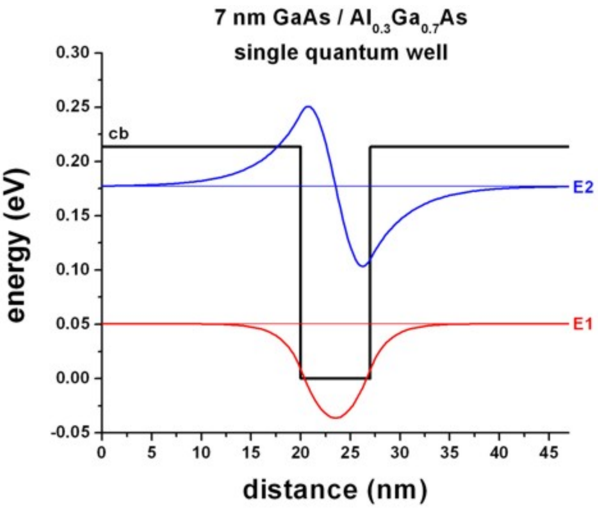


Figure 4.6.3.6: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1 and E_2)

Double quantum wells

Input file: *1DGaAs_AlGaAs_DoubleQW_7nm_nonsymmetric.in*

Here, we study the electron energy levels of a non-symmetric double quantum well structure as a function of quantum well width of the right quantum well: `$right_QW_width`. The right quantum well width can be varied easily by making use of the variable:

```
$right_QW_width = 7     # (DisplayUnit:nm) (ListOfValues:7.0,8.0,10.0,12.5,15.0,17.5,
↪20.0,22.5,25.0,27.5,30.0,35.0,37.5,40.0,45.0,47.5,50.0,55.0,57.5,60.0,65.0,67.5,70.
↪0,75.0,77.5,80.0,85.0,87.5,90.0,95.0,97.5,100.0)
```

which can be swept automatically using the *nextnanomat*'s Template feature. Open input file and select "List of values" and variable "right_QW_width". For the following figures, a grid resolution of 0.25 nm had been used.

Figure 4.6.3.7 shows the energy levels of a non-symmetric double quantum well structure ($GaAs$ / $Al_{0.3}Ga_{0.7}As$) where the left quantum well always has the width 7 nm, and the right quantum well varies from 7 nm to 100 nm. The two $GaAs$ wells are separated by a 5 nm $Al_{0.3}Ga_{0.7}As$ barrier. The figure shows the energy levels as a function of the width of the larger quantum well.

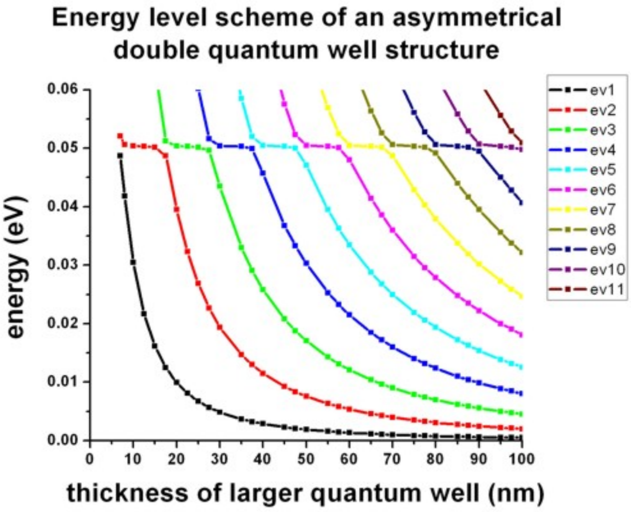


Figure 4.6.3.7: Calculated energy levels for different energy states as a function of $L_{\text{QW,right}}$.

One can see, that for certain widths of the larger quantum well, an anti-crossing due to bonding and anti-bonding states occurs. This happens whenever an eigenstate of the larger well matches the energy of the ground state of the smaller (7 nm) quantum well (which is at 50.4 meV, see example shown above: *1DGaAs_AlGaAs_SingleQW_7nm.in*). Our calculations are in very good agreement with Fig. 9 in *[FerreiraBastard1989]*.

The anti-crossing behavior and the plateaus at 50.4 meV of the energy level scheme (see Figure 4.6.3.7) can be illustrated by plotting the wave functions for different values of $L_{\text{QW,right}}$, see Figure 4.6.3.8, Figure 4.6.3.9, Figure 4.6.3.10 and Figure 4.6.3.11.

Figure 4.6.3.8 shows a symmetric double quantum well where both wells have the width 7 nm including the wave functions of the lowest confined states. If the barrier between these two wells had been very large, both wells would have had a ground state at 50.4 meV. However, due to the small barrier, coupling between these two wells becomes possible. The two lowest states form a bonding and an anti-bonding state. The bonding state now has a reduced energy of 48.7 meV and the anti-bonding state has an increased energy of 52.1 meV.

Figure 4.6.3.9 shows a non-symmetric double QW where the right QW has a width of 12.5 nm. In this case, the ground state can be found in the larger well, the second state in the 7 nm QW, whereas the third eigenstate is again localized in the larger well. Here, no bonding or anti-bonding states exist.

Figure 4.6.3.10 shows a non-symmetric double QW where the right QW has a width of 17.5 nm. In this case, the ground state can be again found in the larger well (similar to Figure 4.6.3.9), but this time, the third state of moves

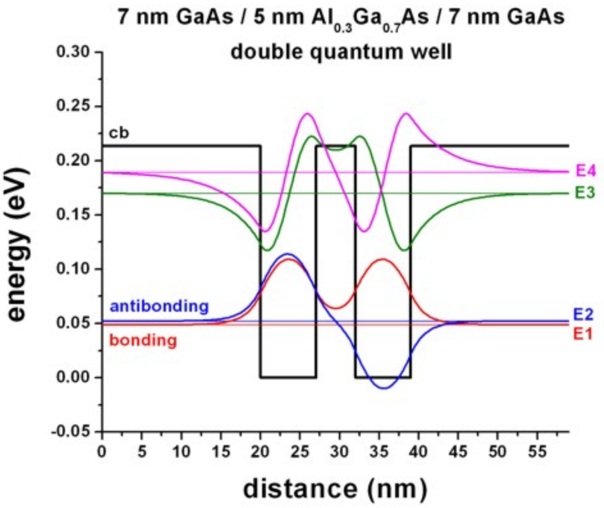


Figure 4.6.3.8: Calculated conduction band edge profile (black) for symmetric double quantum well: L_{left} = 7 nm and L_{right} = 7 nm, with wave functions of confined electron states.

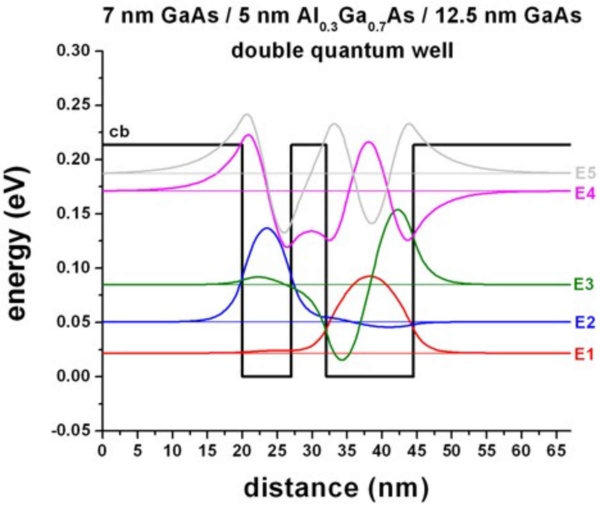


Figure 4.6.3.9: Calculated conduction band edge profile (black) for non-symmetric double quantum well: L_{left} = 7 nm and L_{right} = 12.5 nm, with wave functions of confined electron states.

down in energy and couples to the 7 nm ground state of the left well (compare with Figure 4.6.3.9). This coupling leads to the formation of a bonding and an anti-bonding states.

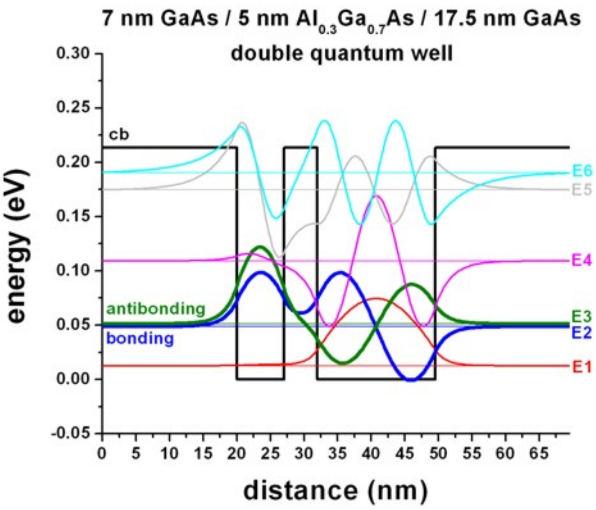


Figure 4.6.3.10: Calculated conduction band edge profile (black) for non-symmetric double quantum well: L_{left} = 7 nm and L_{right} = 17.5 nm, with wave functions of confined electron states.

Figure 4.6.3.11 shows a non-symmetric double QW where the right QW has a width of 25 nm. In this case, the ground state and the second state can be found in the larger well, whereas the third eigenstate is localized in the smaller (7 nm) well. The forth eigenstate is localized in the larger well. Again, no bonding or anti-bonding states exist

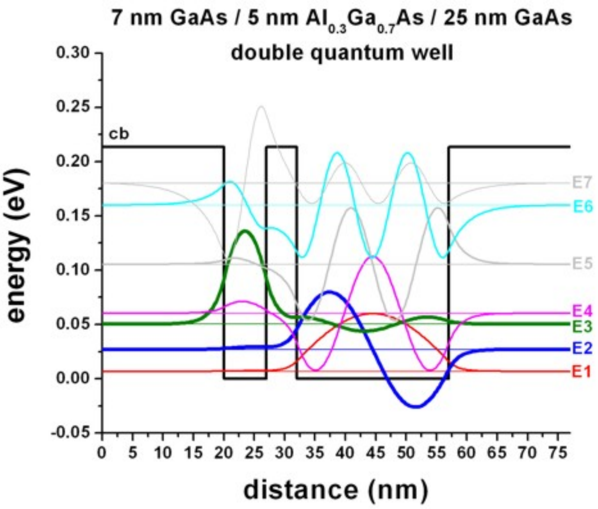


Figure 4.6.3.11: Calculated conduction band edge profile (black) for symmetric double quantum well: L_{left} = 7 nm and L_{right} = 25 nm, with wave functions of confined electron states.

Biased double quantum well

Input file: *1DGaAs_AlGaAs_DoubleQW_LifetimeFig12_field.in*

Figure 4.6.3.12 shows the lifetime of the 2 → 1 transition ("ground state of left quantum well to ground state of right quantum well transition") as a function of electric field. The variable d is the thickness of the left well and the barrier region. The right well is assumed to have the same thickness as the left quantum well, i.e. d/2.

The parameter d can be varied easily by making use of the variable

```
$QWBarrierThickness = 6     # (DisplayUnit:nm) (ListOfValues:6,9)
```

which can be swept automatically using the *nextnanomat*'s Template feature. Open input file and select "List of values" and variable "QWBarrierThickness".

There seems to be qualitative agreement to Fig. 12 in *[FerreiraBastard1989]*. For d = 9 nm, the LO phonon emission is forbidden for electric fields smaller than ~ | 40 kV/cm | because the transition energy is smaller than the LO phonon energy of 36 meV.

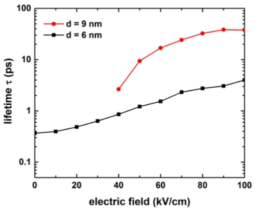


Figure 4.6.3.12: Calculated lifetimes τ in a single quantum well as a function of applied electric field.

This tutorial also exists for *nextnano³*.

Last update: 27/05/2025

4.6.4 — DEV — Strain effects in freestanding nitride nanostructures

Attention

This tutorial is under construction

Input files:

- *Strained-QW_AlGaN-GaN_Povolotskyi_PSS_2005_3D_nnp.in*

Scope:

This tutorial aims to simulate strain effects in a lattice mismatched, freestanding heterostructure with wurtzite crystal structure consisting of an AlGaN/GaN quantum well. This tutorial is based on *[Povolotskyi2005]*.

Output files:

- *Strain/strain_simulation_2d_slice_middle_along_yz.vtr*
- *Strain/hydrostatic_strain_2d_slice_middle_along_yz.vtr*
- *Strain/strain_simulation_2d_slice_boundary_along_xz.vtr*
- *Strain/elastic_energy_density_2d_slice_middle_along_yz.vtr*

Structure

Figure 4.6.4.1 shows the AlGaN/GaN/AlGaN quantum well structure, which is simulated in this tutorial. A 4 nm wide GaN QW layer is embedded between two $\mathrm{Al}_{0.28}\mathrm{Ga}_{0.72}\mathrm{N}$ layers. The bottom AlGaN layer has a width of 10 nm, whereas the top AlGaN layer has a width of 6 nm. The overall shape of this nitride nanowire structure has the form of a cuboid with 50 nm x 50 nm extensions in the x- and y-directions. The height in the z-direction is 20 nm. The overall structure is surrounded by air (i.e. with a material where all elastic constants are zero).

The calculated strain pattern of this AlGaN/GaN structure is found to be highly non-homogeneous. The elastic energy has been minimized using continuum elasticity theory. We assume that the external stress applied to the structure is zero (freestanding structure).

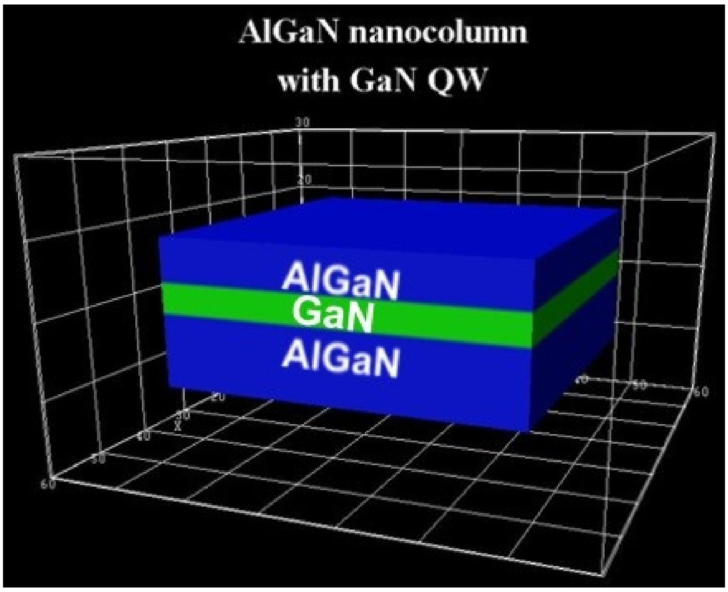


Figure 4.6.4.1: Simulated heterostructure consisting of a GaN (green) layer sandwiched in between two AlGaN (blue) layers.

Results

Strain tensor components

In this section we show several strain tensor components $\epsilon_{ij}(x, y, z)$ as a function of position (x, y, z) for slices through the structure (vertical cross-section of the structure). Note that GaN has a larger lattice constant than AlGaN. Consequently, we expect the GaN layer to be compressively strained and the AlGaN layers to be tensely strained (or rather unstrained).

Figure 4.6.4.2 shows the strain tensor component ϵ_{xx} along y, z at $x = 25.0\,\mathrm{nm}$. The corresponding data can be found in the file *Strain/strain_simulation_2d_slice_middle_along_yz.vtr*. The bottom AlGaN layer is rather unstrained (at the bottom), the GaN QW layer is strained compressively along the x-direction (blue region). This is not a surprise as we assumed coherent interfaces, i.e. the atomic planes match each other perfectly (pseudomorphic strain). The GaN QW induces a tensile strain to the AlGaN top layer (red region).

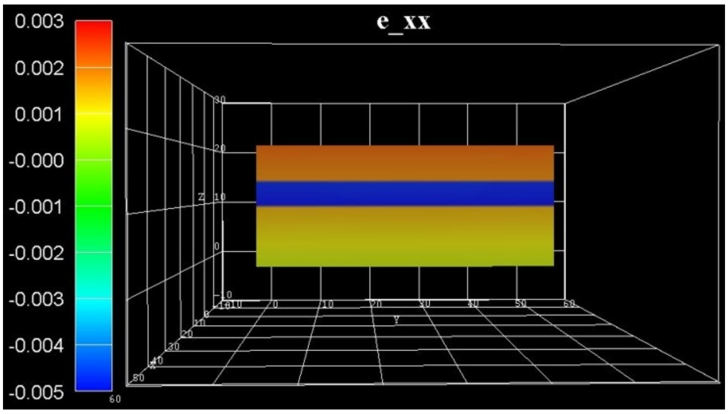


Figure 4.6.4.2: Calculated strain component ϵ_{xx} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

Figure 4.6.4.3 shows the strain tensor component ϵ_{yy}. Similar to Figure 4.6.4.2, the GaN layer is compressively strained (blue region), but only in the center and not at the boundaries, where it is nearly relaxed. Note that the ϵ_{xx} and ϵ_{yy} strain tensor components are not symmetric. This is due to the nitride crystal structure which has hexagonal symmetry perpendicular to the (x, y) plane (and not cubic symmetry as the geometry of the structure).

- In the center (blue region), the GaN takes on the lattice constant of AlGaN (compressive strain).

- At the QW boundaries, the GaN takes on the lattice constant of ~GaN (nearly fully relaxed).
- Below and above the QW boundaries (red regions), the AlGaN takes on the lattice constant of ~GaN (tensile strain).

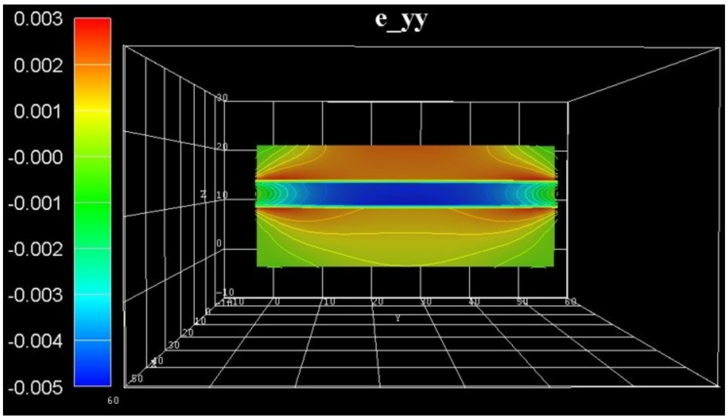


Figure 4.6.4.3: Calculated strain component ϵ_{yy} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

Figure 4.6.4.4 shows the strain tensor component ϵ_{zz}. As the GaN layer is compressively strained along both the x- and y-directions, it is tensilely strained (green region) along the z-direction (biaxially strained GaN layer, Poisson ratio).

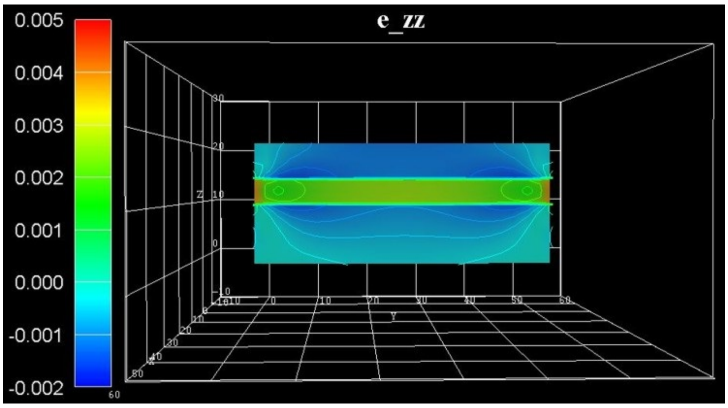


Figure 4.6.4.4: Calculated strain component ϵ_{zz} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

Figure 4.6.4.5 shows the hydrostatic strain $\epsilon_{\mathrm{hydro}} = \epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}$, which is the trace of the strain tensor, i.e. the sum of the diagonal strain tensor components. It corresponds to the overall volume change. The data can be found in the file *Strain/hydrostatic_strain_2d_slice_middle_along_yz.vtr*. The blue region indicates that the GaN is strained compressively. AlGaN is mostly unstrained apart from the red regions at the left and right boundaries of the material interfaces. In a real sample, due to the deformation, the heterostructure changes its shape and becomes bent. In our case, the strain is small (less than 1%), so the shape of the structure does not change significantly.

In contrast to heterostructures, which are infinitely large and homogeneous in the lateral directions (i.e. in the (x, y) plane), the deformation of a structure of a finite size is not homogeneous, as e.g. in GaN nanowire heterostructures. Since the structure is grown along the high symmetry direction [0001], the strain patterns exhibits reflection symmetry along the axis through the center (oriented parallel to the z-axis). The deformation becomes homogeneous in the region near the central axis, reproducing the case of an infinitely large structure.

Figure 4.6.4.6 shows the off-diagonal strain tensor component ϵ_{yz}. The strain tensor components ϵ_{xy} and ϵ_{xz} are zero for this particular slice. (In fact, the maximum value of ϵ_{xy} is an order of magnitude smaller that the maximum value of ϵ_{xz} or ϵ_{yz}.)

Figure 4.6.4.7 shows the same off-diagonal strain tensor component ϵ_{yz}, but this time at slices at the left and right boundaries. The corresponding data can be found in the file *Strain/strain_simulation_2d_slice_boundary_along_xz.vtr*

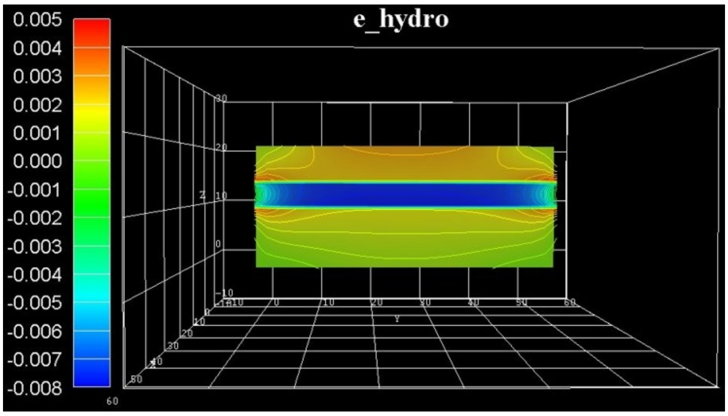


Figure 4.6.4.5: Calculated hydrostatic strain component $\epsilon_{\mathrm{hydro}} = \epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}$ in the y, z plane at $x = 25.0\,\mathrm{nm}$.

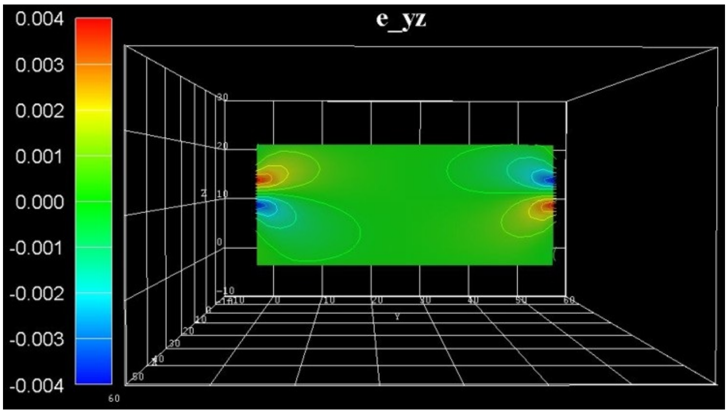


Figure 4.6.4.6: Calculated strain component ϵ_{yz} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

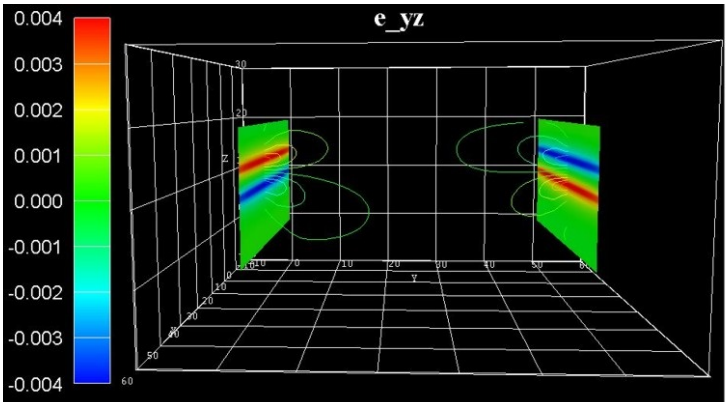


Figure 4.6.4.7: Calculated strain component ϵ_{yz} in the x, z plane at the boundaries.

Elastic energy density

Due to a possible usage of such structures as a light emitter, the strain in the GaN layer where charge carriers are confined, is or particular interest, i.e. the influence of stain on the conduction and valence band structure through deformation potentials. Additional, piezoelectric and pyroelectric fields have to be taken into account. The piezoelectric fields depend on the strain distribution in the sample. Thus, both the piezoelectric field and the GaN energy gap will vary along the lateral direction.

Figure 4.6.4.8 shows the energy density of the elastic deformation in units of $[\mathrm{eV}/\mathrm{nm}^3]$. The corresponding data can be found in the file *Strain/elastic_energy_density_2d_slice_middle_along_yz.vtr*. The accumulated elastic energy in the pseudomorphically grown GaN QW is gradually reduced towards the free surface along the lateral direction. Consequently, the GaN QW center is almost fully strained, whereas towards the lateral surface there is a continuous relaxation.

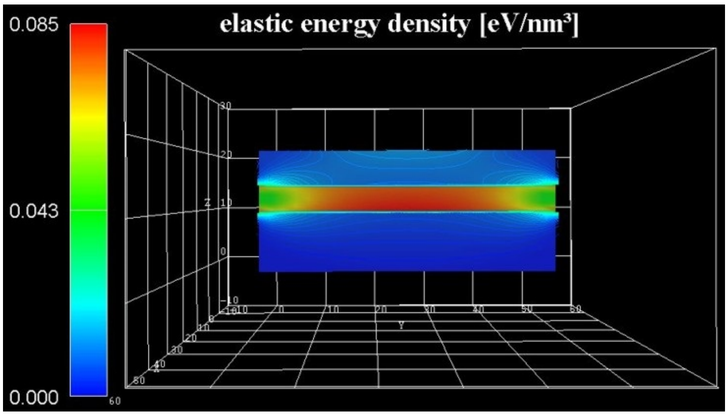


Figure 4.6.4.8: Elastic energy density in the x, z plane at $x = 25.0\,\mathrm{nm}$.

Last update: 17/07/2024

4.7 Quantum Wires

4.7.1 Hexagonal GaAs/AlGaAs nanowires

Input files:

- *2DGaAs_AlGaAs_circle_nnp.in*
- *2DGaAs_AlGaAs_hexagon_nnp.in*
- *2D_Hexagonal_Nanowire_2DEG_nnp.in*

Scope:

In this tutorial we simulate a circular and a hexagonal $GaAs/Al_{0.33}Ga_{0.67}As$ core-shell structure (Part A) and a hexagonal $GaAs/AlGaAs$ nanowire structure (Part B).

Output files:

- *bias_00000\Quantum\probabilities_quantum_region_Gamma_.vtr*

Part A: Schrödinger equation of a two-dimensional core-shell structure

In this part of the tutorial, we solve the two-dimensional Schrödinger equation of a circular and a hexagonal $GaAs/Al_{0.33}Ga_{0.67}As$ core-shell structure.

Circular core-shell structure

Input file: *2DGaAs_AlGaAs_circle_nnp.in*

Figure 4.7.1.1 shows the probability density of the 6th eigenstate of the circular $GaAs/AlGaAs$ structure. The data is contained in the file *bias_00000\Quantum\probabilities_quantum_region_Gamma_.vtr*. It's energy level is higher than the $AlGaAs$ barrier energy, i.e. this state is not confined in the circular shaped $GaAs$ quantum well. The horizontal and vertical slices are through the center and show the square of the probability amplitude of this eigenstate.

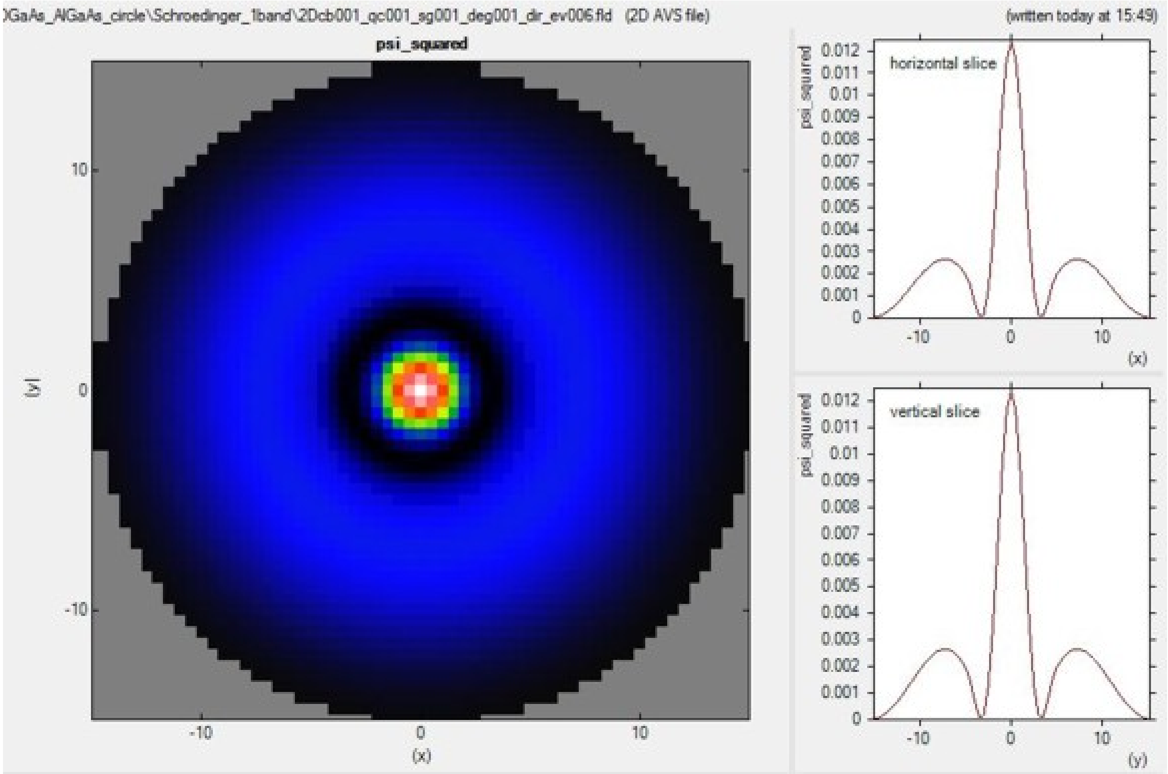


Figure 4.7.1.1: Ψ^2 of the 6th electron eigenstate.

The $GaAs$ core has a radius of 5 nm. (It cannot be recognized on this plot.) The $AlGaAs$ shell has a radius of 15 nm. It is surrounded by an infinite barrier which comes from the "band offset" due to the surrounding material "air".

Hexagonal core-shell structure

Input file: *2DGaAs_AlGaAs_hexagon_nnp.in*

Figure 4.7.1.2 shows the conduction band edge of the hexagonal $GaAs/AlGaAs$ structure. The $GaAs$ region is indicated in black, the $AlGaAs$ region in blue. Horizontal and vertical slices through the center show the energy of the conduction band edge profile. The data is contained in the file *bias_00000\bandedges.fld*

The diameter of the hexagonal shaped $GaAs$ core is ~8.66 nm (corresponding to an outer radius of the core of 5 nm), and the diameter of the hexagonal shaped $AlGaAs$ shell is ~26 nm (corresponding to an outer radius of the shell of 15 nm).

Figure 4.7.1.3 shows the probability density of the 10th eigenstate of the circular $GaAs/AlGaAs$ structure. The data is contained in the file *bias_00000\Quantum\probabilities_quantum_region_Gamma_.vtr*. It's energy level is higher than the $AlGaAs$ barrier energy, i.e. this state is not confined in the hexagonal shaped $GaAs$ quantum well. The horizontal and vertical slices are through the center and show the square of the probability amplitude of this eigenstate.

The hexagonal $GaAs$ core has an outer radius of 5 nm. It cannot be seen on this plot. The $AlGaAs$ shell has a diameter of 26 nm. It is surrounded by an infinite barrier which case comes from the "band offset" due to the

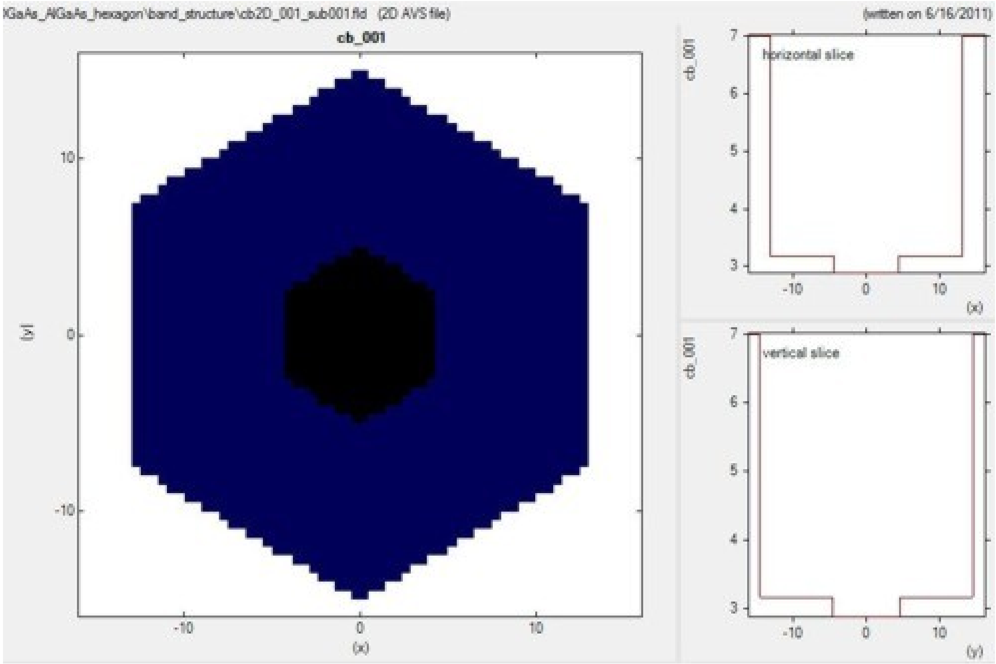


Figure 4.7.1.2: Conduction band edge profile of the hexagonal core-shell structure.

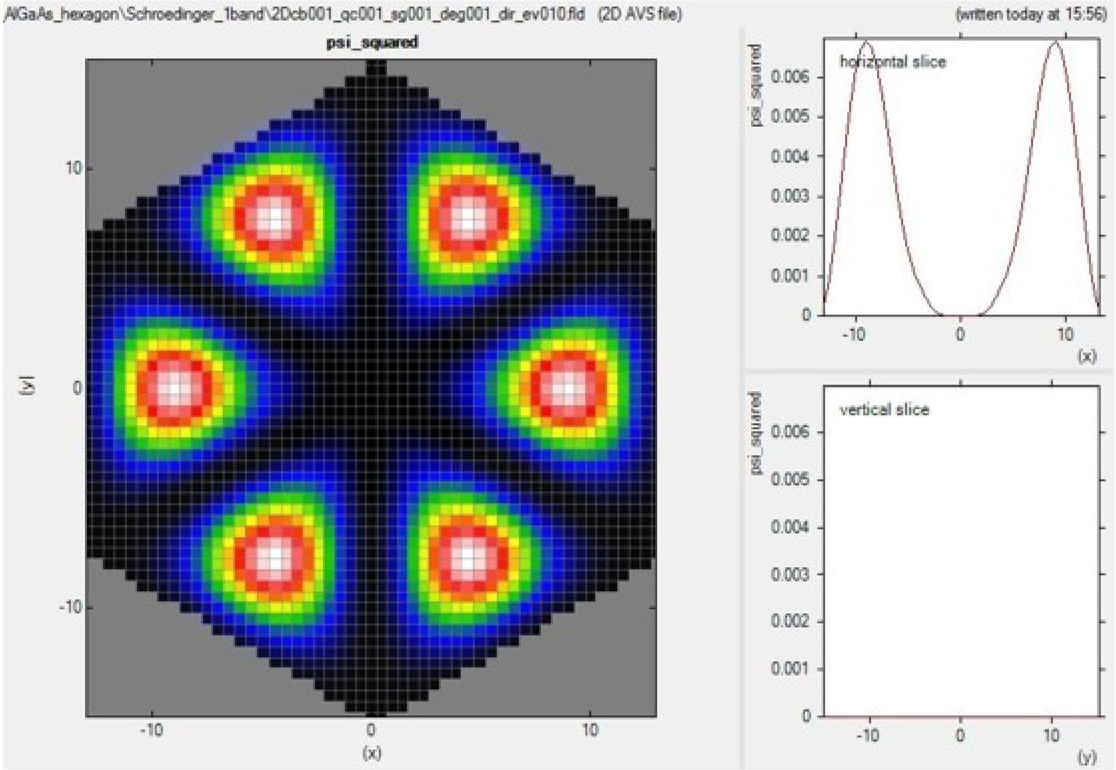


Figure 4.7.1.3: Ψ^2 of 10th electron eigenstate.

surrounding material “air”.

Alloy sweep

In the following, we vary the alloy content x of the ternary $Al_xGa_{1-x}As$ from 0 to 0.33 in 11 steps. For x = 0, we have pure $GaAs$. For x = 0.33 we have an $AlGaAs/GaAs$ conduction band offset of 0.285 eV, and a valence band offset of -0.168 eV. In the latter case, the quantum confinement is stronger. Even for x = 0 we have “quantum confinement” due to the Dirichlet boundary conditions (corresponding to infinite barriers) at the shell surface that we use for the Schrödinger equation. Consequently, even for x = 0, we get an e1 - h1 transition energy from the lowest electron state (e1) to the highest heavy hole state (h1) that is larger than the band gap as shown in Figure 4.7.1.4.

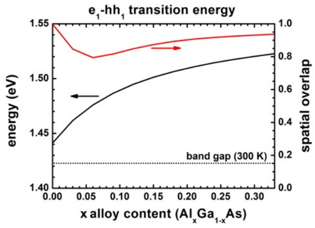


Figure 4.7.1.4: Transition energy and spatial overlap of e1 to h1 transition as a function of alloy content x.

The transition energies (e1 - h1), as well as the spatial overlap integral of the electron and hole ground state wave functions, are contained in this file: *bias_00000\Quantum\momentum_matrix_elements_quantum_region_Gamma_001.txt*

```
alloy      type                          e1-hl[eV]     e1[eV]        h1[eV]       ␣
↪matrix_element
0.330000   |<psi_vb001|psi_cb001_>|^2   1.522777615   2.965889676   1.443112062   0.
↪936344908
0.300000   |<psi_vb001|psi_cb001_>|^2   1.520316794   2.963669699   1.443352905   0.
↪931291593
...
```

The spatial overlap of electron and hole wave functions is always very high. When there is only confinement due to the shell boundary, the matrix element is very high (99.8 %). The matrix element must be smaller than 1 for x = 0 because the electron and hole masses are different. The matrix element must be even smaller (94 %) for x = 0.33 (strong confinement) because in addition to the mass difference, the conduction and valence band offsets are not equivalent. The matrix element has a minimum at around x = 0.06 because in this case the electron wave function penetrates into the barrier much stronger that the hole wave function does. Thus the differences in well and barrier masses (as well as band offsets) play an important role for the spatial extension of the wave functions.

Part B: Hexagonal 2DEG - Two-dimensional electron gas in a delta-doped hexagonal shaped $GaAs/AlGaAs$ nanowire heterostructure

Input file: *2D_Hexagonal_Nanowire_2DEG_nnp.in*

The following example deals with a delta-doped $GaAs/AlGaAs$ 2DEG (two-dimensional electron gas) structure. In this case, the heterostructure consists of a hexagonal $GaAs/AlGaAs$ nanowire, see Figure 4.7.1.5.

The self-consistently calculated conduction band edge (*bandedges.fld*) is shown in Figure 4.7.1.6. The horizontal and vertical slices through the center indicate the triangular potential well (conduction band minimum) where the 2DEG is located.

The resulting 2DEG electron density (*bias_00000\density_electron_fld*) is shown Figure 4.7.1.7. At the corners, the electron density is significantly higher, thus one-dimensional conducting channels are formed. Although the structure itself has a hexagonal symmetry, our rectangular grid breaks this symmetry. Therefore the density in the upper/lower corner are different from the density at the left/right corners.

The 2D Poisson equation and the 2D Schrödinger equation have been solved self-consistently. The dimension of the Schrödinger matrix is 28,625. The CPU time for this calculation was about 18 minutes.

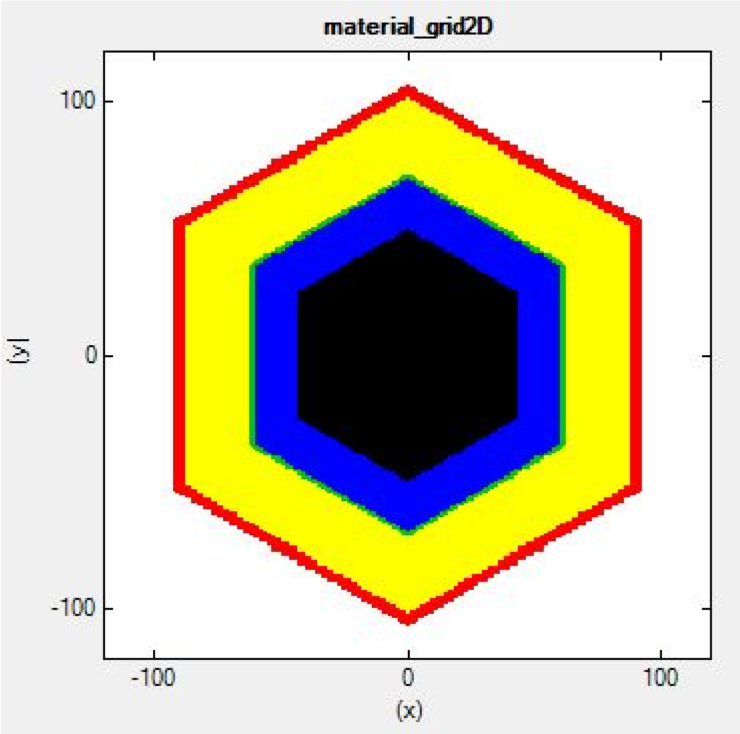


Figure 4.7.1.5: The material layers of the struture: $GaAs$ core (black), $AlGaAs$ spacer (blue), Si-doped $AlGaAs$ (green), $AlGaAs$ (yellow), $GaAs$ capping layer (red) and Schottky barrier contact (black) are shown. (The white layer itself is not included in the calculation. It only serves as a boundary condition)

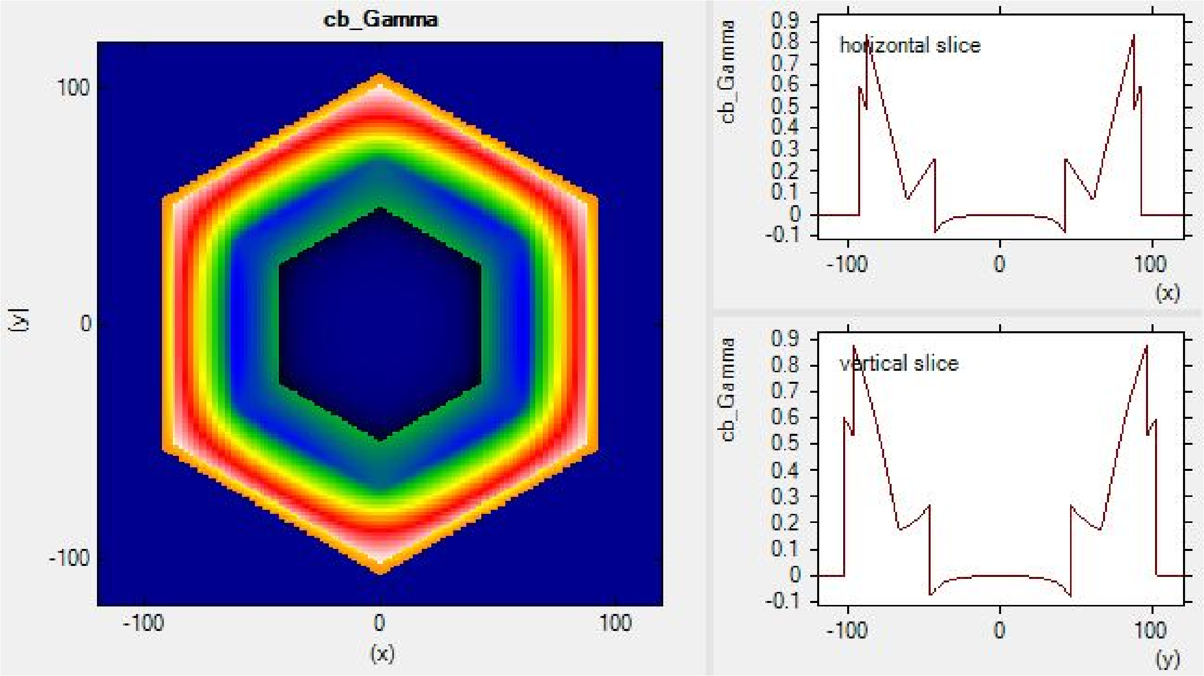


Figure 4.7.1.6: Conduction band edge profile.

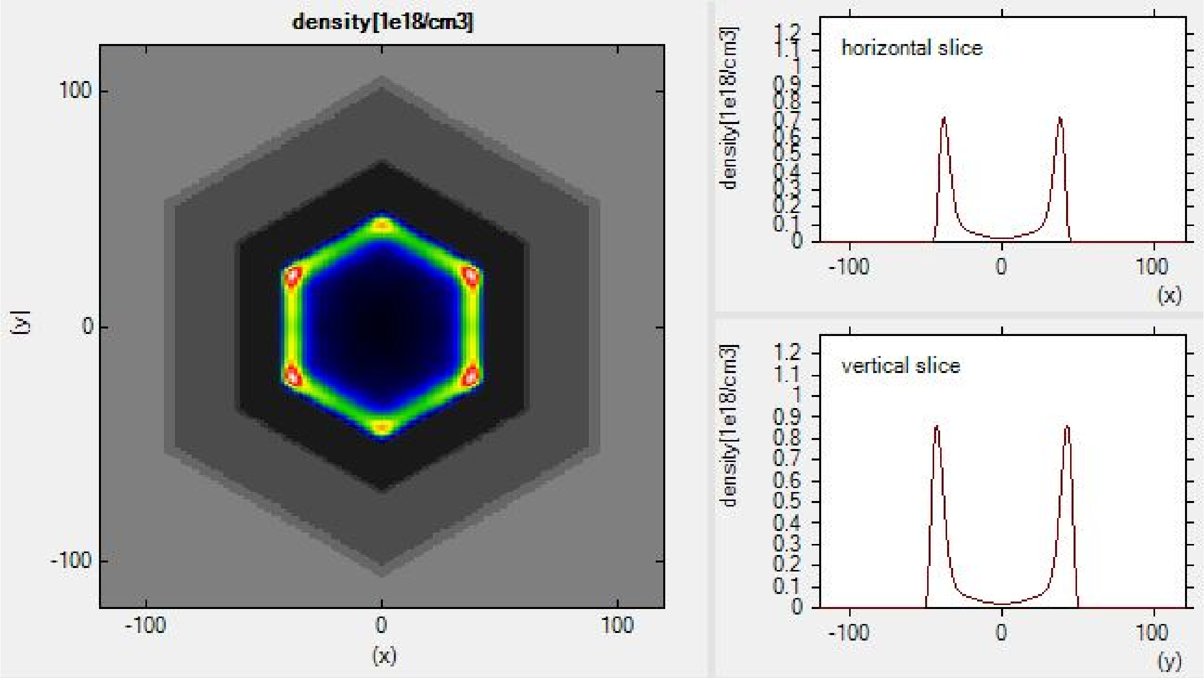


Figure 4.7.1.7: Charge density profile.

This tutorial also exists for *nextnano³*.

Last update: 27/05/2025

4.7.2 Electron wave functions in a cylindrical well (2D Quantum Corral)

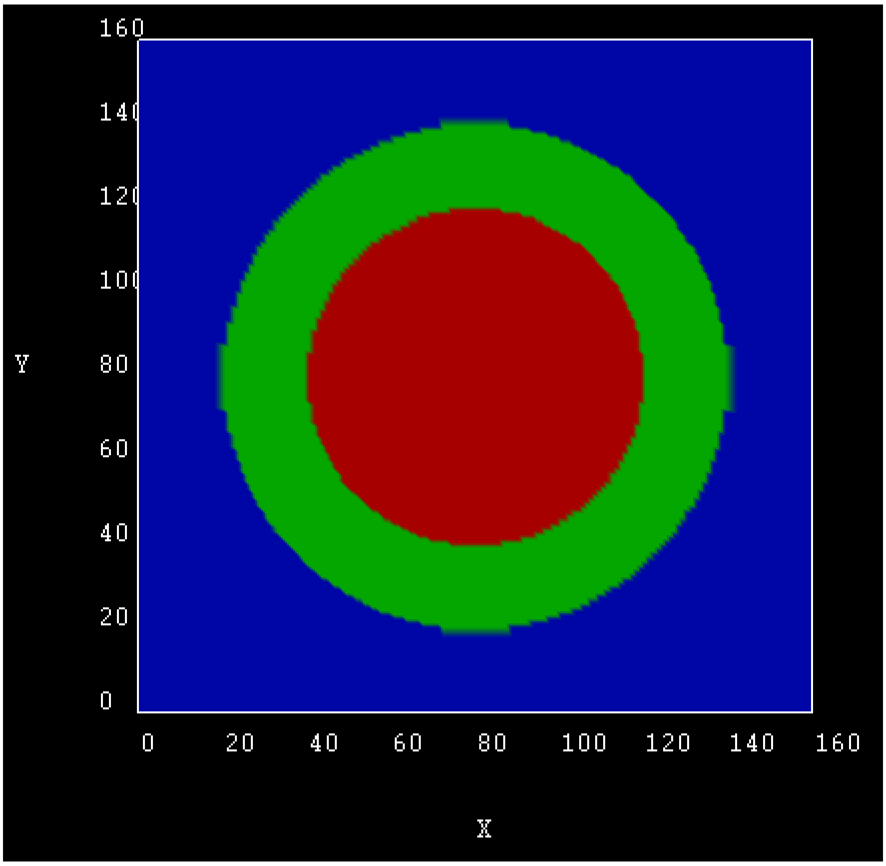
In this tutorial we demonstrate 2D simulation of a cilindrical quantum well. We will see the electron eigenstates and their degeneracy.

Input files used in this tutorial are the followings:

- *2DQuantumCorral_nn3.in / *_nnp.in*

Structure

- A cylindrical InAs quantum well (diameter 80 nm) is surrounded by a cylindrical GaAs barrier (20 nm) which is surrounded by air. The whole sample is 160 nm x 160 nm.
- We assume **infinite** GaAs barriers. This can be achieved by a circular quantum cluster with **Dirichlet** boundary conditions, i.e. the wave function is forced to be zero in the GaAs barrier.
- The electron mass of InAs is assumed to be isotropic and parabolic ($m_e = 0.026 m_0$).
- Strain is not taken into account.



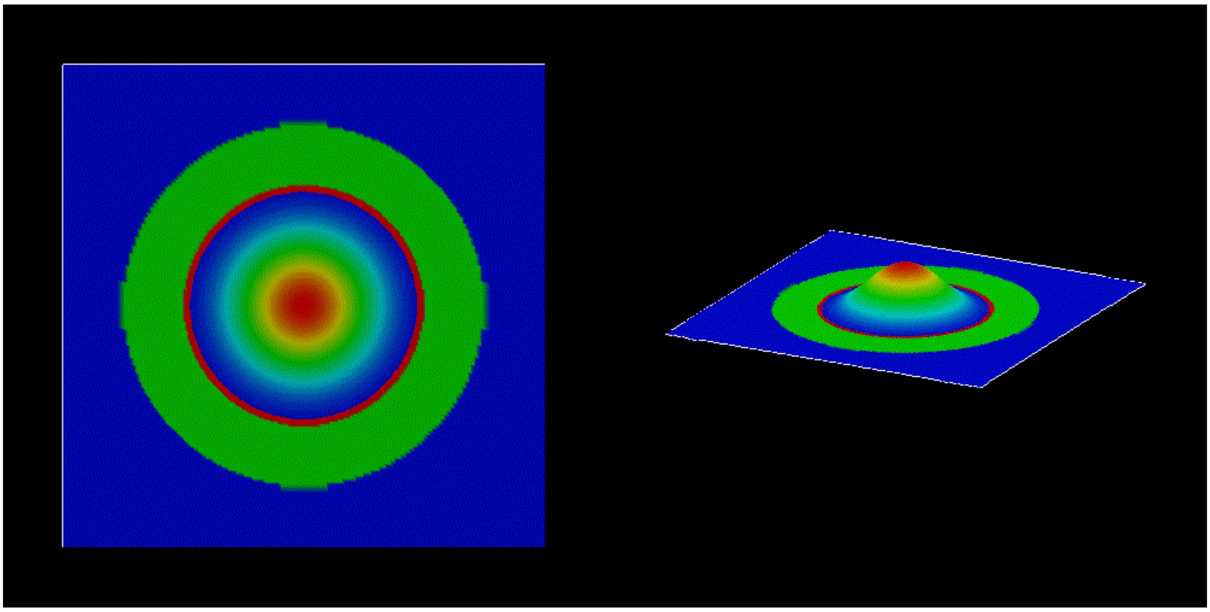
Simulation outcome

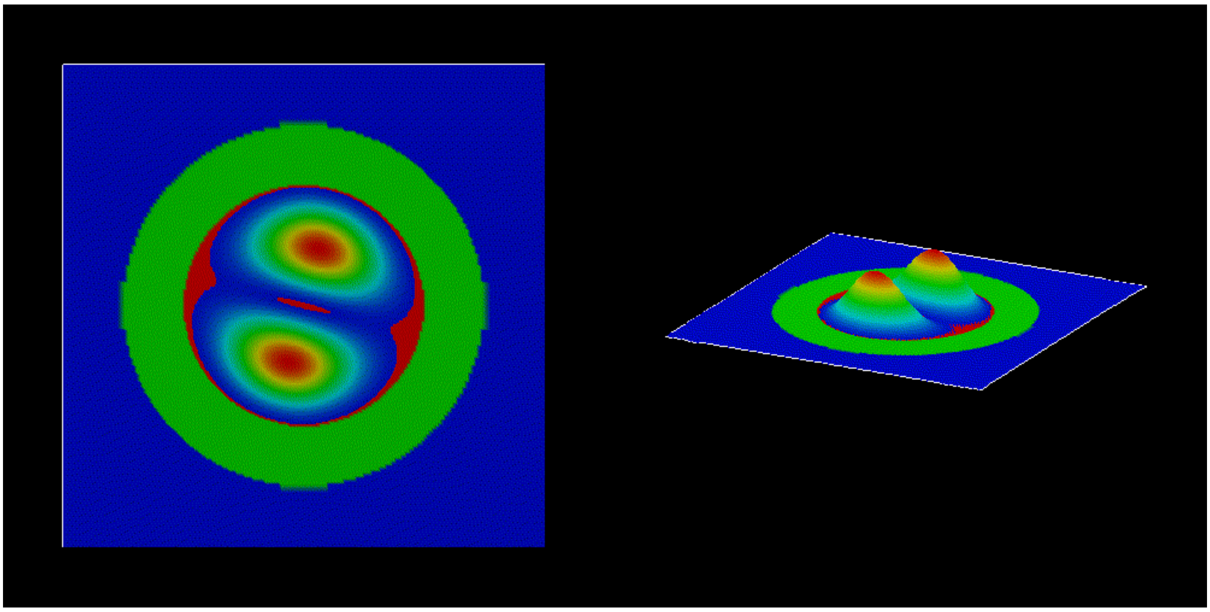
Electron wave functions

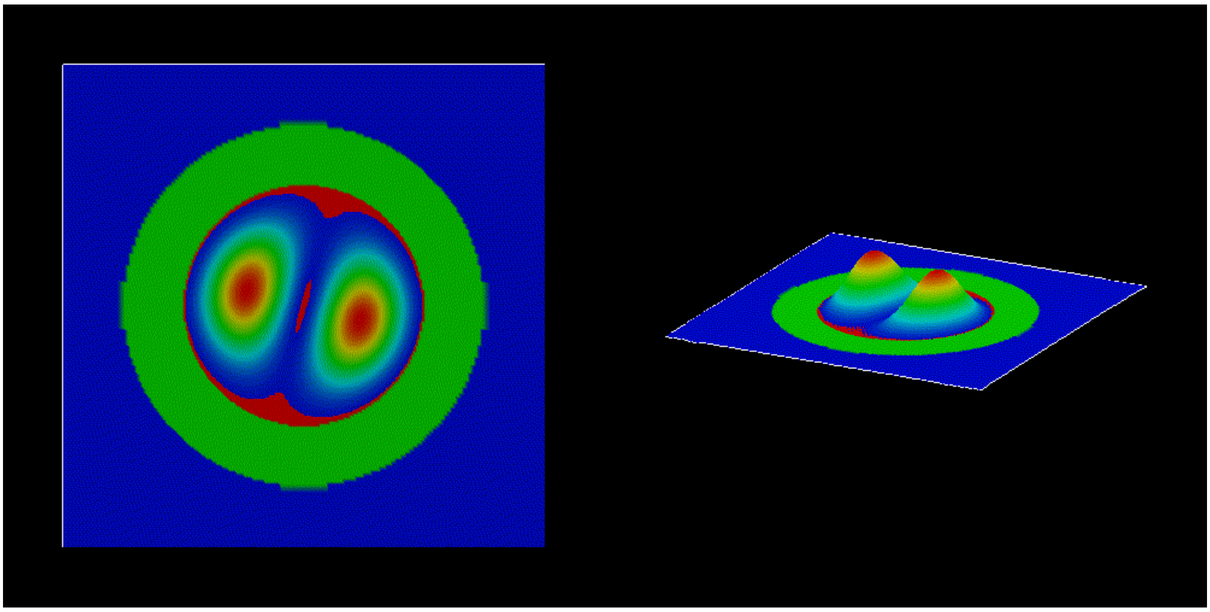
The size of the quantum cluster is a circle of diameter $2a = 80$ nm.

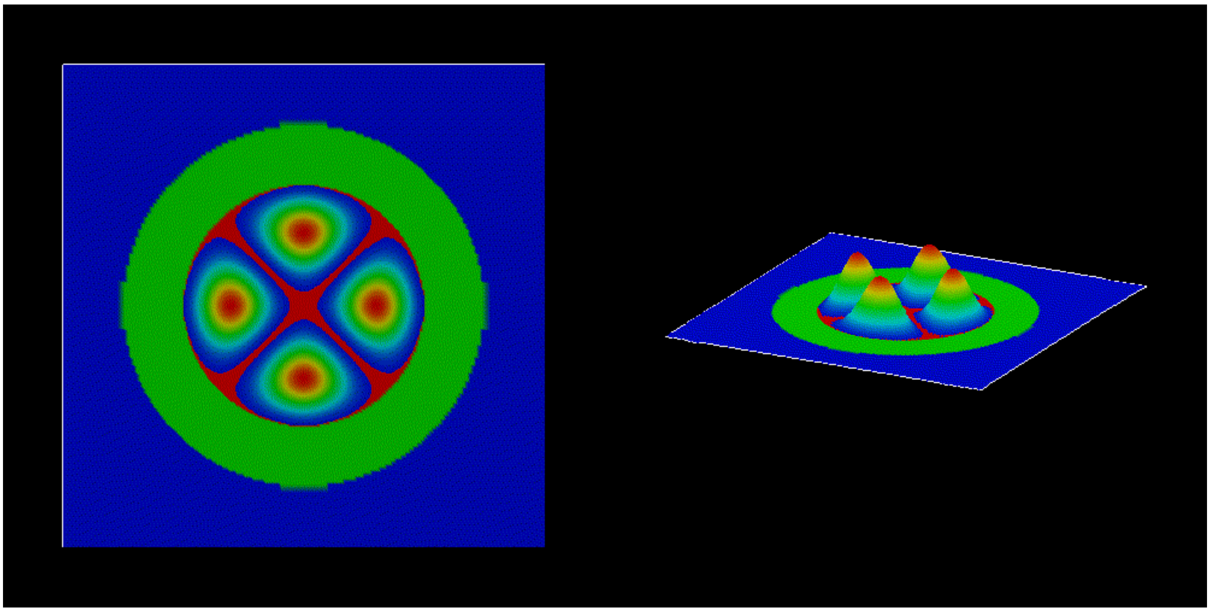
The following figures shows the square of the electron wave functions (i.e. ψ^2) of the corresponding eigenstates. They were calculated within the effective-mass approximation (single-band) on a rectangular finite-differences grid.

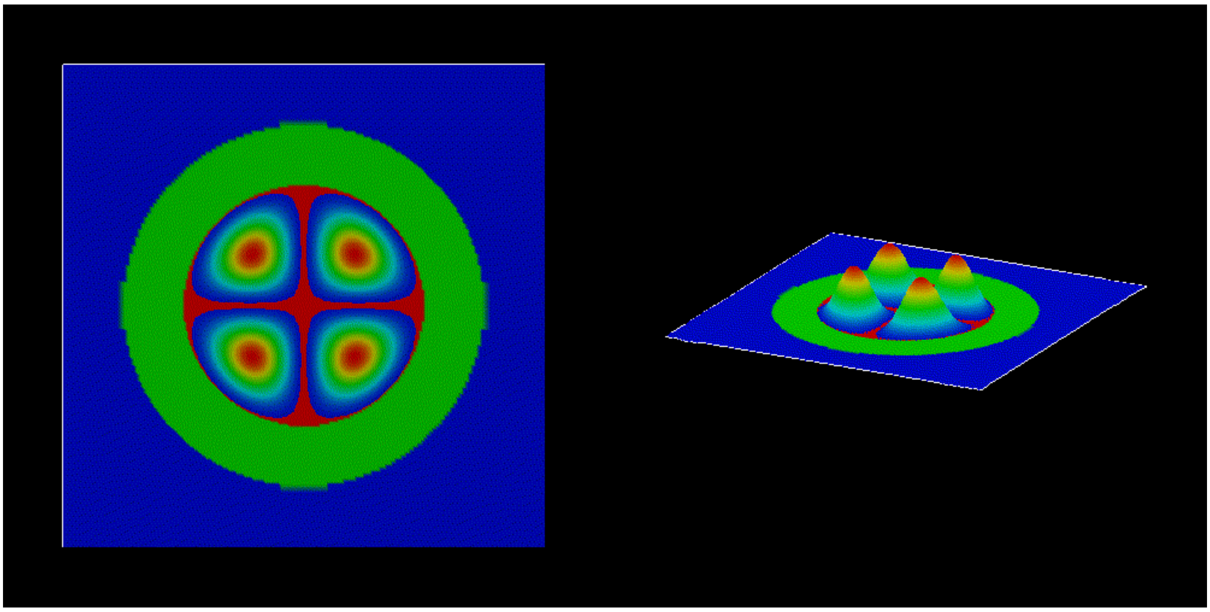
- 1st eigenstate, $(n,\ l) = (1,\ 0)$
- 2nd eigenstate, $(n,\ l) = (1,\ 1)$
- 3rd eigenstate, $(n,\ l) = (1, -1)$
- 4th eigenstate, $(n,\ l) = (1,\ 2)$
- 5th eigenstate, $(n,\ l) = (1, -2)$
- 6th eigenstate, $(n,\ l) = (2,\ 0)$
- 15th eigenstate, $(n,\ l) = (3,\ 0)$

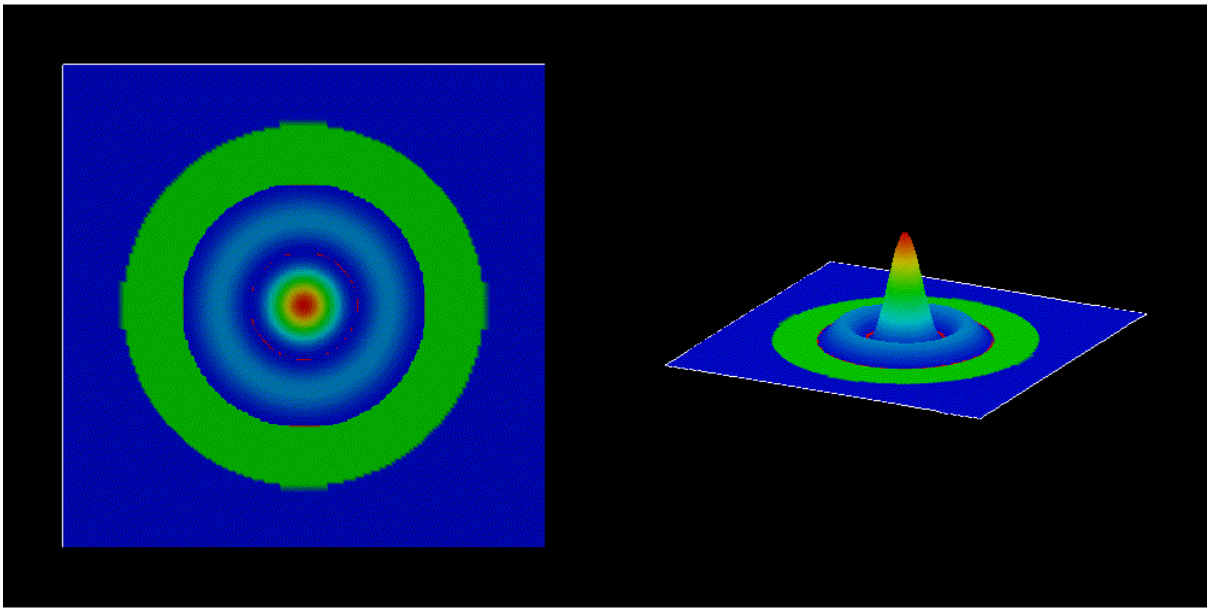


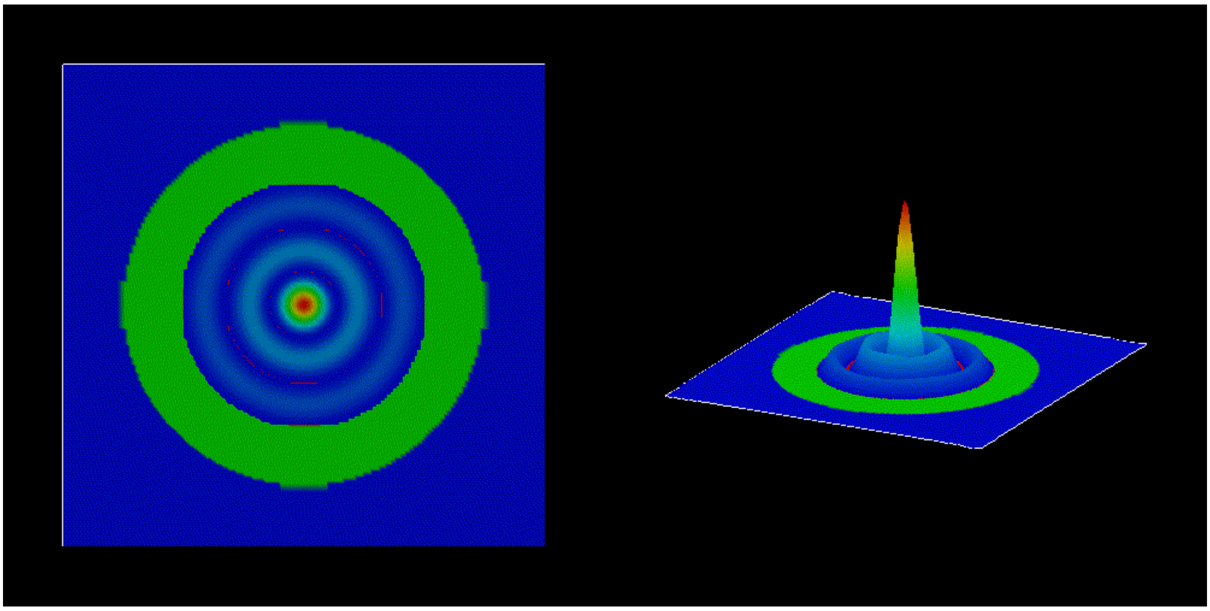




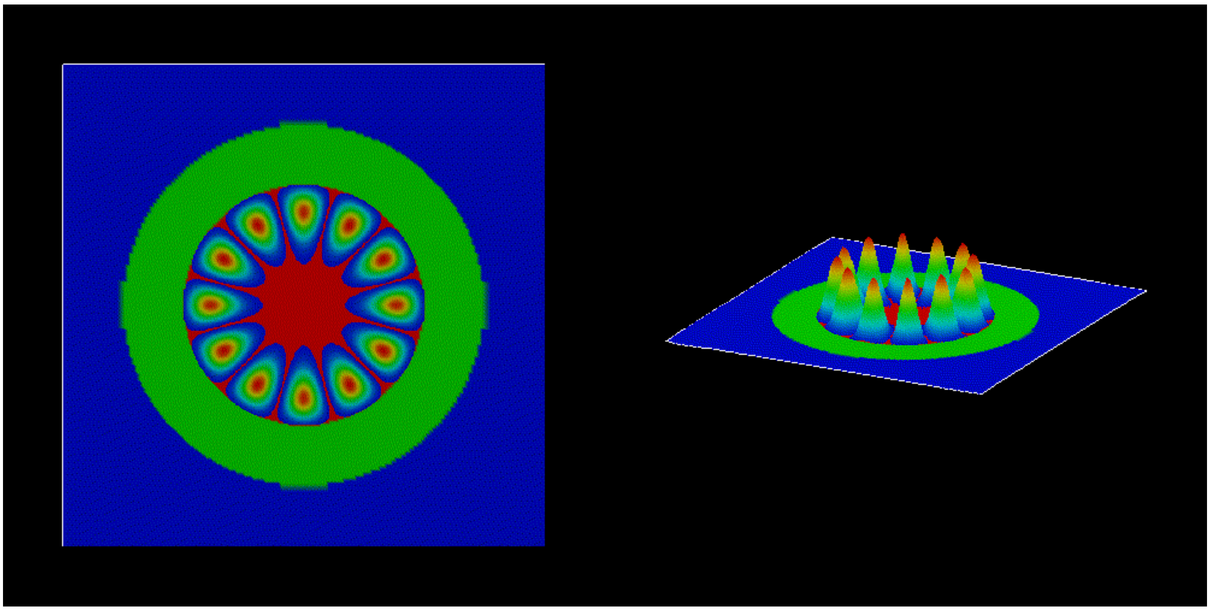








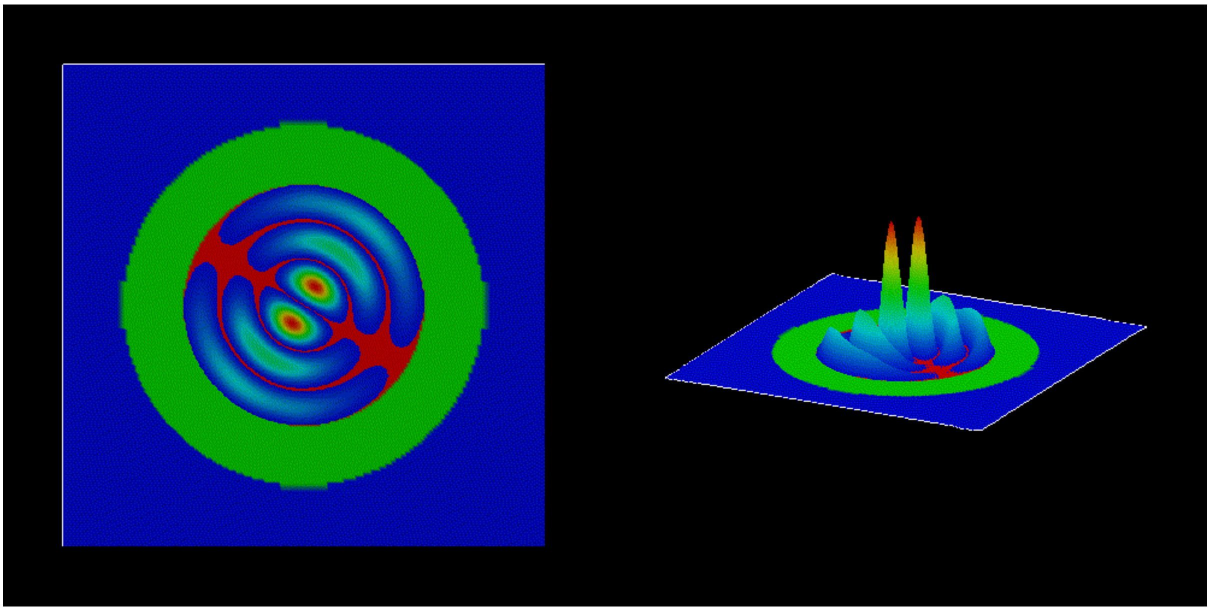
- 20th eigenstate, $(n,\ l) = (1,\ 6)$



- 22th eigenstate, $(n,\ l) = (3,\ 1)$

The parameters of the quantum corral are the followings:

- radius: $a = 40$ nm
- $m_e = 0.026m_0$
- $V(r) = 0$ for $r < a$
- $V(r) = \infty$ for $r > a$



The analytical solution of the eigenstates of this quantum well is:

$$\psi_{n,l}(r,\theta) \propto J_l\left(\frac{j_{l,n} r}{a}\right)[A\cos(l\theta) + B\sin(l\theta)] \tag{4.7.2.1}$$

where

- $J_l(x)$ is the Bessel function of the first kind (We cite them for $l = 0, 1, 2$ below.)
- $j_{l,n}$ is its zero point i.e. $J_l(j_{l,n}) = 0$ and $n = 1, 2, ...$
- A, B are constant
- $l = 0, \pm 1, \pm 2, ...$

The corresponding eigenenergies are: $E_{nl} = \frac{\hbar^2 j_{l,n}^2}{2m_e a^2}$

The Quantum number n comes from the boundary condition $\psi(a,\theta) = 0$. The requirement that ψ has the same value at $\theta = 0$ and 2π leads to the quantum number l. In the above figures of the eigenstates, we can know them through the following relations:

- (the number of zero points in the radial direction) $= n$
- (the number of zero points in the circumferential direction)/2 $= |l|$

Energy spectrum

The following figure shows the energy spectrum of the quantum corral. (The zero of energy corresponds to the InAs conduction band edge.)

The two-fold degeneracies of the states

- (2, 3), (4, 5), (7, 8), (9, 10), (11, 12), (13, 14), (16, 17), (18, 19), (20, 21), (22, 23), (24, 25), (26, 27), (28, 29), (31, 32), (33, 34), (35, 36), (37, 38), (39, 40)

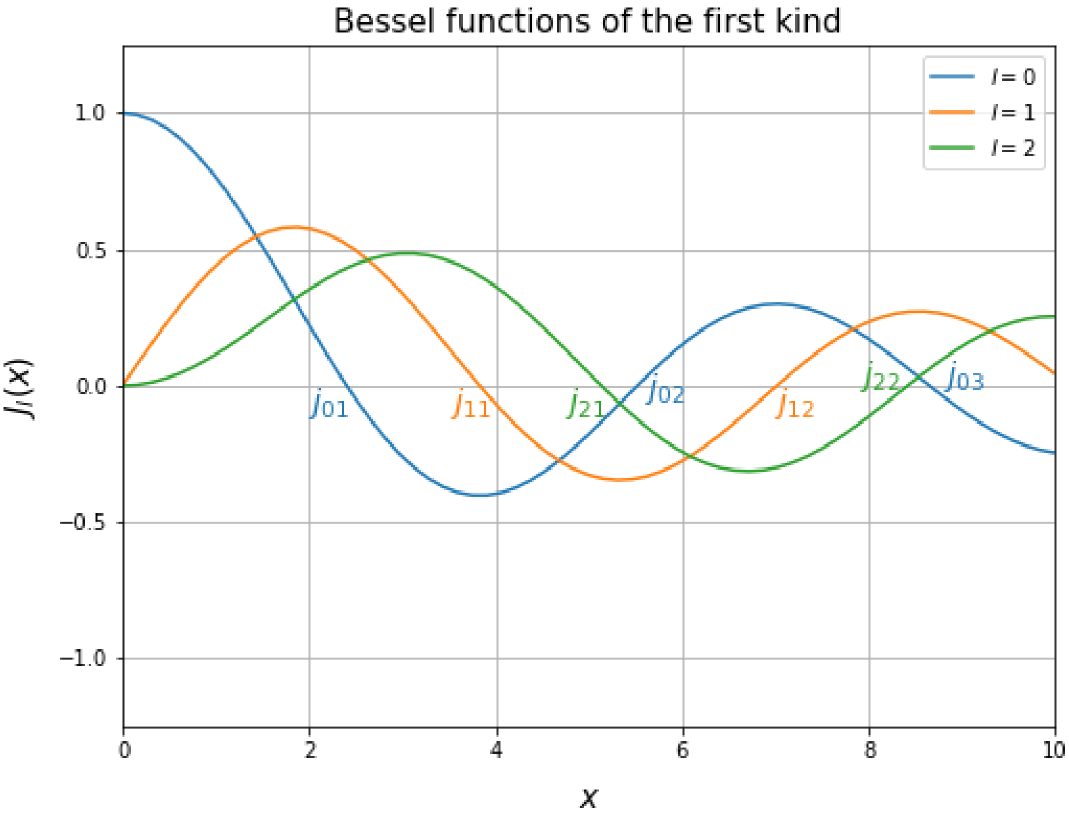
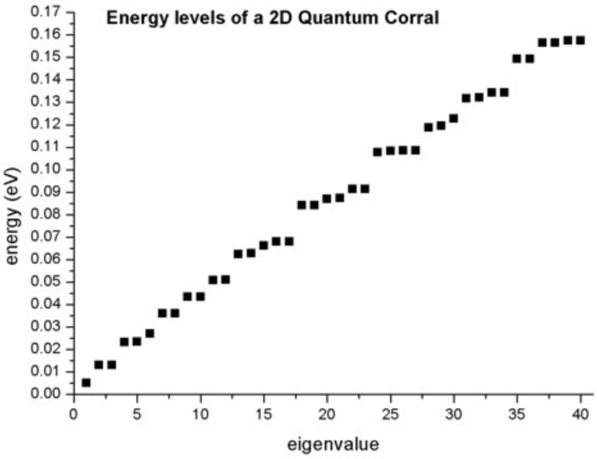


Figure 4.7.2.1: Bessel functions of the first kind for $l = 0, 1, 2$ generated by scipy.



correponds to $|l| \geq 1$. On the other hand, the non-degenerate energy eigenvalues corresponds to $l = 0$

The analytical energy values are: $E_{nl} = \frac{\hbar^2 j_{l,n}^2}{2m_e a^2}$.

There is a formula to approximate $j_{l,n}$: $j_{l,n} = (n + \frac{1}{2}|l| - \frac{1}{4})\pi$ which is accurate as $n \to \infty$.

Here we describe the comparison between the analytical values, approximate values, and *nextnano*++.

	$[n, l]$	$j_{l,n}$	$j_{l,n}$ (approx.)	$E_{n,l}$ [eV]	$E_{n,l}$ [eV] (approx.)	$E_{n,l}$ [eV] (*nextnano*++)
1st	[1, 0]	2.405	$0.75\pi \simeq 2.356$	0.00530	0.00508	0.00510
2nd	[1, 1]	3.832	$1.25\pi \simeq 3.926$	0.01345	0.01412	0.01294
3rd	[1,-1]	3.832	$1.25\pi \simeq 3.926$	0.01345	0.01412	0.01294
4th	[1, 2]	5.136	$1.75\pi \simeq 5.497$	0.02416	0.02768	0.02320
5th	[1,-2]	5.136	$1.75\pi \simeq 5.497$	0.02416	0.02768	0.02329
6th	[2, 0]	5.520	$1.75\pi \simeq 5.497$	0.02791	0.02767	0.02685
7th	[2, 1]	7.016	$2.25\pi \simeq 7.067$	0.04508	0.04574	0.03584

Further details about the analytical solution of the cylindrical quantum well with infinite barriers can be found in:

> **The Physics of Low-Dimensional Semiconductors - An Introduction**
> John H. Davies
> Cambridge University Press (1998)

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.7.3 T-shaped quantum wire grown by cleaved edge overgrowth (CEO): wave functions without strain

Note

The tutorial is related to the PhD Thesis of R. Schuster *[SchusterPhD2005]*

Header

Input files:

- *examples\quantum_wires\T-QWR_zb_III-V_Schuster_PhD_2005_2D_nnp.in*

Scope:

Electron and hole wave functions of a T-shaped quantum wire (QWR).

Output files:

- *\bias_xxxxx\Quantum\probabilities_quantum_region_Gamma.fld*
- *\bias_xxxxx\Quantum\probabilities_quantum_region_HH.fld*
- *\bias_xxxxx\Quantum\probabilities_quantum_region_LH.fld*
- *\bias_xxxxx\Quantum\probabilities_quantum_region_kp6_00000.fld*

Structure

Similar to the 1D confinement in a quantum well, it is possible to confine electrons or holes in two dimensions, i.e. in a quantum wire. In this tutorial we consider the quantum wire, which is formed at the T-shaped intersection of two 10 nm GaAs type-I quantum wells, surrounded by $\mathrm{Al}_{0.35}\mathrm{Ga}_{0.65}\mathrm{As}$ barriers (see Figure 4.7.3.1). The electrons and holes are free to move along the z direction only, thus the wire is oriented along the [0-11] direction. Such a heterostructure can be manufactured by growing the layers along two different growth directions with the CEO (cleaved egde overgrowth) technique. Due to the nearly identical lattice constants of GaAs and AlAs it is possible to assume this heterostructure as being unstrained.

The wave function is indicated at the T-shaped intersection in yellow. Here, the wave function can extend into a larger volume (as compared to the quantum well) and thus reduce its energy. Quantum mechanics tells us that the ground state can be found at this intersection and electrons are only allowed to move one-dimensionally along the z direction. Figure 4.7.3.1 b) shows a 60 nm x 60 nm extract of the schematic layout including the dimensions, the material composition and the orientation of the wire with respect to the crystal coordinate system.

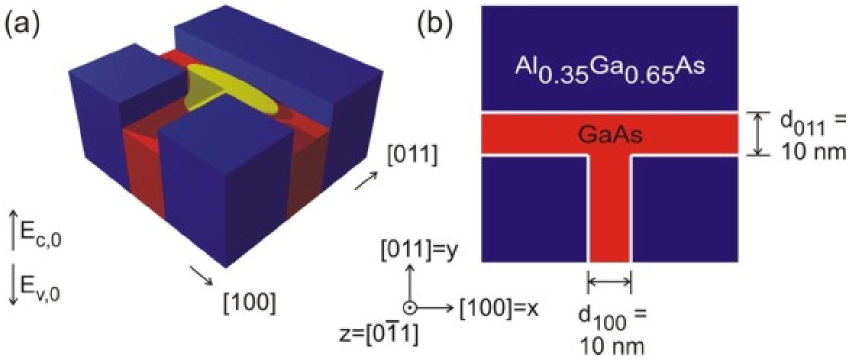


Figure 4.7.3.1: Two-dimensional conduction band edges of the T-shaped quantum wire.

Input file

It is sufficient to describe this heterostructure within a 2D simulation as it is translationally invariant along the z direction. The simulation coordinate system is oriented in the following way:

```
global{
    simulate2D{}
    crystal_zb{
        x_hkl = [1, 0, 0]
        y_hkl = [0, 1, 1]
    }
}
```

As we do not have doping and no piezoelectric fields (the structure is assumed to be unstrained) and as the temperature is assumed to be 4 K, we do not have to deal with charge redistributions. Thus, we can refrain from solving Poisson’s equation, and we also do not have to take care about self-consistency.

Material parameters of relevance are the conduction band and valence band offset between GaAs and $\mathrm{Al_{0.35}Ga_{0.65}As}$:

$$\begin{aligned}\mathrm{CBO} &= 0.2847\,\mathrm{eV}\\ \mathrm{VBO} &= -0.1926\,\mathrm{eV}\\ E_{\mathrm{gap,Al_{0.35}Ga_{0.65}As}} &= 2.2883\,\mathrm{eV}\\ E_{\mathrm{gap,GaAs}} &= 1.5193\,\mathrm{eV}\end{aligned}$$

Results

Using the input file *T-QWR_GaAs-AlGaAs_Schuster_2005_2D_nnp.in* we calculate the electron, heavy hole and light hole wavefunctions for the T-shaped quantum wire structure.

Effective mass approximation

The electron and hole wave functions can be calculated within the effective mass theory (envelope function approximation) by using position dependent effective masses. In our example, the effective masses are constant within each material but have discontinuities at the material interfaces. In *nextnano*++ the effective masses are assumed to be isotropic. Both, the heavy hole and the light hole band edge energies are degenerate but the effective masses differ. Thus, we have to solve three Schrödinger equations, namely for the conduction band, heavy hole band and light hole band. To trigger the 1-band effective mass model for calculating the eigenstates, use the following setting in the input file *T-QWR_GaAs-AlGaAs_Schuster_2005_2D_nnp.in*:

```
$kp6 = 0     # choose 1 (6 band k.p) or 0 (effective mass approximation)␣
↪(ListOfValues: 1,0)
```

In Figure 4.7.3.2 we show the normalized probability densities (ψ^2) for the electron, heavy hole and light hole ground states, which are obtained by the effective mass approximation.

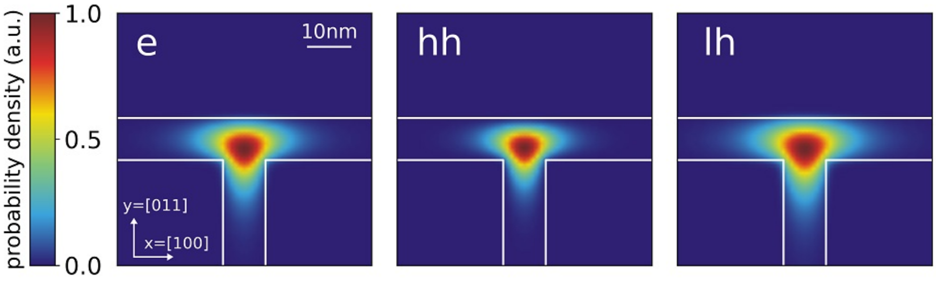


Figure 4.7.3.2: Probability densities of the electron (e), heavy hole (hh) and light hole (lh) state calculated using the effective mass approximation. The wavefunctions are normalized so that the maxima are equal to one.

In addition to these ground states for $k_z = 0$, excited states are possible as well. Similar to the subbands of a 1D quantum well that show a $E(k_x, k_y)$ dispersion one can assign a subband with the energy dispersion $E(k_z)$ to each quantum wire eigenvalue which describes the free motion along the quantum wire axis (z axis). A more advanced treatment would be to use k.p theory to calculate the eigenvalues for different k_z in order to obtain the (nonparabolic) energy dispersion $E(k_z)$.

6-band k.p approximation

For the same structure as above we perform the calculations again, but this time using the 6-band k.p model instead of the single-band effective mass approximation. To trigger the 6-band k.p model for calculating the eigenstates, the following setting in the input file *T-QWR_GaAs-AlGaAs_Schuster_2005_2D_nnp.in* can be used:

```
$kp6 = 1     # choose 1 (6 band k.p) or 0 (effective mass approximation)␣
↪(ListOfValues: 1,0)
```

Figure 4.7.3.3 shows the probability density (ψ^2) for the hole ground state. For the results shown on the left we used the following Luttinger parameters for GaAs: $\gamma_1 = 6.98$, $\gamma_2 = 2.06$, $\gamma_3 = 2.93$, which corresponds to: $L = -16.220$, $M = -3.860$, $N = -17.580$. For the results shown on the right, we modified the Luttinger parameters for GaAs to $\gamma_1 = 6.98$, $\gamma_2 = 2.06 = \gamma_3$, which corresponds to $L = -16.220$, $M = -3.860$, $N = -12.36$. Choosing $\gamma_2 = \gamma_3$ corresponds to an isotropic effective mass.

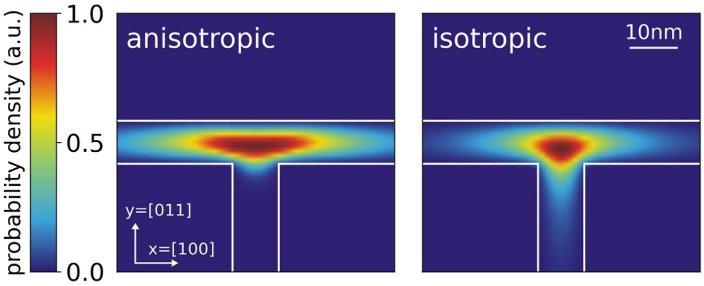


Figure 4.7.3.3: Probability density (ψ^2) for the hole ground state using anisotropic and isotropic k.p parameters.

Eigenenergies

The calculated eigenvalues for the ground states are:

effective-mass			6-band k.p
electron energy (eV)	hh energy (eV)	lh energy (eV)	hole state energy (eV)
3.006	1.455	1.437	1.455

Including anisotropic effects in the effective mass model

The effective mass m^* depends now on the chosen direction, which is described by a tensor. The components of the effective mass tensor, which are mass along the crystal coordinate axes, can be derived from the 6-band k.p parameters (or Luttinger parameters). Using the Luttinger parameters γ_1, γ_2 and γ_3, the effective masses for heavy and light holes along [110] and [010] in units of m_0 can be calculated as follows:

$$\begin{aligned}
m^*_{\text{hh},[100]} &= \frac{1}{\gamma_1 - 2\gamma_2}, \\
m^*_{\text{hh},[011]} &= \frac{1}{\gamma_1 - 0.5 \cdot (\gamma_2 + 3\gamma_3)}, \\
m^*_{\text{lh},[100]} &= \frac{1}{\gamma_1 + 2\gamma_2}, \\
m^*_{\text{lh},[011]} &= \frac{1}{\gamma_1 + 0.5 \cdot (\gamma_2 + 3\gamma_3)}.
\end{aligned}$$

The Luttinger parameters for GaAs are given by: $\gamma_1 = 6.98$, $\gamma_2 = 2.06$ and $\gamma_3 = 2.93$. The relations between the Luttinger parameters and the isotropic effective masses are

$$\begin{aligned}
m^*_{\text{hh,isotropic}} &= \frac{1}{\gamma_1 - 0.8\gamma_2 - 1.2\gamma_3}, \\
m^*_{\text{lh,isotropic}} &= \frac{1}{\gamma_1 + 0.8\gamma_2 + 1.2\gamma_3}.
\end{aligned}$$

Usually the database entries for the effective masses assume spherical symmetry for the holes and are specified with respect to the crystal coordinate system. Their default values (isotropic) and the values which were derived from the Luttinger parameters are given in this table:

	heavy hole (GaAs)	light hole (GaAs)
along [100] direction	0.350	0.090
along [011] direction	0.643	0.081
isotropic	0.551	0.082
nextnano++ database	0.500	0.068

In this tutorial, however, we calculated the effective masses for different directions and, therefore, we do not have spherical symmetry anymore. Thus, we have to rotate the new eigenvalues of the effective mass tensors that are given in the x = [100], y = [011], z = [0-11] simulation coordinate system into the crystal coordinate system where x_{cr} = [100], y_{cr} = [010], z_{cr} = [001]. First, we have to overwrite the default entries in the database so that they contain the eigenvalues of the effective mass tensors in the simulation system:

```
valence-band-masses      = 0.350d0  0.643d0  0.643d0 ! eigenvalues of the heavy hole␣
↪effective mass tensor [100] [011] [0-11]
                           0.090d0  0.081d0  0.081d0 ! eigenvalues of the light  ␣
↪hole effective mass tensor  [100] [011] [0-11]
```

To project these eigenvalues onto the crystal coordinate system we need to know the principal axis system which these eigenvalues refer to (The normalization of these vectors will be done internally by the program):

```
principal-axes-vb-masses = 1d0      0d0      0d0     ! heavy hole [100]
                           0d0      1d0      1d0     !            [011]
                           0d0     -1d0      1d0     !            [0-11]

                           1d0      0d0      0d0     ! light hole [100]
                           0d0      1d0      1d0     !            [011]
                           0d0     -1d0      1d0     !            [0-11]
```

Figure 4.7.3.4 and Figure 4.7.3.5 show the probability densities (ψ^2) of the ground states of the confined electron, heavy and light hole eigenstates of the quantum wire. The lowest hole state is the heavy hole state and the second hole state is the light hole state. No further hole states are confined. Also, in the conduction band only the ground state is confined. One can clearly see that each ground state wave function is localized at the T-shaped intersection and shows the T-shaped symmetry. Due to the anisotropy of the heavy hole effective mass, the heavy hole wave function prefers to extend along the [100] direction and hardly penetrates into the quantum well that is aligned along the [011] direction. The heavy hole mass along the [100] direction is only half the value as along the [011] direction. The light hole anisotropy is only minor and thus its symmetry resembles the one of the isotropic electron.

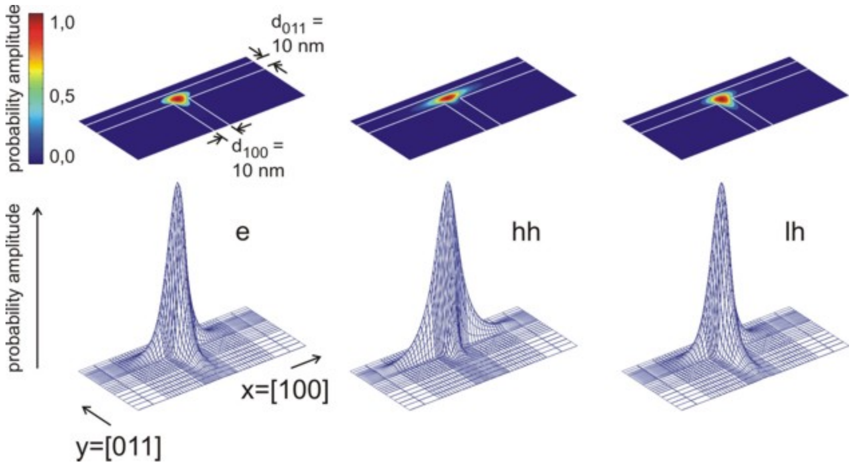


Figure 4.7.3.4: Probability amplitudes of the electron (e), heavy hole (hh) and light hole (lh) envelope functions at an unstrained T-shaped intersection of two 10 nm wide GaAs quantum wells embedded by $\mathrm{Al}_{0.35}\mathrm{Ga}_{0}.65\mathrm{As}$ barriers. The wave functions are normalized so that the maxima are equal to one.

These results are in very good qualitative agreement with the heavy hole and light hole wave functions calculated

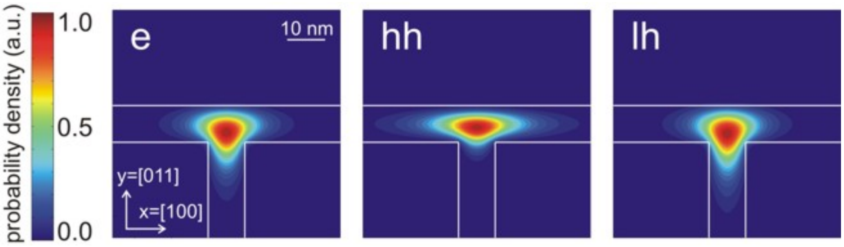


Figure 4.7.3.5: Contour diagram of the probability amplitudes of the electron (e), heavy hole (hh) and light hole (lh) eigenfunctions (same figures as Figure 4.7.3.4, but this time viewed from the top). The wave functions are normalized so that the maxima are equal to one.

within the 6-band k.p calculation This demonstrates the impact of an isotropic (for electrons and light holes) or anisotropic (for heavy hole) effective mass on the obtained wavefunctions.

Acknowledgement:
We would like to thank Robert Schuster from the University of Regensburg for providing experimental data and some figures for this tutorial.

This tutorial also exists for *nextnano³*.

Last update: 27/05/2025

4.7.4 T-shaped quantum wire grown by cleaved edge overgrowth (CEO): wave functions and strain

Note

The tutorial is related to the PhD Thesis of R. Schuster *[SchusterPhD2005]*

Header

Input files:

- *examples\quantum_wires\T-QWR_zb_III-V_Schuster_PhD_2005_1D_nnp_strained-QW.in*
- *examples\quantum_wires\T-QWR_zb_III-V_Schuster_PhD_2005_2D_nnp_strained.in*

Scope:

Strained quantum wires including a discussion of the strain calculation and the strain-induced piezoelectric fields (Poisson equation).

Output files:

- *Strain\hydrostatic_strain.fld* (hydrostatic strain)

- *Strain\strain_*.fld* (strain components)
- *Strain\density_piezoelectric_charges.fld* (piezoelectric charge density)
- *bias_xxxxx\bandedges.fld* (bandedge profiles)
- *bias_xxxxx\Quantum\probabilities_quantum_region_*.fld* (wavefunctions)

Similar to the 1D confinement in a quantum well, it is possible to confine electrons or holes in two dimensions, i.e. in a quantum wire. In this tutorial we consider a quantum wire, which is formed at the T-shaped intersection of a 10 nm GaAs type-I quantum well and a 10 nm $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ barrier. The T-shaped intersection is surrounded by $\mathrm{Al}_{0.3}\mathrm{Ga}_{0.7}\mathrm{As}$ which acts as a barrier to GaAs. The $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ barrier has a larger lattice constant than $\mathrm{Al}_{0.3}\mathrm{Ga}_{0.7}\mathrm{As}$ and is thus strained. The strain affects the GaAs well and thus produces a local decrease (increase) in the conduction (valence) band edge energy and thus confines electrons (holes) at the T-shaped intersection. The electrons and holes are free to move along the z direction only, thus, the wire is oriented along the [0-11] direction. Such a heterostructure can be manufactured by growing the layers along two different growth directions with the CEO (cleaved edge overgrowth) technique. Figure 4.7.4.1 shows the sample layout.

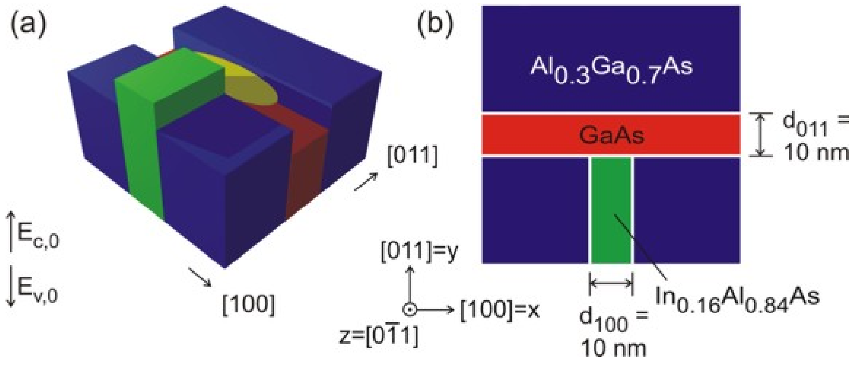


Figure 4.7.4.1: In (a) the two-dimensional conduction band edges of the T-shaped quantum wire without considering strain effects is shown. If one inverts the energy arrow then the left picture corresponds to the valence band edge. The wave function is indicated at the T-shaped intersection in yellow. In (b) a 60 nm x 60 nm extract of the schematic layout including the dimensions, the material composition and the orientation of the wire with respect to the crystal coordinate system is shown.

It is useful to compare the structure above with the *T-shaped quantum wire tutorial*, which consists of two GaAs quantum wells rather than one GaAs well and one $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ barrier (see Figure 4.7.4.2), in order to understand the fundamental difference between these two layouts. As we see in from Figure 4.7.4.2 the wave function can extend into a larger volume as compared to the quantum well and thus reduces its energy. So quantum mechanics tells us that the ground state can be found at this intersection and electrons are only allowed to move one-dimensionally along the z direction. For Figure 4.7.4.1 however this is not true. The confinement only occurs if one takes into account the strain which decreases (increases) the conduction (valence) band edge energy in GaAs at the T-shaped intersection.

Calculation of the strain tensor

First, we have to calculate the strain tensor by minimizing the elastic energy within continuum elasticity theory. Along the translationally invariant z direction the lattice commensurability constraint forced the $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ layer to adopt the lattice constant of $\mathrm{Al}_{0.3}\mathrm{Ga}_{0.7}\mathrm{As}$. The model for strain calculations can be specified inside the `strain{ }` group, where we choose the model: `minimized_strain{ }`.

In Figure 4.7.4.3 the calculated hydrostatic strain $\epsilon_{\mathrm{hyd}} = \epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}$ (trace of the strain tensor) inside the structure is shown. The hydrostatic strain has its maximum at the intersection, where it leads to a reduced band gap, which is the requirement for confining the charge carriers. Thus, the quantum wire is formed in the GaAs quantum well due to the tensile strain field induced by the $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ layer.

Note that in a one-dimensional example, which is provided in the input file *T-QWR_zb_III-V_Schuster_PhD_2005_1D_nnp_strained-QW.in*, the strain tensor components of a $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ layer

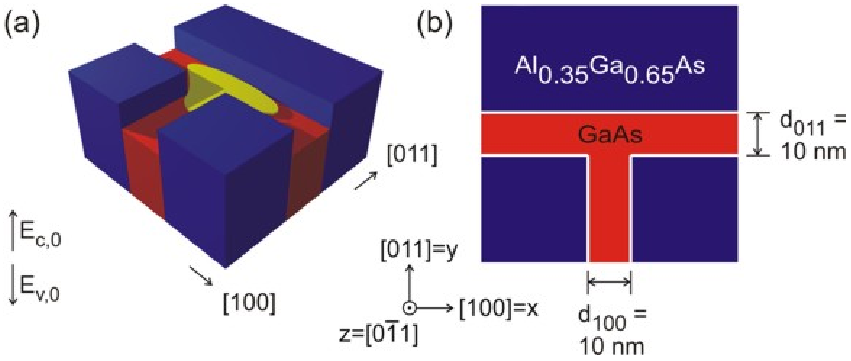


Figure 4.7.4.2: In (a) the two-dimensional conduction band edges of the T-shaped quantum wire (from the T-shaped quantum wire tutorial) without considering strain effects is shown. The wave function is indicated at the T-shaped intersection in yellow. In (b) a 60 nm x 60 nm extract of the schematic layout including the dimensions, the material composition and the orientation of the wire with respect to the crystal coordinate system is shown.

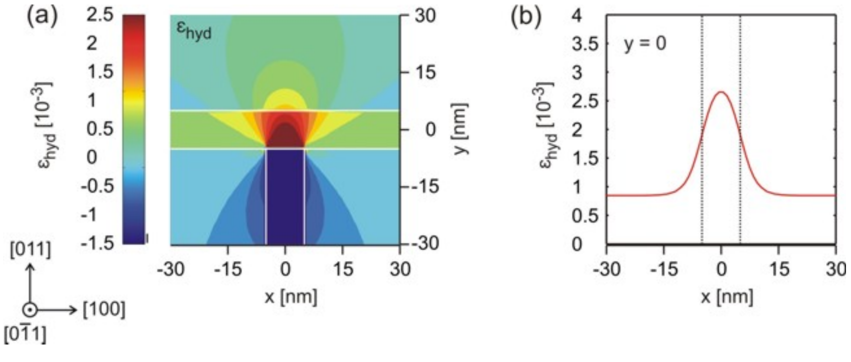


Figure 4.7.4.3: In (a) the hydrostatic strain $\epsilon_{\rm hyd}$ inside the T-shaped quantum wire structure is shown. In (b) a cross-section of $\epsilon_{\rm hyd}$ along x at $y = 0$ is shown.

that is strained pseudomorphically with respect to an $\mathrm{Al_{0.30}Ga_{0.7}As}$ substrate are the following:

$$\begin{aligned}\epsilon_{xx} &= 10.9 \cdot 10^{-3}\\ \epsilon_{yy} &= \epsilon_{zz} = -12.4 \cdot 10^{-3}\\ \epsilon_{xy} &= \epsilon_{xz} = \epsilon_{yz} = 0\\ \epsilon_{\mathrm{hyd}} &= \mathrm{Tr}(\epsilon_{ij}) = -13.9 \cdot 10^{-3}\end{aligned}$$

Here, the growth direction is along the x direction, i.e. along [100]. The temperature is assumed to be $40\,\mathrm{K}$ and the lattice constants are assumed to be temperature dependent (i.e. we use the $40\,\mathrm{K}$ lattice constants).

In Figure 4.7.4.4 the individual strain tensor components (ϵ_{xx}, ϵ_{yy}, ϵ_{xy}) with respect to the simulation coordinate system are presented. In our 2D simulation, the sample layout is homogeneous along the z direction, i.e. the lattice constant of $\mathrm{In_{0.16}Al_{0.84}As}$ is forced to have the same lattice constant as $\mathrm{Al_{0.3}Ga_{0.7}As}$ along the z direction. Then the strain tensor component must be $\epsilon_{zz} = -12.4 \cdot 10^{-3}$, in agreement with our 1D example, i.e. $\mathrm{In_{0.16}Al_{0.84}As}$, which has a larger lattice constant than $\mathrm{Al_{0.3}Ga_{0.7}As}$ is strained compressively along the z direction. Similar to the 1D case, it is also expected that the ϵ_{yy} component inside the $\mathrm{In_{0.16}Al_{0.84}As}$ barrier has a similar value to ϵ_{zz}, which is clearly the case. The dark blue area in Figure 4.7.4.4 (c) thus has a value around $-12 \cdot 10^{-3}$. However, this value deviates from the ideal 1D value at the T-shaped intersection as expected (see also Figure 4.7.4.5). The same applies to the value of ϵ_{xx}, which is similar to the 1D value inside the $\mathrm{In_{0.16}Al_{0.84}As}$ barrier: $\epsilon_{xx} = 11 \cdot 10^{-3}$. The strain tensor components ϵ_{xz} and ϵ_{yz} with respect to the simulation coordinate system are equal to zero as in our 1D example.

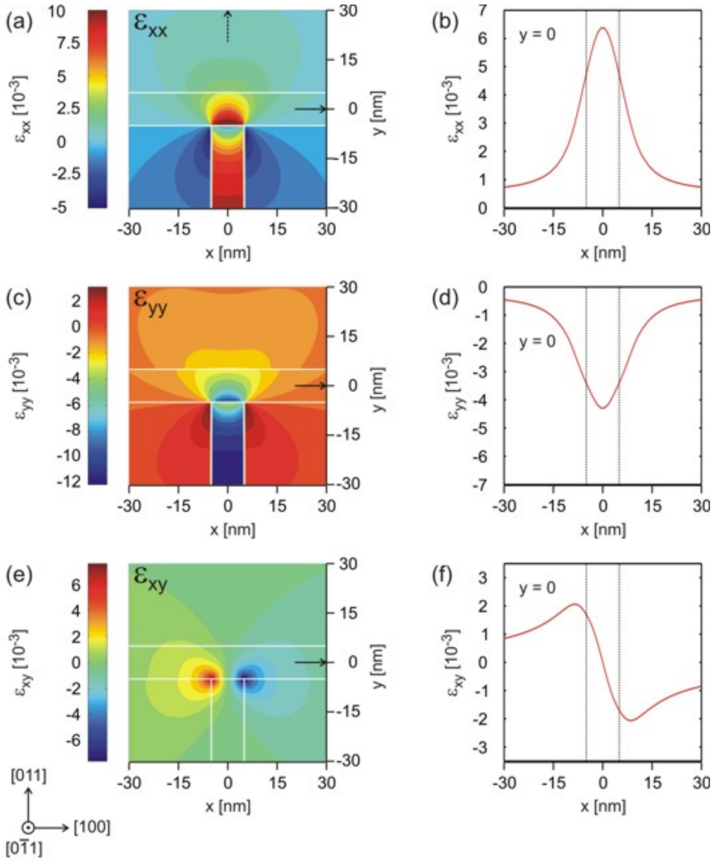


Figure 4.7.4.4: In (a), (c), (e) the strain components ϵ_{xx}, ϵ_{yy}, ϵ_{xy} are shown. In (b), (d), (f) a cut through the structure along x at $y = 0$ is shown.

The important difference with respect to the 1D case is the existence of a non-vanishing strain tensor component ϵ_{xy} which brakes the symmetry of the sample layout. Usually, the ϵ_{xy} component is attributed to be responsible for piezoelectricity. However, note that in the discussion before all strain tensor components refer to the simulation coordinate system (and not to the crystal coordinate system). So we have to plot the off-diagonal strain tensor

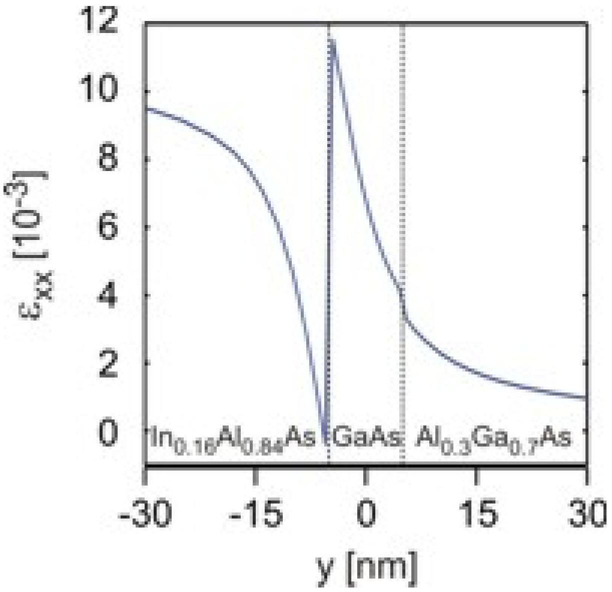


Figure 4.7.4.5: Strain tensor component ϵ_{xx} along y direction at position $x = 0$.

components that are expressed with respect the crystal coordinate system orientation and then check if the off-diagonal components are non-zero, which is clearly the case as we can see from Figure 4.7.4.6.

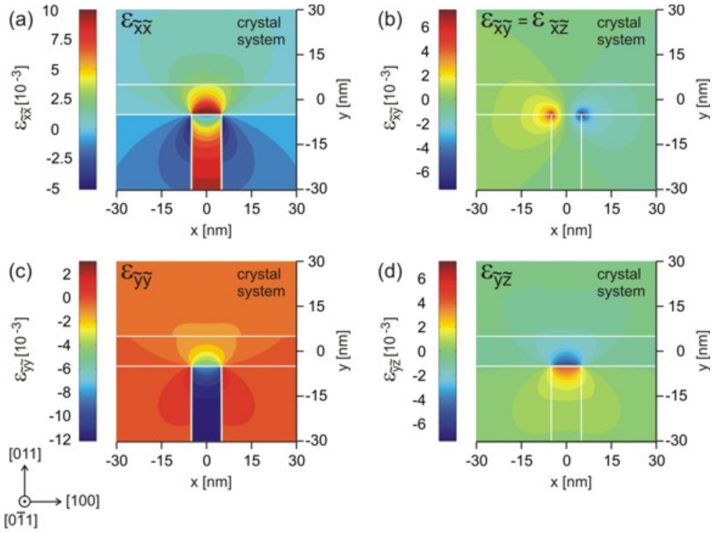


Figure 4.7.4.6: Strain tensor components $\epsilon_{\tilde{x}\tilde{x}}$, $\epsilon_{\tilde{y}\tilde{y}}$, $\epsilon_{\tilde{x}\tilde{y}} = \epsilon_{\tilde{x}\tilde{z}}$ and $\epsilon_{\tilde{y}\tilde{z}}$ with respect to the crystal coordinate system. The rotation with respect to the simulation system is a rotation of 45 degrees around the x axis, i.e. the [100] axis.

By comparing Figure 4.7.4.4 (a) and Figure 4.7.4.6 (a) we observe that $\epsilon_{\tilde{x}\tilde{x}} = \epsilon_{xx}$, because the x coordinate axes coincide. Symmetry arguments show that the following holds:

$$\epsilon_{\tilde{y}\tilde{y}} = \frac{1}{2}\left(\epsilon_{yy} + \epsilon_{zz}\right)$$

$$\epsilon_{\tilde{x}\tilde{y}} = \epsilon_{\tilde{x}\tilde{z}} = \frac{1}{\sqrt{2}}\epsilon_{xy}$$

Calculation of the piezoelectric charge density

The off-diagonal strain tensor components $\epsilon_{\tilde{x}\tilde{y}}$, $\epsilon_{\tilde{x}\tilde{z}}$ and $\epsilon_{\tilde{y}\tilde{z}}$ are responsible for the piezoelectric polarization $\mathbf{P}_{\mathrm{piezo}}$, given by

$$\mathbf{P}_{\mathrm{piezo}} = e_{14} \begin{pmatrix} 2\epsilon_{\tilde{y}\tilde{z}} \\ 2\epsilon_{\tilde{x}\tilde{z}} \\ 2\epsilon_{\tilde{x}\tilde{y}} \end{pmatrix},$$

where e_{14} is the piezoelectric constant in units of $[\mathbf{C}/\mathbf{m^2}]$. Once having determined the piezoelectric polarization, one is able to compute the piezoelectric charge density:

$$\rho_{\mathrm{piezo}}(x,y) = -\mathrm{div}\,\mathbf{P}_{\mathrm{piezo}}(x,y).$$

In Figure 4.7.4.7 the piezo electric charge density inside the quantum wire structure is shown. The strain-induced piezoelectric fields are then obtained from ρ_{piezo} by solving Poisson's equation.

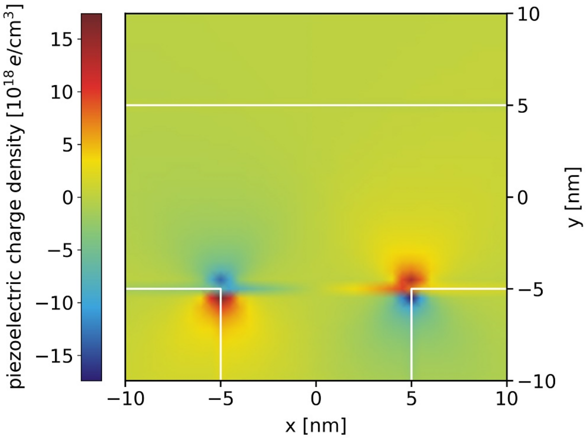


Figure 4.7.4.7: Piezoelectric charge density $\rho_{\mathrm{piezo}}(x,y)$.

Calculation of the conduction and valence band edges

In Figure 4.7.4.8 the conduction and valence band edges of the structure are shown. The conduction and valence band edges were determined by taking into account the shifts and splittings due to the relevant deformation potentials as well as the changes due to the piezoelectric fields. We observe that the electron feels a conduction band minimum which is located left with respect to the T-shaped intersection. For the valance bands, we see that the valence band maximum for the heavy hole is not at the same position as the valence band maximum for the light hole.

Electron and heavy hole wave functions

Figure 4.7.4.9 shows the square of the electron (e) and heavy hole (hh) wave functions (i.e. ψ^2). They were calculated within the effective-mass approximation (single-band).

In Figure 4.7.4.9 (a) the piezoelectric effect was not included. As one can clearly see in Figure 4.7.4.9 (b), the piezoelectric effect destroys the symmetry of the sample layout. The piezoelectric field results from the ϵ_{xy} strain tensor component which is also not symmetric with respect to the T-shaped geometry.

Acknowledgement:

We would like to thank Robert Schuster from the University of Regensburg for providing experimental data and some figures for this tutorial.

Last update: 13/09/2024

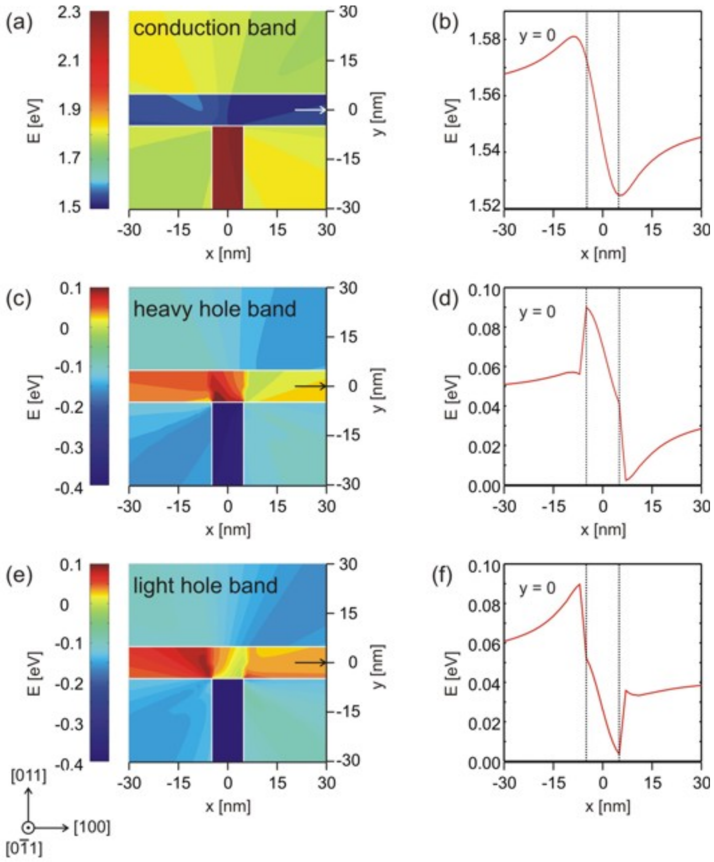


Figure 4.7.4.8: In (a), (c), (e) a 2D plot of the conduction, heavy hole and light hole band edge energies are shown. In (b), (d), (f) a cut through the conduction, heavy hole and light hole band edge energies at $y = 0$.

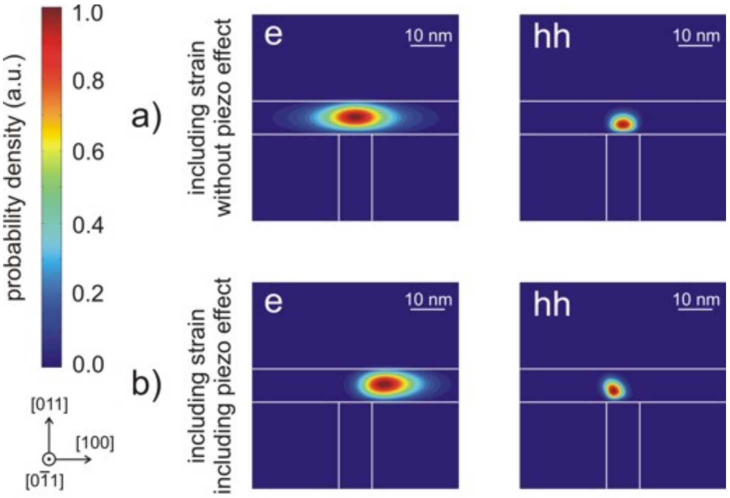


Figure 4.7.4.9: In (a) the contour diagram of the square of the electron (e) and heavy hole (hh) wave functions (i.e. ψ^2) for the case where strain is included in the simulations, but piezoelectricity is not. Subplot (b) shows the same results as in (a), but this time including the piezoelectric effect. Note that in the plot the wave functions are normalized so that the maximum equals one, respectively.

4.8 Quantum Dots

4.8.1 Energy levels in idealistic 3D cubic and cuboidal shaped quantum dots

Input files:

- *3D_wave_functions_cubic_QD_nnp.in*
- *3D_wave_functions_cuboid_QD_nnp.in*
- *3D_wave_functions_cubic_QD_nn3.in*
- *3D_wave_functions_cuboid_QD_nn3.in*

Scope:

The energy levels and the wave functions of a cubic and cuboidal quantum dot

Output files:

- *bias_00000\Quantum\energy_spectrum_quantum_region_Gamma.dat*
- *bias_00000\Quantum\probability_quantum_region_Gamma_xxxx.fld*

Energy levels in an idealistic 3D cubic quantum dot

Input file: *3D_wave_functions_cubic_QD_nnp.in*

Here, we want to calculate the energy levels and the wave functions of a cubic quantum dot with lengths L_x = L_y = L_z = 10 nm. We assume that the barriers at the QD boundaries are infinite. This way we can compare our numerical calculations to analytical results. The potential inside the QD is assumed to be 0 eV. As effective mass we take the electron effective mass of $InAs$, i.e. m_e = 0.026 m_0.

A discussion of the analytical solution of the 3D Schrödinger equation of a particle in a box (i.e. quantum dot) with infinite barriers can be found in e.g. *[MitinKochelapStroscio1999]*. The solution of the Schrödinger equation leads to the following eigenvalues:

$$\begin{aligned} E_{n_1,n_2,n_3} &= \frac{\hbar^2\pi^2}{2m_e}\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 1.4462697\cdot 10^{-17}\ \mathrm{eVm}^2\cdot\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 0.1446269\ \mathrm{eV}\cdot(n_1^2+n_2^2+n_3^2) \end{aligned} \tag{4.8.1.1}$$

where

- E_{n_1,n_2,n_3} is the total electron energy,
- n_1, n_2 and n_3 are three discrete quantum numbers(because we have three directions of quantization) and
- L_x, L_y and L_z are the lengths along the x, y and z directions.

In the last line of eq. (4.8.1.1) we used the fact that L_x = L_y = L_z and factored out 1/(10 nm)^2.

Generally, the energy levels are not degenerate, i.e. all energies are different. However, some energy levels with different quantum numbers coincide, if the lengths along two or three directions are identical or if their ratios are integers. In our cubic QD case, all three lengths are identical. Consequently, we expect the following degeneracies:

- E_{111} = 0.43388 eV (ground state)
- E_{112} = E_{121} = E_{211} = 0.86776 eV = $2E_{111}$
- E_{122} = E_{212} = E_{221} = 1.30164 eV = $3E_{111}$
- E_{113} = E_{131} = E_{311} = 1.59090 eV = $11/3E_{111}$
- E_{222} = 1.73552 eV = $4E_{111}$
- E_{123} = E_{132} = E_{213} = E_{231} = E_{312} = E_{321} = 2.02478 eV = $14/3E_{111}$
- E_{333} = 3.90493 eV = $17/3E_{111}$

The *nextnano*++ numerical results for a 10 nm cubic quantum dot with 0.50 nm grid spacing (The grid spacing is rather coarse but has the advantage that the calculation takes only a few seconds.):

```
num_ev: eigenvalue [eV]:
        (0.50 nm grid)
 1      0.432989 = E111
 2      0.862425      (three-fold degenerate) E112/E121/E211
 3      0.862425      (three-fold degenerate) E112/E121/E211
 4      0.862425      (three-fold degenerate) E112/E121/E211
 5      1.291860      (three-fold degenerate) E122/E212/E221
 6      1.291860      (three-fold degenerate) E122/E212/E221
 7      1.291860      (three-fold degenerate) E122/E212/E221
 8      1.566392      (three-fold degenerate) E113/E131/E311
 9      1.566392      (three-fold degenerate) E113/E131/E311
10      1.566392      (three-fold degenerate) E113/E131/E311
11      1.721296 = E222
12      1.995828      (six-fold degenerate)  E123/E132/E213/E231/E312/E321
13      1.995828      (six-fold degenerate)  E123/E132/E213/E231/E312/E321
14      1.995828      (six-fold degenerate)  E123/E132/E213/E231/E312/E321
15      1.995828      (six-fold degenerate)  E123/E132/E213/E231/E312/E321
16      1.995828      (six-fold degenerate)  E123/E132/E213/E231/E312/E321
17      1.995828      (six-fold degenerate)  E123/E132/E213/E231/E312/E321
18      2.425263      (three-fold degenerate) E223/E232/E322
19      2.425263      (three-fold degenerate) E223/E232/E322
20      2.425263      (three-fold degenerate) E223/E232/E322
21      2.527557      (three-fold degenerate) E114/E141/E411
22      2.527557      (three-fold degenerate) E114/E141/E411
23      2.527557      (three-fold degenerate) E114/E141/E411
24      2.699795      (three-fold degenerate) E233/E323/E332
25      2.699795      (three-fold degenerate) E233/E323/E332
26      2.699795      (three-fold degenerate) E233/E323/E332
27      2.956993      (six-fold degenerate)  E124/E142/E214/E241/E412/E421
28      2.956993      (six-fold degenerate)  E124/E142/E214/E241/E412/E421
29      2.956993      (six-fold degenerate)  E124/E142/E214/E241/E412/E421
30      2.956993      (six-fold degenerate)  E124/E142/E214/E241/E412/E421
31      2.956993      (six-fold degenerate)  E124/E142/E214/E241/E412/E421
32      2.956993      (six-fold degenerate)  E124/E142/E214/E241/E412/E421
...
48      3.833198 = E333
...
```

Figure 4.8.1.1 show the isosurfaces of the electron wave function (Ψ^2) of the ground state and the 11th state, respectively. Both states are nondegenerate.

The 2D slices of the probability density of the states from the figure above is shown in the Figure 4.8.1.2.

Intraband (= intersubband) transitions

```
quantum{
    region{
        ...
        momentum_matrix_elements{
            direction = [0,0,1]                 # along z direction
            Gamma{}                             # Calculates the matrix element < psi_
↪f* | p_z  | psi_i > for electron states at Gamma.
            output_oscillator_strengths = yes   # Output oscillator strength f_fi
```

(continues on next page)

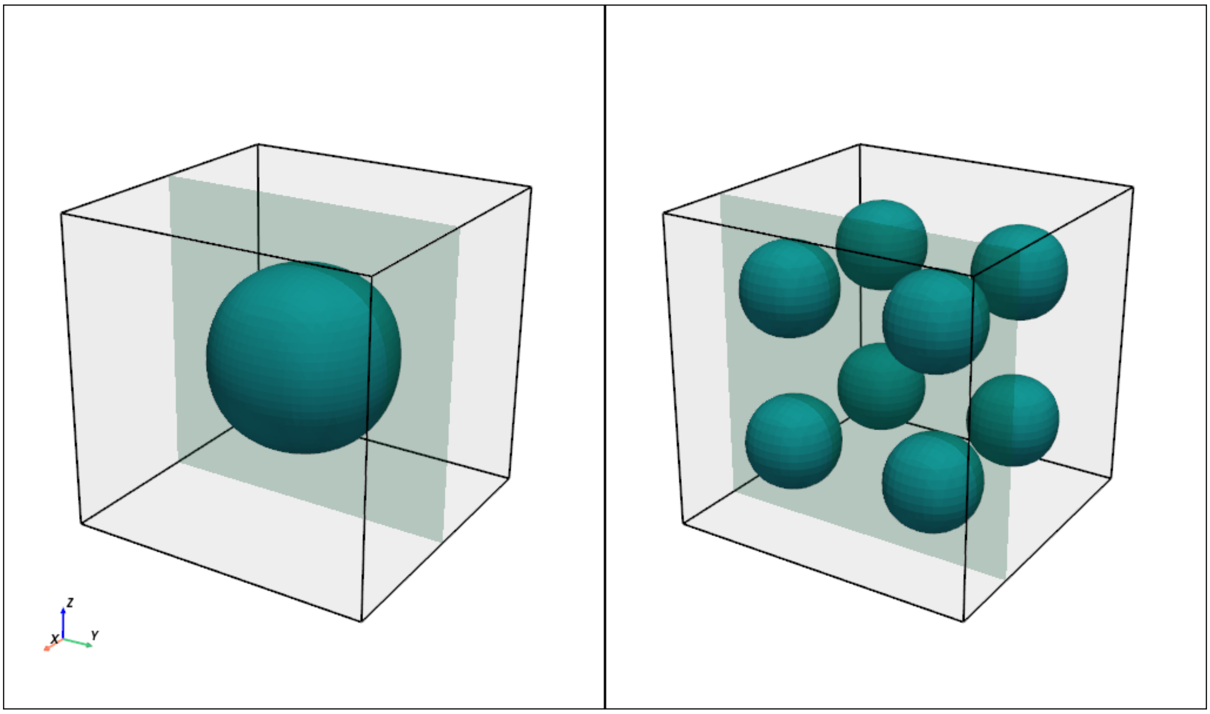


Figure 4.8.1.1: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm cubic quantum dot with infinite barriers for the ground state E_{111} (left) and the 11th eigenstate E_{222} (right). The isosurface is set at the value of 0.005. The green shadow plane is a slice at (left) and (right).

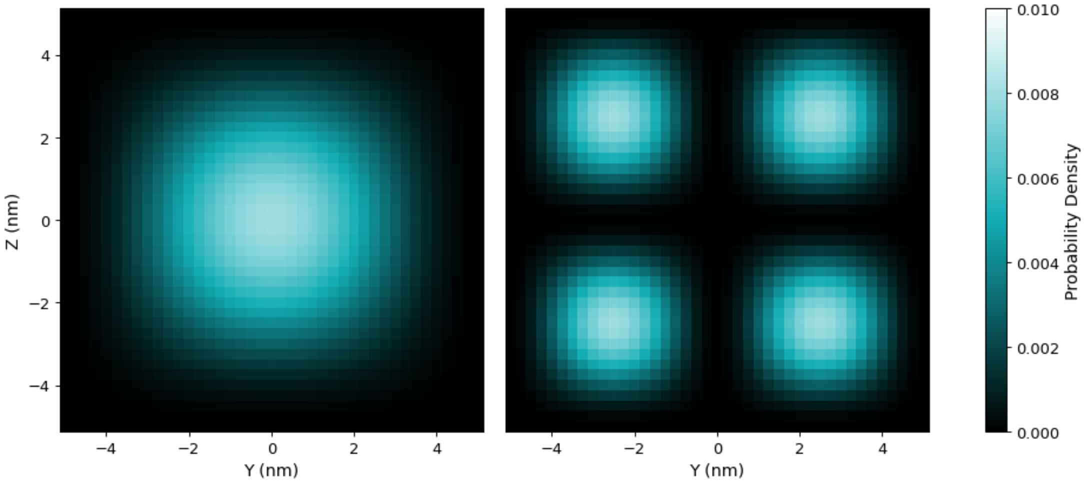


Figure 4.8.1.2: The probability density of the ground state E_{111} at $x = 0nm$ (left) and the 11th eigenstate E_{222} at $x = 2.5nm$ (right) of a 10 nm cubic quantum dot with infinite barriers.

(continued from previous page)

```
        }
    }
}
```

In this *cubic* QD with *infinite* barriers, optical intraband transitions are only **allowed** between states with **odd** difference quantum numbers along the same axes:

$E_{111} \Leftrightarrow E_{112}/ E_{121}/ E_{211}$	$1 \Leftrightarrow$ 2 / 3 / 4
$E_{111} \Leftrightarrow E_{114}/ E_{141}/ E_{411}$	$1 \Leftrightarrow$ 21 / 22 / 23
$E_{211} \Leftrightarrow E_{311}$	$2 \Leftrightarrow$ 8
$E_{121} \Leftrightarrow E_{131}$	$3 \Leftrightarrow$ 9
$E_{112} \Leftrightarrow E_{113}$	$4 \Leftrightarrow$ 10

The following transitions are **forbidden**:

$E_{111} \Leftrightarrow E_{113}/ E_{131}/ E_{311}$	$1 \Leftrightarrow$ 8 / 9 / 10
$E_{211} \Leftrightarrow E_{112}/ E_{121}$	$2 \Leftrightarrow$ 3 / 4
$E_{121} \Leftrightarrow E_{211}/ E_{112}$	$3 \Leftrightarrow$ 2 / 4
$E_{112} \Leftrightarrow E_{211}/ E_{121}$	$4 \Leftrightarrow$ 2 / 4

Energy levels in an idealistic 3D cuboidal shaped quantum dot with $L_x = L_y \neq L_z$

Input file: *3D_wave_functions_cuboid_QD_nnp.in*

This time we use a similar quantum dot as above, but the lengths are now L_x = L_y = 10 nm and L_z = 5 nm. Therefore, the degeneracies of the eigenenergies are different. We expect the following:

$$\begin{aligned} E_{n_1,n_2,n_3} &= \frac{\hbar^2\pi^2}{2m_e}\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 1.4462697 \cdot 10^{-17}\ \mathrm{eVm}^2 \cdot \left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 0.1446269\ \mathrm{eV} \cdot (n_1^2+n_2^2) + 0.5785079\ \mathrm{eV} \cdot n_3^2 \end{aligned} \tag{4.8.1.2}$$

Generally, the energy levels are **not degenerate**, i.e. all energies are different. However, some energy levels with different quantum numbers **coincide**, if the lengths along two or three directions are identical or if their ratios are integers. In our cubic QD case, all three lengths are identical. Consequently, we expect the following degeneracies:

- E_{111} = 0.86776 eV (ground state)
- E_{121} = E_{211} = 1.301642 eV
- E_{221} = 1.73552 eV = $2E_{111}$ (This is a coincidence because $L_{x,y}$ / L_z are integers and have the value 2.)
- E_{131} = E_{311} = 2.02478 eV
- E_{231} = E_{321} = 2.45866 eV
- E_{112} = 2.60329 eV = $2E_{121}$ (This is a coincidence because $L_{x,y}$ / L_z are integers and have the value 2.)
- E_{122} = E_{212} = E_{141} = E_{411} = 3.03717 eV (This is a coincidence because $L_{x,y}$ / L_z are integers and have the value 2.)
- E_{331} = 3.18180 eV
- E_{222} = $2E_{221}$ = E_{241} = E_{421} = 3.47105 eV (This is a coincidence because $L_{x,y}$ / L_z are integers and have the value 2.)
- E_{132} = E_{312} = 3.76030 eV
- E_{341} = E_{431} = E_{232} = E_{322} = 4.19418 eV (This is a coincidence because $L_{x,y}$ / L_z are integers and have the value 2.)

- E_{151} = E_{511} = 4.33881 eV
- E_{142} = E_{412} = E_{251} = E_{521} = 4.77269 eV (This is a coincidence because $L_{x,y}$ / L_z are integers and have the value 2.)
- E_{332} = 4.91731 eV
- E_{441} = E_{242} = E_{422} = 5.20657 eV (This is a coincidence because $L_{x,y}$ / L_z are integers and have the value 2.)
- E_{113} = 5.49582 eV
- E_{123} = 5.92971 eV

The *nextnano*++ numerical results for a 10 nm cubic quantum dot with 0.50 nm grid spacing (left column) and 0.25 nm grid spacing (right column). (The grid spacing is rather coarse (for 0.50 nm) but has the advantage that the calculation takes only a few seconds.)

```
num_ev: eigenvalue [eV]:
       (0.50 nm grid)       (0.25 nm grid)
 1      0.862425             0.866424      = E111
 2      1.291860             1.299191      (two-fold degenerate) = E121/E211
 3      1.291860             1.299191      (two-fold degenerate) = E121/E211
 4      1.721296             1.731958      = E221
 5      1.995828             2.017504      (two-fold degenerate) = E131/E311
 6      1.995828             2.017504      (two-fold degenerate) = E131/E311
 7      2.425263             2.450270      (two-fold degenerate) = E231/E321
 8      2.425263             2.450270      (two-fold degenerate) = E231/E321
 9      2.527557             2.584167      = E112
10      2.956993             3.016933      (four-fold degenerate) = E122/E212/E141/
↪E411
11      2.956993             3.016933      (four-fold degenerate) = E122/E212/E141/
↪E411
12      2.956993             3.016933      (four-fold degenerate) = E122/E212/E141/
↪E411
13      2.956993             3.016933      (four-fold degenerate) = E122/E212/E141/
↪E411
14      3.129231             3.168583      = E331
15      3.386428             3.449700      (three-fold degenerate) = E222/E241/E421
16      3.386428             3.449700      (three-fold degenerate) = E222/E241/E421
17      3.386428             3.449700      (three-fold degenerate) = E222/E241/E421
18      3.660960             3.735246      (two-fold degenerate) = E132/E312
19      3.660960             3.735246      (two-fold degenerate) = E132/E312
20      4.090396             4.168013      (four-fold degenerate) = E341/E431/E232/
↪E322
21      4.090396             4.168013      (four-fold degenerate) = E341/E431/E232/
↪E322
22      4.090396             4.168013      (four-fold degenerate) = E341/E431/E232/
↪E322
23      4.090396             4.168013      (four-fold degenerate) = E341/E431/E232/
↪E322
24      4.151688             4.291319      (two-fold degenerate) = E151/E511
25      4.151688             4.291319      (two-fold degenerate) = E151/E511
26      4.581124             4.724086      (four-fold degenerate in theory) = E142/
↪E412/E251/E521
27      4.581124             4.724086      (four-fold degenerate in theory) = E142/
↪E412/E251/E521
28      4.622125             4.734676      (four-fold degenerate in theory) = E142/
↪E412/E251/E521
29      4.622125             4.734676      (four-fold degenerate in theory) = E142/
↪E412/E251/E521
```

(continues on next page)

(continued from previous page)

```
30      4.794363              4.886326      = E332
...
34      5.121061              5.400036      = E441
```

The Figure 4.8.1.3 show the isosurfaces of the electron wave function (ψ^2) of the 1st, 4th, 9th and 14th, respectively. All these states are nondegenerate.

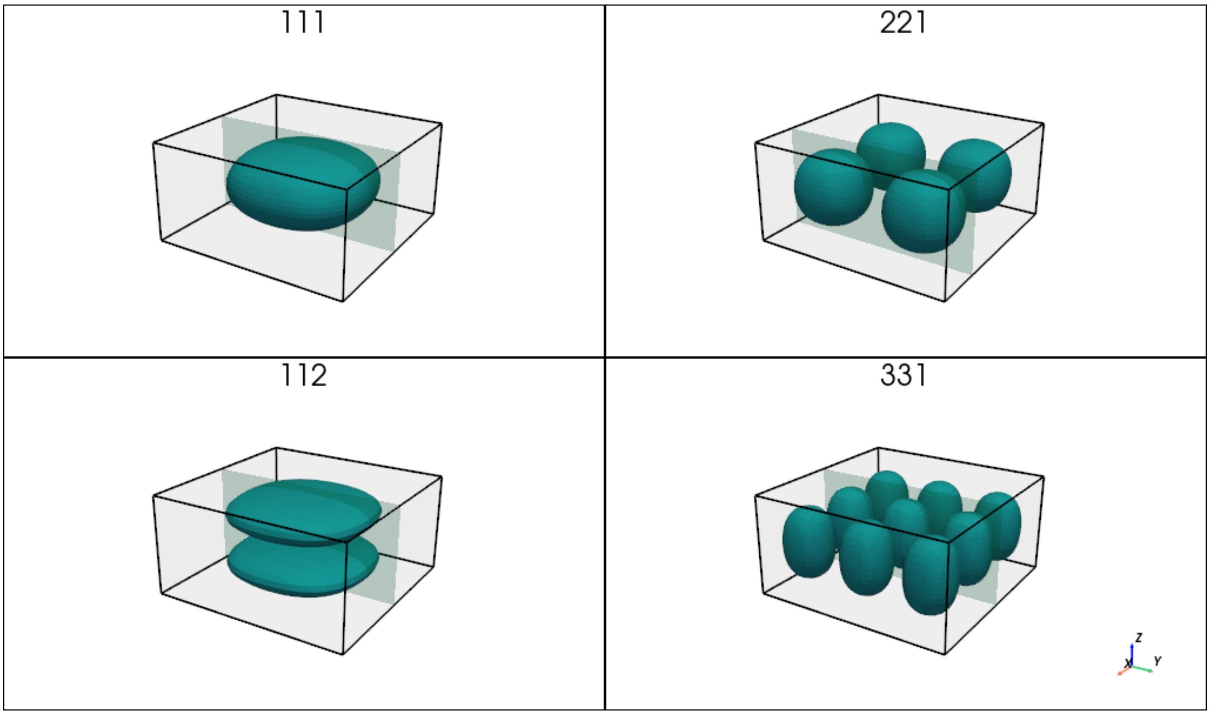


Figure 4.8.1.3: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm by 10 nm by 5 nm cuboidal shaped quantum dot with infinite barriers for the state E_{111}, E_{221}, E_{112}, E_{331}.

Last update: 19/06/2025

4.8.2 Hole energy levels of an "artificial atom" - Spherical Si Quantum Dot (6-band k.p)

Input files:

- *3DsphericSiQD_d5nm_6bandkp_nnp.in*

Scope:

In this tutorial, we calculate the energy spectrum of a spherical Si quantum dot of radius 2.5 nm.

Output Files:

- *bias_00000\Quantum\energy_spectrum_qr_6band_kp6_00000.dat*

Introduction

We assume that the barriers at the QD boundaries are infinite. The potential inside the QD is assumed to be 0 eV. We use a grid resolution of 0.25 nm. We solve the 6-band k.p Schrödinger equation for the hole eigenstates.

The following 6-band k.p parameters are used:

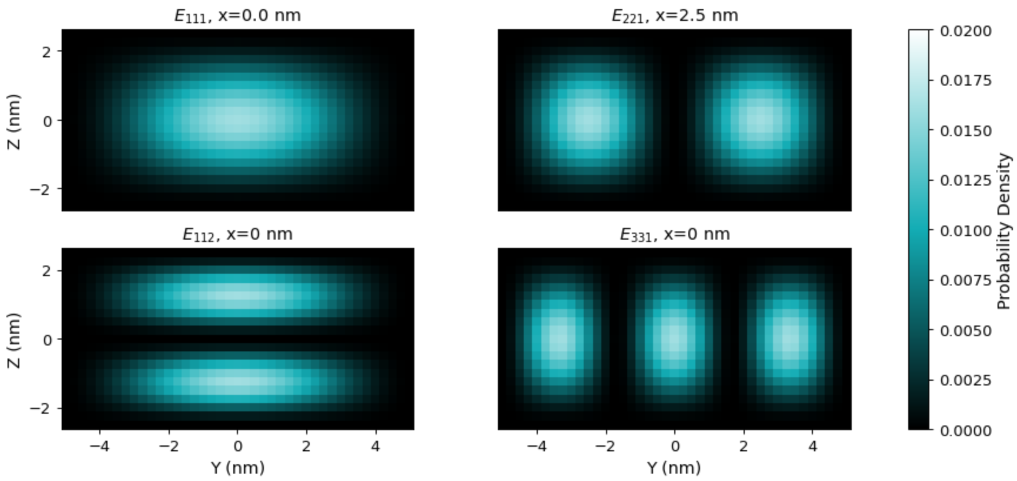


Figure 4.8.1.4: The 2D slices of probability density of the E_{111}, E_{221}, E_{112}, E_{331} states of a 10 nm by 10 nm by 5 nm cuboidal shaped quantum dot with infinite barriers.

```
 kp_6_bands{
     L = -6.8          # [Burdov] V.A. Burdov, JETP 94, 411 (2002)
     M = -4.43         # [Burdov]
     N = -8.61         # [Burdov]
}
```

These L, M, N parameters correspond to the following Luttinger parameters:

- γ_1 = 4.22
- γ_2 = 0.395
- γ_3‘ = 1.435

Results

Figure 4.8.2.1 shows the the isosurfaces of the probability density of the first six hole eigenstates of a spherical Si quantum dot.

Figure 4.8.2.2 shows the hole eigenenergy spectrum of the Si QD (diameter = 5 nm) calculated with a 6-band k.p Hamiltonian.

For comparison, we also display the energy spectrum where we assumed zero spin-orbit splitting energy. In this case there is a six-fold symmetry. Spin-orbit splitting reduces this degeneracy to 4 and 2. In general, each state is two-fold degenerate due to spin.

Note

The *nextnano*++ tool only allows a cuboidal shaped quantum region, thus we can’t employ a spherical quantum region that would reduce the dimension of the 6-band k.p Hamiltonian matrix and thus the overall execution time.

Following the paper of *[Burdov2002]*, one can calculate the ground state energy for this particular system from the L and M parameters:

$$E_1 = -\frac{\hbar^2\pi^2}{2m_h R^2} = -0.314eV$$

using m_h = 0.192 m_0 as *[Burdov2002]*, where he uses incorrect k.p parameters: In his definition L must be -5.8

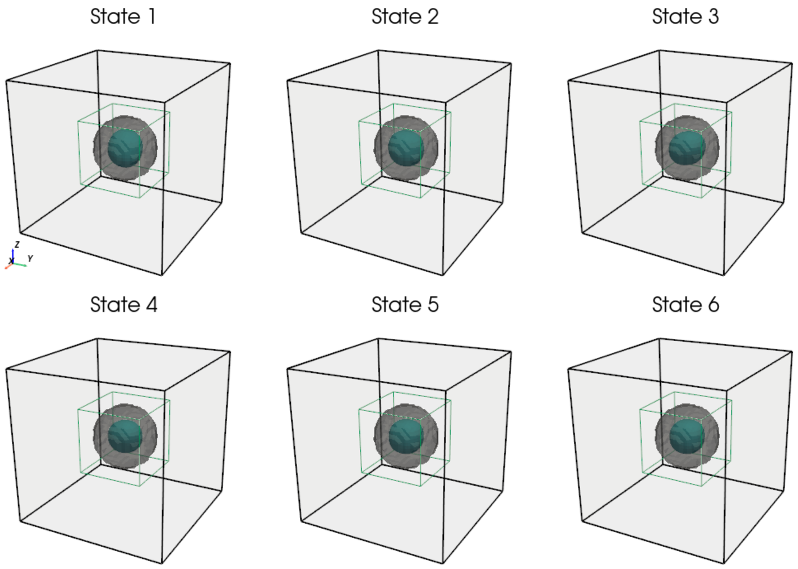


Figure 4.8.2.1: Isosurfaces of the first six hole eigenstates in a spherical Si QD calculated with a 6-band k.p Hamiltonian.

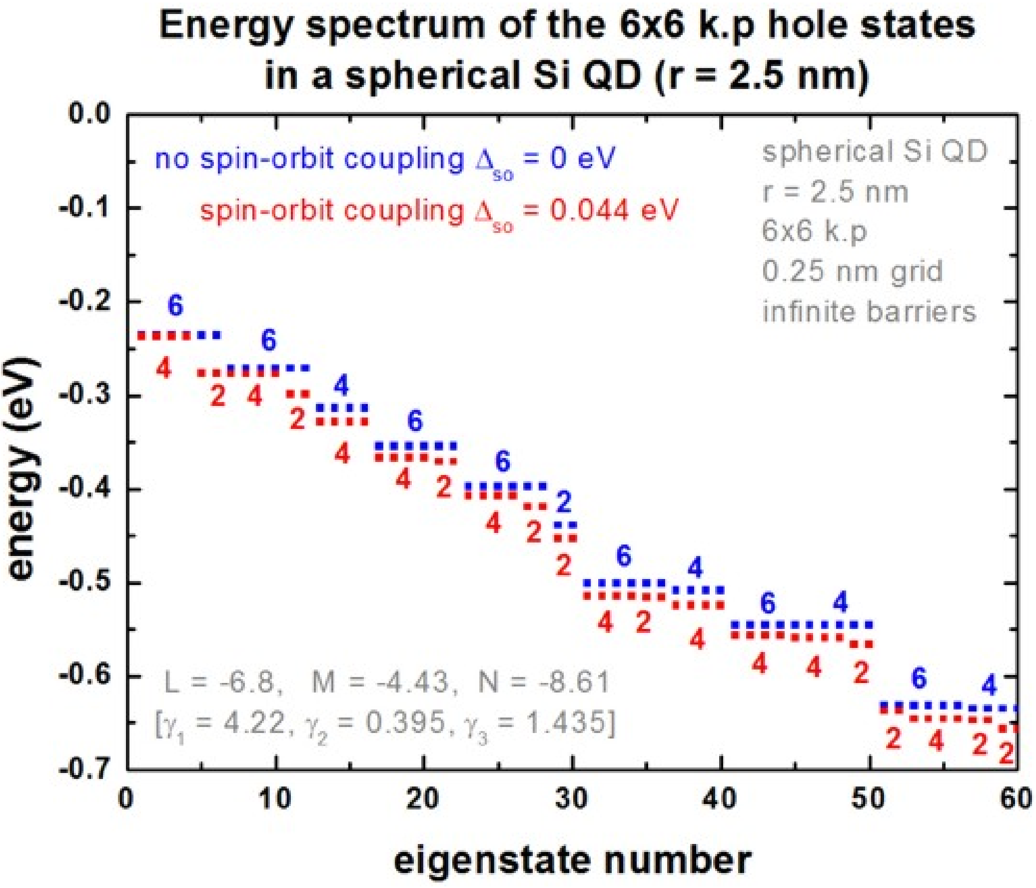


Figure 4.8.2.2: Energy spectrum of the 6-band k.p hole states in a spherical Si QD.

and M = -3.43.

$$E_1 = -\frac{\hbar^2\pi^2}{2m_h R^2} = -0.254eV$$

using m_h = 0.237 m_0 as *[BelyakovBurdov2008]*. The latter is in much better agreement to our calculations. m_h is given by:

$$m_h = \frac{3m_0}{L+2M} = \frac{3m_0}{-6.8+2\cdot(-4.43)} = -0.192m_0$$

in *[Burdov2002]* and

$$m_h = \frac{3m_0}{(L+1)+2(M+1)} = \frac{3m_0}{-5.8+2*(-3.43)} = -0.237m_0$$

in *[BelyakovBurdov2008]*. The latter definition is consistent to our implementation of the k.p Hamiltonian. The discrepancy of these equations arises because there are two different definitions of the L, M parameters available in the literature.

Comparison of nextnano^3 and nextnano++

Figure 4.8.2.3 compares the *nextnano^3* results with the *nextnano*++ results. The results of both simulators are in excellent agreement.

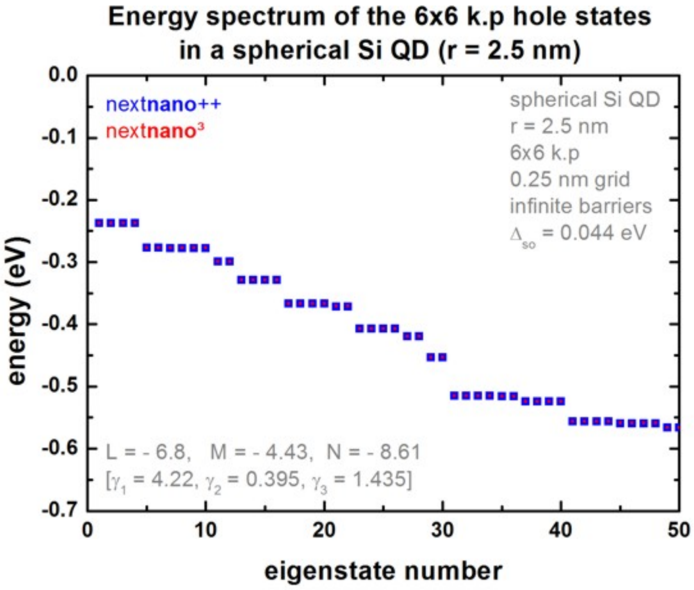


Figure 4.8.2.3: Energy spectrum of the 6-band k.p hole states in a spherical Si QD (Comparison *nextnano*++ and *nextnano^3*).

Additional comment for experts

For this particular geometry, the eigenvalues are highly degenerate, not only due to spin, but also due to geometry. This might cause problems for certain eigenvalue solvers as they might miss some of these degenerate eigenvalues. So the tool should be used with care. In our case, the 'chearn' eigenvalue solver (Arnoldi method that uses Chebyshev polynomials as preconditioner) missed some degenerate eigenvalues. So probably one has to adjust some eigenvalue solver parameters to increase the accuracy. For this reason it is of great advantage if any numerical software has redundancy in terms of several eigensolvers where one can choose from in order to check results for consistency and accuracy, as well as performance.

This tutorial also exists for *nextnano^3*.

Last update: 2025/06/25

4.8.3 Quantum Dot Molecule

In this tutorial, we study two coupled quantum dots (QDs), i.e. two “artificial atoms” that form an “artificial molecule”. The two QDs are asymmetric and differ with respect to their height (4 nm and 6 nm).

With no electric field, the groundstates of both elcectron and hole are localized at the larger QD. By applying the electric field and increasing its strength, the hole groundstate becomes bonding state and then localizes at the smaller QD. At the same time the electron groundstate is still localized at the larger QD because of the weaker coupling between the two QDs due to the heigher barrier height. We will see this leads to the change from an direct exciton to indirect exciton.

The relevant input files are as followings:

- *3DQD_molecule_cuboid_asymmetric_nn3.in / *_nnp.in*

Some of the material parameters that are used in this tutorial are based on the paper of

> M. Grundmann, D. Bimberg
> Formation of quantum dots in twofold cleaved edge overgrowth
> Phys. Rev. B 55 (7), 4054 (1997).

Simulation

This simulation has the following features:

- We keep things simple by using cuboidal shaped GaAs QDs surrounded by $Al_{0.35}Ga_{0.65}As$ barriers, i.e. we neglect strain and piezoelectric effects which is reasonable as the two materials GaAs and $Al_{0.35}Ga_{0.65}As$ have pretty similar lattice constants.
- We also neglect the wetting layers and excitonic effects.
- In order to keep the CPU time to a minimum, we do not use the k.p approximation, i.e. we use for both electrons and the heavy hole a single-band effective mass approximation for the Schrödinger equation (parabolic and isotropic effective mass tensor). Nevertheless, this is sufficient to show some basic quantum physical effects of this QD molecule.
- We use different electron and hole masses in the barrier and well material, respectively.
- The left QD has the dimensions 10 nm x 10 nm x **4 nm** (smaller dot). The right QD has the dimensions 10 nm x 10 nm x **6 nm** (larger dot).
- The two QDs are separated by a 2 nm $Al_{0.35}Ga_{0.65}As$ barrier.
- The grid resolution is 0.5 nm (rectangular tensor grid). This leads to a 3D Schrödinger matrix of dimension 50,225.
- We apply Dirichlet boundary conditions to the Schrödinger equation, i.e. the wave functions are allowed to penetrate the following distances into the barrier material (on each side): - along the x and y directions: 4 nm - along the z direction: 4.5 nm
- We vary the electric field along the growth direction (z axis) in steps of 2.5 kV/cm, i.e. from 0 kV/cm to 40 kV/cm. Note that the direction of the electric field is opposite to the z-direction.

Results

Electron and heavy hole ground states

The following figure show the square of the first electron (**el1**, top) and heavy hole (**h1**, bottom) wave functions (isosurfaces at $\psi^2 = 0.0015$) for different applied electric fields (0 kV/cm, 17.5 kV/cm, 40 kV/cm).

The following figure shows the electron (**el1**, top) and heavy hole (**h1**, bottom) probabilities from the 2D slices defined in Figure 4.8.3.1 at different electric fields.

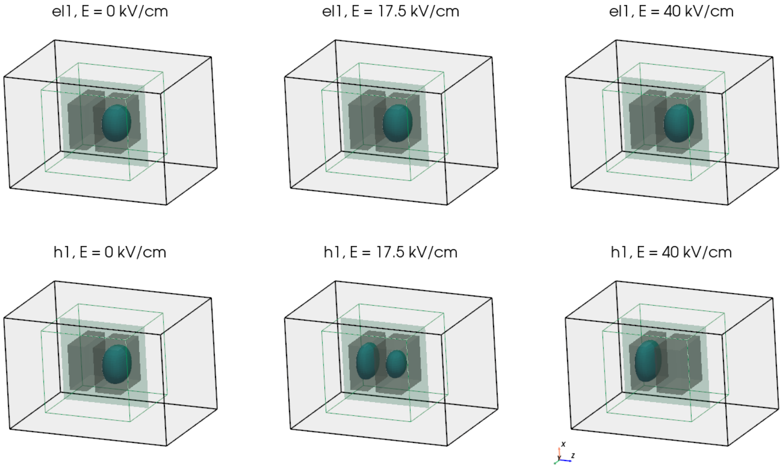


Figure 4.8.3.1: Probabilities of first electron and first heavy hole states at different electric fields. The green wireframe shows the quantum region, the grey shaded areas are the quantum dots. The shown isosurfaces are at $\psi^2 = 0.0015$. The shaded green slice is at $y = 0$, i.e. through the center of the QDs.

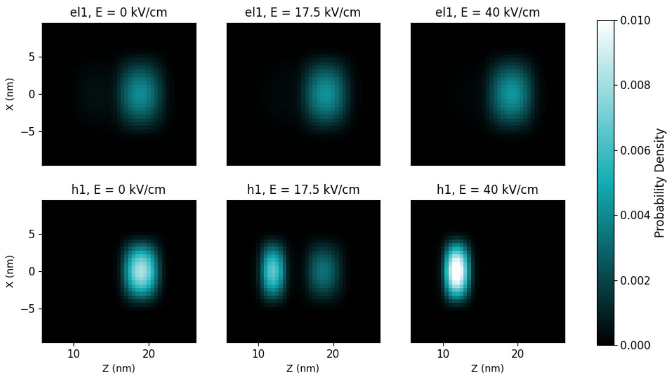


Figure 4.8.3.2: 2D slices at $y = 0$ of probabilities of first electron and first heavy hole states at different electric fields.

At zero applied electric field, both electron and heavy hole are located in the larger dot and form a **direct (bright) exciton**.

At an electric field of 17.5 kV/cm, the electron is still located in the larger dot on the right side, whereas the heavy hole located in both wells (strong coupling). The heavy hole states at the field of 17.5 kV/cm are discussed further below. The exciton that is formed is something in between a direct and an indirect exciton.

At an electric field of 40 kV/cm, the electron is still located in the larger dot on the right side, whereas the heavy hole ground state is now located in the left QD. An indirect (dark) exciton is formed. The exciton is called dark because the electron-hole overlap is much smaller and thus its oscillator strength (probability of optical transition) is much weaker (see Figure 4.8.3.6 below on spatial electron-hole overlap integrals).

Electron and heavy hole energies

Figure 4.8.3.3 shows the electron (left) energies of the ground state (**e1**) and the first excited electron state (**e2**) of the QD molecule, as well as the heavy hole (right) energies of the ground state (**h1**) and the excited hole states (**h2-h5**) of the QD molecule. The electron ground state (**e1**) is always located in the larger QD (right side) whereas the first excited electron state (**e2**) is always located in the smaller QD (left side). The third and the forth eigenstate (e3, e4) are degenerate (not shown) because our QD molecule has a symmetry with respect to the x and y coordinates. They are always located in the right QD. In contrast to the electrons, the hole coupling between the two QDs is much stronger due to the smaller barrier height. At 17.5 kV/cm anticrossing between the states occur due to the formation of bonding and antibonding states.

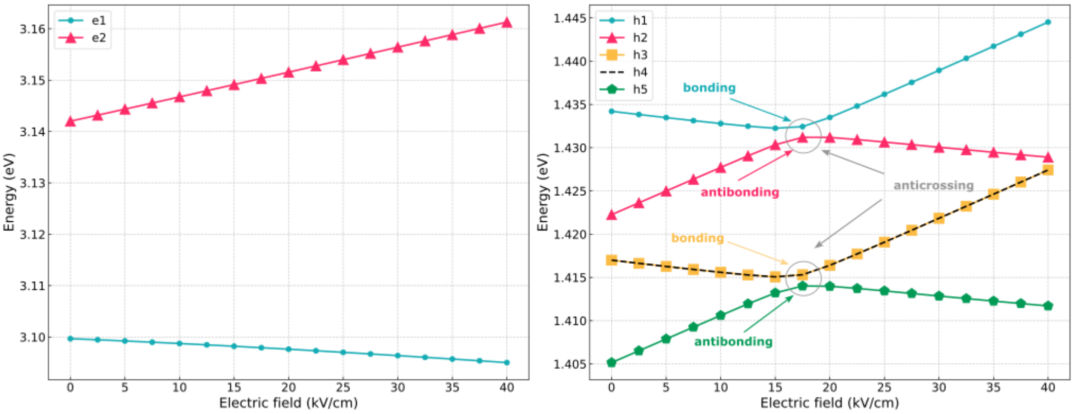


Figure 4.8.3.3: Electron (left) and heavy hole (right) energies

Bonding and antibonding heavy hole state at anticrossing point

The following figure shows the envelopes of the first six hole wave functions (isosurfaces $\psi = \pm 0.015$) at an electric field of 17.5 kV/cm.

The following figure shows hole envelopes from the 2D slices defined in Figure 4.8.3.4.

Electron-hole transition energies and overlap

To understand the strength of the optical transitions we have to evaluate the matrix elements of the envelope functions, i.e. the spatial overlap integral over the electron and hole wave functions.

$$\int \psi^*_{el,i}(x)\psi_{hl,j}dx$$

The following figure shows the five lowest electron-hole transition energies of the QD molecule as a function of electric field (left) and the overlap between the first electron (**el1**) and the first and second heavy hole (**h1** and **h2**) wave functions (right).

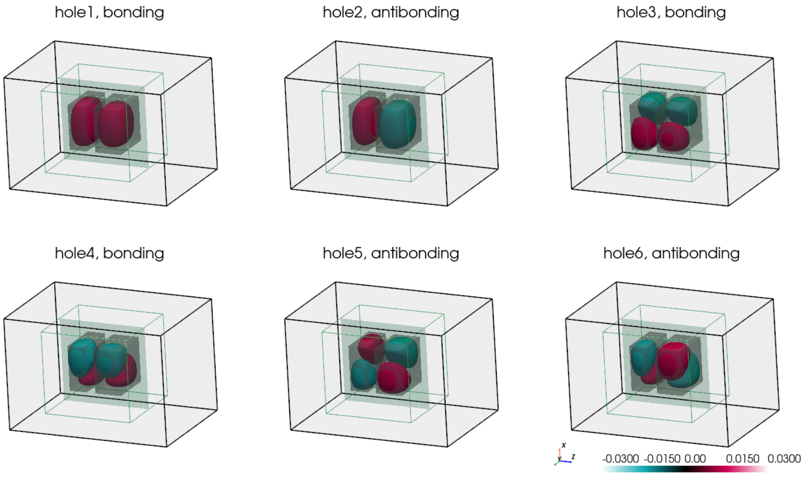


Figure 4.8.3.4: Envelope of the first 6 hole states at electric field of 17.5 kV/cm (some are bonding and some are antibonding states). The green wireframe shows the quantum region, the grey shaded areas are the quantum dots. The shown isosurfaces are at $\psi = \pm 0.015$. The shaded green slice is at $y = 0$, i.e. through the center of the QDs.

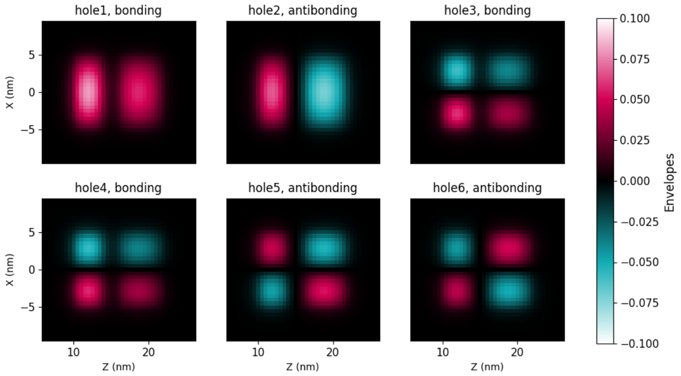


Figure 4.8.3.5: 2D slices at $y = 0$ of envelopes of the first 6 hole states at electric field of 17.5 kV/cm (some are bonding and some are antibonding states).

For fields smaller than 17.5 kV/cm a direct (bright) exciton is the ground state (both electron and hole wave function are located in the larger QD (right side), whereas for fields larger than 17.5 kV/cm an indirect (dark) exciton is the ground state where the electron is located in the larger QD (right side) and the hole is located in the smaller QD (left side). Therefore, the nature of the QD molecule ground state changes from **direct** to **indirect**.

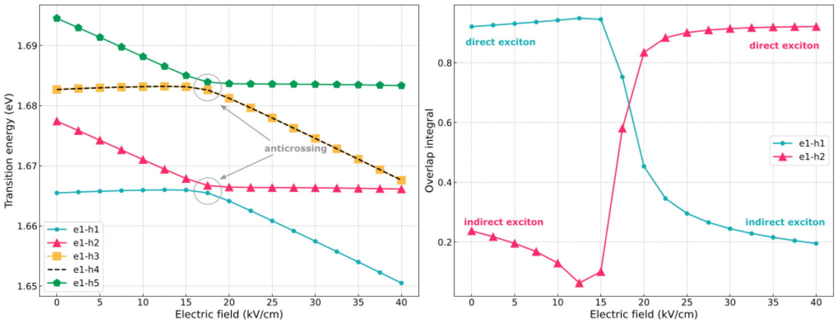


Figure 4.8.3.6: Electron-hole transition energies (left) and overlap (right) as a function of electric field.

This tutorial also exists for *nextnano*^{3}.

Last update: 2025/06/25

4.8.4 Energy levels in a pyramidal shaped InAs/GaAs quantum dot including strain and piezoelectric fields

Input files:

- *3DInAsGaAsQDPyramid_PryorPRB1998_10nm_nnp.in*

Scope:

In this tutorial we calculate the energy levels in a pyramidal shaped quantum dot. This tutorial is based on *[Pryor1998]*. We use identical material parameters with respect to this paper in order to make it possible to reproduce Pryor's results. We note that meanwhile more realistic material parameters are available and that for the simulation of realistic quantum dots the inclusion of the wetting layer and an appropriate nonlinear $InGaAs$ alloy profile is recommended.

Output files:

- *bias_00000\bandedges_1d_x.dat*
- *bias_00000\bandedges_1d_z.dat*
- *bias_00000\Quantum\probability_shift_dot_Gamma_0001.avs.fld*
- *bias_00000\Qunatum\energy_spectrum_dot_Gamma_00000.dat*
- *bias_00000\Qunatum\energy_spectrum_dot_HH_00000.dat*

Introduction

We make the following simplifications in order to be consistent with *[Pryor1998]*:

- The wetting layer is omitted for simplicity.
- The QD material is purely $InAs$.
- The barrier material is purely $GaAs$.
- The dielectric constant in the barrier material ($GaAs$) is the one for $InAs$.
- Periodic boundary conditions are assumed in all three directions for the strain equation.
- The QD shape is a pyramid with a square base (base length = 10 nm) and a height of 5 nm.
- The four side walls of the pyramid are oriented in the (011), (0-11), (101) and (-101) planes, respectively.

- The whole simulation area has the dimensions 44 nm x 44 nm x 40 nm.

Figure 4.8.4.1 shows the structure of the simulation region with the pyramidal shaped quantum dot in the center.

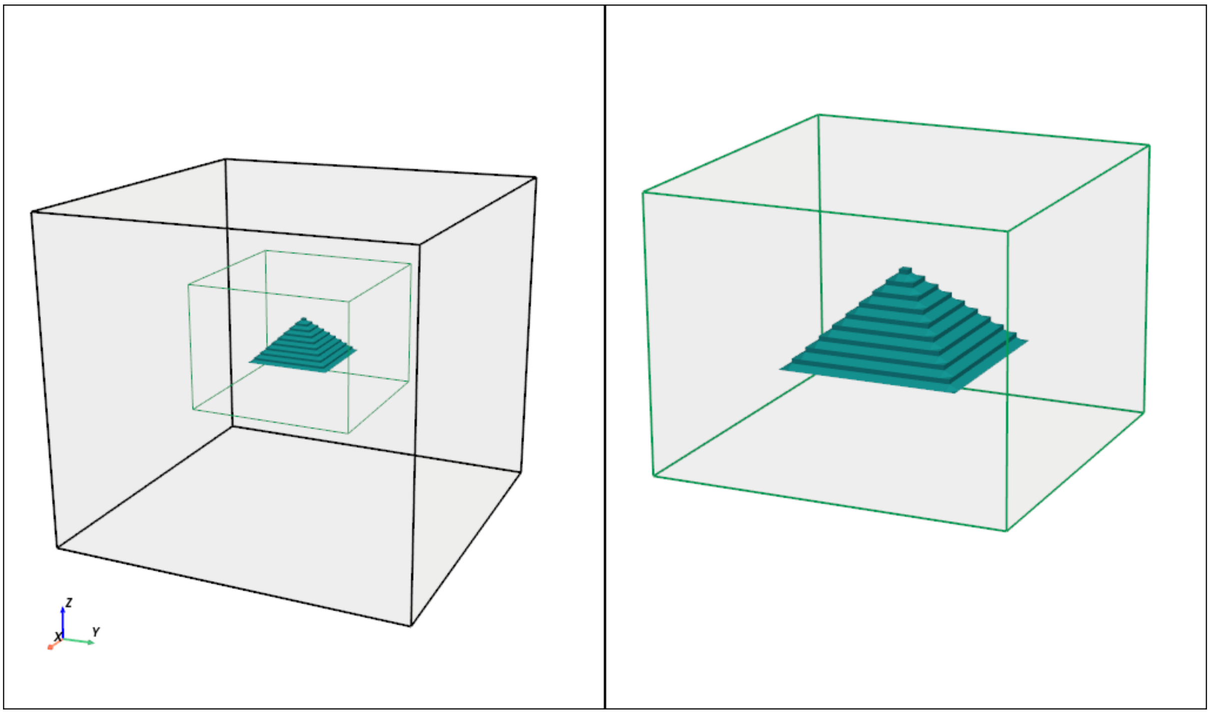


Figure 4.8.4.1: The material structure of the simulation region with the zoom of the quantum region on the right side.

Conduction and valence band profiles

The following figures show the conduction and valence band edges (heavy hole, light hole and split-off hole) for a 10 nm pyramidal shaped QD along two different line scans. Figure 4.8.4.2 shows the band profile along the z axis through the center of the QD (x = y = 0 nm), and Figure 4.8.4.3 shows the band profile along the x axis through the base of the QD (y = z = 0 nm).

The energies of the bands have been obtained by diagonalizing the 8-band k.p Hamiltonian at $k = 0$ (including the Bir-Pikus strain Hamiltonian) for each grid point, taking into account the local strain tensor and deformation potentials. Note that piezoelectric effects are not included yet in this band profile.

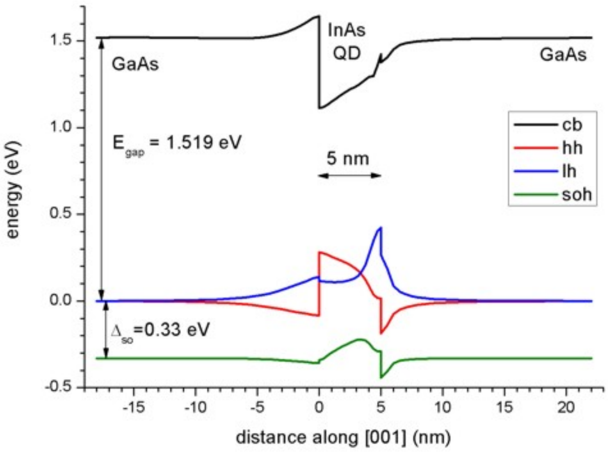


Figure 4.8.4.2: Calculated band edge profile along z axis.

The figures compare well with Figs. 2(a) and 2(b) of *[Pryor1998]*. However, there are some differences: Due to

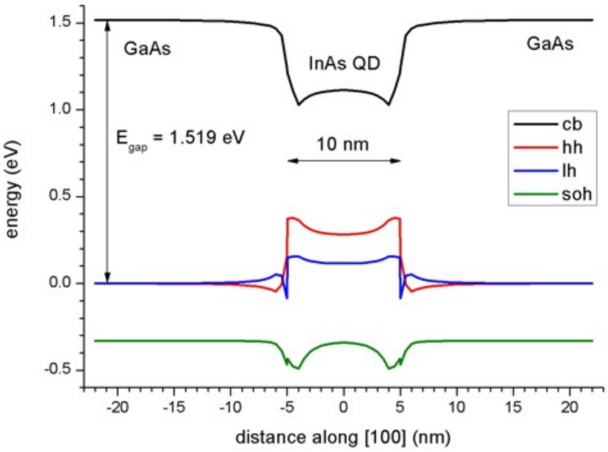


Figure 4.8.4.3: Calculated band edge profile along x axis.

valence band mixing of the states in the k.p Hamiltonian, we do not have pure heavy and light hole eigenstates anymore. Thus there is some arbitrariness to assign the labels "heavy" and "light" to the relevant eigenstates h1 and h2. Obviously, when solving the full 6-band or 8-band k.p Hamiltonian, this labelling becomes irrelevant because all three hole band edges enter the Hamiltonian simultaneously (in contrast to a single-band effective mass approach where only individual "heavy" hole or "light" hole band edges would be considered).

Electron wave functions (single-band effective-mass approximation)

Figure 4.8.4.4 shows the envelopes of the electron wave function Ψ^2 of the first 9 electron eigenstates inside of the quantum dot.

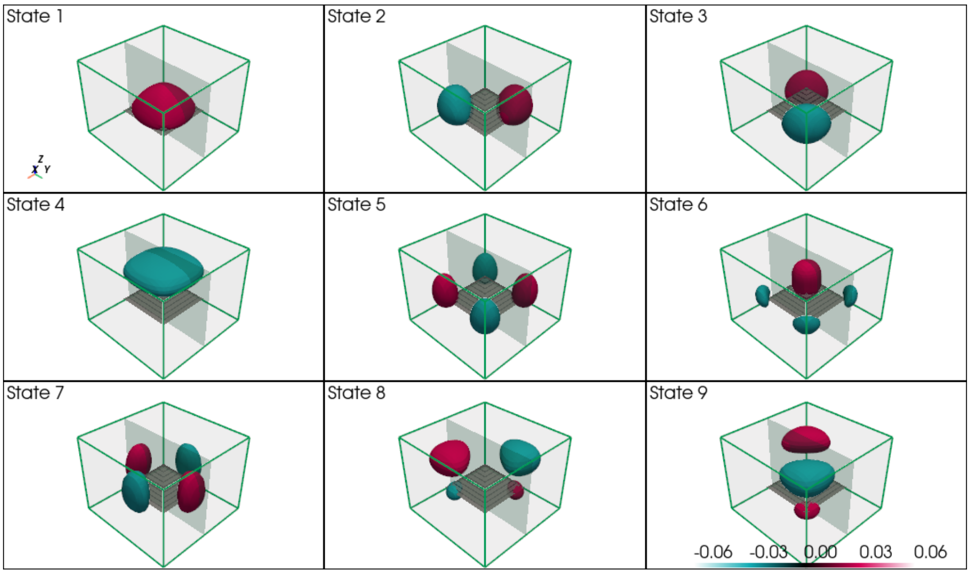


Figure 4.8.4.4: Envelope function of the first 9 electron states of the quantum dot. The isosurfaces shown are at $\Psi = \pm 0.03$. The green wireframe shows the quantum region, the grey shaded area is the quantum dot. THe shaded green slice is at $x = 0$, i.e. through the center of the QD.

Figure 4.8.4.5 shows the 2D slices of electron wave function from the slices defined in Figure 4.8.4.4.

Note

The following sections are preliminary and yet to be updated.

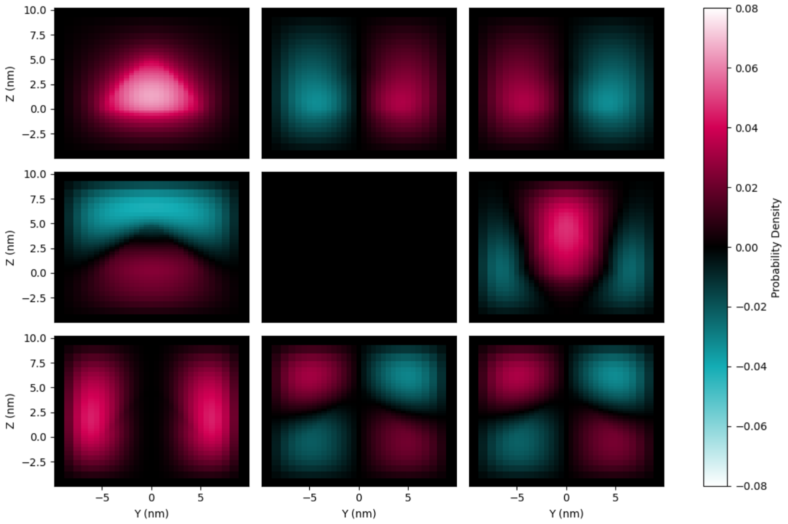


Figure 4.8.4.5: 2D slices of the envelope function of the first 9 electron states of the quantum dot at $x = 0$

10 nm quantum dot

(Note: Pryor's Fig. 7 shows the energies for a 14 nm quantum dot). The band gap is 1.519 eV.

Electron energies

```
(i)   effective mass (me = 0.023 m0) => 0.7000983 eV (only one confined electron␣
↪state)
(ii)  effective mass (me = 0.04   m0) =>  eV
(iii) effective mass (me(r) = ...   m0) => not implemented in nextnano³
(iv)  8-band k.p                     =>  eV
```

Hole energies

```
( )   effective mass (mhh = 0.41 m0) => hh1 = -0.585198481 eV
                                     => hh1 = -0.61776 eV
                                     => hh1 = -0.62275 eV
(i)   6-band k.p                     => 1.0081402 eV (?) (bad eigenvalues using 6-band k.p␣
↪with finite-differences)
(ii)  8-band k.p                     =>  eV
```

Transition energy electron - hole

```
- (i) - ( ): exciton correction 2.9 meV (Pryor: 27 meV)
E_ex [eV]   E_el - E_hl   E_el0 - E_hl0   Delta_Ex     REAL(inter_matV(1))
1.28238     1.27958       1.28530         0.00291947   0.428169
```

14 nm quantum dot (Pryor's Fig. 7)

```
(i)   effective mass (me = 0.023 m0) => 0.6458949 eV (only one confined electron␣
↪state) + (1.519 - 0.752916) eV = 1.412 eV (in substrate layer below QD)
(i)   effective mass (me = 0.023 m0) => 0.6458949 eV (only one confined electron␣
↪state) + (1.519 - 0.765522) eV = 1.399 eV (in substrate layer at corner)
(i)   effective mass (me = 0.04   m0) => 0.6248762 eV (only one confined electron␣
↪state) + (1.519 - 0.765522) eV = 1.378 eV (in substrate layer at corner

14 nm, 6x6k.p, box, nonsym:
-0.56607270
```

(continues on next page)

(continued from previous page)

```
-0.58734305
-0.59621434
-0.60757551
-0.62802221
-0.63650764
```

Last update: nnnn/nn/nn

4.8.5 Hexagonal shaped GaN quantum dot embedded in AlN (wurtzite)

Attention

They differ from results obtained with *nextnano++* as these input file has been improved, including change of simulation domain and boundary conditions, to represent results from the cited publication more accurately. However, qualitative tendencies are preserved.

Header

Note

The tutorial is based on *[Andreev2000]*.

Input files in *examples\quantum_dots*:

- *QD_GaN_Andreev_PRB_2000_1D_nnp_band-offsets.in*
- *QD_GaN_Andreev_PRB_2000_1D_nnp_strain.in*
- *QD_GaN_Andreev_PRB_2000_1D_nnp_strain-PzPr-poisson-1b.in*
- *QD_GaN_Andreev_PRB_2000_1D_nnp_strain-PzPr-poisson-6kp.in*
- *QD_GaN_Andreev_PRB_2000_3D_nnp.in*

Scope:

The influence of strain and pyro-/ piezoelectric fields on the electronic structure of hexagonal shaped GaN/AlN quantum dots.

Conduction and valence band alignment in AlN/GaN QWs (unstrained)

In this section the input file *QD_GaN_Andreev_PRB_2000_1D_nnp_band-offsets.in* is used to compute band offsets.

Figure 4.8.5.1 shows the conduction and valence band edge alignment in AlN/GaN structures (unstrained). In AlN, the light hole (LH) is the highest valence band whereas in GaN, this is the heavy hole (HH). We assumed a valence band offset of VBO = 0.5 eV, the conduction band offset is much larger (CBO = 2.3 eV). All material parameters are based on *[Andreev2000]* although meanwhile better parameters are available.

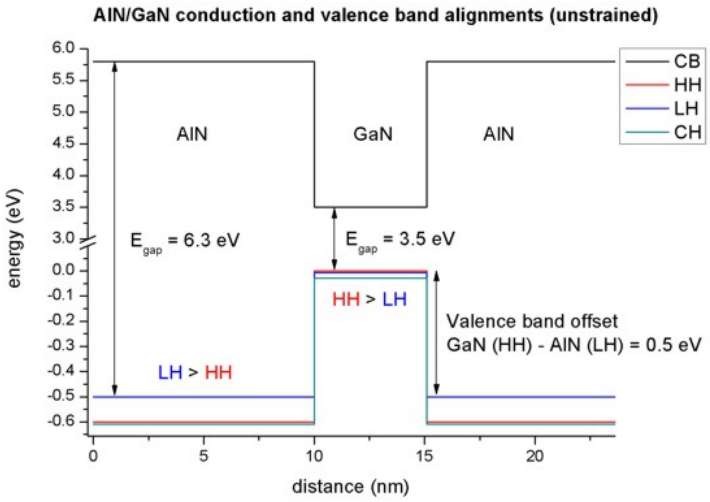


Figure 4.8.5.1: Conduction band edge (CB) and valance band edges (HH, LH, CH) of the 1D AlN/ GaN QD (unstrained).

Conduction and valence band alignment in AlN/GaN QWs (pseudomorphically strained)

In this section the input file *QD_GaN_Andreev_PRB_2000_1D_nnp_strain.in* is used to show impact of the strain on the band edges without piezo effects.

Figure 4.8.5.2 shows the conduction and valence band edge alignment in the AlN/GaN structure, which is strained with respect to the AlN substrate. The lattice constants in GaN are larger than in AlN, thus GaN is compressively strained. The AlN band edges are the same as in Figure 4.8.5.1, only the GaN edges have changed:

- The band gap of GaN has increased (compressive strain increases the band gap).
- Now the crystal-field split-hole (CH) in GaN lies above the light hole (LH) and close to the heavy hole (HH).
- The valence band offset has decreased to VBO = 0.46 eV.
- The conduction band offset has decreased to CBO = 2.15 eV.

Conduction and valence band edges in AlN/GaN QWs (pseudomorphically strained, including piezo- and pyroelectric fields)

In this section the input file *QD_GaN_Andreev_PRB_2000_1D_nnp_strain-PzPr-poisson-1b.in* or *QD_GaN_Andreev_PRB_2000_1D_nnp_strain-PzPr-poisson-6kp.in* can be used to observe piezo effect on the design.

In Figure 4.8.5.3 the effect of piezo- and pyroelectric fields on the band edge is shown. The band edge gets tilted due to the additional electric potential arising from the piezo- and pyroelectric charges. The electrostatic potential which is the solution to the Poisson equation is also shown in Figure 4.8.5.3.

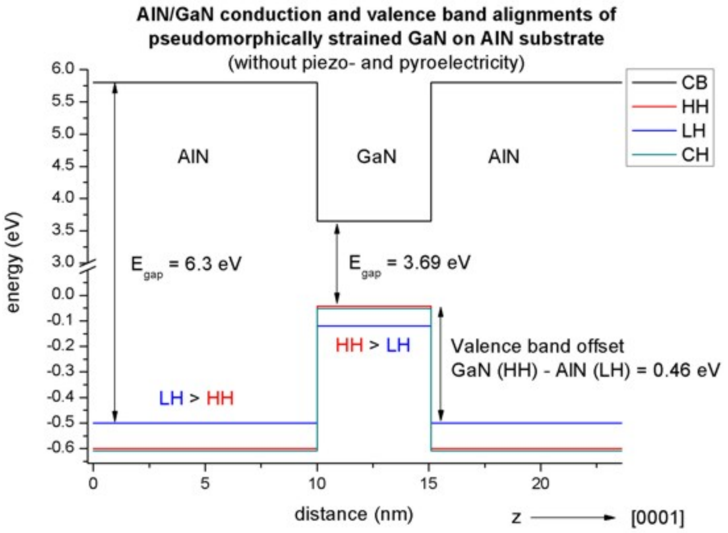


Figure 4.8.5.2: Conduction band edge (CB) and valance band edges (HH, LH, CH) of the 1D AlN/GaN QD (strained on AlN substrate).

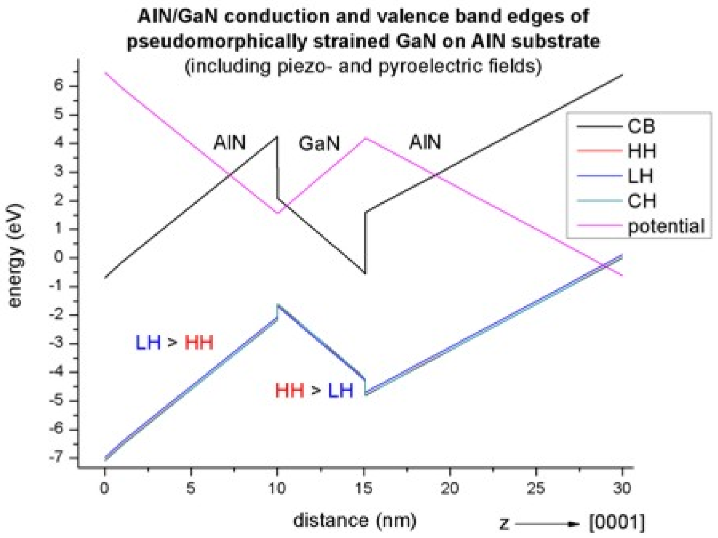


Figure 4.8.5.3: Conduction band edge (CB) and valance band edges (HH, LH, CH) of the 1D AlN/GaN QD (strained on AlN substrate) including piezo- and pyroelectric fields.

Electron and hole wave functions in AlN/GaN QWs

Figure 4.8.5.4 shows the electron and hole wavefunctions (Ψ^2) in a 5.1 nm AlN/GaN/AlN quantum well. For the electrons, the single-band effective-mass approximation was used whereas for the holes the 6-band k.p model was used. The figure shows the four lowest electron eigenstates and the 6 highest valence band eigenstates. All eigenstates are two-fold degenerate due to spin.

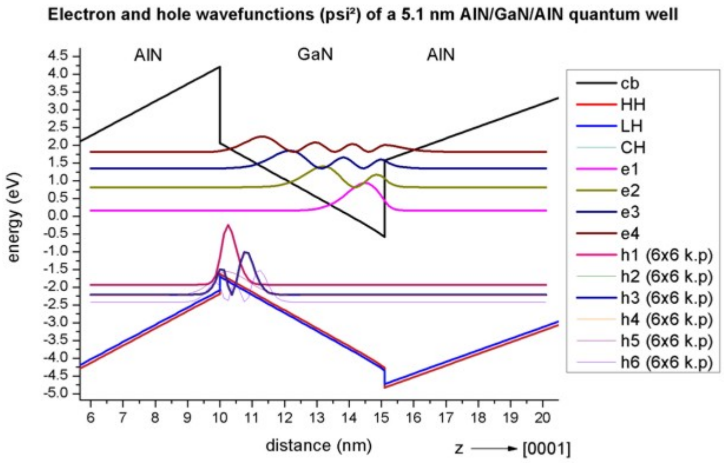


Figure 4.8.5.4: Electron and hole wavefunctions Ψ^2 of a 5.1 mm AlN/GaN/AlN quantum well.

Hexagonal shaped GaN quantum dot embedded in AlN (wurtzite)

The simulated hexagonal GaN quantum dot (height = 4.1 nm) is embedded in an AlN matrix, input file *QD_GaN_Andreev_PRB_2000_3D_nnp.in*. The wetting layer is 1 nm thick and consists of GaN. The structure and a cross-section of the structure along x-y are shown in Figure 4.8.5.5 and Figure 4.8.5.6, respectively.

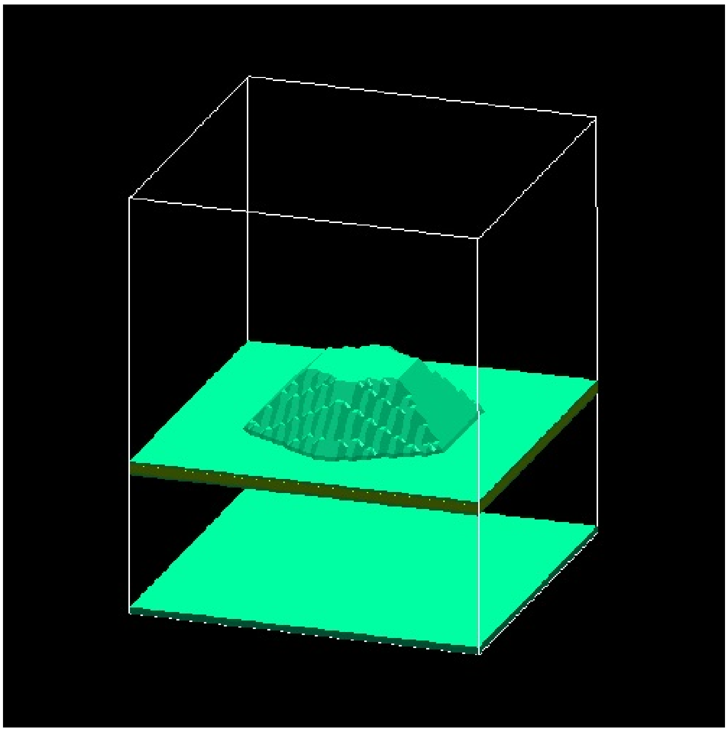


Figure 4.8.5.5: 3D AlN/GaN quantum dot.

The strain tensor components of a line through the center of the quantum dot along the z axis are shown in Figure 4.8.5.7. This figure is related to Fig. 2 (a) in *[Andreev2000]*.

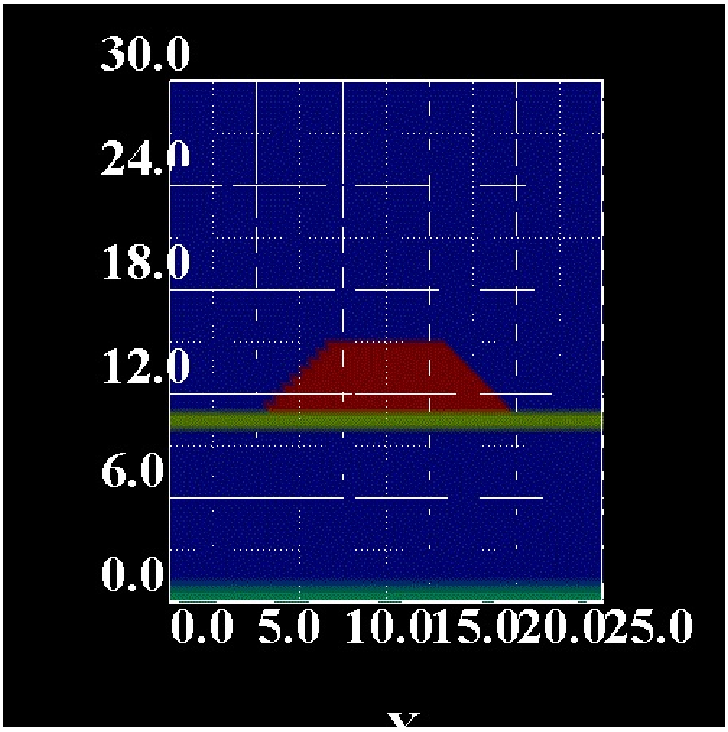


Figure 4.8.5.6: Cross-section of the hexagonal shaped AlN/GaN quantum dot.

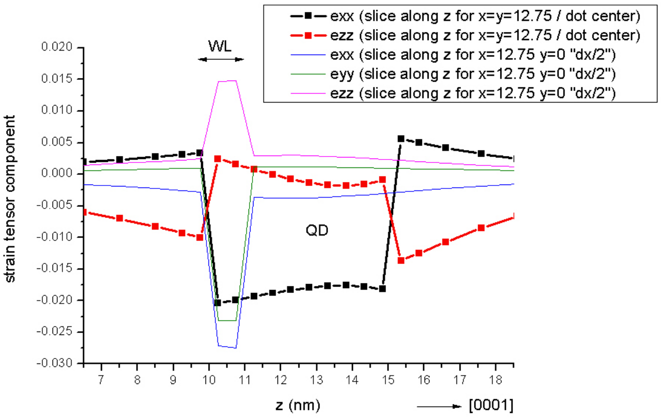


Figure 4.8.5.7: Strain tensor along the z-axis through the points (x, y) = (12.75, 12.75) nm and (x, y) = (12.75, 0) nm.

Figure 4.8.5.8 and Figure 4.8.5.9 show the strain tensor components along the [10-10] direction (y direction) for a line through the bottom of the quantum dot and for a line through the wetting layer, respectively. These figures are related to Fig. 2 (b) in *[Andreev2000]*.

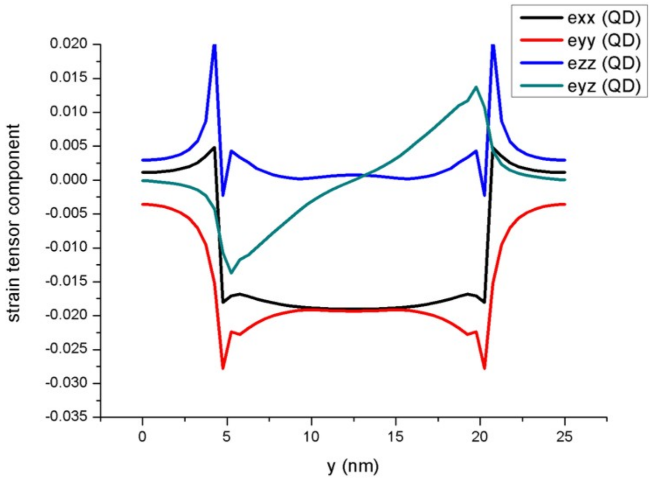


Figure 4.8.5.8: Strain tensor along the y-axis through the quantum dot (QD).

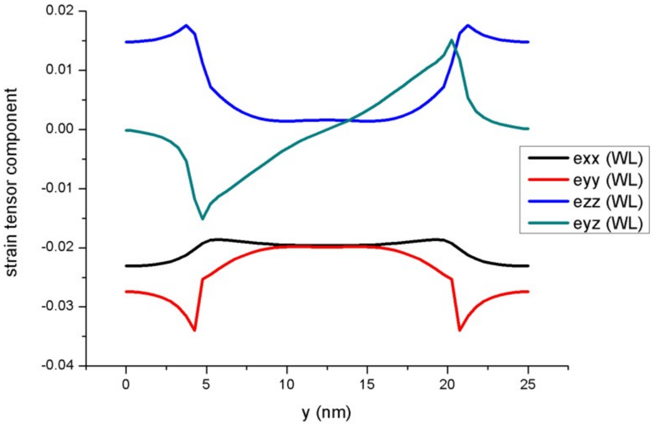


Figure 4.8.5.9: Strain tensor along the y-axis through the wetting layer (WL).

The strain induced piezoelectric fields and the pyroelectric fields lead to the electrostatic potential which is shown in Figure 4.8.5.10 and Figure 4.8.5.11. The figures of the potential are related to Fig. 4 in *[Andreev2000]*. In Figure 4.8.5.10 one can clearly see that the electrostatic potential has its maximum at the top of the QD and its minimum in the wetting layer area just below the QD. Figure 4.8.5.11 shows a cut of the electrostatic potential through the wetting layer plane.

The conduction and valence band edges are shown in Figure 4.8.5.12 and Figure 4.8.5.13. One can clearly see that the conduction band minimum is located in the top of the quantum dot whereas the maximum for the valence band is located inside the wetting layer (WL) (which is equivalent to the bottom of the quantum dot). Thus, one expects the electrons, which are located in the top area of the QD, to be spatially separated from the holes, which are located in the WL (bottom of the QD). The energy scale is in units of [eV]. The figures of the conduction and valence band edges are related to Figs. 5 and 6 in *[Andreev2000]*.

The electron states are located near the top of the quantum dot where the conduction band has a minimum. Figure 4.8.5.14 shows the electron ground state.

The following figures show the six lowest electron states of the quantum dot. The 2^{nd} and 3^{rd} eigenstates are

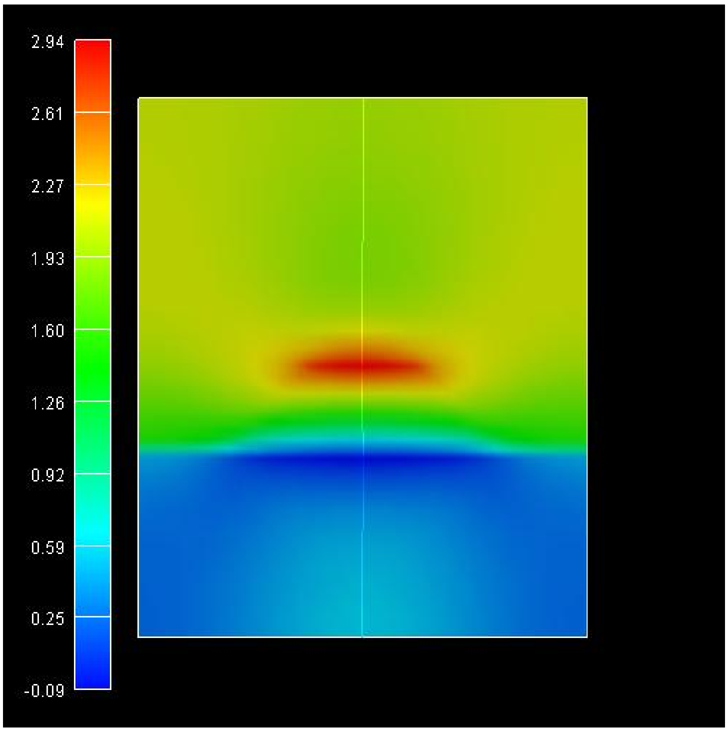


Figure 4.8.5.10: Electrostatic potential inside the quantum dot.

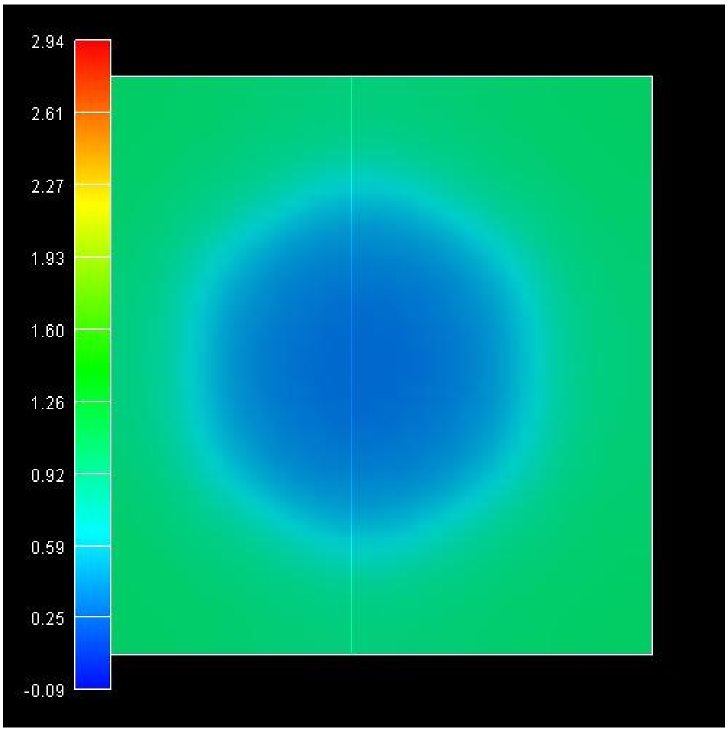


Figure 4.8.5.11: Electrostatic potential inside the wetting layer.

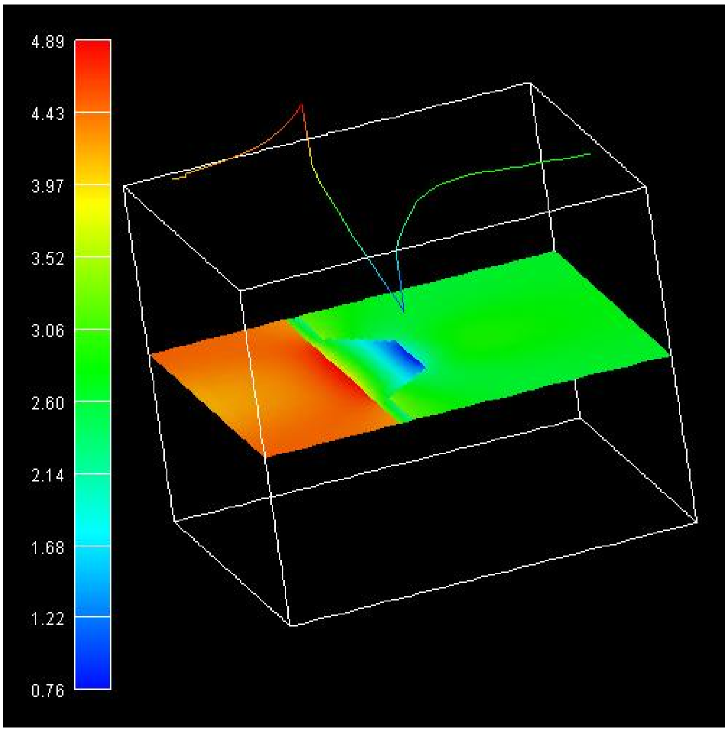


Figure 4.8.5.12: Conduction band edge of the QD.

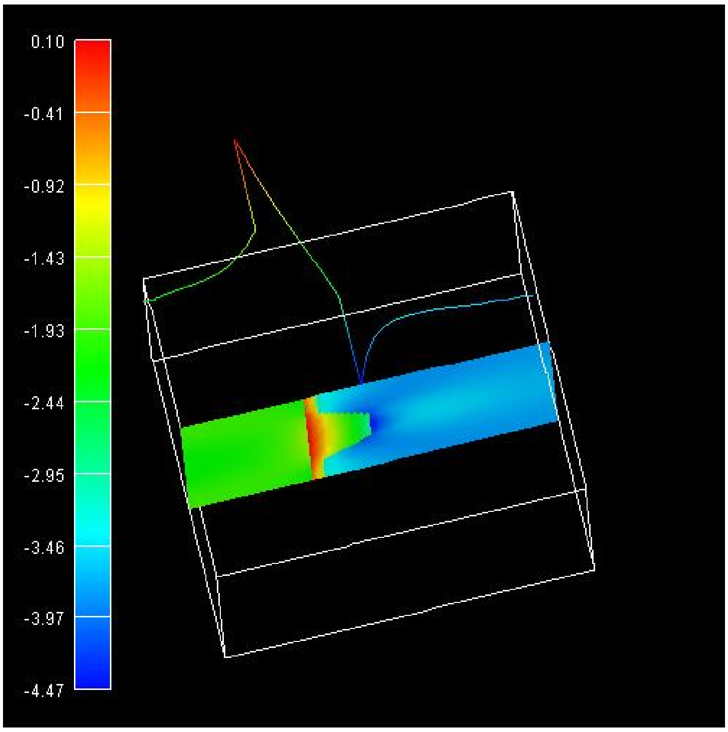


Figure 4.8.5.13: Valence band edge of the QD.

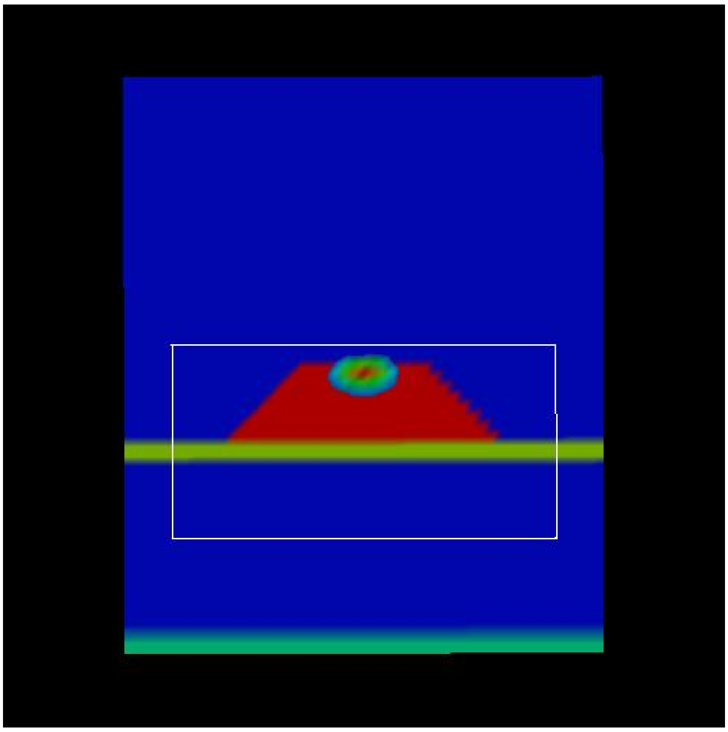


Figure 4.8.5.14: Electron ground state of the QD.

degenerate, as well as the 4^{th}, 5^{th} and 6^{th}. The figures of the wave functions (Ψ^2) are related to Fig. 7 in *[Andreev2000]*.

This tutorial also exists for *nextnano³*.

Last update: 11/07/2024

4.8.6 — DEV — Energy levels of an "artificial atom" - Spherical and ellipsoidal CdSe Quantum Dot

Attention

This tutorial is under construction

Input files:

- *QDArtificialAtom_CdSe_3D_spherical_nnp.in*
- *QDArtificialAtom_CdSe_3D_ellipsoidal_nnp.in*
- *ParabolicQW_GaAs_2D_nnp.in*

Scope:

- In this tutorial we calculate the eigenenergies of a spherical and ellipsoidal CdSe quantum dot ("artificial atom"). The tutorial is based on *[Ferreira2006]*.

Output files:

- *bias00000\Quantum\energy_spectrum_quantum_region_Gamma_00000.dat*
- *bias00000\Quantum\overlap_integrals_quantum_region_HH_Gamma.txt*

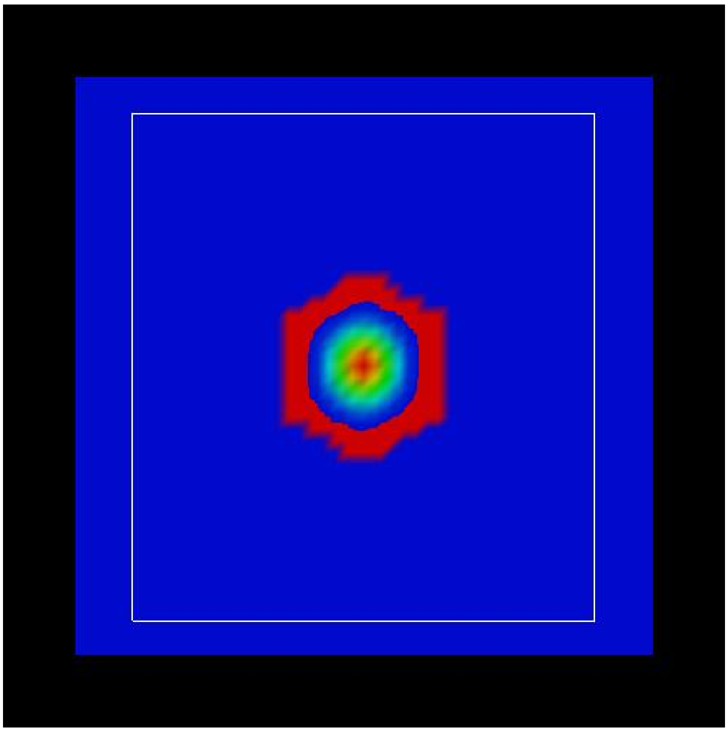


Figure 4.8.5.15: Probability density Ψ^2 of the 1st electron state in the QD.

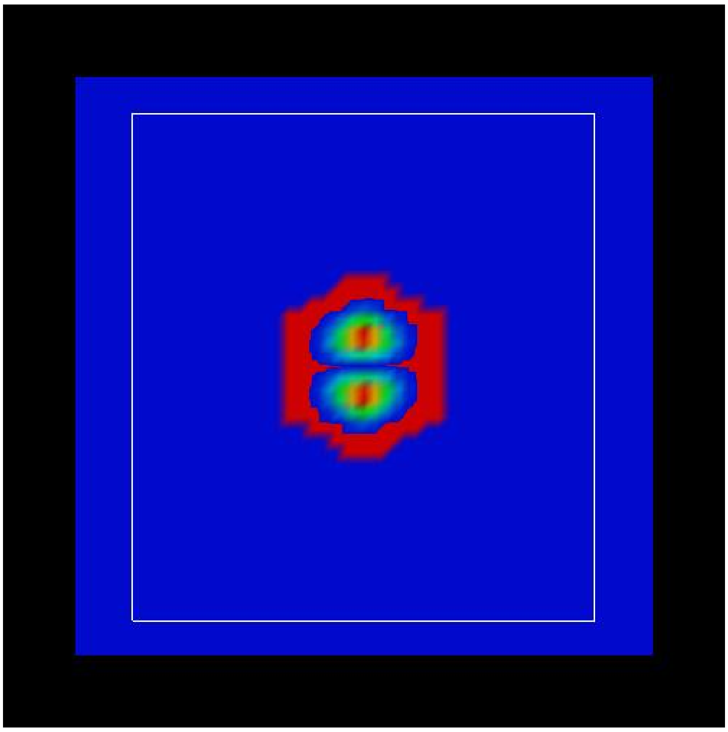


Figure 4.8.5.16: Probability density Ψ^2 of the 2nd electron state in the QD.

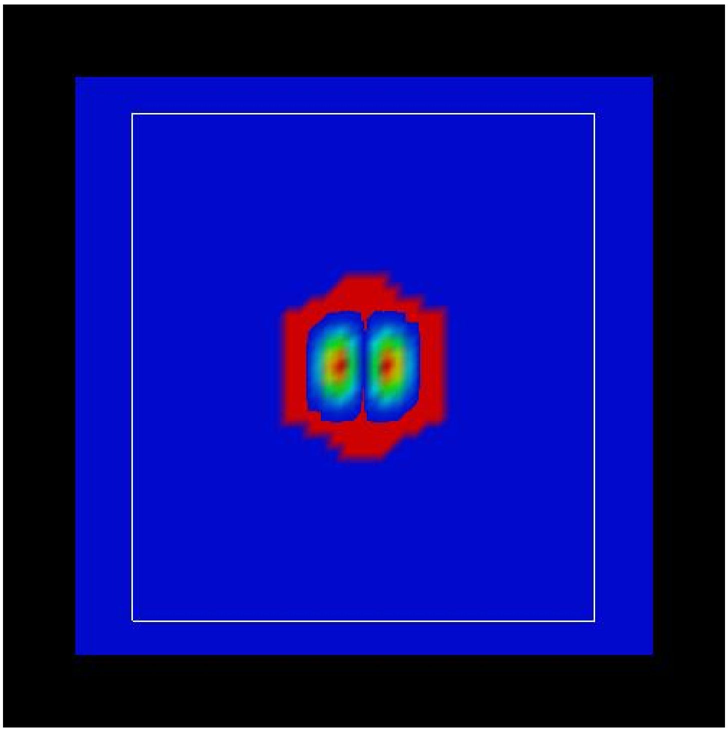


Figure 4.8.5.17: Probability density Ψ^2 of the 3^{rd} electron state in the QD.

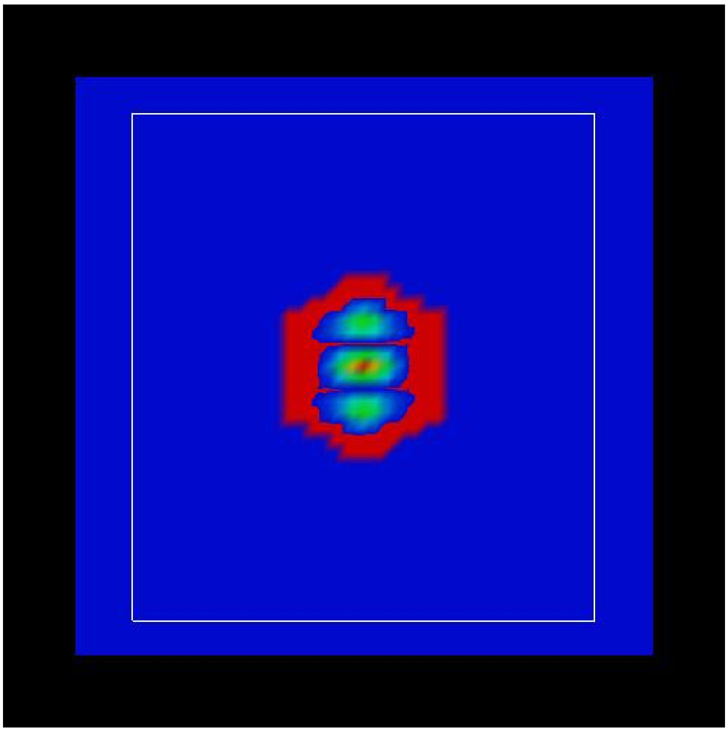


Figure 4.8.5.18: Probability density Ψ^2 of the 4^{th} electron state in the QD.

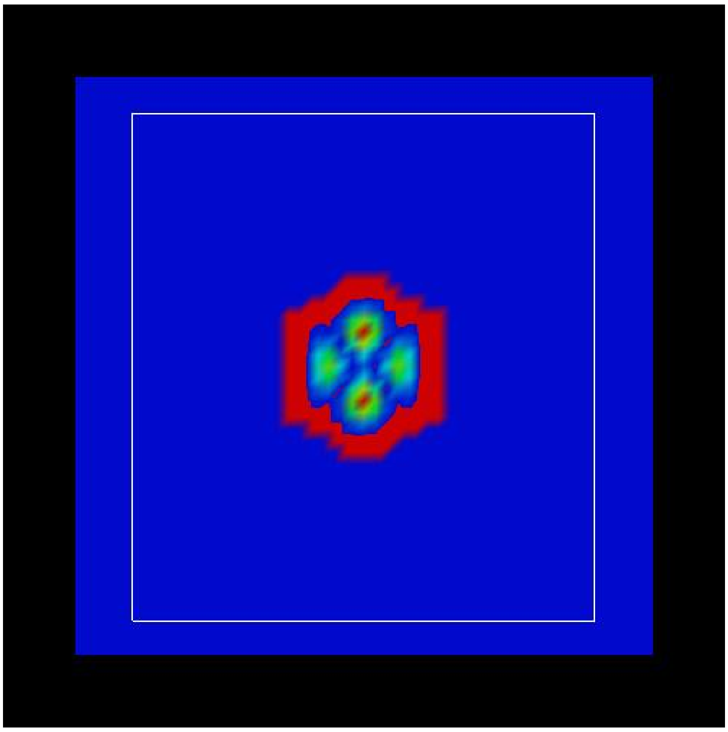


Figure 4.8.5.19: Probability density Ψ^2 of the 5^{th} electron state in the QD.

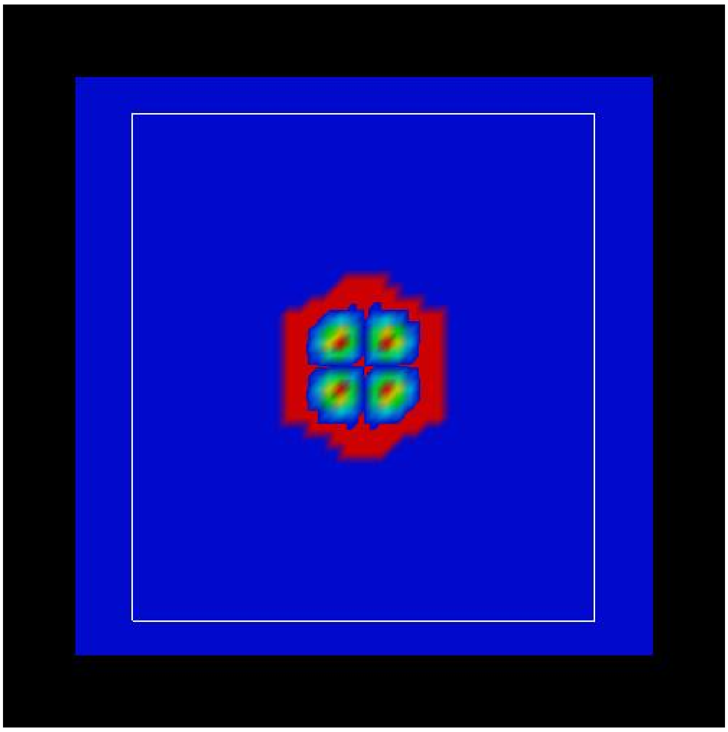


Figure 4.8.5.20: Probability density Ψ^2 of the 6^{th} electron state in the QD.

Energy levels of an "artificial atom" - Spherical CdSe Quantum Dot

Here, we want to calculate the energy levels and the wave functions of a spherical CdSe quantum dot of radius $r = 5\ \mathrm{nm}$ shown in Figure 4.8.6.1.

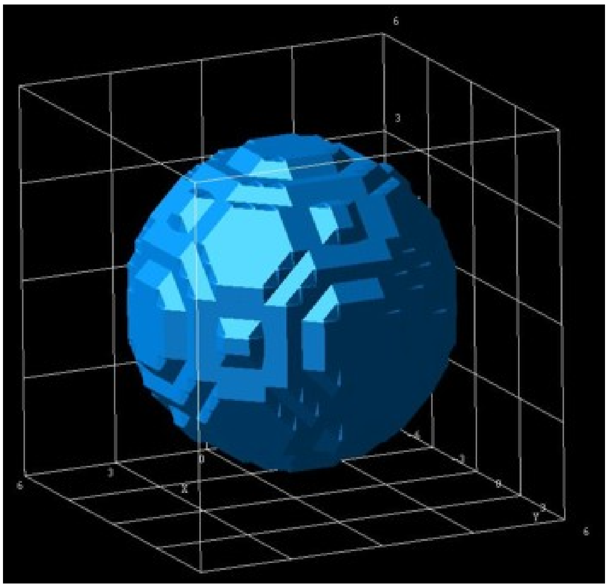


Figure 4.8.6.1: Spherical quantum dot.

We assume that the barriers at the QD boundaries are infinite. The potential inside the QD is assumed to be 0 eV. We use a grid resolution of 0.5 nm. We solve the single-band Schrödinger equation within the effective-mass approximation. The electron effective mass of CdSe is assumed to be $m_e = 0.112 m_0$.

A spherically symmetric potential leads to an energy spectrum where some eigenvalues are degenerate. We want to study the "shell structure" (degeneracy scheme) of a CdSe quantum dot of radius 5 nm. Figure 4.8.6.2 shows the calculated energy spectrum for the lowest 20 electron eigenvalues. One can clearly identify the shell structure 1s, 2p, 3d, 2s, 4f and 3p which is similar to the shell structure of the periodic table. This is the reason why quantum dots are often called "artificial atoms". Note that each eigenstate is two-fold degenerate due to spin. Thus, the s states are two-fold degenerate, the p states are six-fold degenerate, the d states are ten-fold degenerate and the degeneracy of the f states is 14.

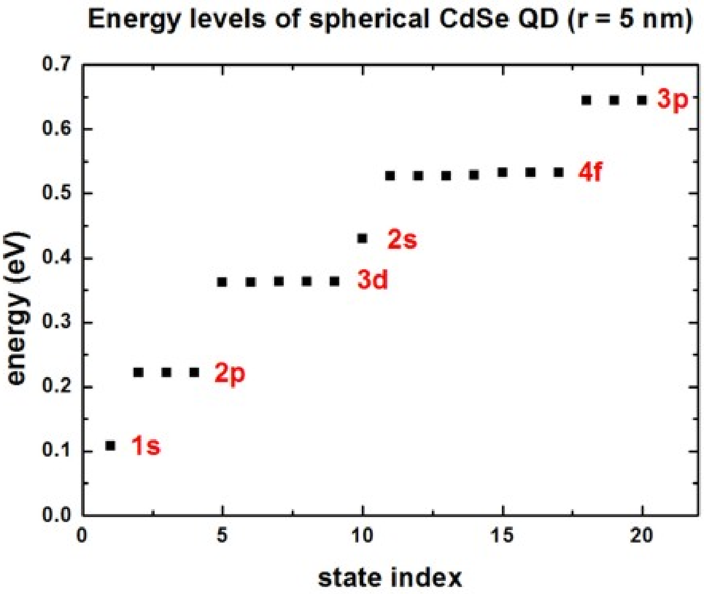


Figure 4.8.6.2: Eigenenergies of the lowest 20 states in the QD.

We have also solved the single-band Schrödinger equation for the holes assuming an isotropic effective mass for simplicity. Obviously, this is a crude approximation. From the electron and hole wave functions, we calculate their spatial overlap matrix elements (overlap integrals). In this simple model, due to symmetry arguments, only the following transitions are allowed: 1s - 1s, 2p - 2p, 3d - 3d, 2s - 2s, 4f - 4f, . . .

Figure 4.8.6.3 shows the calculated overlap integrals as a function of energy. (Note: The figure has to be updated: Now we output the square of this matrix element.)

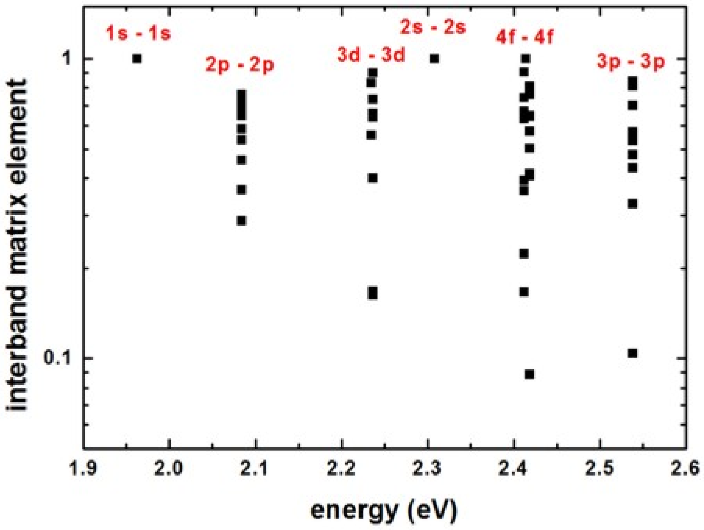


Figure 4.8.6.3: Overlap integrals.

Both figures are in reasonable agreement with Fig. 1 and Fig. 2 (inset) in *[Ferreira2006]*.

Energy levels of an "artificial atom" - Ellipsoidal, cigar-shaped CdSe quantum dot

For an ellipsoidal, cigar-shaped CdSe quantum dot ($r_x = 5\ \mathrm{nm}$, $r_y = 5\ \mathrm{nm}$, $r_z = 10\mathrm{nm}$), we calculate the lowest 30 eigenvalues.

The energy spectrum (degeneracy spectrum) looks very different from the spherical QD spectrum (c.f. Figure 4.8.6.5)

The overlap integrals are shown in Figure 4.8.6.6 (Note: The figure has to be updated: Now we output the square of this matrix element.)

Energy levels of an "artificial atom" - 2D harmonic potential

The following figure shows the energy spectrum of a "two-dimensional disc" which we approximate as a cylindrically symmetric parabolic (harmonic) potential. We solve the 2D Schrödinger equation for this system. The harmonic potential is assumed to be $\hbar\omega = 3\ \mathrm{meV}$. Each shell is thus separated by 3 meV. From the energy spectrum of this two-dimensional shell structure, one can derive "magic numbers". (They include spin degeneracy.)

This tutorial also exists for *nextnano³*.

Last update: 27/05/2025

4.9 Electronic Band Structures

4.9.1 k.p dispersion in bulk GaAs (strained / unstrained)

Input files:

- *bulk_kp_dispersion_GaAs_nnp.in*

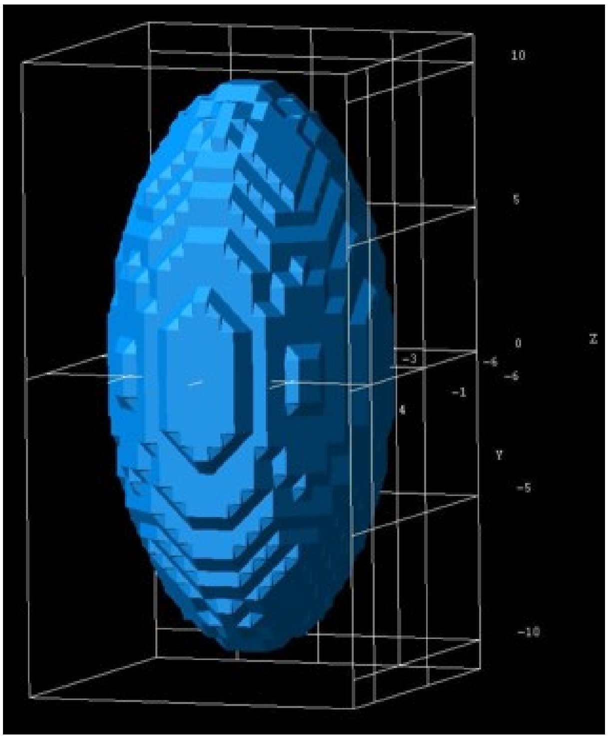


Figure 4.8.6.4: Ellipsoidal quantum dot.

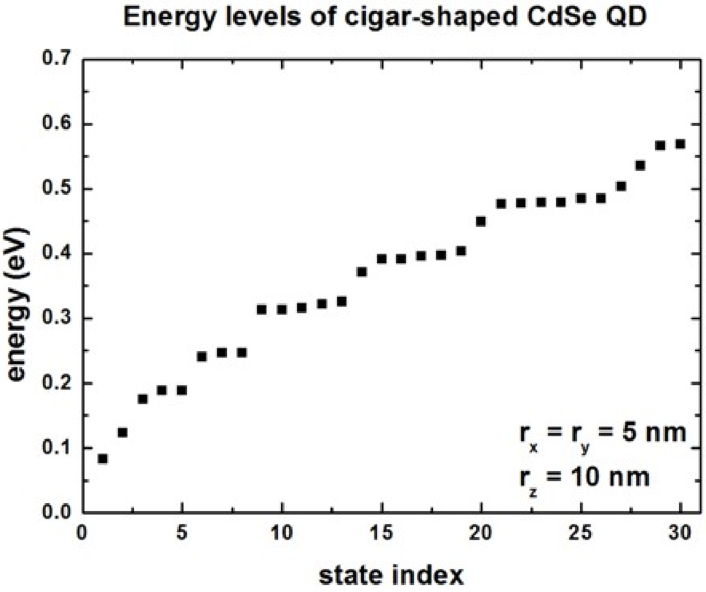


Figure 4.8.6.5: Eigenenergies of the lowest 30 states in the QD.

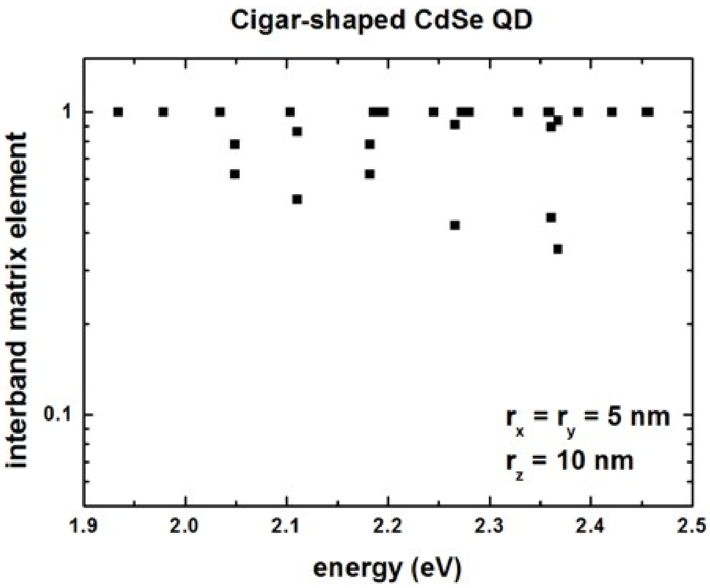


Figure 4.8.6.6: Overlap integrals.

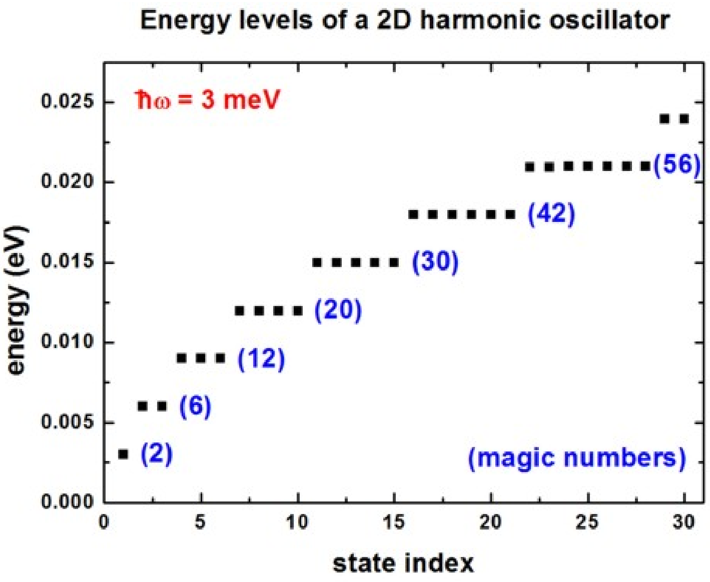


Figure 4.8.6.7: Eigenenergies of the lowest 30 states in a harmonic potential.

- *bulk_kp_dispersion_GaAs_nnp_strained.in*

Scope:
We calculate $E(k)$ of strained and unstrained $GaAs$.

Band structure of bulk $GaAs$

Input file: *bulk_kp_dispersion_GaAs_nnp.in*

We want to calculate the dispersion $E(k)$ from $|k|$ = 0 nm^{-1} to $|k|$ = 1.0 nm^{-1} along the following directions in k space:

- [000] to [110]
- [000] to [100]

We compare 6-band and 8-band k.p theory results. We calculate $E(k)$ for bulk $GaAs$ at a temperature of 300 K.

Bulk dispersion along [100] and along [110]

```
quantum{
    region{
        ...
        bulk_dispersion{
            lines{ # set of dispersion lines along crystal directions of high symmetry
                name = "lines"
                position{ x = 5.0 }
                k_max = 1.0
                spacing  = 0.01
                shift_holes_to_zero = yes
            }

            path{ # dispersion along arbitrary path in k-space
                name = "user_defined_path"
                position{ x = 5.0 }
                point{ k = [0.7071, 0.7071, 0.0] }
                point{ k = [0.0, 0.0, 0.0] }
                point{ k = [1.0, 0.0, 0.0] }
                spacing  = 0.01
                shift_holes_to_zero = yes
            }
        }
    }
}
```

We calculate the pure bulk dispersion at position x = 5 nm. In our case this is $GaAs$, but it could be any strained alloy. In the latter case, the k.p Bir-Pikus strain Hamiltonian will be diagonalized. The grid point at `position{ x = 5.0 }` must be located inside a quantum region. `shift_holes_to_zero = yes` forces the top of the valence band to be located at 0 eV. How often the bulk k.p Hamiltonian should be solved can be specified via `spacing`. To increase the resolution, just increase this number. We use two direction in k space, i.e. from [000] to [110] and from [000] to [100]. In the latter case the maximum value of $|k|$ is

$$k_{\max} = \sqrt{0.7071^2 + 0.7071^2} = 1.0$$

Note that for values of $|k|$ larger than 1.0 nm^{-1}, k.p theory might not be a good approximation anymore.

The results of the calculation can be found in the folder *bias_00000\Quantum\Bulk_dispersions*. Figure 4.9.1.1 visualizes the results.

The split-off energy of 0.341 eV is identical to the split-off energy as defined in the database:

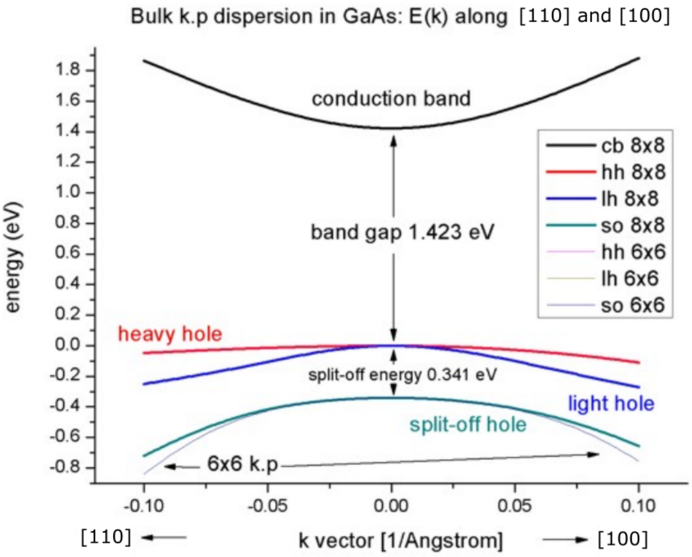


Figure 4.9.1.1: Bulk k.p dispersion in $GaAs$: $E(k)$ along [100] and [110].

```
...
valence_bands{ delta_SO = 0.341 } # [eV] Vurgaftman1
...
```

If one zooms into the holes and compares 6-band vs. 8-band k.p, one can see that 6-band and 8-band coincide for $|k|$ < 1.0 nm^{-1} for the heavy and light hole but differ for the split-off hole at larger $|k|$ values, see Figure 4.9.1.2.

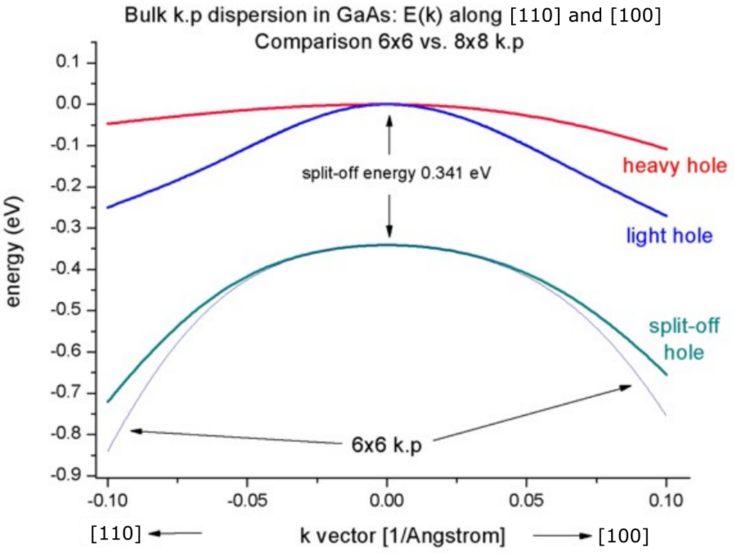


Figure 4.9.1.2: Bulk k.p dispersion in GaAs: $E(k)$ along [100] and [110] - Comparision between 6x6 and 8x8 k.p

8-band k.p vs. effective-mass approximation

Now we want to compare the 8-band k.p dispersion with the effective-mass approximation. The effective mass approximation is a simple parabolic dispersion which is isotropic (i.e. no dependence on the k vector direction). For low values of k ($|k|$ < 0.4 nm^{-1}) it is in good agreement with k.p theory, see Figure 4.9.1.3.

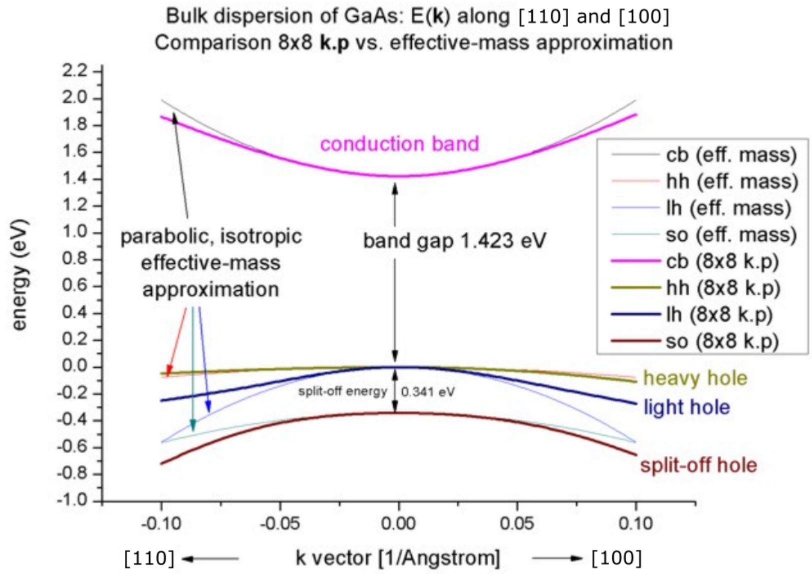


Figure 4.9.1.3: Bulk k.p dispersion in $GaAs$: $E(k)$ along [100] and [110] - Comparision between 8x8 k.p and effective-mass approximation

Band structure of strained $GaAs$

Input file: *bulk_kp_dispersion_GaAs_nnp_strained.in*

Now we perform these calculations again for $GaAs$ that is strained with respect to $In_{0.2}Ga_{0.8}As$. The $InGaAs$ lattice constant is larger than the $GaAs$ one, thus $GaAs$ is strained tensely. The changes that we have to make in the input file are the following:

```
strain{
    pseudomorphic_strain{ }
}
```

```
run{
    strain{ }
}
```

As substrate material we take $In_{0.2}Ga_{0.8}As$ and assume that $GaAs$ is strained pseudomorphically (`pseudomorphic_strain{ }`) with respect to this substrate, i.e. $GaAs$ is subject to a biaxial strain. Due to the positive hydrostatic strain (i.e. increase in volume or negative hydrostatic pressure) we obtain a reduced band gap with respect to the unstrained $GaAs$. Furthermore, the degeneracy of the heavy and light hole at $k` = 0 is lifted, see : numref : `fig - 1D - kp - dispersion - bulk - GaAs - kp - bandedges - strained.$ Now, the anisotropy of the holes along the different directions [100] and [110] is very pronounced. There is even a band anti-crossing along [100]. (Actually, the anti-crossing looks like a “crossing” of the bands but if one zooms into it (not shown in this tutorial), one can easily see it.) Note: If biaxial strain is present, the directions along x, y or z are not equivalent anymore. This means that the dispersion is also different in these directions ([100], [010], [001]).

If one zooms into the holes and compares 6-band vs. 8-band k.p, one can see that the agreement between heavy and light holes is not as good as in the unstrained case where 6-band and 8-band k.p lead to almost identical dispersions, compare Figure 4.9.1.5.

Note that in the strained case, the effective-mass approximation is very poor.

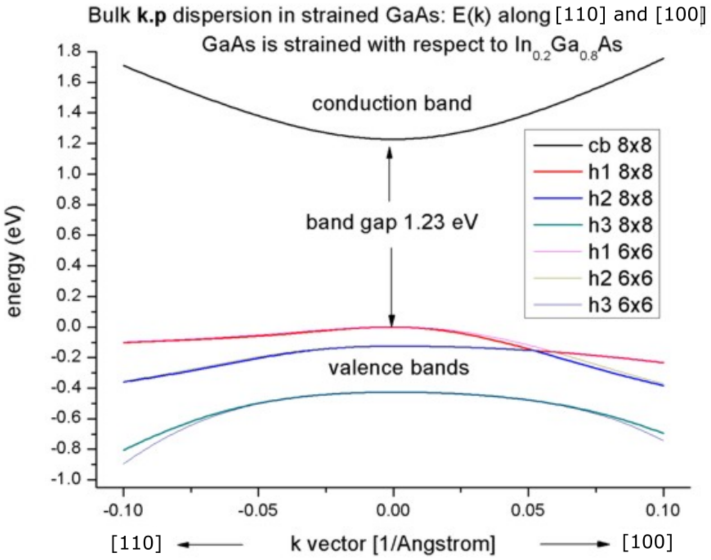


Figure 4.9.1.4: Bulk k.p dispersion in $GaAs$ strained with respect to $In_{0.2}Ga_{0.8}As$: $E(k)$ along [100] and [110].

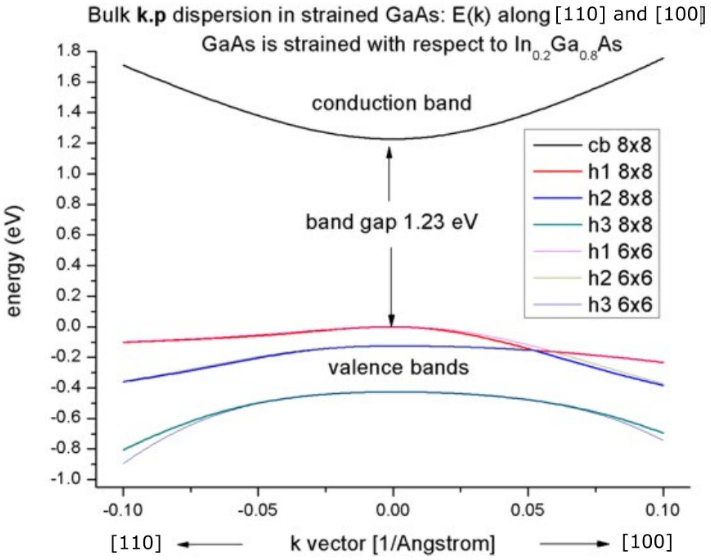


Figure 4.9.1.5: Bulk valence band k.p dispersion in $GaAs$ strained with respect to $In_{0.2}Ga_{0.8}As$: $E(k)$ along [100] and [110] - Comparision between 6x6 and 8x8 k.p approximation.

Analysis of eigenvectors

(preliminary)

Using the Voon-Willatzen-Bastard-Foreman k.p basis one obtains the following output for the eigenvectors at the Gamma point, $k = (k_x, k_y, k_z) = 0$.

Example: The x_up component contains a complex number. Here, we show the square of X_up. This gives us information on the strength of the coupling of the mixed states.

```
   eigenvalue   S+     S-     HH     LH     LH     LH     SO     SO
1           0      1.0    0      0      0      0      0      0
2           1.0    0      0      0      0      0      0      0
3           0      0      0      1.0    0      0      0      0
4           0      0      0      0      1.0    0      0      0
5           0      0      0      0      0      1.0    0      0
6           0      0      1.0    0      0      0      0      0
7           0      0      0      0      0      0      0      1.0
8           0      0      0      0      0      0      1.0    0

eigenvalue  S+          S-          X+          Y+          Z+          X-          Y-
↪           Z-
1           1.0         0           0           0           0           0           0␣
↪           0
2           0           1.0         0           0           0           0           0␣
↪           0
3           0           0           0           0           0.5         0.5         0␣
↪           0
4           0           0           0           0           0.166       0.166       0.
↪666        0
5           0           0           0.5         0           0           0           0␣
↪           0.5
6           0           0           0.166       0.666       0           0           0␣
↪           0.166
7           0           0           0           0           0.333       0.333       0.
↪333        0
8           0           0           0.333       0.333       0           0           0␣
↪           0.333
```

+: spin up, -: spin down

- The electron eigenstates are 2-fold degenerate, i.e. have the same energy, and are decoupled from the holes.

1	$\lvert S\downarrow\rangle$
2	$\lvert S\uparrow\rangle$

- The hole eigenstates are 4-fold (heavy and light holes) and 2-fold degenerate (split-off holes).

3	$\left\lvert\frac{3}{2},\frac{3}{2}\right\rangle$	hh spin up	$\frac{1}{\sqrt{2}}\lvert(X+iY)\uparrow\rangle$
4	$\left\lvert\frac{3}{2},\frac{1}{2}\right\rangle$	lh	$\frac{1}{\sqrt{6}}\lvert(X+iY)\downarrow\rangle-\sqrt{\frac{2}{3}}\lvert Z\uparrow\rangle$
5	$\left\lvert\frac{3}{2},-\frac{1}{2}\right\rangle$	lh	$\frac{1}{\sqrt{6}}\lvert(X-iY)\uparrow\rangle-\sqrt{\frac{2}{3}}\lvert Z\downarrow\rangle$
6	$\left\lvert\frac{3}{2},-\frac{3}{2}\right\rangle$	hh spin down	$\frac{1}{\sqrt{2}}\lvert(X-iY)\downarrow\rangle$
7	$\left\lvert\frac{1}{2},\frac{1}{2}\right\rangle$	s/o split	$\frac{1}{\sqrt{3}}\lvert(X+iY)\downarrow\rangle-\frac{1}{\sqrt{3}}\lvert Z\uparrow\rangle$
8	$\left\lvert\frac{1}{2},-\frac{1}{2}\right\rangle$	s/o split	$\frac{1}{\sqrt{3}}\lvert(X-iY)\downarrow\rangle-\frac{1}{\sqrt{3}}\lvert Z\downarrow\rangle$

$$\frac{1}{\sqrt{2}} = 0.707 \rightarrow \left(\frac{1}{2}\right)^2 = 0.5$$

$$\frac{1}{\sqrt{3}} = 0.577 \rightarrow \left(\frac{1}{3}\right)^2 = 0.333$$
$$\frac{1}{\sqrt{6}} = 0.408 \rightarrow \left(\frac{1}{6}\right)^2 = 0.166$$

Last update: nnnn/nn/nn

4.9.2 k.p dispersion in bulk unstrained, compressively and tensely strained GaN (wurtzite)

Input files:

- *bulk_kp_dispersion_GaN_unstrained_0_nnp.in*
- *bulk_kp_dispersion_GaN_unstrained_90_nnp.in*
- *bulk_kp_dispersion_GaN_strained_compressive_0_nnp.in*
- *bulk_kp_dispersion_GaN_strained_compressive_90_nnp.in*
- *bulk_kp_dispersion_GaN_strained_tensile_0_nnp.in*
- *bulk_kp_dispersion_GaN_strained_tensile_90_nnp.in*
- *bulk_kp_dispersion_GaN_strained_tensile_90_3D_nnp.in*

Scope:

We calculate $E(k)$ for bulk GaN (unstrained), with compressive and tensile strain, along two different growth directions. In this tutorial we aim to reproduce results of *[ParkChuangPRB1999]* and *[KumagaiChuangAndoPRB1998]*.

k.p dispersion in bulk unstrained GaN (wurtzite)

We want to calculate the dispersion $E(k)$ from $|k| = 0$ to $|k| = 1.0$ [1/nm] along the following directions in k space:

- [010] to [100]
- [011] to [100]
- [111] to [100]

We compare 6-band k.p theory results vs. single-band (effective-mass) results for unstrained GaN. Material parameters used in the calculations are taken from *[KumagaiChuangAndoPRB1998]*.

Calculating the bulk k.p dispersion

```
quantum{
    region{
        ...
        bulk_dispersion{
            path{ # dispersion along arbitrary path in k-space
                name = "user_defined_path"
                position{ x = 2.0 }
                point{ k = [0.0, 0.0, 0.0] }
                point{ k = [1.2, 0.0, 0.0] }
                spacing  = 0.012                # [1/nm]
                shift_holes_to_zero = no
            }
        }
    }
}
```

The maximum value of $|k|$ is 1.2 nm^{-1}. Note that for values of $|k|$ larger than 1.0 nm^{-1}, k.p theory might not be a good approximation anymore. We calculate the pure bulk dispersion at x = 2.0, i.e. for the material located at the grid point at 2 nm. In our case this is GaN, but it could be any strained alloy. If strain is present (see below), the k.p Bir-Pikus strain Hamiltonian will be diagonalized at each k point. The grid point at grid-position must be located inside a quantum region. `shift_holes_to_zero = yes` forces the top of the valence band to be located at 0 eV. In this tutorial, however, we use no. The "average" energy of all three valence bands is set to the zero point of energy. Here, "average" means without taking crystal field and spin-orbit splitting into account. This is added afterwards to get the energies of heavy hole (HH), light hole (LH) and crystal-field split-off hole (CH). How often the bulk k.p Hamiltonian should be solved can be specified via `spacing`. To increase the resolution, just increase this number. The results can be found in the folder *bias_00000\Quantum\Bulk_dispersions*. Figure 4.9.2.1 shows the bulk k.p dispersion of unstrained GaN (wurtzite). The results are in excellent agreement to Fig. 4 (b) of *[KumagaiChuangAndoPRB1998]*.

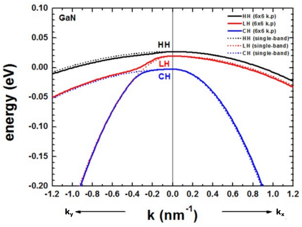


Figure 4.9.2.1: Calculated 1-band and k.p dispersion of HH, LH and CH valence bands (unstrained). The k_x direction corresponds to the c axis [0001]. The dispersion along k_y and k_z is identical (only k_y is shown), i.e. the dispersion in the (100) plane is isotropic.

The dispersion along the hexagonal c axis is substantially different.

If the average of the three valence band edges (without taking crystal-field and spin-orbit splitting into account) is defined to be at zero, i.e. $E_{v,av}$ = 0 eV, then the energies E_1, E_2 and E_3 are defined as follows for the unstrained case:

$$E_1 = \Delta_1 + \Delta_2$$

$$E_2 = B + A$$

$$E_3 = B - A$$

where Δ_1 is the crystal field split energy Δ_{cr}, and Δ_2 and Δ_3 are related to the spin-orbit split off-energy Δ_{so} as follows:

$$\Delta_2 = \Delta_3 = 1/3\Delta_{so}$$

$$B = (\Delta_1 - \Delta_2)/2$$

$$A = \sqrt{B^2 + 2(\Delta_3)^2}$$

The Delta parameters are defined in the database

```
valence_bands{
    defpotentials = [ -1.70, 6.30, 8.00, -4.00, -4.0, -5.5 ]
    delta = [ 0.0220, 0.005, 0.005 ]                        # Delta1(cr), Delta2␣
↪= Delta_so/3, Delta3 = Delta_so/3
}
```

leading to:

$$B = 0.0085$$

$$A = 0.01106$$

$$E_1 = \Delta_1 + \Delta_2 = 0.027eV$$

$$E_2 = B + A = 0.0085eV + 0.01106eV = 0.01956eV$$

$$E_3 = B - A = 0.0085 - 0.01106 = -0.00256eV$$

In contrast to zincblende materials, even in the unstrained case, the heavy and light hole are not degenerate at k = 0. For comparison, we also show the dispersion using the single-band effective mass approximation (dotted lines). We used the following values for the effective hole masses, according to reference http://www.ioffe.rssi.ru/SVA/NSM/Semicond/GaN/bandstr.html.

$$m_{HH,a} = 1.6\ [m_0], \quad m_{HH,c} = 1.1\ [m_0]$$

$$m_{LH,a} = 0.15\ [m_0], \quad m_{LH,c} = 1.1\ [m_0]$$

$$m_{CH,a} = 1.1\ [m_0], \quad m_{CH,c} = 0.15\ [m_0]$$

The effective mass approximation is a simple parabolic dispersion which is isotropic in zincblende materials (i.e. no dependence on the k vector direction) but is anisotropic for wurtzite materials due to the different effective masses parallel and perpendicular to the c axis.

k.p dispersion in compressively and tensilely strained GaN (wurtzite)

We compare two different orientations of the crystal coordinate system with respect to the simulation coordinate system.

- Case a) Default orientation: hexagonal c axis oriented along the x direction [100]
- Case b) Rotation of hexagonal c axis by 90 degrees so that it oriented along the default y direction [010]

The orientation of the z axis remains the same.

The following figures compare the 6-band k.p valence band dispersion relation of compressively (-0.5%, Figure 4.9.2.2) vs. tensely (+0.5%, Figure 4.9.2.3) strained GaN. Assuming that the substrate material is $Al_xIn_{1-x}N$,

- a compressive strain of -0.5% corresponds to $Al_{0.785}In_{0.215}N$ (e_{yy} = e_{zz} = -0.005)
- a tensile strain of 0.5% corresponds to $Al_{0.859}In_{0.141}N$ (e_{yy} = e_{zz}‘ = 0.005)

using the lattice constants given *[ParkChuangPRB1999]*, *[KumagaiChuangAndoPRB1998]*. The results for tensile strain indicate that the light hole (LH) band is higher in energy than the heavy hole (HH) band.

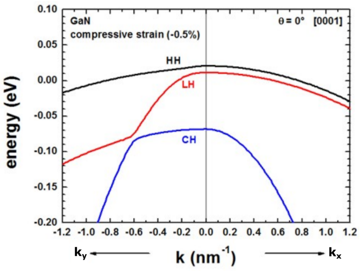


Figure 4.9.2.2: Calculated k.p dispersion of HH, LH and CH valence bands (compressive strain)

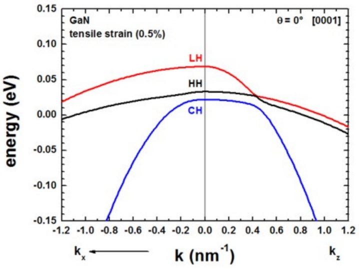


Figure 4.9.2.3: Calculated k.p dispersion of HH, LH and CH valence bands (tensile strain)

The results of these two figures can be found in this file: *bulk_dispersion_qr_6band_kp6_010_to_100.dat*, where 010 represents the k_y direction, 000 the Gamma point and 100 the k_x direction, i.e. the plotted dispersion is a cut through the 3D Brillouin zone along these lines. We only plotted the result for k_y. The dispersion along k_z is identical in this case, also the dispersion along [011], i.e. the dispersion is isotropic with respect to the (100) plane.

Once the c axis is oriented along the x axis of the simulation coordinate system (rotation by 90° around the z axis), the corresponding results look as follows.

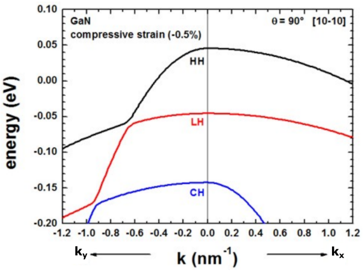


Figure 4.9.2.4: Calculated k.p dispersion of HH, LH and CH valence bands (compressive strain)

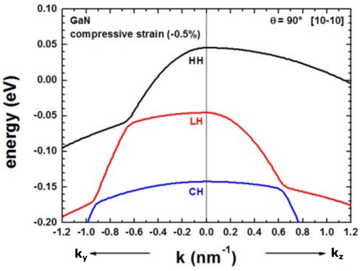


Figure 4.9.2.5: Calculated k.p dispersion of HH, LH and CH valence bands (compressive strain)

The results of Figure 4.9.2.4 and Figure 4.9.2.6 can be found in this file: *bulk_dispersion_qr_6band_kp6_010_to_100.dat*, where 010 represents the k_y direction, 000 the Gamma point and 100 the k_x direction, i.e. the plotted dispersion is a cut through the 3D Brillouin zone along these lines. The results for the dispersion along k_z is now different from the dispersion along k_y. The results for k_z are contained in this file *bulk_dispersion_qr_6band_kp6_010_to_001.dat*, because here we specified in the input file to calculate the dispersion from the Gamma point (0,0,0) to (k_x, k_y, k_z) = (0, 0, 1.0 nm^{-1}).

```
bulk_dispersion{
    path{
        name = "010_to_001"
        position{ x = 5.0 }
        point{ k = [0.0, 1.0, 0.0] }
        point{ k = [0.0, 0.0, 0.0] }
        point{ k = [0.0, 0.0, 1.0] }
        spacing  = 0.01
        shift_holes_to_zero = yes
    }
}
```

Note: For θ = 90°, we have rotated the crystal (cr) coordinate system with respect to the simulation (sim) coordinate system. Therefore, for our new orientation it holds $e_{zz,cr}$ = $e_{zz,sim}$ = $\pm$ 0.005 and $e_{yy} \neq e_{zz}$.

The results of our figures are in excellent agreement to figures 5 and 6 of the paper *[ParkChuangPRB1999]*.

Note that for the case of tensile strain and orientation of the c axis along the [10-10] orientation, the strain tensor component along the z direction of the simulation system is tensilely strained, whereas the component along the y direction is compressively (!) strained.

For a discussion of the figures please refer to *[ParkChuangPRB1999]*.

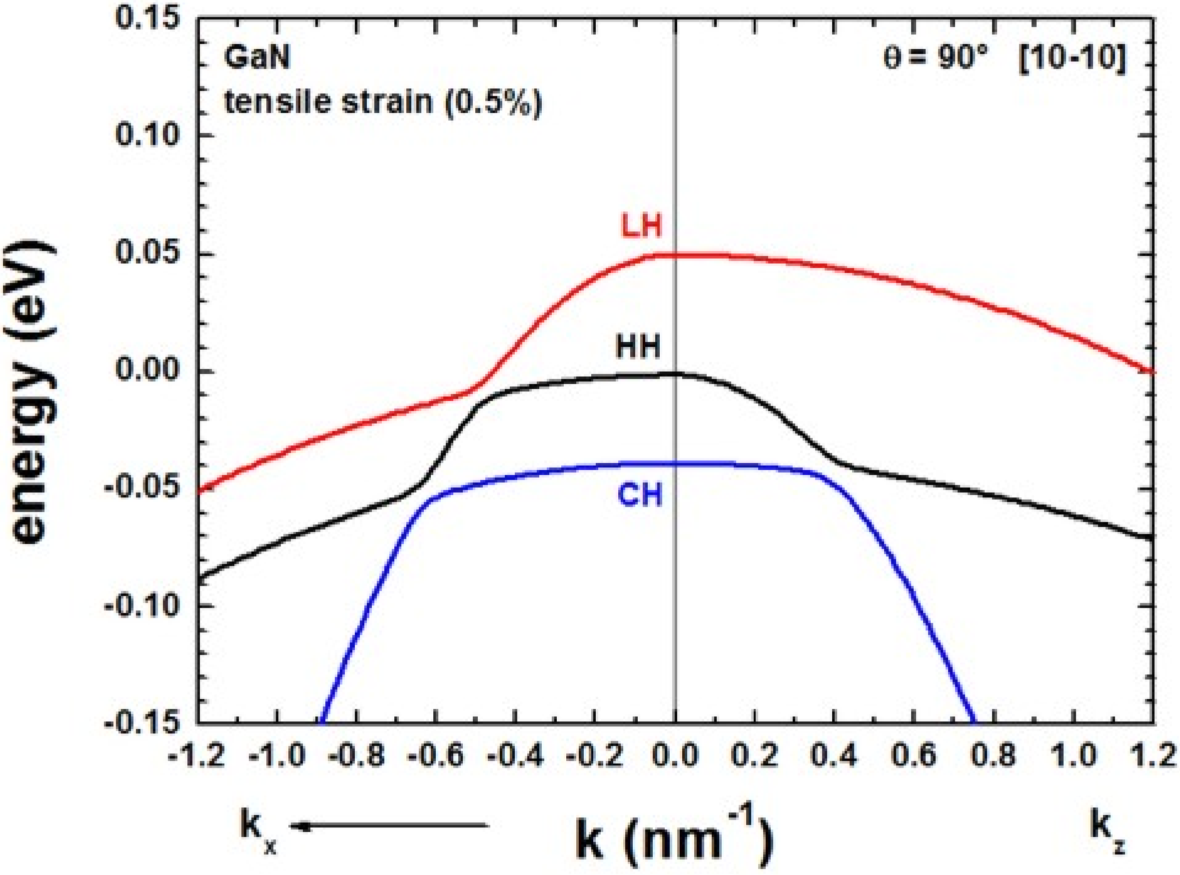


Figure 4.9.2.6: Calculated k.p dispersion of HH, LH and CH valence bands (tensile strain)

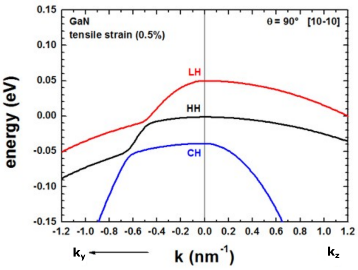


Figure 4.9.2.7: Calculated k.p dispersion of HH, LH and CH valence bands (tensile strain)

Energy dispersion E(k) in three dimensions

Alternatively one can print out the 3D data field of the bulk $E(\mathbf{k}) = E(k_x, k_y, k_z)$ dispersion.

```
full{ # 3D dispersion on rectilinear grid in k-space
    name = "3D"
    position{ x = 5.0 }
    kxgrid {
        line{ pos = -1  spacing = 0.04 }
        line{ pos =  1  spacing = 0.04 }
    }
    kygrid {
        line{ pos = -1  spacing = 0.04 }
        line{ pos =  1  spacing = 0.04 }
    }
    kzgrid {
        line{ pos = -1  spacing = 0.04 }
        line{ pos =  1  spacing = 0.04 }
    }
    shift_holes_to_zero = yes
    }
}
```

The grid in k space is determined by `spacing` and `pos`.

Figure 4.9.2.8 shows a 2D slice in the (k_y, k_z) plane for $k_x = 0$ of the highest lying hole state for the tensely strained GaN (oriented along 90°, i.e. x is oriented along [10-10]) is shown in this figure. Right: Horizontal and vertical slice through the center coordinate at (k_x, k_y, k_z) = (0, 0, 0).

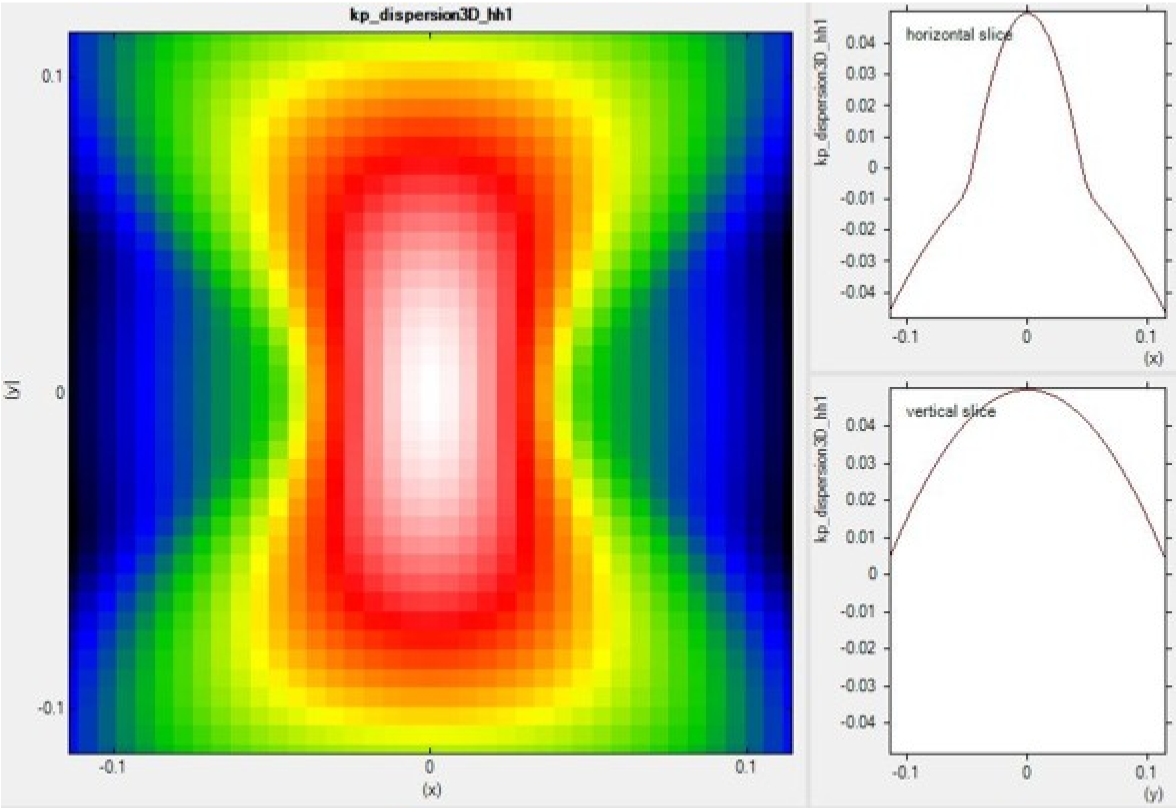


Figure 4.9.2.8: 2D slice at $k_x = 0$ of calculated 3D dispersion.

Last update: nnnn/nn/nn

4.9.3 k.p dispersion in bulk unstrained ZnS, CdS, CdSe and ZnO (wurtzite)

Input files:

- *bulk_6x6kp_dispersion_ZnS_nnp.in*
- *bulk_6x6kp_dispersion_CdS_nnp.in*
- *bulk_6x6kp_dispersion_CdSe_nnp.in*
- *bulk_6x6kp_dispersion_ZnO_nnp.in*

Scope:

We calculate $E(k)$ for bulk ZnS, CdS, $CdSe$ and $Zn0$ (unstrained). In this tutorial we aim to reproduce results of *[Jeon1996]*.

Introduction

We want to calculate the dispersion $E(k)$ from $|k|$ = 0 [1/nm] to $|k|$ = 1.0 [1/nm] along the following directions in k space:

- [000] to [0001], i.e. parallel to the c axis (Note: The c axis is parallel to the z axis.)
- [000] to [110], i.e. perpendicular to the c axis (Note: The (x, y) plane is perpendicular to the c axis.)

We compare 6-band k.p theory results vs. single-band (effective-mass) results.

Bulk dispersion along [0001] and [110]

```
quantum{
    region{
        ...
        bulk_dispersion{
            path{ # dispersion along arbitrary path in k-space
                name = "user_defined_path"
                position{ x = 5.0 }
                point{ k = [0.7071, 0.7071, 0.0] }
                point{ k = [0.0, 0.0, 1.0] }
                spacing  = 0.01                 # [1/nm]
                shift_holes_to_zero = yes
            }
        }
    }
}
```

We calculate the pure bulk dispersion at grid position `x = 5.0`, i.e. for the material located at the grid point at 5 nm. In our case this is ZnS but it could be any strained alloy. In the latter case, the k.p Bir-Pikus strain Hamiltonian will be diagonalized. The grid point inside `position{}` must be located inside a quantum region. `shift_holes_to_zero = yes` forces the top of the valence band to be located at 0 eV. How often the bulk k.p Hamiltonian should be solved can be specified via `spacing`. To increase the resolution, just increase this number. The maximum value of $|k|$ is 1.0 [1/nm]. Note that for values of $|k|$ larger than 1.0 [1/nm], k.p theory might not be a good approximation any more. This depends on the material system, of course. Start the calculation. The results can be found in the folder *bias_00000\Quantum\Bulk_dispersions*.

The files *bulk_6x6kp_dispersion_as_in_inputfile_kxkykz_000_kxkykz.dat* for instance contain 6-band k.p dispersions: The first column contains the $|k|$ vector in unitsHere we visualize the results. The final figures will look like this (left: dispersion along [0001], right: dispersion along [110]): of [1/nm], the next six columns the six eigenvalues of the 6-band k.p Hamiltonian for this $k = (k_x, k_y, k_z)$ point.

The resulting energy dispersion in 6-band k.p theory is usually discussed in terms of a nonparabolic and anisotropic energy dispersion of heavy, light and split-off holes, including valence band mixing.

The single-band effective mass dispersion is parabolic and depends on a single parameter: The effective mass m^*. Note that in wurtzite materials, the mass tensor is usually anisotropic with a mass m_{zz} parallel to the c axis, and two masses perpendicular to it $m_{xx} = m_{yy}$.

Results

We visualize now the results in Figure 4.9.3.1, Figure 4.9.3.2 and Figure 4.9.3.3. The final figures will look like this (left: dispersion along [0001], right: dispersion along [110]):

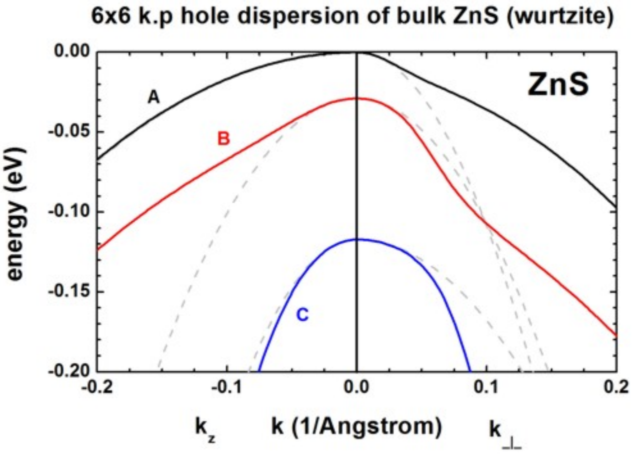


Figure 4.9.3.1: Calculated 1-band (dotted gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

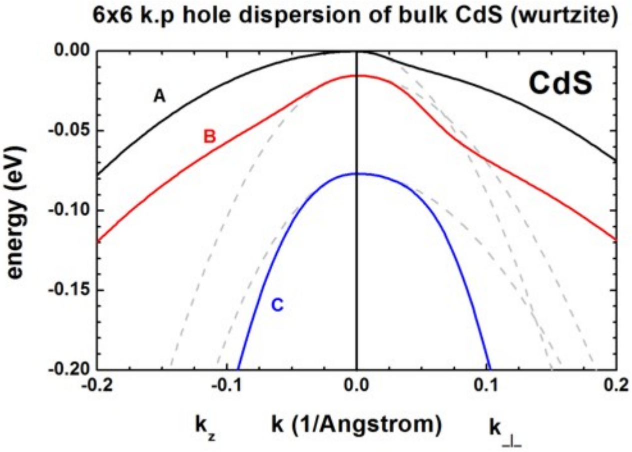


Figure 4.9.3.2: Calculated 1-band (dotted gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

These three figures are in excellent agreement to Fig. 1 of the paper by *[Jeon1996]*. The dispersion along the hexagonal c axis is substantially different from the dispersion in the plane perpendicular to the c axis. The effective mass approximation is indicated by the dashed, gray lines. For the heavy holes (A), the effective mass approximation is very good for the dispersion along the c axis, even at large k vectors.

For comparison, the single-band (effective-mass) dispersion is also shown. For ZnS, it corresponds to the following effective hole masses:

```
valence_bands{
    HH{ mass_l = 2.23  mass_t = 0.35}   # [m0] heavy hole A  (2.23 along c axis)
    LH{ mass_l = 0.53  mass_t = 0.485}  # [m0] light hole B  (0.53 along c axis)
    SO{ mass_l = 0.32  mass_t = 0.75}   # [m0] crystal hole C  (0.32 along c axis)
}
```

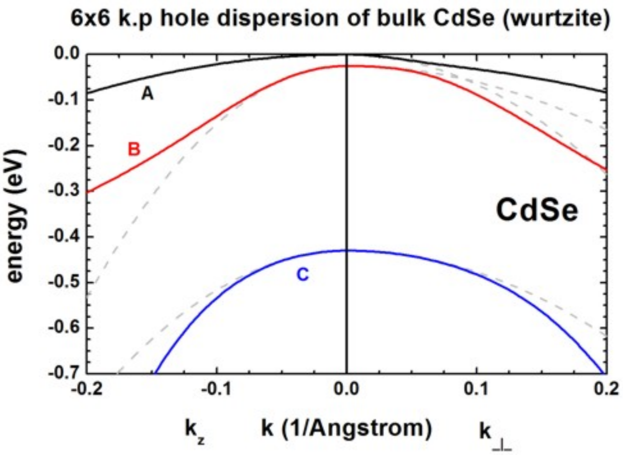


Figure 4.9.3.3: Calculated 1-band (dotted gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

The effective mass approximation is a simple parabolic dispersion which is anisotropic if the mass tensor is anisotropic (i.e. it also depends on the k vector direction).

One can see that for $|k| < 0.5$ [1/nm] the single-band approximation is in excellent agreement with 6-band k.p, but differs at larger $|k|$ values substantially.

Plotting $E(k)$ in three dimensions

Alternatively one can print out the 3D data field of the bulk $E(k) = E(k_x, k_y, k_z)$ dispersion.

```
full{ # 3D dispersion on rectilinear grid in k-space
    name = "3D"
    position{ x = 5.0 }
    kxgrid {
        line{ pos = -1  spacing = 0.04 }
        line{ pos =  1  spacing = 0.04 }
    }
    kygrid {
        line{ pos = -1  spacing = 0.04 }
        line{ pos =  1  spacing = 0.04 }
    }
    kzgrid {
        line{ pos = -1  spacing = 0.04 }
        line{ pos =  1  spacing = 0.04 }
    }
    shift_holes_to_zero = yes
    }
}
```

k.p dispersion in bulk unstrained ZnO

Figure 4.9.3.4 shows the bulk 6-band k.p energy dispersion for ZnO. The gray lines are the dispersions assuming a parabolic effective mass.

The following files are plotted:

- *bulk_6x6kp_dispersion_as_in_inputfile_kxkykz_000_kxkykz.dat*
- *bulk_sg_dispersion.dat*

The files

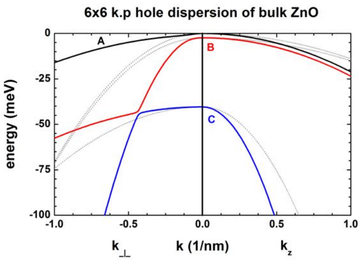


Figure 4.9.3.4: Calculated parabolic effective mass (dotted, gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

- *bulk_6x6kp_dispersion_axis_-100_000_100.dat* and
- *bulk_6x6kp_dispersion_diagonal_-110_000_1-10.dat*

contain the same data because for a wurtzite crystal due to symmetry. The dispersion in the plane perpendicular to the k_z direction (corresponding to [0001]) is isotropic.

Last update: nnnn/nn/nn

4.9.4 Energy dispersion of holes in a quantum well

Input files:

- *1Dwell_GaAs_AlAs_nnp.in*
- *1Dwell_GaSb_AlSb_nnp.in*
- *1Dwell_InGaAs_InP_nnp.in*

Scope:

In this tutorial we aim to reproduce results of *[FranceschiJancuBeltram1999]* and *[Holleitner2007]*.

a) Unstrained $GaAs/AlAs$ quantum well

Input file: *1Dwell_GaAs_AlAs_nnp.in*

This input file simulates a $GaAs$ (well)/ $AlAs$ (barrier) structure - The well is 17 molecular layers thick (4.8 nm), located between x = 20 nm and x = 24.8 nm.

Figure 4.9.4.1 shows the valence band edges of the quantum well structure together with three quantized states. The heavy and light hole band edges are degenerate. The red band is the split-off hole band edge. Note that these artificial band edges correspond to the bulk band edges. Also shown are the probability densities of the three uppermost subbands (Ψ^2). Note that each eigenstate is twofold spin-degenerate at $k_{||}$ = 0. These eigenfunctions are plotted as positions on the energy scale that correspond to their eigenenergies, i.e. Ψ^2 + eigenvalue (eV). The energy scale is shifted by -1.45967 eV to refer to the bulk valence band edge of the quantum well material, i.e. the $GaAs$ valence band edge (hh, lh) is at 0 eV.

We use a 6-band k.p model for the holes.

```
quantum {
    region{
        name = "quantum_region"
        x = [10, 34.8]
        no_density = yes
        boundary{ x = dirichlet }
        kp_6band{ # 6-band k.p model
```

(continues on next page)

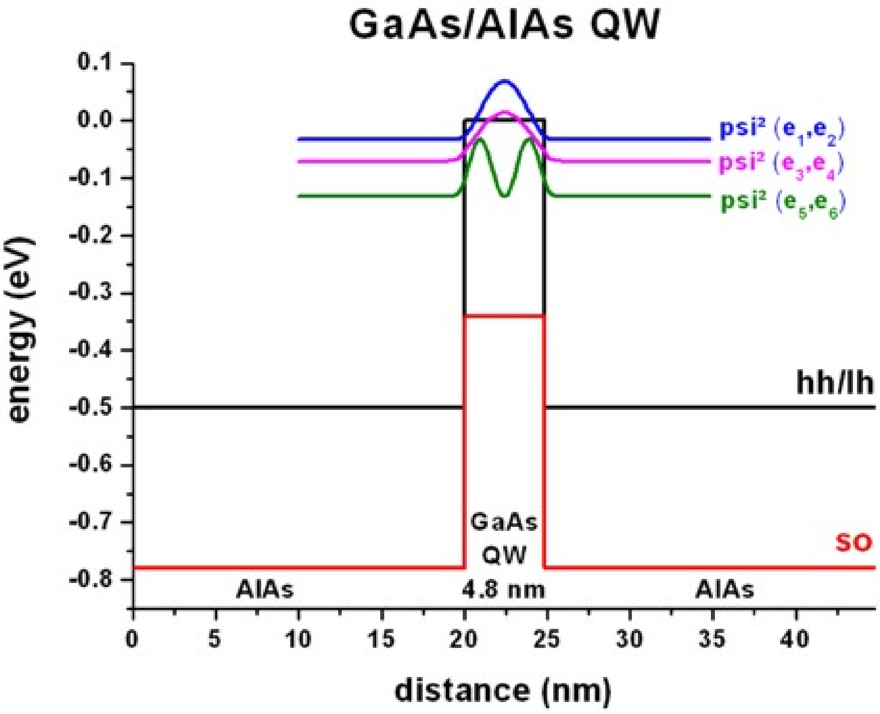


Figure 4.9.4.1: Calculated valence band edges with Ψ^2 of the lowest hole states.

(continued from previous page)

```
            num_ev = 10 # number of hole states
            dispersion{
                ...
            }
            k_integration{
                ...
            }
        }
        output_wavefunctions{ # k.p output
            max_num = 9999
            all_k_points = yes
            amplitudes = no
            probabilities = yes
        }
    }
}
```

Database: We used the Luttinger parameters (γ_1, γ_2, γ_3) given in *[FranceschiJancuBeltram1999]* and also their valence band offset (0.5 eV). The conversion from Luttinger parameters to Dresselhaus parameters (L, M, N) is described here. For details on the bandoffset see *here*. So the changes to the *database_nnp.in* file are as follows:

```
database{
    binary_zb{
        name    = AlAs
        valence = III_V

        valence_bands{
            bandoffset = 0.86633 # Ev,av [eV]
        }

```

(continues on next page)

(continued from previous page)

```
        kp_6_bands{     # Dresselhaus parameters
            L = -7.64   # [hbar^2/2m]
            M = -3.50   # [hbar^2/2m]
            N = -8.76   # [hbar^2/2m]
        }
    }

    binary_zb{
        name    = GaAs
        valence = III_V

        valence_bands{
            bandoffset = 1.346 # Ev,av [eV]
        }

        kp_6_bands{     # Dresselhaus parameters
            L = -16.050 # [hbar^2/2m]
            M = -4.050  # [hbar^2/2m]
            N = -18.000 # [hbar^2/2m]
        }
    }
}
```

The valence band offset between $InAs$ and $GaAs$ is 0.5 eV (*[FranceschiJancuBeltram1999]*) and calculated as follows:

$$\left(E_{v,av}^{GaAs} + \Delta_{SO}^{GaAs}/3\right) - \left(E_{v,av}^{InAs} + \Delta_{SO}^{InAs}/3\right)$$

$$= (1.346 + 0.341/3) - (0.86633 + 0.28/3) = 0.5eV$$

$k_{||}$ dispersion for the three uppermost subbands

The eigenvalues are twofold degenerate due to spin (and because the quantum well is symmetric). Thus, eigenvalue 1 and 2 correspond to Figure 4.9.4.2, 3 and 4 to Figure 4.9.4.3 and 5 and 6 to Figure 4.9.4.4. For the following three pictures, the energy is referred to the bulk valence band edge of the quantum well material, i.e. hh/lh($GaAs$) = 0 eV. The colors and the color bar correspond to the energy given in eV. The x and y coordinate axes refer to the in-plane wave vector. The units are in 1/Angstrom.

Now we will plot a cut through the above three pictures from [010] to the zone center and from the zone center to [011], see Figure 4.9.4.5. This plot was obtained by plotting the following file: *dispersion_quantum_region_kp6_kpar_10_00_11.dat*.

The value of the abscissa is found as follows:

- From [10] to zero we just take $-k_x$.
- From zero to [11] we take $\sqrt{k_x^2 + k_y^2}$.

The above figure shows the eigenvalues as a function of $k_{||}$ vector. The three lines correspond to the upper three eigenvalues (which are two-fold spin-degenerate) as shown in the above QW figure. The thick lines are for the nonsymmetrized k.p Hamiltonian (which is closer to the more accurate tight-binding results), the thin lines are for the symmetrized k.p Hamiltonian. The two sets of k.p subbands coincide at the Brillouin-zone center (i.e. at $k_{||}$ = 0). They do not show pronounced discrepancies at nonzero in-plane k vectors. This follows from the rather small difference between the effective-mass parameters of $GaAs$ and $AlAs$. Obviously, for larger k values, the discrepancies are more significant.

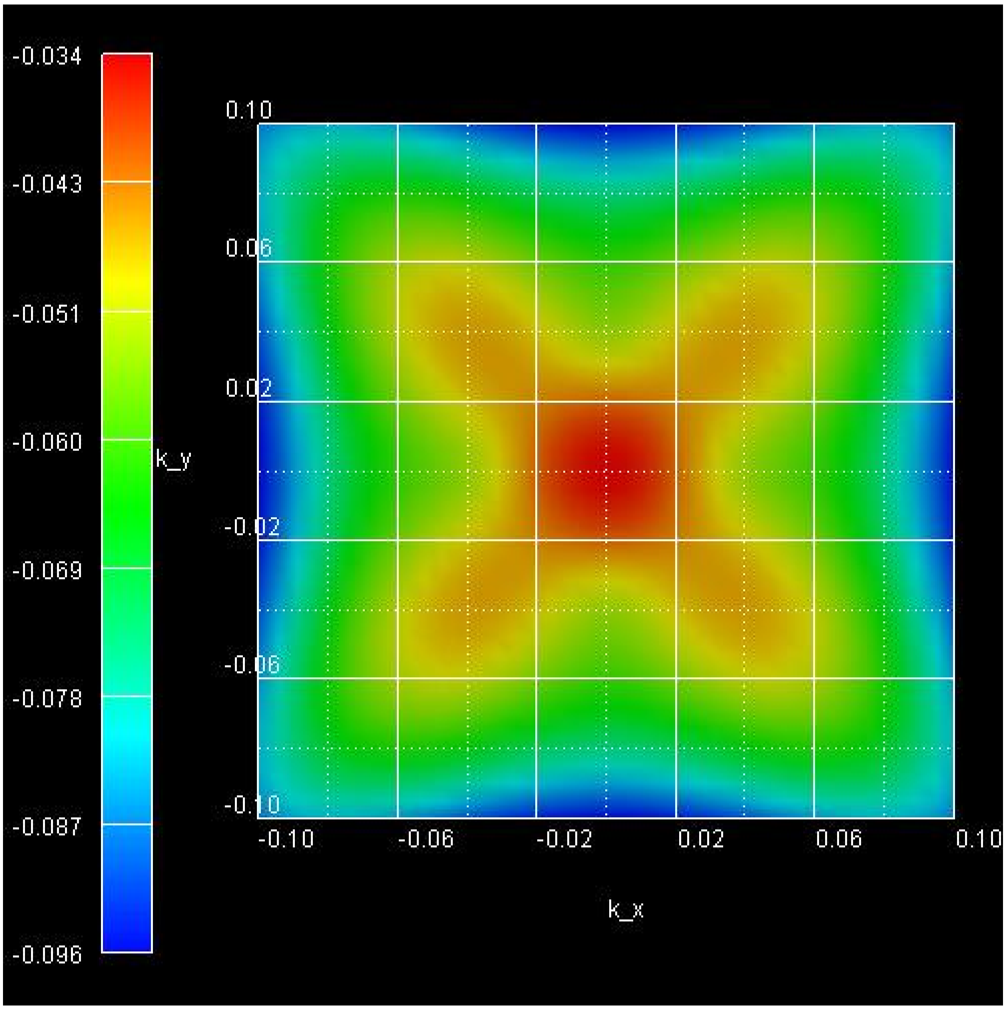


Figure 4.9.4.2: Subband 1 (eigenvalue 1 and 2)

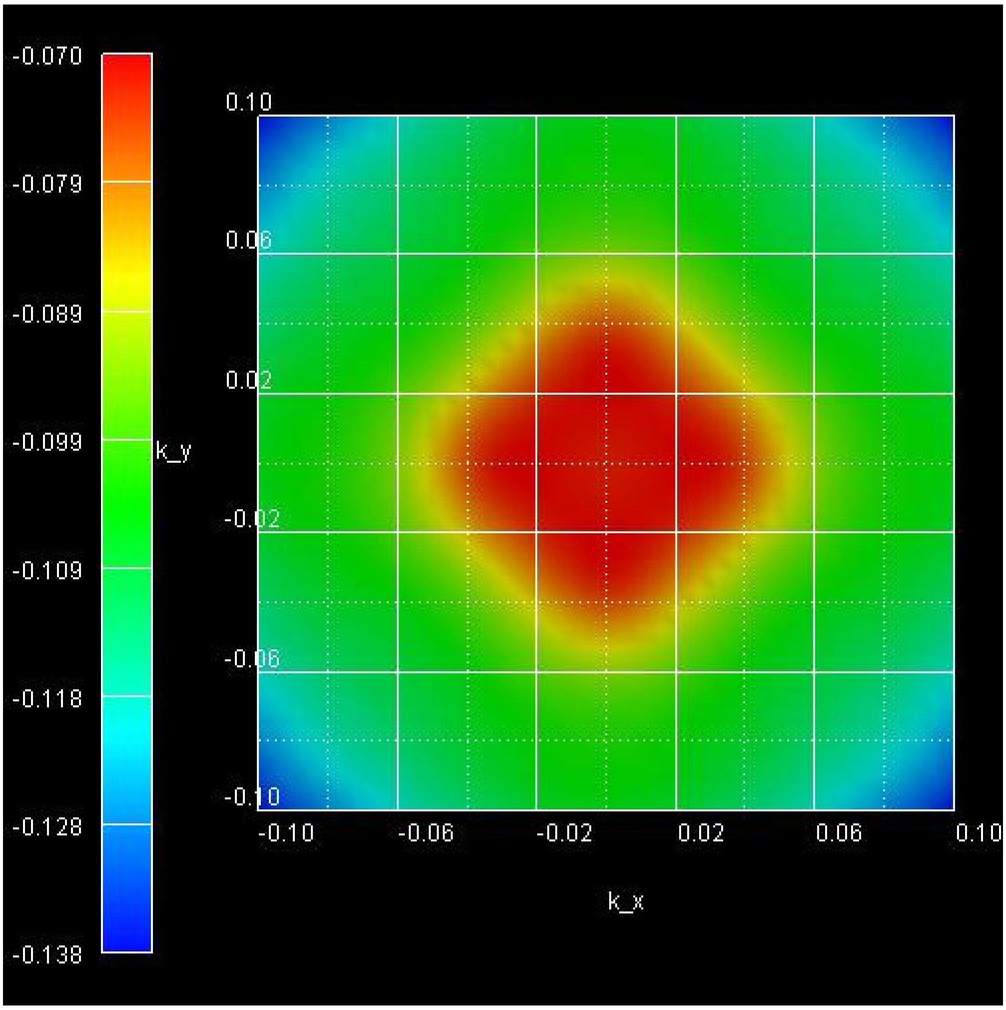


Figure 4.9.4.3: Subband 2 (eigenvalue 3 and 4)

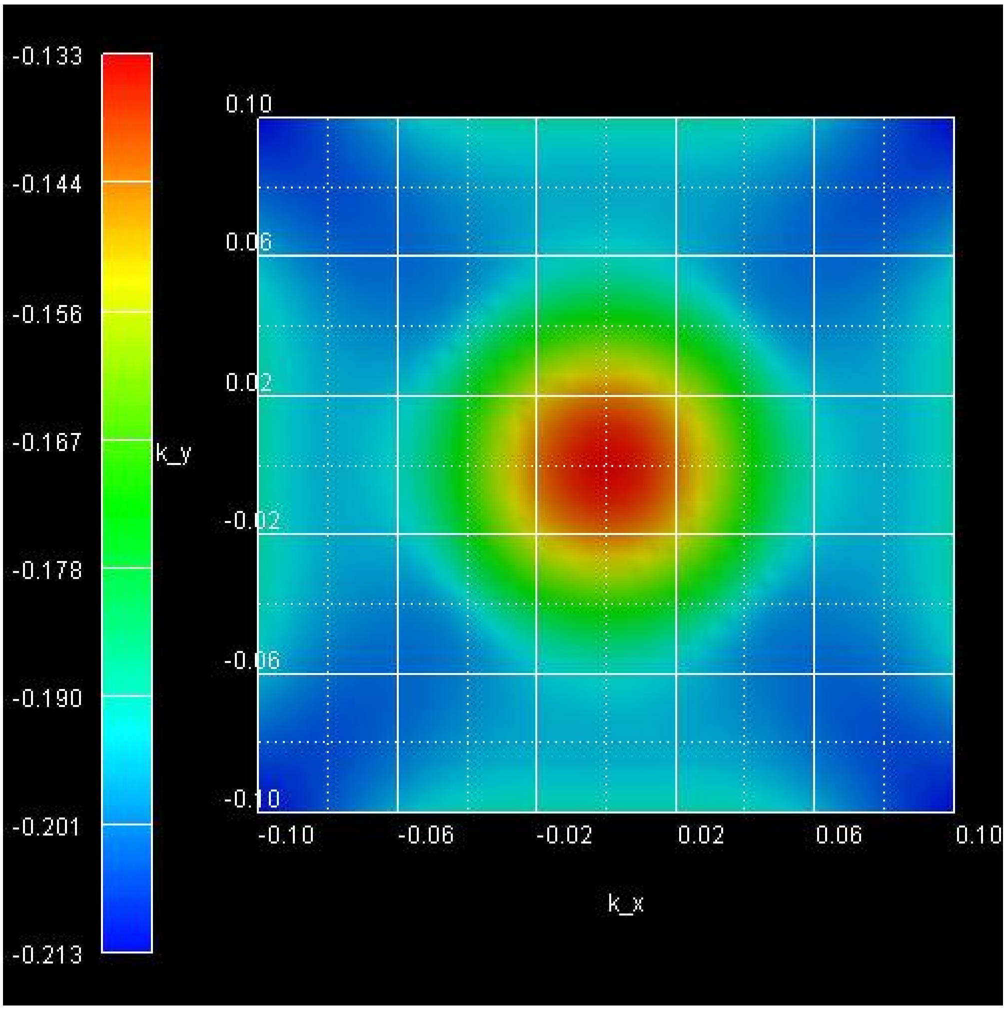


Figure 4.9.4.4: Subband 3 (eigenvalue 5 and 6)

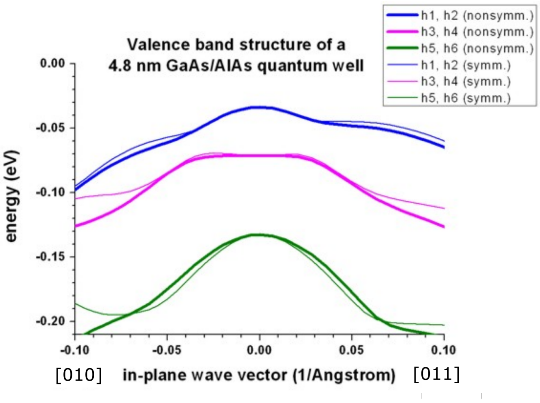


Figure 4.9.4.5: Calculated valence band structure of a $GaAs/AlAs$ QW.

b) Tensely strained $GaSb/AlSb$ quantum wells

Input file: *1Dwell_GaSb_AlSb_nnp.in*

Figure 4.9.4.6 reproduces Fig. 2 of *[FranceschiJancuBeltram1999]* very well. It is a tensely strained 5.1 nm $GaSb$ quantum well embedded between unstrained $AlSb$ barriers. The biaxial strain is 0.65 % and breaks the degeneracy of the bulk heavy and light hole band edge. Now the light hole band edge lies above the heavy hole band edge.

The figure shows that the first two subbands are nearly degenerate at the Brillouin zone center and show strong coupling.

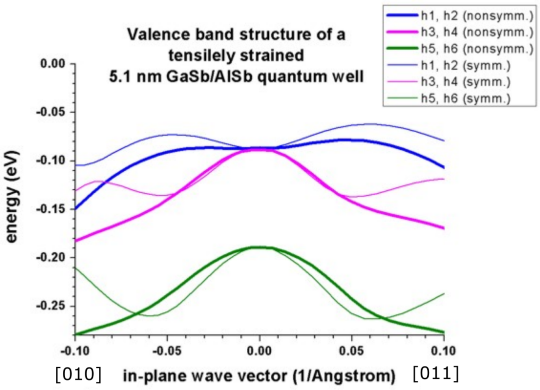


Figure 4.9.4.6: Calculated valence band structure of a tensely strained $GaSb/AlSb$ QW.

A large discrepancy between the **non**symmetrized and the symmetrized k.p Hamiltonian can be seen. (See also the discussion in *[FranceschiJancuBeltram1999]* and their tight-binding results.)

c) Tensely strained $In_{0.43}Ga_{0.57}As/InP$ quantum wells

Input file: *1Dwell_InGaAs_InP_nnp.in*

The following figure reproduces Fig. 3 of *[FranceschiJancuBeltram1999]* very well. It is a tensely strained 5.7 nm $In_{0.43}Ga_{0.57}As$ quantum well embedded between unstrained InP barriers. The biaxial strain is 0.73 % and breaks the degeneracy of the bulk heavy and light hole band edge. Now the light hole band edge lies above the heavy hole band edge.

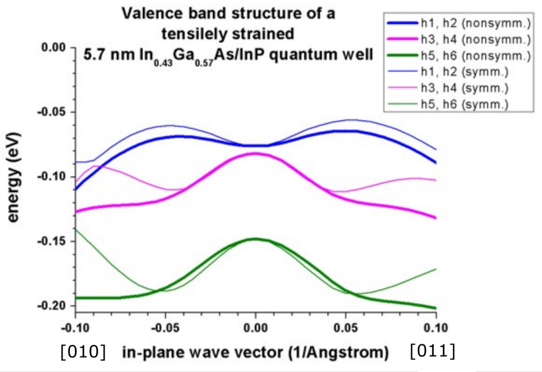


Figure 4.9.4.7: Calculated valence band structure of a tensely strained $In_{0.43}Ga_{0.57}As/InP$ QW.

Again, a large discrepancy between the **non**symmetrized and the symmetrized k.p Hamiltonian can be seen. (See also the discussion in *[FranceschiJancuBeltram1999]* and their tight-binding results.)

d) Strained $In_{0.2}Ga_{0.8}As/GaAs$ quantum well

Input files:

- *1DIn20Ga80AsQW_75nm_sg.in*
- *1DIn20Ga80AsQW_75nm_kp.in*
- *1DIn20Ga80AsQW_75nm_kp_dispersion.in*

These input files have been used for Fig. 8 in the following paper: *[Holleitner2007]*.

1DIn20Ga80AsQW_75nm_sg.in

A 7.5 nm $In_{0.2}Ga_{0.8}As$ quantum well is sandwiched between two $GaAs$ layers. The quantum well is grown pseudomorphically on a $GaAs$ substrate and is thus strained compressively with respect to the $GaAs$ substrate.

The $GaAs$ is n-type doped with Si with a concentration of $3 \cdot 10^{17}$ cm^{-3} in the regions between x = 50 nm and x = 80 nm and between x = 127.5 nm and x = 137.5 nm.

Consequently, we first have to solve the single-band Schrödinger equation together with the Poisson equation self-consistently, in order to obtain the electrostatic potential. The electron ground state is below the Fermi level.

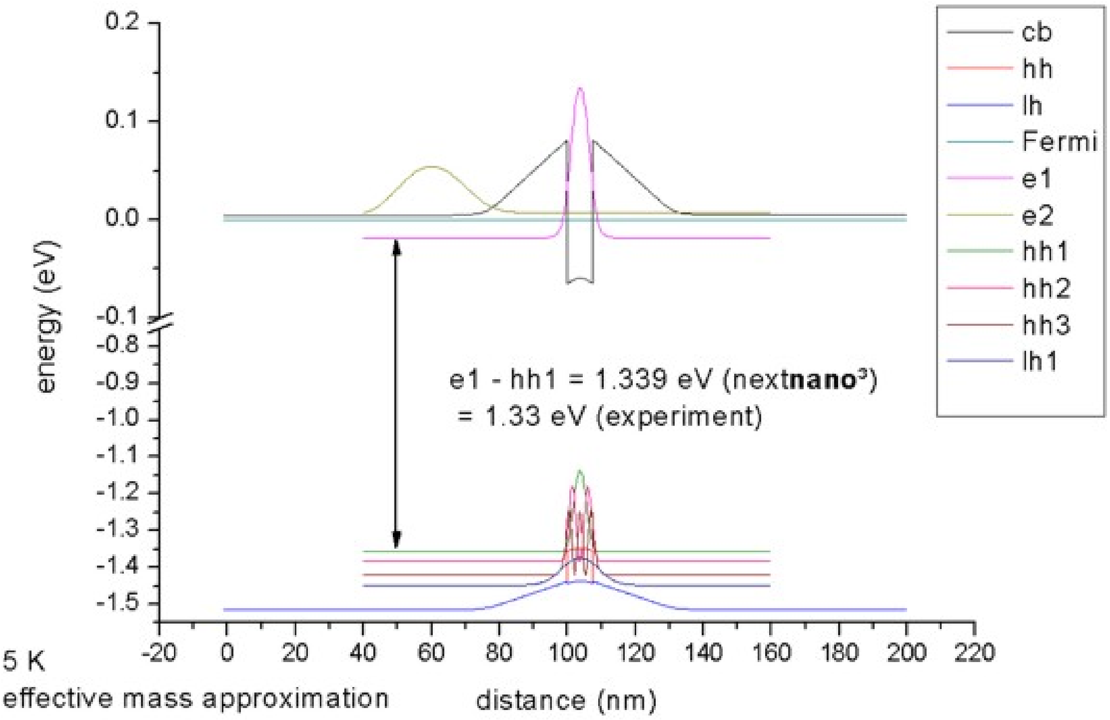


Figure 4.9.4.8: Calculated band edge profile of a compressively strained $In_{0.2}Ga_{0.8}As/GaAs$ QW (single-band Schrödinger equation).

1DIn20Ga80AsQW_75nm_kp.in

The calculated electrostatic potential is read in and then the 8-band k.p equation is solved to get the eigenstates for $k_{||}$ = 0. The calculated transition energy between the ground state electron and the ground state (heavy) hole is 1.340 eV. (Note: The exciton correction has not been considered and is of the order 4 meV.)

For $k_{||} = 0$, the three highest hole states have heavy hole character whereas the forth state has light hole character. No further states are confined. The split-off hole band edge is far away from the heavy and light hole band edges (~ 0.3 eV).

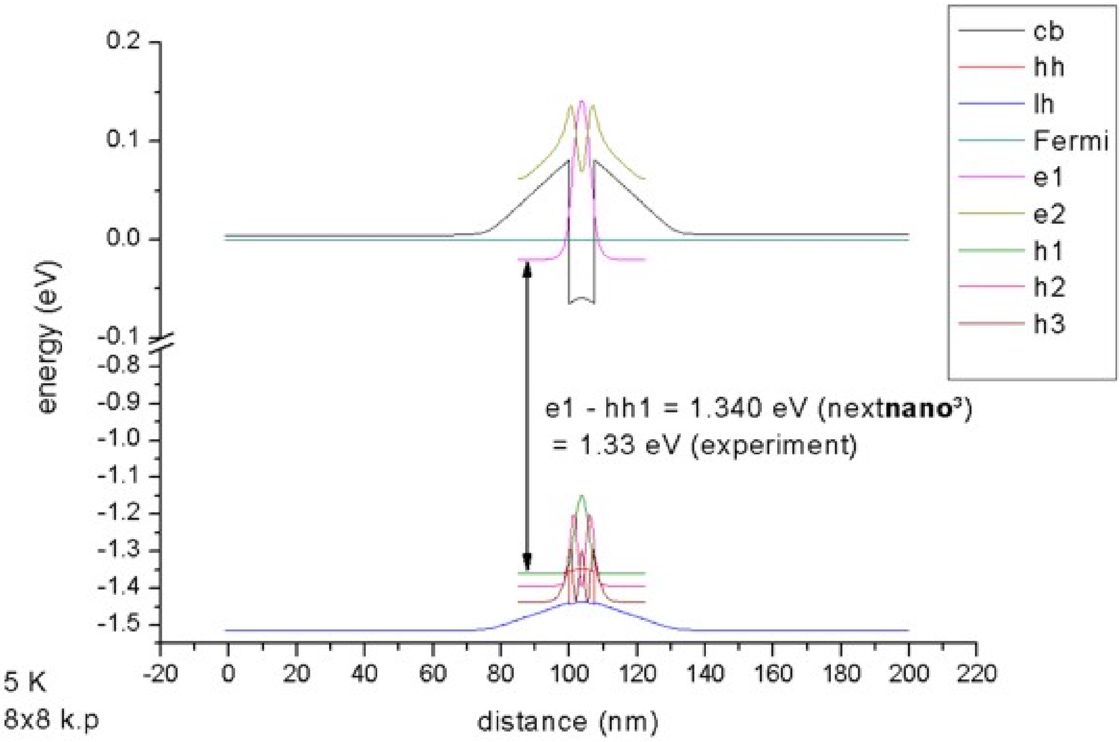


Figure 4.9.4.9: Calculated band edge profile of a compressively strained $In_{0.2}Ga_{0.8}As/GaAs$ QW (8-band k.p).

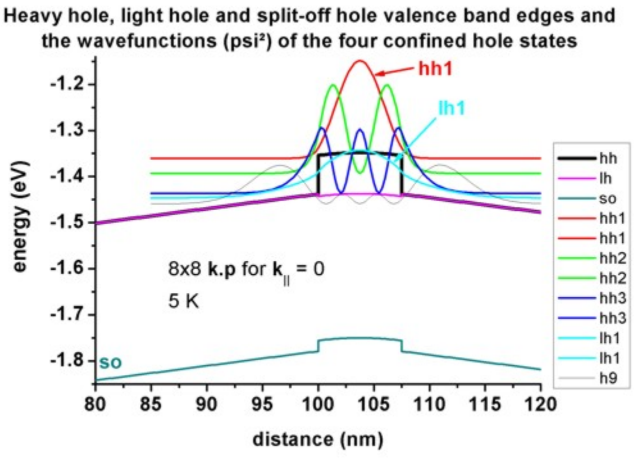


Figure 4.9.4.10: Calculated valence band structure and lowest hole states of a compressively strained $In_{0.2}Ga_{0.8}As/GaAs$ QW (8-band k.p).

1DIn20Ga80AsQW_75nm_kp_dispersion.in

We read in the electrostatic potential again and calculate the 8-band k.p dispersion for $k_{||} \neq 0$. This time the calculation is more time-consuming as the Schrödinger equation has to be solved for 250 different $k_{||}$ points, i.e. the CPU time is 250 times larger than for $k_|$| = 0 only.

For $|k_{||}|$ <= 0.02 1/Angstrom, the directions [10] and [11] are practically identical for the uppermost hole level, see Figure 4.9.4.11.

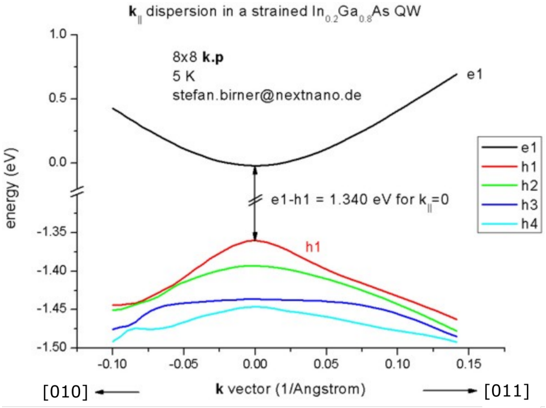


Figure 4.9.4.11: Calculated subband dispersions in $In_{0.2}Ga_{0.8}As/GaAs$ QW.

Figure 4.9.4.12 shows the $k_{||}$ dispersion of the highest hole state (h1). The x axis shows the kx value between -0.10 [1/Angstrom] and 0.10 [1/Angstrom], the y axis shows k_y. The maximum energy of the hole state occurs at -1.3603 eV at (k_x, k_y) = (0, 0), i.e. in the center of the figure (Gamma point).

Last update: nnnn/nn/nn

4.9.5 k.p dispersion of an unstrained GaN QW embedded between strained AlGaN layers

Input files:

- *1DGaN_AlGaN_QW_k_zero_nnp.in*
- *1DGaN_AlGaN_QW_k_parallel_nnp.in*
- *1DGaN_AlGaN_QW_k_zero_10m10_nnp.in*
- *1DGaN_AlGaN_QW_k_parallel_10m10_nnp.in*
- *1DGaN_AlGaN_QW_k_parallel_10m10_whole_nnp.in*

Scope:

In this tutorial we aim to reproduce results of *[Park2000]*. The material parameters are taken from *[ParkChunag2000]*, except those listed in Table 1 of *[Park2000]*.

[0001] growth direction

Calculation of electron and hole energies and wave functions for $k_{||}$ = 0

Input file: *1DGaN_AlGaN_QW_k_zero_nnp.in*

The structure consists of a 3 nm unstrained GaN quantum well, embedded between 8.4 nm strained $Al_{0.2}Ga_{0.8}N$ barriers. The $AlGaN$ layers are strained with respect to the GaN substrate. The GaN quantum well is assumed to be unstrained.

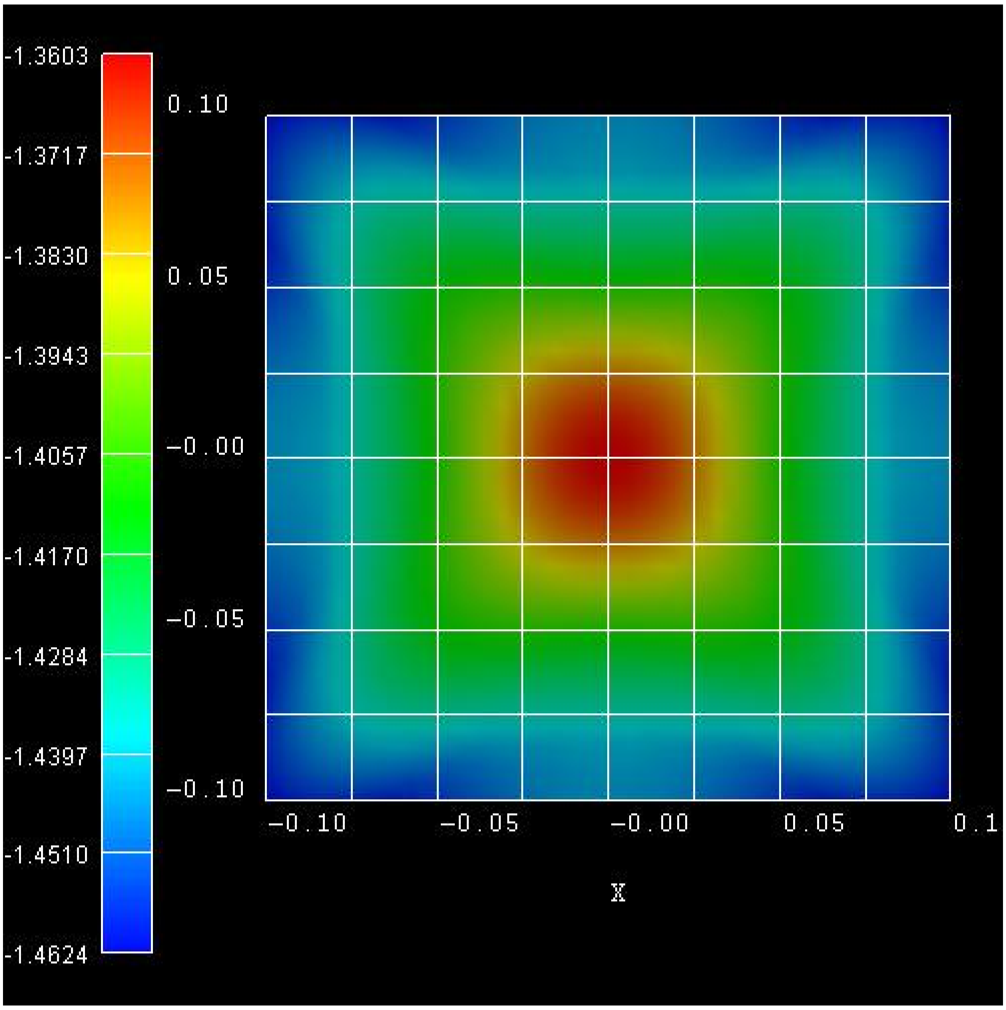


Figure 4.9.4.12: Calculated dispersion of h1 state in a $In_{0.2}Ga_{0.8}As/GaAs$ QW.

The structure is modeled as a superlattice (or multi quantum well, MQW), i.e. we apply periodic boundary conditions to the Poisson equation.

The growth direction is along the hexagonal axis, i.e. along [0001].

Conduction and valence band profile

Figure 4.9.5.1 shows the conduction and valence (heavy hole, light hole and crystal-field split-off hole) band edges of our structure, including the effects of strain, piezo- and pyroelectricity. The ground state electron and the ground state heavy hole wave functions (Ψ^2) are shown. Due to the built-in piezo- and pyroelectric fields, the electron wave function are shifted to the right and the hole wave function to the left (Quantum Confined Stark Effect, QCSE)

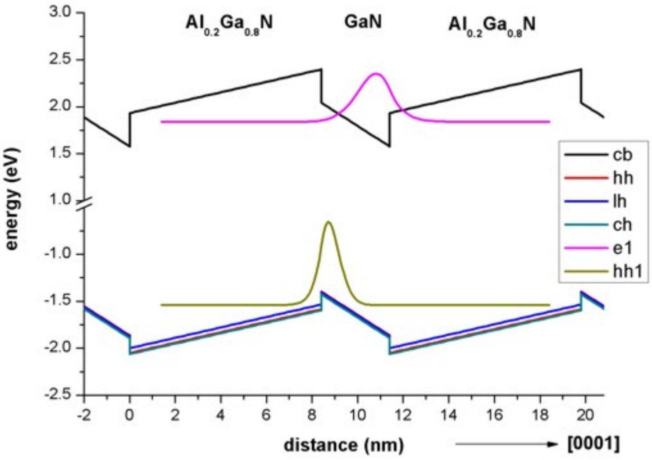


Figure 4.9.5.1: Calculated band edge profile.

Strain

The strain inside the GaN quantum well layer is zero. The tensile strain in the $Al_{0.2}Ga_{0.8}N$ barriers has been calculated to be

$$e_{xx} = e_{yy} = \frac{a_{\text{substrate}} - a}{a} = 0.486.$$

[Park2000] gives a value of 0.484.

The output of the strain tensor can be found in this file: *strain\strain_crystal.dat*

Piezoelectric polarization

The piezoelectric polarization for the [0001] growth direction is zero inside the GaN QW, because the strain is zero in the QW. In the $Al_{0.2}Ga_{0.8}N$ barriers, the piezoelectric polarization has been calculated to be 0.0081 C/m^2 in agreement with Fig. 1(a) of *[Park2000]* for angle θ = 0. The resulting piezoelectric polarization

- at the $Al_{0.2}Ga_{0.8}N/GaN$ interface -0.0081 C/m^2 and
- at the $GaN/Al_{0.2}Ga_{0.8}N$ interface is 0.0081 C/m^2.

Pyroelectric polarization

The pyroelectric polarization for the [0001] growth direction is -0.029 C/m^2 inside the GaN QW. In the $Al_{0.2}Ga_{0.8}N$ barriers, the pyroelectric polarization has been calculated to be -0.0394 C/m^2. The resulting pyroelectric polarization

- at the $Al_{0.2}Ga_{0.8}N/GaN$ interface is -0.0104 C/m^2 and
- at the $GaN/Al_{0.2}Ga_{0.8}N$ interface is 0.0104 C/m^2.

These results are in excellent agreement with Fig. 1(a) of *[Park2000]* for angle θ = 0.

Poisson equation

Solving the Poisson equation with periodic boundary conditions (to mimic the superlattice) leads to the following electric fields: Inside the GaN QW the electric field has been calculated to be -1.551 MV/cm. *[Park2000]* reports an electric field of -1.55 MV/cm inside the QW. The electric field in the $AlGaN$ barrier has been found to be 0.554 MV/cm.

The output of the electrostatic potential (units [V]) and the electric field (units [kV/cm]) can be found in these files:

- *bias_00000\potential*
- *bias_00000\electric_filed.dat*

Schrödinger equation

Figure 4.9.5.2 shows the electron and hole wave functions (Ψ^2) of the $GaN/AlGaN$ structure for $k_{||}$ = 0. The heavy and light hole wave functions are very similar in shape.

In agreement with *[Park2000]*, we calculated the electron levels within the single-band effective mass approximation and the hole levels within the 6-band k.p approximation.

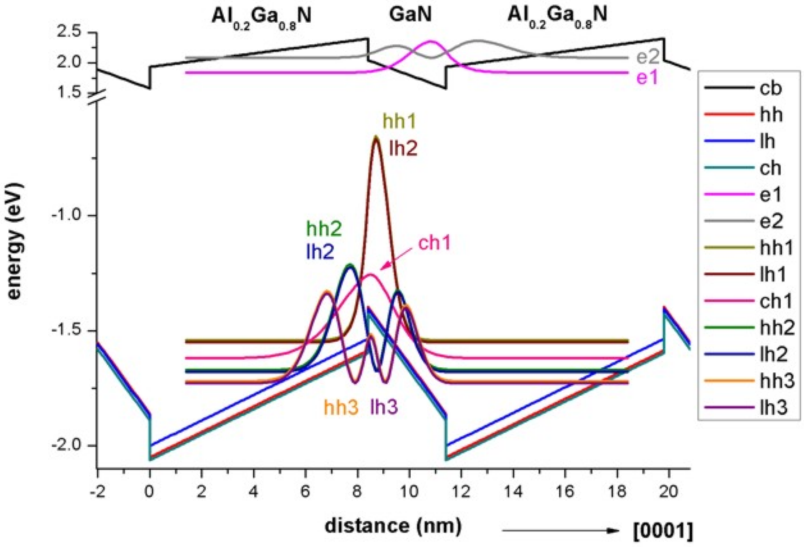


Figure 4.9.5.2: Calculated wave functions of lowest eigenstates.

$k_{||}$ dispersion: Calculation of the electron and hole energies and wave functions for $k_{||} \neq 0$.

Input file: *1DGaN_AlGaN_QW_k_parallel_nnp.in*

The grid has a spacing of 0.1 nm leading to a sparse matrix of dimension 1050 which has to be solved for each $k_{||}$ point for the eigenvalues (and wave functions).

We chose as input:

```
calculate_dispersion{
    num_points = 1849 # This corresponds to 1849 k|| points in the 2D (kx,ky) plane,␣
↪i.e. (2 * 21 + 1) * (2 * 21 + 1) = 1849.
}
```

Due to symmetry arguments, we solved the Schrödinger equation only for the $k_{||}$ points along the line (k_x > 0, k_y = 0), i.e. we had to solve the Schrödinger equation 22 times (i.e. to calculate the eigenvalues of a 1050 x 1050 matrix 22 times).

The energy dispersion $E(k_{||}) = E(k_y, k_z)$ displayed in Figure 4.9.5.3 is contained in this folder: *bias_00000\Quantum\Dispersion*

Because our quantum well is not symmetric (due to the piezo- and pyroelectric fields), the eigenvalues for spin up and spin down are not degenerate anymore. They are only degenerate at $k_{||}$ = 0. This lifting of the so-called Kramer's degeneracy in the in-plane dispersion relations is because of the field-induced asymmetry. In Fig. 3 (a) of

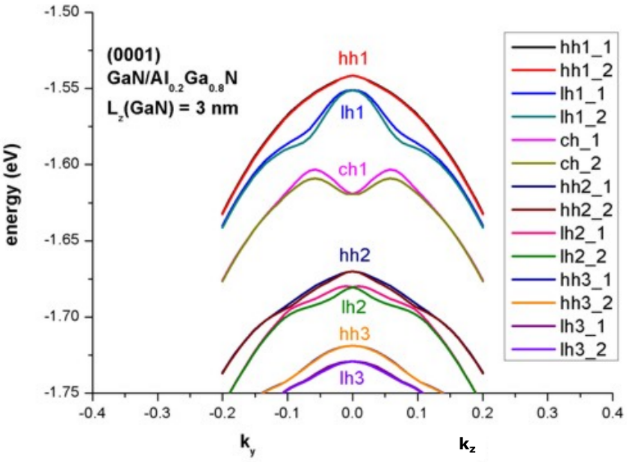


Figure 4.9.5.3: Calculated energy dispersion $E(k_{||}) = E(k_y, k_z)$.

[Park2000] only the spin-up eigenstates are plotted because the splitting of the Kramer's degeneracy was assumed to be very small.

[10-10] growth direction (m-plane)

Input file: *1DGaN_AlGaN_QW_k_zero_10m10_nnp.in*

If one grows the quantum well along the [10-10] growth direction, then the pyroelectric and piezoelectric fields along the [10-10] direction are zero. In this case, the quantum well (i.e. the conduction and valence band profile) is symmetric.

Figure 4.9.5.4 shows the electron and hole wave functions (ψ^2) of the (10-10)-oriented $GaN/AlGaNQW$ for $k_{||} = 0$. Obviously, the interband transition matrix elements (i.e. the probability for electron-hole transitions) are much larger than for the [0001] growth direction.

In agreement with *[Park2000]*, we calculated the electron levels within the single-band effective mass approximation and the hole levels within the 6-band k.p approximation.

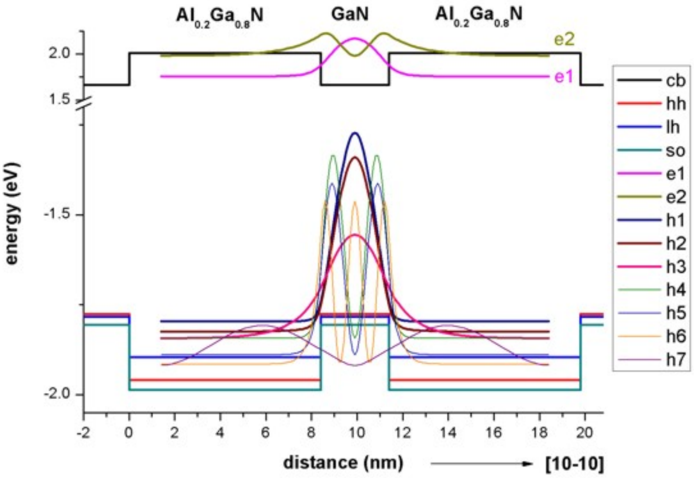


Figure 4.9.5.4: Calculated wave functions of lowest eigenstates.

$k_{||}$ dispersion: Calculation of the electron and hole energies and wave functions for $k_{||} \neq 0$.

Input file: *1DGaN_AlGaN_QW_k_parallel_10m10_nnp.in*

Due to the symmetry of the quantum well, we expect degenerate eigenvalues for the in-plane dispersion relation (Kramer's degeneracy). Our results, depicted in Figure 4.9.5.5, compare well with Fig. 3(c) of *[Park2000]*.

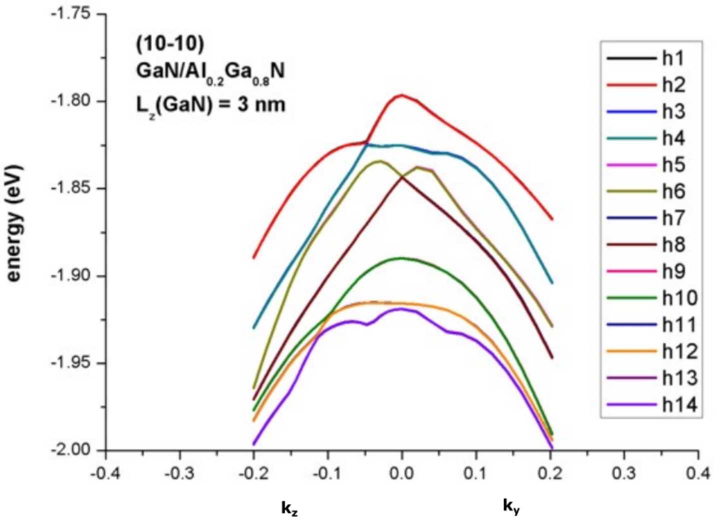


Figure 4.9.5.5: Calculated energy dispersion $E(k_{||}) = E(k_y, k_z)$.

Last update: nnnn/nn/nn

4.9.6 Energy dispersion of a cylindrical shaped GaN nanowire

Input files:

- *2DGaN_nanowire_nnp.in*

Scope:

In this tutorial we study the electron and hole energy levels of a two-dimensional freestanding GaN nanowire of cylindrical shape. We aim to reproduce results of *[ZhangXia2006]*.

Output files:

- *bias_00000\Quantum\Dispersions\dispersion_quantum_region_kp6_path_as_in_input_file.dat*
- *bias_00000\Quantum\probabilities_quantum_region_kp6_00000.fld*

Introduction

We assume a cylindrical shaped GaN nanowire (wurtzite structure) that has a radius of 2 nm with infinite barriers so that the wave functions are zero at the nanowire boundary. This assumption is consistent to *[ZhangXia2006]*. The GaN nanowire is shown in red in Figure 4.9.6.1. The GaN nanowire is discretized on a mesh with a grid resolution of 0.05 nm.

Electrons

Figure 4.9.6.2 shows the electron states as a function of k of the GaN nanowire. It is in excellent agreement with Fig. 1 of *[ZhangXia2006]*. All states are two-fold degenerate due to spin. In addition, the 2nd and 3rd state are degenerate, as well as the 4th and the 5th. The ground state has quantum number L = 0. For $L \neq 0$, the states are degenerate due to $L = \pm$ 1. The energy levels increase with increasing k as quadratic terms of k (parabolic dispersion).

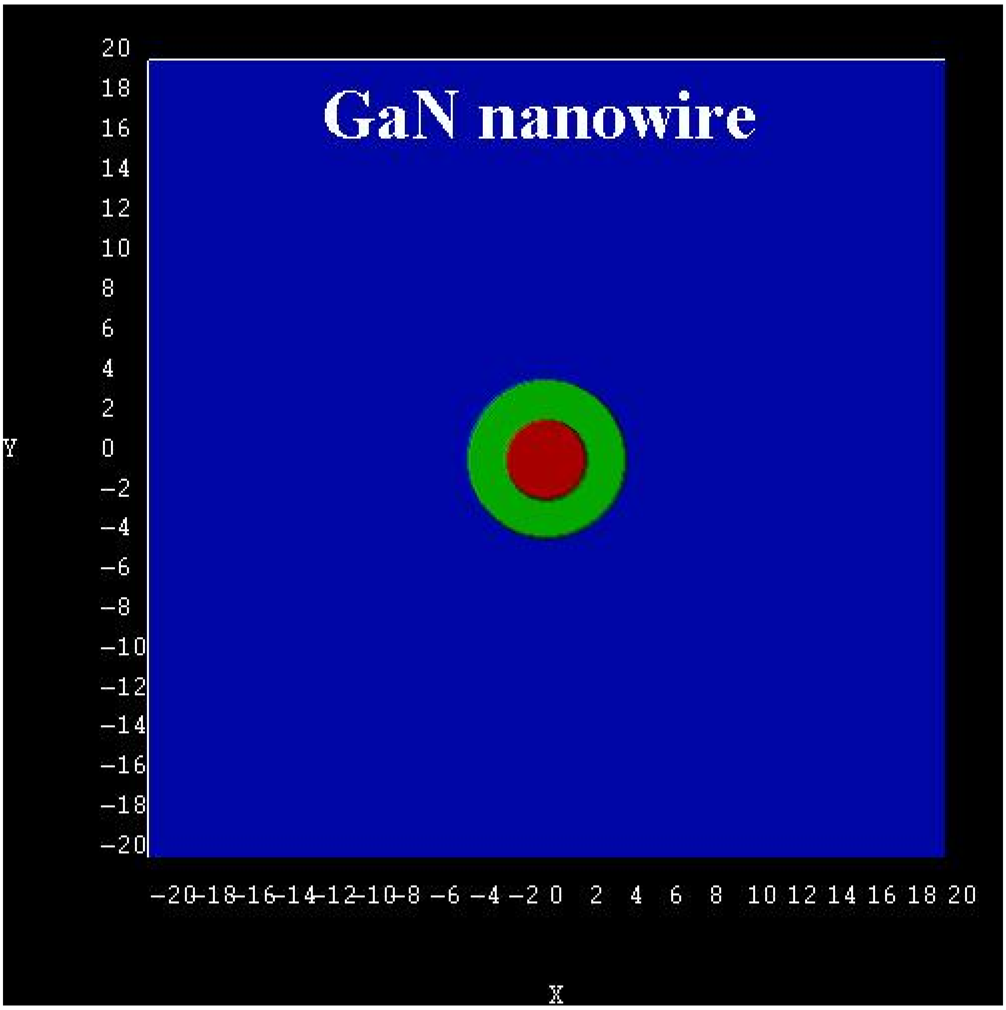


Figure 4.9.6.1: GaN nanowire structure.

Technical details: We calculated the electron energy levels at $k_x = 0$ with *nextnano++* numerically by solving the 2D single-band Schrödinger equation. The parabolic dispersion for $k_x \neq 0$ has been calculated analytically using

$$E_i(k_x) = E_i + \frac{\hbar^2 k_x^2}{2m^*}$$

i.e. not with *nextnano++*. The eigenvalues for $k_x = 0$ can be found in the following file: *bias_00000\Quantum\energy_spectrum_quantum_region_Gamma_00000.dat*

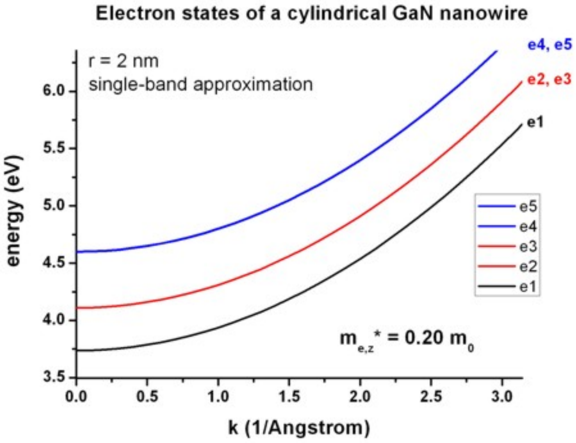


Figure 4.9.6.2: Energy dispersion $E(k)$ of electron states.

The wave function (Ψ^2) of the electron ground state at k = 0 is shown in Figure 4.9.6.3.

Holes

The following figures show the ground state wave function (psi^2) of the hole (Figure 4.9.6.4) and the 1st excited hole state (Figure 4.9.6.5) as calculated within the 6-band k.p approximation at k = 0. According to the above cited paper, the right figure would be the ground state for GaN nanowires with a radius r < 0.7 nm. Because our nanowire has a radius of 2 nm, the ground state wave function is according to the left figure. Following *[ZhangXia2006]*, this means that the probability for electron-hole transitions (e1 - h1) is not very high at a radius of 2 nm because the wave functions do not have much overlap and the electron ground state has L = 0, whereas the hole ground state has $L = \pm$ 1 (dark exciton effect).

Figure 4.9.6.6 shows the hole states as a function of k of the GaN nanowire as calculated with 6-band k.p theory. It corresponds to Fig. 2 and Fig. 3 of the paper of *[ZhangXia2006]*. Note that the authors assumed the hole energies to be positive. All states are two-fold degenerate, i.e. h1 = h2, h3 = h4, h5 = h6, …

The *nextnano++* results are a bit different. Several reasons could explain this:

- The authors use the "cylindrical approximation" for the k.p parameters. However, the parameters that they are citing are not exactly cylindrical. Thus, for our calculations, we had to employ the parameters that they were citing (without making use of the cylindrical approximation).
- Our cylinder does not have exactly cylindrical symmetry. It is approximated to be cylindrical by a rectangular grid with a grid resolution of 0.05 nm.
- For the k.p parameters that are given in *[ZhangXia2006]*, it must hold that

  $$A_5 = \frac{1}{2}(L_1 - M_1)$$

  is equal to

  $$A_5 = \frac{1}{2}N_1.$$

  However, they differ by 0.0064.

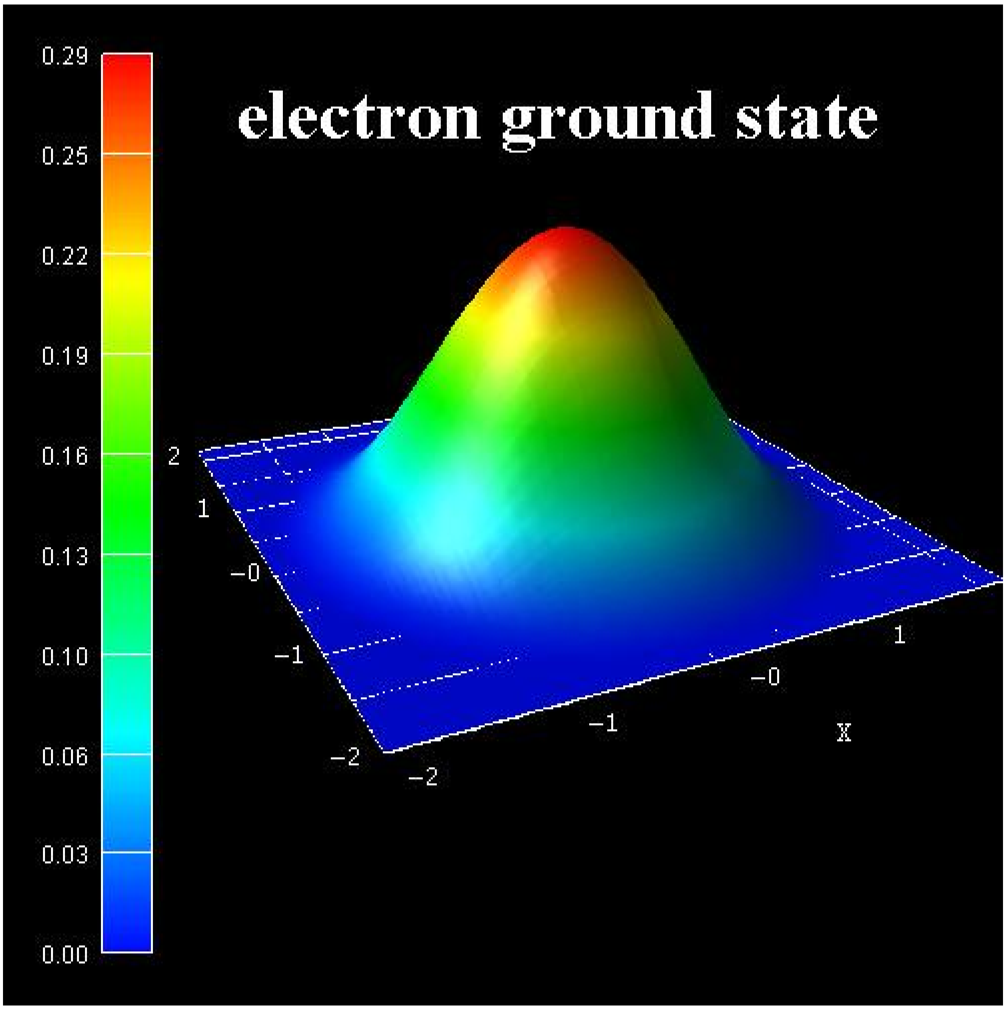


Figure 4.9.6.3: Ψ^2 of electron ground state.

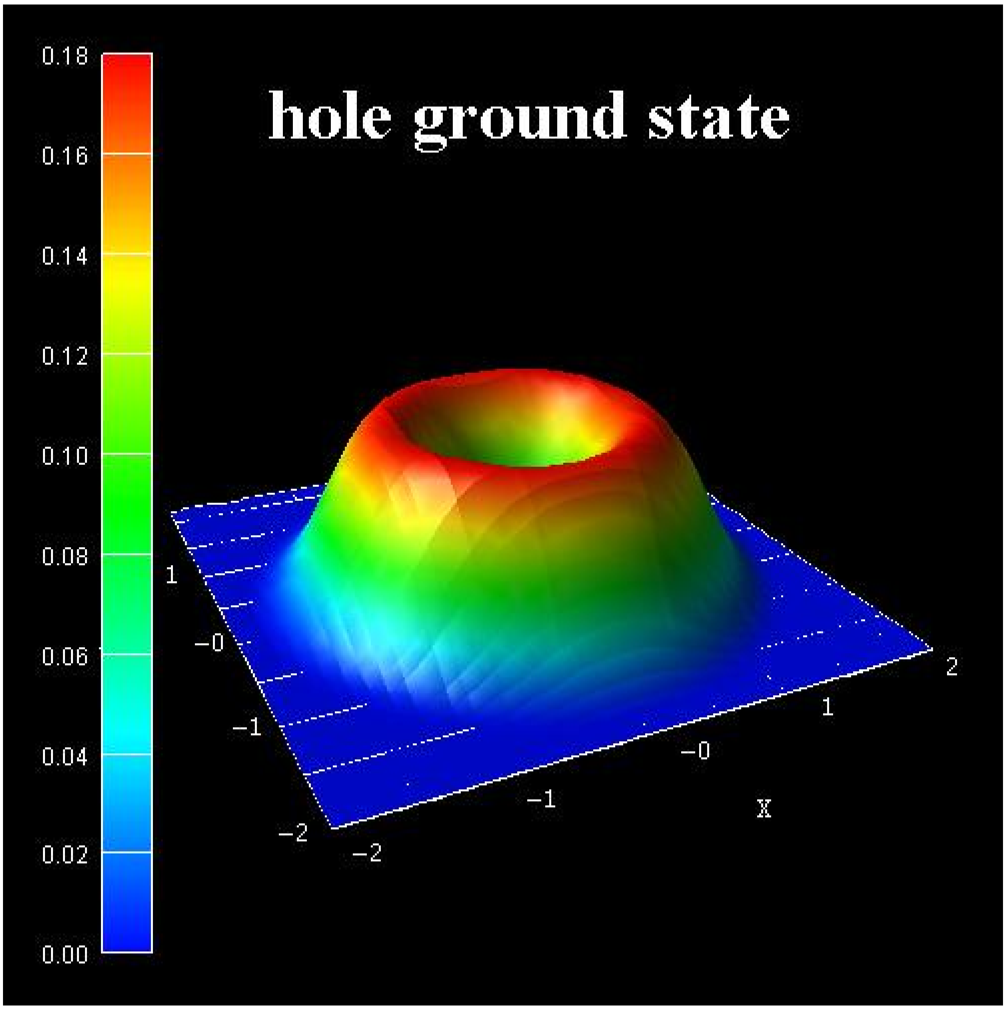


Figure 4.9.6.4: Ψ^2 of hole ground state.

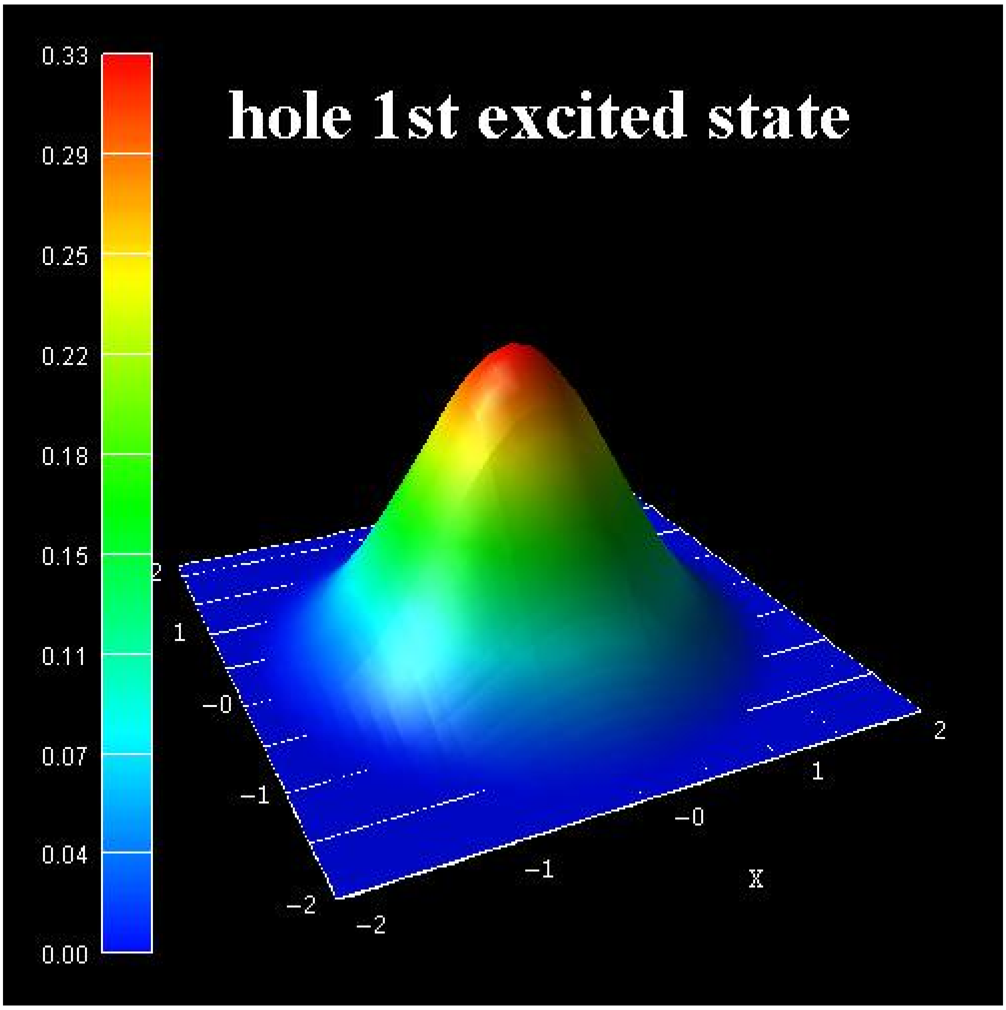


Figure 4.9.6.5: Ψ^2 of 1st excited hole state.

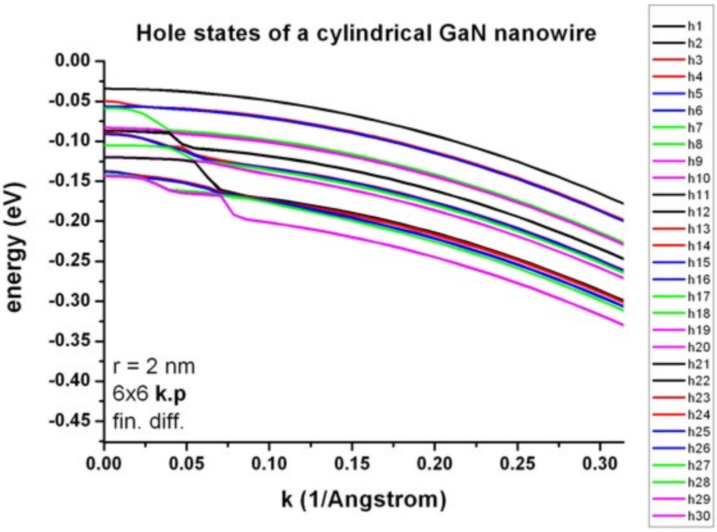


Figure 4.9.6.6: Energy dispersion $E(k)$ of hole states.

The data that has been plotted in Figure 4.9.6.6 is contained in this file: *bias_00000\Quantum\Dispersions\dispersion_quantum_region_kp6_lines_type1_00-1_001.dat*

In the input file, one can specify the number of $k_{||}$ = k_x points.

```
quantum{
    region{
        ...
        kp_6band{
            dispersion{
                line{
                    name = "lines"
                    spacing = 2 * $k_max / $number_of_k_parallel_points # Unit: [nm-
↪1].
                    k_max = $k_max                                      # specifies a␣
↪maximum absolute value (radius) for the k-vector. Unit: [nm-1].
                }
            }
        }
    }
}
```

Note that e.g. `$number_of_k_parallel_points = 41` means 14 minutes CPU time (Intel i5, 2015). If one uses only 1, then one only calculates the k.p states at k_x = 0 and the calculation takes less than a minute.

[ZhangXia2006] used the following 6-band k.p parameters:

- Crystal field and spin-orbit splitting energies:

$$\Delta_{cr} = 0.021$$

$$\Delta_{so} = 0.018$$

- "Dresselhaus" parameters:

[ZhangXia2006]	*nextnano++*	
L = 6.3055	L_1 = -6.3055 - 1 = -7.3055	⇒ The definition of the k.p Hamiltonians differs.
M = 0.1956	M_1 = -0.1956 - 1 = -1.1956	⇒ The definition of the k.p Hamiltonians differs.
N = 0.3813	M_2 = -0.3813 - 1 = -1.3813	⇒ The definition of the k.p Hamiltonians differs.
R = 6.1227	N_1 = -0.3813 - 1 = -6.1227	
S = 0.4335	M_3 = -0.4335 - 1 = -1.4335	⇒ The definition of the k.p Hamiltonians differs
T = 7.3308	L_2 = -7.3308 - 1 = -8.3308	⇒ The definition of the k.p Hamiltonians differs
Q = 4.0200	N_2 = -4.0200	

- Conversion to "Luttinger" parameters:

A_1 = L_2 + 1 = -8.3308 + 1 = -7.3308	⇒ The definition of the k.p Hamiltonians differs.
A_2 = M_3 + 1 = -1.4335 + 1 = -0.4335	⇒ The definition of the k.p Hamiltonians differs.
A_3 = M_2 - L_2 = -0.3813 + 7.3308 = 6.9495	
A_4 = 1/2 (L_1 + M_1 - 2 M_3) = -2.81705	
A_5 = 1/2 (L_1 - M_1) = -3.05495	⇒ inconsistent to -3.06135
A_5 = 1/2 (N_1) = -3.06135	⇒ inconsistent to -3.05495
A_6 = $\sqrt{2}/2 N_2$ = -2.84256926	

Cylindrical (axial) approximation:

- *[ZhangXia2006]*:

$$L - M - R = 0$$

- *nextnano++*:

$$L_1 - M_1 - N_1 = 0$$

$$\Rightarrow (A_2 + A_4 + A_5 - 1) - (A_2 + A_4 - A_5 - 1) - 2A_5 = 0.$$

$$A_1 - A_2 = -A_3 = 2A_4$$

$$A_3 + 4A_5 = \sqrt{2}A_6$$

$$\Delta_2 = \Delta_3 = \frac{1}{3}\Delta_{so}$$

Last update: nnnn/nn/nn

4.9.7 Electronic band structure of 2DHG in Silicon inversion layers under pseudomorphic strain | 1D

Input files in *examples\electronic_band_structures*:

- *band-structure-kp_inv-layer-Si_Fischetti_2003_1D_(001)_nnp.in*
- *band-structure-kp_inv-layer-Si_Fischetti_2003_1D_(011)_nnp*
- *band-structure-kp_inv-layer-Si_Fischetti_2003_1D_(111)_nnp*
- *band-structure-kp_inv-layer-Si_Fischetti_2003_1D_(001)_tensile_nnp*
- *band-structure-kp_inv-layer-Si_Fischetti_2003_1D_(001)_compressive_nnp*

Relevant output files:

- *bias_00000\Quantum\probabilities_shift_quantum_region_kp6_00000.dat*
- *bias_00000\Quantum\Dispersions\dispersion_quantum_region_kp6_XXXX.fld*

This tutorial aims to reproduce the figures presented in *[FischettiJAP2003]* Note that the crystal growth direction is along the z axis although it becomes along x axis in *nextnano*++.

Unstrained silicon inversion layer with (001) surface orientation

kpdispersion_Si_Fischetti_2003_1D_(001)_nnp is used in this section.

The figures below (Figure 4.9.7.1) aim to reproduce Fig.1(a), and Fig.4(a) of *[FischettiJAP2003]*.

Figure 4.9.7.1 (a) shows the valence edges (where the heavy and light hole band edges are degenerate) and the six lowest hole wave functions of a Si inversion layer (triangular-well approximation) for $\mathbf{k} = \mathbf{0}$ (i.e. $k_x = k_y = 0$) where the z axis is oriented along the [001] direction.

The potential energy of the well is given by

$$V(z') = eF_s z'$$

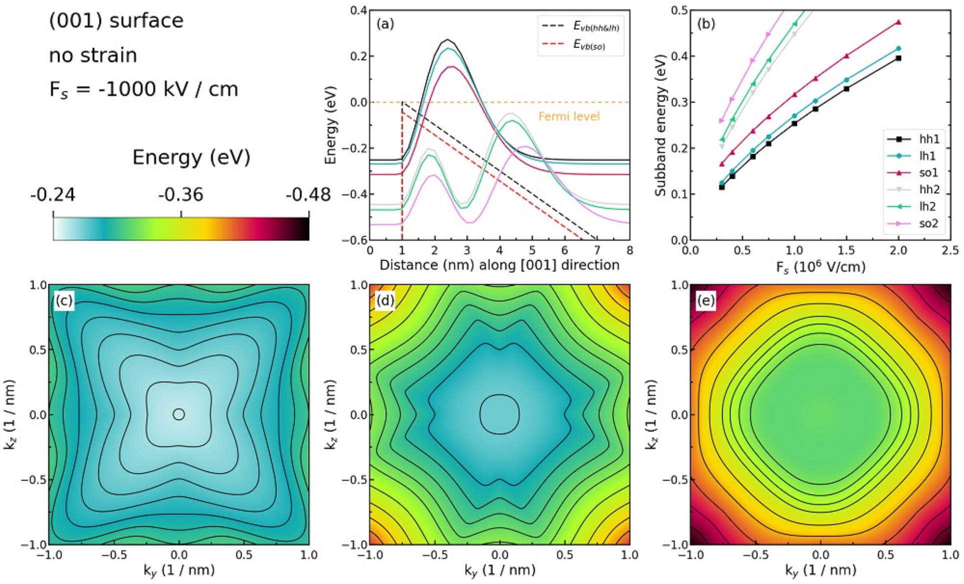


Figure 4.9.7.1: Some characteristics of the unstrained silicon inversion layer with (001) surface orientation.

where F_s is the surface field. In the figure, the electric field is $F_s = -1000\ \mathrm{kV/cm}$. Note that in the figure z is shifted by 1 nm: $V(z = 1) = V(z' = 0)$. One can clearly distinguish the holes by their characters (heavy-hole-like, light-hole-like, split-off-hole-like).

The energies of the six lowest-lying hole subbands for the (001) surface of the unstrained Si inversion layer are plotted as a function of the applied electric field (i.e. as a function of the triangular-well potential) in Figure 4.9.7.1 (b). The subband energies are measured from the surface potential. Our results are in excellent agreement with Fischetti's results. The symbols are calculated values, the connecting lines are added as a guide to the eye. The hole energies are taken to be positive, in contrast to the figure above (Figure 4.9.7.1 (a)) The labels of the curves (**hh**, **lh**, and **so**) are taken from Fischetti's paper. We do not perform this analysis within *nextnano++* because it is not important for quantitative results.

Figure 4.9.7.1 (c), (d), and (e) show the equienergy lines of the lowest lying **heavy hole**, **light hole**, and **split-off hole** subbands for the (001) surface of the unstrained silicon, respectively. Only one spin state is plotted for clarity. The x axis points along the [100], the y axis along the [010] direction of the crystal coordinate system.

The eigenvalues are spin-degenerate only at $\mathbf{k} = (k_x, k_y) = 0$, but differ for non-zero $\mathbf{k}$. The plots show the $k_{||}$ dispersions of the lowest **heavy hole (1st eigenstate, (c))**, the lowest **light hole (3rd eigenstate, (d))**, and the lowest **split-off hole (5th eigenstate, (e))**.

Unstrained silicon inversion layer with (011) surface orientation

kpdispersion_Si_Fischetti_2003_1D_(011)_nnp is used in this section.

The figures below (Figure 4.9.7.2) aim to reproduce Fig.2(a), and Fig.4(b) of *[FischettiJAP2003]*.

Figure 4.9.7.2 (a) shows the valence edges (where the heavy and light hole band edges are degenerate) and the six lowest hole wave functions of a Si inversion layer (triangular-well approximation) for $\mathbf{k} = \mathbf{0}$ (i.e. $k_x = k_y = 0$) where the z axis is oriented along the [011] direction. The potential energy of the well is given in the same way as in Figure 4.9.7.1, with a magnitude of $-1000\ \mathrm{kV/cm}$. One can clearly distinguish the holes by their characters (heavy-hole-like, light-hole-like, split-off-hole-like).

The energies of the six lowest-lying hole subbands for the (011) surface of the unstrained Si inversion layer are plotted as a function of the applied electric field in Figure 4.9.7.2 (b). The subband energies are measured from the surface potential. Our results are in excellent agreement with Fischetti's results. The plotting method is the same as in Figure 4.9.7.1 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 4.9.7.2 (c), (d), and (e) show the equienergy lines of the lowest lying **heavy hole**, **light hole**, and **split-off hole** subbands for the (011) surface of unstrained silicon, respectively. Only one spin state is plotted for clarity. The x axis points along the [100], the y axis along the $[01\bar{1}]$ direction of the crystal coordinate system.

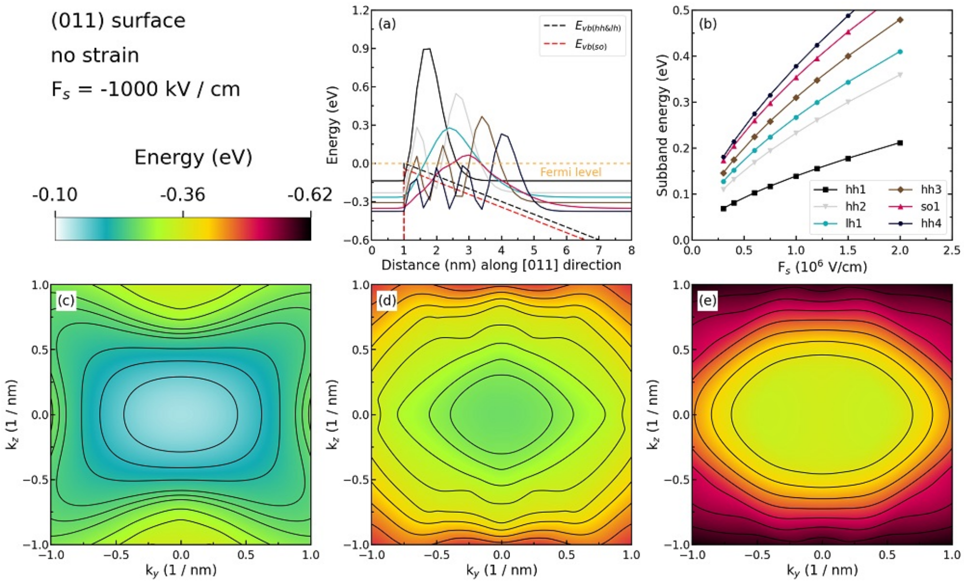


Figure 4.9.7.2: Some characteristics of the unstrained silicon inversion layer with (011) surface orientation.

The eigenvalues are spin-degenerate only at $\mathbf{k} = (k_x, k_y) = 0$, but differ for non-zero $\mathbf{k}$. The plots show the $k_{||}$ dispersions of the lowest **heavy hole (1st eigenstate, (c))**, the lowest **light hole (5th eigenstate, (d))**, and the lowest **split-off hole (9th eigenstate, (e))**.

Unstrained silicon inversion layer with (111) surface orientation

kpdispersion_Si_Fischetti_2003_1D_(111)_nnp is used in this section.

The figures below (Figure 4.9.7.3) aim to reproduce Fig.3(a), and Fig.4(c) of *[FischettiJAP2003]*.

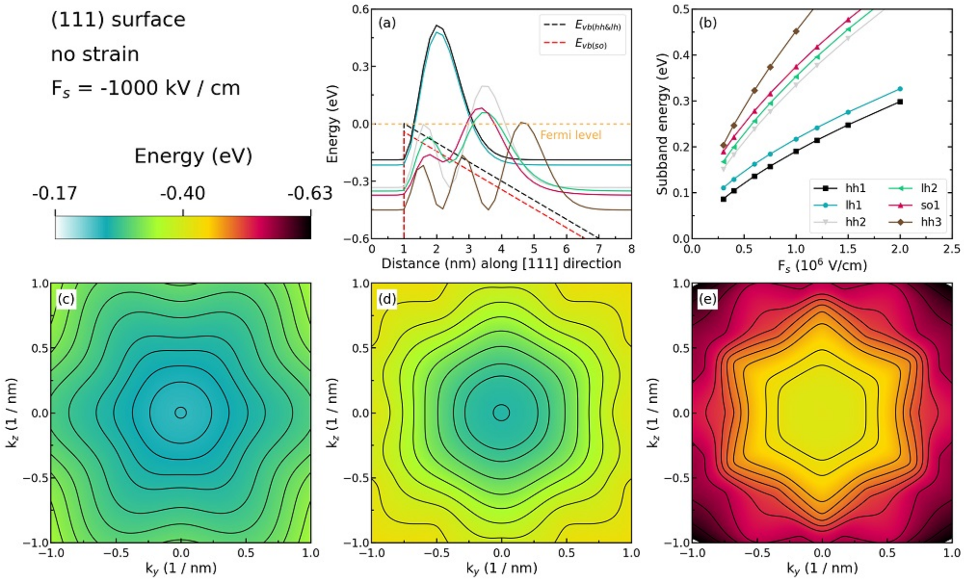


Figure 4.9.7.3: Some characteristics of the unstrained silicon inversion layer with (111) surface orientation.

Figure 4.9.7.3 (a) shows the valence edges (where the heavy and light hole band edges are degenerate) and the six lowest hole wave functions of a Si inversion layer (triangular-well approximation) for $\mathbf{k} = \mathbf{0}$ (i.e. $k_x = k_y = 0$) where the z axis is oriented along the [111] direction. The potential energy of the well is given in the same way as in Figure 4.9.7.1, with a magnitude of -1000 kV/cm.

The energies of the six lowest-lying hole subbands for the (111) surface of the unstrained Si inversion layer are plotted as a function of the applied electric field in Figure 4.9.7.3 (b). The subband energies are measured from the surface potential. Our results are in excellent agreement with Fischetti's results. The plotting method is the same as in Figure 4.9.7.1 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 4.9.7.3 (c), (d), and (e) show the equienergy lines of the lowest lying **heavy hole**, **light hole**, and **split-off hole** subbands for the (111) surface of unstrained silicon, respectively. Only one spin state is plotted for clarity. The x axis points along the $[11\bar{2}]$, the y axis along the $[\bar{1}10]$ direction of the crystal coordinate system.

The eigenvalues are spin-degenerate only at $\mathbf{k} = (k_x, k_y) = 0$, but differ for non-zero **k**. The plots show the $k_{||}$ dispersions of the lowest **heavy hole (1st eigenstate, (c))**, the lowest **light hole (3rd eigenstate, (d))**, and the lowest **split-off hole (9th eigenstate, (e))**.

1% tensilely strained silicon inversion layer with (001) surface orientation

kpdispersion_Si_Fischetti_2003_1D_(001)_tensile_nnp is used in this section.

The figures below (Figure 4.9.7.4) aim to reproduce Fig.5(a), and Fig.7(a) of *[FischettiJAP2003]*.

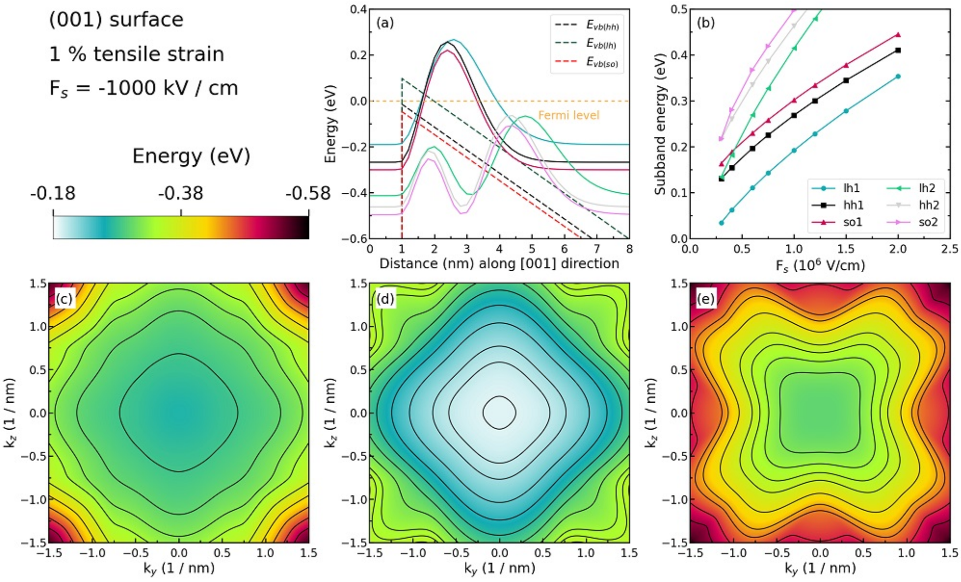


Figure 4.9.7.4: Some characteristics of the $1\ \%$ tensilely strained silicon inversion layer with (001) surface orientation.

Figure 4.9.7.4 (a) shows the valence edges (where the heavy and light hole band edges are no longer degenerate) and the six lowest hole wave functions of a tensilely strained Si inversion layer (triangular-well approximation) for $\mathbf{k} = \mathbf{0}$ (i.e. $k_x = k_y = 0$) where the z axis is oriented along the [001] direction. The tensile in-plane strain in the (x, y) plane is $1\ \%$. The potential energy of the well is given in the same way as in Figure 4.9.7.1, with a magnitude of $-1000\ \mathrm{kV/cm}$.

The energies of the six lowest-lying hole subbands for the (001) surface of the tensilely strained Si inversion layer are plotted as a function of the applied electric field in Figure 4.9.7.4 (b). The subband energies are measured from the surface potential which is assumed to be at $0\ \mathrm{eV}$ for the unstrained valence band edges. After application of strain, the highest valence band edge is the **light hole** band edge at $96.72\ \mathrm{meV}$. Our results are in excellent agreement with Fischetti's results.

At low electric fields ($-300\ \mathrm{kV/cm}$ and $-400\ \mathrm{kV/cm}$), the third hole eigenstate is the **second light hole state (lh2)**, whereas for higher fields this is the **split-off hole state (so1)**.

The plotting method is the same as in Figure 4.9.7.1 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 4.9.7.4 (c), (d), and (e) show the equienergy lines of the lowest lying **heavy hole**, **light hole**, and **split-off hole** subbands for the (001) surface of $1\ \%$ tensilely strained silicon, respectively. Only one spin state is plotted for clarity. The x axis points along the [100], the y axis along the [010] direction of the crystal coordinate system.

The eigenvalues are spin-degenerate only at $\mathbf{k} = (k_x, k_y) = 0$, but differ for non-zero **k**. The plots show the

$k_{||}$ dispersions of the lowest **light hole (1st eigenstate, (d))**, the lowest **heavy hole (3rd eigenstate, (c))**, and the lowest **split-off hole (5th eigenstate, (e))**.

1% compressively strained silicon inversion layer with (001) surface orientation

kpdispersion_Si_Fischetti_2003_1D_(001)_compressive_nnp is used in this section.

The figures below (Figure 4.9.7.5) aim to reproduce Fig.6(a), and Fig.7(b) of *[FischettiJAP2003]*.

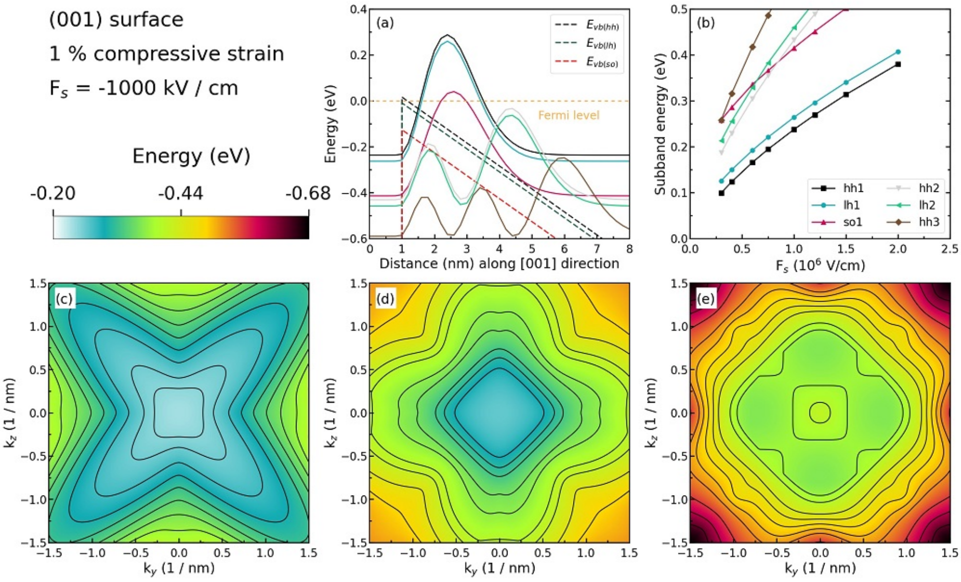


Figure 4.9.7.5: Some characteristics of the $1\ \%$ compressively strained silicon inversion layer with (001) surface orientation.

Figure 4.9.7.5 (a) shows the valence edges (where the heavy and light hole band edges are no longer degenerate) and the six lowest hole wave functions of a compressively strained Si inversion layer (triangular-well approximation) for $\mathbf{k} = \mathbf{0}$ (i.e. $k_x = k_y = 0$) where the z axis is oriented along the [001] direction. The compressive in-plane strain in the (x, y) plane is $1\ \%$. The potential energy of the well is given in the same way as in Figure 4.9.7.1, with a magnitude of $-1000\ \mathrm{kV/cm}$.

The energies of the six lowest-lying hole subbands for the (001) surface of the compressively strained Si inversion layer are plotted as a function of the applied electric field in Figure 4.9.7.5 (b). The subband energies are measured from the surface potential which is assumed to be at $0\ \mathrm{eV}$ for the unstrained valence band edges. After application of strain, the highest valence band edge is the **heavy hole** band edge at $15.47\ \mathrm{meV}$. Our results are in excellent agreement with Fischetti's results.

Again, we have crossings of the subbands. At small confining fields, the effect of confinement is compensated by the effect of strain.

The plotting method is the same as in Figure 4.9.7.1 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 4.9.7.5 (c), (d), and (e) show the equienergy lines of the lowewst lying **heavy hole**, **light hole**, and **split-off hole** subbands for the (001) surface of $1\ \%$ compressively strained silicon, respectively. Only one spin state is plotted for clarity. The x axis points along the [100], the y axis along the [010] direction of the crystal coordinate system.

The eigenvalues are spin-degenerate only at $\mathbf{k} = (k_x, k_y) = 0$, but differ for non-zero $\mathbf{k}$. The plots show the $k_{||}$ dispersions of the lowest **heavy hole (1st eigenstate, (c))**, the lowest **light hole (3rd eigenstate, (d))**, and the lowest **split-off hole (5th eigenstate, (e))**.

Unstrained silicon inversion layer with (001) surface orientation with different $k_{||}$ points

kpdispersion_Si_Fischetti_2003_1D_(001)_nnp is used in this section. However, the number of $k_{||}$ points in `dispersion` is different from the result above.

The figure below (Figure 4.9.7.6) show how the number of $k_{||}$ points affects the simulation results. The system is the same as the one we use in Figure 4.9.7.1, however, with different $k_{||}$ points. The equienergy lines are plotted for $E - E_0 = -25$ meV where E_0 is the eigenvalue of corresponding subbands at $\mathbf{k} = (k_x, k_y) = 0$.

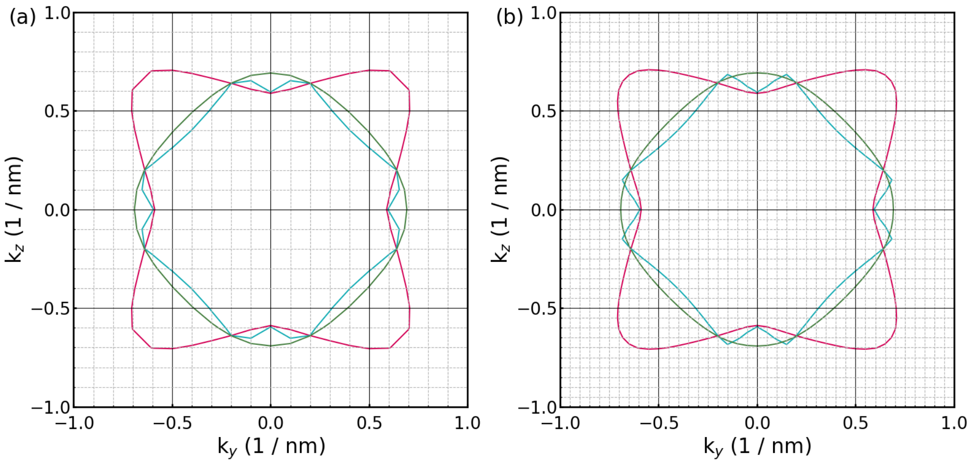


Figure 4.9.7.6: The dispersion for 441 $k_{||}$ points ((a)), and for 1681 $k_{||}$ points ((b)) for the (001) surface of the unstrained silicon under the electric field ($F_s = -1000$ kV/cm).

The grid points on the Figure 4.9.7.6 correspond to the $k_{||}$ points in the simulation. The figure shows that a smaller number of $k_{||}$ points is sufficient to obtain accurate results in this system.

Last update: 10/07/2024

4.9.8 Electronic band structure of 2DHG in Si inversion layers under arbitrary stress | 1D

Header

Files for the tutorial located in *nextnano++\examples\electronic_band_structures*

- *band-structure-2DHG_Si_Wang_2004_1D_nnp.in* - the input file

- *band-structure-2DHG_Si_Wang_2004_1D_nnp_uniax_strain.dat* - strain tensor for importing
- *band-structure-2DHG_Si_Wang_2004_1D_nnp_biax_tens_strain.dat* - strain tensor for importing
- *band-structure-2DHG_Si_Wang_2004_1D_nnp_biax_comp_strain.dat* - strain tensor for importing

Scope of the tutorial:

- strain effects
- anisotropy of electronic band structure

Main adjustable parameters in the input file:

- `$include_strain` - turn on and off computation of the strain
- `$strain_file` - name of the file with strain tensor to import
- `$electric_field` - choosing electric field

Relevant output files:

- *bias_00000\Quantum\probabilities_shift_quantum_region_kp6_00000.dat*
- *bias_00000\Quantum\Dispersions\dispersion_quantum_region_kp6_XXXX.fld*

Introduction

This tutorial aims at reproducing figures Fig. 2, Fig. 3., and Fig. 5 of *[Wang2004]*. These figures are presenting first subband energy contours of 2D hole gas (2DHG) in Si inversion layer with an effective field of 0.5 MV/cm and under several types of stress conditions:

- without any stress applied,
- with an uniaxial 1 GPa stress applied along [110],
- with a biaxial 1.7 GPa compressive stress applied along [100] and [010],
- with a biaxial 1.7 GPa tensile stress applied along [100] and [010].

Coordinate systems

As the growth direction [001] is set along the z-axis in *[Wang2004]*, the electronic band structures are spanned by [100] and [010] corresponding to x-axis and y-axis, respectively. Therefore, the wave-vector coordinates for electronic band structures k_x and k_y correspond to [100] and [010], repectively, as well.

Differently, the growth direction in the simulations presented in this tutorial is always set along the x-axis with [001] set along it. The remaining directions [100] and [010] are permutated accordingly to align with y-axis and z-axis, respectively. Therefore, the wave-vector coordinates for electronic band structures in the simulations k_y and k_z correspond to [100] and [010], repectively, as well.

As a result, crystallographic directions in the simulations of this tutorial are exactly aligned with the *[Wang2004]* while the simulation coordinate system is defined differently.

Defining the strain tensor

Here, we introduce how to calculate strain and import it to the simulation.

The relationship between the stress tensor (σ_{ij}) and the strain tensor (e_{ij}) for the crystals with zincblende symmetry is expressed as (4.9.8.1).

$$\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{zz}\\ \sigma_{yz}\\ \sigma_{zx}\\ \sigma_{xy}\end{bmatrix} = \begin{bmatrix}C_{11} & C_{12} & C_{12} & & & \\ C_{12} & C_{11} & C_{12} & & & \\ C_{12} & C_{12} & C_{11} & & & \\ & & & C_{44} & & \\ & & & & C_{44} & \\ & & & & & C_{44}\end{bmatrix} \begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{zz}\\ 2\varepsilon_{yz}\\ 2\varepsilon_{zx}\\ 2\varepsilon_{xy}\end{bmatrix} \tag{4.9.8.1}$$

Hint

See *Introduction to strain calculation* for further reference.

Uniaxial stress along [110]

First, we consider 1 GPa of uniaxial stress along the [110] direction. Uniaxial stress in the orthogonal coordinate system can be calculated using the method shown in uniaxial stress. Then, related stress tensor in GPa units is

$$\sigma_{[110]} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{zx} \\ * & \sigma_{yy} & \sigma_{yz} \\ * & * & \sigma_{zz} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.5 & -0.5 \\ 0 & -0.5 & -0.5 \end{bmatrix}.$$

Thus, you can solve the following simultaneous equations to obtain the strain components.

$$\begin{aligned}
\sigma_{xx} &= C_{11}\varepsilon_{xx} + C_{12}\varepsilon_{yy} + C_{12}\varepsilon_{zz} = 165.77 \cdot \varepsilon_{xx} + 63.93 \cdot \varepsilon_{yy} + 63.93 \cdot \varepsilon_{yy} = 0 \\
\sigma_{yy} &= C_{12}\varepsilon_{xx} + C_{11}\varepsilon_{yy} + C_{12}\varepsilon_{zz} = 63.93 \cdot \varepsilon_{xx} + 165.77 \cdot \varepsilon_{yy} + 63.93 \cdot \varepsilon_{zz} = -0.5 \\
\sigma_{zz} &= C_{12}\varepsilon_{xx} + C_{12}\varepsilon_{yy} + C_{11}\varepsilon_{zz} = 63.93 \cdot \varepsilon_{xx} + 63.93 \cdot \varepsilon_{yy} + 165.77 \cdot \varepsilon_{zz} = -0.5 \\
\sigma_{yz} &= 2C_{44}\varepsilon_{yz} = 2 \cdot 79.62 \cdot \varepsilon_{yz} = -0.5 \\
\sigma_{zx} &= 2C_{44}\varepsilon_{zx} = 2 \cdot 79.62 \cdot \varepsilon_{zx} = 0 \\
\sigma_{xy} &= 2C_{44}\varepsilon_{xy} = 2 \cdot 79.62 \cdot \varepsilon_{xy} = 0
\end{aligned}$$

As a result,

$$\begin{aligned}
\varepsilon_{xx} &= 0.00214 \\
\varepsilon_{yy} &= \varepsilon_{zz} = -0.00277 \\
\varepsilon_{yz} &= -0.00314 \\
\varepsilon_{zx} &= \varepsilon_{xy} = 0
\end{aligned}$$

This data is contained at *2DHG-strained-bands_Si_Wang_2004_1D_nnp_uniax_strain.dat*.

Hint

For guidance on importing strain to simulation follow *Importing files*.

Biaxial compressive stress along [100] and [010]

Next, we consider 1.7 GPa of biaxial compressive stress along [100] and [010]. Related stress tensor in GPa units is

$$\sigma_{[110]} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{zx} \\ * & \sigma_{yy} & \sigma_{yz} \\ * & * & \sigma_{zz} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1.7 & 0 \\ 0 & 0 & -1.7 \end{bmatrix}.$$

Thus, you can solve the following simultaneous equations to obtain the strain components as well as in the case of uniaxial stress.

$$\begin{aligned}
\sigma_{xx} &= C_{11}\varepsilon_{xx} + C_{12}\varepsilon_{yy} + C_{12}\varepsilon_{zz} = 165.77 \cdot \varepsilon_{xx} + 63.93 \cdot \varepsilon_{yy} + 63.93 \cdot \varepsilon_{yy} = 0 \\
\sigma_{yy} &= C_{12}\varepsilon_{xx} + C_{11}\varepsilon_{yy} + C_{12}\varepsilon_{zz} = 63.93 \cdot \varepsilon_{xx} + 165.77 \cdot \varepsilon_{yy} + 63.93 \cdot \varepsilon_{zz} = -1.7 \\
\sigma_{zz} &= C_{12}\varepsilon_{xx} + C_{12}\varepsilon_{yy} + C_{11}\varepsilon_{zz} = 63.93 \cdot \varepsilon_{xx} + 63.93 \cdot \varepsilon_{yy} + 165.77 \cdot \varepsilon_{zz} = -1.7 \\
\sigma_{yz} &= 2C_{44}\varepsilon_{yz} = 2 \cdot 79.62 \cdot \varepsilon_{yz} = 0 \\
\sigma_{zx} &= 2C_{44}\varepsilon_{zx} = 2 \cdot 79.62 \cdot \varepsilon_{zx} = 0 \\
\sigma_{xy} &= 2C_{44}\varepsilon_{xy} = 2 \cdot 79.62 \cdot \varepsilon_{xy} = 0
\end{aligned}$$

As a result,

$$\begin{aligned}
\varepsilon_{xx} &= 0.00727 \\
\varepsilon_{yy} &= \varepsilon_{zz} = -0.00277 \\
\varepsilon_{yz} &= \varepsilon_{zx} = \varepsilon_{xy} = 0
\end{aligned}$$

This data is contained at *band-structure-2DHG_Si_Wang_2004_1D_nnp_biax_comp_strain.dat*.

Biaxial tensile stress along [100] and [010]

Next, we consider 1.7 GPa of biaxial tensile stress along [100] and [010]. You just need to change the signs of the strain components in the previous section.

Therefore,

$$\begin{aligned}\varepsilon_{xx} &= -0.00727\\ \varepsilon_{yy} = \varepsilon_{zz} &= 0.00277\\ \varepsilon_{yz} = \varepsilon_{zx} = \varepsilon_{xy} &= 0\end{aligned}$$

This data is contained at *band-structure-2DHG_Si_Wang_2004_1D_nnp_biax_tens_strain.dat*.

Simulation results

No stress applied

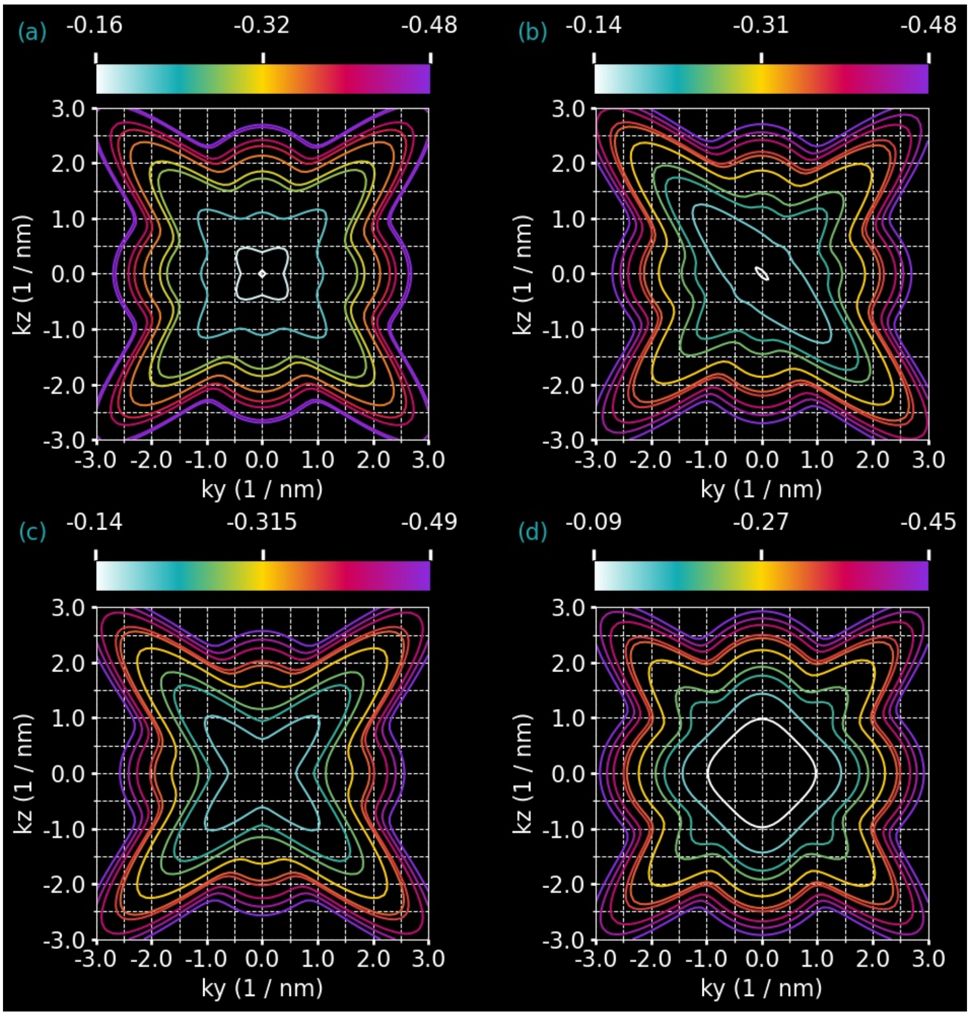


Figure 4.9.8.1: The calculated equienergy lines under no strain (a), under uniaxial strain (b), under biaxial compressive strain (c), and under biaxial tensile strain (d) are shown. Only one spin state is plotted for clarity. The axes represent k_y and k_z in units of $[1/\mathrm{nm}]$.

First, Figure 4.9.8.1 (a) shows the energy dispersion under no strain. This corresponds to Fig.2 in *[Wang2004]*. The electric field is applied to model a triangular well potential, which causes the inversion layer. The magnitude is $0.5\ \mathrm{MV/cm}$ along the crystal growth direction.

The energy dispersion is in *Dispersions\dispersion_quantum_region_kp6_XXXX.fld*.

Next, Figure 4.9.8.1 (b) shows the energy dispersion under uniaxial compressive strain. This is equivalent to Fig.3 in *[Wang2004]*. Note that the uniaxial stress is 1.0 GPa and the direction is [110]. Furthermore, the same magnitude of the electric field is applied as well as in under no strain.

Next, Figure 4.9.8.1 (c), (d) shows the energy dispersion under compressive / tensile biaxial strain, respectively. This corresponds to Fig.5 in *[Wang2004]*. Note that the biaxial stress is 1.7 GPa and the direction is in-plane. The same magnitude of the electric field is applied as well as in under no strain. Here, the lowest subband is composed by **heavy hole** in (a), whereas **light hole** composes the lowest subband in (b).

Overall, our simulation results match very well with the results in *[Wang2004]*.

Last update: 07/03/2024

4.10 Superlattices

4.10.1 Dispersion in infinite superlattices: Minibands (Kronig-Penney model)

Input files:

- *1Dsuperlattice_dispersion_4nm_nnpp.in*
- *1Dsuperlattice_dispersion_6nm_nnpp.in*
- *1Dsuperlattice_dispersion_bulk_GaAs_nnpp.in*
- *Superlattice_1D_nnpp.in*

Scope:

This tutorial aims to reproduce two figures (Figs. 2.27, 2.28, p. 56f) of *[HarrisonQWWD2005]*, thus the following description is based on the explanations made therein.

Superlattice 1: 4 nm $AlGaAs$ / 4 nm $GaAs$

Input file: *1Dsuperlattice_dispersion_4nm_nnpp.in*

Our infinite superlattice consists of a 4 nm $GaAs$ quantum well surrounded by 2 nm $Al_{0.4}Ga_{0.6}As$ barriers on each side. The choice of periodic boundary conditions leads to the following sequence of identical quantum wells: 4 nm $AlGaAs$ / 4 nm $GaAs$ / 4 nm $AlGaAs$ / 4 nm $GaAs$ / $\ldots$. So our superlattice period has the length L = 8 nm. (Actually it has the length L = 8.25 due to the grid point resolution of 0.25 nm.)

Figure 4.10.1.1 shows the conduction band edge and the first eigenstate that is confined inside the well and its corresponding charge density (Ψ^2) for the superlattice vector k_z = 0. Note that periodic boundary conditions are employed for solving the Schrödinger equation. The second eigenstate is not confined inside the well and is therefore not shown here. (Note that the energies were shifted so that the conduction band edge of $GaAs$ equals 0 eV.)

In a superlattice the electrons (and holes) see a periodic potential which is similar to the periodic potential in bulk crystals. This means that the particle wave functions are no longer localized in one quantum well. They extend to infinity, and they are equally likely to be found in any of the quantum wells. The eigenstates are called **Bloch states** (as in bulk) and the wave functions are periodic:

$$\Psi(x) = \Psi(x + L)$$

For a travelling wave of the form $exp(ik_x x)$ it holds that

$$\Psi(x + L) = e^{ik_x(x+L)} = e^{ik_x x}e^{ik_x L}$$

$$\Leftrightarrow \quad \Psi(x + L) = \Psi(x)e^{ik_x L}$$

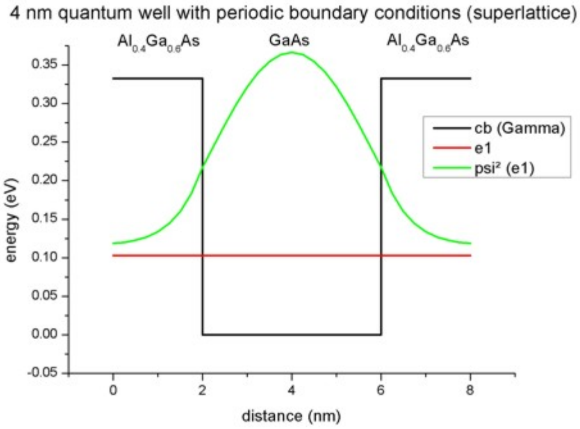


Figure 4.10.1.1: Calculated conduction band edge profile of single 4 nm GaAs QW with periodic boundary conditions.

k_x is the wave vector of the electron (or hole) along the growth direction of the infinite superlattice. In Figure 4.10.1.2 we plot the dispersion curve, i.e. the energy of the electron as a function of its superlattice wave vector k_x for the lowest eigenstate. As the energy is a periodic function of k_x with period $2\pi/L$, we plot only the interval $[-\pi/L, \pi/L]$.

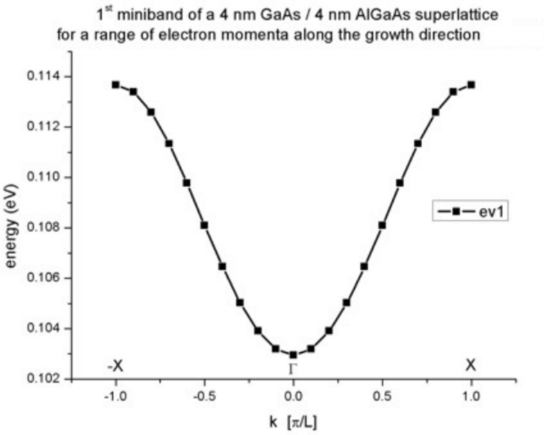


Figure 4.10.1.2: Calculated subband dispersion (= miniband)

The plot is in excellent agreement with Fig. 2.27 (page 56) of *[HarrisonQWWD2005]*. When the electron is at rest (k_x = 0), the dispersion curve shows a minimum. As the electron momentum k_x increases, its energy also increases and reaches a maximum at $k_x = -\pi/L$ and $k_x = +\pi/L$. Thus, the electron within the superlattice occupies a continuum of energies. This continuum that is bound by a maximum and a minimum of energy is called miniband. Due to the similarity with the energy bands of a bulk crystal, the point in the superlattice Brillouin zone for k_x = 0 is called Gamma and for $k_x = \pi/L$ it is called X.

Superlattice 2: 6 nm $AlGaAs$ / 6 nm $GaAs$

Input file: *1Dsuperlattice_dispersion_6nm_nnpp.in*

Our second infinite superlattice consists of a 6 nm $GaAs$ quantum well surrounded by 3 nm $Al_{0.4}Ga_{0.6}As$ barriers on each side. The choice of periodic boundary conditions leads to the following sequence of identical quantum wells: 6 nm $AlGaAs$ / 6 nm $GaAs$ / 6 nm $AlGaAs$ / 6 nm $GaAs$ / $\ldots$. So our superlattice period has the length L = 12 nm. (Actually it has the length L = 12.25 due to the grid point resolution of 0.25 nm.)

Figure 4.10.1.3 shows the conduction band edge and the two lowest eigenstates that are confined inside the well and their corresponding probaility density (Ψ^2) for the superlattice vector k_x = 0. Note that periodic boundary conditions are employed for solving the Schrödinger equation. The third eigenstate is not confined inside the well and is therefore not shown here. In contrast to the 4 nm quantum well superlattice described above, two confined electron states exist. (Note that the energies were shifted so that the conduction band edge of $GaAs$ equals 0 eV.)

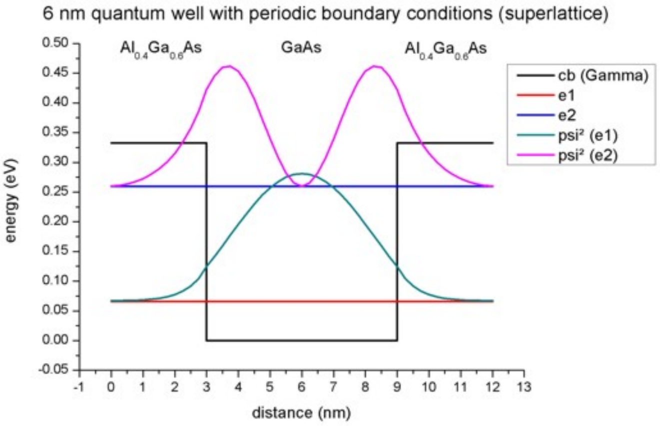


Figure 4.10.1.3: Calculated conduction band edge profile of single 6 nm $GaAs$ QW with periodic boundary conditions.

The following figure (Figure 4.10.1.4) shows the first two minibands for this superlattice. They arise from the first and the second eigenstate. Note that due to the scale of this figure the first miniband looks almost flat. It is also interesting that for the second miniband the minimum is not at the center (i.e. at Gamma) but at the edges of the superlattice Brillouin zone at X (and -X).

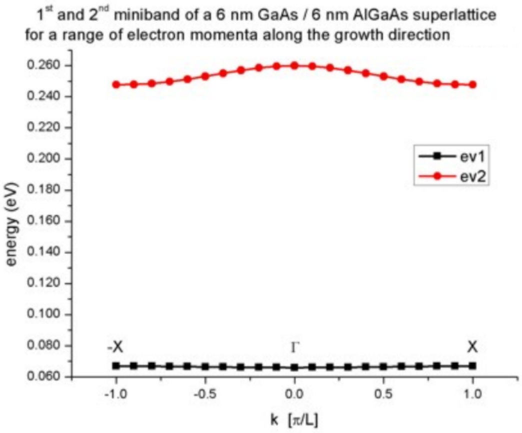


Figure 4.10.1.4: Calculated subband dispersion (= miniband)

Again, the plot is in excellent agreement with Fig. 2.28 (page 57) of *[HarrisonQWWD2005]*.

Technical details

The resolution of the miniband plot has to be specified within the group `quantum{ region{ dispersion{} } }`:

```
quantum{
    region{
        ...
        dispersion{
            output_dispersions{}
            path{
                name = "superlattice_dispersion"
                point{ k = [$left_dispersion,  0.0, 0.0] }
                point{ k = [$right_dispersion, 0.0, 0.0] }
```

(continues on next page)

(continued from previous page)

```
                num_points = $num_points_dispersion          # number of superlattice␣
↪vectors along x direction
            }
        }
    }
}
```

For each superlattice vector k_x, the Schrödinger equation has to be solved. The 11th superlattice vector corresponds to k_x = 0 which is obviously identical to the case when no superlattice is specified at all. The miniband dispersion is written to this file: *dispersion_quantum_region_Gamma_superlattice_dispersion.dat*.

Dispersion in bulk $GaAs$

Input file: *1Dsuperlattice_dispersion_bulk_GaAs_nnpp.in*

The input file is basically equivalent to *1Dsuperlattice_dispersion_6nm_nnpp.in*, except that we replace the $AlGaAs$ barrier with $GaAs$ so that we have only pure bulk $GaAs$ with a length of 12 nm. So our superlattice period has the length L = 12 nm. (Actually it has the length L = 12.25 due to the grid point resolution of 0.25 nm.) At the boundaries we apply periodic boundary conditions and the same superlattice options (number of k values and direction in k space) as above.

Figure 4.10.1.5 shows the conduction band edge and the three lowest eigenstates and their corresponding probability density (Ψ^2) for the superlattice vector k_x = 0. Note that periodic boundary conditions are employed for solving the Schrödinger equation.

- The ground state wave function is constant with its energy equal to the conduction band edge energy.
- The energies of the second and third eigenstate are degenerate.

(Note that the energies were shifted so that the conduction band edge of $GaAs$ equals 0 eV.)

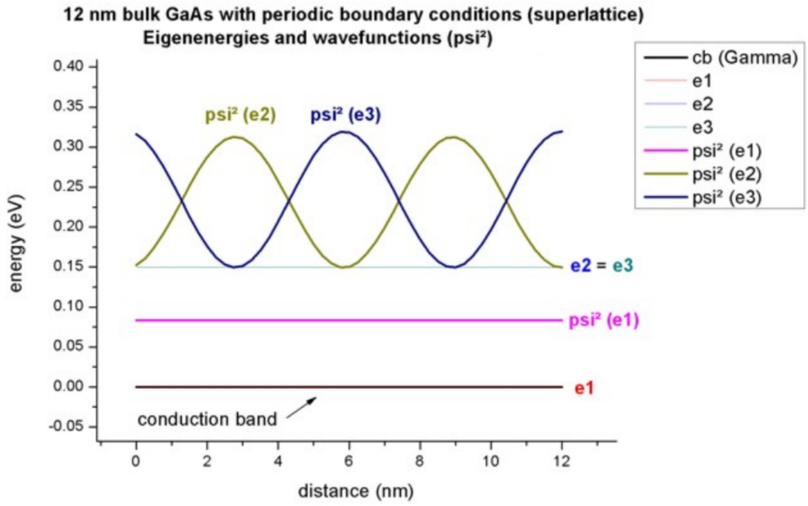


Figure 4.10.1.5: Calculated conduction band edge profile of bulk $GaAs$ and Ψ^2 of lowest electron eigenstates (periodic boundary conditions were used).

The following figure (Figure 4.10.1.6) shows the first three minibands for this superlattice. They arise from the first, second and third eigenstate. The second and third eigenstate are degenerate at k_x = 0 as can be seen also in the figure above. Also at k_x = -1 and k_x = 1, the first and second eigenstate are degenerate. This is as expected because the dispersion should look like the parabolic dispersion $E(k)$ of bulk $GaAs$.

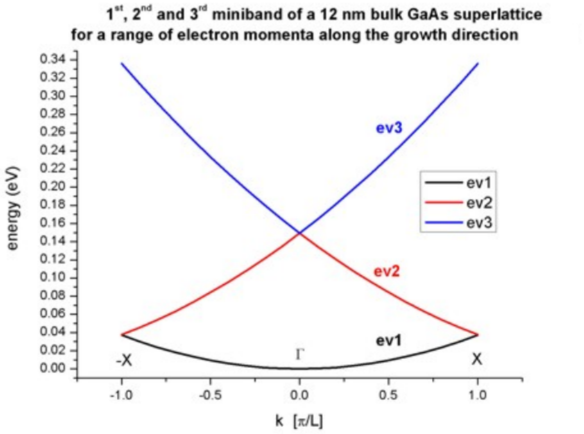
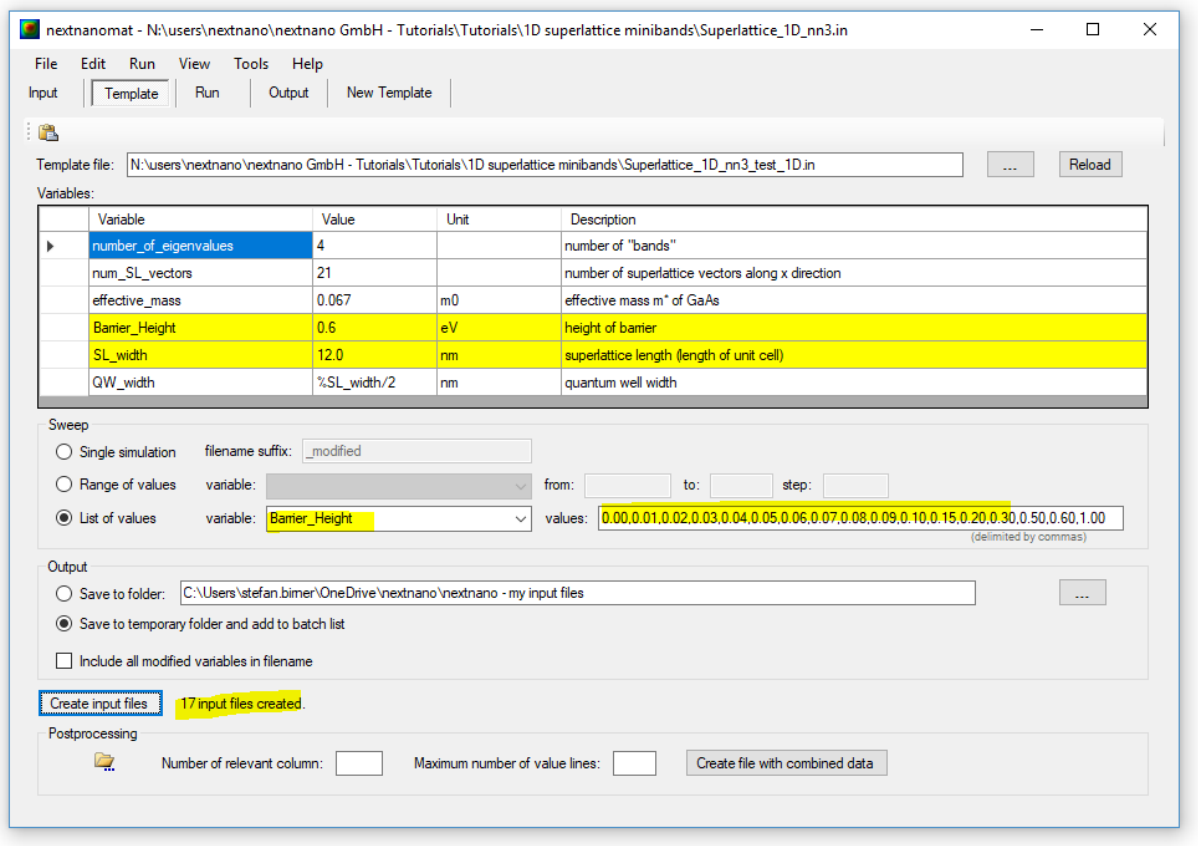


Figure 4.10.1.6: Calculated subband dispersion (= miniband)

Template

Input file: *Superlattice_1D_nnpp.in*

We want to study the energy levels of a superlattice in order to understand how they form bands in a periodic structure. One can easily see this by calculating the energy levels for various barrier heights, i.e. we automatically generate input files for the variable “Barrier_Height”. Once done, we visualize the subband dispersions contained in the file *dispersion_quantum_region_Gamma_superlattice_dispersion.dat*.



Template file: N:\users\nextnano\nextnano GmbH - Tutorials\Tutorials\1D superlattice minibands\Superlattice_1D_nn3_test_1D.in

Variables:

Variable	Value	Unit	Description
number_of_eigenvalues	4		number of "bands"
num_SL_vectors	21		number of superlattice vectors along x direction
effective_mass	0.067	m0	effective mass m* of GaAs
Barrier_Height	0.6	eV	height of barrier
SL_width	12.0	nm	superlattice length (length of unit cell)
QW_width	%SL_width/2	nm	quantum well width

Sweep

○ Single simulation filename suffix: _modified

○ Range of values variable: from: to: step:

● List of values variable: Barrier_Height values: 0.00,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.10,0.15,0.20,0.30,0.50,0.60,1.00 (delimited by commas)

Output

○ Save to folder: C:\Users\stefan.birner\OneDrive\nextnano\nextnano - my input files

● Save to temporary folder and add to batch list

☐ Include all modified variables in filename

17 input files created.

Postprocessing

Number of relevant column: Maximum number of value lines:

Figure 4.10.1.7 compares the dispersion of a superlattice for two different QW barrier heights.

Last update: nnnn/nn/nn

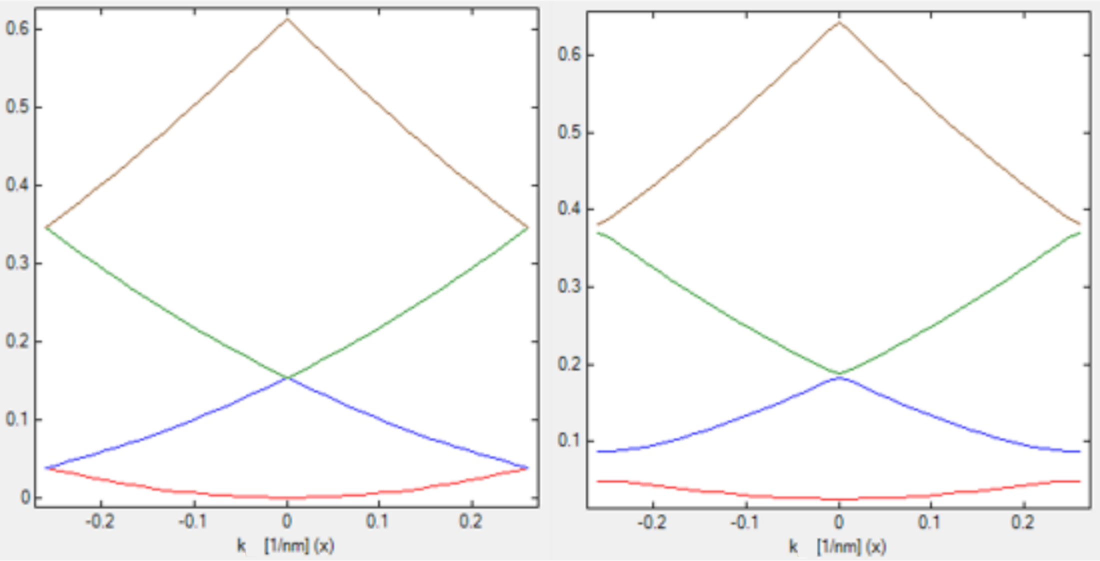


Figure 4.10.1.7: The left figure contains a quantum well superlattice with a barrier height of 0 eV, i.e. a bulk semiconductor, while the figure on the right shows the dispersion for a barrier height of 0.06 eV.

4.10.2 InAs / $In_{0.4}Ga_{0.6}Sb$ superlattice dispersion with 8-band k.p (type-II band alignment)

Authors: Stefan Birner, Michael Povolotskyi

Input Files:

- *T2SL_InAs-GaInSb_Grein_JAP_1995_1D_nnp*

This tutorial aims to reproduce Fig. 2(a) of *"Long wavelength InAs/InGaSb infrared detectors: Optimization of carrier lifetimes"* by Grein and Young.

Conduction and valence band edges

The heterostructure is a **superlattice** with 3.98 nm InAs and 1.5 nm $In_{0.4}Ga_{0.6}Sb$, where both constituents are strained with respect to the GaSb substrate.

The structure has a **type-II** band alignment, i.e. the electrons are confined in the InAs layer, whereas the holes are confined in the $In_{0.4}Ga_{0.6}Sb$ layer.

The $In_{0.4}Ga_{0.6}Sb$ layer is strained pseudomorphically with respect to the GaSb substrate, leading to a **compressive** strain (-2.5%) which splits the degeneracy of the heavy and light hole band edges in this layer. Thus, the heavy hole band edge lies above the light hole band edge.

The InAs layer is also strained pseudomorphically with respect to the GaSb substrate, and is thus under slight biaxial **tension** (+0.6 %). The splitting of the hole band edges is the opposite as in InGaSb, i.e. the light hole band edge is above the heavy hole band edge.

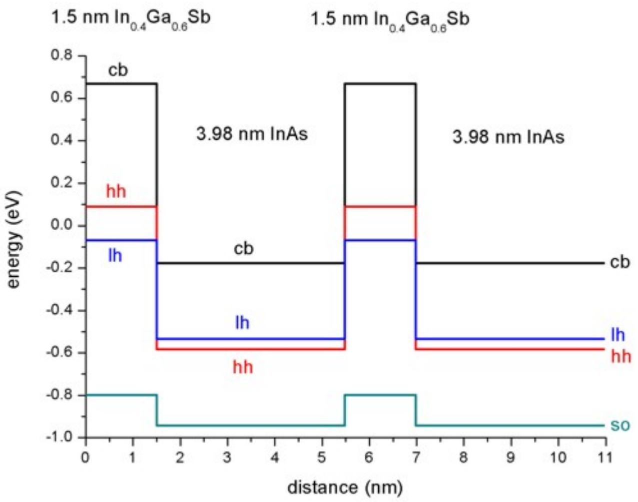
The following figure shows the electron and hole band edges.

Note that the origin of the energy scale is set to the GaSb valence band edge energy.

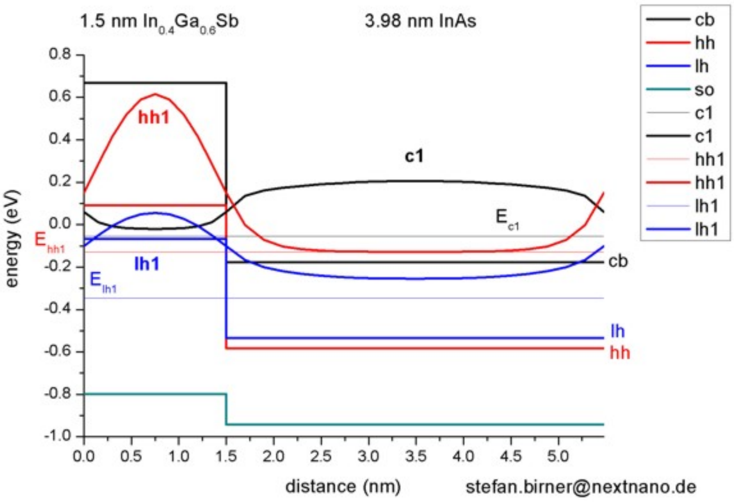
Electron and hole wave function for $k_{||} = 0$

We simulate **one period only** (i.e. from 0 nm to 5.48 nm) and solve the Schrödinger equation with periodic boundary conditions to mimic an infinite superlattice.

The following figure shows the conduction band edge and the heavy, light and split-off hole valence band edges in this superlattice structure together with the electron (**c1**), heavy hole (hh1) and light hole (lh1) energies and wave functions (ψ^2), calculated within 8-band k.p theory.



One can clearly see that the electron state (**c1**) is confined in the InAs layer (right part of the figure), whereas the heavy (hh1) and light hole (lh1) states are confined in the $In_{0.4}Ga_{0.6}Sb$ layer (left part of the figure).



We used the same material parameters as given in the above cited paper by Grein *et al.*, apart from the $\mathbf{k} \cdot \mathbf{p}$ parameters.

Electron and hole energies for $k_{||} \neq 0$

The following figure shows the $E(\mathbf{k}_{||})$ dispersion of the electron ground state and the two highest hole states along two different directions in (k_x, k_y) space.

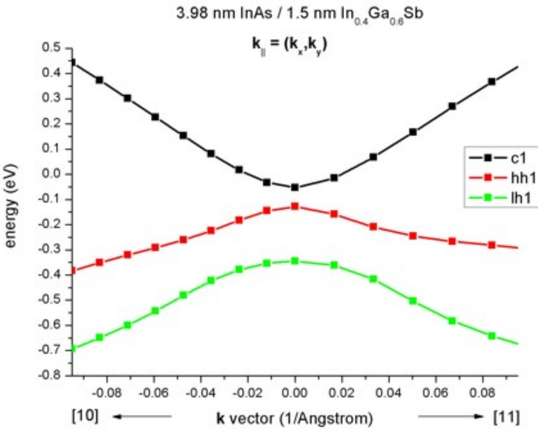
This data is contained in this file: `Schroedinger_kp/par1D_disp_01_00_11_hl_8x8kp_ev_min001_ev_max010.dat`. Note that the band gap is not determined by the band gap of one individual layer. It is determined by the electron ground state in the InAs layer, and the hole ground state in the InGaSb layer. This means more freedom for band gap engineering.

Electron and hole energies for $k_z \neq 0$

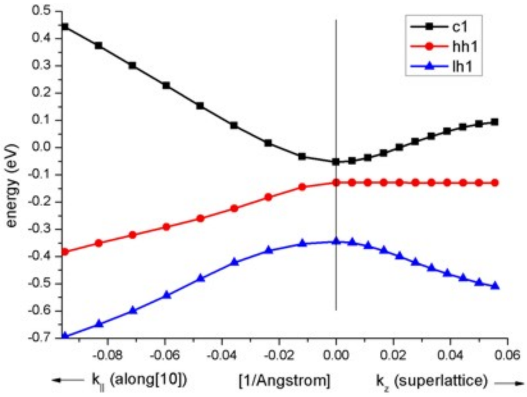
The input file used is `1DInAs_InGaSb_k_superlattice_nnp.in`

The right part of the following figure shows the $E(k_z)$ superlattice dispersion of the electron ground state and the two highest hole states. k_z is the superlattice vector between 0 and 1 $\frac{\pi}{L}$ where L = 5.48 nm is the length of one superlattice period. ($1\frac{\pi}{L} = 0.05731/\text{Å}$)

This data is contained in this file: `Schroedinger_kp/8x8kp_dispSL_hl_qc001_evmin001_evmax016.dat`



The left part of the figure shows the E($k_{||}$) dispersion along [10], i.e. from (k_x, k_y) = (0,0) to (k_x, k_y) = (-0.1,0) which is shown in the figure above already.



One can clearly see that these heterostructure bands are highly nonparabolic.

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.10.3 Multiple quantum wells and finite superlattices

Author: Brandon Loke

This tutorial simulates a real layered structure with a finite number of quantum wells. The transition between a finite superlattice and a multiple quantum well system is also observed. This tutorial aims to reproduce the figures in Paul Harrison's book *"Quantum Wells, Wires and Dots"* (Section 3.10, "Multiple Quantum Wells and Finite Superlattices")

The input file used for this tutorial is

- *Superlattice_N_wells_nnp.in*

The corresponding Jupyter Notebook for this tutorial can be found over here: MQW_Superlattices.ipynb.

Structure

The structure consists of *N* repeats of 4 nm GaAs wells and 4 nm Ga$_{0.8}$Al$_{0.2}$As quantum wells. This superlattice structure is sandwiched between 20 nm Ga$_{0.8}$Al$_{0.2}$As barriers.

We first define key variables, such as the well width, the right and left wall width, and the number of wells.

```
# Global constants
$TEMPERATURE                          = 300           # Temperature␣
↪(DisplayUnit:K)(ListOfValues:270, 280, 290, 300, 310, 320, 330)

# Structure

$WELL_WIDTH                           = 10.0           # Width of the quantum well␣
↪(DisplayUnit:nm)(HighlightInUserInterface)(ListOfValues:5.0, 6.0, 7.0, 8.0, 9.0) ␣
↪(RangeOfValues:From=5.0,To=9.0,Step=1.0)
$BARRIER_WIDTH                        = 10.0           # Width of the barrier     ␣
↪(DisplayUnit:nm)(HighlightInUserInterface)(ListOfValues:7.0, 8.0, 9.0, 10.0, 11.
↪0)(RangeOfValues:From=57.0,To=11.0,Step=1.0)
$NUMBER_OF_WELLS                      = 4              # number of quantum wells  ␣
↪(DisplayUnit:)(HighlightInUserInterface)(ListOfValues:4, 5, 6, 7,␣
↪8)(RangeOfValues:From=3,To=10,Step=1)

$SUPERLATTICE_WIDTH                   = $NUMBER_OF_WELLS * ( $BARRIER_WIDTH + $WELL_
↪WIDTH ) - $BARRIER_WIDTH    # (DisplayUnit:nm)(DoNotShowInUserInterface)

$LEFT_BARRIER_WIDTH                       = 10             # Width of the Separate␣
↪Confinement Heterostructure (SCH) (on the left) (DisplayUnit:nm)
$RIGHT_BARRIER_WIDTH                      = 10             # Width of the Separate␣
↪Confinement Heterostructure (SCH) (on the right)(DisplayUnit:nm)

$FINE_GRID_SPACING                    = 0.1                #␣
↪(DisplayUnit:nm)(ListOfValues:0.1, 0.5, 1.0)(DoNotShowInUserInterface)
$COARSE_GRID_SPACING                  = 1.0                #␣
↪(DisplayUnit:nm)(ListOfValues:0.5, 1.0, 5.0)(DoNotShowInUserInterface)

# Materials and doping
$ALLOY_X = 0.8
```

Following this, we are able to generate the structure of the GaAs/Ga$_{0.8}$Al$_{0.2}$As superlattice under `structure{ }`. The keywords `array_x{}` duplicate the structure in the x-direction to give us the number of wells required.

```
region{                                            # LEFT WALL
   line{
      x = [-$LEFT_BARRIER_WIDTH,   0]
   }
   ternary_constant{
      name     = "Ga(x)Al(1-x)As"                  # Ga0.8Al0.2As
      alloy_x  = $ALLOY_X
   }
}

# SUPERLATTICE

region{                                            # Barrier
   array_x{
      shift          = $BARRIER_WIDTH + $WELL_WIDTH
      max            = $NUMBER_OF_WELLS - 1
   }
   line{
      x = [0,   $BARRIER_WIDTH]
   }
   binary{
      name     = "GaAs"
```

(continues on next page)

(continued from previous page)

```
   }
}
region{                                            # Quantum well
   array_x{
      shift          = $BARRIER_WIDTH + $WELL_WIDTH
      max            = $NUMBER_OF_WELLS - 2
   }
   line{
      x = [$BARRIER_WIDTH,   $BARRIER_WIDTH + $WELL_WIDTH]
   }
   ternary_constant{
      name     = "Ga(x)Al(1-x)As"
      alloy_x  = $ALLOY_X
   }
}
region{                                            # RIGHT WALL
   line{
      x = [$SUPERLATTICE_WIDTH,   $SUPERLATTICE_WIDTH + $RIGHT_BARRIER_WIDTH]
   }
   ternary_constant{
      name    = "Ga(x)Al(1-x)As"
      alloy_x = $ALLOY_X
   }
}
```

Simulation Settings

Under `quantum{ }`, we specify

```
quantum {
   region{
       name = "quantum_region"
       x = [                         - $LEFT_BARRIER_WIDTH, $SUPERLATTICE_WIDTH +
↪$RIGHT_BARRIER_WIDTH ]                         # Schrödinger equation is solved␣
↪only in region of LEFT WALL + SUPERLATTICE + RIGHT WALL

       boundary{
   #       x  = dirichlet                                  # Dirichlet boundary␣
↪condition for the Schrödinger equation,     psi = 0
           x  = neumann                                    #   Neumann boundary␣
↪condition for the Schrödinger equation, dpsi/dx = 0
       }

       Gamma{
           num_ev = 70
                                   # 70 eigenvalues have to be calculated
       }

       HH{
           num_ev = 250                                    # 150 eigenvalues have to␣
↪be calculated
       }

       LH{
           num_ev = 70                                    # 70 eigenvalues have to be␣
↪calculated
```

(continues on next page)

(continued from previous page)

```
        }

        SO{
            num_ev = 100                                    # 100 eigenvalues have to␣
↪be calculated
        }

        output_wavefunctions{
            max_num = 20                                    # only 20 eigenfunctions␣
↪from 100 calculated are shown in output
            amplitudes = yes
            probabilities = yes
        }
    }
}
```

We want to obtain the energies and the amplitudes of the wave functions outputted.

Ground state energies

After generating the input file, we are able to run the simulation for a variable number of quantum wells using the variable sweep functionality in *nextnanomat*. One can go to "Template" on the tabs at the top, under "Sweep", select the variable of interest and the range or list of values to iterate over. Click on "Create input file" at the bottom and run the simulations in the "Simulation" tab.

The reference potential energy used in Harrison's book and *nextnano*++ is different. Thus, post-processing was done in Python to match the reference energy levels.

Wave function in a superlattice

The wave functions can also be plotted. The first example in Harrison's book has the following parameters:

- 10 wells
- 4 nm $Ga_{0.8}Al_{0.2}As$ barrier
- 4 nm $Ga_{0.8}Al_{0.2}As$ quantum well width
- 20 nm left and right $Ga_{0.8}Al_{0.2}As$ walls

This figure is in agreement with Harrison's result. It is observed that the system functions as a superlattice as the wave function in each well overlaps with the wave function of the adjacent wells.

Wave function in a multiple quantum well system

Harrison's final figure uses the following parameters:

- 4 wells
- 10 nm $Ga_{0.6}Al_{0.4}As$ barriers
- 10 nm $Ga_{0.6}Al_{0.4}As$ quantum wells
- 10 nm $Ga_{0.6}Al_{0.4}As$ left and right walls

This figure is also in good agreement with Harrison's results. It is observed from the figure that this system functions as a multiple quantum well because the wave function reaches zero between the wells.

Last update: nnnn/nn/nn

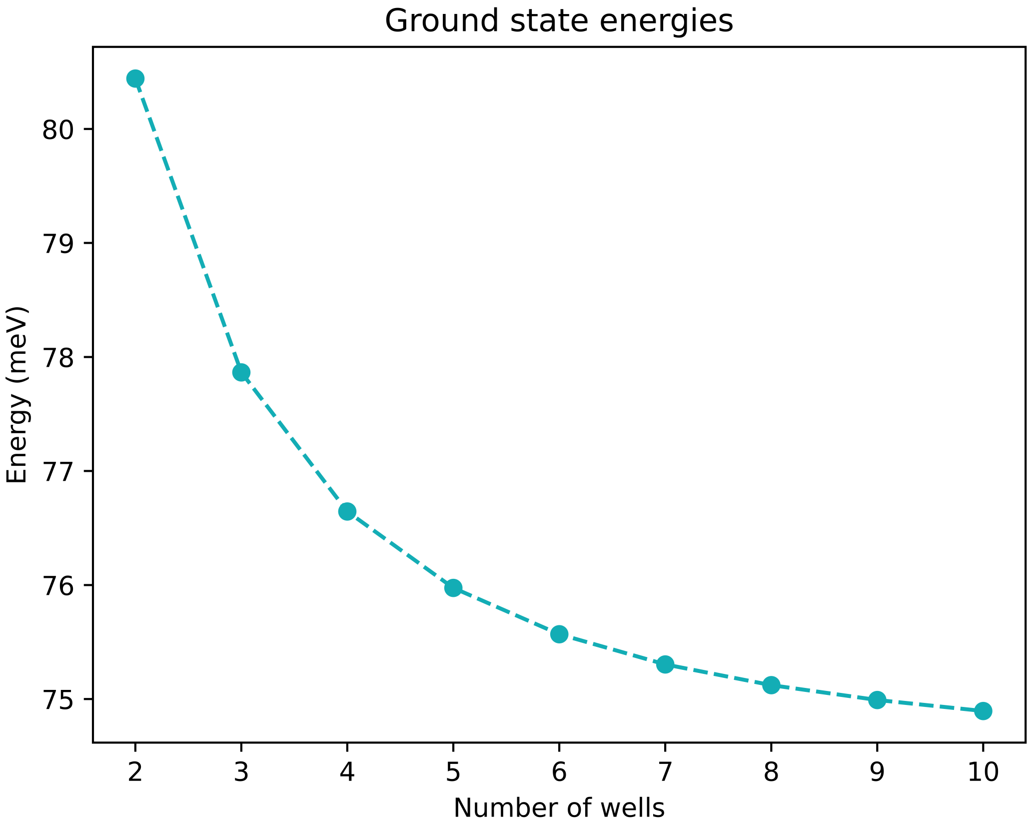


Figure 4.10.3.1: Ground state energies plotted as a function of N. Convergence at higher number of wells is observed.

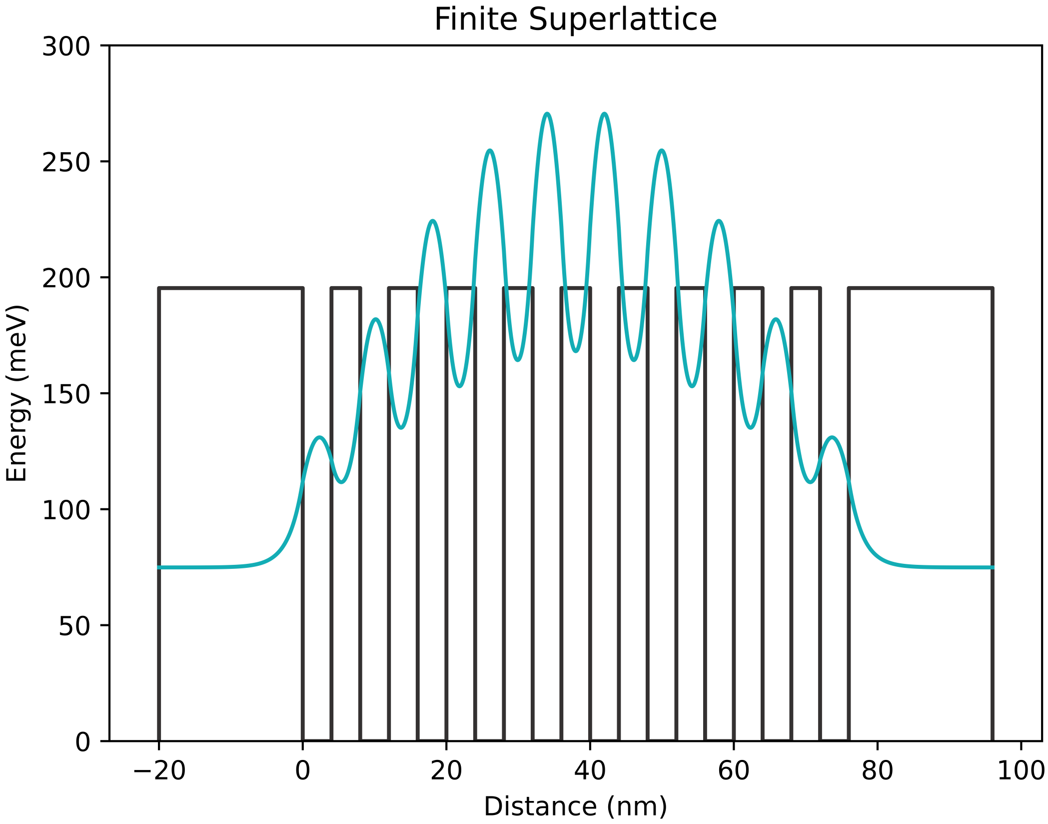


Figure 4.10.3.2: The wave function for a superlattice system

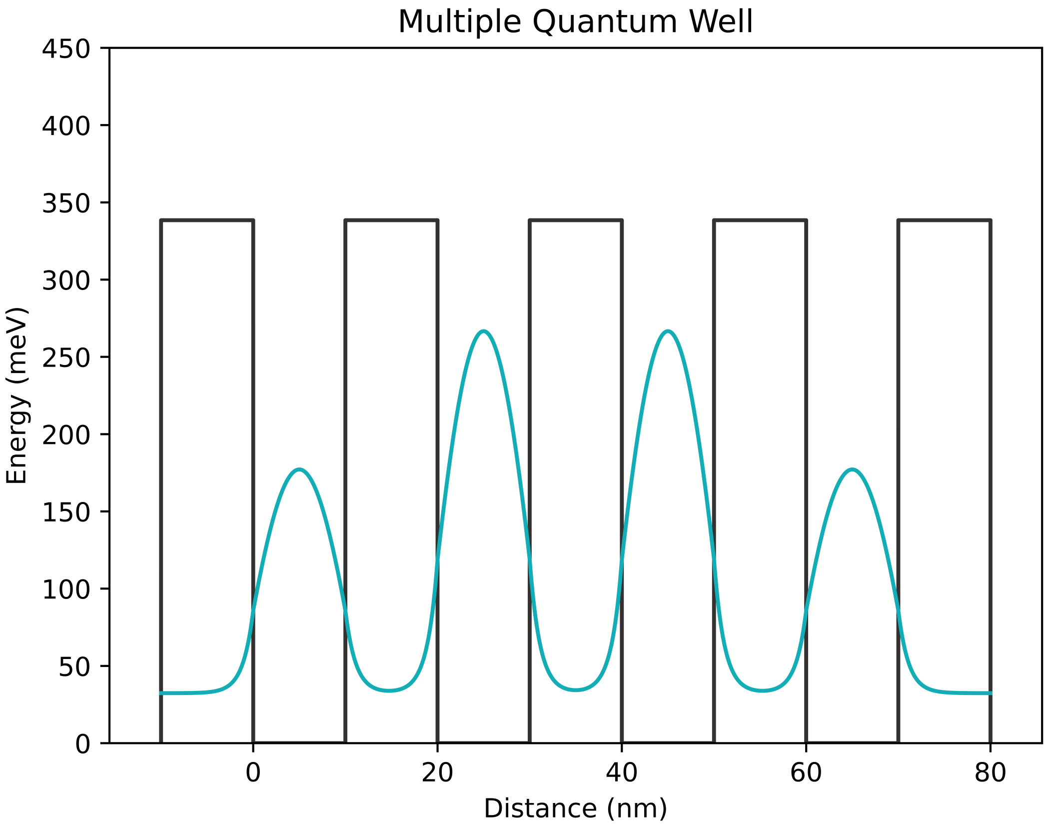


Figure 4.10.3.3: The wave function for a multiple quantum well system

4.10.4 — NEW — Modeling type-II superlattice using interface Hamiltonian within 8-band $\mathrm{k} \cdot \mathrm{p}$ method

Contents

Files for the tutorial located in *nextnano++\examples*

- *T2SL_H-if_zb_III-V_Livneh_PRB_2012_1D_bandstructure.in* (Figure 4.10.4.1)
- *T2SL_H-if_zb_III-V_Livneh_PRB_2012_1D_absorption.in* (Figure 4.10.4.2)

Parameters

- `$if_interface`: 1 or 0, switches on and off the interface Hamiltonian
- `$Nperiods`: Number of periods in the superlattice.

Output files

- *bias_00000\Quantum\Dispersions\dispersion_quantum_region_kp8_inplane_11_00_10.dat* (Figure 4.10.4.1)
- *bias_00000\Quantum\Dispersions\dispersion_quantum_region_kp8_superlattice.dat* (Figure 4.10.4.1)
- *bias_00000\OpticsQuantum\absorption_quantum_region_TE_y_nm.dat* (Figure 4.10.4.2)

Introduction

This tutorial validates *nextnano++* interface Hamiltonoian implementation with the results from *[Livneh-PRB2012]*. The model and the implementation are described in detail in the *Interface Hamiltonian 8-band Zinc-Blende* section.

The objective is to calculate dispersion and optical properties of type-II InAs/GaSb superlattice.

The samples modeled are described in the table below, The thickness of the layers is expressed in monolayers (ML). In the original paper, the dispersion relation of sample 1 is given in figure 4, and the optical absorption of sample 2 is given in figure 6.

Parameter	Sample 1	Sample 2
InAs thickness, ML	13.8	8.7
GaSb thickness, ML	7.8	10.6
Temperature, K	77	77

Results

The disperson relation of the sample one is computed using the input file *T2SL_H-if_zb_III-V_Livneh_PRB_2012_1D_bandstructure.in*. For calculating dispersion, it is enough to include single period in the simulation. The dispersion relation is plotted in the figure below.

The optical absorption of the sample two is computed using the input file *T2SL_H-if_zb_III-V_Livneh_PRB_2012_1D_absorption.in*. For adequat computation of the absorption in superlattice, few periods has to be included. In the figure below, 20 periods were fused in the computation.

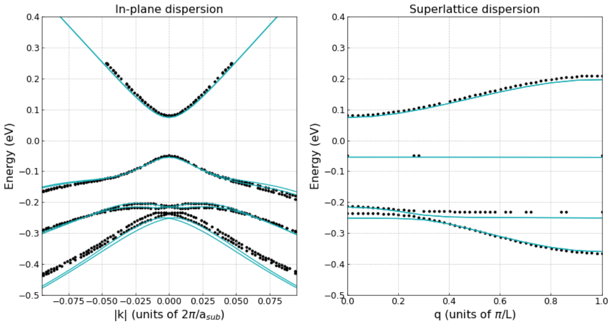


Figure 4.10.4.1: Dispersion relation of the sample 1, k-parallel dispersion on the left and superlattice dispersion on the right. Solid lines are *nextnano*++ results, dots represent data from *[LivnehPRB2012]*.

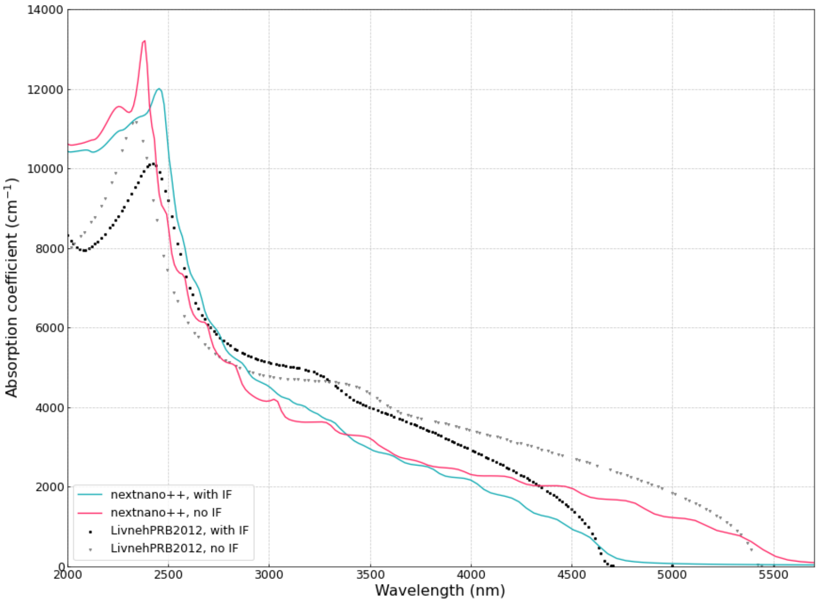


Figure 4.10.4.2: Absorption coefficient calculated for the sample 2 with and without the interface (IF) Hamiltonian.

In the figure Figure 4.10.4.2, one can see the blue shift of the absorption edge with inclusion of the interface Hamiltonian.

Last update: 31/01/2025

4.10.5 — DEV — Artificial quantum dot crystal - Superlattice dispersion (minibands)

⚠ **Attention**

This tutorial is under construction

Input files:

- *QDSL_Ge-Si_Lazarenkova_JAP_2001_3D_cubic_nnp.in*
- *QDSL_Ge-Si_Lazarenkova_JAP_2001_3D_tetragonal_nnp.in*

Scope:

- In this tutorial, the superlattice energy dispersion for artificial crystals consisting of quantum dots (QDs) are calculated. The QDs are made of Ge embedded in Si. The simulations are performed for cubic and tetragonal QDs.
- This tutorial aims to reproduce figure 2 and 6 in *[Lazarenkova2001]*.

Output files:

- *bias_00000\Quantum\Dispersion\dispersion_quantum_region_HH_along_100.dat*
- *bias_00000\Quantum\Dispersion\dispersion_quantum_region_HH_along_110.dat*
- *bias_00000\Quantum\Dispersion\dispersion_quantum_region_HH_along_111.dat*

Cubic Quantum Dots

The QDs have a cubic shape with $L_x = L_y = L_z = 6.5\,\mathrm{nm}$ and are separated by a distance of $H_x = H_y = H_z = 1.5\,\mathrm{nm}$. We model only one QD and assume periodic boundary conditions along the x, y and z direction giving a superlattice period of $d_x = d_y = d_z = 8\,\mathrm{nm}$ ($d_i = L_i + H_i$).

```
global{
    ...
    periodic{
        x = yes
        y = yes
        z = yes
    }
}
```

The single-band Schrödinger equation is solved for the valence band only (heavy hole). The valence band offset is assumed to be VBO = 0.45 eV, i.e. assuming the valence band edge of the QD is at +0.45 eV, the valence band edge of the barrier is at 0 eV. The energy dispersion relation is calculated along the [100], [110] and [111] direction.

```
quantum{
    ...
    HH{
        num_ev = $num_states
        dispersion{
           output_dispersions{}
```

(continues on next page)

(continued from previous page)

```
            path{
               name = "along_100"
               point{ k = [0.0, 0.0, 0.0] }
               point{ k = [1.0, 0.0, 0.0] }
               num_points = $num_k_points
            }
            path{
               name = "along_110"
               point{ k = [0.0, 0.0, 0.0] }
               point{ k = [1.0, 1.0, 0.0] }
               num_points = $num_k_points
            }
            path{
               name = "along_111"
               point{ k = [0.0, 0.0, 0.0] }
               point{ k = [1.0, 1.0, 1.0] }
               num_points = $num_k_points
            }
        }
    }
}
```

Results

Figure 4.10.5.1 shows the calculated dispersion relation along the [100] direction. The figure agrees very well with Fig. 2(a) of the paper by Lazarenkova et.al.*[Lazarenkova2001]* in the energy region where the confinement inside the QD is strong. For the higher lying states inside the QD and above the QD, our results differ because we use the correct 3D QD confinement potential whereas Lazarenkova et.al.*[Lazarenkova2001]* approximated the potential landscape with an analytical ansatz that allows for the separation of the x, y and z variables. (This ansatz is only justified for states confined deep inside the QD.) The right part of the figure shows schematically the valence band edge of the QD with the energy levels of the single, uncoupled QD.

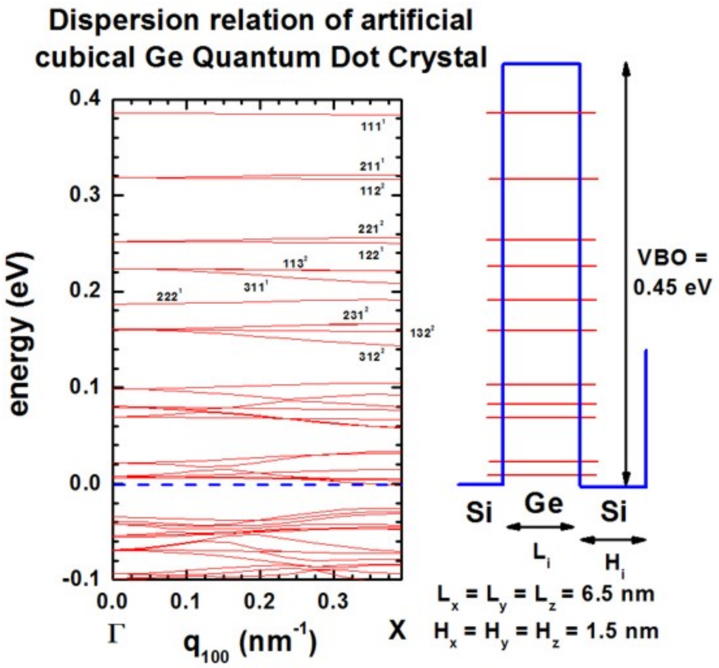


Figure 4.10.5.1: (left) Superlattice dispersion along [100] for heavy-hole states in an artificial cubic Ge quantum dot crystal and (right) valence-band profile through the center of the QD and eigenstates of the lowest heavy-hole states.

In all figures, the eigenstates are labeled with the quantum numbers n_x, n_y, n_z, e.g. 111. The superscript refers to their degeneracy. At $K_{\mathrm{SL}} = 0$, the degeneracy is higher than at nonzero KSL vectors where the symmetry in the Brillouin zone is lower. (The superlattice vector $K_{\mathrm{SL}} = 0$ is often denoted as q.)

The following figures (Figure 4.10.5.2 and Figure 4.10.5.3) show the calculated dispersion relations along the [110] and [111] directions, resepectively. The agreement to Fig. 2(b) and 2(c) of *[Lazarenkova2001]* is again very good for the states that lie deep inside the QD (see also comments above). Note that the eigenstates along the [111] direction show a higher degree of degeneracy throughout the superlattice Brillouin zone as compared to [100] and [110].

Both, the QD itself and the QD superlattice have the same cubic symmetry in this example. Thus the degeneracy of the 123 (incl. permutations) energy band is sixfold throughout the Brillouin zone along the [111] directions (as shown in Figure 4.10.5.3).

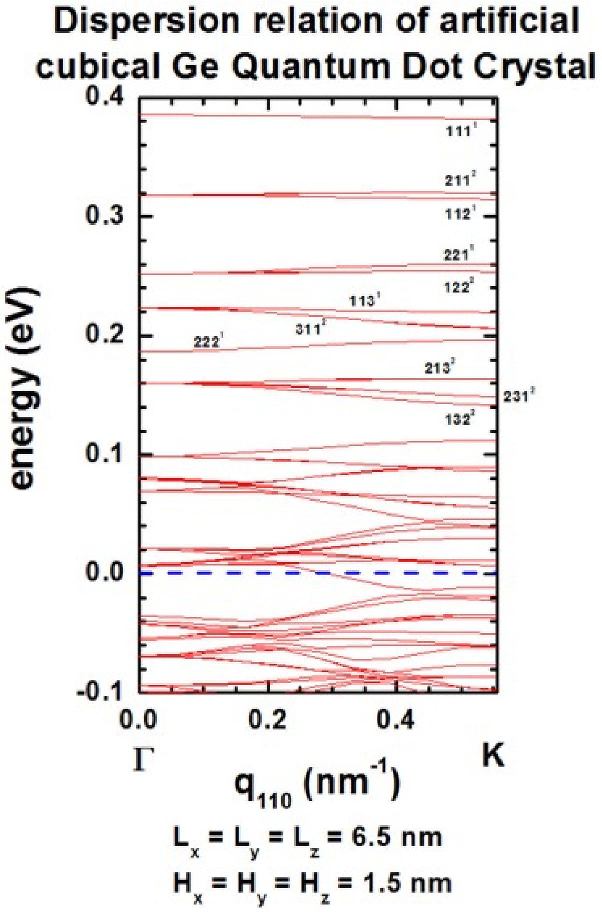


Figure 4.10.5.2: Superlattice dispersion along [110] for heavy-hole states in an artificial cubic Ge quantum dot crystal.

Tetragonal Quantum Dot

The QD has a tetragonal shape with $L_x = L_y = 5$ nm and $L_z = 2.5$ nm and are separated by a distance of $H_x = H_y = 2.5$ nm and $H_z = 1.25$ nm. This gives a superlattice period ($d_i = L_i + H_i$) of $d_x = d_y = 7.5$ nm and $d_z = 3.75$ nm. The grid spacing was chosen to 0.25 nm in x, y and z direction, i.e. 30 grid points in x and y direction, 15 grid points in z direction. Therefore, the size of Schrödinger matrix to be solved is $30 \cdot 30 \cdot 15 = 13500$. All other assumptions are the same as for the cubic QD example above.

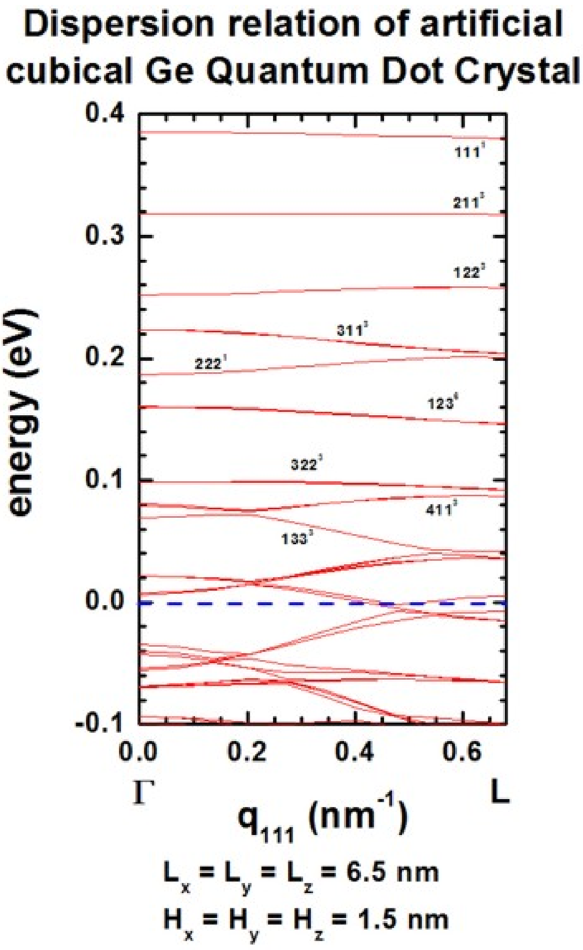


Figure 4.10.5.3: Superlattice dispersion along [111] for heavy-hole states in an artificial cubic Ge quantum dot crystal.

Results

Figures show the dispersion along the [100], [110] and [111] directions, resepectively. All results are in very good agreement to Fig. 6 of *[Lazarenkova2001]*.

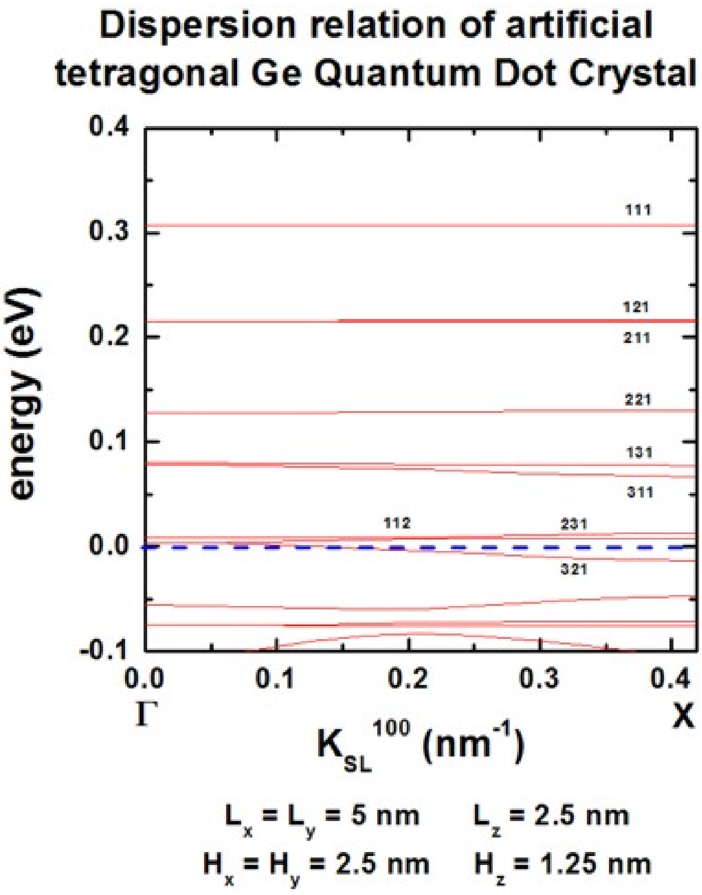


Figure 4.10.5.4: Superlattice dispersion along [100] for heavy-hole states in an artificial tetragonal Ge quantum dot crystal.

This tutorial also exists for *nextnano³*.

Last update: 17/07/2024

4.11 Cascade Structures

4.11.1 Simple quantum cascade structure

Input Files:

- *1DQCL_simple_nnp.in*

In this tutorial we simulate a simple quantum cascade structure that has been presented in an article by Capasso et al. (Figures 12 (b) and 16 (b) of *[CapassoIEEE1986]*).

We can generate the following picture that is based on Fig. 3 of *[BirnerPhotonikInt2008]* and *[BirnerPhotonik2008]*.

It shows the conduction band edge profile of an $Al_{0.48}In_{0.52}As/In_{0.53}Ga_{0.47}As$ superlattice at an electric field of -89 kV/cm. The single-band effective-mass Schrödinger equation is solved for this band profile. The wave functions (ψ^2) of this quantum cascade structure are shown.

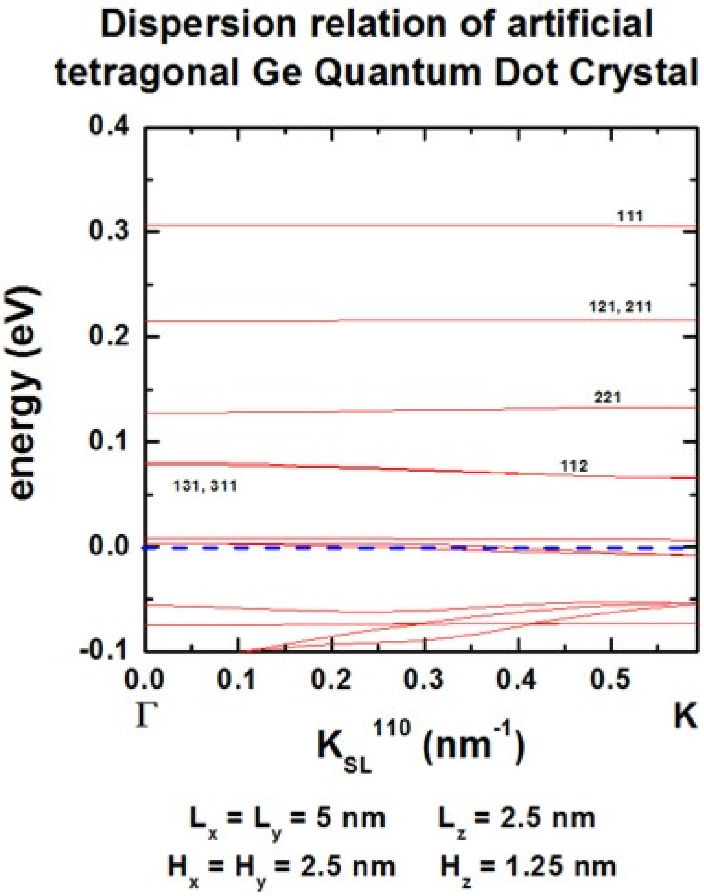


Figure 4.10.5.5: Superlattice dispersion along [110] for heavy-hole states in an artificial tetragonal Ge quantum dot crystal.

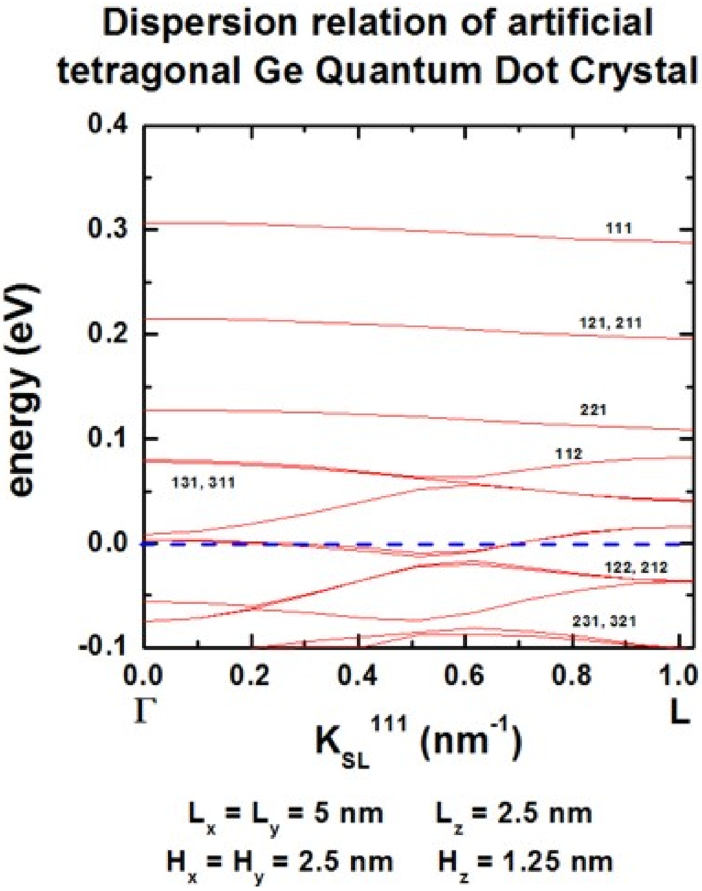
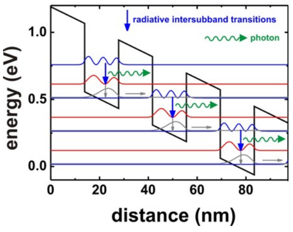


Figure 4.10.5.6: Superlattice dispersion along [111] for heavy-hole states in an artificial tetragonal Ge quantum dot crystal.



The basic idea of such a structure is to depopulate the lowest eigenstate of each quantum well efficiently by bringing it into resonance with the third eigenstate of the next quantum well (resonant tunneling).

The transition second eigenstate → lowest eigenstate should be a nonradiative intersubband transition.

On the other hand, the transition third eigenstate → second eigenstate should be a radiative intersubband transition, i.e. a photon is emitted.

Another important condition for a quantum cascade laser is **population inversion**, i.e. the occupation of the third eigenstate must be much higher than the occupation of the second eigenstate and lowest eigenstate.

- The input file *1DQCL_simple_nnp/*nn3.in* should be rather intuitive and self-explanatory. Documentation for each keyword and each specifier can be found here: *Keywords*
- In the *nextnano++* sample file, the electric field is applied by specifying the keyword `contacts` as follows:

  ```
  contacts{
      charge_neutral{
      name = "leftgate"
      bias = 0.0
      }
      charge_neutral{
      name = "rightgate"
      bias = 1.36081          # corresponds to electric field of F = -89␣
  ↪kV/cm
      }
  }
  ```

  In the keyword `structure`, "leftgate" is defined at `x = [-1, 0]` and "rightgate" is at `x = [152.9, 153.9]`. Thus the electric field applied by this specification is -1.36081 [V] / 152.9 [nm] = -89 [kV/cm]
- Alternatively, we can apply a constant electric field by providing a value for the field.

  ```
  poisson{
     electric_field{ strength = -89e5 } # [V/m]
     output_potential{}
     output_electric_field{}
  }
  ```

Output

The output files are ASCII files.

Bandedges

The conduction and valence band edges can be found in the following file:

- *bias_0000/Quantum/bandedges.dat*

If one plots the conduction band profile, one gets the following figure.

There are six $Al_{0.48}In_{0.52}As$ barriers and five $In_{0.53}Ga_{0.47}As$ wells. The conduction band offset is 0.51 eV.

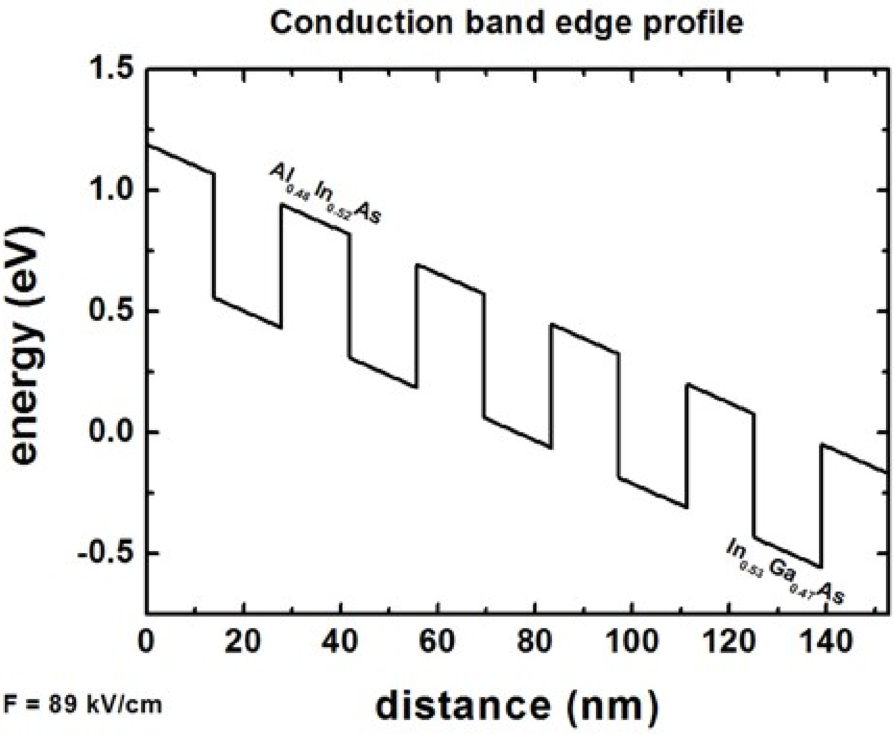
Eigenvalues

The 40 eigenvalues that were calculated can be found in these files. The units are [eV].

- *bias_0000/Quantum/wf_energy_spectrum_quantum_region_Gamma_0000.dat*

The eigenvalues are also contained in these files, i.e. the eigenvalues for each grid point

- *bias_0000/Quantum/wf_probabilities_shift_quantum_region_Gamma_0000.dat*



1st column	2nd column	3rd column	…	41st column
grid points in units of [nm]	1st eigenvalue in units of [eV]	2nd eigenvalue in units of [eV]	…	40th eigenvalue in units of [eV]

If one plots these columns (together with the conduction band edge) one obtains the following picture:

Note

The figure shows only the following energy levels: 1,2,3,4,5,9,10,12,16,18,20,26,27,30,37

Wave Functions

The square of the wave functions (ψ^2) of the 40 eigenstates can be found in these files.

- *bias_0000/Quantum/wf_probabilities_shift_quantum_region_Gamma_0000.dat*

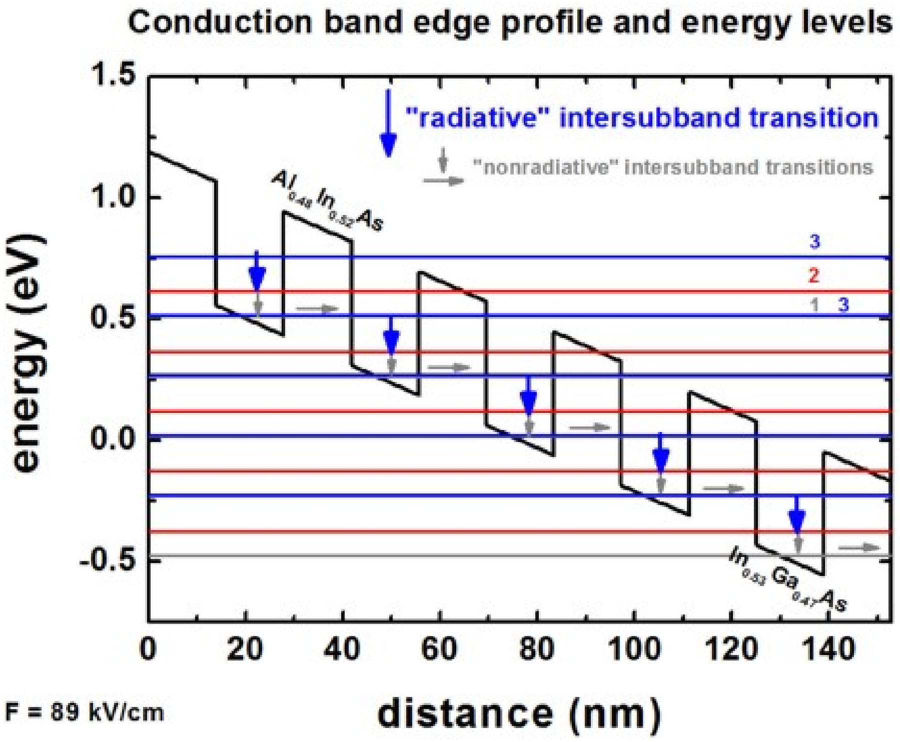
1st column	…	42nd column	43rd column	…	81st column
grid points in units of [nm]	…	$\psi^2$of 1st eigenstate	$\psi^2$of 2nd eigenstate	…	$\psi^2$of 40st eigenstate

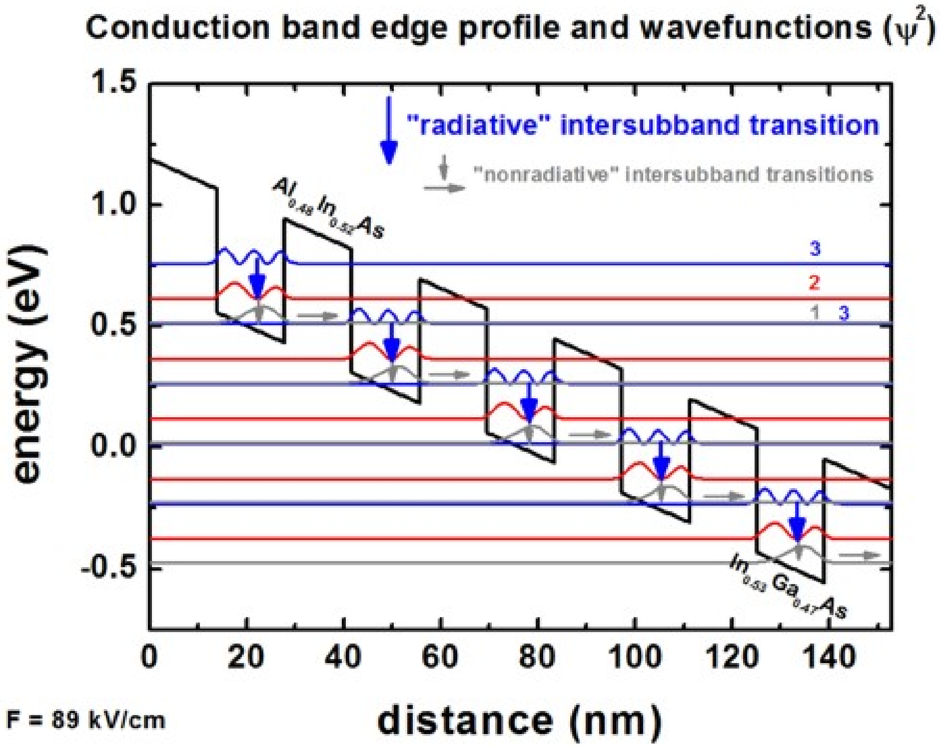
Note

In order to be able to plot the wave functions nicely into the conduction band edge profile, we shift the square of the wave function by its corresponding energy.

If one plots these columns (together with the conduction band edge) one obtains the following picture:

Note


Conduction band edge profile and energy levels
"radiative" intersubband transition
"nonradiative" intersubband transitions
$Al_{0.48}In_{0.52}As$
$In_{0.53}Ga_{0.47}As$
3
2
1 3
energy (eV)
1.5
1.0
0.5
0.0
-0.5
0
20
40
60
80
100
120
140
distance (nm)
F = 89 kV/cm


Conduction band edge profile and wavefunctions (ψ^2)
"radiative" intersubband transition
"nonradiative" intersubband transitions
$Al_{0.48}In_{0.52}As$
$In_{0.53}Ga_{0.47}As$
3
2
1 3
energy (eV)
1.5
1.0
0.5
0.0
-0.5
0
20
40
60
80
100
120
140
distance (nm)
F = 89 kV/cm

The figure shows only the following wave functions: 1,2,3,4,5,9,10,12,16,18,20,26,27,30,37

Now the lowest eigenstate of each quantum well is in resonance with the third eigenstate of the next quantum well. This leads to the depopulation of the lowest eigenstate of each quantum well.

Photon should be emitted with the radiative intersubband transition 3 $\rightarrow$ 2 whreas 2 $\rightarrow$ 1 should be nonradiative intersubband transition.

Effective masses

The effective masses that were used for each grid point can be found in these files.

- *Structure/charge_carrier_masses.dat*

Note

We need to add the following option into the sample file for *nextnano++*.

```
output{
    material_parameters{
        charge_carrier_masses{ boxes = yes }
        }
    }
```

- 1st column: grid points in units of [nm]
- other columns:
  - effective mass tensor components of Gamma and HH valley in units of [m0]. When we use other valleys for the simulation, then these columns shows the effective mass tensor components in that valleys.

These masses have been calculated from the binaries InAs, GaAs and AlAs for the relevant ternaries, including bowing parameters.

Intersubband matrix elements

Experienced users might be interested in having a look at the intersubband matrix elements.

We can find the intersubband (or intraband) matrix elements p_z, the oscillator strengths and the transition energies by adding the followings into `quantum{ }` in *1DQCL_simple_nnp.in*:

```
intraband_matrix_elemets{
    Gamma{}
    output_matrix_elements = yes
    output_transition_energies = yes
    output_oscillator_strengths = yes
}
```

The relevant output files are

- *bias_0000/Quantum/momentum_matrix_elements_quantum_region_Gamma_100.txt*
- *bias_0000/Quantum/transition_energies_quantum_region_Gamma_Gamma.txt*

See *Optical spectra* for more information on the matrix elements.

This tutorial also exists for *nextnano³*.

Last update: 27/05/2025

4.11.2 Quantum-Cascade Lasers

Input files:

- *examples\quantum_cascade_lasers\1DQuantumCascadeLaser_nnp.in*
- *examples\quantum_cascade_lasers\1DQuantumCascadeLaserSiGe_nnpp.in*
- *examples\quantum_cascade_lasers\1DQCL_AlGaAs_Sirtori_APL73_1998_nnp.in*
- *examples\quantum_cascade_lasers\1DQCL_Andrea_Friedrich_NoInjector_InGaAs_APL86_2005_kp_nnp.in*
- *examples\quantum_cascade_lasers\1DQCL_Andrea_Friedrich_NoInjector_InGaAs_APL86_2005_sg_nnp.in*
- *examples\quantum_cascade_lasers\1DQCL_Rochat_APL81_2002_nnp.in*
- *examples\quantum_cascade_lasers\1DQCL_THz_MIT_Sandia_SemicScTech20_2005_nnp.in*
- *examples\quantum_cascade_lasers\THzQCL_Andrews_Vienna_MatSciEng2008_nnp.in*
- *examples\quantum_cascade_lasers\THzQCL_Andrews_Vienna_MatSciEng2008_nnp_electric_field.in*
- *examples\quantum_cascade_lasers\THzQCL_Andrews_Vienna_MatSciEng2008_nnp_no_repeat.in*

> **Note**
>
> If you want to obtain the input files that are used within this tutorial, please check if you can find them in the installation directory. If you cannot find them, please submit a Support Ticket.

Scope:
This tutorial aims to simulate different quantum-cascade structures proposed in the literature.

GaAs/ AlGaAs Quantum-Cascade Laser

This tutorial is based on the quantum-cascade structure that has been presented in *[Page2001]*. Here, we are trying to reproduce fig. 1 of this paper. The corresponding input file is *1DQuantumCascadeLaser.in*.

The quantum-cascade structure consists of a sequence of $GaAs$ wells and $Al_{0.45}Ga_{0.55}As$ barriers. The sequence is as follows (from 0 nm to 45 nm; it is repeated outside this region):

	Layer	Thickness [nm]
1	$Al_{0.45}Ga_{0.55}As$	4.6
2	$GaAs$	1.9
3	$Al_{0.45}Ga_{0.55}As$	1.1
4	$GaAs$	5.4
5	$Al_{0.45}Ga_{0.55}As$	1.1
6	$GaAs$	4.8
7	$Al_{0.45}Ga_{0.55}As$	2.8
8	$GaAs$	3.4
9	$Al_{0.45}Ga_{0.55}As$	1.7
10	$GaAs$	3.0
11	$Al_{0.45}Ga_{0.55}As$	1.8
12	$GaAs$	2.8
13	$Al_{0.45}Ga_{0.55}As$	2.0
14	$GaAs$	3.0
15	$Al_{0.45}Ga_{0.55}As$	2.6
16	$GaAs$	3.0

In *[Page2001]*, a conduction band offset of 390 meV was used. Consequently, we modify our default band offset by shifting the AlGaAs ternary to get a 390 meV offset. We also apply an electric field of -48 kV/cm.

```
$ElectricField      = 48e5    # Electric field in units of [V/m] - Here: 48 kV/cm
$ReferencePotential = 0.092   # Set the potential at the leftmost point of the grid
```

For simplicity, in contrast to *[Page2001]*, we do not include doping here. In the original paper, the areas between 15.2 nm and -5.6 nm (9.8 nm) and 29.8 nm and 39.4 nm (9.8 nm), corresponding to layer 11 - 14, were n-type doped with silicon, with a sheet density of n_{Si} = 3.8 · 10^{11} cm^{-2}. In this example, we do not have to calculate the strain, because piezo and any pyroelectric fields do not exist. We use single-band (effective-mass) rather than 8-band k.p model.

Bandedge profile

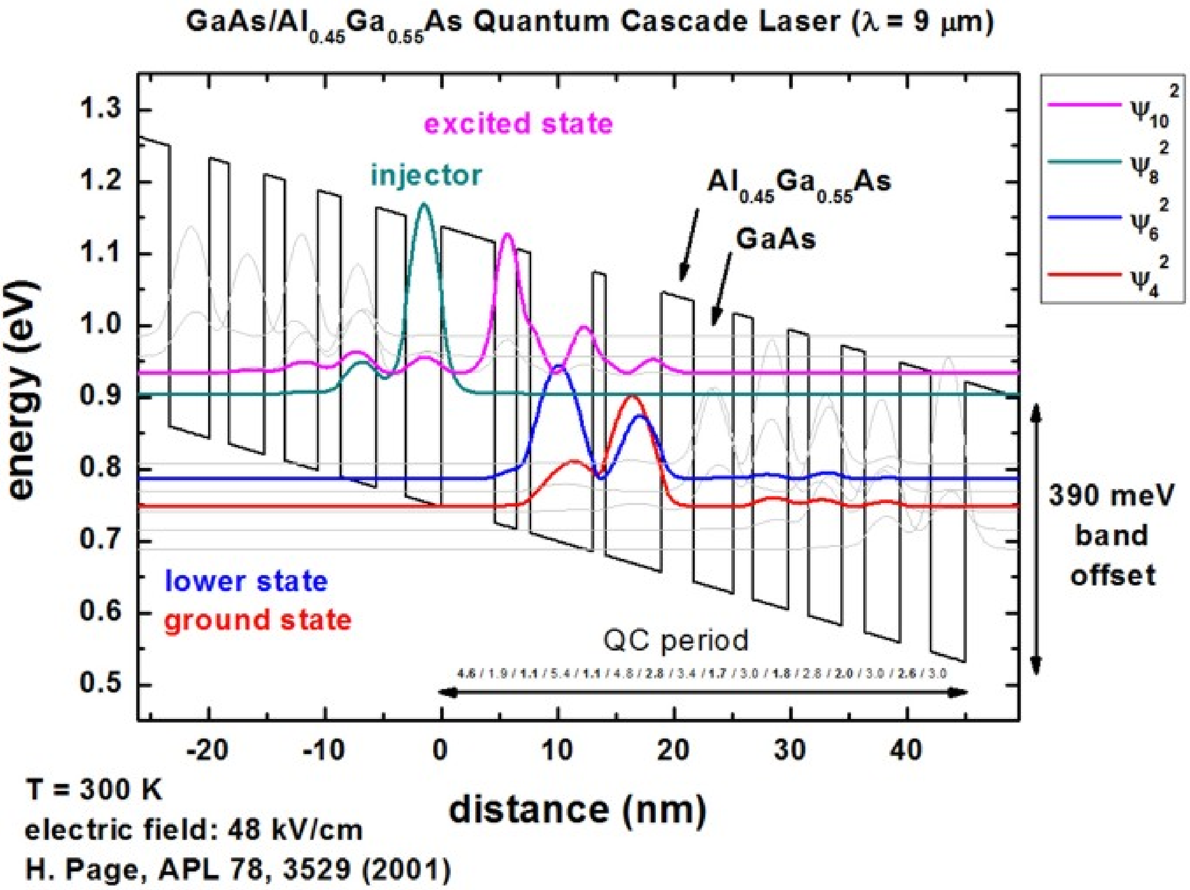


Figure 4.11.2.1: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 48 kV/cm applied. Also shown are the probability densities (Ψ^2) of four electron states, which are shifted by their corresponding eigenenergies.

Figure 4.11.2.1 shows the conduction band energy of the Gamma conduction band edge and the probability densities (Ψ^2) of the ground state 4 (red), the lower state 6 (blue), the excited state 10 (pink) and the injector state 8 (green). The above shown structure of the conduction band edge and the wave functions is in excellent agreement with fig. 1 of *[Page2001]*.

Note that periodic boundary conditions for the Schrödinger and Poisson equation do not make sense because of the application of an electric field. Thus, we used Dirichlet boundary conditions. However, this will lead to some artificial, wrong wave functions at the boundaries because the wave function is forced to be zero at the boundaries. For the states in the middle of the device where the wave function decays to zero in any case at the boundaries, the boundary conditions do not have any influence at all and so these states are fine. So the suggestion is to calculate 3 or 5 periods, and then take the energy levels and wave functions of the center period. In this way, boundary effects should not be very severe.

```
global{
    periodic{ x = yes }  # apply period boundary conditions along the x-direction
}
```

Dipole matrix elements

The files:

- *bias_00000\Quantum\overlap_integrals_qr1_Gamma_100.txt*
- *bias_00000\Quantum\dipole_moment_matrix_elements_qr1_Gamma_100.txt*

contain the p_x and z matrix elements for all transitions. Our result for the z matrix element for the transition between the excited state to lower state is in excellent agreement with the result of *[Page2001]*:

	nextnano³	*[Page2001]*
$\langle\Psi_{10}\|z\|\Psi_6\rangle$	$z_{10,6}$ = 1.6655138016 nm	$z_{3,2}$ = 1.7 nm
$\Delta E_{\text{transition}}$	147.7 meV	160 meV

QCL examples

Note

Please submit a support ticket if you want to obtain the input files for the following structures.

1. λ = 9 μm, i.e. 33 THz or 138 meV

The simulated QCL structure is taken from *[Page2001]*, see Figure 4.11.2.1. The corresponding input is *1DQuantumCascadeLaser.in*.

2. λ = 9.4 μm or 132 meV

The simulated quantum-cascade structure, shown in Figure 4.11.2.2, is based on *[Sirtori1998]*. The corresponding input file is *1DQCL_AlGaAs_Sirtori_APL73_1998.in*.

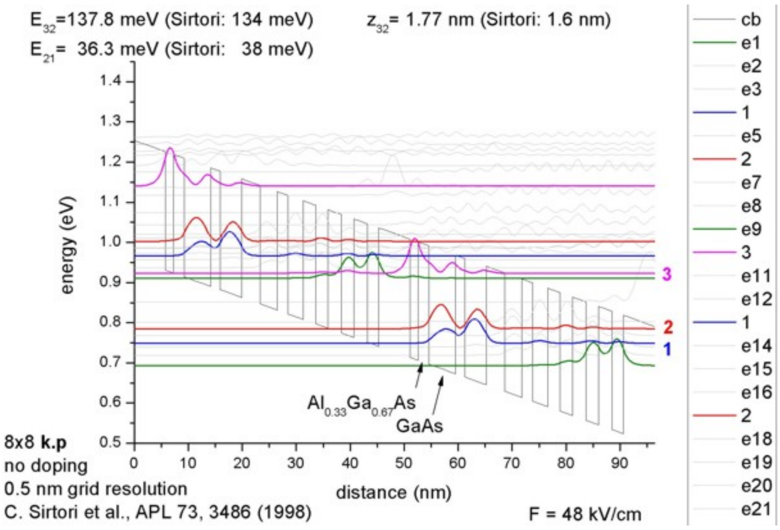


Figure 4.11.2.2: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 48 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

3. λ = 10 μm or 124 meV (77 K)

The simulated quantum-cascade structure, shown in Figure 4.11.2.3 and Figure 4.11.2.4, is based on *[Friedrich2005]*. The corresponding input file is *1DQCL_Andrea_Friedrich_NoInjector_InGaAs_APL86_2005_kp.in*.

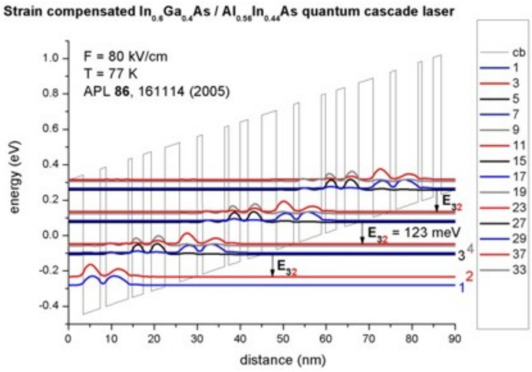


Figure 4.11.2.3: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 80 kV/cm applied (T = 77 K). Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

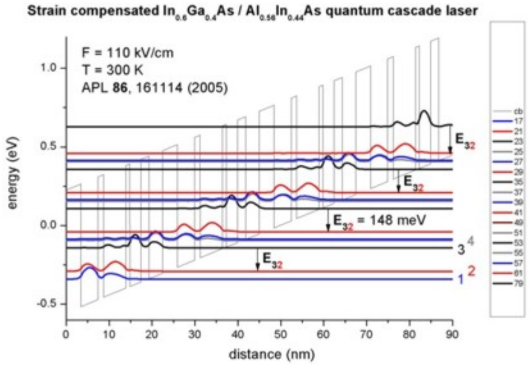


Figure 4.11.2.4: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 110 kV/cm applied (T = 300 K). Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

4. λ = 66 μm, i.e. 4.54 THz or 18.8 meV

The simulated quantum-cascade structure, shown in Figure 4.11.2.5, is based on *[Rochat2002]*. The corresponding input file is *1DQCL_Rochat_APL81_2002.in*.

5. λ = 89.2 μm, i.e. 3.4 THz or 13.9 meV

The simulated quantum-cascade structure, shown in Figure 4.11.2.6, is based on *[Hu2005]*. The corresponding input file is *1DQCL_THz_MIT_Sandia_SemicScTech20_2005.in*.

6. λ = 107 μm, i.e. 2.8 THz or 11 meV

The simulated quantum-cascade structure, shown in Figure 4.11.2.7, is based on *[Andrews2008]*. The corresponding input file is *THzQCL_Andrews_Vienna_MatSciEng2008_nnp.in*.

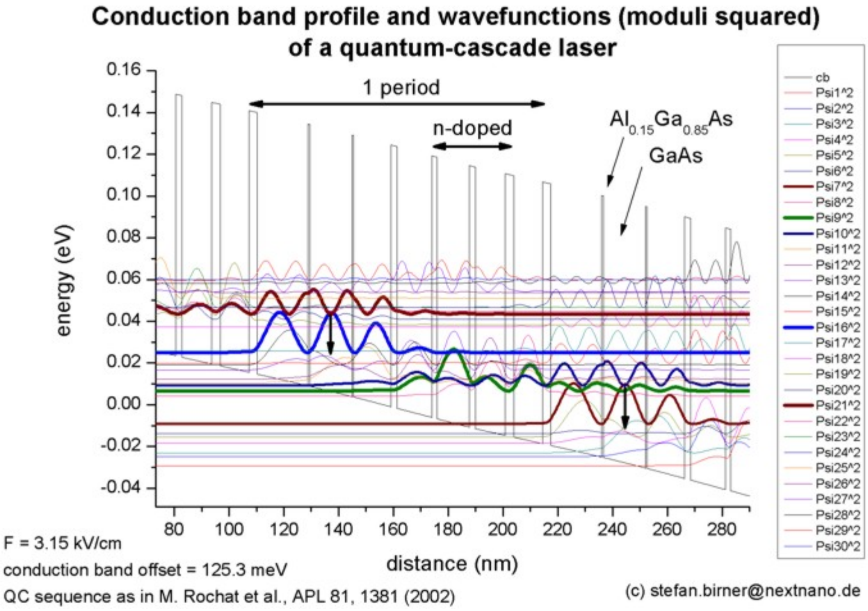


Figure 4.11.2.5: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 3.15 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

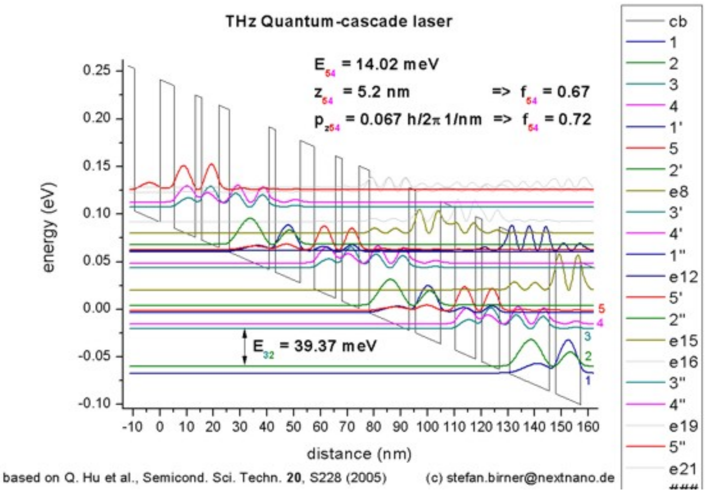


Figure 4.11.2.6: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 12.2 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

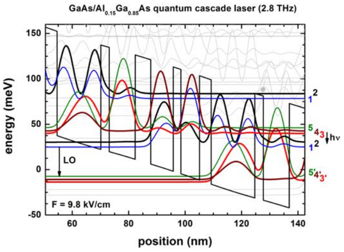


Figure 4.11.2.7: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 9.8 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

7. λ = 9.9 μm, i.e. 30.2 THz or 125 meV

The simulated quantum-cascade structure, shown in Figure 4.11.2.8, is based on *[Dehlinger2000]*. This corresponding input file is *1DQuantumCascadeLaserSiGe_nnpp.in*.

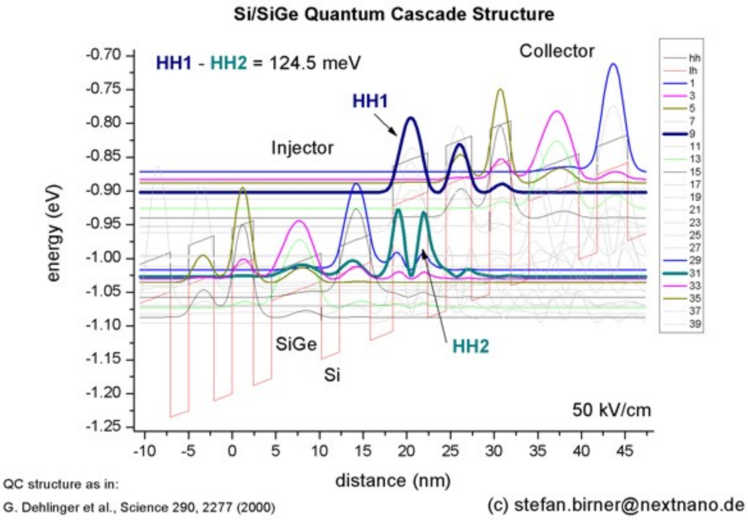


Figure 4.11.2.8: Calculated valance band edge (black line) of the quantum-cascade structure with electric field of strength 50 kV/cm applied. Also shown are the probability densities (Ψ^2) of several hole states, which are shifted by their corresponding eigenenergies.

This tutorial also exists for *nextnano³*.

Last update: 27/05/2025

4.12 Optical Spectra and Transitions

4.12.1 Single Particle

Optical absorption for interband and intersubband transitions

Section author: Takuma Sato

Input Files:

- *QWIP_singleQW_GaAs_AlGaAs_nnp.in*
- *QWIP_singleQW_InAs_AlSb_nnp.in*
- *QWIP_Gunapala_JAP_1991_nnp.in*
- *AlGaAs_QW_Frankenberger_Simple_nnp.in*
- *AlGaAs_QW_Frankenberger_Simple_nnp_fast.in*
- *AlGaAs_QW_Frankenberger_Doping_schottky07_nnp.in*
- *AlGaAs_QW_Frankenberger_Doping_schottky07_nnp_fast.in*

Contents

In this tutorial we illustrate the *optics{ }* module to demonstrate what *nextnano++* can simulate for optoelectronic devices. This module performs a detailed calculation to **optical absorption phenomena**, **using 8 (or 6) band $\mathbf{k} \cdot \mathbf{p}$ models**. If you are interested in

- the background physics of this module and how to write the input file, go to *Principle and nextnano++ implementation*.
- the simulation results for intersubband transitions, go to *1D tutorial for intersubband transitions: Quantum well infrared photodetector*.
- the simulation results for interband transitions, go to *1D tutorial for interband transitions: Frankenberger*.
- optical absorption in 2D devices, *(under construction)*
- optical absorption in broken-gap structures, *(under construction)*

This algorithm is implemented based on the following diploma thesis:

- Thomas Eißfeller, *Linear Optical Response of Semiconductor Nanodevices*, Technische Universität München (2008)

For the physics of optical transition in semiconductors and its application, we refer to

- Shun L. Chuang, *Physics of Optoelectronic Devices* (Wiley, 1995)
- S.M. Sze & Kwok K. Ng, *Physics of Semiconductor Devices* (Wiley, 2007)

Principle and nextnano++ implementation

$\mathrm{k}_{||}$ space

In the k.p analysis of one- (or two-) dimensional structures we have a projection of the Bloch wave vector along translation-invariant directions. We denote them as $\mathbf{k}_{||} = k_y\hat{y} + k_z\hat{z}$ (1D) and $\mathbf{k}_{||} = k_z\hat{z}$ (2D). Under envelope function approximation the $\mathbf{k}\cdot\mathbf{p}$ model yields the following equation to determine the confined states in structured directions

$$\sum_{\mu=1}^{8} H^{\mathrm{kp8}}_{\nu\mu}(\mathbf{k}_{||}, \mathbf{r}_{\perp}) f_{m,\mu}(\mathbf{r}_{\perp}) = E_m(\mathbf{k}_{||}) f_{m,\nu}(\mathbf{r}_{\perp}) \quad (\nu = 1, \dots, 8), \tag{4.12.1.1}$$

where the Greek indices label the k.p bands and m denotes eigenvalues, $\mathbf{r}_{\perp} = x\hat{x}$ in 1D and $\mathbf{r}_{\perp} = x\hat{x} + y\hat{y}$ in 2D. H^{kp8} is the 8×8 matrix whose elements are given by the k.p parameters in the database. $f_{m,\mu}(\mathbf{r}_{\perp})$‘ are the envelopes in the structured directions. The full wave function is given at each $\mathbf{k}_{||}$ as

$$\Psi_n(\mathbf{k}_{||}, \mathbf{r}) = \sum_{\mu=1}^{8} F_{m,\mu}(\mathbf{k}_{||}, \mathbf{r}) u_{\mu}(\mathbf{r}) = \sum_{\mu=1}^{8} \frac{e^{i\mathbf{k}_{||}\cdot\mathbf{r}_{||}}}{\sqrt{A}} f_{m,\mu}(\mathbf{r}_{\perp}) u_{\mu}(\mathbf{r}), \tag{4.12.1.2}$$

where $u_{\mu}(\mathbf{r})$ is the Bloch function of the band μ at $\mathbf{k} = 0$ and $A = \int d\mathbf{r}_{||}$. In general, both the conduction band (Γ) and valence bands contribute to this full wave function. The spinor composition is exported to `Quantum\spinor_composition`. After solving this “Schrödinger” equation, the wave function is integrated over a limited region in $\mathbf{k}_{||}$ space to obtain the charge density, which is used in the quantum-current-Poisson iteration. The region is specified under *quantum{ }* as

```
quantum{
  region{
      kp_8band{
        k_integration{
            relative_size = $r_quantum     # size of k||-space in quantum{ }␣
↪(relative to the Brillouin zone)
            num_points    = $N_quantum     # number of k|| points where Schrödinger␣
↪eq. is solved
            num_subpoints = $Nsub_quantum  # number of points between k|| points␣
↪where wave functions and eigenvalues are interpolated
```

(continues on next page)

(continued from previous page)

```
            force_k0_subspace =           # (optional) use the eigenfunctions of the␣
↪Schrödinger equation at k=0 as the basis for the Schrödinger equation at all k-
↪point (default: no)
          }
        }
    }
}
```

Note

When `force_k0_subspace=yes` in `quantum{ }` or `optics{ }`, the Schrödinger equations at non-zero k-points are solved in the subspace of the eigenfunctions obtained by the Schrödinger equation at $\mathbf{k}_{||} = 0$. This approximation largely improves the computational speed. In case you are planning to use this approximation for final results, please make sure to check whether the resulting loss of accuracy in density is acceptable (`quantum{ }`) or the loss in optical spectra is acceptable (`optics{ }`).

Optical absorption spectrum

When 1) Schrödinger equation is solved with k.p method, 2) *optics{ }* flag is present and 3) the specifier `optics{ }` is present under `run{ }` flag, *nextnano*++ calculates the absorption spectrum.

```
optics{
  region{
  ...          # see below for details
  }
}

run{
  quantum{ }
  optics{ }
}
```

The optical absorption accompanied by excitation of charge carriers (state $n \to m$) in a condensed matter is calculated from Fermi's golden rule *[ChuangOpto1995]*. The absorption coefficient has the dimension of (length)$^{-1}$.

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega} \frac{1}{V} \sum_{n>m} \sum_{\mathbf{k}_{||}} |\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathbf{k}_{||})|^2 (f_m - f_n)\delta(E_n - E_m - \hbar\omega), \tag{4.12.1.3}$$

where the first sum runs over bands that fulfill $E_n > E_m$, and $f_m(\mathbf{k}_{||}) = [1 + e^{[E_m(\mathbf{k}_{||}) - E_F]/k_B T}]^{-1}$ is the occupation of eigenstate m. When `optics{ occupation_ignore=yes }` (default is `no`), the program assumes

$$\begin{cases} f_m(\mathbf{k}_{||}) = 0 \;\text{ if }\; m \in \text{conduction band} \\ f_m(\mathbf{k}_{||}) = 1 \;\text{ if }\; m \in \text{valence band} \end{cases}$$

The light polarization $\vec{\epsilon}$ and refractive index n_s are specified in the input file. The refractive index is in general frequency-dependent, but we assume it to be constant and equal to the substrate value.

```
optics{
  region{
      polarization{ name="TM" re = [1,0,0] }    # in 1D simulation, x is the growth␣
↪direction
```

(continues on next page)

(continued from previous page)

```
      polarization{ name="TE" re = [0,1,0] }    # complex (circular) polarization is␣
↪also allowed

      refractive_index =                   # (optional) use alternative value for the␣
↪refractive index (default: substrate value)
   }
}
```

The core of the optical transition is the **optical matrix elements** $\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathbf{k}_\parallel)$ because the kinematic momentum operator $\vec{\pi} = (\pi_x, \pi_y, \pi_z)$ couples linearly to the vector potential that accounts for the electromagnetic field. Here $\vec{\pi}$ is the sum of the conventional momentum operator $\mathbf{p}$ and the contribution of spin-orbit interaction. The optical matrix elements are calculated as

$$\vec{\pi}_{nm}(\mathbf{k}_\parallel) = \langle n|\vec{\pi}|m\rangle = \int d\mathbf{r} \begin{pmatrix} F^*_{n1} & \cdots & F^*_{n8} \end{pmatrix} \begin{pmatrix} & \vec{\pi}^{\mathrm{kp8}}_{\nu\mu} & \end{pmatrix} \begin{pmatrix} F_{m1} \\ \vdots \\ F_{m8} \end{pmatrix}, \tag{4.12.1.4}$$

where the 8×8 matrix representation of the momentum operator, $\vec{\pi}^{\mathrm{kp8}}_{\nu\mu}$, has been derived using the Hellmann-Feynman theorem extended to the 8-band k.p model up to first order in $\mathbf{k}$ [Eißfeller]. For the analysis of the absorption spectrum, *nextnano++* also prints out some fractions of the absorption coefficient formula in the output folder, namely

1. occupation (if `output_occupations=yes`) *\Optics\occupation_~.dat* $f_m(\mathbf{k}_\parallel)$
2. eigenvalue dispersion (if `output_energies=yes`) *\Optics\energy_disp_~.dat* $E_m(\mathbf{k}_\parallel)$
3. transition intensity (if `output_transitions=yes`) *Optics\transition_disp_~.dat* $T_{nm}(\vec{\epsilon}, \mathbf{k}_\parallel) = \frac{2}{m_0}|\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathbf{k}_\parallel)|^2$
4. imaginary part of the dielectric function for each transition (if `output_spectra{ output_components yes }`) *\Optics\imepsilon_~.dat* $\mathrm{Im}\varepsilon_{nm}(\vec{\epsilon}, \omega) = \frac{m_0}{2\omega^2} \frac{\pi e^2}{m_0^2 \varepsilon_0} \frac{1}{V} \sum_{\mathbf{k}_\parallel} T_{nm}(\vec{\epsilon}, \mathbf{k}_\parallel)(f_m - f_n)\delta(E_n - E_m - \hbar\omega)$
5. total imaginary part of the dielectric function *\Optics\imepsilon_~.dat* $\mathrm{Im}\varepsilon(\vec{\epsilon}, \omega) = \sum_{n>m} \mathrm{Im}\varepsilon_{nm}(\vec{\epsilon}, \omega)$
6. total absorption spectrum *\Optics\absorption_~.dat* $\alpha(\vec{\epsilon}, \omega) = \sum_{n>m} \alpha_{nm}(\vec{\epsilon}, \omega) = \sum_{n>m} \frac{\omega}{n_s c} \mathrm{Im}\varepsilon_{nm}(\vec{\epsilon}, \omega)$

The following part of the input specifies how much transitions to be taken into account. The setting for `k_integration{}` is explained in the next section.

```
optics{
  region{
      interband = $INTERBAND    # yes or no
      intraband = $INTRABAND    # yes or no

      energy_min        = $ENERGY_MIN          # minimum energy of the absorption␣
↪spectrum
      energy_max        = $ENERGY_MAX          # maximum energy of the absorption␣
↪spectrum
      energy_resolution = $ENERGY_RESOLUTION   # energy grid spacing

      k_integration{
        relative_size = $r_optics     # size of k||-space in optics{ } (relative to␣
↪the Brillouin zone)
        num_points    = $N_optics     # number of k|| points where transition␣
↪intensities are computed
        num_subpoints = $Nsub_optics  # number of points between k|| points where␣
↪transition intensity is interpolated
        force_k0_subspace =           # (optional) use the eigenfunctions of the␣
```

(continues on next page)

(continued from previous page)

```
↪Schrödinger equation at k=0 as the basis for the Schrödinger equation at all k-
↪point (default: no)
        }
    }
}
```

Parameters in k_integration{} (for fine tuning)

Parameters in `k_integration{}` in `optics{ }` flag (hereafter $r_\text{opt}, N_\text{opt}, N'_\text{opt}$) specify **the size and resolution of the $\mathbf{k}_\parallel$ space integration in absorption spectrum calculation**, $\sum_{\mathbf{k}_\parallel}$. This should not be confused with the specifier `k_integration{}` in *quantum{ }* flag used for **quantum mechanical charge density integration** (hereafter r_q, N_q, N'_q, see Figure 4.12.1.1).

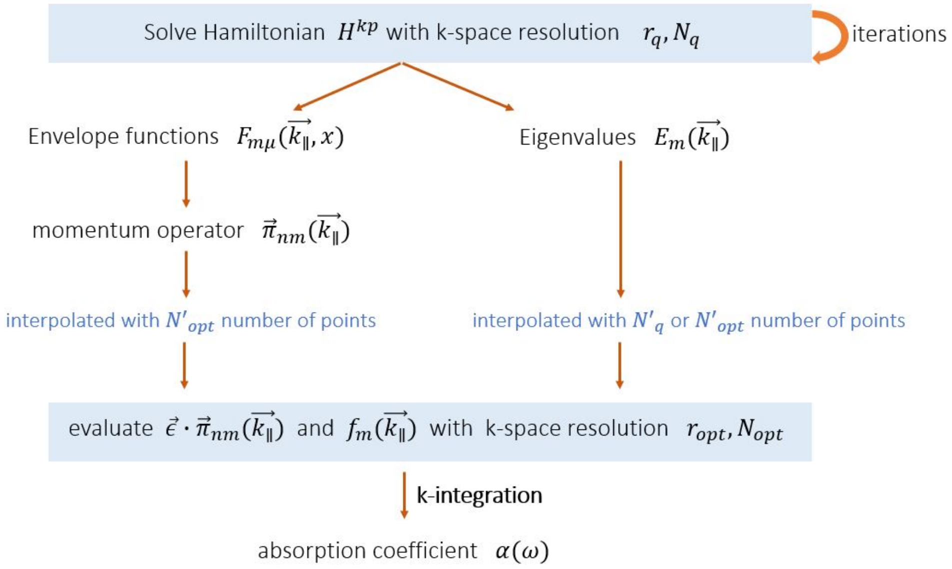
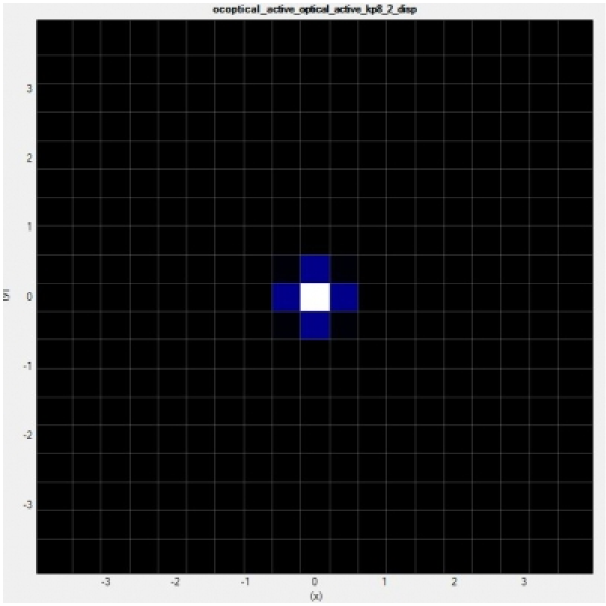


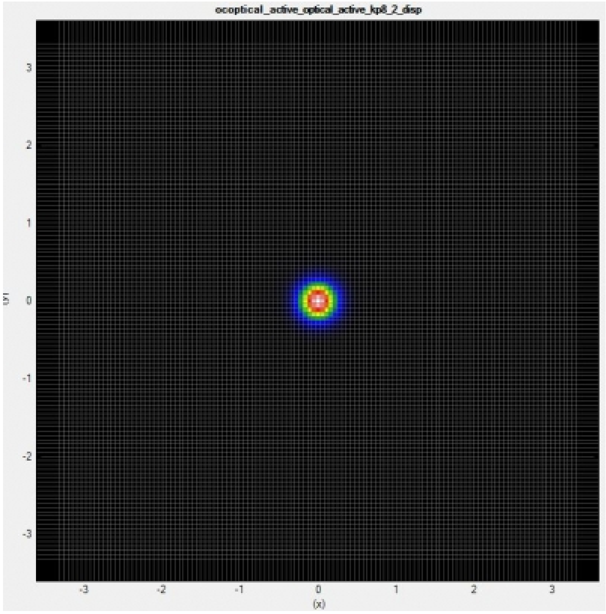
Figure 4.12.1.1: Calculation algorithm of optical absorption spectrum and its relation to the parameters in `k_integration{}`. r_q, N_q, N'_q and $r_\text{opt}, N_\text{opt}, N'_\text{opt}$ are specified in `quantum{ }` and `optics{ }`, respectively. *To do; the energy dispersion is interpolated with* N'_q *or* N'_opt*?*

First we discuss the parameters r_opt and N_opt. The size of k|| space may affect the validity of simulation results. It also determines the simulation load. Here are some hints to determine the appropriate parameter sets:

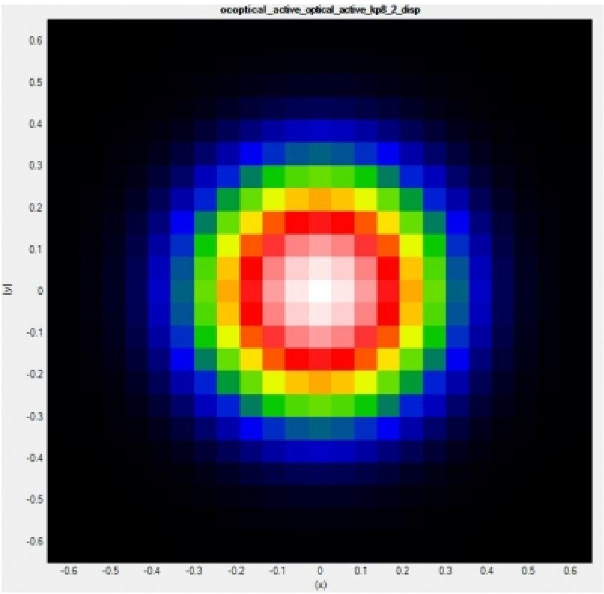
- In undoped systems, integrating up to $|\mathbf{k}_\parallel|$ that gives in-plane kinetic energy $\hbar^2 k_\parallel^2/2m$ corresponding to $2k_B T$ or $3k_B T$ should be sufficient. Usually $r_\text{opt} = 0.3$ is sufficiently large to include all occupied states. In doped systems, it depends on the Fermi energy.
- To see the range of occupied states in $\mathbf{k}_\parallel$ space, run a simulation and look at the output `\Optics\occupation_~.dat`. We recommend checking the box "Show grid" on the left panel in Output tab of *nextnanomat*. This shows the occupation $f_m(\mathbf{k}_\parallel)$ as a function of $\mathbf{k}_\parallel$. Let us consider 1D simulation and suppose you got the following:

where $(r_\text{opt}, N_\text{opt}) = (0.3, 8)$. The horizontal- and vertical axes are k_y and k_z, respectively. The area $|k_{y,z}| \leq r_\text{opt}\frac{\pi}{a}$ is shown with the k||-space gridding (thin white lines). The number of k|| points in one direction is $2N_\text{opt} + 3$. The occupation profile is not smooth, and you might want a higher resolution by increasing the parameter $(r_\text{opt}, N_\text{opt}) \rightarrow (0.3, 60)$:


ocoptical_active_optical_active_kp8_2_disp
3
2
1
0
-1
-2
-3
(y)
-3
-2
-1
0
1
2
3
(x)


ocoptical_active_optical_active_kp8_2_disp
3
2
1
0
-1
-2
-3
(y)
-3
-2
-1
0
1
2
3
(x)

The occupation becomes smooth, but at the same time this significantly increases the number of k points (in 1D simulation, (the number of k points)$\propto (r_{\mathrm{opt}} N_{\mathrm{opt}})^2$). Noting that the black region, where occupation is zero, does not contribute to the absorption, you can “zoom in” to the colored region by decreasing r_{opt} and N_{opt} in such a way that the ratio $r_{\mathrm{opt}}/N_{\mathrm{opt}}$ remains constant. This will cut down the irrelevant region without changing the resolution. For example, if you set $(r_{\mathrm{opt}}, N_{\mathrm{opt}}) = (0.05, 10)$, you obtain



and this should be sufficient for the k||-space integration.

After tuning the parameters $r_{\mathrm{opt}}, N_{\mathrm{opt}}$, we can further optimize the setting regarding the interpolation. The number of subpoints N'_{opt} determines at how many k|| points the transition intensity should be interpolated. Increasing N'_{opt} gives $E_m(\mathbf{k}_{\|})$ of higher resolution and makes the absorption spectrum smooth. Figure 4.12.1.2 shows that this parameter improves the absorption spectrum.

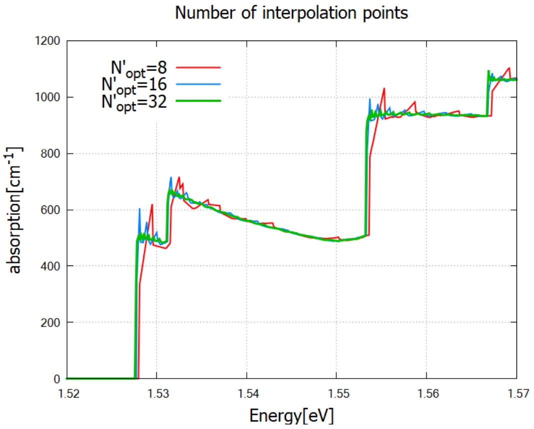


Figure 4.12.1.2: The effect of the parameter N'_{opt} specified in `optics{ k_integration{}}` on absorption spectrum output *\Optics\absorption*. Larger N'_{opt} smoothens the k||-dependence of the integrand, which leads to smoother spectrum.

To do: investigate `spin_degeneracy=yes/no` *and* `dipole_approximation = yes/no`

1D tutorial for intersubband transitions: Quantum well infrared photodetector

In the following we apply the formalism to several devices. As a first example, we model the absorption spectrum of an AlGaAs/GaAs quantum well infrared photodetector (QWIP). The QWIP is based on photoconductivity due to intersubband excitation.

Input files

- *QWIP_singleQW_GaAs_AlGaAs_nnp.in*
- *QWIP_singleQW_InAs_AlSb_nnp.in*
- *QWIP_Gunapala_JAP_1991_nnp.in*

The first example uses the same parameters used in

- FIG. 20 in B.F. Levine, J. Appl. Phys. 74 (8), 15 (1993),

while the third example is based on *[GunapalaJAP1991]*

GaAs/AlGaAs single QW - band structure, eigenstates and absorption

We first illustrate the first example *QWIP_singleQW_GaAs_AlGaAs_nnp.in*. In this example, we model optical absorption in single quantum well structure. The following input is required for self-consistent quantum-current-Poisson simulation:

```
quantum{
  region{
      name = "optical_active"
      no_density = no
      kp_8band{
        num_electrons = $OptNumE
        num_holes     = $OptNumH
      }
  }
}

poisson{ }

current{}

run{
  strain{ }             # strain calculation
  current_poisson{ }
  quantum_current_poisson{ }
  optics{ }             # absorption calculation
}
```

The specifier `no_density=no` lets the program calculate quantum mechanical charge density (default). Current-Poisson equation takes over this value. The band structure and wave functions are shown in Figure 4.12.1.3 and Figure 4.12.1.4, respectively.

The output folder `\Optics` contains computed absorption spectra. Let us first check the occupation $f_m(\mathbf{k}_{\|})$ used in the calculation. When comparing the results `\Optics\occupation`, please mind the autoscale mode of *nextnanomat*:

The autoscale mode in *nextnanomat* is set off here. We clearly see that the first state is well occupied, whereas for the second state is not (precisely speaking $f_1(0)$=0.897 while $f_2(0)$<0.07).

The absorption coefficient for TE ($\vec{\epsilon} = \hat{y}$) and TM ($\vec{\epsilon} = \hat{x}$) light polarization is shown in Figure 4.12.1.7. The energy grid spacing here is `$ENERGY_RESOLUTION`=0.5meV. For single-band models the peak becomes very sharp unless one introduces phenomenological broadening function such as Lorentzian. In k.p calculation, in contrast,

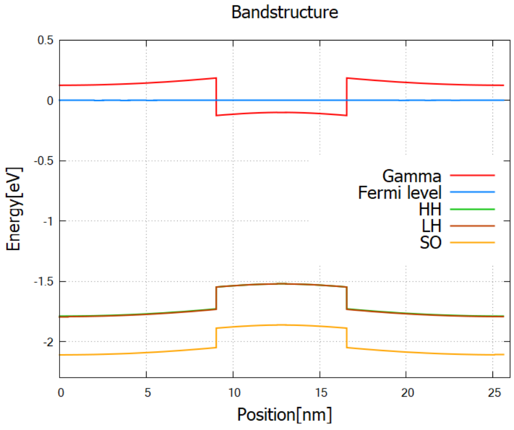


Figure 4.12.1.3: Single quantum well structure `\bandedges.dat`. The bias voltage between two contacts is set to 2mV.

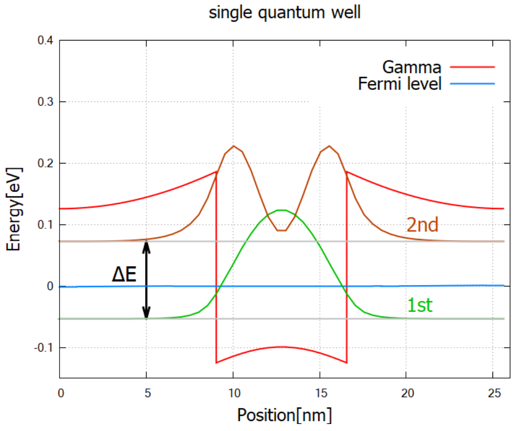


Figure 4.12.1.4: Probability distribution $|\psi(x)|^2$ of the confined states at $\mathbf{k}_{||} = 0$ (`\Quantum\probabilities_shift_optical_active`). The wave functions here are the solution to the 8-band k.p model. The energy separation is ΔE=0.06960-(-0.05589)=0.1255[eV] according to the output data. The electron Fermi energy lies between two bound states.

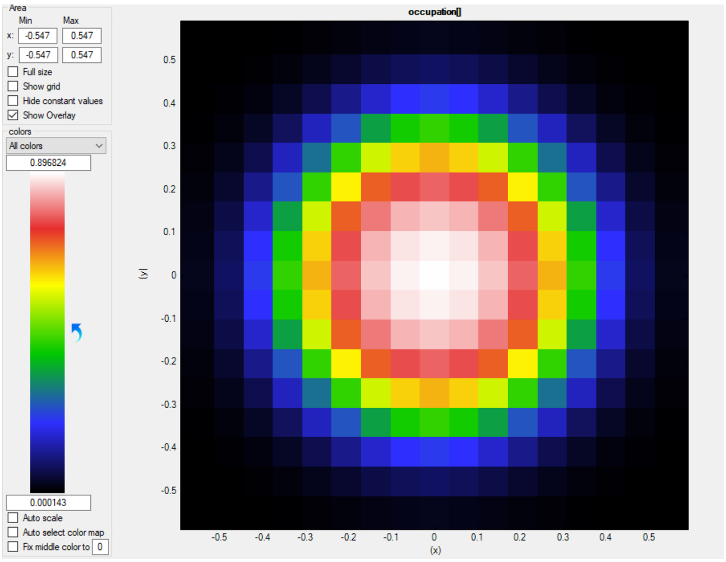

Area
Min Max
x: -0.547 0.547
y: -0.547 0.547
☐ Full size
☐ Show grid
☐ Hide constant values
☑ Show Overlay
colors
All colors
0.896824
0.000143
☐ Auto scale
☐ Auto select color map
☐ Fix middle color to 0

Figure 4.12.1.5: Occupation of the first (m=1) bound states as a function of $\mathbf{k}_{||}$.

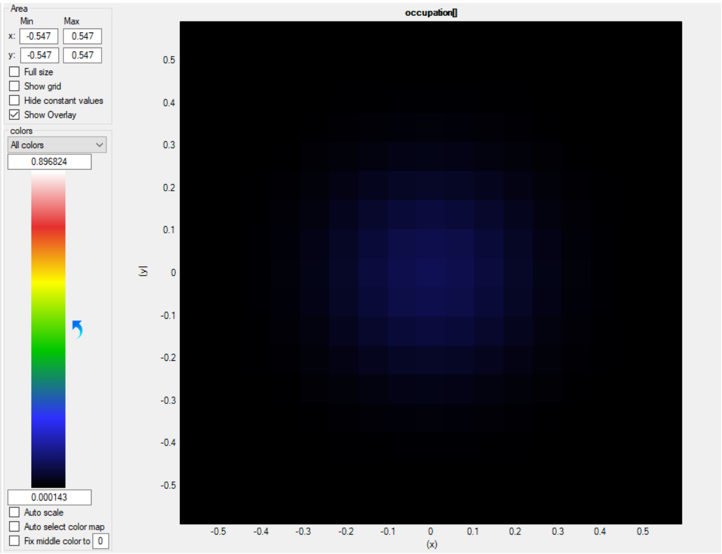


Figure 4.12.1.6: Occupation of the second (m=2) bound states as a function of $\mathbf{k}_{||}$.

peaks gets broadened because the transition energies, $E_n(\mathbf{k}_{||}) - E_m(\mathbf{k}_{||})$, depends on k||. One can confirm this by comparing the output `\Optics\energy_disp_~.dat` for states m=1 and 2 (not shown). In intersubband transitions the transition energies can be concave downward in k|| space, i.e., $E_n(\mathbf{k}_{||}) - E_m(\mathbf{k}_{||}) \propto -k^2$, depending on the masses. In the present case the absorption spectrum has a tail in the region $\hbar\omega < \Delta E$.

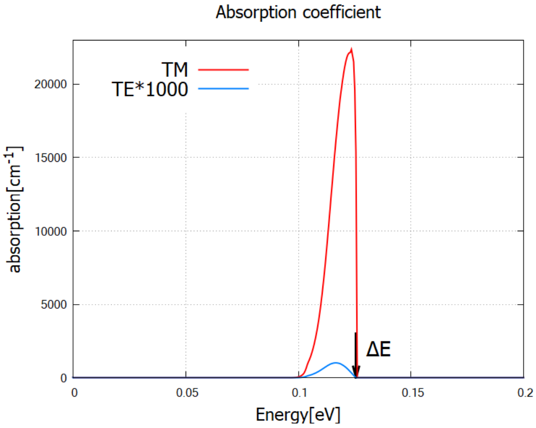


Figure 4.12.1.7: Absorption coefficient in `\Optics\absorption_~.dat` as a function of photon energy, for TE and TM. Black arrow points the energy separation ΔE. The broadening of the spectrum is due to the k||-dependence of wave functions and corresponding eigenvalues.

The optical transitions between conduction band states (intersubband transitions) in response to TE-polarized light is only allowed when eigenstates have finite spinor components in valence bands. In the present case its large band gap and small confinement leads to small band-mixing, rendering TE absorption spectrum orders of magnitude smaller than TM polarization (Figure 4.12.1.7). As seen in the output `\Quantum\spinor_composition_~.dat`, eigenstates contain approximately 98% contribution from conduction band and 2% from valence band.

InAs/AlSb single QW - small band gap & large confinement

In the second example *QWIP_singleQW_InAs_AlSb_nnp.in*, single quantum well is narrower and the band gap is smaller than the first example. The small band gap and large confinement of the wave function (Figure 4.12.1.8) leads to large band mixing. In fact, the output `\Quantum\spinor_composition_~.dat` shows that the ground states in Figure Figure 4.12.1.8 consists of 80.7% of conduction band and 19.3% of valence band contribution.

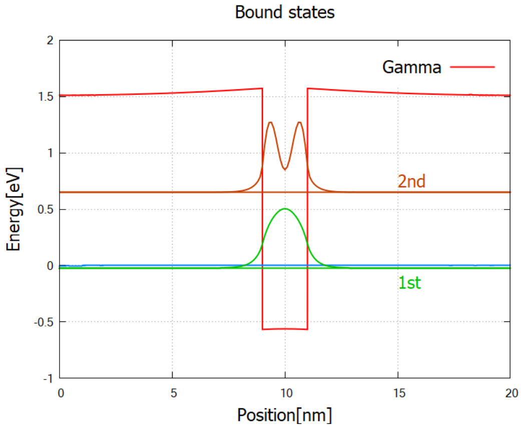


Figure 4.12.1.8: Confined states at $\mathbf{k}_{\|} = 0$ (`\Quantum\probabilities_shift_optical_active`) in a narrower and deeper quantum well. The blue line marks the electron Fermi energy (0eV).

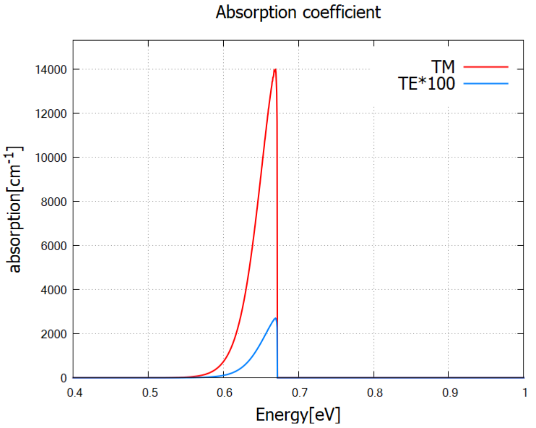


Figure 4.12.1.9: Absorption spectrum for TE and TM. TE absorption becomes relevant compared to Figure 4.12.1.7 because of the large band-mixing. Note that TE spectrum here is multiplied by a factor of 100, instead of 1000 in Figure 4.12.1.7.

Periodic case

In the third example *QWIP_Gunapala_JAP_1991_nnp.in*, we set the bias to zero and impose the periodic boundary condition. The GaAs/Al$_x$Ga$_{1-x}$As superlattice structure induces miniband states below the barriers, enabling bound-to-continuum absorptions of sub-eV photons. This μm-wavelength photodetector works without electron tunneling through the barriers, thereby improving the detectivity [Gunapala]. The band structure `bandedges.dat` and wave functions `\Quantum\probabilities_shift.dat` are shown in Figure 4.12.1.10. We have continuum states above the barriers as well as bound states in the superlattice (miniband).

The absorption coefficient is exported to `\Optics\absorption`. The indices in the filename `*_kp8_TE_m_n.dat` refer to the transition from state m to state n. The files without indices contain the total absorption spectrum (sum

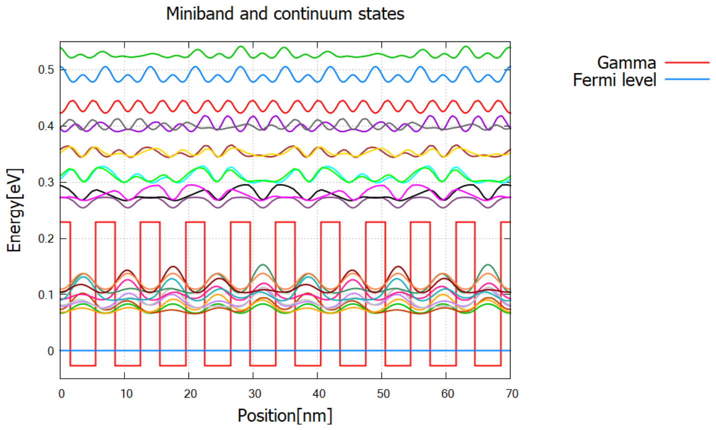


Figure 4.12.1.10: Gamma band profile and probability distribution of the bound miniband states and continuum states above the top of the barriers.

over all transitions). The total absorption spectrum for TE and TM polarization looks like this:

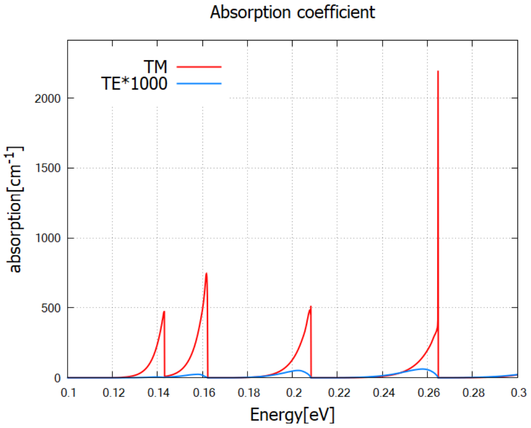


Figure 4.12.1.11: Absorption spectrum for TE ($\vec{\epsilon} = \hat{y}$) and TM ($\vec{\epsilon} = \hat{x}$) polarization. TE spectrum is magnified by factor of 1000. We observe that TM absorption is much larger than TE, while the peak positions are the same.

The peak positions do not depend on polarization, while the peak height is much larger for TM polarization compared to the one for TE. Looking at the absorption spectrum for each transition, we identify which transition contributes to which peak (Figure 4.12.1.12).

Let us look at the eigenvalue and occupation of each state to confirm this result. The eigenvalues of the bound- and continuum states are written in the output `\Quantum\probabilities_shift.dat` or `\Quantum\energy_spectrum`.

> **Note**
>
> `quantum{ }` uses spin-resolved index for the eigenstates, so there are 80 states in total. In `optics{ }`, however, two spin-degenerate states are summed up and there are only 40 states. This number (1 to 40) is used in the `\Optics` output filenames. For the consistency, we use the latter notation throughout. *(To do: examine the specifier* `spin_degeneracy`*)*

Based on the indices in Figure 4.12.1.12, we identify the first four peaks to the following four different transitions (Figure 4.12.1.13). We have confirmed that the peak energies in Figure 4.12.1.12 are consistent to the energy separation of the corresponding states.

Lastly we check the occupation (Fermi-Dirac distribution) $f_m(\mathbf{k}_\parallel)$. In the output `\Optics\eigenvaluespectrum` (Figure 4.12.1.14), occupation at k||=0 of m-th state, $f_m(\mathbf{k}_\parallel = 0)$, is plotted at

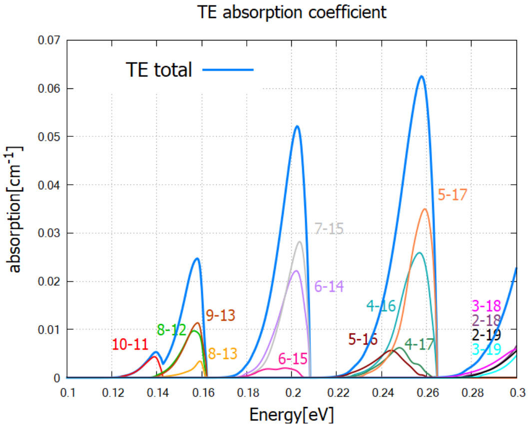


Figure 4.12.1.12: Contributions from different transitions to the total TE absorption spectrum.

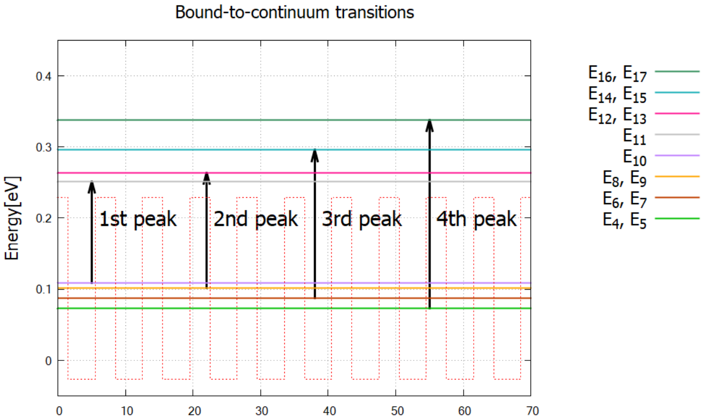


Figure 4.12.1.13: Eigenenergies of relevant bound- and continuum states. Many other transitions have little contribution due to the shape of the wave functions and/or occupation of the states. When we calculate for wider energy range, i.e. increase the parameter `$ENERGY_MAX`, there will be many more peaks that are attributed to higher energy transitions.

corresponding eigenvalues E_m. The function takes the maximum value at the origin $\mathbf{k}_{||} = 0$. In the present system, $f_1(0) = 0.087, f_2(0) = 0.077, \ldots, f_{10}(0) = 0.0148$ for the bound states, whereas $f_m(0) < 10^{-4}$ for continuum states ($m \geq 11$). Therefore the initial states in Figure 4.12.1.13 are well occupied and the final states are mostly empty. This enables optical absorption via bound-to-continuum excitation of electrons, thereby realizing a quantum well photodetector with high detectivity.

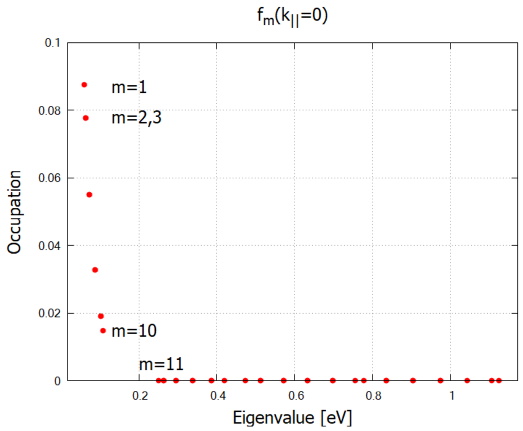


Figure 4.12.1.14: Occupation of eigenstates showing a noticeable difference for bound (m=1-10) and continuum (m=11,…) states.

1D tutorial for interband transitions: Frankenberger

Input files

- *AlGaAs_QW_Frankenberger_Simple_nnp.in*
- *AlGaAs_QW_Frankenberger_Simple_nnp_fast.in*
- *AlGaAs_QW_Frankenberger_Doping_schottky07_nnp.in*
- *AlGaAs_QW_Frankenberger_Doping_schottky07_nnp_fast.in*

These files are located in the sample files folder. The *fast* examples reduce the computation load by limiting exact solution only to $k = 0$ point and computing all other k points in the basis of the $k = 0$ wave functions (`force_k0_subspace`; see *quantum{ }* and *optics{ }* documentations).

Optical absorption and interband transitions

In the input file `AlGaAs_QW_Frankenberger_Simple_nnp.in`, we consider a single quantum well structure:

The program solves the 8-band k.p model coupled to the Poisson equation to find the eigenstates and compute the absorption coefficient. Figure Figure 4.12.1.16 shows the absorption spectrum for circularly polarized light ($\vec{\epsilon} = \hat{y} - i\hat{z}$). In contrast to QWIP examples above, peaks have long tails toward higher energy. This is because the transition energies $E_n(\mathbf{k}_{||}) - E_m(\mathbf{k}_{||})$ in interband transitions are concave upward $\sim +k^2$ (here we do not consider Type 2 semiconductors).

The steps of this absorption spectrum are associated with the following interband transitions:

Note

In the end of the log file, you find the message "Integration reliable up to —eV". This tells you up to which energy the absorption spectrum is reliable. Since we only consider the vicinity of the origin $\mathbf{k}_{||} = 0$, the reliable

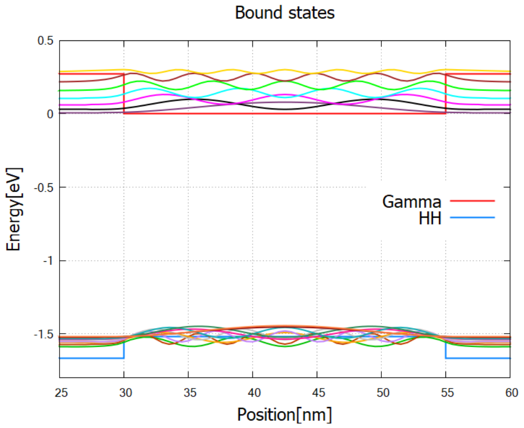


Figure 4.12.1.15: The conduction band edge profile (`bandedges.dat`) and wave functions of the bound states (`\Quantum\probabilities_shift`).

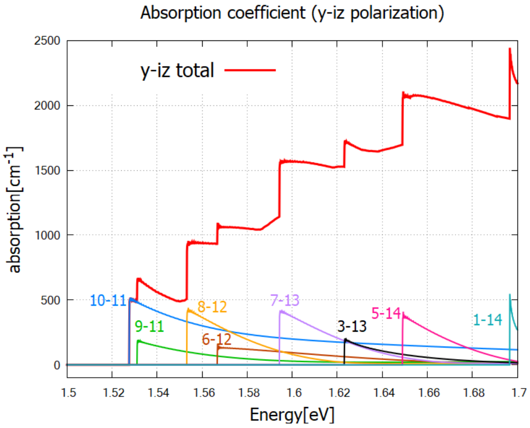
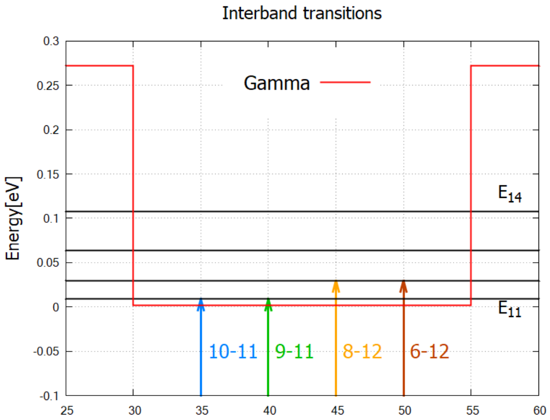


Figure 4.12.1.16: Absorption coefficient of circularly polarized lights. Numbers "m-n" denote each transition $m \to n$. The first four transitions are sketched in Figure 4.12.1.17 .



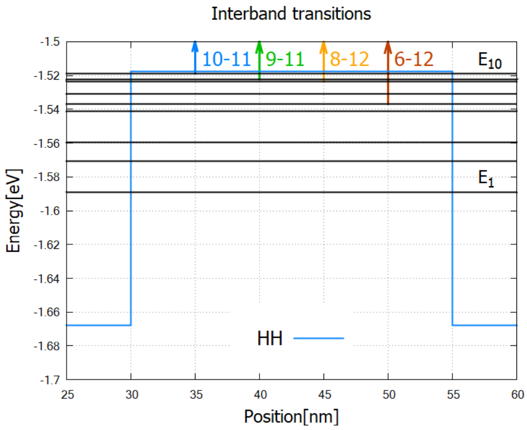


Figure 4.12.1.17: Eigenvalues (black) and transitions from valence-band to conduction band bound states (arrows) which are responsible for the first four steps in Figure 4.12.1.16. Here spin-degenerate states are counted as one state (eigenstate numbering in `optics{ }`).

energy interval is bound from above by the energy difference of the initial and final states at the edge of the k||-space considered. The upper limit d [eV] is given by

$d = \min_{\mathbf{k}_{||} \in \, \Omega^* \text{ edge}} |E_n(\mathbf{k}_{||}) - E_m(\mathbf{k}_{||})|$

where Ω^* is the region in k||-space specified in `optics{ region{ k_integration{} } }` with parameters r_{opt} and N_{opt}. In the present case d=3.2eV, while the calculation is safely performed for the interval [1.4, 1.7] (eV). This message appears only when interband transitions are computed, i.e. when `interband=yes` and `intraband=no` in `optics{ }` flag.

Doping and Schottkey barrier

In the second input file `AlGaAs_QW_Frankenberger_Doping_schottky07_nnp.in`, we consider the following structure:

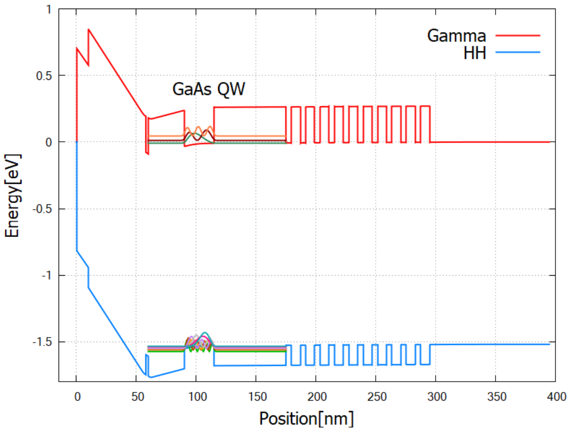


Figure 4.12.1.18: The band structure and eigen functions used for optics calculation. The Fermi level is at 0eV.

Figure 4.12.1.20 compares the results for different settings for occupation $f_m(\mathbf{k}_{||})$. When `optics{ occupation_ignore=yes }`, valence bands and conduction bands are considered to be fully occupied and fully empty, respectively. When the actual occupation of eigenstates are taken into account, in contrast, optical transi-

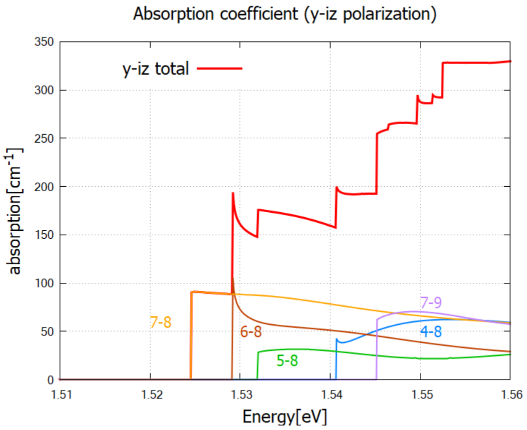


Figure 4.12.1.19: Absorption coefficient of circularly polarized lights. Numbers "m-n" denote each transition $m \rightarrow n$.

tions to conduction band states just above the Fermi energy are prohibited because of the thermal distribution of electrons.

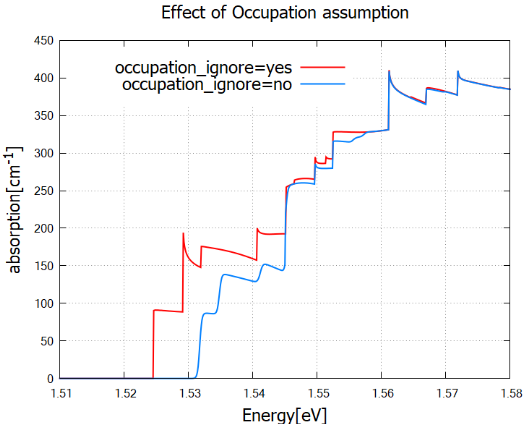


Figure 4.12.1.20: Absorption coefficient for different settings of occupation. The red curve is identical to the total absorption spectrum in Figure 4.12.1.19. When `occupation_ignore=no`, absorption of low energy photons is suppressed due to the occupation of the lowest conduction band states (also see Figure 4.12.1.18).

Last update: nnnn/nn/nn

Optical interband transitions in a quantum well - Matrix elements and selection rules

Input files:

- *1DQW_interband_matrixelements_finite_nnpp.in*
- *1DQW_interband_matrixelements_infinite_nnpp.in*

Scope:

We consider a 5 nm $GaAs$ quantum well embedded between $AlAs$ barriers. The structure is assumed to be unstrained. We distinguish between two cases:

a) finite $AlAs$ barriers

b) infinite $AlAs$ barriers (This can be achieved by choosing Dirichlet boundary conditions at the quantum well boundaries.)

Eigenstates and wave functions in the quantum well

a) Finite quantum well

Input file: *1DQW_interband_matrixelements_finite_nnpp.in*

For finite barriers we obtain using single-band Schrödinger effective-mass approximation (i.e. isotropic and parabolic effective masses)

- 3 confined electron states in the Gamma conduction band (we do not consider L and X bands here)
- 5 confined heavy hole states
- 2 confined light hole states
- 3 confined split-off hole states

Figure 4.12.1.21 shows the band edges of the Gamma conduction band and the heavy, light and split-off hole band edges together with wave functions of the confined states. Note that the heavy and light hole band edge is degenerate.

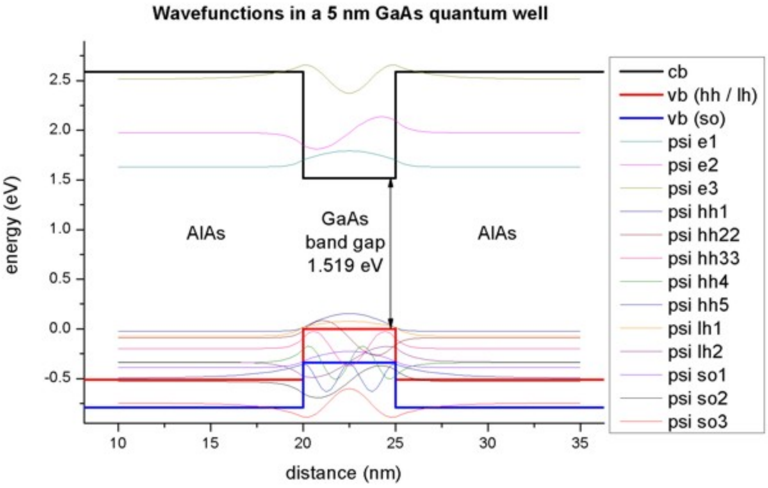


Figure 4.12.1.21: Calculated conduction band edge (black), hh/ lh valence bands (red) and split-off hole valence band (blue) with wave functions of lowest electron and hole states.

As one can see, the valence band looks rather messy. Thus, we zoom into it, see Figure 4.12.1.22 The 5 heavy hole wave functions are indicated in black, the 2 light hole wave function in red and the 3 split-off hole wave functions in blue.

Overlap integrals

Case b) Infinite quantum well

Input file: *1DQW_interband_matrixelements_infinite_nnpp.in*

To understand the optical transitions we first examine the matrix elements of the envelope functions, i.e. the spatial overlap which is the integral over their product with no dependence on polarization:

$$\langle \psi_{\mathrm{cn}} | \psi_{\mathrm{vm}} \rangle = \delta_{\mathrm{nm}}$$

This leads to the so-called ‘Delta n = 0’ selection rule, i.e. only transitions between levels with the same index are allowed. Of course, this rule is not valid anymore for case a), where we have finite $AlAs$ barriers, but nevertheless this rule gives the strongest transitions.

```
quantum{
    ...
```

(continues on next page)

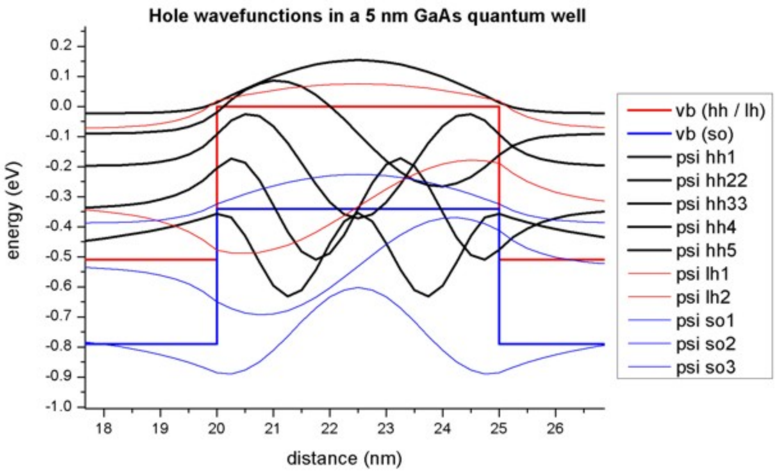


Figure 4.12.1.22: Calculated valence band edges and hole wave functions. The 5 heavy hole wave functions are indicated in black, the 2 light hole wave function in red and the 3 split-off hole wave functions in blue.

(continued from previous page)

```
    overlap_integrals{ # output matrix elements
        HH_Gamma{}
        LH_Gamma{}
        SO_Gamma{}
    }
}
```

The spatial overlap integrals of the envelope functions are contained in these files:

- *bias_00000Quantumoverlap_integrals_quantum_region_HH_Gamma.txt* - (heavy hole)
- *bias_00000Quantumoverlap_integrals_quantum_region_LH_Gamma.txt* - (light hole)
- *bias_00000Quantumoverlap_integrals_quantum_region_SO_Gamma.txt* - (split-off hole)

For instance, the matrix elements of the envelope functions for the 'heavy hole' to 'conduction band' transitions read:

```
Spatial overlap matrix elements < psi_hl_i | psi_el_j > and
                                       energy of transition in [eV].
heavy hole <-> Gamma conduction band
--------------------------------------------------------
<psi_vb001|psi_cb001>  1.001844         1.729371   ('Delta n = 0' selection rule)
<psi_vb001|psi_cb002>  3.456436E-016
<psi_vb001|psi_cb003>  7.866970E-016
<psi_vb002|psi_cb001>  7.463647E-016
<psi_vb002|psi_cb002>  1.007268         2.355209   ('Delta n = 0' selection rule)
<psi_vb002|psi_cb003>  2.844946E-016
<psi_vb003|psi_cb001>  9.575673E-016
<psi_vb003|psi_cb002>  1.450228E-015
<psi_vb003|psi_cb003>  1.015938         3.384106   ('Delta n = 0' selection rule)
<psi_vb004|psi_cb001>  1.076395E-015
<psi_vb004|psi_cb002>  1.422473E-015
<psi_vb004|psi_cb003>  2.019218E-015
<psi_vb005|psi_cb001>  1.960237E-016
<psi_vb005|psi_cb002>  1.346145E-015
<psi_vb005|psi_cb003>  1.217775E-015
```

The results shown above are for a 0.25 nm grid spacing (which is rather coarse). For a 0.1 nm grid spacing one obtains the following values for the relevant transitions:

```
<psi_vb001|psi_cb001>  1.000140        1.754633
<psi_vb002|psi_cb002>  1.000559        2.459675
<psi_vb003|psi_cb003>  1.001251        3.631886
```

Case a) finite quantum well

We now calculate the same matrix elements as above but this time for the finite $AlAs$ barriers.

```
Spatial overlap matrix elements < psi_hl_i | psi_el_j > and
                                        energy of transition in [eV].
heavy hole <-> Gamma conduction band
-------------------------------------
<psi_vb001|psi_cb001>  0.987507        1.654103   ('Delta n = 0' selection rule)
<psi_vb001|psi_cb002>  1.336279E-014
<psi_vb001|psi_cb003>  0.145559        2.538366   (same parity: symmetric)
<psi_vb002|psi_cb001>  1.133344E-014
<psi_vb002|psi_cb002>  0.964789        2.065139   ('Delta n = 0' selection rule)
<psi_vb002|psi_cb003>  7.879180E-015
<psi_vb003|psi_cb001>  0.128041        1.829856   (same parity: symmetric)
<psi_vb003|psi_cb002>  4.286800E-015
<psi_vb003|psi_cb003>  0.839306        2.714118   ('Delta n = 0' selection rule)
<psi_vb004|psi_cb001>  6.263441E-015
<psi_vb004|psi_cb002>  0.215428        2.315853   (same parity: antisymmetric)
<psi_vb004|psi_cb003>  1.246759E-015
```

The results shown above are for a 0.25 nm grid spacing (which is rather coarse). For a 0.1 nm grid spacing one obtains the following values for the relevant transitions:

```
<psi_vb001|psi_cb001>  0.987955        1.652509
<psi_vb001|psi_cb003>  0.142978        2.541682
<psi_vb002|psi_cb002>  0.966524        2.062825
<psi_vb003|psi_cb001>  0.127100        1.828683
<psi_vb003|psi_cb003>  0.838394        2.717855
<psi_vb004|psi_cb002>  0.211786        2.317309
```

6-band k.p calculations for the infinite barrier $AlAs$/ $GaAs$/ $AlAs$ quantum well

Input file: *1DQW_interband_matrixelements_infinite_kp_nnpp.in*

Figure 4.12.1.23 shows the lowest 26 eigenstates obtained with 6-band k.p for the 5 nm $GaAs$ quantum well with infinite barriers. Each k.p state is two-fold degenerate (spin up / spin down)

One can easily relate the transitions to the 'Delta n = 0' selection rule. However, in contrast to the single-band approximation, the matrix elements are not necessarily equal to 1 anymore because the hole states are mixed and thus the hole envelope functions are significantly different to the electron envelope functions, even for an infinitely deep square well.

6-band k.p calculations for the finite barrier $AlAs$/ $GaAs$/ $AlAs$ quantum well

Input file: *1DQW_interband_matrixelements_finite_kp_nnpp.in*

Figure 4.12.1.24 shows the 6-band k.p hole wave functions for the quantum well having finite $AlAs$ barriers. Their energies and Psi^2 are two-fold degenerate due to spin but the wave functions Ψ are different! (not shown here). The electron wave functions (3 confined states) are the same as above.

The calculated spatial overlap integrals nicely show that in addition to the transitions where the 'Delta n = 0' selection rule is responsible, additional transitions arise due to symmetric/antisymmetric parity. All other transitions are zero. This is in agreement with the single-band results.

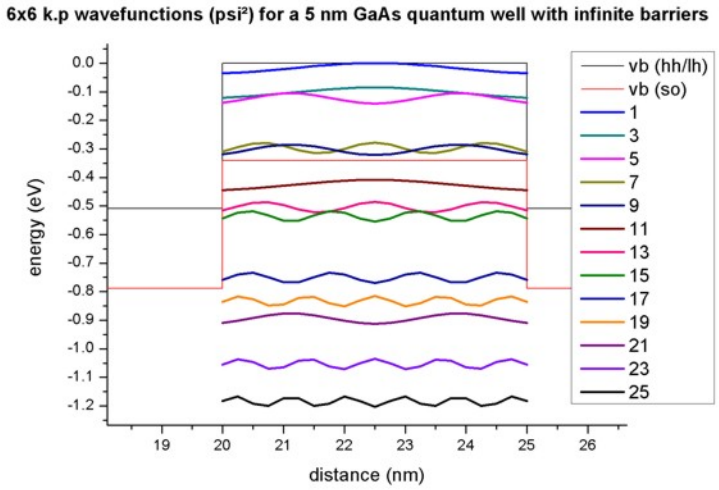


Figure 4.12.1.23: 6-band k.p wave functions (Psi^2) for a 5 nm $GaAs$ quantum well with finite barriers

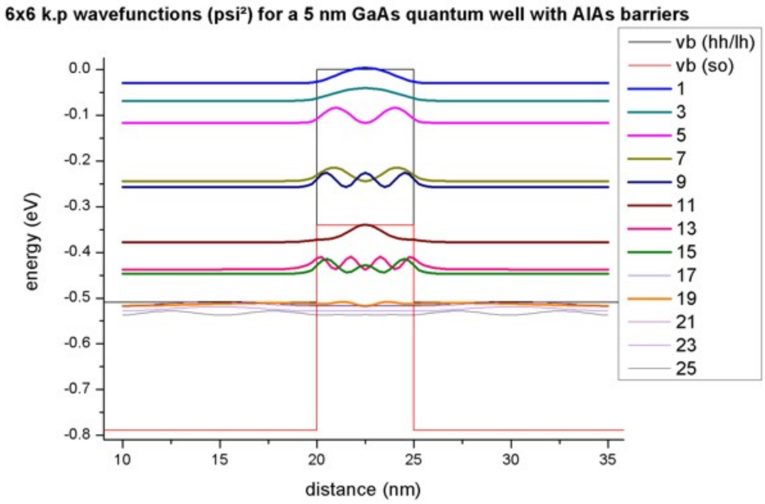


Figure 4.12.1.24: 6-band k.p wave functions (Psi^2) for a 5 nm $GaAs$ quantum well with $AlAs$ barriers

---

Last update: 27/05/2025

Optical intraband transitions in a quantum well - Momentum matrix elements and selection rules

Input files:

- *1DQW_intraband_matrixelements_infinite_nnpp.in*
- *1DQW_intraband_matrixelements_infinite_kp_nnpp.in*

Scope:

We consider a 10 nm $GaAs$ quantum well embedded between $AlAs$ barriers. The structure is assumed to be unstrained. We assume "infinite" $AlAs$ barriers. (This can be achieved by choosing a band offset of 100 eV.) This way we can compare our results to analytical text books results.

Eigenstates and wave functions in the quantum well

Input file: *1DQW_intraband_matrixelements_infinite_nnpp.in*

```
quantum{
    ...
    momentum_matrix_elements{ # output spatial overlap of wave functions
        Gamma{}
        HH{}
        LH{}
        SO{}
        output_oscillator_strengths = yes # default is no
    }
    dipole_moment_matrix_elements{ # output dipole moment matrix elements
        Gamma{}
        HH{}
        LH{}
        SO{}
        output_oscillator_strengths = yes # default is no
    }
    transition_energies{ # output transition energies
        Gamma{}
        HH{}
        LH{}
        SO{}
    }
}
```

Figure 4.12.1.25 shows the six lowest eigenfunctions of the 1D $GaAs$ quantum well. The conduction band edge of $GaAs$ is assumed to be located at 0 eV.

For "infinite" barriers we obtain using single-band Schrödinger effective-mass approximation (i.e. isotropic and parabolic effective masses) the following eigenvalues:

```
E1 = 0.05652 eV    (0.05655)
E2 = 0.22601 eV    (0.22618 = 2² E1)
E3 = 0.50831 eV    (0.50891 = 3² E1)
E4 = 0.90314 eV    (0.90473 = 4² E1)
E5 = 1.41011 eV    (1.41365 = 5² E1)
E6 = 2.02872 eV    (2.03565 = 6² E1)
```

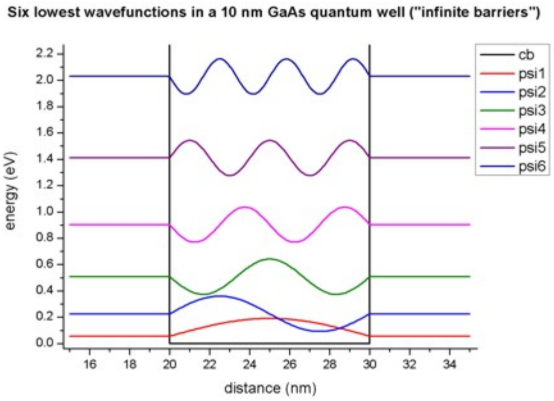


Figure 4.12.1.25: Calculated conduction band edge (black) and wave functions of confined electron states.

The analytic formula in the infinite barrier QW model reads:

$$E_n = \frac{\hbar^2}{2m_0}\left(\frac{\pi n}{L}\right)^2 = 0.056546 \cdot n^2 \text{ eV}$$

where L is the width of the quantum well (L = 10 nm). The analytically calculated values are given in brackets and are in excellent agreement.

Momentum matrix elements

Light that propagates normal to the quantum well layers cannot be absorbed by intraband transitions. However, if the light propagates in the plane of the well (i.e. the electric field is oriented normal to the quantum well layers), intersubband absorption occurs.

To understand optical intraband (= intersubband) transitions for light that travels in the plane of the QW, we have to examine the intersubband dipole moment:

$$M_{fi} = \langle \psi_{\mathrm{f}} | x | \psi_{\mathrm{i}} \rangle = \int_{-\infty}^{\infty} \psi_f^*(x) x \psi_i(x) dx$$

where $|\psi\rangle$ is the envelope function of the relevant state (within the same band).

In our case, we have a symmetric quantum well with infinite barriers, thus our envelope functions are either symmetric or antisymmetric. The intersubband matrix elements will vanish if the envelope functions have the same parity, e.g. M_{13} = M_{31} = 0. In this simple example, the matrix elements can be calculated analytically, e.g. M_{12} = (16/9π^2) L = 1.8013 nm. *nextnano*++ gives the following results:

$$M_{12} = M_{21} = 1.80143 \text{ nm}$$

$$M_{13} = M_{31} = 1.9463e^{-15} \text{ nm}$$

For the "infinite" QW barrier model, this matrix element is independent of the effective mass, thus the matrix elements in the conduction band are the same as in the valence bands (single-band approximation).

A useful quantity is the oscillator strength f_{fi} which is defined as follows:

$$f_{fi} = \frac{2m_0}{\hbar^2}(E_f - E_i) \cdot |M_{fi}|^2.$$

f_{21} for our simple infinite barrier example is given by f_{21} = 256/(27 π^2) = 0.9607 and is independent of the well width. The *nextnano*++ result is:

$$f_{21} = 0.9603 = -f_{12}$$

We can also see that this is a strong transition because all transitions from state '1' to state 'f' must add up to unity (so-called "f-sum rule"):

$$\sum_f f_{fi} = 1.0$$

(Thomas-Kuhn sum rule for constant effective mass m*.) Thus all other transitions are much weaker.

It is interesting to look at the transitions starting from the second level i = 2. The lowest oscillator strength f_{12} = -0.96 is negative, but the sum over all f_{f2} must still give unity, thus oscillator strengths larger than 1.0 are possible, e.g. f_{32} = 1.87.

The intersubband dipole moments and the oscillator strengths are contained in these files:

- *bias_00000\Quantum\dipole_moment_matrix_elements_quantum_region_Gamma_100.txt*
- *bias_00000\Quantum\dipole_moment_matrix_elements_quantum_region_HH_100.txt*
- *bias_00000\Quantum\dipole_moment_matrix_elements_quantum_region_LH_100.txt*
- *bias_00000\Quantum\dipole_moment_matrix_elements_quantum_region_SO_100.txt*

For each transition, the transition energy is given in

- *bias_00000\Quantum\transition_energies_quantum_region_Gamma.txt*
- *bias_00000\Quantum\transition_energies_quantum_region_HH.txt*
- *bias_00000\Quantum\transition_energies_quantum_region_LH.txt*
- *bias_00000\Quantum\transition_energies_quantum_region_SO.txt*

The effective masses that have been used for the calculation of the oscillator strengths are also indicated. They are calculated by building an average of the parallel effective masses for each grid point, weighted by the square of the wave function on each grid point. In this particular example, the effective masses are constant and do not vary with position ($m_{||}$ = 0.0665 m_0). (Assuming that the masses are isotropic, it is fine to use the parallel effective masses.)

```
-------------------------------------------------------------------------------
Intersubband transitions
=> Gamma conduction band
-------------------------------------------------------------------------------
Electric field in z-direction [kV/cm]: 0.0000000E+00
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
                Intersubband dipole moment  | < psi_f* | z | psi_i > |  [Angstrom]
------------------|------------------------------------------------------------
                                Oscillator strength []
------------------|--------------|--------------------------------------------
                                                Energy of transition [eV]
------------------|--------------|--------------|-----------------------------
                                                            m* [m_0]
------------------|--------------|--------------|----------|-------------------
<psi001*|z|psi001>  249.0000
<psi002*|z|psi001>  18.01673       0.9602799      0.1694912  6.6500001E-02
<psi003*|z|psi001>  6.1430171E-07  2.9757722E-15  0.4517909  6.6500001E-02 (same␣
↪parity: symmetric)
<psi004*|z|psi001>  1.441336       3.0698571E-02  0.8466209  6.6500001E-02
<psi005*|z|psi001>  1.6007220E-07  6.0536645E-16  1.353592   6.6500001E-02 (same␣
↪parity: symmetric)
<psi006*|z|psi001>  0.3971010      5.4281605E-03  1.972205   6.6500001E-02
<psi007*|z|psi001>  5.1874160E-08  1.2690011E-16  2.701849   6.6500001E-02 (same␣
↪parity: symmetric)
<psi008*|z|psi001>  0.1634139      1.6508275E-03  3.541806   6.6500001E-02
...
<psi020*|z|psi001>  1.0178176E-02  3.9451432E-05  21.81846   6.6500001E-02
Sum rule of oscillator strength: f_psi001 = 0.9994023

<psi001*|z|psi002>  18.01673      -0.9602799     -0.1694912  6.6500001E-02
```

(continues on next page)

(continued from previous page)

```
<psi002*|z|psi002>  249.0000
<psi003*|z|psi002>  19.45806       1.865556       0.2822997  6.6500001E-02
<psi004*|z|psi002>  2.0636767E-06  5.0333130E-14  0.6771297  6.6500001E-02 (same␣
↪parity: antisymmetric)
<psi005*|z|psi002>  1.838436       6.9852911E-02  1.184101   6.6500001E-02
<psi006*|z|psi002>  1.4976163E-08  7.0571038E-18  1.802713   6.6500001E-02 (same␣
↪parity: antisymmetric)
<psi007*|z|psi002>  0.5605143      1.3886644E-02  2.532358   6.6500001E-02
<psi008*|z|psi002>  8.7380023E-08  4.4941879E-16  3.372315   6.6500001E-02 (same␣
↪parity: antisymmetric)
<psi009*|z|psi002>  0.2461317      4.5697703E-03  4.321757   6.6500001E-02
<psi010*|z|psi002>  8.3240280E-07  6.5062044E-14  5.379748   6.6500001E-02 (same␣
↪parity: antisymmetric)
<psi011*|z|psi002>  0.1302904      1.9393204E-03  6.545245   6.6500001E-02
...
<psi020*|z|psi002>  2.7233656E-07  2.8025147E-14  21.64897   6.6500001E-02
Sum rule of oscillator strength: f_psi002 = 0.9975320

<psi001*|z|psi003>  6.1430171E-07 -2.9757722E-15 -0.4517909  6.6500001E-02 (same␣
↪parity: symmetric)
<psi002*|z|psi003>  19.45806      -1.865556      -0.2822997  6.6500001E-02
<psi003*|z|psi003>  249.0000
<psi004*|z|psi003>  19.85515       2.716784       0.3948300  6.6500001E-02
<psi005*|z|psi003>  6.4708888E-07  6.5907892E-15  0.9018011  6.6500001E-02 (same␣
↪parity: symmetric)
<psi006*|z|psi003>  2.001849       0.1063465      1.520414   6.6500001E-02
<psi007*|z|psi003>  3.9201248E-07  6.0352080E-15  2.250058   6.6500001E-02 (same␣
↪parity: symmetric)
<psi008*|z|psi003>  0.6432316      2.2314854E-02  3.090015   6.6500001E-02
<psi009*|z|psi003>  2.6240454E-07  4.8547223E-15  4.039457   6.6500001E-02 (same␣
↪parity: symmetric)
...
<psi020*|z|psi003>  3.1797737E-02  3.7707522E-04  21.36667   6.6500001E-02
Sum rule of oscillator strength: f_psi003 = 0.9945912
```

The commonly used intersubband dipole moment $\langle \psi_\mathrm{f} | x | \psi_\mathrm{i} \rangle$ [nm] depends on the choice of origin for the matrix elements when f = i, thus the user might prefer to output the Intersubband dipole moment $\langle \psi_\mathrm{f} | p_x | \psi_\mathrm{i} \rangle$ which are the intersubband dipole moments

$$N_{fi} = \langle \psi_\mathrm{f} | \hat{p}_x | \psi_\mathrm{i} \rangle = \int_{-\infty}^{\infty} \psi_f^*(x) \hat{p}_x \psi_i(x) dx = -i\hbar \int_{-\infty}^{\infty} \psi_f^*(x) \frac{\partial}{\partial x} \psi_i(x) dx$$

and the oscillator strengths

$$f_{fi} = \frac{2m_0}{\hbar^2}(E_f - E_i) \cdot |M_{fi}|^2 = \frac{2}{m_0(E_f - E_i)} \cdot |N_{fi}|^2$$

between all calculated states in each band from min to max eigenvalues. In the simple QW of this tutorial, the matrix elements can be calculated analytically, e.g. N_{21} = 8$\hbar$/3L = 0.2666 $\hbar$/nm. The *nextnano++* result is:

$$N_{21} = N_{12} = 0.265957 \ \hbar/\mathrm{nm}$$

$$N_{31} = N_{13} = 7.05011e^{-17}$$

The oscillator strength f_{21} for our simple infinire barrier example is given by f_{21} = 256/(27π^2) = 0.9607 and is independent of the well width. The *nextnano++* result is:

$$f_{21} = -f_{12} = 0.9603$$

The intersubband dipole moments and the oscillator strengths are contained in these files:

- *bias_00000\Quantum\momentum_matrix_elements_quantum_region_Gamma_100.txt*
- *bias_00000\Quantum\intraband__matrix_elements_quantum_region_HH_100.txt*
- *bias_00000\Quantum\momentum_matrix_elements_quantum_region_LH_100.txt*
- *bias_00000\Quantum\momentum_matrix_elements_quantum_region_SO_100.txt*

The numbers show a comparison between the x and the p_x matrix elements for *nextnano*[3]:

```
-------------------------------------------------------------------------------
                   Intersubband dipole moment  | < psi_f* | z | psi_i > |  [Angstrom]
                   Intersubband dipole moment  | < psi_f* | p | psi_i > |  [h_bar /␣
↪Angstrom]
------------------|------------------------------------------------------------
                                 Oscillator strength []
------------------|--------------|---------------------------------------------
                                                Energy of transition [eV]
------------------|--------------|--------------|------------------------------
                                                            m*  [m_0]
------------------|--------------|--------------|-----------|------------------
<psi001*|z|psi001>  249.0000       (matrix element <1|1> depends on choice of origin!)
<psi001*|p|psi001>  4.3405972E-19  (matrix element <1|1> independent of origin)

<psi002*|z|psi001>  18.01673       0.9602799      0.1694912  6.6500001E-02
<psi002*|p|psi001>  2.6649671E-02  0.9602799      0.1694912  6.6500001E-02

<psi003*|z|psi001>  6.1430171E-07  2.9757722E-15  0.4517909  6.6500001E-02 (same␣
↪parity: symmetric)
<psi003*|p|psi001>  2.7325134E-18

<psi004*|z|psi001>  1.441336       3.0698571E-02  0.8466209  6.6500001E-02
<psi004*|p|psi001>  1.0649348E-02  3.0698579E-02  0.8466209  6.6500001E-02

<psi005*|z|psi001>  1.6007220E-07  6.0536645E-16  1.353592   6.6500001E-02 (same␣
↪parity: symmetric)
<psi005*|p|psi001>  6.9518724E-18

<psi006*|z|psi001>  0.3971010      5.4281605E-03  1.972205   6.6500001E-02
<psi006*|p|psi001>  6.8347314E-03  5.4281540E-03  1.972205   6.6500001E-02

<psi007*|z|psi001>  5.1874160E-08  1.2690011E-16  2.701849   6.6500001E-02 (same␣
↪parity: symmetric)
<psi007*|p|psi001>  2.8686024E-19

<psi008*|z|psi001>  0.1634139      1.6508275E-03  3.541806   6.6500001E-02
<psi008*|p|psi001>  5.0510615E-03  1.6508278E-03  3.541806   6.6500001E-02
...
<psi020*|z|psi001>  1.0178176E-02  3.9451432E-05  21.81846   6.6500001E-02
<psi020*|p|psi001>  1.9380626E-03  3.9452334E-05  21.81846   6.6500001E-02

Sum rule of oscillator strength: f_psi001 = 0.9994023
Sum rule of oscillator strength: f_psi001 = 0.9994023
```

8-band k.p calculation for $k_{||}$ = (K_y, k_z) = 0

The following input file performs the same calculations as above but this time using the 8-band k.p model: *1DQW_intraband_matrixelements_infinite_kp_nnpp.in*.

We modified the 8-band k.p parameters and decoupled (!) the electrons from the holes (EP = 0 eV, S = $1/m_e$).

This way we have an effective single-band model, and thus we are able to compare the k.p results to the single-band results in order to check for consistency.

The numbering of the k.p eigenstates differs slightly from the single-band eigenstates because the k.p eigenstates are two-fold spin-degenerate. The actual values for the matrix elements are identical (assuming a decoupled k.p Hamiltonian, i.e. a single-band Hamiltonian).

Note that the single-band definition of the oscillator strength does not really make sense for a k.p calculation where the masses usually are anisotropic, non-parabolic and are different on each grid point (due to different materials and different strain tensors).

For the calculation of the oscillator strength in a k.p calculation, the user can specify suitable masses by overwriting the default entries. Of course, the masses that are used to calculate the k.p eigenstates have to be specified via the 6-band and 8-band k.p parameters (inside the `database{ }` group).

The intersubband dipole moments and the oscillator strengths are contained in this file:

- *bias_00000\Quantum\momentum_matrix_elements_quantum_region_kp8_100.txt* (p_x elements)
- *bias_00000\Quantum\dipole_moment_matrix_elements_quantum_region_kp8_100.txt* (x elements)

Note that the two-fold spin-degeneracy in single-band is counted explicitly in k.p.

```
-------------------------------------------------------------------------------
                  Intersubband dipole moment  | < psi_f* | z | psi_i > |  [Angstrom]
                  Intersubband dipole moment  | < psi_f* | p | psi_i > |  [h_bar / Angstrom]
------------------|------------------------------------------------------------
                                 Oscillator strength []
------------------|--------------|---------------------------------------------
                                                Energy of transition [eV]
------------------|--------------|--------------|------------------------------
                                                            m* [m_0]
------------------|--------------|--------------|-----------|------------------
<psi001*|z|psi001>  249.0000       (matrix element <1|1> depends on choice of origin!)
<psi002*|z|psi001>  249.0000       (matrix element <2|1> depends on choice of origin!)
<psi001*|p|psi001>  1.8126842E-18  (matrix element <1|1> independent of origin)
<psi002*|p|psi001>  1.8126842E-18  (matrix element <2|1> independent of origin)

<psi003*|z|psi001>  18.01673       0.9602799      0.1694912  6.6500001E-02
<psi004*|z|psi001>  18.01673       0.9602799      0.1694912  6.6500001E-02
<psi003*|p|psi001>  2.6649671E-02  0.9602798      0.1694912  6.6500001E-02
<psi004*|p|psi001>  2.6649671E-02  0.9602798      0.1694912  6.6500001E-02

<psi005*|z|psi001>  3.5382732E-13
<psi006*|z|psi001>  3.5382732E-13
<psi005*|p|psi001>  2.1414240E-15
<psi006*|p|psi001>  2.1414240E-15

<psi007*|z|psi001>  1.441336       3.0698583E-02  0.8466209  6.6500001E-02
<psi008*|z|psi001>  1.441336       3.0698583E-02  0.8466209  6.6500001E-02
<psi007*|p|psi001>  1.0649348E-02  3.0698583E-02  0.8466209  6.6500001E-02
<psi008*|p|psi001>  1.0649348E-02  3.0698583E-02  0.8466209  6.6500001E-02

<psi009*|z|psi001>  7.2598817E-13
<psi010*|z|psi001>  7.2598817E-13
<psi009*|p|psi001>  1.0445775E-14
<psi010*|p|psi001>  1.0445775E-14

<psi011*|z|psi001>  0.3971008      5.4281550E-03  1.972205   6.6500001E-02
<psi012*|z|psi001>  0.3971008      5.4281550E-03  1.972205   6.6500001E-02
```

(continues on next page)

(continued from previous page)

```
<psi011*|p|psi001>  6.8347319E-03  5.4281550E-03  1.972205   6.6500001E-02
<psi012*|p|psi001>  6.8347319E-03  5.4281550E-03  1.972205   6.6500001E-02

...
<psi039*|z|psi001>  1.0178294E-02  3.9452352E-05  21.81846   6.6500001E-02
<psi040*|z|psi001>  1.0178294E-02  3.9452352E-05  21.81846   6.6500001E-02
<psi039*|p|psi001>  1.9380630E-03  3.9452349E-05  21.81846   6.6500001E-02
<psi040*|p|psi001>  1.9380630E-03  3.9452349E-05  21.81846   6.6500001E-02

Sum rule of oscillator strength: f_psi001 = 0.9994023
Sum rule of oscillator strength: f_psi001 = 0.9994023
```

This tutorial also exists for *nextnano*^{3}.

Last update: 27/05/2025

Optical absorption of an InGaAs quantum well | 1D

Header

Files for the tutorial located in *nextnano++\examples\optical_spectra*

- *absorption_InGaAs-QW_Dumitras_PRB_2002_1D_nnp.in*

Scope of the tutorial:

- InGaAs quantum well
- simple absorption spectrum

Main adjustable parameters in the input file:

- `$run_optics`
- `$w_well`
- `$w_barrier`
- `$alloy_composition`

Relevant output files:

- *bias_00000\bandedges.dat* - energy profile (see Figure 4.12.1.26)
- *bias_00000\Quantum\probabilities_shift_quantum_region_kp8_00000.dat* - probability distributions (see Figure 4.12.1.26)
- *bias_00000\Quantum\absorption_coeff_quantum_region_TE_y_eV.dat* - absorption spectrum TE (see Figure 4.12.1.27)

- *bias_00000\Quantum\absorption_coeff_quantum_region_TM_z_eV.dat* - absorption spectrum TM (see Figure 4.12.1.27)

Introduction

This tutorial presents a simple setup to calculate optical absorption coefficient as a function of photon energy for transitions in a quantum well (QW) by means of 8-band $\mathbf{k} \cdot \mathbf{p}$ method. As an example, we chose 8-nm-wide $In_{0.2}Ga_{0.8}As$ quantum well with barriers made of GaAs, as in *[DumitrasPRB2002]*. The InGaAs QW is pseudomorphically strained to the GaAs (001) substrate and the temperature of the system is assumed to be 150 K.

Simulation

Input file

The input file *absorption_InGaAs-QW_Dumitras_PRB_2002_1D_nnp.in* is prepared to solve Schrödinger and Poisson equations without self-consistency, with included strain effects. A couple of variables defined within the input file are especially interesting to play with when trying the simulation for the first time. The first of them is `$run_optics` which allows turning calculation of the optical spectra on and off. When the spectra are computed, the Fermi's Golden Rule is used. Other parameters are temperature of the system `$temperature` and parameters characterizing the dimensions, `$w_well` and `$w_barrier`, and content of the QW `$alloy_composition`. We encourage modifying other parameters as well to explore the simulation capabilities.

> **Note**
>
> The `bandoffset` bowing parameter for the `In(x)Ga(1-x)As` alloy has been set to `0` at the end of the input file to obtain energy profile similar with the one reported in *[DumitrasPRB2002]*.

Eigenstates in the quantum well

Energy profiles together with probability densities of all states confined in the InGaAs QW (at $k_{||} = 0$) are showed in the Figure 4.12.1.26. The energy profiles can be found in *bias_00000\bandedges.dat* while the probability densities in *bias_00000\Quantum\probabilities_shift_quantum_region_kp8_00000.dat*.

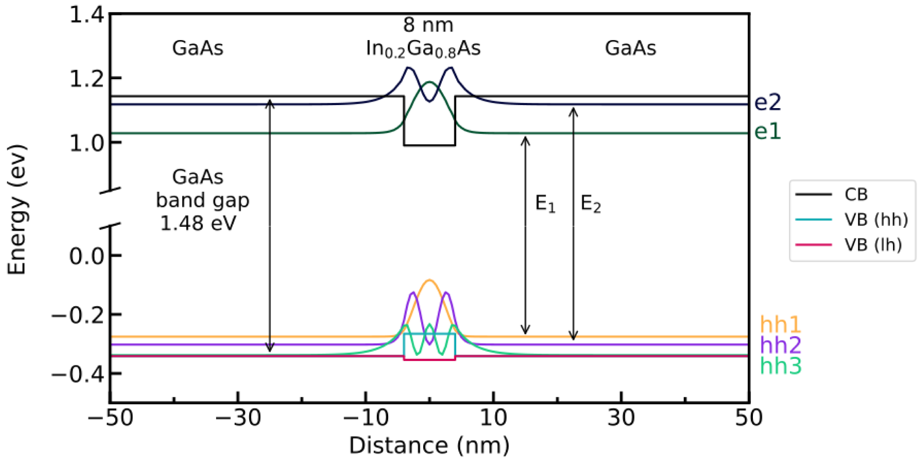


Figure 4.12.1.26: Energy profiles and probability distributions of confined electrons and holes states at $k_{||} = 0$. The conduction band is labeled as **CB**. The heavy-hole valence bands is denoted as **VB (hh)** while the light-hole valence band as **VB (lh)**. The first and the second electron states are labeled as **e1** and **e2**, respectively. Similarly, heavy-hole states are labeled as **hh1** and **hh2**. E_1 is a transition energy between **e1** and **hh1**. E_2 is a transition energy between **e2** and **hh2**.

The prepared simulation computes 20 electron states and 40 hole states (sum of light-hole and heavy-hole states). All of these states (at each wave vector) are used for computation of the optical spectra as they contribute to the part representing continuum. However, only the bound states are crucial for the analysis of the quantum well. One can quickly compute the most relevant interband transition energies, E_1 and E_2, if omitting the exciton corrections. These transitions are the strongest ones, following the selection rule $\Delta n = 0$, between two states with the same quantum number, e.g., between **e1** and **h1** or between **e2** and **h2**.

The transition energies E_1 and E_2 are defined as

$$E_1 = E_{\mathrm{e1}} - E_{\mathrm{hh1}},$$
$$E_2 = E_{\mathrm{e2}} - E_{\mathrm{hh2}},$$

where E_{e1}, E_{e2}, E_{hh1}, and E_{hh2} are eigenenergies of the states **e1**, **e2**, **hh1**, and **hh2**, respectively. Using respective values from the output file *bias_00000\Quantum\probabilities_shift_quantum_region_kp8_00000.dat* one can calculate

$$E_1 = 1.028 \text{ eV} - [-0.275 \text{ eV}] = 1.303 \text{ eV},$$
$$E_2 = 1.118 \text{ eV} - [-0.302 \text{ eV}] = 1.420 \text{ eV}.$$

Note that these transition energies are calculated at $k_{\|} = 0$.

Hint

One can use **Show Differences** feature in *nextnanomat* to extract these numbers from the eigenenergies stored in *bias_00000\Quantum\probabilities_shift_quantum_region_kp8_00000.dat*. Also, *nextnano++* can produce an output file containing all transition energies, see `output_transition` in *optics{ quantum_spectra{} }*.

Optical absorption spectrum

When `$run_optics = 1` in the input file for this tutorial, then optical spectra are also computed. The simulation is prepared to model optical spectra for two kinds of light polarization modes.

The transverse electric (TE) mode corresponds to the optical field (could be light) polarized parallel to the plane of the QW, that is in the **yz** plane of the simulation. In the input file we choose the direction **y**. Choosing **z** direction for the TE mode brings the same results. The light in this mode can propagate either in the plane of the QW or perpendicular to it.

The transverse magnetic (TM) mode corresponds to the optical field polarized perpendicular to the plane of the QW, that is in the **x** direction of the simulation. The light in this mode can propagate only in the pane of the QW.

Figure 4.12.1.27 shows the optical absorption spectrum as a function of photon energy for TE and TM polarized optical field.

While optical transitions involving both **heavy holes** and **light holes** can be observed within TE mode (**heavy holes** are dominating), only absorption with contribution of **light holes** is visible in the TM mode.

Attention

The above does not hold exactly in realistic conditions because the TM modes also have a component of the electric field parallel to the plane. However, this component is small in weakly guiding structures. Therefore, typically only the transition involving the **light holes** is seen (**e1-lh1**) and the heavy hole transitions are suppressed (**e1-hh1**, **e2-hh2**) in Figure 4.12.1.27.

The transitions E_1 and E_2 are clearly visible in the computed TE absorption spectrum as steps at 1.303 eV and 1.420 eV, respectively. Both computed TE and TM spectra exhibit series of transitions at around 1.37 eV and 1.46 eV. These are **numerical artifacts** related to transitions between the states confined in the InGaAs QW and numerically limited continuum in the GaAs. To explore this aspect of the simulation one can modify the width of the barrier `$w_barrier` and number of computed quantum states `$eigen_e` and `$eigen_v`.

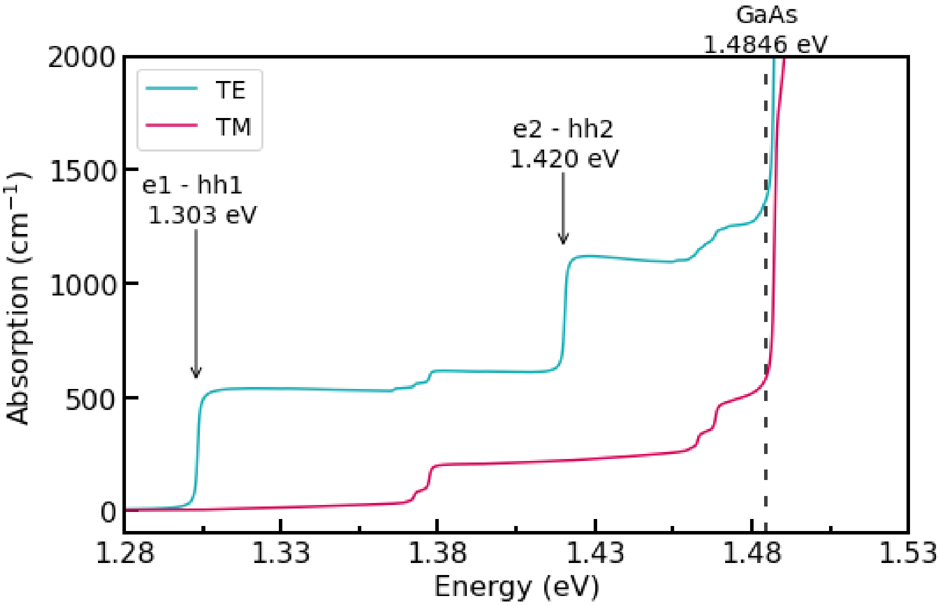


Figure 4.12.1.27: Absorption spectrum for TE (turquoise) and TM (magenta) modes of optical field.

Hint

Using `normalization_volume` may become very helpful when comparing spectra computed for different dimensions of the structure, see *optics{ quantum_spectra{} }*.

Last update: 07/03/2024

Intersubband absorption of an infinite quantum well

Input files for *nextnano++*:

- *1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_sg_nnp.in*
- *1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_Gamma_nnp.in*
- *1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_kp_nnp.in*

This tutorial presents calculation of intersubband absorption spectrum of a GaAs quantum well with infinite barriers.

The following input file was used:

- `1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_sg_nn3.in` (single-band effective mass approximation)

This tutorial aims to reproduce the example discussed on p. 376f of Section 9.6.2 *Intersubband Absorption Spectrum* of *[ChuangOpto1995]*.

Structure

Property	Symbol	unit	*[ChuangOpto1995]*	nextnano
quantum well width	L	nm	10.0	10.0
barrier height	E_b	eV	infinite quantum well model	1000
effective electron mass	m_e	m_0	0.0665	0.0665
refractive index	n_r		3.3	3.3
doping concentation (n-type)	N_D	cm^{-3}	$1 \cdot 10^{18}$	$1 \cdot 10^{18}$
linewidth (FWHM)	Γ	meV	30	30
temperature	T	K	300	300

[ChuangOpto1995] models the infinite quantum well using the analytical solution while we are using a numerical model with a barrier height of 1000 eV.

Results

[ChuangOpto1995] uses the analytical infinite quantum well model and calculates the energy levels, and the intersubband dipole moment exactly. Our calculated transition energies differ by 3 meV which is acceptable as we use a finite grid spacing of 0.05 nm. Our calculated dipole moment is also reasonable. More difficult are the densities. In our calculation we solve the Schrödinger-Poisson equation self-consistently. For that reason, the quantum well bottom is not entirely flat but slightly bent. At T = 300 K, the second subband shows a small density which is larger than in the model of *[ChuangOpto1995]*. The difference in subband densities leads to a slight deviation for the peak of the absorption spectrum because the occupation of the second level N_2 reduces absorption. Nevertheless, the agreement is reasonable.

Property	Symbol	unit	*[ChuangOpto1995]*	nextnano
energy level	E_1	meV	56.5 (exact)	
energy level	E_2	meV	226 (exact)	
transition energy	E_{21}	meV	169.5 (exact)	166.5
dipole moment	x_{21}	nm	-1.8 (exact)	-1.82
E_F - E_1		eV	78	28.2
subband density	N_1	cm^{-2}	$7.19 \cdot 10^{11}$	$9.92 \cdot 10^{11}$
subband density	N_2	cm^{-2}		$3 \cdot 10^{9}$
peak in absorption	α_{peak}	cm^{-1}	$1.015 \cdot 10^{4}$	$0.986 \cdot 10^{4}$

The following figures show the

- lowest eigenstates (probability densities) of the infinite quantum well
- absorption spectra $\alpha(\omega)$ in units of cm^{-1}
- position dependent absorption spectra $\alpha(\omega, x)$ in units of cm^{-1}

The peak in the absorption spectra occurs at the transition energy E_{21}.

Then we perform two parameter sweeps:

- We vary the **quantum well width** (Variable: `$QuantumWellWidth`).
- We vary the **doping concentration** (Variable: `$DopingConcentration`).

Results and explanations for the sweeps can be found further below.

— Begin —

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

---

The following figures have been generated using *nextnano³*. Self-consistent Schrödinger-Poisson calculations have been performed for an infinite quantum well.

A single-band effective mass approach has been used, i.e. not $\mathbf{k} \cdot \mathbf{p}$.

The absorption spectra have been calculated assuming a parabolic energy dispersion $E(k)$.

Infinite Quantum Well (QuantumWellWidth = 10 nm)

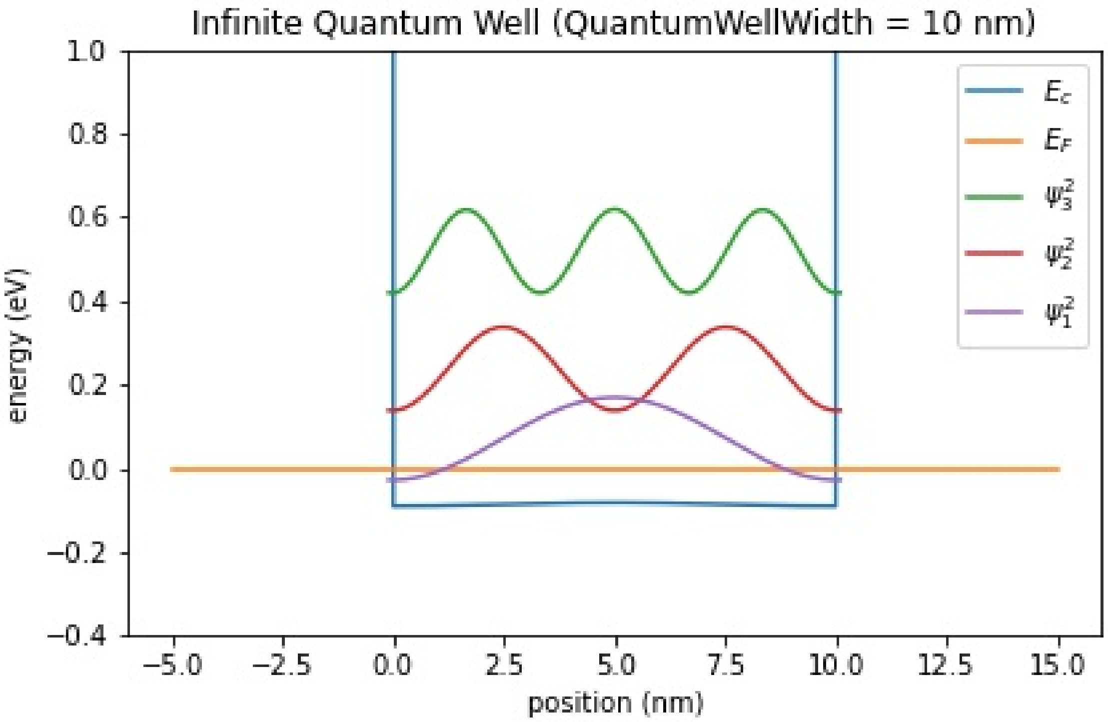


Figure 4.12.1.28: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 10 nm)

Infinite Quantum Well (QuantumWellWidth = 13 nm)

Infinite Quantum Well (QuantumWellWidth = 16 nm)

Infinite Quantum Well (QuantumWellWidth = 19 nm)

Parameter sweep: Well width

Figure 4.12.1.37 shows the absorption spectra for different **quantum well widths** (Variable: `$QuantumWellWidth`). The larger the well, the closer the energy level spacings. Therefore the peak occurs at smaller energies. The larger wells show absorption also for transitions other than E_{21}.

Parameter sweep: Doping concentration

Figure 4.12.1.38 shows the absorption spectra for different **doping concentrations** (Variable: `$DopingConcentration`). The peak absorption coefficient increases with the doping concentration N_D.

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

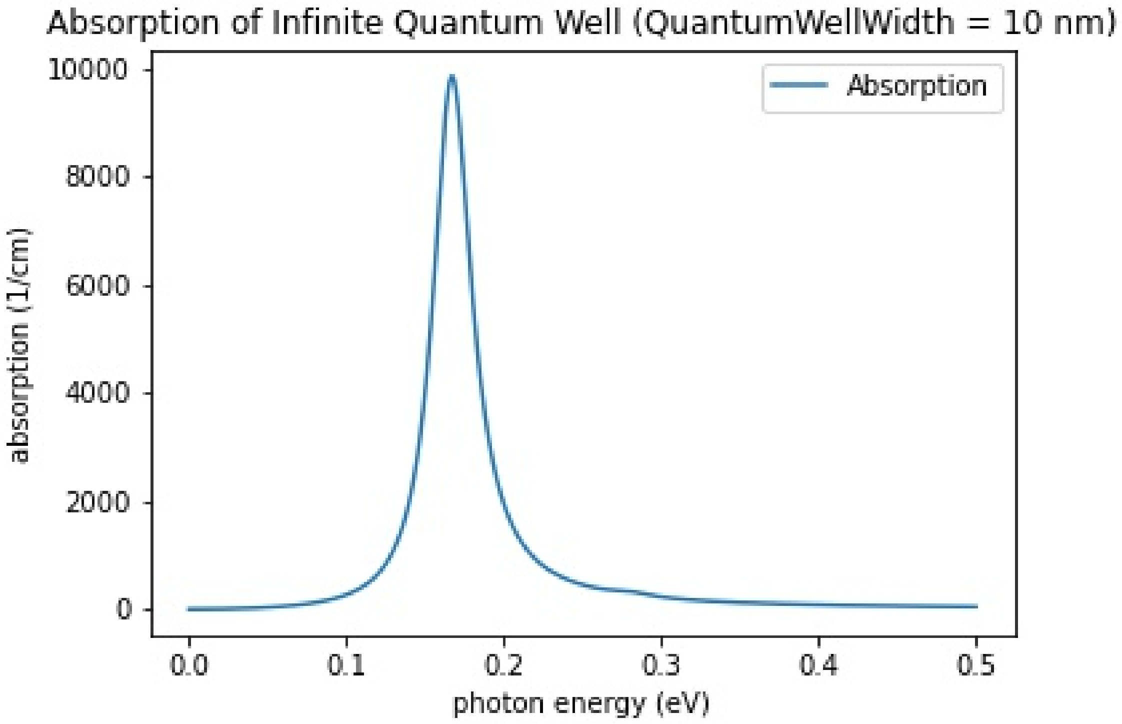


Figure 4.12.1.29: Calculated absorption spectra $\alpha(E)$ of an infinite quantum well (QuantumWellWidth = 10 nm)

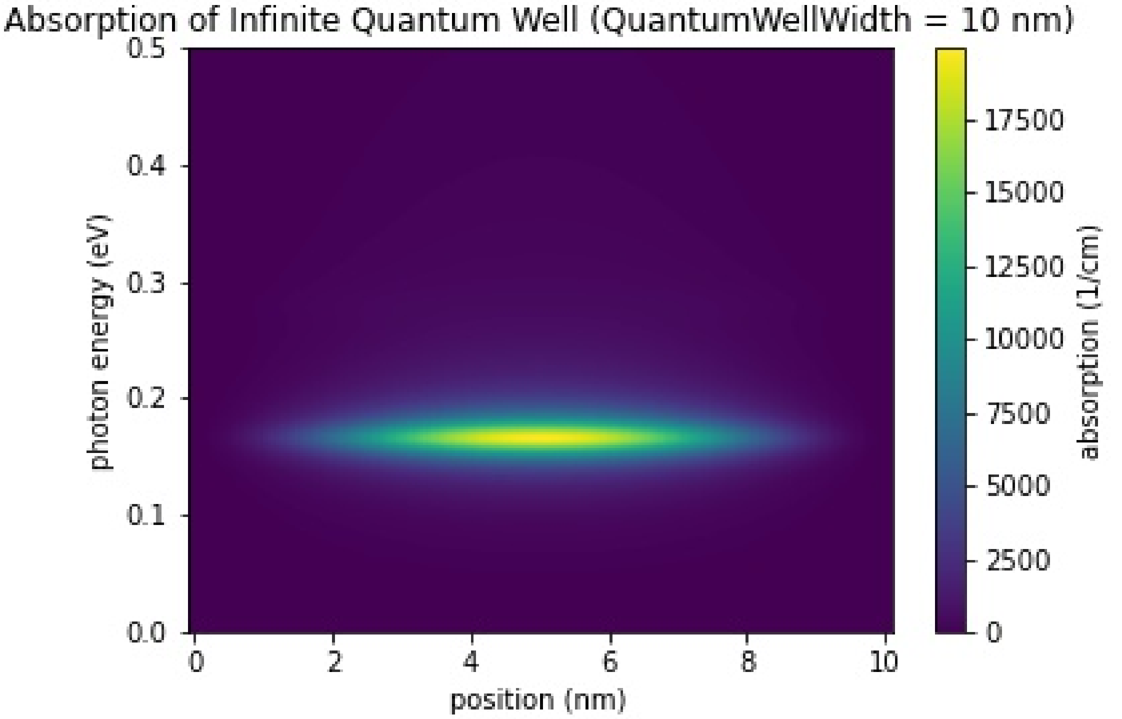


Figure 4.12.1.30: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 10 nm)

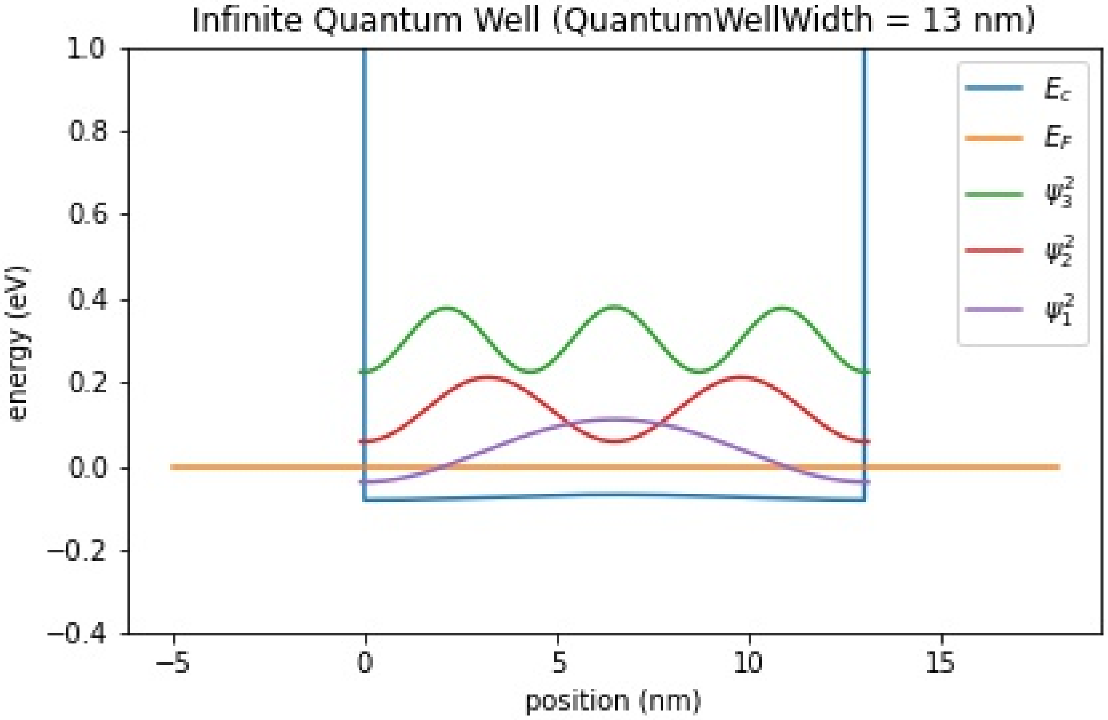


Figure 4.12.1.31: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 13 nm)

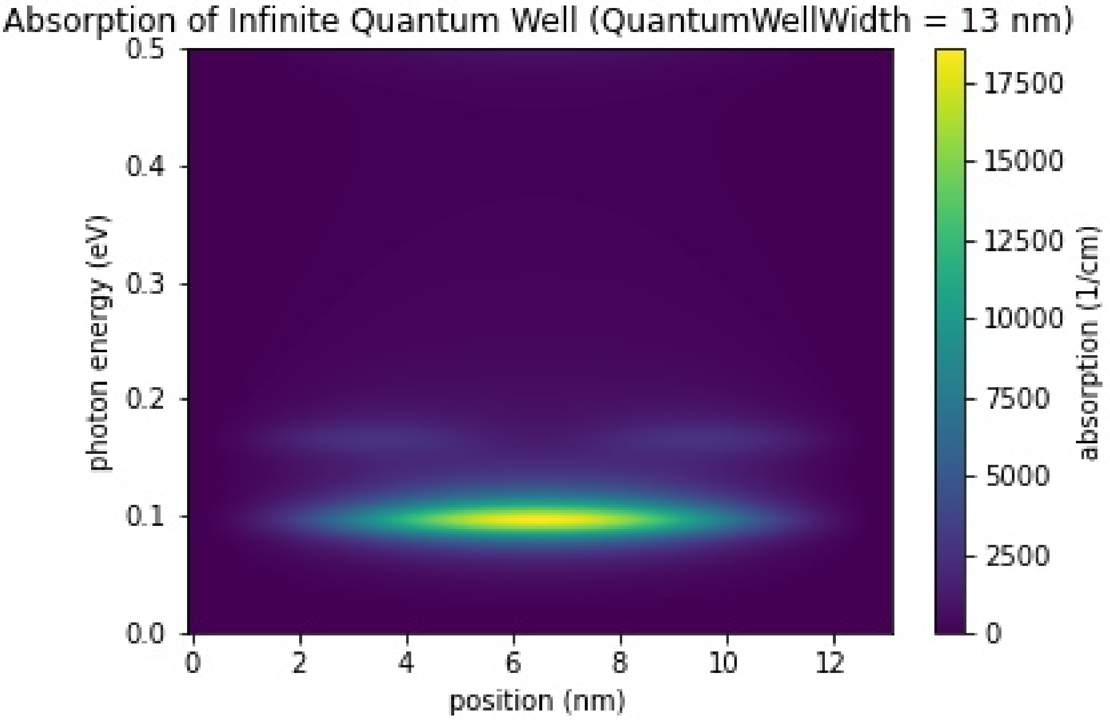


Figure 4.12.1.32: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 13 nm)

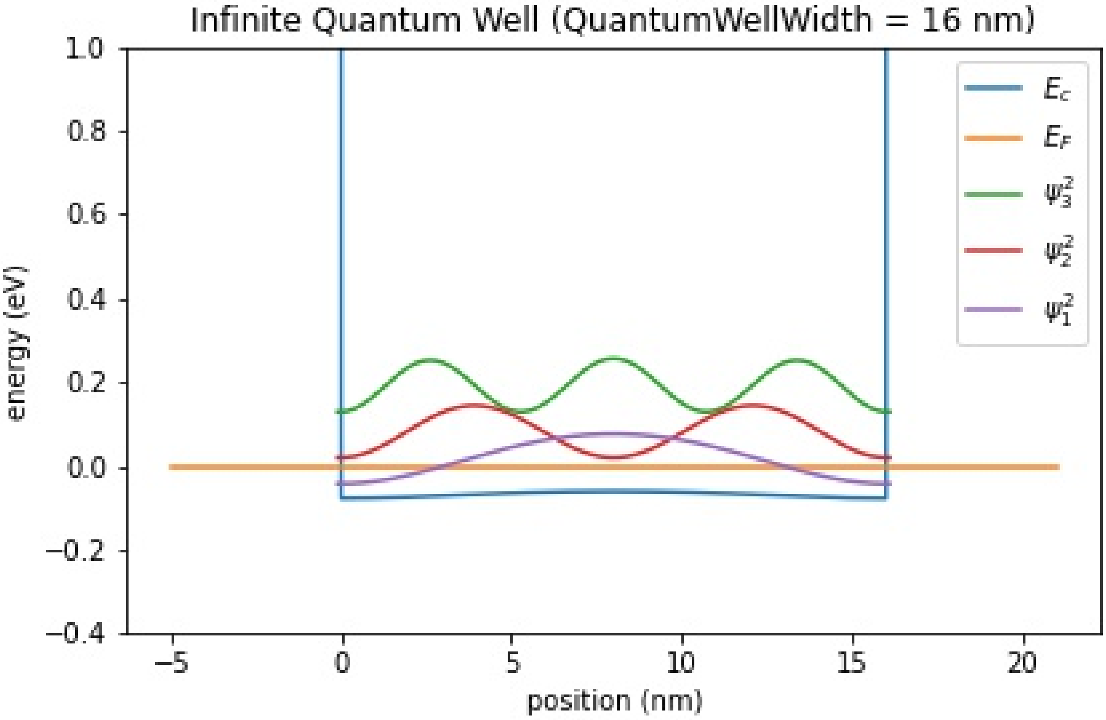


Figure 4.12.1.33: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 16 nm)

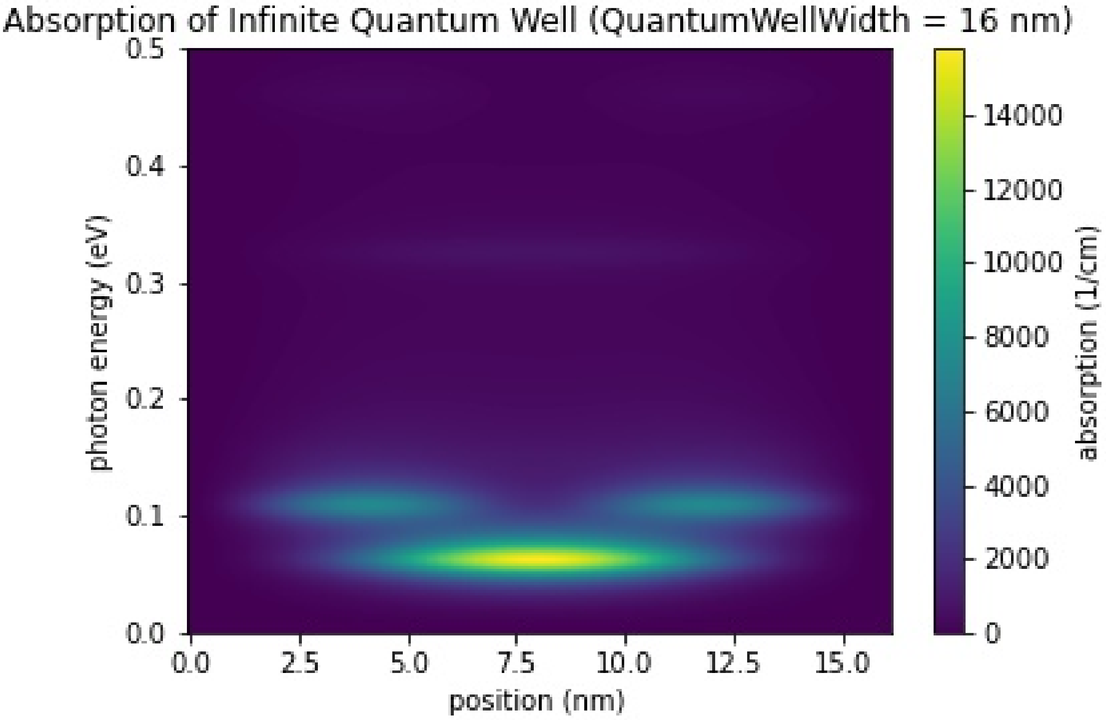


Figure 4.12.1.34: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 16 nm)

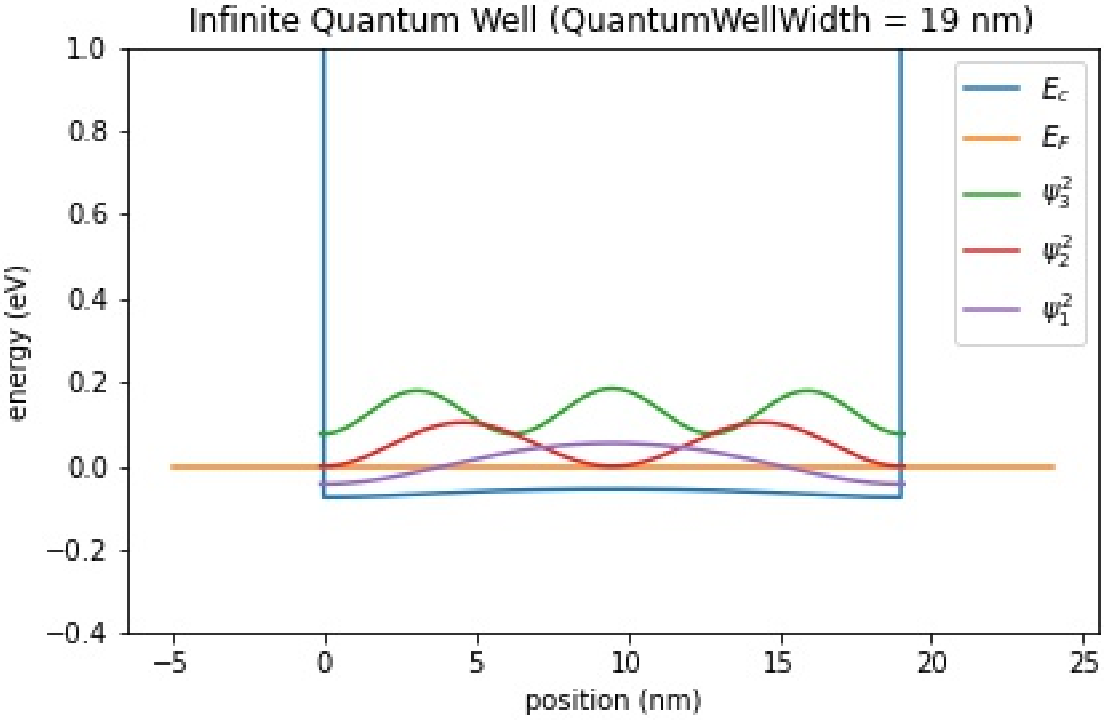


Figure 4.12.1.35: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 19 nm)

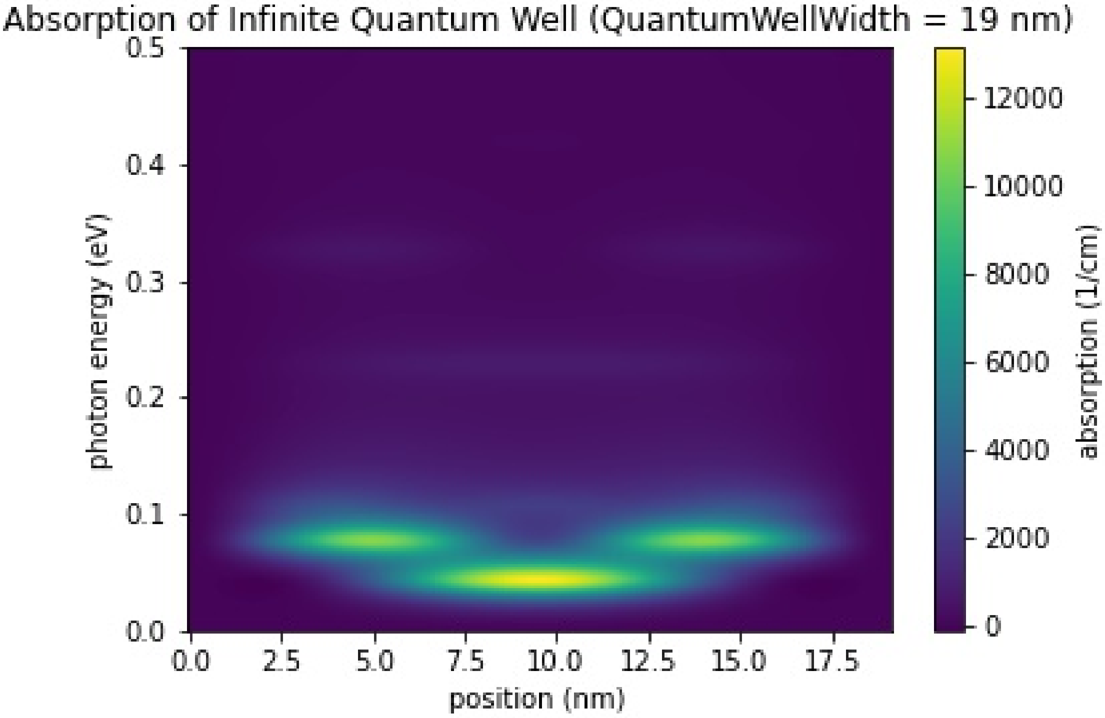


Figure 4.12.1.36: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 19 nm)

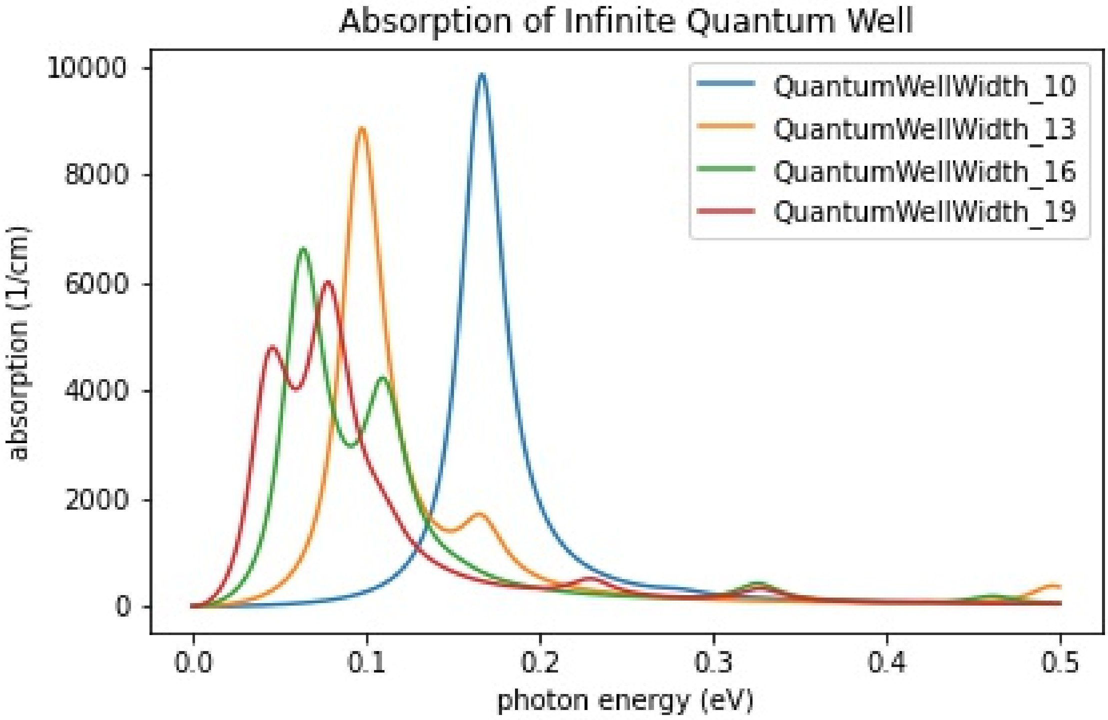


Figure 4.12.1.37: Calculated absorption spectra $\alpha(E)$ of an infinite quantum well for different well widths

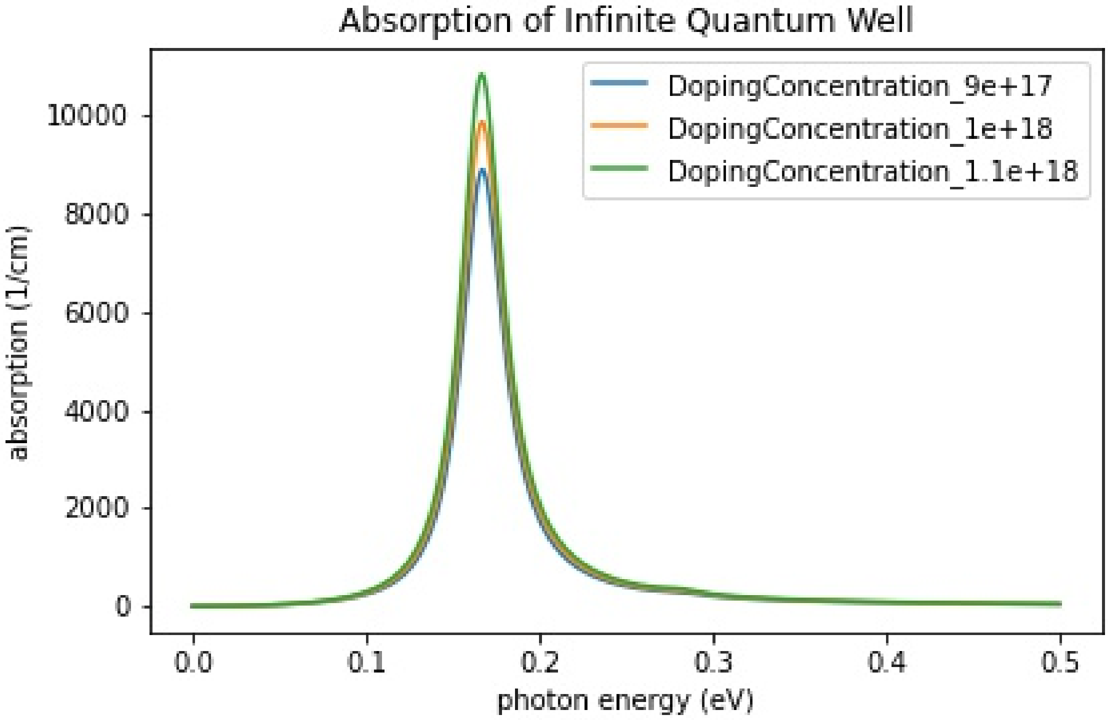


Figure 4.12.1.38: Calculated absorption spectra $\alpha(E)$ of an infinite quantum well for different doping concentrations

Intersubband transitions in InGaAs/AlInAs multiple quantum well systems

This tutorial calculates the eigenstates of a single, double and triple quantum wells. It compares the energy levels and wave functions of the single-band effective mass approximation with the 8-band $\mathbf{k} \cdot \mathbf{p}$ model. Finally, the intersubband absorption spectrum is calculated.

The following input files were used:

- Single Quantum Well
  - `1DSirtoriPRB1994_OneWell_sg_self-consistent_nnp.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_OneWell_kp_self-consistent_nnp.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
  - `1DSirtoriPRB1994_OneWell_sg_quantum-only_nnp.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_OneWell_kp_quantum-only_nnp.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
- Two coupled Quantum Wells
  - `1DSirtoriPRB1994_TwoCoupledWells_sg_self-consistent_nnp.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_TwoCoupledWells_kp_self-consistent_nnp.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
  - `1DSirtoriPRB1994_TwoCoupledWells_sg_quantum-only_nnp.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_TwoCoupledWells_kp_quantum-only_nnp.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
- Three coupled Quantum Wells
  - `1DSirtoriPRB1994_ThreeCoupledWells_sg_self-consistent_nnp.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_ThreeCoupledWells_kp_self-consistent_nnp.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
  - `1DSirtoriPRB1994_ThreeCoupledWells_sg_quantum-only_nnp.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_ThreeCoupledWells_kp_quantum-only_nnp.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)

This tutorial aims to reproduce Figs. 4 and 5 of *[SirtoriPRB1994]*.

This tutorial nicely demonstrates that for the ground state energy the single-band effective mass approximation is sufficient whereas for the higher lying states a nonparabolic model, like the 8-band $\mathbf{k} \cdot \mathbf{p}$ approximation, is necessary. This is important for e.g. quantum cascade lasers where higher lying states have a dominant role.

Layer sequence

We investigate three structures:

a) a single quantum well

b) two coupled quantum wells

c) three coupled quantum wells

Material parameters

We use $In_{0.53}$ $Ga_{0.47}$ As as the quantum well material and $Al_{0.48}$ $In_{0.52}$ As as the barrier material. Both materials are lattice matched to the substrate material InP. Thus we assume that the InGaAs and AlInAs layers are unstrained with respect to the InP substrate. The publication *[SirtoriPRB1994]* lists the following material parameters:

conduction band offset	$Al_{0.48}$ $In_{0.52}$ As / $In_{0.53}$ $Ga_{0.47}$ As	0.510 eV
conduction band effective mass	$Al_{0.48}$ $In_{0.52}$ As	0.072 m_0
conduction band effective mass	$In_{0.53}$ $Ga_{0.47}$ As	0.043 m_0

The temperature is set to 10 Kelvin.

Method

Single-band effective mass approximation

Because our structure is doped, we have to solve the single-band Schrödinger-Poisson equation self-consistently. The doping is such that the electron ground state is below the Fermi level and all other states are far away from the Fermi level, i.e. only the ground state is occupied and contributes to the charge density.

For *nextnano++* we use:

```
# '0' solve Schrödinger equation only
# '1' solve Schrödinger and Poisson equations self-consistently
$SELF_CONSISTENT = 1

run{
  !IF($SELF_CONSISTENT)
    poisson{ }
    quantum_poisson{ iterations = 50 }  # Schrödinger-Poisson
  !ELSE
    quantum{ }                           # Schrödinger only
  !ENDIF
}

quantum {
  region{
    ...
    Gamma{                              # single-band
      num_ev = 3                        # 3 eigenstates
    }
```

Note

Single-band eigenstates are two-fold spin degenerate.

The Fermi level is always equal to 0 eV in our simulations and the band profile is shifted accordingly to meet this requirement.

8-band k.p approximation

Old version of this tutorial:

Becauce both, the single-band and the 8-band $\mathbf{k}\cdot\mathbf{p}$ ground state energy and the corresponding wave functions are almost identical, we can read in the self-consistently calculated electrostatic potential of the single-band approximation and calculate for this potential the 8-band $\mathbf{k}\cdot\mathbf{p}$ eigenstates and wave functions for $k_{||}=0$.

Note

One $\mathbf{k}\cdot\mathbf{p}$ eigenstate for each spin component.

New version of this tutorial:

We provide input files for:

a) self-consistent single-band Schrödinger equation (because the structure is doped)

b) single-band Schrödinger equation (without self-consistency)

c) 8-band $\mathbf{k}\cdot\mathbf{p}$ single-band Schrödinger equation (without self-consistency)

For a), although the structure is doped, the band bending is very small. Thus we omit for the single-band / $\mathbf{k}\cdot\mathbf{p}$ comparison in b) and c) the self-consistent cycle.

Results

Single quantum well

Figure 4.12.1.39 shows the lowest two electron eigenstates for an $In_{0.53}$ $Ga_{0.47}$ As / $Al_{0.48}$ $In_{0.52}$ As quantum well structure calculated with single-band effective mass approximation and with a nonparabolic 8-band $\mathbf{k}\cdot\mathbf{p}$ model.

The energies (and square of the wave functions ψ^2) for the ground state are identical in both models but the second eigenstate differs substantially. Clearly the single-band model leads to an energy which is far too high for the upper state.

Our calculated value for the intersubband transition energy E_{12} of 255 meV compares well with both, the calculated value of *[SirtoriPRB1994]* (258 meV) and their measured value (compare with absorption spectrum in Fig. 4 of *[SirtoriPRB1994]*).

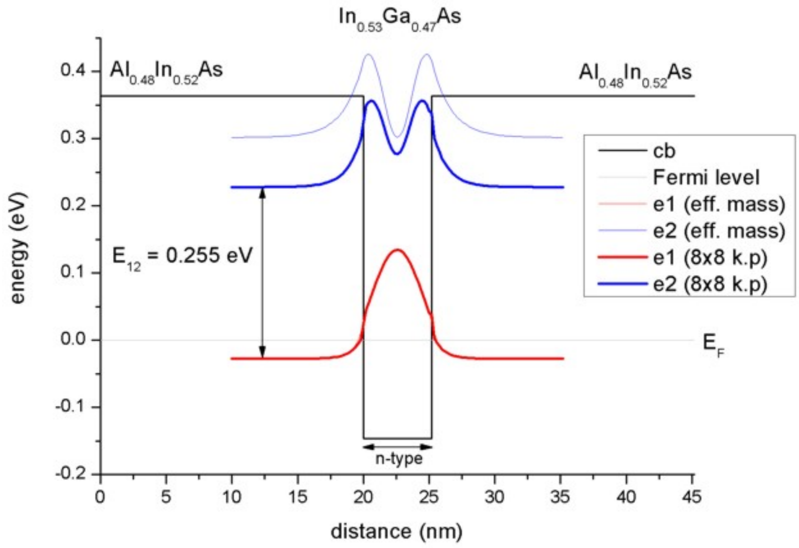


Figure 4.12.1.39: Conduction band edge, Fermi level and confined electron states of a quantum well

The calculated intersubband dipole moments are:

- z_{12} = 1.55 nm (single-band)

For comparison: z_{12} = 1.53 nm (exp.), z_{12} = 1.48 nm (th.) (*[SirtoriPRB1994]*)

The influence of doping on the eigenenergies is negligible (smaller than 1 meV).

Two coupled quantum wells

Figure 4.12.1.40 shows the lowest three electron eigenstates for an $In_{0.53}$ $Ga_{0.47}$ As / $Al_{0.48}$ $In_{0.52}$ As double quantum well structure calculated with single-band effective mass approximation and with a nonparabolic 8-band $\mathbf{k}\cdot\mathbf{p}$ model.

The energies (and square of the wave functions ψ^2) for the ground state are very similar in both models but the second and especially the third eigenstate differ substantially. Clearly the single-band model leads to energies which are far too high for the higher lying states.

Our calculated values for the intersubband transition energies E_{12} = 150 meV and E_{13} = 267 meV compare well with both, the calculated values of *[SirtoriPRB1994]* (150 meV and 271 meV) and their measured values (compare with absorption spectrum in Fig. 5 (a) of *[SirtoriPRB1994]*).

The calculated intersubband dipole moments are:

- z_{12} = 1.61 nm (single-band)
- z_{13} = 1.11 nm (single-band)

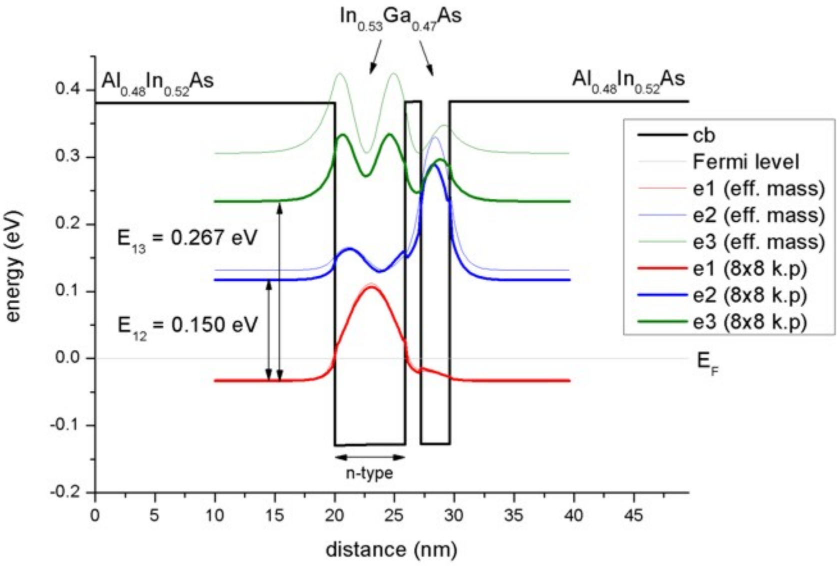


Figure 4.12.1.40: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

For comparison: z_{12} = 1.64 nm (exp.), z_{12} = 1.65 nm (th.) (*[SirtoriPRB1994]*)

The influence of doping on the eigenenergies is almost negligible (between 0 and 2 meV).

Three coupled quantum wells

Figure 4.12.1.41 shows the lowest four electron eigenstates for an In$_{0.53}$ Ga$_{0.47}$ As / Al$_{0.48}$ In$_{0.52}$ As triple quantum well structure calculated with single-band effective mass approximation and with a nonparabolic 8-band $\mathbf{k} \cdot \mathbf{p}$ model.

The energies (and square of the wave functions ψ^2) for the ground state are similar in both models but the second and especially the third and forth eigenstates differ substantially. Clearly the single-band model leads to energies which are far too high for the higher lying states.

Our calculated values for the intersubband transition energies E_{12} = 118 meV, E_{13} = 261 and E_{14} = 370 meV compare well with both, the calculated values of *[SirtoriPRB1994]* (116 meV, 257 meV and 368 meV) and their measured values (compare with absorption spectrum in Fig. 5 (b) of *[SirtoriPRB1994]*).

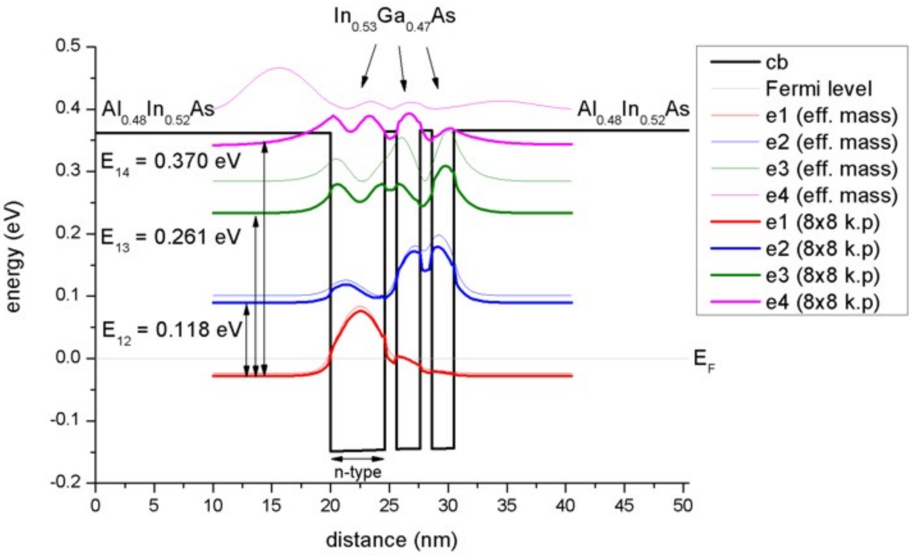


Figure 4.12.1.41: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells

The calculated intersubband dipole moments are:

- z_{12} = 1.81 nm (single-band)
- z_{13} = 0.77 nm (single-band)
- z_{14} = 0.30 nm (single-band)

For comparison: z_{12} = 1.86 nm (exp.), z_{12} = 1.84 nm (th.) *[SirtoriPRB1994]*

The influence of doping on the eigenenergies is almost negligible (between 0 and 4 meV).

— Begin —

The following documentation and figures were generated automatically using *nextnanopy*.

The following figures have been generated using *nextnano³*. Self-consistent Schrödinger-Poisson calculations have been performed for three different structures.

- Single Quantum Well
- Two coupled Quantum Wells
- Three coupled Quantum Wells

The single-band effective mass and the 8-band $\mathbf{k} \cdot \mathbf{p}$ results are compared to each other. In both cases the wave functions and the quantum density are calculated self-consistently. The $\mathbf{k} \cdot \mathbf{p}$ quantum density has been calculated taking into account the solution at different $k_{||}$ vectors.

The absorption spectrum has been calculated using a simple model assuming a parabolic energy dispersion. The dipole moment $z_{ij} =< i|z|j >$ has been evaluated only at $k_{||} = 0$. The subband density is used to calculate the absorption spectrum. For the $\mathbf{k} \cdot \mathbf{p}$ calculation, the density was calculated taking into account a nonparabolic energy dispersion, i.e. including all relevant $k_{||}$ vectors.

Quantum Well (single-band)

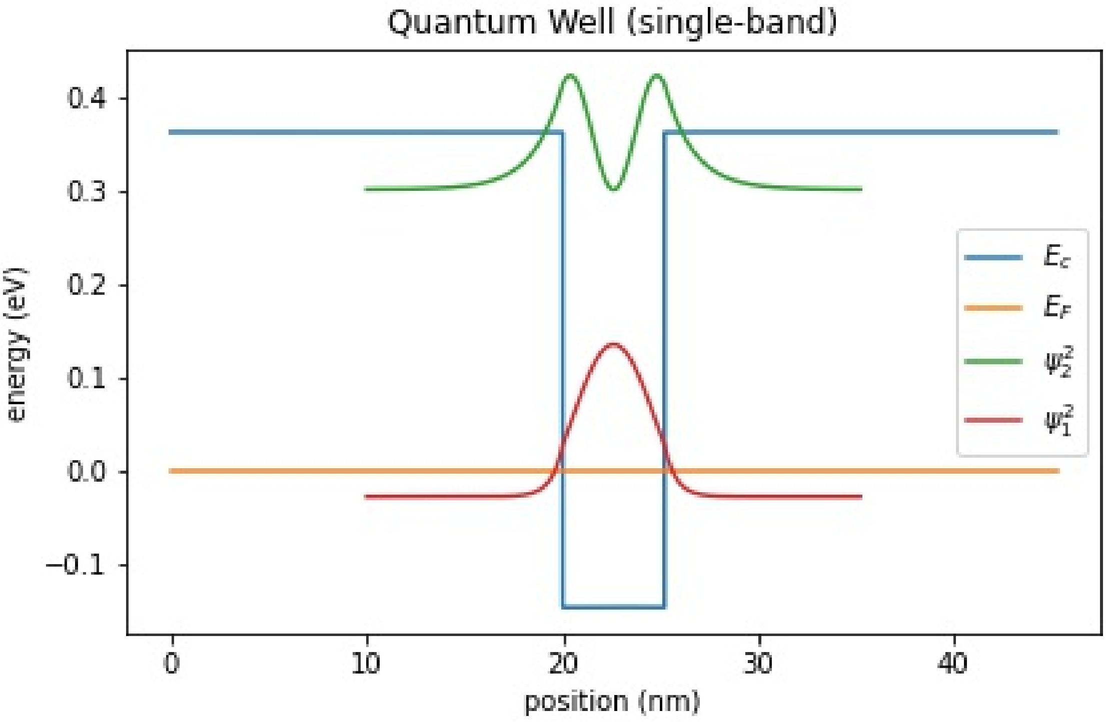


Figure 4.12.1.42: Conduction band edge, Fermi level and confined electron states of a quantum well

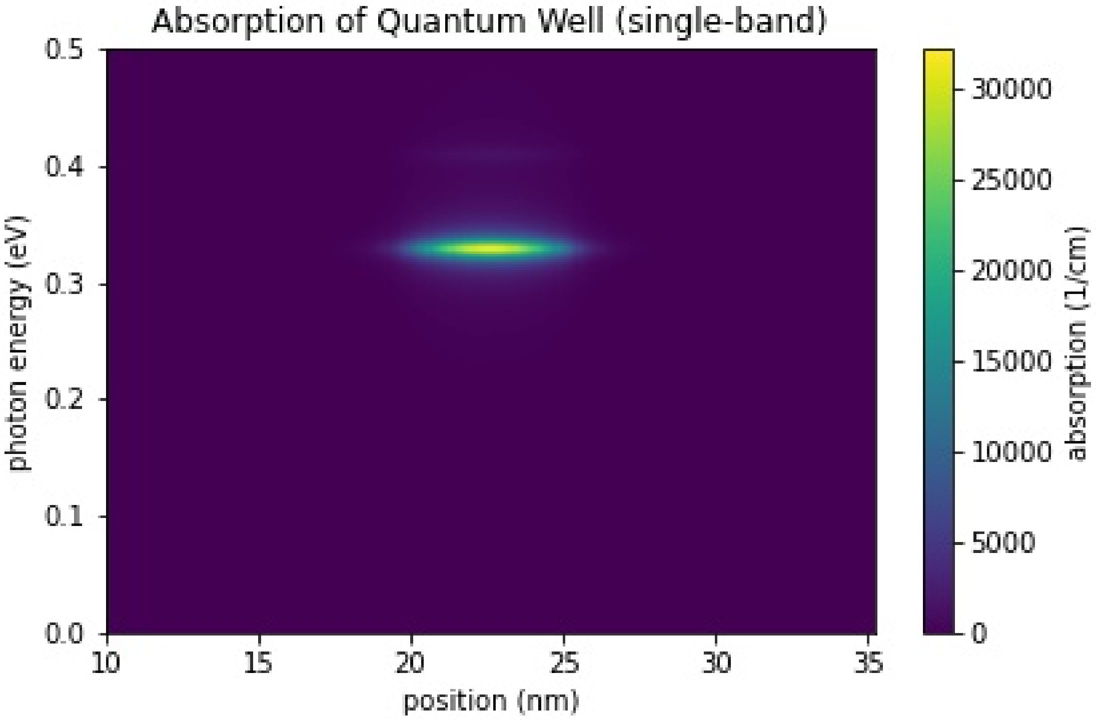


Figure 4.12.1.43: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of a quantum well

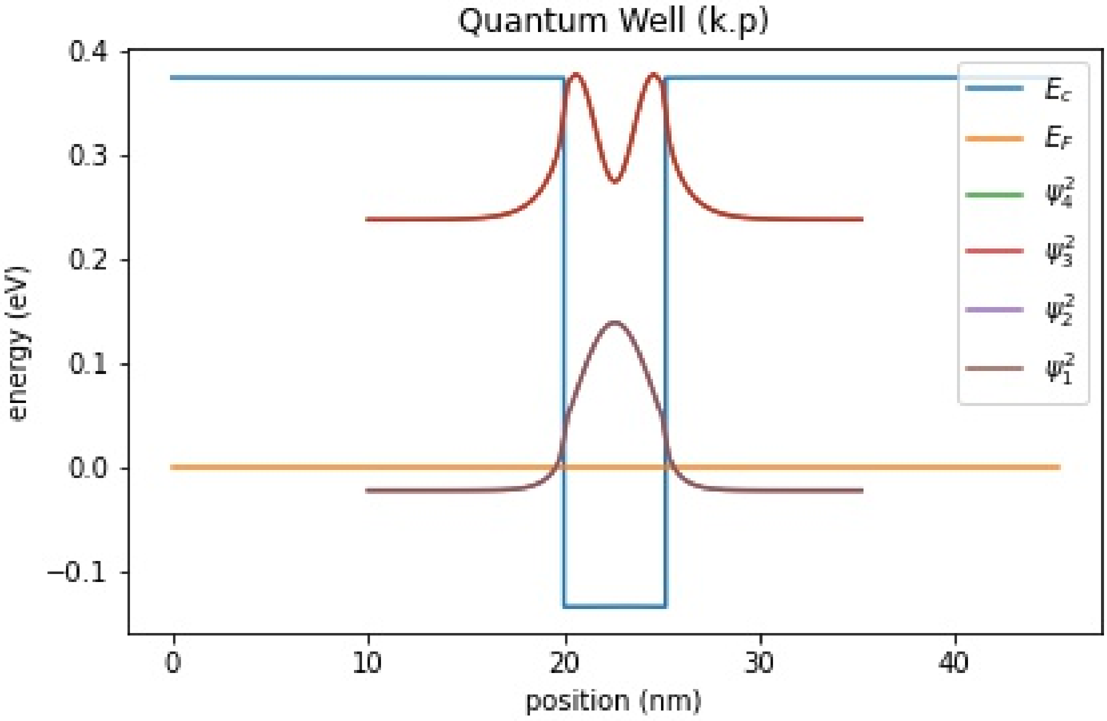


Figure 4.12.1.44: Conduction band edge, Fermi level and confined electron states of a quantum well

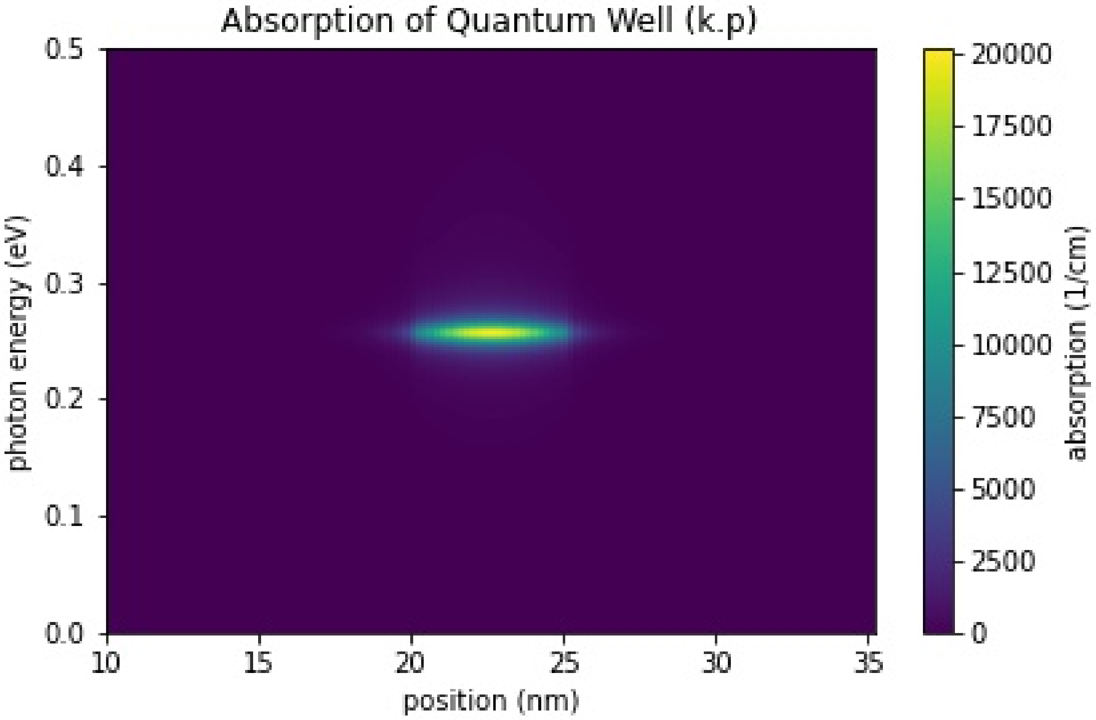


Figure 4.12.1.45: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of a quantum well

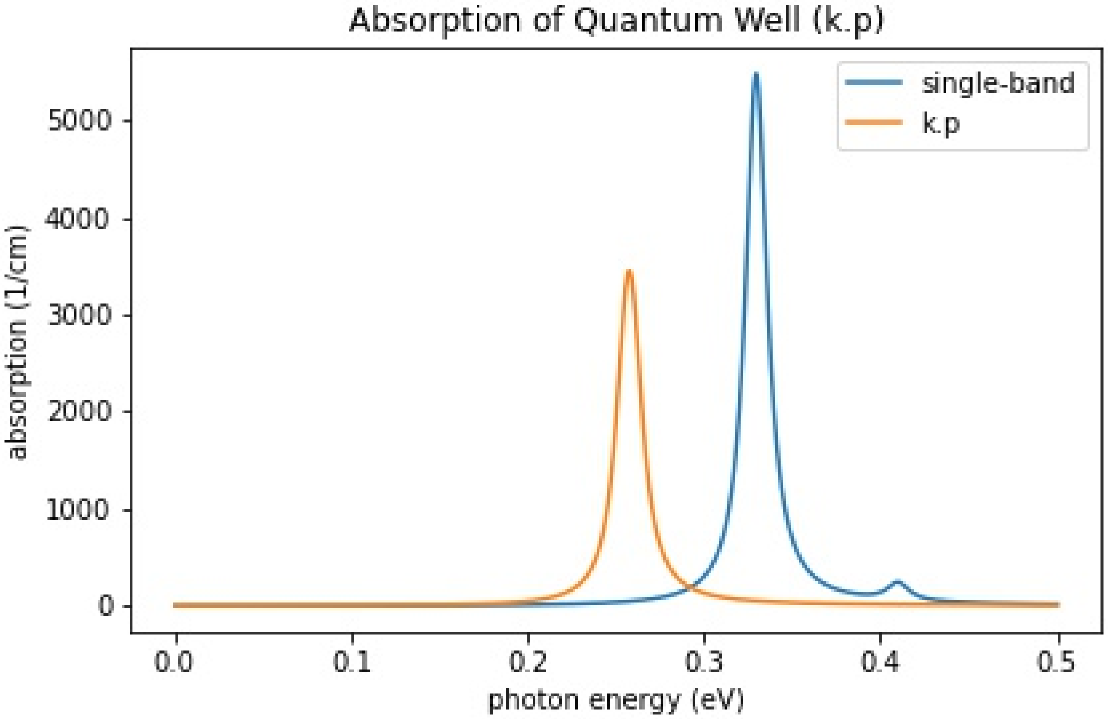


Figure 4.12.1.46: Calculated absorption spectra $\alpha(E)$ of a quantum well

Quantum Well (k.p)

Two Coupled Quantum Wells (single-band)

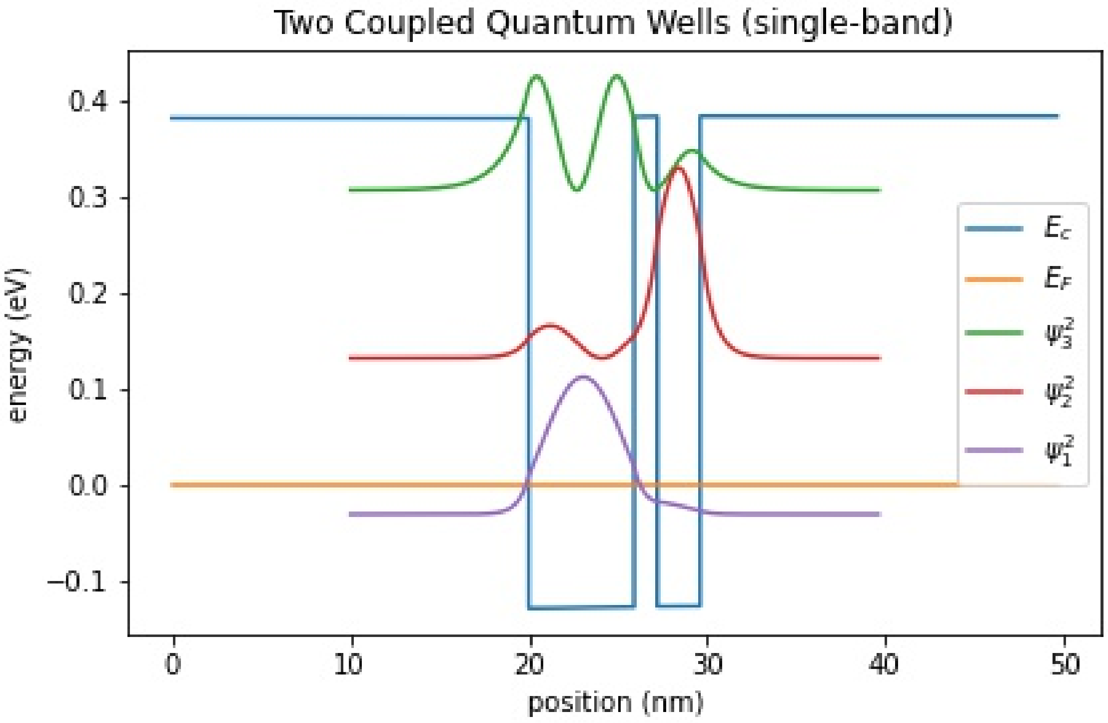


Figure 4.12.1.47: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

Two Coupled Quantum Wells (k.p)

Three Coupled Quantum Wells (single-band)

Three Coupled Quantum Wells (k.p)

— End —

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

Interband absorption of a GaAs cylindrical quantum wire

Section author: Naoki Mitsui (simulation), Brandon Loke (write-up and visualisation)

This tutorial calculates the optical spectrum of a GaAs cylindrical quantum wire with infinite barriers. The formulas used to calculate absorption spectra will be highlighted and a brief explanation of the output files will be given.

For the detailed scheme of the calculation of the optical matrix elements or absorption spectrum, please see our 1D optics tutorial: *Optical absorption for interband and intersubband transitions* For the corresponding tutorial for the **intraband** absorption, please see *Intersubband absorption of a GaAs cylindrical quantum wire* Input file:

- *2Dcircular_infinite_wire_GaAs_inter_nnp.in*

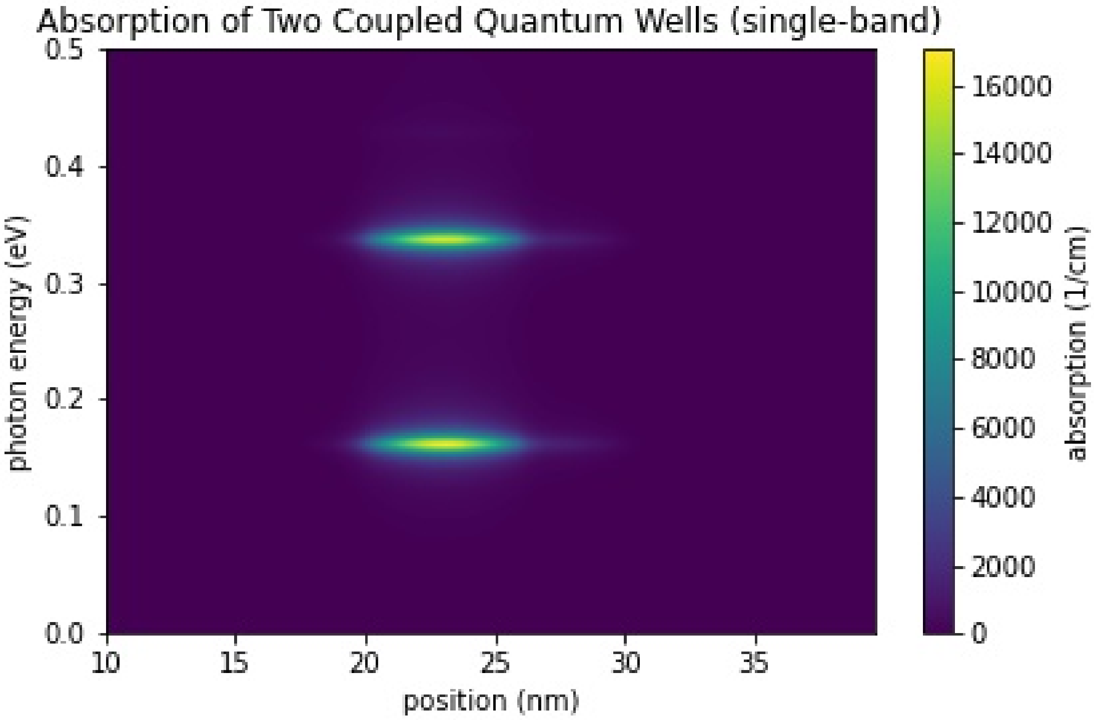


Figure 4.12.1.48: Calculated spatially resolved absorption spectrum:math:*alpha(x,E)* of two coupled quantum wells

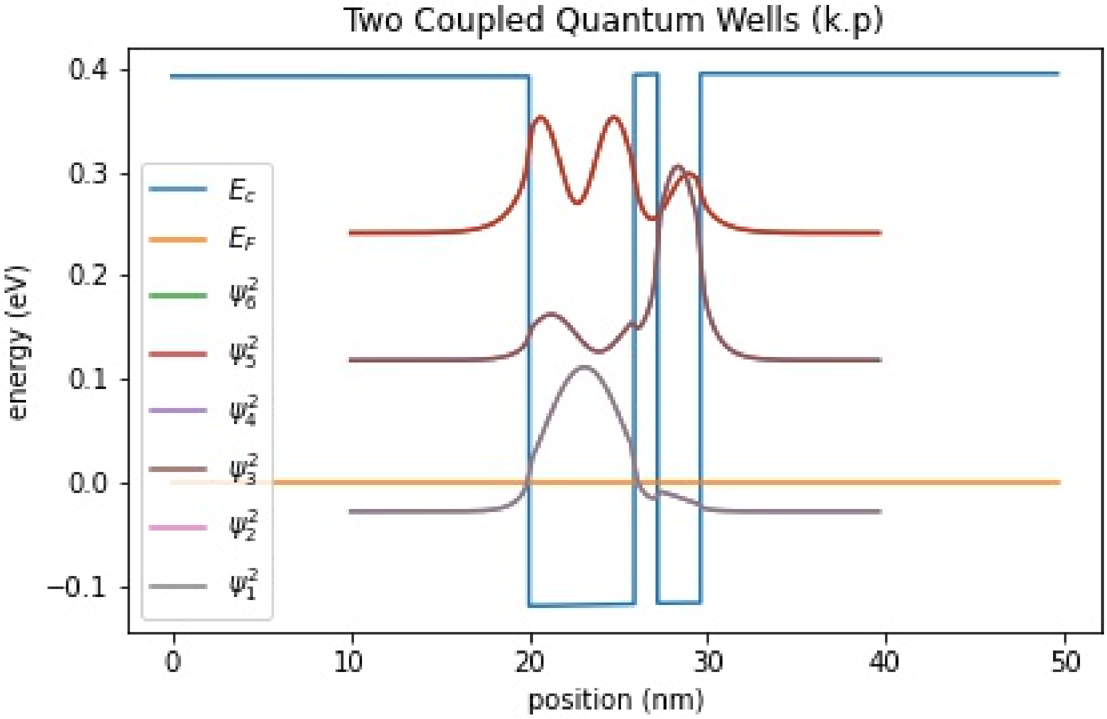


Figure 4.12.1.49: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

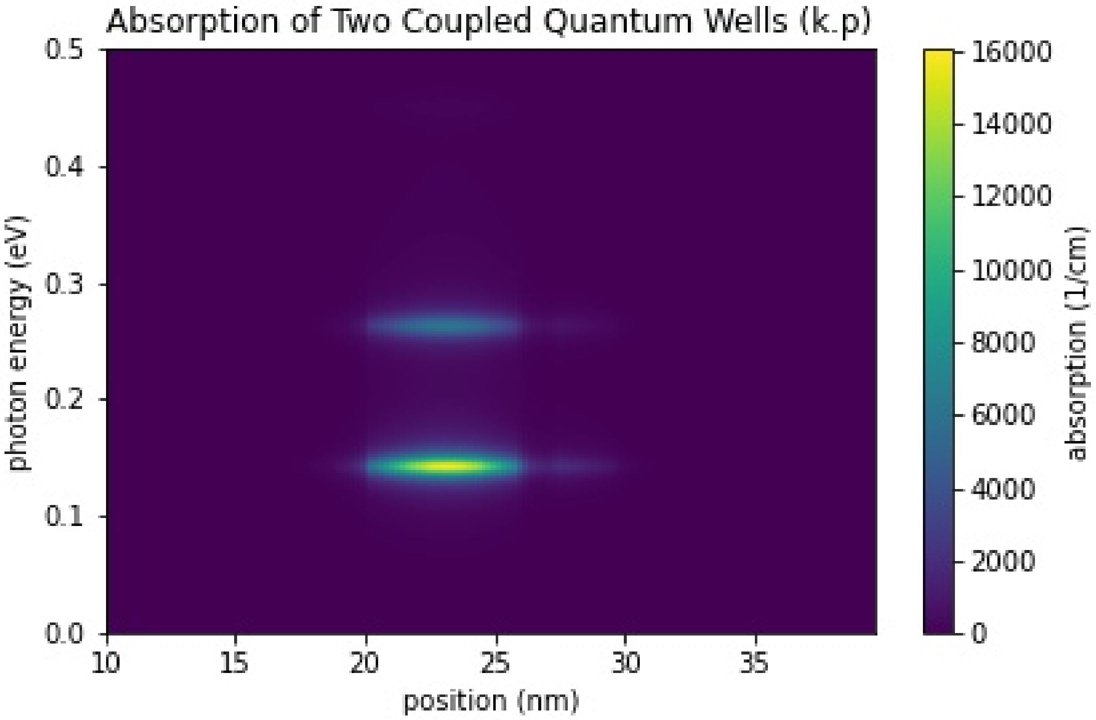


Figure 4.12.1.50: Calculated spatially resolved absorption spectrum:math:*alpha(x,E)* of two coupled quantum wells

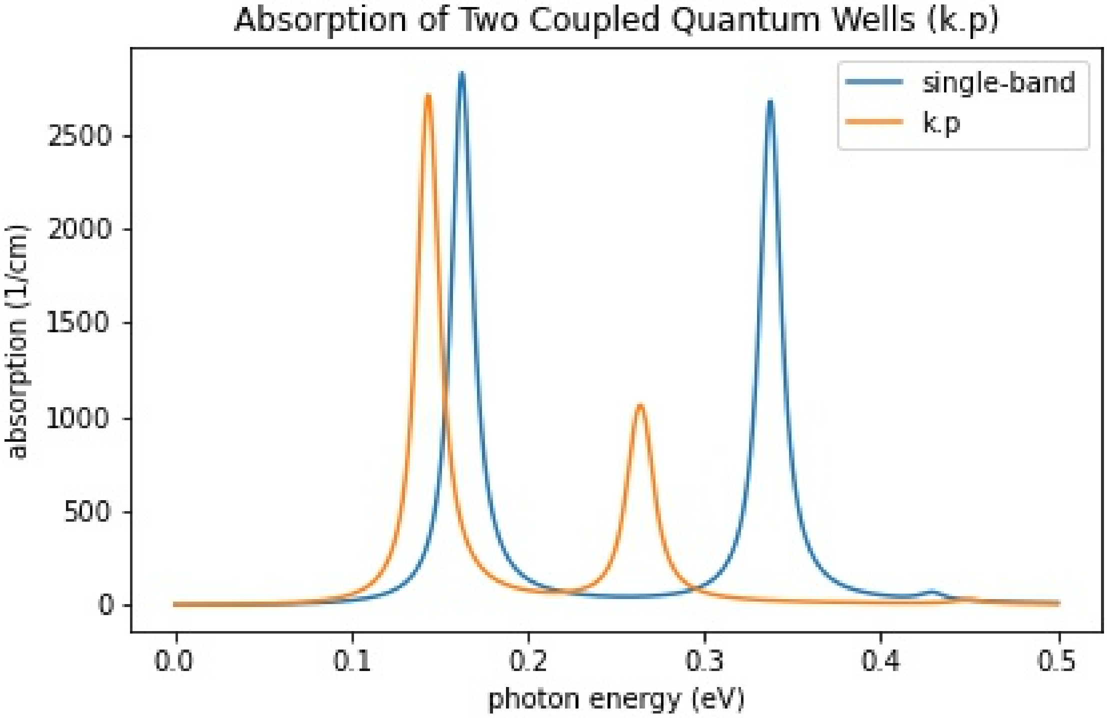


Figure 4.12.1.51: Calculated absorption spectra $\alpha(E)$ of two coupled quantum wells

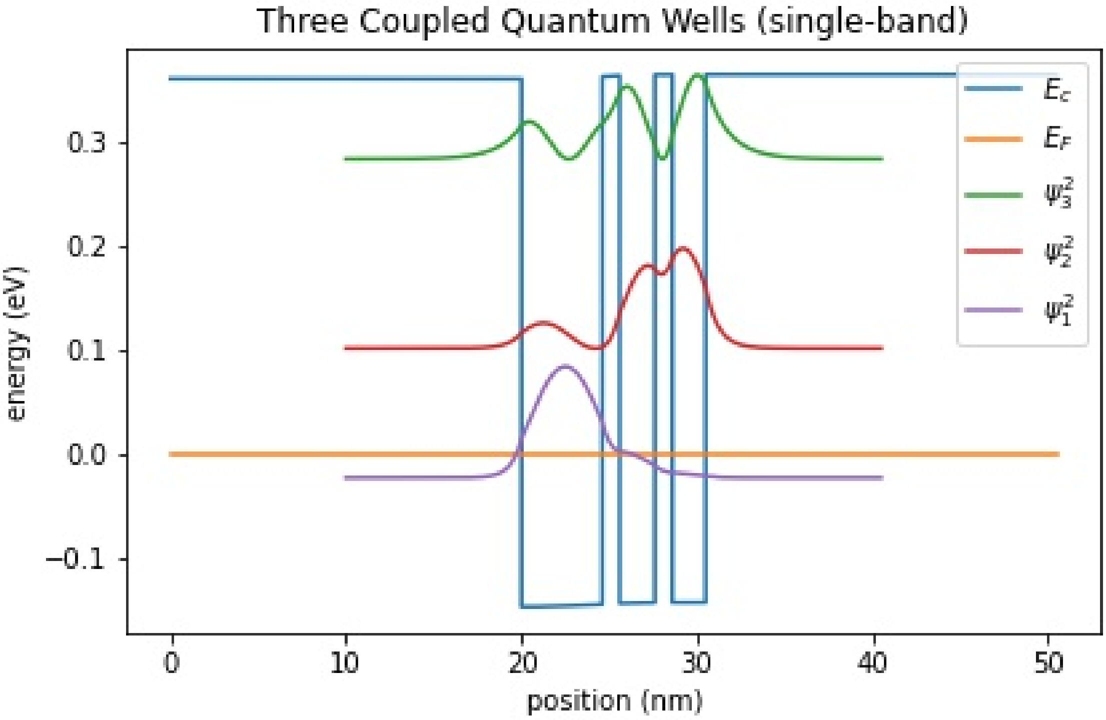


Figure 4.12.1.52: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells

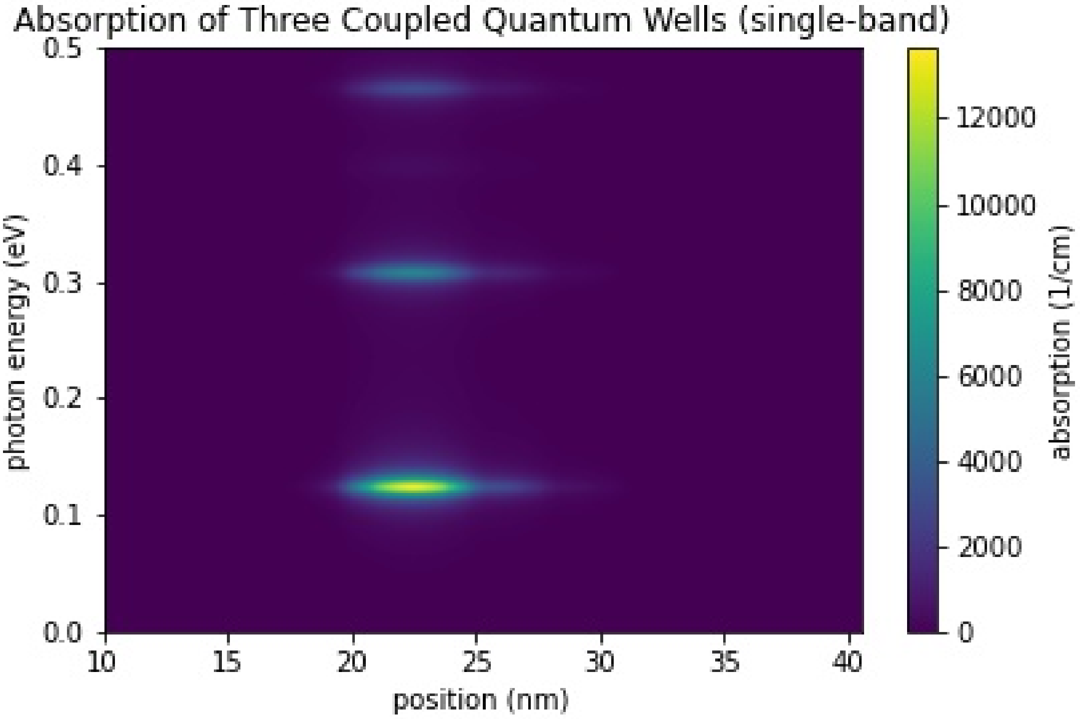


Figure 4.12.1.53: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of three coupled quantum wells

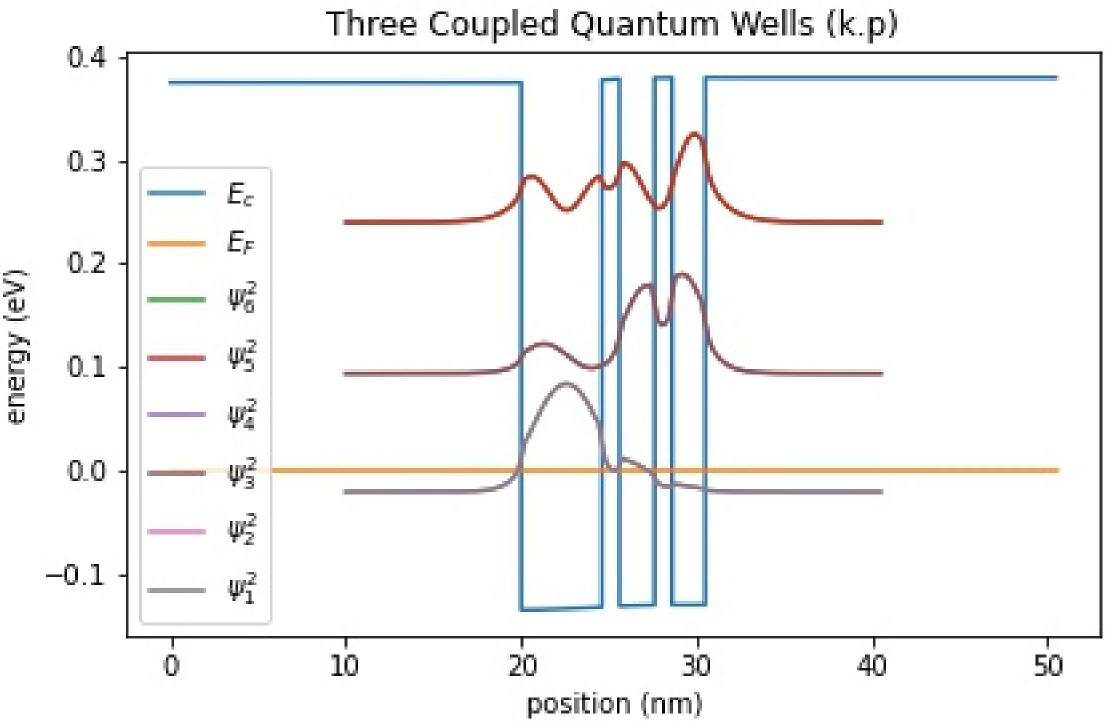


Figure 4.12.1.54: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells

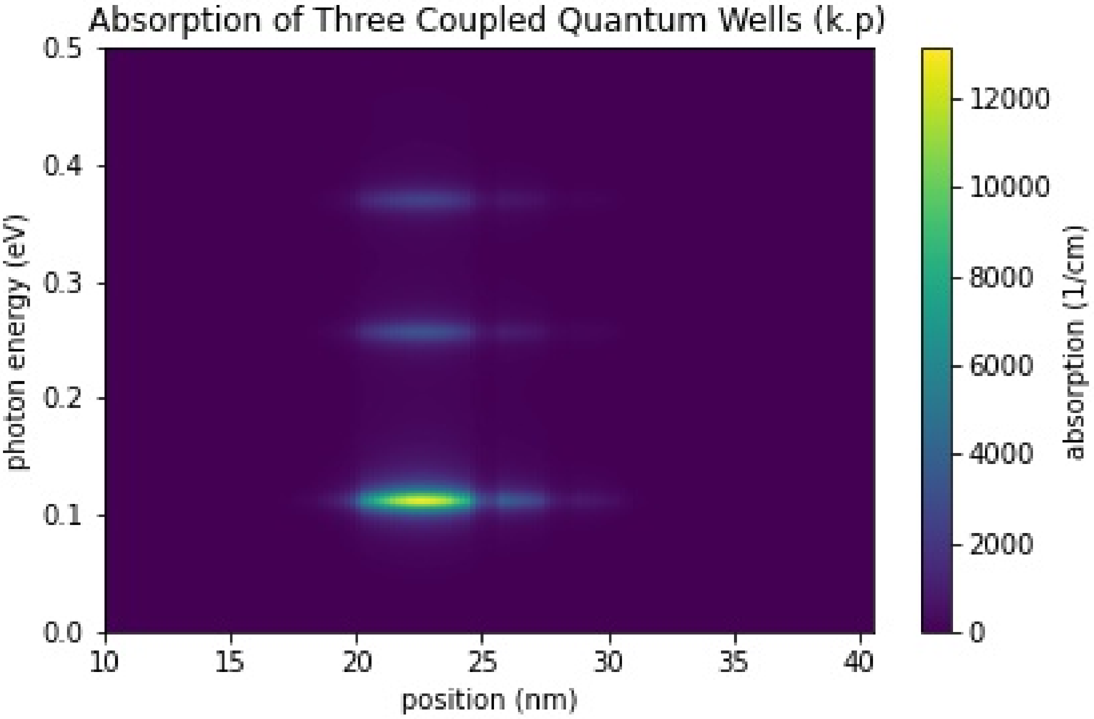


Figure 4.12.1.55: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of three coupled quantum wells

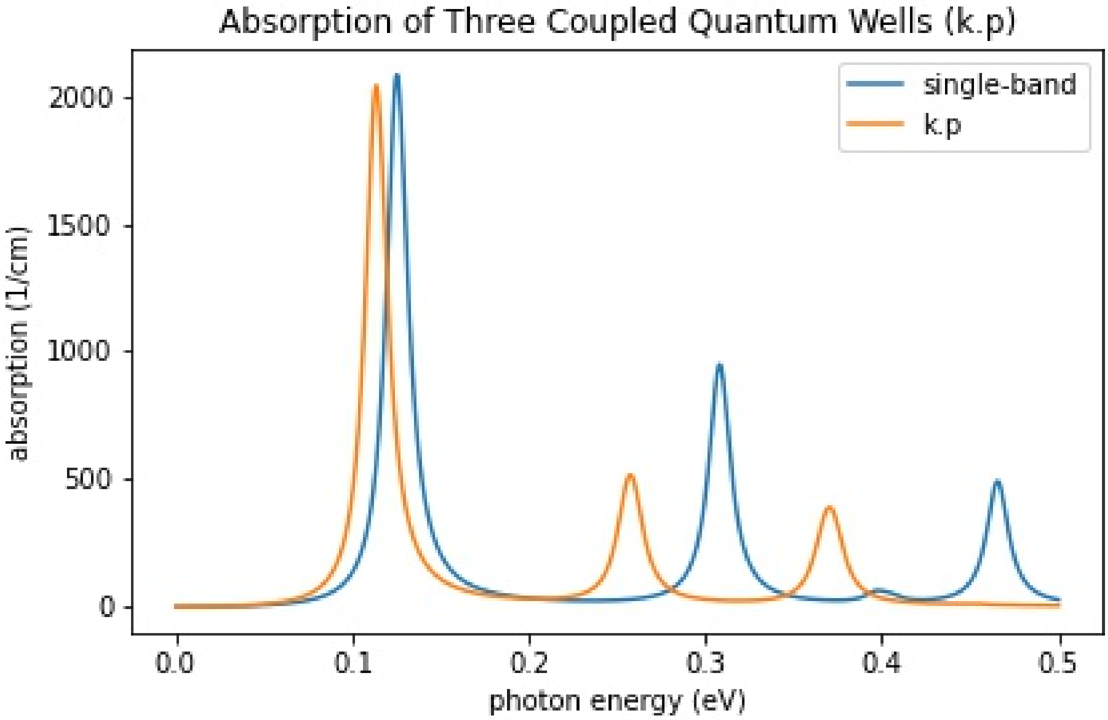


Figure 4.12.1.56: Calculated absorption spectra $\alpha(E)$ of three coupled quantum wells

Note

Figures in this tutorial will be generated with nextnanopy.

The corresponding Jupyter Notebook used to generate the figures in this tutorial can be found here at 2DInterbandQuantumCylinder.ipynb.

Structure

The above figures show the Gamma band edge of the circular GaAs region and the barrier region. We model the infinite barrier by assigning 100 eV for the band edge of AlAs barrier region from `database{ }` section. Please see the input file for the details.

The parameters used in this simulation are as follows.

Property	Symbol	Value [unit]
quantum wire radius	R	5 [nm]
barrier height	E_b	92 [eV]
effective electron mass	m_e	0.0665
refractive index	n_r	3.3
doping concentation (n-type)	N_D	$5{\cdot}10^{18}$ [cm^{-3}]
linewidth (FWHM)	Γ	0.01 [eV]
temperature	T	300 [K]

The `run{ }` section is specified as follows:

```
run{
    poisson{ }
```

(continues on next page)

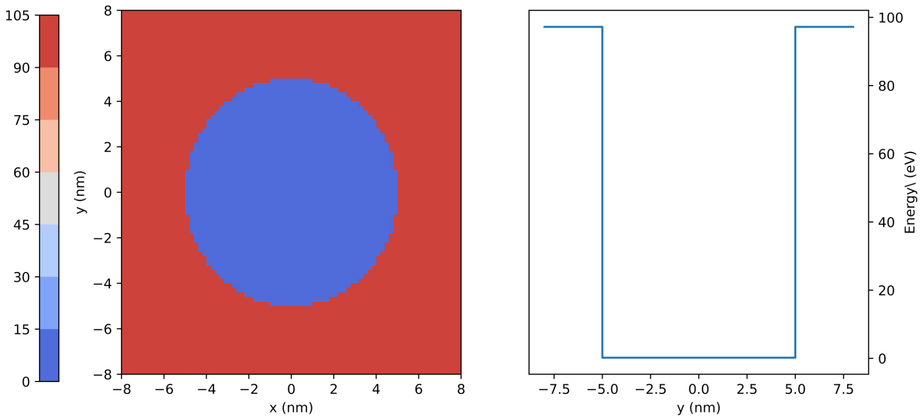


Figure 4.12.1.57: Left: Conduction band edge for cylindrical quantum wire. Right: Slice of the band edge along $x = 0$.

(continued from previous page)

```
    quantum{ }
    optics{ }
}
```

Then the simulation follows these steps:

1. Poisson equation is solved with the setting specified in the *poisson{ }* section.
2. "Schrödinger" equation is solved with the setting specified in the *quantum{ }* section.
3. "Schrödinger" equation is solved again with the setting specified in the *optics{ }* section and optical properties are calculated.

Note

- If `quantum_poisson{ }` is specified instead of `quantum{ }`, Poisson and Schrödinger equations are solved self-consistently.
- `optics{ }` requires that kp8 model is used in the quantum region specified in `quantum{ }`.
- In this tutorial the kp parameters are adjusted so that the conduction and valence bands are decoupled from each other. Thus the single-band Schrödinger equations are solved effectively by the kp solver.

Spectra of optical absorption accompanied by the excitation of charge carriers (state $n \to m$) in condensed matter is calculated on the basis of Fermi's golden rule [*ChuangOpto1995*] in the dimension of (length)$^{-1}$:

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega}\frac{1}{V}\sum_{n>m}\sum_{\mathrm{k}_z} |\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathrm{k}_z)|^2 (f_m(\mathrm{k}_z) - f_n(\mathrm{k}_z))\mathcal{L}(E_n(\mathrm{k}_z)-E_m(\mathrm{k}_z)-\hbar\omega), \tag{4.12.1.5}$$

where

- k_z is the Bloch wave vector along translation-invariant directions. In 2D simulation this is 1D vector.
- $E_n(\mathrm{k}_z)$ is the energy of eigenstate n. The first sum runs over the pair of states where $E_n(\mathrm{k}_z) > E_m(\mathrm{k}_z)$.
- $f_n(\mathrm{k}_z)$ is the occupation of eigenstate n.
- $\vec{\epsilon}$ is the optical polarization vector defined in *optics{ quantum_spectra{ polarization{ } } }*.
- $\vec{\pi} = \vec{p} + \frac{1}{4m_0c^2}(\sigma\times\nabla V)$ where $\vec{p}$ is the canonical momentum operator and $\frac{1}{4m_0c^2}(\sigma\times\nabla V)$ is the contribution of spin-orbit interaction.

- $\vec{\pi}_{nm}(\mathrm{k}_z) = \langle n|\vec{\pi}|m\rangle$.
- $\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathrm{k}_z)$ is known as the optical matrix elements.
- $\mathcal{L}(E_n(\mathbf{k}_z) - E_m(\mathbf{k}_z) - \hbar\omega)$ **is the energy broadening function.**
  - When `energy_broadening_lorentzian` is specified in *optics{ quantum_spectra{ energy_broadening_lorentzian } }*,

    $$\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\pi}\frac{\Gamma/2}{(E_n - E_m - \hbar\omega) + (\Gamma/2)^2}$$

    where Γ is the FWHM defined by `energy_broadening_lorentzian`.
  - When `energy_broadening_gaussian` is specified in *optics{ quantum_spectra{ energy_broadening_gaussian } }*,

    $$\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\sqrt{2\pi}\sigma}\exp\Big\{\Big(-\frac{(E_n - E_m - \hbar\omega)^2}{2\sigma^2}\Big)\Big\}$$

    where `energy_broadening_lorentzian` defines the FWMH $\Gamma = 2\sqrt{\ln 2}\cdot\sigma$
  - When neither `energy_broadening_lorentzian` nor `energy_broadening_gaussian` is specified in *optics{ quantum_spectra{ } }*, $\mathcal{L}$ is replace by the delta function $\delta(E_n - E_m - \hbar\omega)$.
  - It is also possible to include both Lorentzian and Gaussian broadening (Voigt profile).

The detailed calculation scheme of the optical matrix elements $\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathrm{k}_z)$ and the absorption spectrum α is described in *Optical absorption for interband and intersubband transitions*.

Results

Absorption

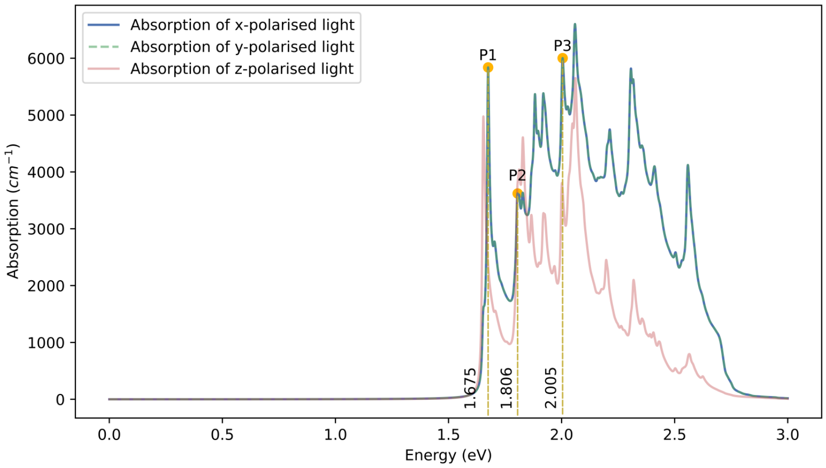


Figure 4.12.1.58: Calculated absorption spectrum $\alpha(\vec{\epsilon}, E)$ for $\vec{\epsilon} = \hat{x}, \hat{y}, \hat{z}$.

Figure 4.12.1.58 shows the calculated $\alpha(\vec{\epsilon}, E)$ specified in *\Optics\absorption_~.dat* for each polarization x, y, and z. The absorptions for the x- and y-polarisation are identical due to the rotational symmetry of the quantum cylinder in the x-y plane. It is observed that there are peaks at 1.675 eV (P1), 1.806 eV (P2) and 2.005 eV (P3).

Note

$\alpha(\vec{\epsilon}, E)$ for z-polarization is generally non-zero in the calculation through k.p model. This is because the eigenstates above the conduction band edge can have the component of valence band Bloch functions and vice versa (band-mixing).

Eigenvalues, transition energies, and occupations

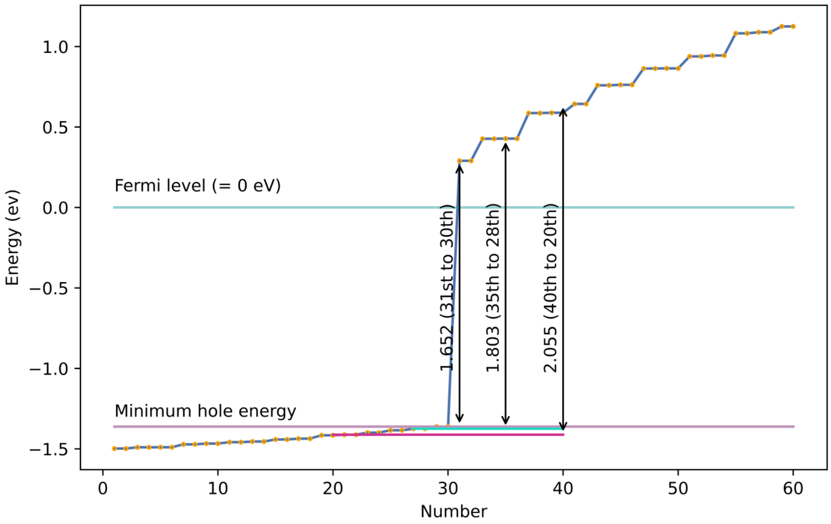


Figure 4.12.1.59: Calculated energy spectrum and the minimum hole energy.

Figure 4.12.1.59 shows the calculated energy eigenvalues at k$_z = 0$ specified in *\Quantum\energy_spectrum_~.dat*.

Please note that the output in *Quantum* counts the eigenstates with different spins individually when k.p model is used, while they are counted jointly in *Optics*.

The valence band states lie below the Fermi level (0 eV). The minimum hole energy is indicated in Figure 4.12.1.59 with the purple line. It can be seen through a comparison with Figure 4.12.1.58 that the peak in absorption spectrum at P1 corresponds to the transition energy from the minimum hole energy level to the first conduction band state (number 31, 32). Similarly, the peak at P2 corresponds to the transition energy between the minimum hole energy state and the second conduction band state (number 33-36).

The occupation probabilities for each state can be checked from *\Optics\occupation_disp_~.dat*as a function of the 1D Bloch wave vector k$_z$:

In the above figure, the occupation probabilities are plotted for the 1st and the 16th excited state. The 16th excited state corresponds to the lowest conduction band level.

Note

The eigenstates with different spins are counted individually in *Quantum* when k.p model is used, while they are counted jointly in *Optics*.

For example, the two ground states in the conduction band counted as no.31 and 32 in Figure 4.12.1.59 due to spin are put together as one eigenstate in *Optics*. Thus *\Optics\occupation_disp_~_kp8_16.dat* shows the

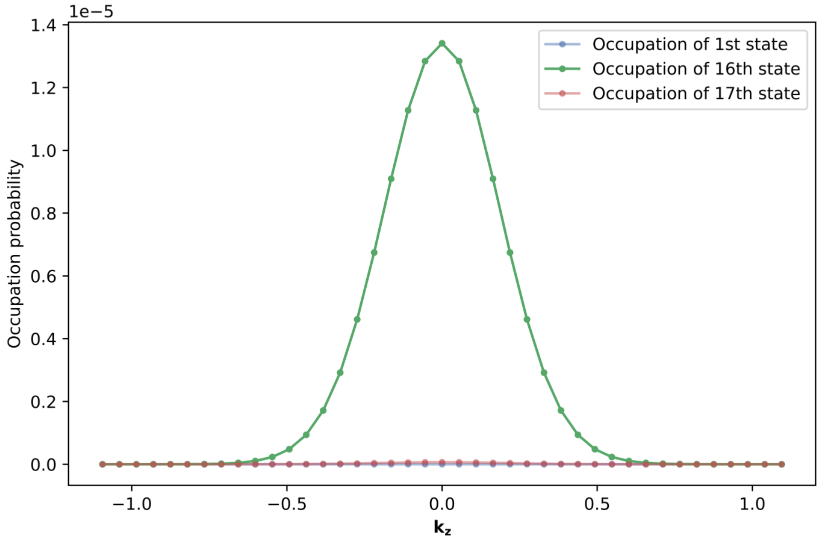


Figure 4.12.1.60: Calculated occupation probabilities for the ground state and 16th excited state as a function of k_z.

occupation of the ground state in the conduction band and *\Optics\occupation_disp_~_kp8_2.dat* and *\Optics\occupation_disp_~_kp8_17.dat* show the 1st excited state in the conduction band (number 33 & 34) in Figure 4.12.1.59.

At $T = 300\mathrm{K},\ k_B T \simeq 0.026$ eV , which is insufficient energy to excite electron carriers to the upper conduction band states.

From the above data of eigenvalues and occupations, we could see which pair of states contributes to each peak in the absorption spectrum Figure 4.12.1.58. In order to understand the magnitude of the peaks and why some pairs of states do not appear as peaks, we will see the output data for $|\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathrm{k}_z)|^2$ next.

Transition intensity (Momentum matrix element)

An important part of the calculation of optical absorption spectra is the transition intensity:

$$T_{nm}(\vec{\epsilon}, \mathrm{k}_z) = \frac{2}{m_0} |\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathrm{k}_z)|^2 \tag{4.12.1.6}$$

which has dimensions of energy [eV].

The intensity at $\mathrm{k}_z = 0$ $\left(T_{nm}(\vec{\epsilon}, \mathrm{k}_z = 0)\right)$ for each pair of states (n, m) is specified in *Optics\transitions_~.txt*. These intensities whose “From” states are the ground state are shown here (x-polarization). We can also check the transition energy of each pair of states.

```
Energy[eV]          From            To              Intensity_k0[eV]                1/
↪Radiative_Rate[s]
2.00196                 10                  19                  5.9913                  ␣
↪     1.74672e-09
2.00394                 10                  20                  1.79227                 ␣
↪         5.83325e-09
1.67437                 13                  16                  19.9021                 ␣
↪         6.2871e-10
1.80179                 14                  17                  6.25494                 ␣
↪         1.85897e-09
```

Above are the transitions of interest. The other transitions are ommitted for brevity. The “From” and “To” states tell us which band the transition belongs to. Using this information, we can identify which peaks (P1, P2, P3) correspond to transitions between which bands. This is marked in Figure 4.12.1.59.

There are also the output files that specify the k-dispersion of the transition intensities for each light polarization in *Optics\transition_disp_~.dat*.

Eigenstates

The probability distributions of the eigenfunctions $|\psi(\mathbf{r})|^2$ can be found in *Quantum\probabilities_~.vtr*. The amplitude of the envelope function on each Bloch function $|S\rangle, |X\rangle, |Y\rangle, |Z\rangle$ can be found in *Quantum\amplitudes_~_SXYZ.vtr*.

The analytcal expression of the eigenfunctions for the cylindrical quantum wire is shown as eq. (4.7.2.1) in this tutorial: *Electron wave functions in a cylindrical well (2D Quantum Corral)*. According to this analytical solution, the eigenfunction has 2 quantum numbers: n for radial direction and l for circumferential direction.

Here the amplitudes of eigenfunctions calculated by single-band model are shown. We can see the optical transition from ground state ($n = 1, l = 0$) occurs only to the states with $l = \pm 1$. The file used for this plot is *amplitudes_quantum_region_Gamma_00000.vtr* in the single band calculation.

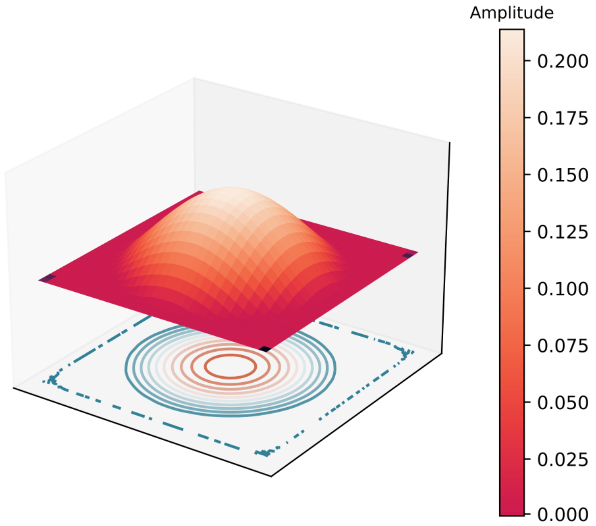


Figure 4.12.1.61: Wave function of the ground state. $(n, l) = (1, 0)$

Wave functions of the energy eigenstates calculated by the single-band model.

Last update: nnnn/nn/nn

Intersubband absorption of a GaAs cylindrical quantum wire

Section author: Naoki Mitsui

This tutorial calculates the optical absorption spectrum of a GaAs cylindrical quantum wire with infinite barriers. We will see which output file we should refer to in order to understand the absorption spectrum.

Also, the formula used for calculation of the absorption spectra is presented. For the detailed scheme of the calculation of the optical matrix elements or absorption spectrum, please see our 1D optics tutorial: *Optical absorption for interband and intersubband transitions*

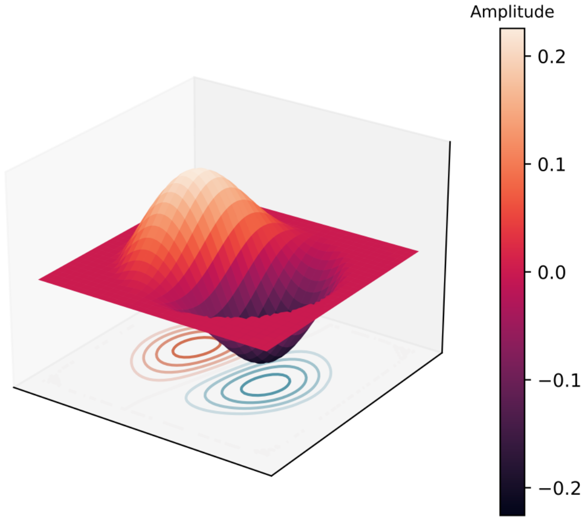


Figure 4.12.1.62: Wave function of the first excited state. $(n, l) = (1, \pm 1)$

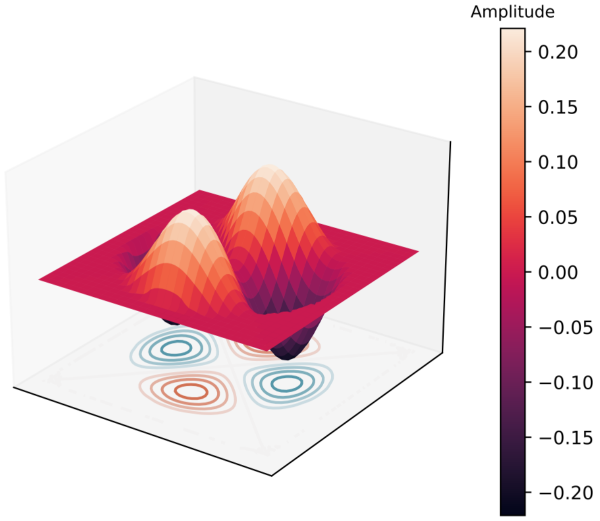


Figure 4.12.1.63: Wave function of the second excited state. $(n, l) = (1, \pm 2)$

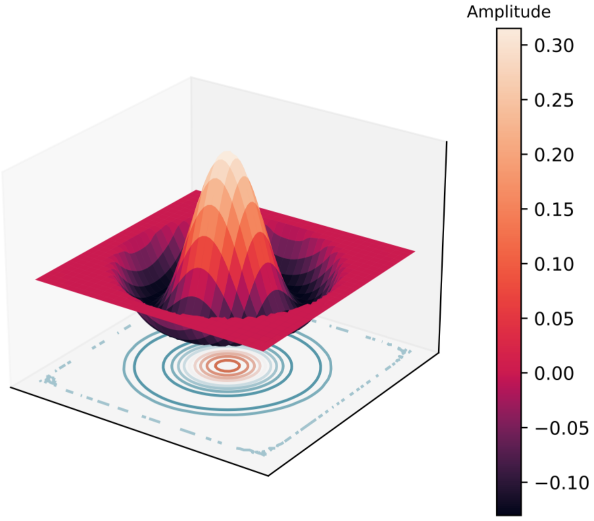


Figure 4.12.1.64: Wave function of the third excited state. $(n, l) = (2, 0)$

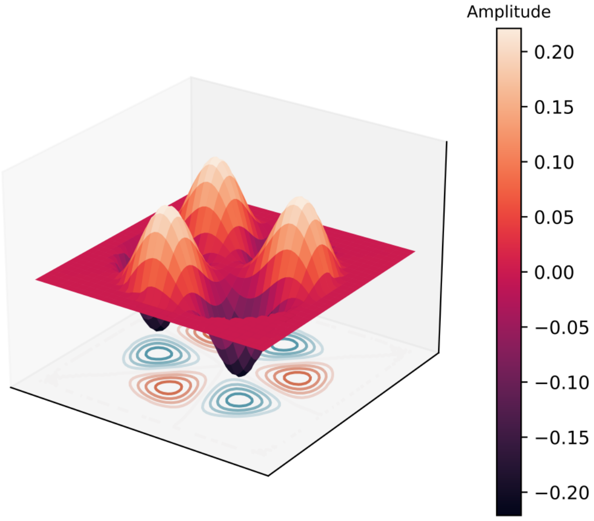


Figure 4.12.1.65: Wave function of the fourth excited state. $(n, l) = (1, \pm 3)$

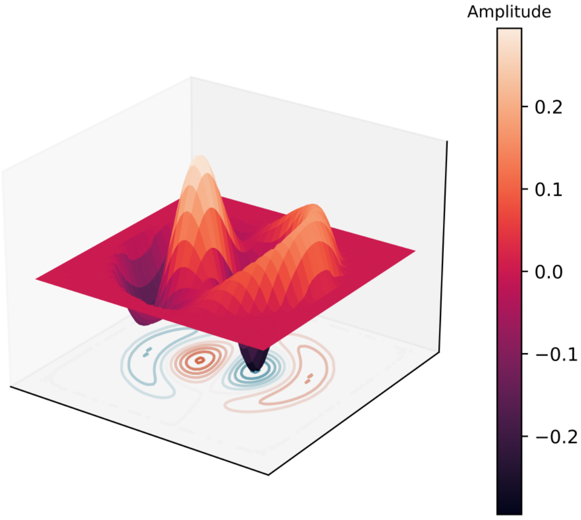


Figure 4.12.1.66: Wave function of the fifth excited state. $(n, l) = (2, \pm 1)$

Input file:

- *2Dcircular_infinite_wire_GaAs_intra_nnp.in*

Structure

The above figures show the Gamma band edge of the circular GaAs region and the barrier region. We model the infinite barrier by assigning 100 eV for the band edge of AlAs barrier region from `database{ }` section. Please see the input file for the details.

The parameters used in this simulation are as follows.

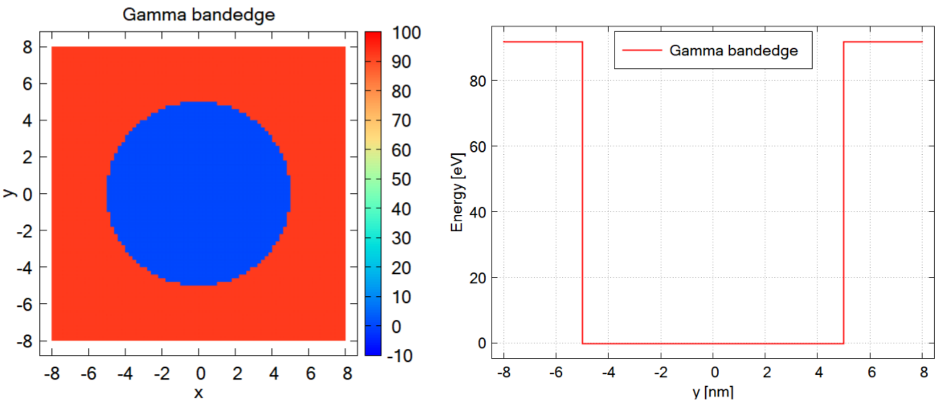


Figure 4.12.1.67: Left: Conduction band edge for a cylindrical quantum wire. Right: Slice of the band edge along $x = 0$.

Property	Symbol	Value [unit]
quantum wire radius	R	5 [nm]
barrier height	E_b	92 [eV]
effective electron mass	m_e	0.0665
refractive index	n_r	3.3
doping concentation (n-type)	N_D	$5 \cdot 10^{18}$ [cm^{-3}]
linewidth (FWHM)	Γ	0.01 [eV]
temperature	T	300 [K]

Scheme

The `run{ }` section is specified as follows:

```
run{
    poisson{ }
    quantum{ }
    optics{ }
}
```

Then the simulation follows these steps:

1. Poisson equation is solved with the setting specified in the *poisson{ }* section.
2. “Schrödinger” equation is solved with the setting specified in the *quantum{ }* section.
3. “Schrödinger” equation is solved again with the setting specified in the *optics{ }* section and optical properties are calculated.

Note

- If `quantum_poisson{ }` is specified instead of `quantum{ }`, Poisson and Schrödinger equations are solved self-consistently.
- `optics{ }` requires that kp8 model is used in the quantum region specified in `quantum{ }`.

- In this tutorial the kp parameters are adjusted so that the conduction and valence bands are decoupled from each other. Thus the single-band Schrödinger equations are solved effectively by the kp solver.

The optical absorption accompanied by the excitation of charge carriers (state $n \to m$) in a condensed matter is calculated on the basis of Fermi's golden rule [*ChuangOpto1995*] in the dimension of (length)$^{-1}$:

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega}\frac{1}{V}\sum_{n>m}\sum_{\mathrm{k}_z}|\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathrm{k}_z)|^2(f_m(\mathrm{k}_z)-f_n(\mathrm{k}_z))\mathcal{L}(E_n(\mathrm{k}_z)-E_m(\mathrm{k}_z)-\hbar\omega), \tag{4.12.1.7}$$

where

- k_z is the Bloch wave vector along translation-invariant directions. In 2D simulation this is 1D vector.
- $E_n(\mathrm{k}_z)$ is the energy of eigenstate n. The first sum runs over the pair of states where $E_n(\mathrm{k}_z) > E_m(\mathrm{k}_z)$.
- $f_n(\mathrm{k}_z)$ is the occupation of eigestate n.
- $\vec{\epsilon}$ is the optical polarization vector defined in *optics{ quantum_spectra{ polarization{ } } }*.
- $\vec{\pi} = \vec{p} + \frac{1}{4m_0c^2}(\sigma\times\nabla V)$ where $\vec{p}$ is the canonical momentum operator and $\frac{1}{4m_0c^2}(\sigma\times\nabla V)$ is the contribution of spin-orbit interaction.
- $\vec{\pi}_{nm}(\mathrm{k}_z) = \langle n|\vec{\pi}|m\rangle$.
- we call $\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathrm{k}_z)$ as the optical matrix elements.
- $\mathcal{L}(E_n(\mathbf{k}_z) - E_m(\mathbf{k}_z) - \hbar\omega)$ **is the energy broadening function.**
  - When `energy_broadening_lorentzian` is specified in *optics{ quantum_spectra{ energy_broadening_lorentzian } }*,

    $\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\pi}\frac{\Gamma/2}{(E_n-E_m-\hbar\omega)+(\Gamma/2)^2}$

    where Γ is the FWHM defined by `energy_broadening_lorentzian`.
  - When `energy_broadening_gaussian` is specified in *optics{ quantum_spectra{ energy_broadening_gaussian } }*,

    $\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\sqrt{2\pi}\sigma}\exp\Big\{\Big(-\frac{(E_n-E_m-\hbar\omega)^2}{2\sigma^2}\Big)\Big\}$

    where `energy_broadening_lorentzian` defines the FWMH $\Gamma = 2\sqrt{\ln 2}\cdot\sigma$
  - When neither `energy_broadening_lorentzian` nor `energy_broadening_gaussian` is specified in *optics{ quantum_spectra{ } }*, $\mathcal{L}$ is replace by the delta function $\delta(E_n - E_m - \hbar\omega)$.
  - It is also possible to include both Lorentzian and Gaussian broadening (Voigt profile).

The detailed calculation scheme of the optical matrix elements $\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathrm{k}_z)$ and the absorption spectrum α is described in *Optical absorption for interband and intersubband transitions*.

Results

Absorption

Figure 4.12.1.68 shows the calculated $\alpha(\vec{\epsilon}, E)$ specified in *\Optics\absorption_~.dat* for each polarization x, y, and z. The absorptions for x- and y-polarization, which are identical due to the rotational symmetry in x-y plane, have two peaks at around 0.2 eV (P1) and 0.95 eV (P2). $\alpha(\vec{\epsilon}, E) = 0$ for z-polarization, which is characteristic for intersubband transtion. These results can be understood from the output data explained below.

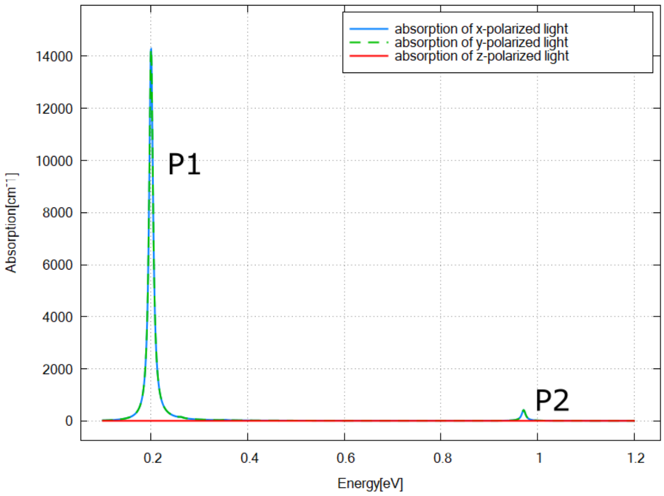


Figure 4.12.1.68: Calculated absorption spectrum $\alpha(\vec{\epsilon}, E)$ for $\vec{\epsilon} = \hat{x}, \hat{y}, \hat{z}$.

Note

$\alpha(\vec{\epsilon}, E)$ for z-polarization is generally non-zero in the calculation through k.p model. This is because the eigenstates above the conduction band edge can have the component of valence band Bloch functions and vice versa (band-mixing).

$\alpha(\hat{z}, E) = 0$ in Figure 4.12.1.68 is reasonable since the single-band model is emulated in this tutorial.

Eigenvalues, transition energies, and occupations

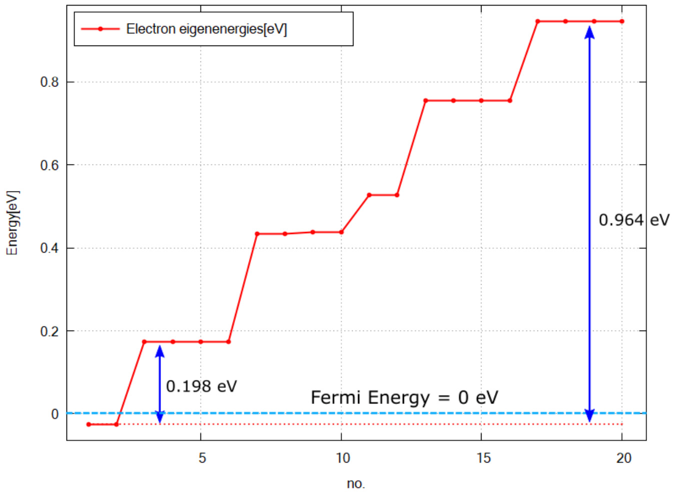


Figure 4.12.1.69: Calculated energy spectrum and Fermi energy (=0 eV).

Figure 4.12.1.69 shows the calculated energy eigenvalues at $k_z = 0$ specified in *Quantum**energy_spectrum_~.dat*.

Please note that the output in *Quantum*\ counts the eigenstates with different spins individually when k.p model is used, while they are counted jointly in *Optics*\.

The only states below the Fermi energy are the ground states (no. 1 and 2). Comparing the excitation energy of other upper states to $k_B T \simeq 0.026$ eV at $T = 300$ K, we can expect the occupation probability of each excited state is almost 0 and the optical transition will occur only from the ground states in this case.

We can see the peak energy of P1 in Figure 4.12.1.68 corresponds to the transition energy from the ground states (no. 1 and 2) to the 1st excited states (no. 3,4,5, and 6). Also the peak energy of P2 corresponds to the transition energy from the ground states to 5th excited states (no. 17,18,19, and 20).

The occupation probabilities for each state can be checked from *\Optics\occupation_disp_~.dat*as a function of the 1D Bloch wave vector k_z:

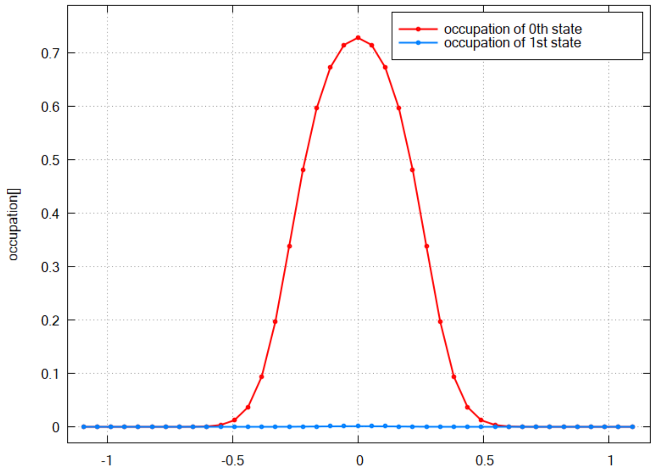


Figure 4.12.1.70: Calculated occupation probabilities for the ground state and 1st excited state as a function of k_z.

As we expected above, the ground state is well occupied for small k_z and the occupation of the 1st excited state is alomost 0.

Note

The eigenstates with different spins are counted individually in *Quantum*\ when k.p model is used, while they are counted jointly in *Optics*\.

For example, the two ground states counted as no.1 and 2 in Figure 4.12.1.69 due to spin are put together as one eigenstate in *Optics*\. Thus *\Optics\occupation_disp_~_kp8_1.dat* shows the occupation of the ground state and *\Optics\occupation_disp_~_kp8_2.dat* and *\Optics\occupation_disp_~_kp8_3.dat* show the 1st excited state in this case.

From the above data of eigenvalues and occupations, we could see which pair of states contributes to each peak in the absorption spectrum Figure 4.12.1.68. In order to understand the magnitude of the peaks and why some pairs of states do not appear as peaks, we will see the output data for $|\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathrm{k}_z)|^2$ next.

Transition intensity (Momentum matrix element)

One of the key element for the calculation of absorption spectra is the transition intensity

$$T_{nm}(\vec{\epsilon}, \mathrm{k}_z) = \frac{2}{m_0} |\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathrm{k}_z)|^2 \tag{4.12.1.8}$$

which has the dimension of energy [eV].

The intensity at $\mathrm{k}_z = 0$ $\left(T_{nm}(\vec{\epsilon}, \mathrm{k}_z = 0)\right)$ for each pair of states (n, m) is specified in *Optics\transitions_~.txt*. These intensities whose "From" states are the ground state are shown here (x-polarization). We can also check the transition energy of each pair of states.

```
Energy[eV]          From            To              Intensity_k0[eV]                1/
↪Radiative_Rate[s]
0.19824             1                   2           2.77912                         3.
↪80277e-08
0.19824             1                   3           2.9137                          3.
↪62712e-08
0.775938            1                   7           8.37435e-06                     0.
↪00322418
0.775938            1                   8           6.88813e-06                     0.
↪00391985
0.964304            1                   9           0.368533                        5.
↪89532e-08
0.964304            1                   10          0.427067                        5.
↪0873e-08
```

We can explain the large P1 (~0.198 eV) and small P2 (~0.964 eV) by the large and small transition intensities in these output data. Also we can see the transtions from 1 to 4,5,6,7 are almost zero and these pairs of states do not contribute to the absorption (transitions from 1 to 4,5 are omitted here since `Intensity_k0` are too small).

There is also the output files that specify the k-dispersion of the transition intensities for each light polarization in *Optics\transition_disp_~.dat*.

Eigenstates

The probability distribution of eigenfunctions $|\psi(\mathbf{r})|^2$ is output in *Quantum\probabilities_~.vtr*. The amplitude of the envelope function on each Bloch function $|S\rangle, |X\rangle, |Y\rangle, |Z\rangle$ can be found in *Quantum\amplitudes_~_SXYZ.vtr*.

The analytcal expression of the eigenfunctions for the cylindrical quantum wire is shown as eq. (4.7.2.1) in this tutorial: *Electron wave functions in a cylindrical well (2D Quantum Corral)*. According to this analytical solution, the eigenfunction has 2 quantum numbers: n for radial direction and l for circumferential direction.

Here the amplitudes of eigenfunctions calculated by single-band model are shown. We can see the optical transition from ground state ($n = 1, l = 0$) occurs only to the states with $l = \pm 1$.

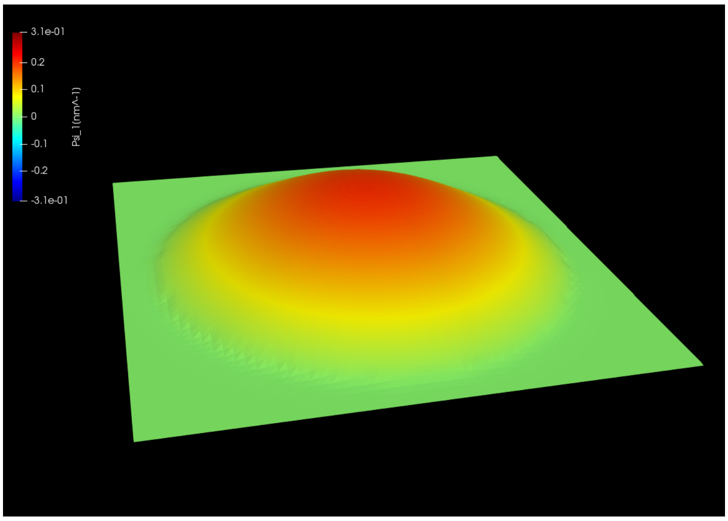


Figure 4.12.1.71: Wave function of the ground state. $(n, l) = (1, 0)$

Wave funstions of the energy eigenstates calculated by the single-band model.

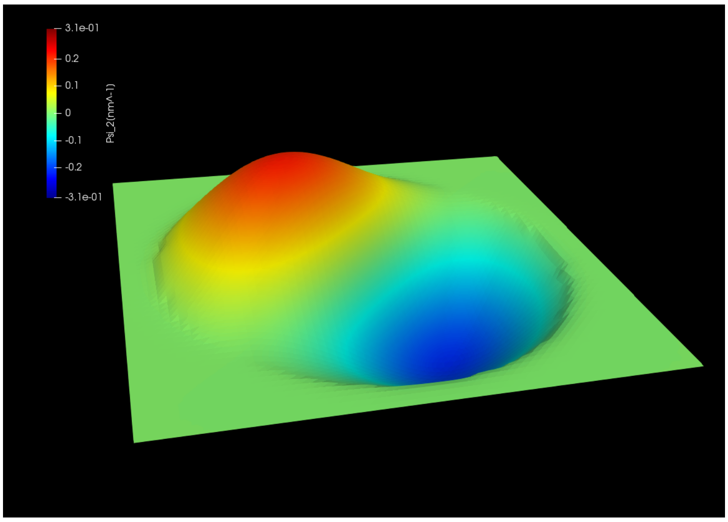


Figure 4.12.1.72: Wave function of the 1st excited state. $(n, l) = (1, \pm 1)$

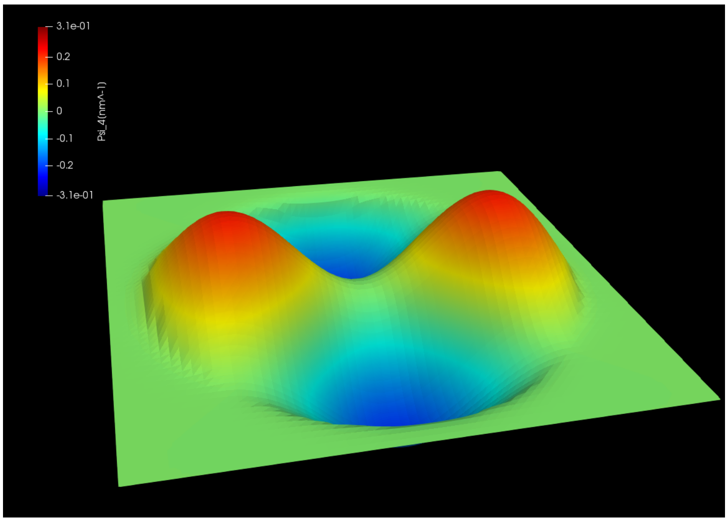


Figure 4.12.1.73: Wave function of the 2nd excited state. $(n, l) = (1, \pm 2)$

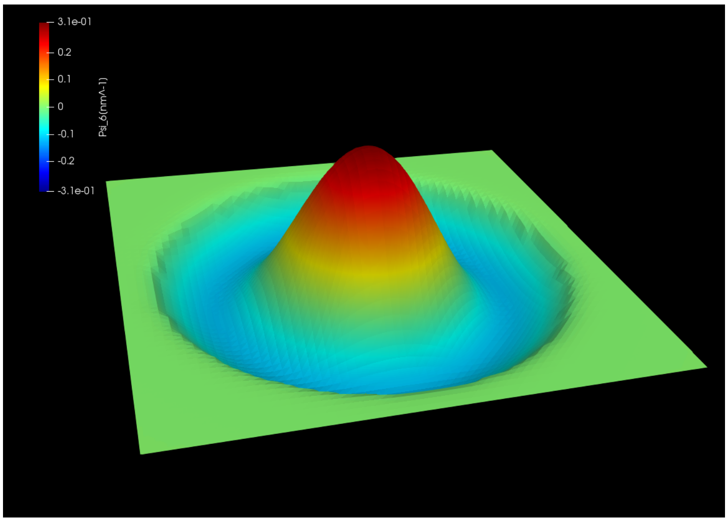


Figure 4.12.1.74: Wave function of the 3rd excited state. $(n, l) = (2, 0)$

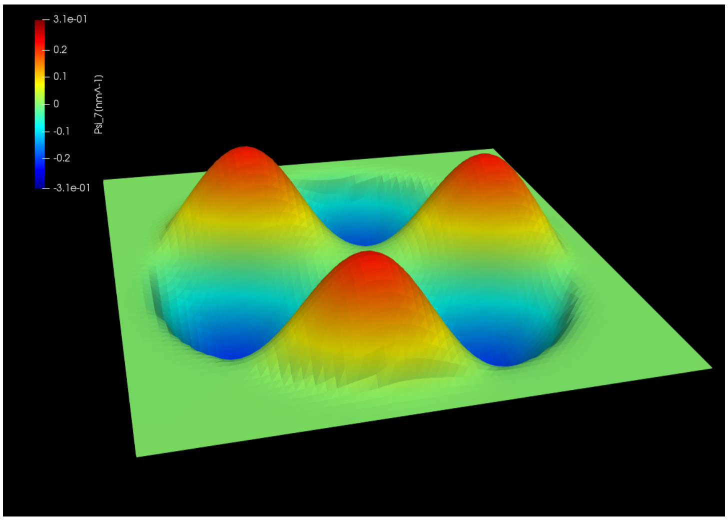


Figure 4.12.1.75: Wave function of the 4th excited state. $(n, l) = (1, \pm 3)$

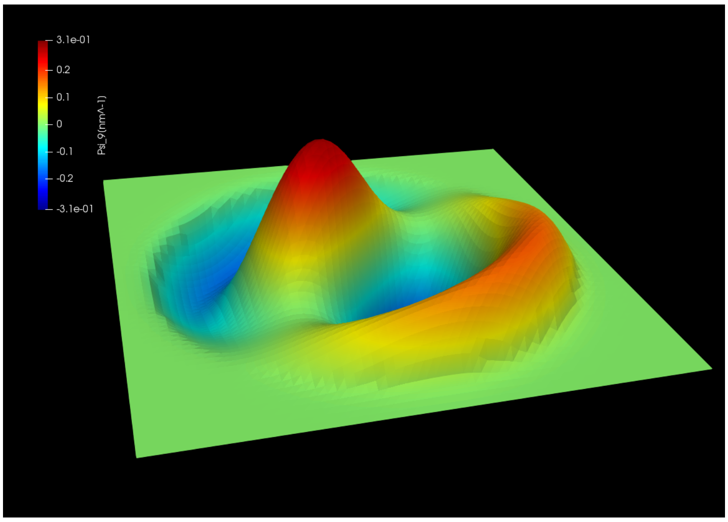


Figure 4.12.1.76: Wave function of the 5th excited state. $(n, l) = (2, \pm 1)$

Last update: nnnn/nn/nn

Absorption of a GaAs spherical quantum dot

This tutorial calculates the optical absorption spectrum of a GaAs spherical quantum dot with infinite barriers. We will see which output file we should refer to in order to understand the absorption spectrum.

Also, the formula used for the absorption calculation is presented. For the detailed scheme of the calculation of the optical matrix elements and absorption spectrum, please see our 1D optics tutorial: *Optical absorption for interband and intersubband transitions*

Input file:

- *3Dspherical_infinite_dot_GaAs_intra_nnp.in*
- *3Dspherical_infinite_dot_GaAs_inter_nnp.in*

Structure

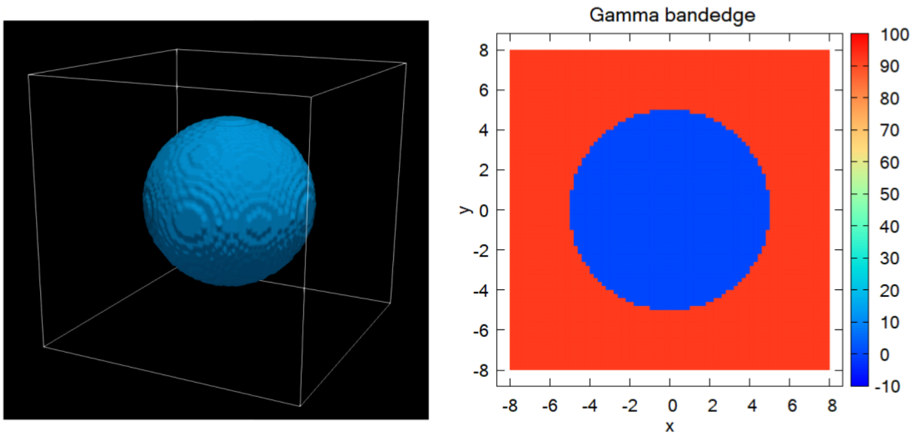


Figure 4.12.1.77: Left: GaAs region as a spherical quantum dot. Right: Slice of the Gamma band edge along $z = 0$.

The above figures show the Gamma band edge of the spherical GaAs region and the barrier region. We model the infinite barrier by assigning 100 eV for the band edge of AlAs barrier region from `database{ }` section. Please see the input file for the details.

The parameters used in this simulation are as follows.

Property	Symbol	Value [unit]
quantum dot radius	R	5 [nm]
barrier height	E_b	92 [eV]
effective electron mass	m_e	0.0665
refractive index	n_r	3.3
doping concentation (n-type)	N_D	$8 \cdot 10^{18}$ [cm^{-3}]
linewidth (FWHM)	Γ	0.01 [eV]
temperature	T	300 [K]

Scheme

The `run{ }` section is specified as follows:

```
run{
    poisson{ }
    quantum{ }
    quantum optics{ }
}
```

Then the simulation follows these steps:

1. Poisson equation is solved with the setting specified in the *poisson{ }* section.
2. "Schrödinger" equation is solved with the setting specified in the *quantum{ }* section.
3. "Schrödinger" equation is solved again with the setting specified in the *optics{ }* section and optical properties are calculated.

Note

- If `quantum_poisson{ }` is specified instead of `quantum{ }`, Poisson and Schrödinger equations are solved self-consistently.
- `optics{ }` requires that kp8 model is used in the quantum region specified in `quantum{ }`.
- In this tutorial the kp parameters are adjusted so that the conduction and valence bands are decoupled from each other. Thus the single-band Schrödinger equations are solved effectively by the kp solver.

The optical absorption accompanied by the excitation of charge carriers (state $n \to m$) in a condensed matter is calculated on the basis of Fermi's golden rule [*ChuangOpto1995*] in the dimension of (length)$^{-1}$:

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega} \frac{1}{V} \sum_{n>m} |\vec{\epsilon} \cdot \vec{\pi}_{nm}|^2 (f_m - f_n) \mathcal{L}(E_n - E_m - \hbar\omega), \tag{4.12.1.9}$$

where

- E_n is the energy of eigenstate n. The first sum runs over the pair of states where $E_n > E_m$.
- f_n is the occupation of eigestate n.
- $\vec{\epsilon}$ is the optical polarization vector defined in *optics{ quantum_spectra{ polarization{ } } }*.
- $\vec{\pi} = \vec{p} + \frac{1}{4m_0c^2}(\sigma \times \nabla V)$ where $\vec{p}$ is the canonical momentum operator and $\frac{1}{4m_0c^2}(\sigma \times \nabla V)$ is the contribution of spin-orbit interaction.
- $\vec{\pi}_{nm} = \langle n|\vec{\pi}|m\rangle$.

- we call $\vec{\epsilon}\cdot\vec{\pi}_{nm}$ as the optical matrix elements.
- $\mathcal{L}(E_n - E_m - \hbar\omega)$ **is the energy broadening function:**
  - When `energy_broadening_lorentzian` is specified in *optics{ quantum_spectra{ energy_broadening_lorentzian } }*,

    $$\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\pi}\frac{\Gamma/2}{(E_n-E_m-\hbar\omega)+(\Gamma/2)^2}$$

    where Γ is the FWHM defined by `energy_broadening_lorentzian`.
  - When `energy_broadening_gaussian` is specified in *optics{ quantum_spectra{ energy_broadening_gaussian } }*,

    $$\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\sqrt{2\pi}\sigma}\exp\Big\{\Big(-\frac{(E_n-E_m-\hbar\omega)^2}{2\sigma^2}\Big)\Big\}$$

    where `energy_broadening_lorentzian` defines the FWMH $\Gamma = 2\sqrt{\ln 2}\cdot\sigma$
  - When neither `energy_broadening_lorentzian` nor `energy_broadening_gaussian` is specified in *optics{ quantum_spectra{ } }*, $\mathcal{L}$ is replace by the delta function $\delta(E_n - E_m - \hbar\omega)$.
  - It is also possible to include both Lorentzian and Gaussian broadening (Voigt profile).

The detailed calculation scheme of the optical matrix elements $\vec{\epsilon}\cdot\vec{\pi}_{nm}$ is described in *Optical absorption for interband and intersubband transitions*. In 3D simulation we do not have the k-summation like 1D and 2D cases.

Results

Absorption

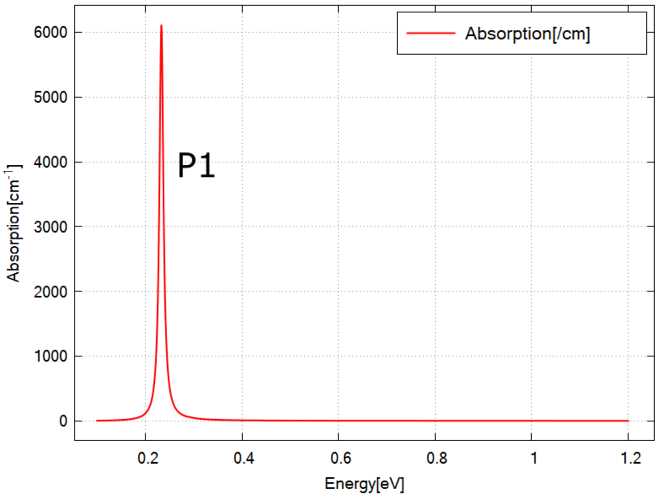


Figure 4.12.1.78: Calculated absorption spectrum $\alpha(\vec{\epsilon}, E)$ for $\vec{\epsilon} = \hat{x}$.

Figure 4.12.1.78 shows the calculated $\alpha(\vec{\epsilon}, E)$ specified in *\Optics\absorption_~.dat* for x-polarization. The absorptions for y- and z-polarization are identical to this graph due to the rotational symmetry. We have one peak at around 0.23 eV (P1). These results can be understood from the output data explained below.

Note

When we use the realistic k.p paramters, $\alpha(\vec{\epsilon}, E)$ for each polarization would no more be identical in general. This is because the eigenstates above the conduction band edge can have the component of valence band Bloch functions (band-mixing).

They are identical in this tutorial since the single-band model is emulated.

Eigenvalues, transition energies, and occupations

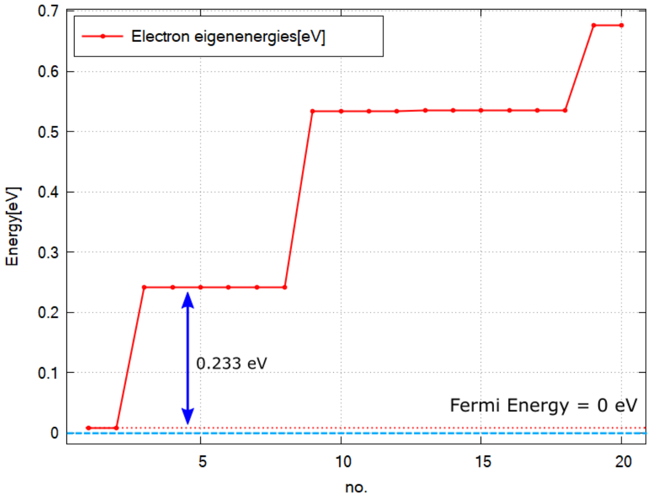


Figure 4.12.1.79: Calculated energy spectrum and Fermi energy (=0 eV).

Figure 4.12.1.79 shows the calculated energy eigenvalues specified in *\Quantum\energy_spectrum_~.dat*.

Please note that the output in *Quantum* counts the eigenstates with different spins individually when k.p model is used, while they are counted jointly in *Optics*.

Comparing the excitation energy of other upper states to $k_B T \simeq 0.026$ eV at $T = 300$ K, we can expect the occupation probability of each excited state is almost 0 and only the ground states have the non-zero occupation. Thus the optical transition will occur only from the ground states in this case.

We can see the peak energy of P1 in Figure 4.12.1.78 corresponds to the transition energy from the ground states (no. 1 and 2) to the 1st excited states (no. 3,4,5,6,7 and 8).

Note

The eigenstates with different spins are counted individually in *Quantum* when k.p model is used, while they are counted jointly in *Optics*.

For example, the two ground states counted as no.1 and 2 in Figure 4.12.1.79 due to spin are put together as one eigenstate in *Optics*.

From the above data of eigenvalues, we could see which pair of states contributes to the peak in the absorption spectrum Figure 4.12.1.78. In order to understand why some pairs of states do not appear as peaks, we will see the output data for $|\vec{\epsilon} \cdot \vec{\pi}_{nm}|^2$ next.

Transition intensity (Momentum matrix element)

One of the key element for the calculation of optical absorption is the transition intensity

$$T_{nm}(\vec{\epsilon}) = \frac{2}{m_0}|\vec{\epsilon} \cdot \vec{\pi}_{nm}|^2 \tag{4.12.1.10}$$

which has the dimension of energy [eV].

The intensity $\left(T_{nm}(\vec{\epsilon})\right)$ for each pair of states (n, m) is specified in *Optics\transitions_~.txt*. These intensities whose “From” states are the ground state are shown here for x-polarization. We can also check the transition energy of each pair of states.

```
Energy[eV]          From            To              Intensity_k0[eV]                1/
↪Radiative_Rate[s]
0.233098            1               2               2.02882         4.43013e-08
0.233098            1               3               2.42777         3.70214e-08
0.233098            1               4               2.30413         3.90079e-08
```

The transtions from 1 to 5~10 are zero and these pairs of states do not contribute to the absorption (They are omitted here since `Intensity_k0` are too small).

Eigenstates

The probability distribution of eigenfunctions $|\psi(\mathbf{r})|^2$ is output in *Quantum\probabilities_~.vtr*. The amplitude of the envelope function on each Bloch function $|S\rangle, |X\rangle, |Y\rangle, |Z\rangle$ can be found in *Quantum\amplitudes_~_SXYZ.vtr*.

Here the probability distribution of eigenfunctions calculated by single-band model are shown.

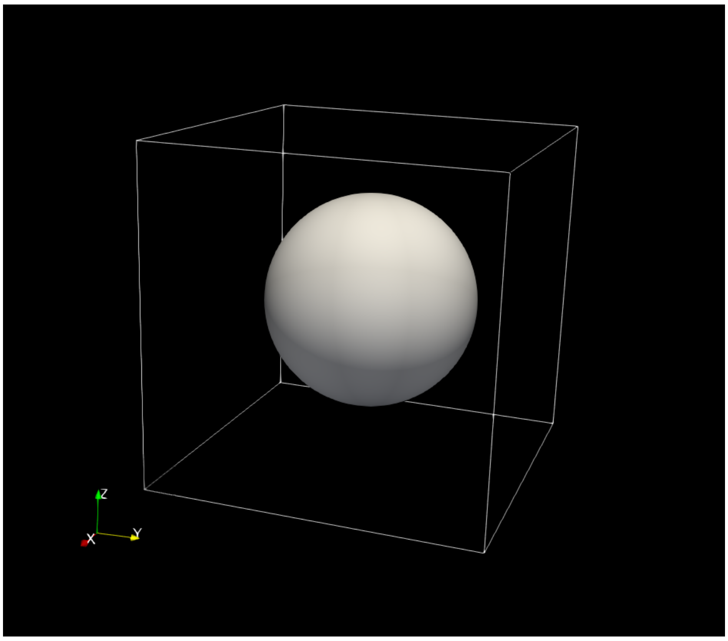


Figure 4.12.1.80: |wave function|^2 of the ground state. (s orbital, not degenerated.)

|wave function|^2 of the energy eigenstates calculated by the single-band model. The contours at the value of $|\psi(\mathbf{r})|^2 = 0.001$ are shown.

Last update: nnnn/nn/nn

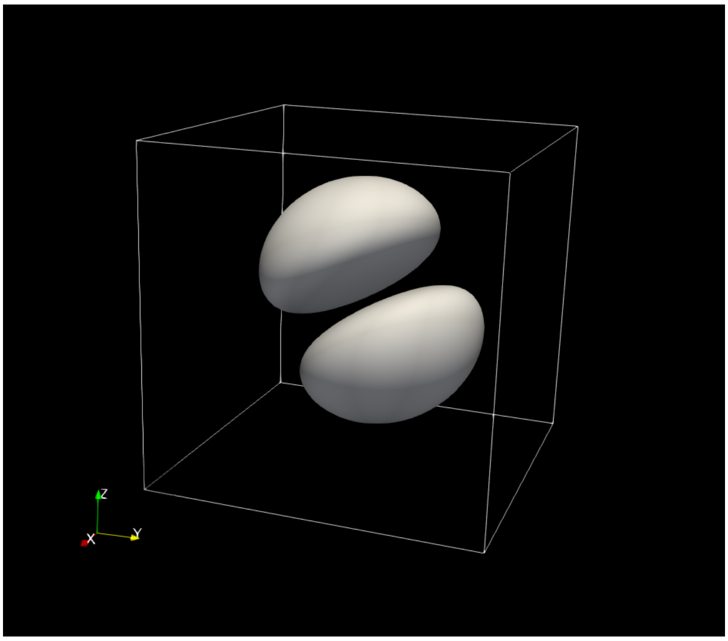


Figure 4.12.1.81: |wave function|^2 of the 1st excited state. (3 times degenerated, p orbital)

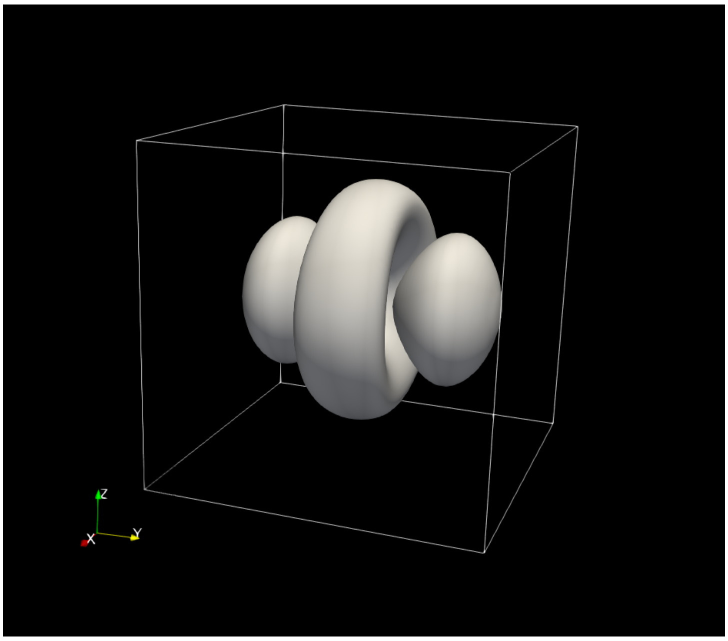


Figure 4.12.1.82: |wave function|^2 of the 2nd excited state. (5 times degenerated, d orbital)

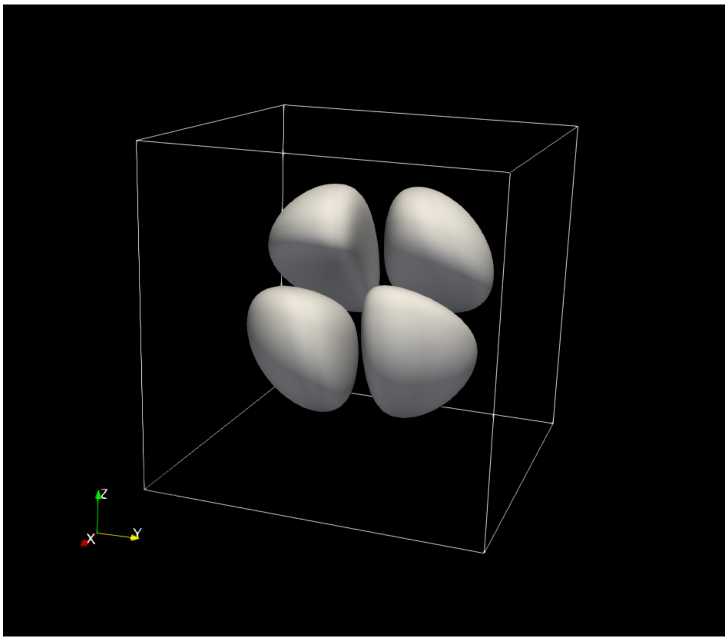


Figure 4.12.1.83: |wave function|^2 of the 2nd excited state. (d orbital)

Optics: Optical gain of InGaAs quantum wells with different strain

Input Files:

- *1D_gain_strained_qw.in*

Scope:

Comparison of the optical gain calculated for differently strained InGaAsP-InGaAs quantum wells using 8-band k.p model with *[ChuangOpto1995]*, Sec. 10.4 .

Most relevant keywords:

- `quantum{ region{ kp_8band{} } }`
- `optics{ quantum_spectra{} }`

Output files:

- *\bias_00000\Optics\absorption_quantum_region_TEy_eV.dat*
- *\bias_00000\Optics\absorption_quantum_region_TMx_eV.dat*
- *\bias_00000\Quantum\Dispersions\dispersion_quantum_region_kp8_11_00_10.dat*
- *\bias_00000\bandedges.dat*

Introduction

We consider a 1D single quantum well system consisting of $In_{1-x}Ga_xAs$ sandwiched between $In_{0.71}Ga_{0.29}As_{0.61}P_{0.39}$ barrier layers. Simulations are performed for three different mole fractions x resulting in three different strain conditions:

- $x = 0.41$ (QW region is compressively strained)
- $x = 0.47$ (QW region is unstrained)
- $x = 0.53$ (QW region is tensely strained).

The parameters for the layer thicknesses, alloy composition and quasi-Fermi levels are taken as follows:

- The well widths L_w are chosen as 4.5 nm, 6.0 nm, and 11.5 nm for each x value, respectively, so that the energy difference between the highest valence band eigenstate and lowest conduction band eigenstate would be around 0.8 eV (~1500 nm). The length of the complete simulation region is the same for all three cases, namely L_t = 20 nm.

- The alloy composition of $InGaAsP$ barrier region is determined, so that its lattice constant matches to InP substrate and the band gap is 0.95 eV (λ_g = 1300 nm). The same barrier composition is used for all x.
- The electron and hole quasi-Fermi levels were determined for each x, so that the carrier densities of electrons and holes integrated over the QW width both equal 3 $\cdot 10^{12}$ cm^{-2}.

Computation of the optical absorption spectra within the Fermi's golden rule and 8-band k.p model is triggered in the `optics{ }` group. Please refer to *our tutorial on absorption* for the details about the calculation scheme of the absorption spectra.

Results

We show for each of the three cases the calculated band edges, subband dispersions of the highest electron and hole states, and the optical gain coefficients of TE and TM mode.

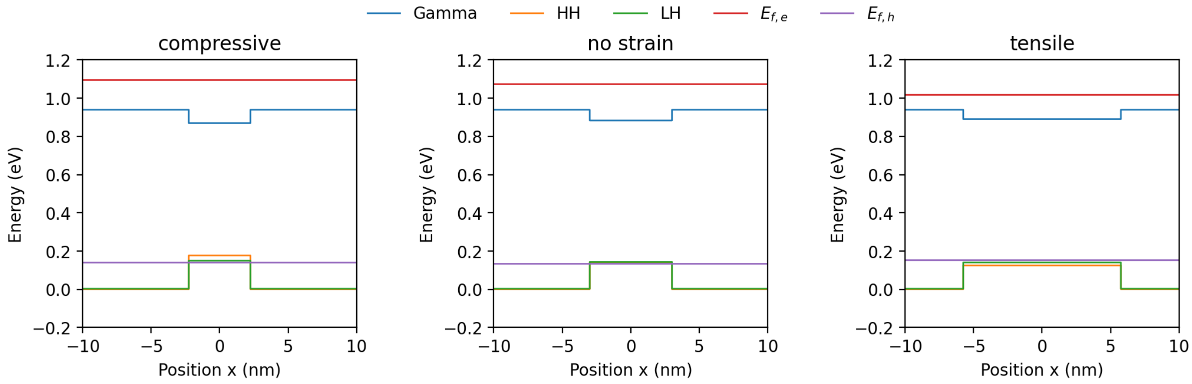


Figure 4.12.1.84: The band edges and Fermi levels of compressively strained QW (L_w = 4.5 nm, x = 0.41) (**left**), unstrained QW (L_w = 6.0 nm, x = 0.47) (**center**) and tensely strained QW (L_w = 11.5 nm, x = 0.50) (**right**). The band profile is shifted so that the valence band edge of the barrier is at 0 eV.

The band profiles for all three cases are depicted in Figure 4.12.1.84. The HH is the highest valence band in the compressive case, HH and LH are degenerated in the unstrained case, and LH is the highest valence band in the tensile case due to the different band-shift of HH and LH. Figure 10.30 in *[ChuangOpto1995]* shows the same qualitative effect of strain on the band edge profile.

Energy dispersions for all three cases are shown in Figure 4.12.1.85. The corresponding output file is *Quantum/dispersion_~.dat*, which is calculated in `quantum{ }` group. We observe an upward shift of the valance bands going from compressive to tensile strain, which is in agreement with figure 10.30 in *[ChuangOpto1995]*.

Figure 4.12.1.86 shows optical gain computed for the differently strained QWs. The gain for TE polarization is dominant in the compressive and unstrained quantum well as related to transitions involving HH, and TM gain is dominant in the tensely strained quantum well due to the lowest energy transitions involving LH. Comparing the gain spectra with the results presented in *[ChuangOpto1995]*, we observe that for all three cases the shapes of the TE spectra relative to the TM spectra are correctly reproduced. However, there are some deviations in the amplitudes of the spectra. In the cases of the compressive strain and no strain, the computed gain spectra are about 100 cm^{-2} higher than the ones presented in *[ChuangOpto1995]*. Conversely, the spectra computed for the tensely strained quantum well are about 100 cm^{-2} smaller than those in the reference.

Discussion

Most possible reasons which account for the deviations between our gain spectra and these shown in *[ChuangOpto1995]* may be differences in:

- the model applied to compute the spectra,
- the number of electron and hole states included in the model,

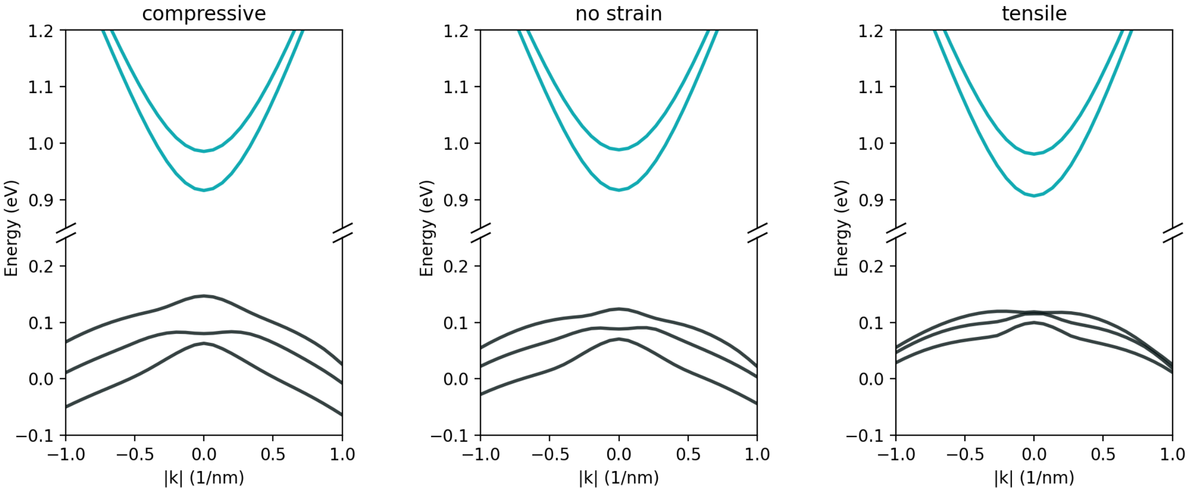


Figure 4.12.1.85: Calculated subband dispersions for the comressively strained QW (**left**), unstrained QW (**center**) and tensely strained QW (**right**). The ground and 1st excited states for electron (cyan), as well as the three highest hole states (black) are shown.

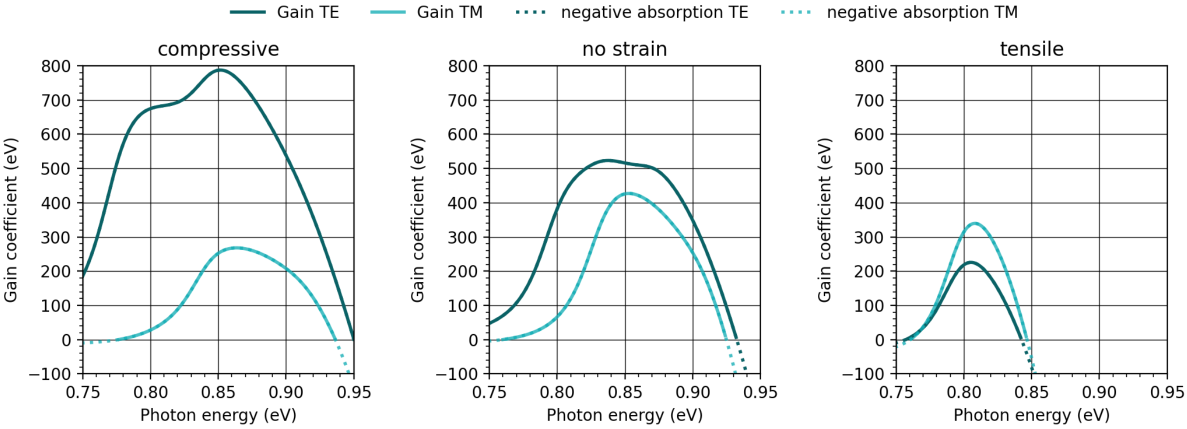


Figure 4.12.1.86: Calculated optical gain of TE and TM optical mode for compressively strained QW (**left**), unstrained QW (**center**) and tensily strained QW (**right**)

- how the surface charge concentration of 3 $\cdot 10^{12}$ cm^{-2} is calculated. In *[ChuangOpto1995]* the surface charge concentration is equal to nL_z, where we assumed an integration of the carrier density over the well width, i.e. $\int n(x)dx$. The surface charge concentration is an important parameter, because it determines the quasi-Fermi levels and therefore the amplitude of the gain spectrum.
- boundary conditions for the wave functions. Here, we used periodic boundary conditions.

Last update: nnnn/nn/nn

— DEV — Optical gain and spontaneous emission rate of strained GaN quantum well

Warning

This tutorial is under development.

In this tutorial, we calculate the optical gain and spontaneous emission rate of strained GaN quantum well using 8-band k.p model implemented in our *optics{ }* section. This tutorial aims to reproduce the results obtained in *[ChuangIEEE1996]*:

- *"Optical gain of strained wurtzite GaN quantum-well lasers" S. L. Chuang, IEEE Journal of Quantum Electronics (1996)*

Related files

- *Chuang_1996_IEEE_GaN_QW_nnp.in*
- *Chuang_1996_IEEE_GaN_QW_postprocess.py* (python script using *nextnanopy*)

Table of contents

The *nextnano++* tool can calculate the spontaneous emission rate and optical gain in 2 different models.

1. "Semiclassical" calculation corresponds to *classical{ }*
2. "Quantum" calculation corresponds to *optics{ }*

For the 1st model, please refer to *InGaAs Multi-quantum well laser diode*. Roughly speaking, this model calculates the carrier densities either quantum mechanically or classically and the emission rate is calculated from them in a phenomenological way (4.4.1.4).

The calculation described here is the 2nd model. This starts from the Fermi's golden rule (time-dependent perturbation theory) and electrons in a condensed matter are treated fully quantum mechanically. This model has the following characteristics:

- able to take into account the band-bending and band-mixing effect by strain
- distinguishes the different polarization
- requires less phenomenological parameter
- require the k.p parameters instead

(For most of the important materials, the parameters are already included in our database file.)

Structure

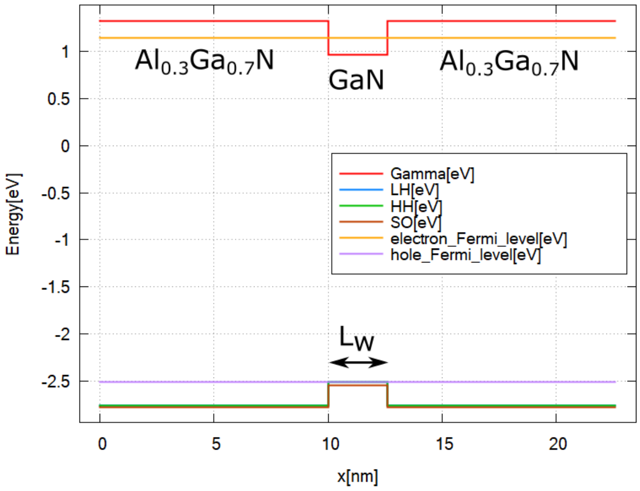


Figure 4.12.1.87: The band edges and Fermi energies for $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 3 \times 10^{19}$ cm^{-3} inside the well region.

The above figures show the Gamma band edge of the $Al_{0.3}Ga_{0.7}N$-GaN quantum well.

Please see the input file for the details.

The parameters used in this simulation are as follows.

Property	Symbol	Value [unit]
quantum well width	L_w	2.6, 5.0 [nm]
doping concentation	N_D	0 [cm^{-3}]
carrier concentration in the well	n	1, 2, 3 × 10^{19} [cm^{-3}]
linewidth (FWHM)	Γ	0.0132 [eV]
temperature	T	300 [K]

Note

The piezo- and pyroelectricity are not yet taken into consideration here for the simplicity.

Results

Spontaneous emission rate

The formula used for the spontaneous emission calculation in optics section is as follows:

$$r^{spon}(\vec{\epsilon},\omega) = \frac{n_r e^2 E}{\pi\hbar^2 c^3 \varepsilon_0 m_0^2}\frac{2}{V}\sum_{n>m}\sum_{\mathbf{k}_\parallel}|\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathbf{k}_\parallel)|^2\mathcal{L}(E_n(\mathbf{k}_\parallel)-E_m(\mathbf{k}_\parallel)-E)f_n(\mathbf{k}_\parallel)(1-f_m(\mathbf{k}_\parallel)), \tag{4.12.1.11}$$

For the detail of the definition of each quantity and calculation scheme, please see our *Optical absorption for interband and intersubband transitions*.

Here we show this $r^{spon}(\vec{\epsilon},\omega)$ calculated for $L_w = 2.6$ [nm], $L_w = 5.0$ [nm] and each polarization. These results well agree with Fig.7 of *[ChuangIEEE1996]*.

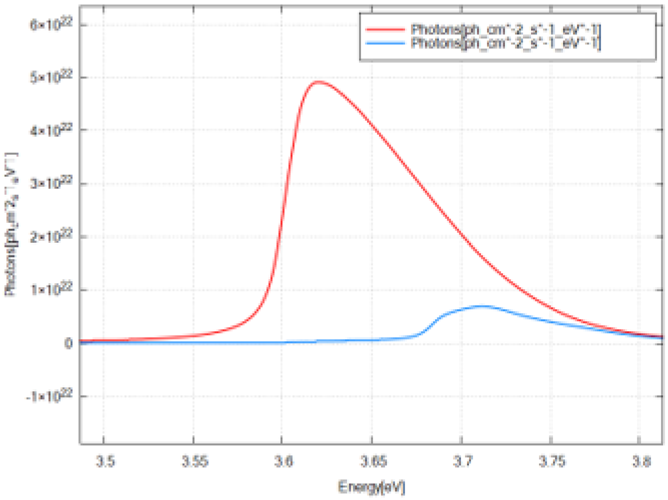


Figure 4.12.1.88: r^{spon} for an $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 2.6$ [nm]

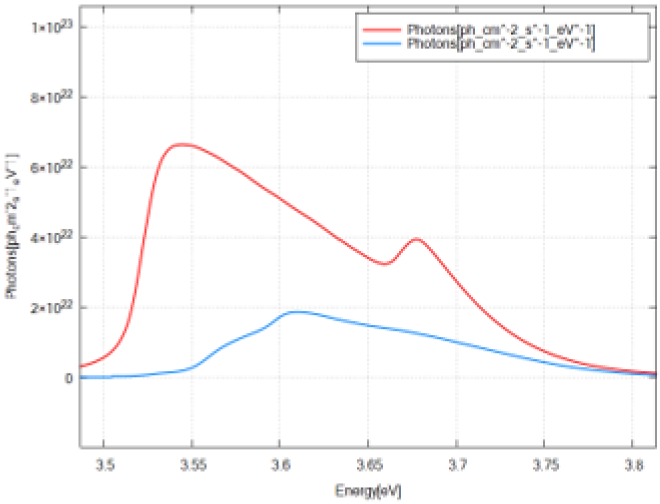


Figure 4.12.1.89: r^{spon} for an $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 5.0$ [nm]

When we do not apply the linewidth broadening, the result shows the exact energy where the emission by each pair of state starts.

Optical Gain

The optics section can calculate the absorption spectra $\alpha(\vec{\epsilon},\omega)$. This can be understood as a negative gain, i.e.

$$\alpha(\vec{\epsilon},\omega) = -g(\vec{\epsilon},\omega) \tag{4.12.1.12}$$

For the details of the calculation scheme of $\alpha(\vec{\epsilon},\omega)$, please see our *Optical absorption for interband and intersubband transitions*.

Here we show this $g(\vec{\epsilon},\omega)$ calculated for $L_w = 2.6$ [nm], $L_w = 5.0$ [nm] and polarization.

These results almost agrees with Fig.8 of *[ChuangIEEE1996]* except for the case when the gain peak is relatively low. This is because the models used here and *[ChuangIEEE1996]* apply the linewidth broadening in different steps.

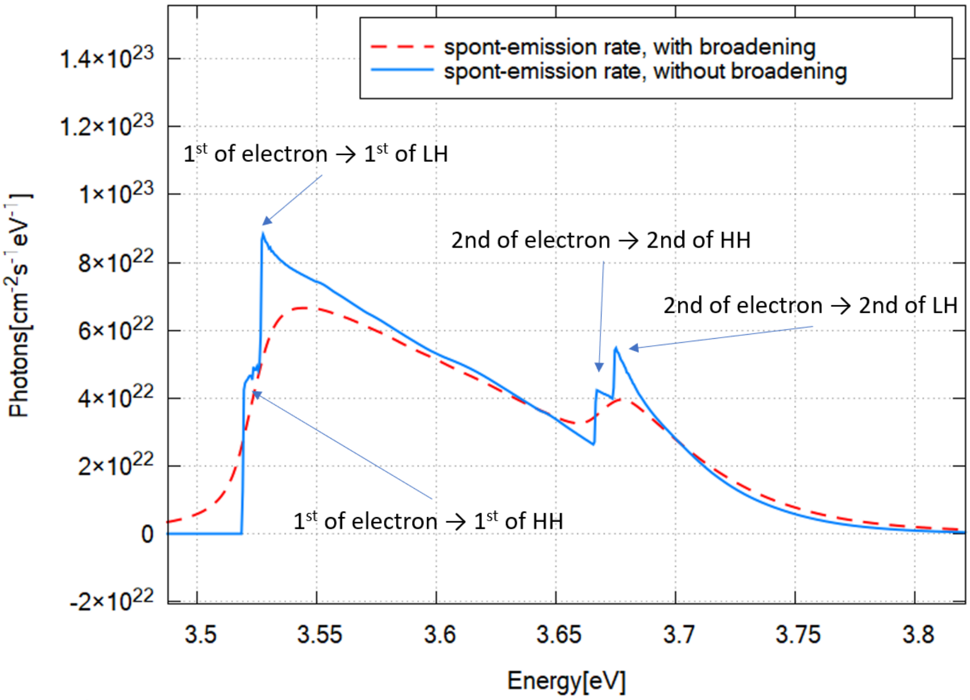


Figure 4.12.1.90: TE emission rate in Figure 4.12.1.89 with (red dashed line) and without (blue line) line width broadening.

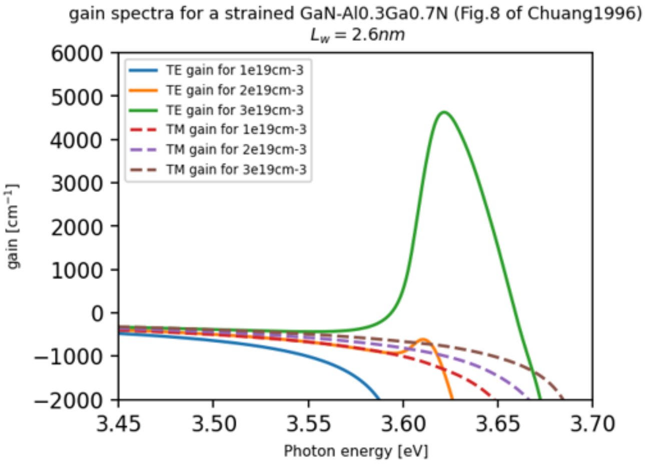


Figure 4.12.1.91: $g(\vec{\epsilon}, \omega)$ for a $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 1, 2, 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 2.6$ [nm]

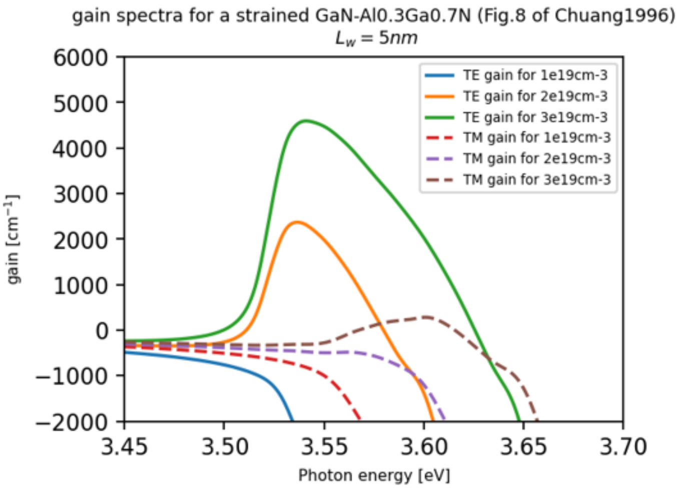


Figure 4.12.1.92: $g(\vec{\epsilon}, \omega)$ for a $Al_{0.3}Ga_{0.7}$N-GaN quantum well with the carrier concentration $n = 1, 2, 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 5.0$ [nm]

Last update: nnnn/nn/nn

4.12.2 Excitons

Exciton absorption in infinite quantum well

Input files:

1D_InterbandExcitonAbsorption_InfiniteWell_GaAs_8kp_nnp.in 1D_InterbandExcitonAbsorption_InfiniteWell_GaAs_effecti

Scope of the tutorial:

In this tutorial, we show how excitonic correction affects the absorption in infinite quantum well.

The most relevant keywords:

- `optics{ quantum_spectra{} }`
- `quantum{ region{ excitons{} } }`

Relevant output files:

bias_xxxxx\bandedges.dat *bias_xxxxx\Optics\absorption_quantum_region_TE_eV.dat* *bias_xxxxx\Quantum\probabilities_shift_quantum_region_kp8_00000.dat*

This tutorial presents calculation of interband absorption spectrum in a quantum well including excitonic effects. The tutorial aims to provide a comprehensive explanation of how excitonic correction significantly influences the optical absorption characteristics in a quantum well.

In this tutorial we calculate the absorption spectrum of a 10 nm GaAs quantum well. The purpose is to calculate the absorption spectrum for a simple model and model that includes excitonic effects on the absorption spectrum.

The tutorial is structured into two parts. The first part involves the computation of valence and conduction states using simple parabolic dispersion models, also known as the “single-band” model. In the second part, the states will be computed using an 8-band kp Hamiltonian.

Theory of optical excitonic correction

An exciton is a bound state of an electron and a hole in a solid material, resulting from the Coulomb attraction between them. The exciton eigenvalue is computed using variational approach with the wave function

$$F(r, x_h, x_e) = f(x_e)g(x_h)\phi(r)$$
$$\phi(r) = \frac{2}{\pi}\frac{1}{\lambda}\exp(-r/\lambda)$$

where $f(x_e), g(x_h)$ – electron and hole wave functions, r – radial variable in plane orthogonal to growth direction, λ – variational parameter.

The exciton correction to absorption consists of 2 terms: exciton peak and Sommerfeld enhancement factor (also known as Coulomb enhancement). The exciton peak is located few meV below the absorption edge of corresponding electron-hole pair (i.e. transition energy is reduced by binding energy of exciton) The intensity of the peak is dependent on the parameter λ.

$$\alpha_{ex} \propto \frac{2}{\pi\lambda^2}V(E_{ij} - E_b, \hbar\omega)$$

where V is Voigt profile, E_{ij} is the transition energy between electron i and hole j, E_b is binding energy of exciton.

The second contribution is enhancemnt of the absorption above transition energy by the Sommerfeld factor

$$S_{2D} = \frac{\exp\left(\pi/\sqrt{\Delta}\right)}{\cosh\left(\pi/\sqrt{\Delta}\right)}$$

where Δ is the total excess energy of the electron-hole pair normalized to $E_b/4$

Input File

In order to include excitonic correction to absorption, `excitons` section should be present both in `quantum{region{}}` and `optics{quantum_spectra{}}`.

In quantum, methods to compute excitons from conduction and valence band eigenstates are defined (see details in keywords documentation *“quantum {region {excitons} }”*)

```
quantum{
    region{
        ...
        excitons{
            density_averaged_masses = yes
            energy_cutoff = 2.5
            accuracy = 1e-5
        }
    }
}
```

In optics, the corrections to optical absorptions is defined. Setting `coulomb_enhancement = no` and `num_exciton_levels = 0` will output absorption without exciton correction (so called single-particel model).

```
optics{
    quantum_spectra{
        ...
        excitons{
            coulomb_enhancement = yes
            num_exciton_levels = 1
        }
    }
}
```

The input files provided for this simulation have three modes, depending on the value of the variable *$calculation*, defined at the top of the input file.

– *$calculation=1* – computes single-particle absorption (no exciton correction)

– *$calculation=2* – the computed absorption includes Coulomb enhancement

– *$calculation=3* – the computed absorption includes both Coulomb enhancement and exctiton peaks

Simulation 1: single-band model

For this simulation, *1D_InterbandExcitonAbsorption_InfiniteWell_GaAs_effective_mass_nnp.in* input file is used.

The parameters used in the calculation are the following

Property	Symbol	unit	Value
quantum well width	L	nm	10.0
barrier height	E_b	eV	1000
Electron effective mass	m_e	m_0	0.065
Heavy hole effective mass	m_{hh}	m_0	0.51
refractive index	n_r		3.3
linewidth (FWHM) Lorentzian	Γ_L	meV	3
linewidth (FWHM) Gaussian	Γ_G	meV	5
temperature	T	K	300

To simplify the calculation, only heavy hole states are computed in the valence band. To include light hole and split off, set *$compute_LH_and_SO* variable to 1 in the input file.

The eigenstates from the calculation are shown in the Figure 4.12.2.1

In the figure below, the computed absorption in the quantum well is shown (Figure 4.12.2.2). The figure shows the absorption without exciton correction, absorption including Sommerfeld enhancement factor and total excitonic absorption (i.e. both exciton peak and Coulomb enhancement).

Simulation 2: 8-band kp model

For this simulation, *1D_InterbandExcitonAbsorption_InfiniteWell_GaAs_8kp_nnp.in* input file is used.

The parameters used in the calculation are the following

Property	Symbol	unit	Value
quantum well width	L	nm	10.0
barrier height	E_b	eV	1000
8-band kp parameters for GaAs	Eg, Ep, L, M, N	n/a	from default database
refractive index	n_r		3.3
linewidth (FWHM) Lorentzian	Γ_L	meV	3
linewidth (FWHM) Gaussian	Γ_G	meV	5
temperature	T	K	300

The eigenstates from the calculation are shown in the Figure 4.12.2.3

In the figure below, the computed absorption in the quantum well is shown (Figure 4.12.2.4). Similarly to the Simulation 1, the figure shows the absorption with and without exciton correction.

In both simulations, exciton correction increase the absorption significantly above the absorption edge and also gives rise to a sharp peak at energy few meV below absorption edge.

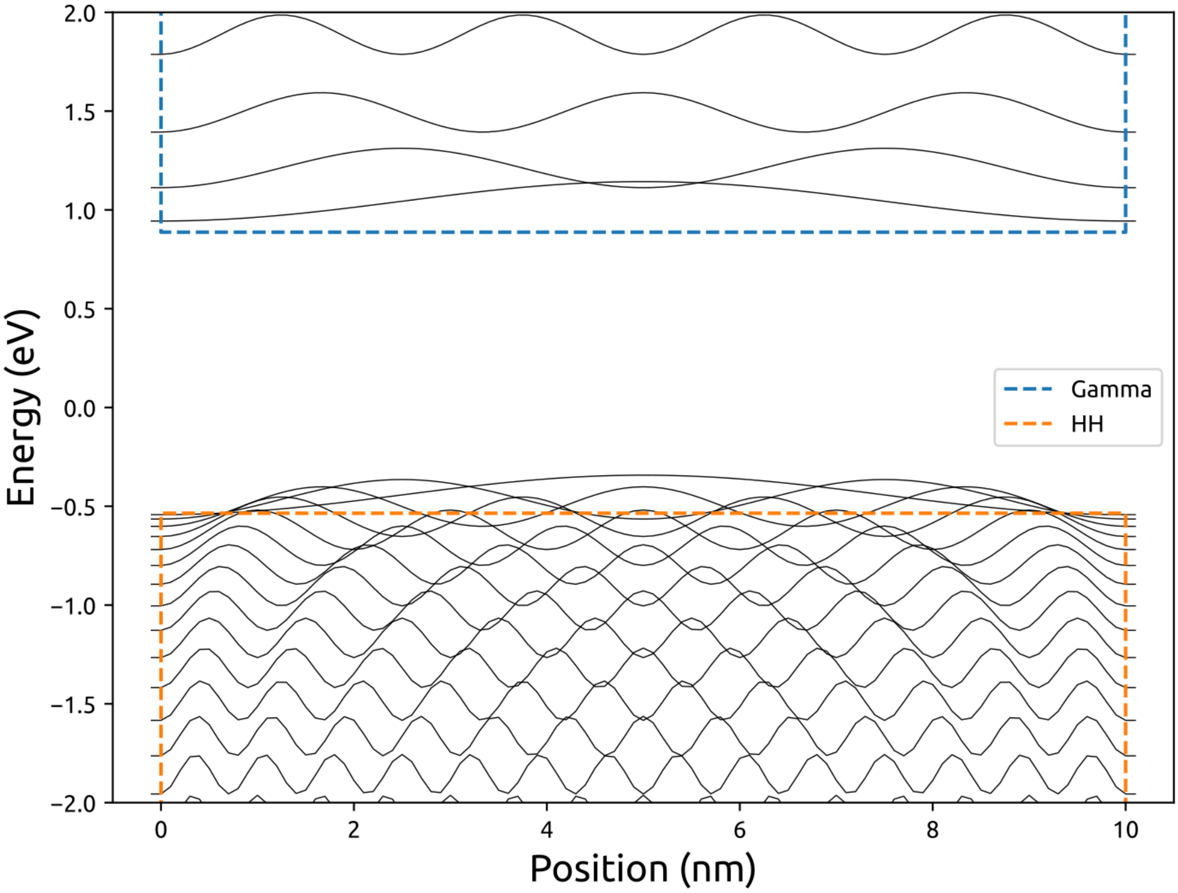


Figure 4.12.2.1: Computed eigenstates in the GaAs infinite quantum well with effective mass Hamiltonian in conduction and valence bands. The colored dashed line are band edges, the solid lines are eigenstates.

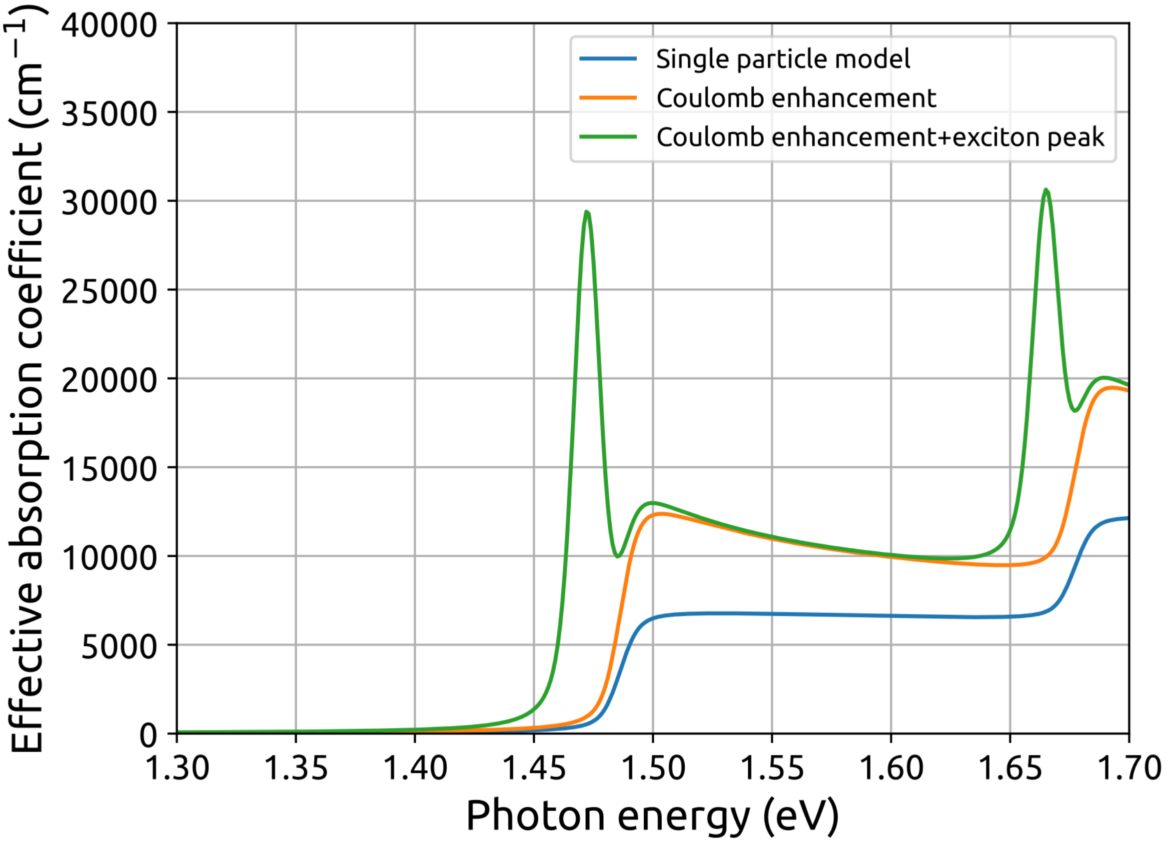


Figure 4.12.2.2: Absorption in infinite quantum well computed with effective mass Hamiltonians. The figure shows absorption with and without exciton correction.

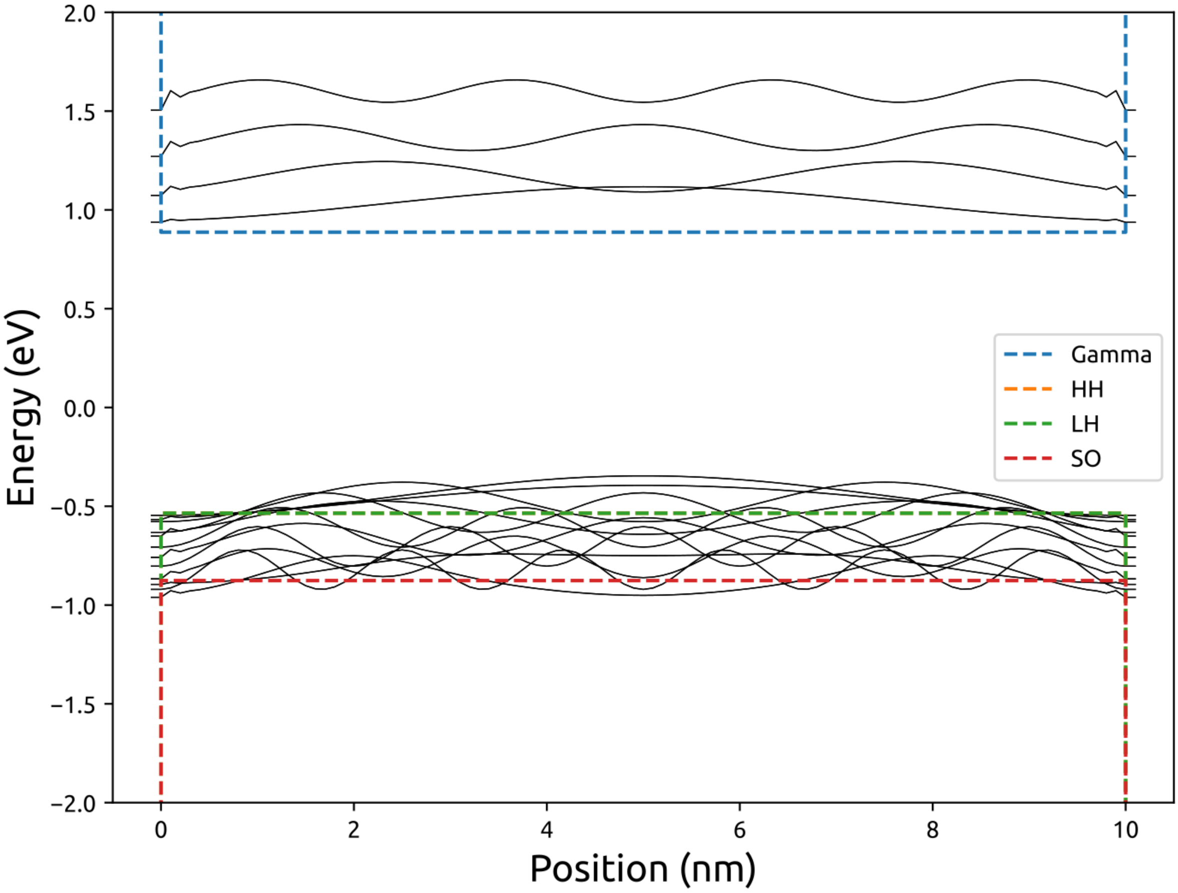


Figure 4.12.2.3: Eigenstates in the GaAs infinite quantum well computed with 8-band kp Hamiltonian. THe colored dashed line are band edges, the solid lines are eigenstates.

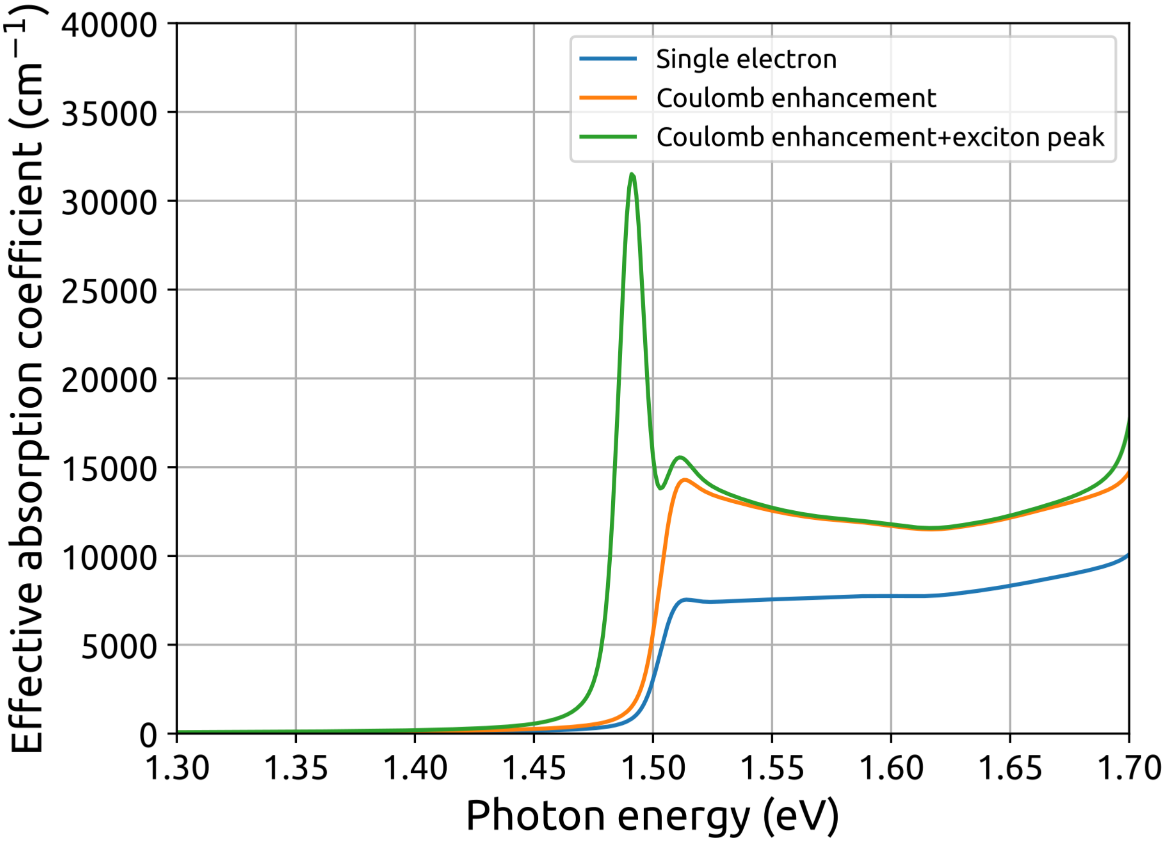


Figure 4.12.2.4: Absorption in infinite quantum well computed with 8-band kp Hamiltonian. The figure shows absorption with and without exciton correction.

This tutorial is based on the nextnano GmbH collaboration in the scope of the SiPho-G Project aiming at development of ultrahigh-speed optical components for next-generation photonic integrated circuits, and it is funded by the European Union's Horizon 2020 research and innovation program under the grant agreement No 101017194.

Last update: 2025/06/27

SiGe QW excitonic absorption

> **Attention**
>
> This tutorial is under construction.

Input files:
1D_Ge_GeSi_QCSE_Lever2010_8kp_nnp_exciton.in

Scope of the tutorial:
In this tutorial, we show an approach how to model absorption spectrum in a quantum well. This tutorial reproduces results from *[LeverJLT2010]*.

The most relevant keywords:

- `contacts`
- `optics{ quantum_spectra{} }`
- `quantum{excitons}`

Solvers:

- `strain`
- `poisson`
- `quantum`
- `quantum_optics`

Relevant output files:
bias_xxxxx\Optics\absorption_quantum_region_TE_eV.dat

Introduction

This tutorial shows how to model an absorption inside a quantum well — an active region of electro-absorption modulator. The tutorial reproduces results from *[LeverJLT2010]* with 9 nm Ge well with 12 nm Si $_{0.4}$ Ge $_{0.6}$ barrier grown on Si $_{0.3}$ Ge $_{0.7}$ substrate. The Ge concentration profile is smoothened by interdiffusion, which is modelled using analytic profile from *[LeverJLT2010]*. The Ge grown on the Si substrate is tensile strained, because the bulk thermal expansion coefficient of Ge is larger than of the Si substrate. In order to take in into account, 0.1% tensile residual strain is added to virtual substrate.

```
strain{
    residual_strain = 0.001
...
}
```

The figure Figure 4.12.2.5 shows the wave functions in conduction and valence bands.

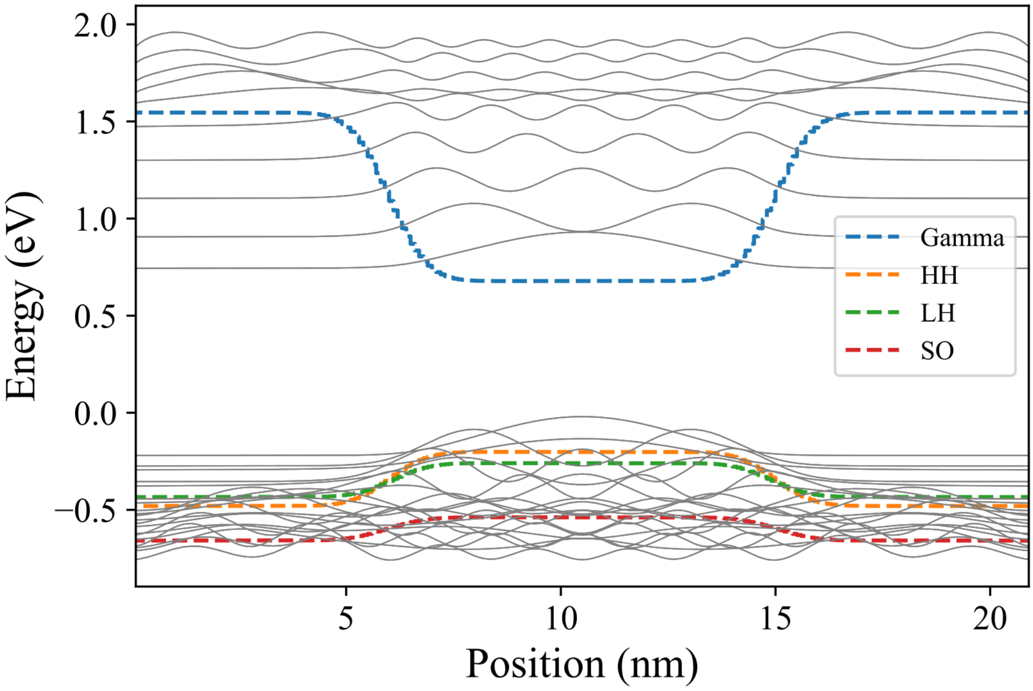


Figure 4.12.2.5: The band edges (colored) and the wave function probabilities (gray) in the quantum well under 0 bias.

The bias sweep from 0 V to 0.5 V is specified in the input file in the contacts

```
$left_bias_start = 0
$left_bias_finish = 0.27
...
contacts{
    ohmic{ name = "left" bias = [$left_bias_start, $left_bias_finish] steps = 3}
    ohmic{ name = "right" bias = 0}
}
```

For each bias the absorption spectrum in the device is calculated. Due to the quantum confinement, the excitonic absorption is still observable at room temperatures. The excitonic correction is added; more details are explained in tutorial “Optical interband absorption in a quantum well including excitonic effects” for *nextnano³*. The absorption spectra at different biases is shows in the figure Figure 4.12.2.6.

The redshift of exciton peak is observed when bias is applied to the structure. At a given wavelength, the absorption increase is significant allowing for electro-optic absorption modulation. The modelling can be used to optimize the parameters of the device and to choose the optimal wavelength of the modulation for a given structure.

The position of exciton peaks are in a good agreement with simulation from *[LeverJLT2010]* — within 1 meV error for each bias. While the relative change of absorption spectra with applied bias also agrees with experimental data, the absolute value differs by a factor 1.4 – 1.6. The nextnano software is continuously improving to meet last criteria as well.

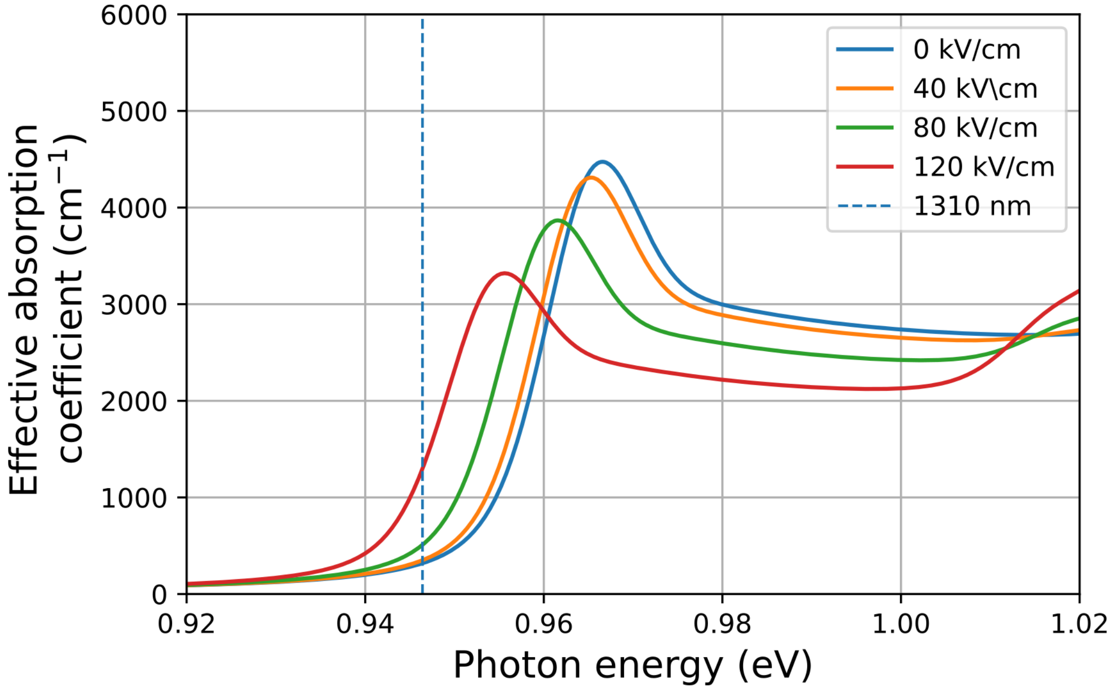


Figure 4.12.2.6: Excitonic absorption spectra in the device. Labels indicate electric field in the middle of the quantum well.

This tutorial is based on the nextnano GmbH collaboration in the scope of the SiPho-G Project aiming at development of ultrahigh-speed optical components for next-generation photonic integrated circuits, and it is funded by the European Union's Horizon 2020 research and innovation program under the grant agreement No 101017194.

SiPho-G
Advanced GeSi components for next-generation silicon photonics applications

---

Last update: 2025/06/27

SiGe MQW QCSE electro-absorption modulator (EAM)

Attention

This tutorial is under construction.

Input files:
1D_Ge_GeSi_QCSE_Kuo2005_8kp_nnp_exciton.in 1D_Ge_GeSi_QCSE_Kuo2005_simplified_8kp_nnp_exciton.in

Scope of the tutorial:
In this tutorial, we show an approach how to model absorption spectrum for a series of quantum wells inside p-i-n junction. This tutorial reproduces experimental results from *[KuoNature2005]*.

The most relevant keywords:

- `contacts`
- `optics{ quantum_spectra{} }`
- `quantum{excitons{} kp8{}}`

Relevant output files:

bias_xxxxx\bandedges.dat bias_xxxxx\Optics\absorption_quantum_region_TEy_eV.dat structure\density_acceptor.dat structure\density_donor.dat

Introduction

In this tutorial, we will explore the physics behind the quantum-confined Stark effect (QCSE) and its application in modulating light absorption in semiconductor structures. The QCSE is a phenomenon in which the absorption edge of a semiconductor is shifted when an electric field is applied perpendicular to its surface, resulting in a redshift of the exciton peak. This effect can be utilized in electro-optic devices such as modulators, switches, and tunable lasers.

We will begin by simulating the QCSE in a simplified structure consisting of only an intrinsic region. In the second example, we will simulate a more complex structure consisting of a p-i-n junction with the quantum well embedded in the intrinsic region. This will provide us with a more accurate representation of the electro-optic properties of a complete device and allow us to investigate the behavior of absorption versus bias.

This tutorial reproduces the experimental results from *[KuoNature2005]*. The design consists of the p-doped buffer, grown on Si substrate, bottom spacer, series of 10 quantum wells, top spacer and n-doped cap layer. The device parameters are given below

Name	Thikness, nm	Ge concentration	doping type	doping concentration, cm^{-3}
Buffer	500	0.9	p-type	5×10^{18}
Bottom spacer	100	0.9	—	—
Barrier	16	0.85	—	—
Well	10	1.0	—	—
Top spacer	100	0.9	—	—
Cap layer	200	0.9	n-type	1×10^{19}

The modeling approach used in this tutorial is similar to the one used in our previous tutorial on SiGe excitons — *"SiGe QW excitonic absorption"*. Specifically, the Ge content profile is smoothed with a characteristic diffusion length of 1 nm, and a residual tensile strain of 0.1% is assumed in the strain-relaxed buffer. The electron states are computed with 8-band kp hamiltonian.

In both examples, the `quantum_region` consists only of 1 well. It is sufficient to model only valence and conduction states in a single quantum well, because the barrier is wide enough, so there is no overlap of wave functions between different wells.

Simulation 1: Only intrinsic region

Solvers:

- `strain`
- `poisson`
- `quantum`
- `quantum_optics`

Omitting the doped layers, the simulation regions consist of bottom spacer, MQW and top spacer. The built-in potential of the junction has to be included in the simulation. Furthermore, the diffusion of dopants from a buffer and cap layer effectively decreases the intrinsic region. Following *[LeverJLT2010]*, the built-in is assumed to be $0.8V$ and the intrinsic region is shortened by $75nm$ (i.e. the bottom spacer region used in the simulation is $25nm$)

Since Simulation 1 example does not include any doped regions, the current equation is not necessary and can be omitted.

The simulated banedges at zero bias are shown in the figure Figure 4.12.2.7

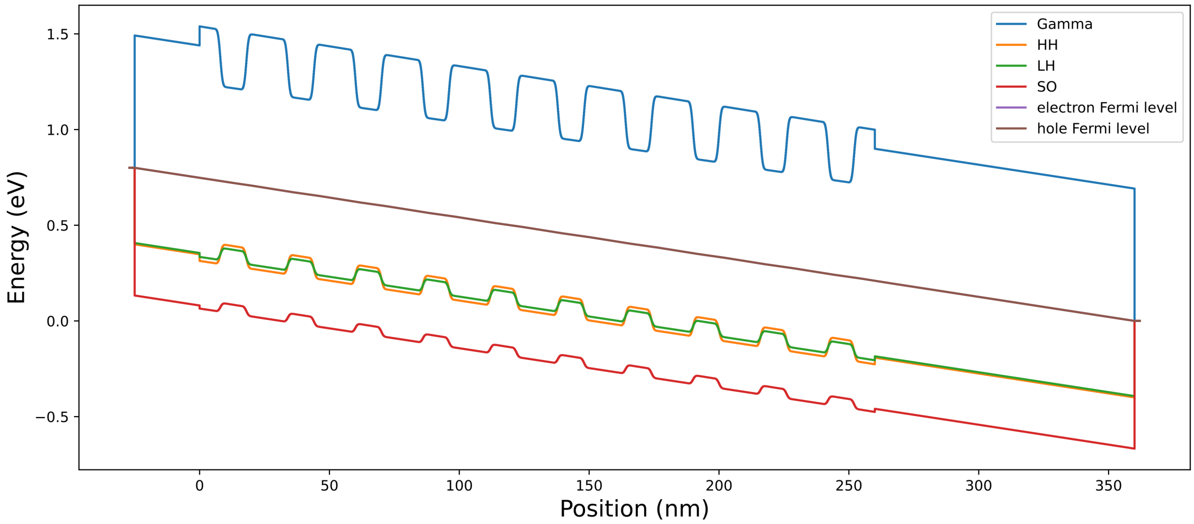


Figure 4.12.2.7: Valence and conduction band edges at zero bias. The electric field is induced by 0.8V built-in potential, included in the simulation

When additional bias applied, the electric filed in the quantum well is increased (figure Figure 4.12.2.7)

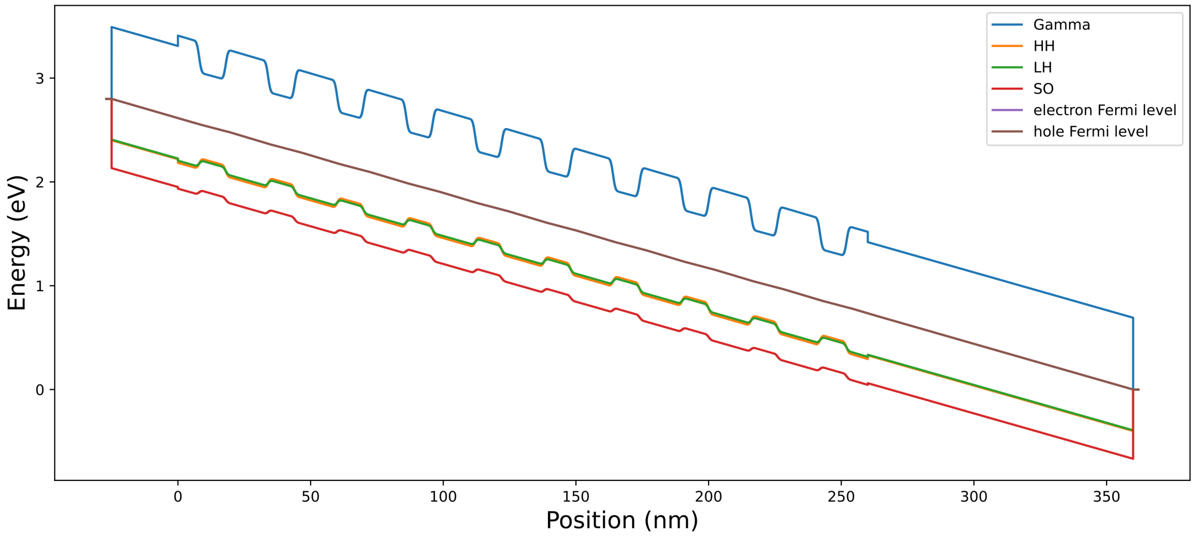


Figure 4.12.2.8: Valence and conduction band edges at 2V external bias.

The absorption spectra computed at different biases is given in the figure

Simulation 2: Whole pin device

Solvers:

- `strain`
- `current_poisson`
- `quantum`
- `quantum_optics`

For a more accurate representation of the electro-optic properties of a complete device, we will consider a more detailed structure consisting of a p-i-n junction with the quantum well embedded in the intrinsic region.

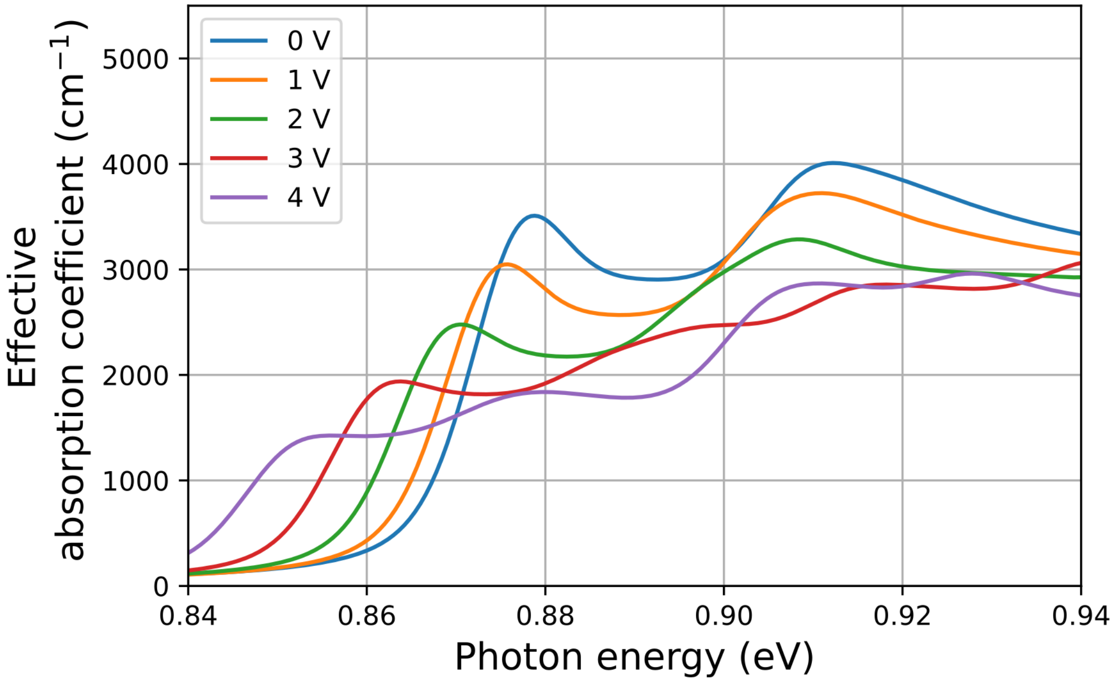


Figure 4.12.2.9: Absorption spectra inside the MQW region at different external bias

In contrast to the simplified example, the second example involves the inclusion of doped regions in the simulation. This necessitates the use of current equation to model the device behavior.

As discussed above, the diffusion of dopants from the p and n regions to the intrinsic region effectively decreases the width of the intrinsic region, which increases the electric field. In order to model this phenomenon, we use a smoothed doping profile corresponding to the analytical solution of diffusion between two infinite half-spaces with a constant initial concentration c_0 in one subspace and zero concentration in the other.

$$c = \frac{c_0}{2} + \frac{c_0}{2} erf(\pm\frac{x - x_0}{d})$$

Here: x_0 is junction position, d is characteristic diffusion length, erf is error function, plus inside error function is for the case when initial nonzero concentration is at $x > x_0$ and vice versa.

We find $d = 30nm$ to give the closest result to experiment. In order to use the diffused doping profile, we initialize doping profile function in import

```
import{
...
    analytic_function{
        name     = "pdoping_profile"
        function = "$pDopingConcentration*0.5 + $pDopingConcentration*0.5*erf(-(x-
↪$pdoping_junction_position)/$diffusion_dopants_length)"
        }
    analytic_function{
        name     = "ndoping_profile"
        function = "$nDopingConcentration*0.5 + $nDopingConcentration*0.5*erf((x-
↪$ndoping_junction_position)/$diffusion_dopants_length)"
        }
}
```

These functions are used in structure to initialize doping

```
impurities{
    donor{
        name = "n-type"
        energy = -1000  # (= all ionized)
        degeneracy = 2  # degeneracy of energy levels, 2 for n-type, 4 for p-type
    }
    acceptor{
        name = "p-type"
        energy = -1000  # (= all ionized)
        degeneracy = 4  # degeneracy of energy levels, 2 for n-type, 4 for p-type
    }
}
...
structure{
...
    region{ # n-doping
        line{
            x = [$x_min, $x_max]
        }
        doping{
            import{
            name = "n-type"
            import_from = "ndoping_profile"
            }
        }

    }

    region{ # p-doping
        line{
            x = [$x_min, $x_max]
        }
        doping{
            import{
                name = "p-type"
                import_from = "pdoping_profile"
            }
        }
    }
...
}
```

The resulting doping profile is shown in the figure Figure 4.12.2.10.

The band edges at zero bias is shown in the figure Figure 4.12.2.11

At zero bias there is no current in the system, therefor the electron and hole Fermi levels coincide. At nonzero reverse bias, the current is induced, separating electron and hole Fermi level and enhancing the electric field in the MQW region (see Figure 4.12.2.12).

The absorption spectra computed in this example are shown in the Figure 4.12.2.13.

The position of exciton peaks are in a good agreement with experiment — within 3 meV error for each bias. While the relative change of absorption spectra with applied bias also agrees with experimental data, the absolute value differs by a factor $1.5 - 1.8$. The nextnano software is continuously improving to meet last criteria as well.

This tutorial is based on the nextnano GmbH collaboration in the scope of the SiPho-G Project aiming at development of ultrahigh-speed optical components for next-generation photonic integrated circuits, and it is funded by the European Union's Horizon 2020 research and innovation program under the grant agreement No 101017194.

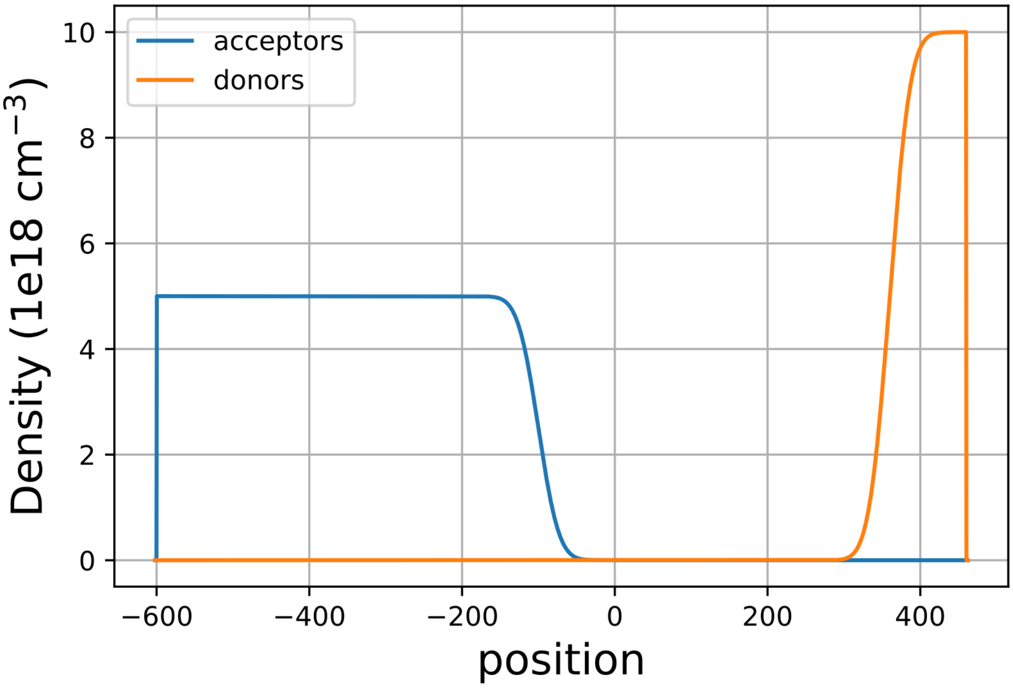


Figure 4.12.2.10: The doping profile in the device

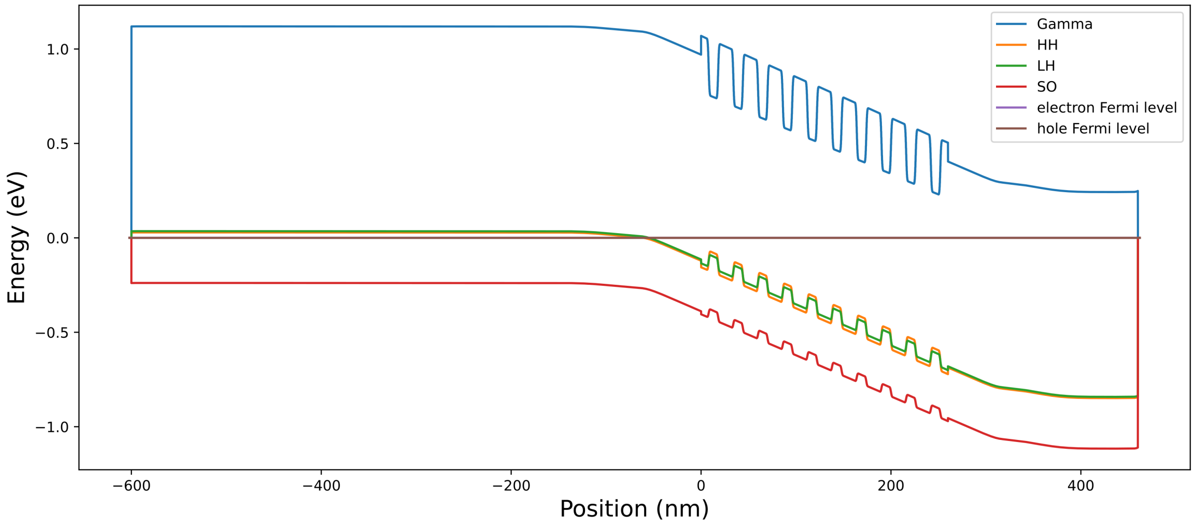


Figure 4.12.2.11: Valence and conduction band edges at zero bias

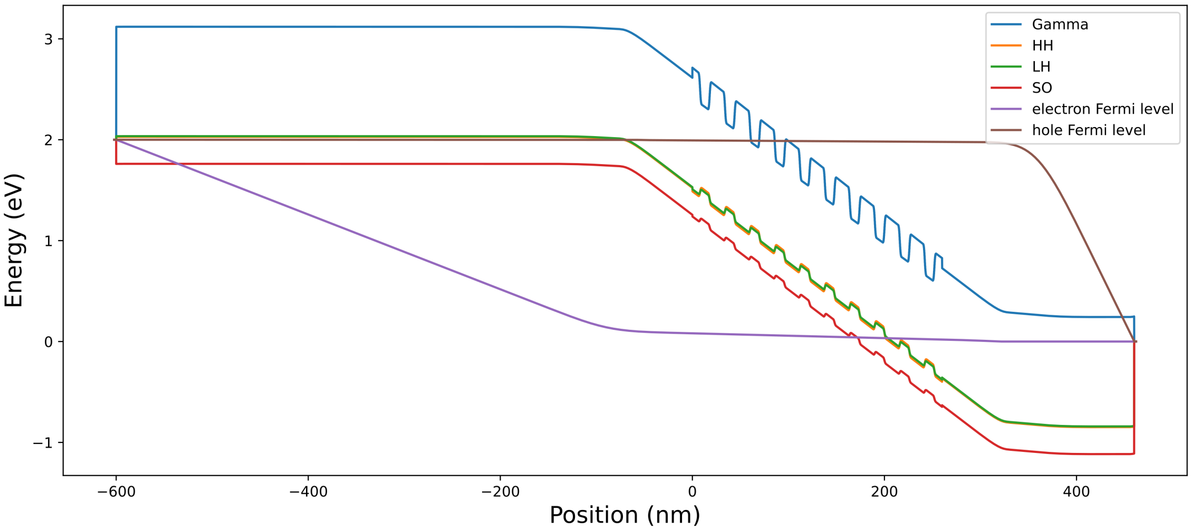


Figure 4.12.2.12: Valence and conduction band edges at zero bias

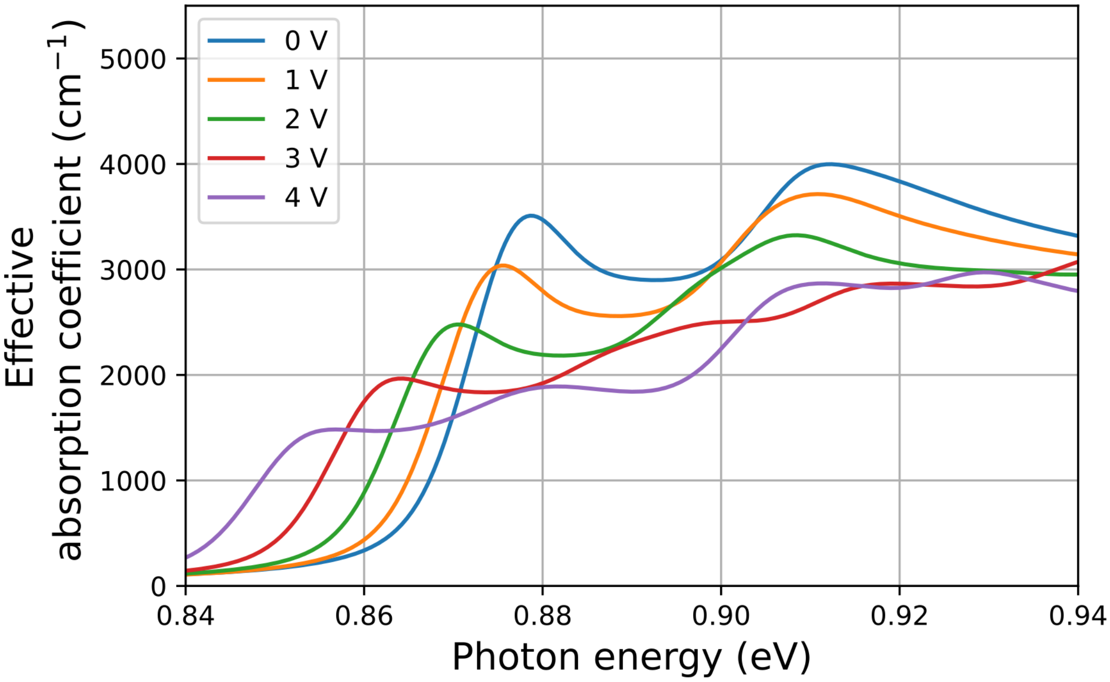


Figure 4.12.2.13: Absorption spectra inside the MQW region at different external bias for Simulation 2

SiPho-G

Advanced GeSi components for next-generation silicon photonics applications

---

Last update: 2025/06/27

4.13 2-Dimensional Electron Gases (2DEGs)

4.13.1 — FREE — Schrödinger-Poisson - A comparison to the tutorial file of Greg Snider's code

In this tutorial we calculate the self-consistent solution of Schrödinger-Poisson equations using *nextnano++* and another code provided by Greg Snider (University of Notre Dame). We compare the two results and see the agreement of them.

We also discuss about the basic concept of the Schrödinger-Poisson solution.

The related input files are followings:

- *Greg_Snider_MANUAL_1D_nn*.in*
- *Greg_Snider_MANUAL_1D_analytic_nn*.in*
- *Greg_Snider_MANUAL_2D_nn*.in*
- *Greg_Snider_MANUAL_2D_analytic_nn*.in*

These are available in the sample file folder. The files which have *analytic* in their names use analytic doping function.

We appreciate that Greg Snider provided his code, the manual and the input files free of charge, so that we could use it here as a test case. His 1D Poisson/Schrödinger code can be obtained from this link. This tutorial is based on his manual (*1D Poisson Manual.pdf*, *MANUAL.EX*).

Structure

We simulate a structure consisting of the following matrerials and doping profile. The additional doping profile based on LSS Theory is explained in the next section.

	surface	Schottky barrier of 0.6V
z = 0 ~ 15 nm	GaAs	n-type doped (10^{18} cm^{-3})
z = 15 ~ 35 nm	$Al_{0.3}$ $Ga_{0.7}$ As	n-type doped (10^{18} cm^{-3})
z = 35 ~ 39.5 nm	$Al_{0.3}$ $Ga_{0.7}$ As	
z = 39.5 ~ 54.5 nm	GaAs	quantum well
z = 54.5 ~ 105 nm	$Al_{0.3}$ $Ga_{0.7}$ As	
z = 105 ~ 355 nm	$Al_{0.3}$ $Ga_{0.7}$ As	p-type doped (10^{17} cm^{-3})
	substrate	

- The grid resolution is 1 nm with the exception of the 250 nm layer which has a resolution of 5 nm and the material interfaces of the quantum well which has a resolution of 0.5 nm.
- The dopants are assumed to be fully ionized.
- The temperature is 300 K.
- The Schrödinger equation will be solved between 5 nm and 195 nm.

Doping

We consider two further impurity profile resulting from ion implantation using **LSS Theory**.

For further details see for example: "Very brief Introduction to Ion Implantation for Semiconductor Manufacturing" by Gerhard Spitzlsperger.

The donor and acceptor profiles are written out of the file *density_acceptor/acceptor.dat* and look as follows:

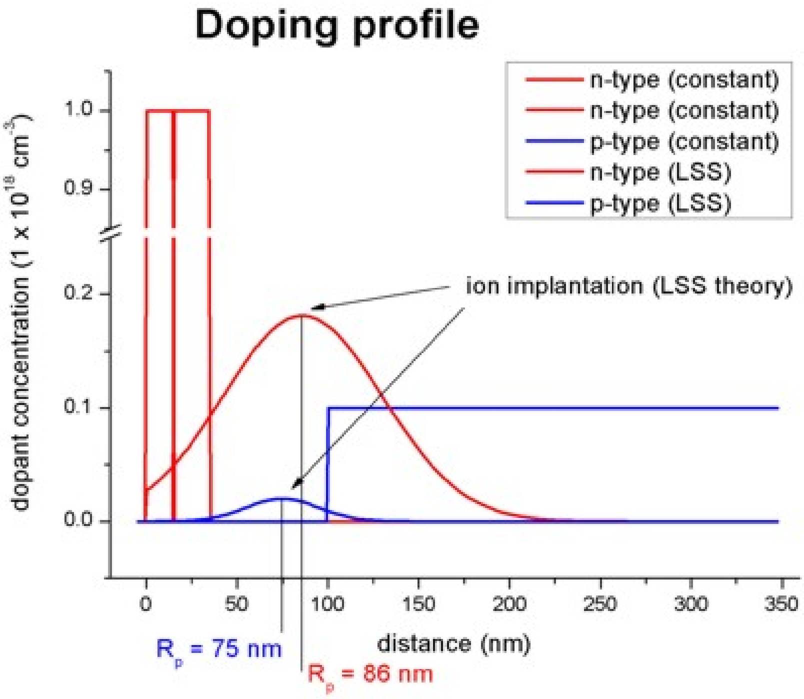


Figure 4.13.1.1: Doping profiles separated by each region.

The relevant parameters are:

implant	dose [cm^{-2}]	projected range R_p [nm]	projected straggle Delta σ_p [nm]
donor	2×10 ^{12}	86	44
acceptor	1×10 ^{11}	75	20

For further details on the LSS theory (ion implantation) and on the doping profiles, please check the relevant keyword *doping{ }*.

Conduction and valence band edges

The following figure shows the conduction and valence band edges as well as the Fermi level (which is constant and has the value of 0 eV) for the structure specified above. These bands are the solutions of the self-consistent Schrödinger-Poisson equation.

Both codes, *nextnano*++ and Greg Snider's "1D Poisson" lead to the same results.

Electron eigenstates and eigenfunctions

Inside the GaAs quantum well there are three confined electron states. The ground state is below the Fermi level and thus occupied. The following figure shows a zoom of the GaAs Quantum well.

The wave functions as calculated with *nextnano*++ are nearly identical to Greg Snider's "1D Poisson" code, as well as the energies. However, there are tiny differences which is not too suprising as the conduction band profile is not completely identical.

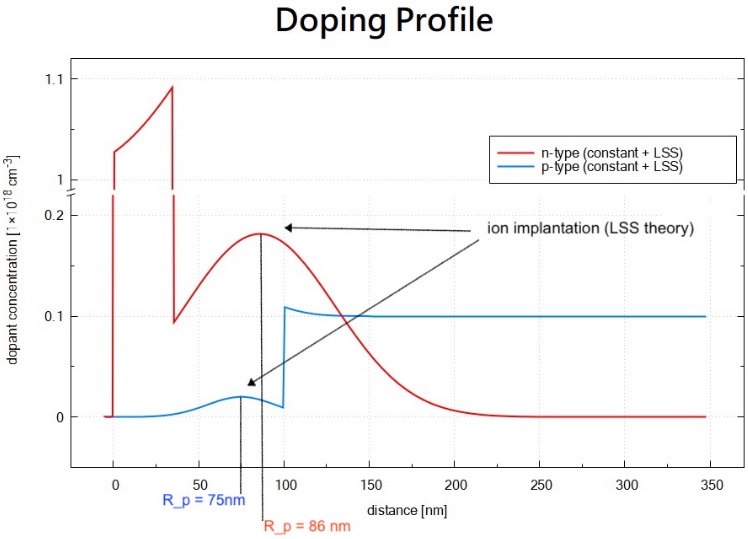
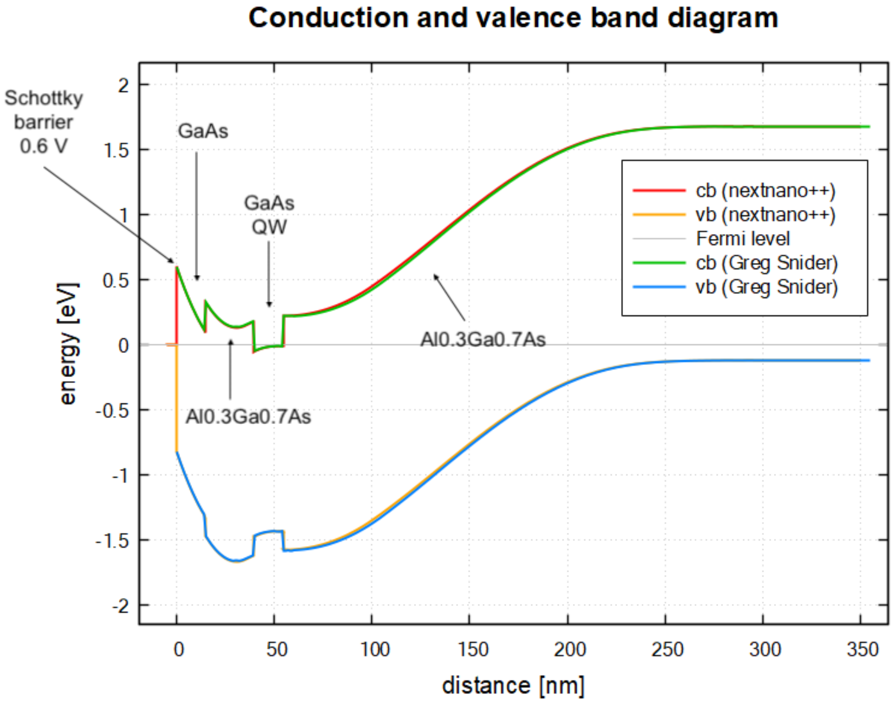
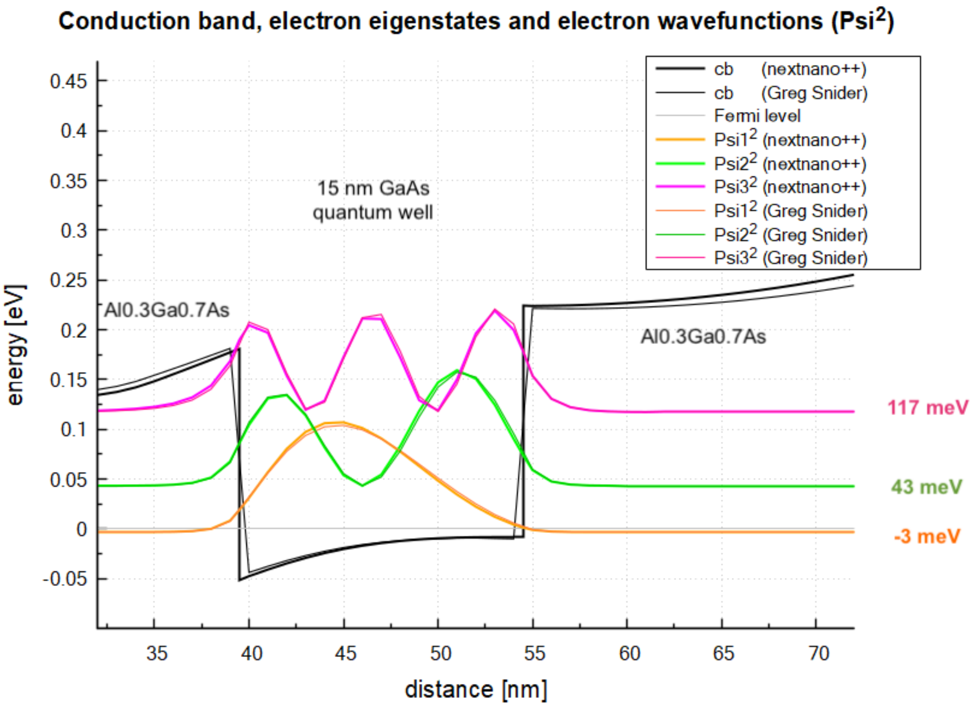


Figure 4.13.1.2: Resulting doping profiles.





Electron states	*nextnano*++	Greg Snider's "1D Poisson" code
E_1 [meV]	-3.1	-1.3
E_2 [meV]	43.4	44.0
E_3 [meV]	117.4	117.8

Electron and hole densities

The electron and hole densities are depicted in this figure, there is also nice agreement between the two codes.

- The integrated electron density in the GaAs quantum well region is 0.667 * 10^{12} cm^{-2}. (Greg Snider's result: 0.636 * 10^{12} cm^{-2})
- The integrated hole density in the right most Al0.3Ga0.7As region is 1.033 * 10^{12} cm^{-2}. (Greg Snider's result: 1.085 * 10^{12} cm^{-2})

The relevant output files are:

- *integrated_density_electron.dat*
- *integrated_density_hole.dat*

This tutorial shows very nicely that both codes, *nextnano*++ and Greg Snider's "1D Poisson" lead to the same results. Greg Snider's 1D Poisson/Schrödinger code can be obtained from here: http://www.nd.edu/~gsnider/

2D simulations

- *Greg_Snider_MANUAL_2D_nn*.in*

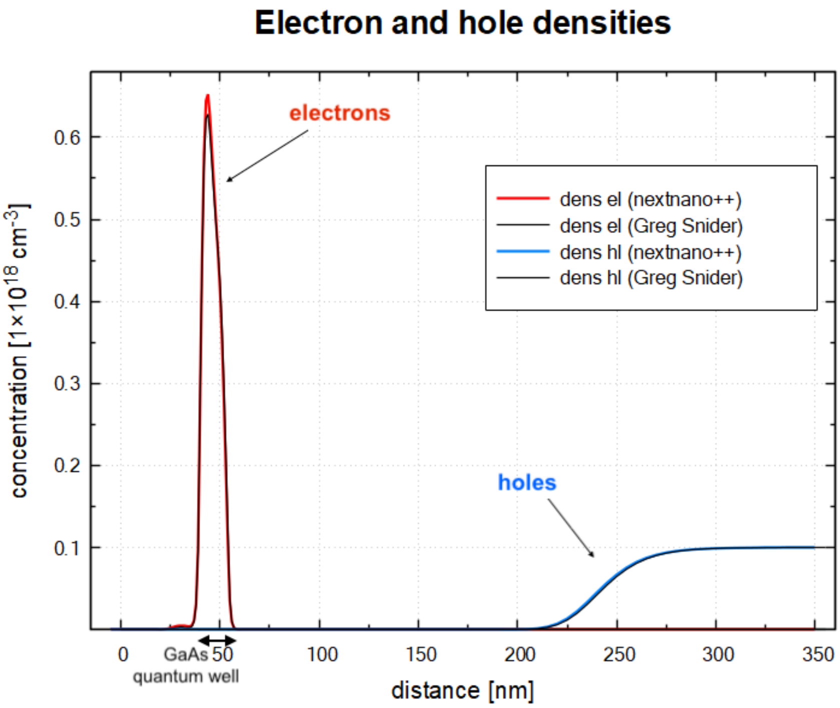
We can also calculate the 2D schrödinger-Poisson equation for the same structure where the y direction has been assumed to be of length 100 nm with periodic boundary conditions.

Self-consisent Schrödinger-Poisson solution

Here we briefly discuss about the basic concept of the method used to get the above results.

In this section, we refer to

- P. Harrison and A. Valavanis, *Quantum Wells, Wires and Dots*, (Wiley, 2016, Fourth Edition)



- I.-H. Tan, G. L. Snider, L. D. Chang, and E. L. Hu, A self-consistent solution of Schrödinger-Poisson equations using a nonuniform mesh, Journal of Applied Physics 68 (1990), no. 8, 4071-4076'

Self-consistent calculation of Schrödinger-Poisson equations is one way to treat the manybody effects associated with Coulomb repulsion.

For example, suppose we calculate Schrödinger equation to obtain the energy eigenvalues and eigenstates for a quantum well only one time. If we add a further test electron into the system, the potential that the test electron feels is the band-edge potential plus Coulomb potential which is caused by the original electrons in the system. In most cases, the carrier density in a single quantum well is so high that it is important to take this additional potential into consideration. (6.67*10^{12} cm^{-2} for the GaAs quantum well in this tutorial.)

In order to obtain the solution which involves this effect, **the potential** used in Schrödinger equation for the electrons and **the charge distribution** which is based on the energy eigenstates from that Schrödinger equation must satisfy Poisson equation. This solution is described as *self-consistent*, rather like Hartree's approach to solving many electron atoms.

The process for obtaining self-consistent solution of Schrödinger-Poisson equations is as follows:

1. Solve Schrödinger equation using band-edge potential $V_{be}(\mathbf{r})$ and obtain the eigenstates of an electron $\Psi^{el}_{\alpha,E}(\mathbf{r})$ and hole $\Psi^{hole}_{\beta,E}(\mathbf{r})$. Here α is the conduction band number, β is the valence band number and E represents the eigenvalue.
2. Calculate the density distribution of the particles $n(\mathbf{r})$ using local density of state $\rho^{el}(\mathbf{r},E) := \sum_\alpha |\Psi^{el}_{\alpha,E}(\mathbf{r})|^2$, $\rho^{hole}(\mathbf{r},E) := \sum_\beta |\Psi^{hole}_{\beta,E}(\mathbf{r})|^2$ and Fermi distribution $f(E) := \frac{1}{e^{(E-E_f)/k_BT}+1}$.

$$n^{el}(\mathbf{r}) := \int dE \rho^{el}(\mathbf{r},E)f(E)$$
$$n^{hole}(\mathbf{r}) := \int dE \rho^{hole}(\mathbf{r},E)f(E)$$

3. Solve Poisson equation and obtain the potential distribution $\phi(\mathbf{r})$ caused by the distributed electrons, holes, and ions.

$$\nabla\cdot(\epsilon_s(\mathbf{r})\nabla)\phi(\mathbf{r}) = \frac{-e[n^{hole}(\mathbf{r}) - n^{el}(\mathbf{r}) + N_D(\mathbf{r}) - N_A(\mathbf{r})]}{\epsilon}$$

where ϵ_s is the dielectric constant, $N_D(\mathbf{r})$ is the donor concentration and $N_A(\mathbf{r})$ represents the acceptor concentration.

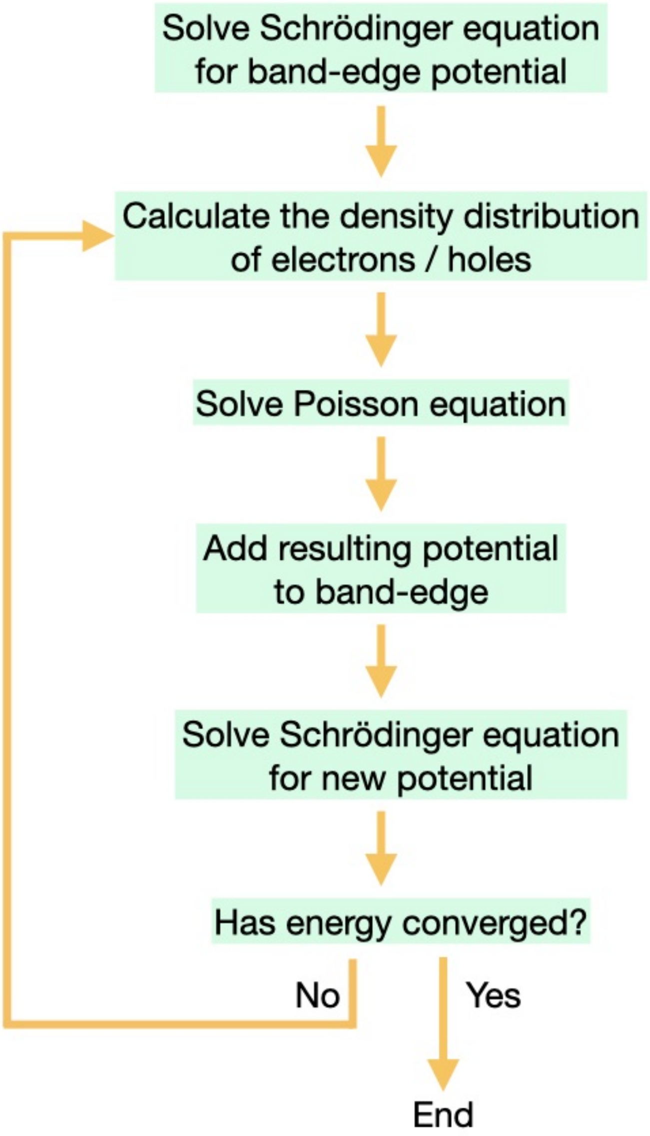
4. Using the new potential

$$V_{new}(\mathbf{r}) := -q\phi(\mathbf{r}) + V_{be}(\mathbf{r})$$

which consists of the result of 3. and band-ege potential, solve Schrödinger equation.

5. Check whether the energy eigenvalues converged or not. Then
   - Yes ⟶ End
   - No ⟶ Go to 2.

The process is iterated until the energy eigenvalues converge. At last, the potential used in Hamiltonian and one calculated from charge distribution which is from Schrödinger equation will be identical.



Schematics of the self-consistent iteration

Last update: nnnn/nn/nn

4.13.2 Si/SiGe MODQW (Modulation Doped Quantum Well)

Input files:

- *1DSiGe_Si_Schaeffler_SemicondSciTechnol1997_nnpp.in*

Scope:

This tutorial aims to reproduce Fig. 11 of *[Schäffler1997]*.

Introduction

Layer sequence

	width [nm]	material	strain	doping [cm^{-3}]
1		Schottky barrier 0.8 eV		
2	15.0	Si cap	strained w.r.t $Si_{0.75}$ $Ge_{0.25}$	
3	22.5	$Si_{0.75}$ $Ge_{0.25}$ layer		
4	15.0	$Si_{0.75}$ $Ge_{0.25}$ doping layer		$2 \cdot 10^{18}$ (fully ionized)
5	10.0	$Si_{0.75}$ $Ge_{0.25}$ barrier		
6	18.0	Si channel	strained w.r.t $Si_{0.75}$ $Ge_{0.25}$	
7	69.5	$Si_{0.75}$ $Ge_{0.25}$ buffer		

Material parameters

The material parameters were taken from *[Schäffler1997]*. The temperature was set to 0.1 K. The Si layers are strained pseudomorphically with respect to a $Si_{0.75}$ $Ge_{0.25}$ substrate (buffer layer).

Method

Self-consistent solution of the Schrödinger-Poisson equation within single-band effective-mass approximation (using ellipsoidal effective mass tensors) for both Delta conduction band edges.

Results

Figure 4.13.2.1 shows the self-consistently calculated conduction band profile and the lowest wave functions of an n-type Si/$Si_{0.75}$ $Ge_{0.25}$ modulation doped quantum well (MODQW) grown on a relaxed $Si_{0.75}$ $Ge_{0.25}$ buffer layer. The strain lifts the sixfold degeneracy of the lowest conduction band (Delta6) and leads to a splitting into a twofold (Delta2) and a fourfold (Delta4) degenerate conduction band edge.

Figure 4.13.2.2 shows the lowest three wave functions (Ψ^2) of the structure. Two eigenstates that have very similar energies and are occupied (i.e. they are below the quasi-Fermi level), whereas the third eigenstate is not occupied at 0.1 K.

The electron density (in units of $1 \cdot 10^{18}$ cm^{-3}) is plotted in Figure 4.13.2.3. The lowest states in each channel are occupied, i.e. are below the Fermi level. The integrated electron densities are:

- in the parasitic $Si_{0.75}$ $Ge_{0.25}$ channel: $0.75 \cdot 10^{12}$ cm^{-2}.
- in the strained Si channel: $0.66 \cdot 10^{12}$ cm^{-2}.

Last update: nnnn/nn/nn

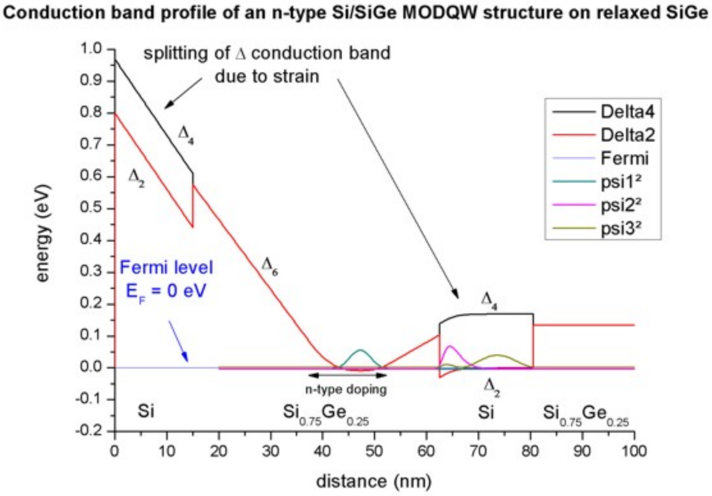


Figure 4.13.2.1: Calculated conduction band edge profile.

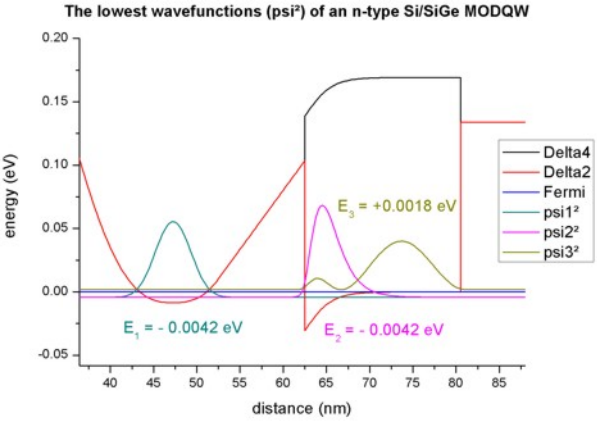


Figure 4.13.2.2: Calculated probability densities of the lowest electron states.

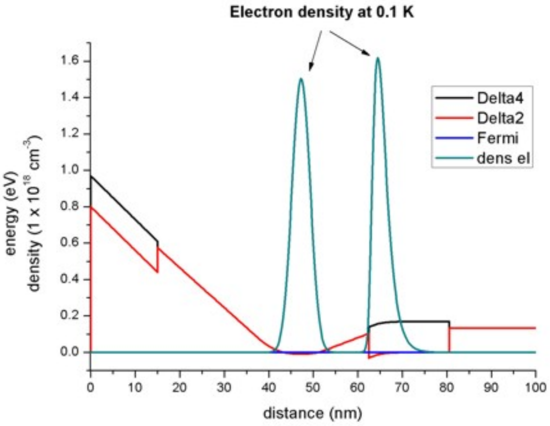


Figure 4.13.2.3: Calculated electron density profile.

4.13.3 — DEV — Shubnikov-de Haas effect and subband occupation of 2DEG

Attention

The tutorial is under development

Last update: nnnn/nn/nn

4.13.4 Depletion of electrons in a two-dimensional electron gas (2DEG)

In this tutorial you will learn how to setup an input file to simulate the electrostatic potential and the density of electrons in a 2DEG formed at the interface of a GaAs/AlGaAs layers.

Structure simulated

Figure 4.13.4.1 and Figure 4.13.4.2 present the simulated structure, where a two-dimensional electron gas is formed at the interface of the AlGaAs and GaAs (the substrate) materials. Doping the AlGaAs with n-type impurities at a certain distance of this interface improves the confinement of electrons in the 2DEG region. A GaAs layer over the n-AlGaAs region acts as a cap of the device. Finally metallic gates with different geometries are directly deposited on the top of surface.

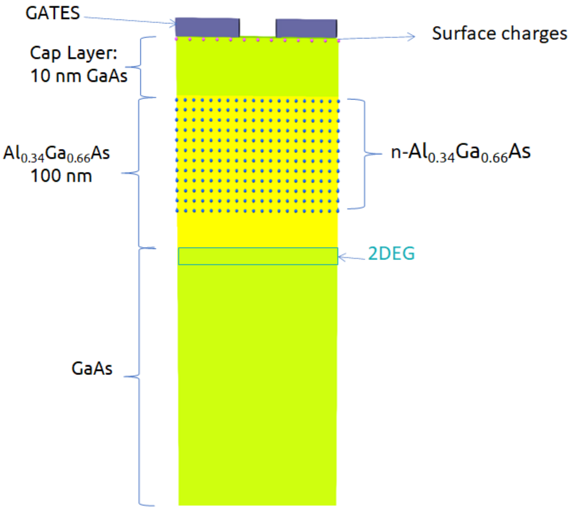


Figure 4.13.4.1: Schematics of a side view of the simulated device

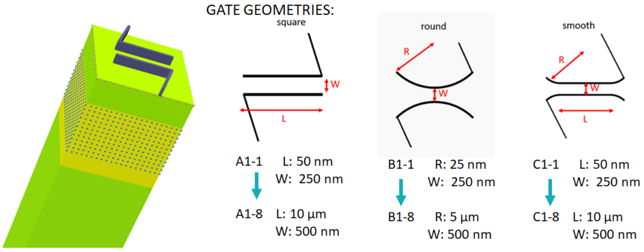


Figure 4.13.4.2: 3-dimensional schematics of the simulated structure and typical shapes of gates

In the scope of the project, the density and mobility of electrons in the 2DEG were measured at low temperatures, which were used in the calibration of the structure, in order to estimate the surface charge concentration at the interface of the cap layer and the surrounding environment (air). Additionally, the calibration also assists in the reduction of the incertainty of the doping concentration of the AlGaAs layer.

The methodology of combining simulations and experimental data was developed in the UltraFastNano project that can be found in the papers: *E. Chatzikyriakou et al., Unveiling the charge distribution of a GaAs-based nanoelectronic device A large experimental data-set approach, arXiv preprint arXiv:2205.00846, 2022* and *H. Edlbauer et al., Semiconductor-based electron flying qubits: review on recent progress accelerated by numerical modelling* (link)

Input files

The bias applied to the gates that depletes the electrons in the 2DEG (the pinch-off voltage) is powerful information that can be used to implement the building blocks of Electron Flying Qubits.

A simple method to define this voltage is by simulation of the same device for different voltages applied symmetric to the gates and to observe the value of the bias that depletes the carriers in some specific point of the 2DEG region (here, the center of the structure). The next two files can be used as an example how to set up the structure and all necessary variables for a self-consistent solution of Schrödinger and Poisson equations for performing 2D and 3D simulations.

Input files:

- QPC_1D_nnp.in
- QPC_2D_nnp.in (uniform grid of 0.25 nm)
- QPC_3D_nnp.in (nonuniform grid)

Using *nextnanopy* or the "Template" feature of *nextnanomat* input files can be automatically modified and executed. Also it is very helpful to define slices and 2D sections of the 2DEG region in the input file: this is a powerful tool for easy analysis of the data.

1D simulations

It is always a good strategy starting simulations in only one dimension in order to understand how the band edge of the conduction and valence bands influence the most important mechanisms under study. The 1D version of the input file is suitable for simulating the density of carriers in the 2DEG region, when a metallic layer deposited over the whole surface is biased at different voltages. This input file can also be used for calibration of the wafer when, for example, the density of electrons in the 2DEG is obtained experimentally.

2D Simulations

Before simulating 3D devices, that usually requires a large runtime to compute all relevant fields in the structure, it is always recommended starting modelling in only one or two, when possible.

In this specific example, 2D simulations can be used to tune the most important parameters of the physical model in order to reproduce, at least, the qualitative behavior of the experimental data.

The first animation (Figure 4.13.4.3) corresponds to the results of a 2D simulation of the device at the left in Figure1. It illustrates that the free electrons are confined in the 2DEG region and its density decreases as a negative bias V_{gate} is applied to both gates.

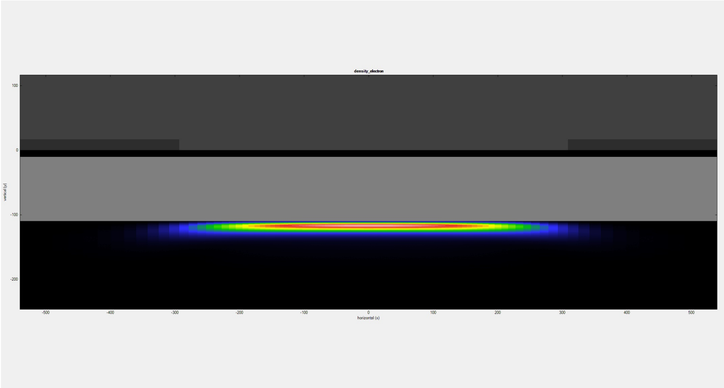


Figure 4.13.4.3: Density of electrons resulting from a 2D simulation as a negative bias is applied to the gates

A slice of the conduction band at the mid-distance between the gates across the growth direction is displayed in Figure 4.13.4.4. The results correspond to the cases where a 0, -1.00, -1.20 and -1.30 V bias is applied to the gates. Overlayed to theses plots, the density of electrons (dashed lines) show that the confinement of carriers at the 2DEG region actually occurs in the 2DEG and a depletion of carriers at this point is expected when very negative value of V_{gate} is applied to the device.

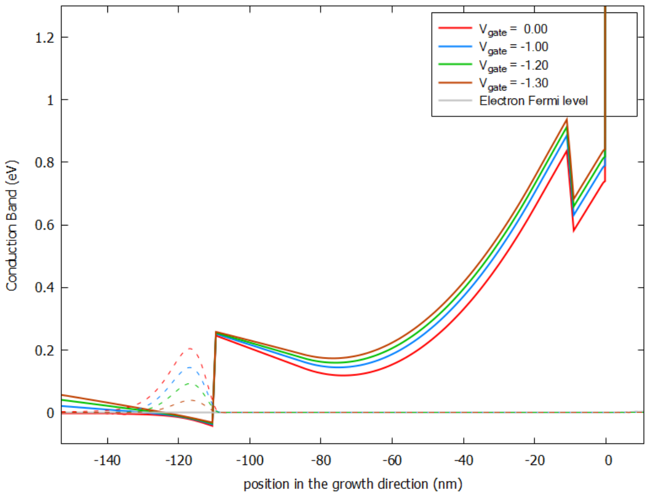


Figure 4.13.4.4: Conduction band (solid lines) and density of electrons (dashed lines) resulting for a 2D simulation as function of the applied bias to the gate. This plot corresponds to the results at the mid-distance of the gates across the growth direction.

3D Simulations

As mentioned before, 2D simulations can be very helpful for a first modeling of the device, and help to reduce the runtime. As shown before they are capable to reproduce the values of the pinch-off voltages for the case when the distance between the gates (W) are very small compared with their lengths (L).

Nevertheless in the most general case, 3D simulations can be required for more accurate estimation of the pinch-off voltage. Additionally, in the development of an Electron Flying Qubit building block computation of the conduction band through the whole device is necessary, in order to reproduce the transport phenomena in the 2DEG layer.

As the simulation time depends on the number of the nodes on the grid, for more complex forms and for large devices (of order of microns) with required fine grid (of order of nm), some computers might not have enough memory for the numerical solution of a self-consistent calculation of the Schrödinger and Poisson equations, with a minimum number of wave functions required for such operation.

In this case, a new algorithm was developed within *nextnano*++ that decomposes the 3D-problem in multiple 1D-problems. In this example, the Schrödinger-Poisson system is solved along the growth direction independently for each pair of coordinates of the nodes of the corresponding perpendicular plane. This decomposition method can be perfect applied to this structure because it is expected that the electrostatic potential does not present any abrupt variation in the any plane perpendicular to the quantization direction. For the application of this algorithm is only required to include the line **quantize_x{}**, **quantize_y{}** or **quantize_z{}** in the quantum section of the input file. In this tutorial the quantum calculations are decomposed in solutions over the growth direction (the z-axis) and, therefore, we use **quantize_z{}**.

Figure 4.13.4.5 presents an animation of the density of electrons obtained from 3D simulations at 111 nm under the surface (in the 2DEG region) as a function of the applied bias for gates with more complex geometry (square in Figure 1). Slices of this plot for the plane passing between both gates (y=0) can be specified in the input file and are very convenient for automatic extraction of the value where the depletion of electrons occurs. Figure 4.13.4.6 and Figure 4.13.4.7 show that in this case, the pinch-off voltage V_{gate} is around -1.20 and 1.30 V.

From an iterative process, accurate values of pinch-off can be extracted from 3D simulations as detailed in the paper from Chatzikyriakou et al. mentioned above, that we strongly recommend to be used.

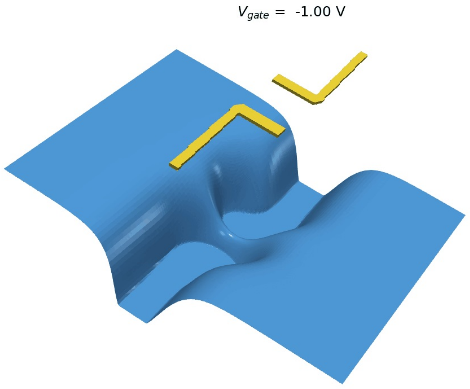


Figure 4.13.4.5: Density of electrons resulting from a 2D simulation as a negative bias is applied to the gates

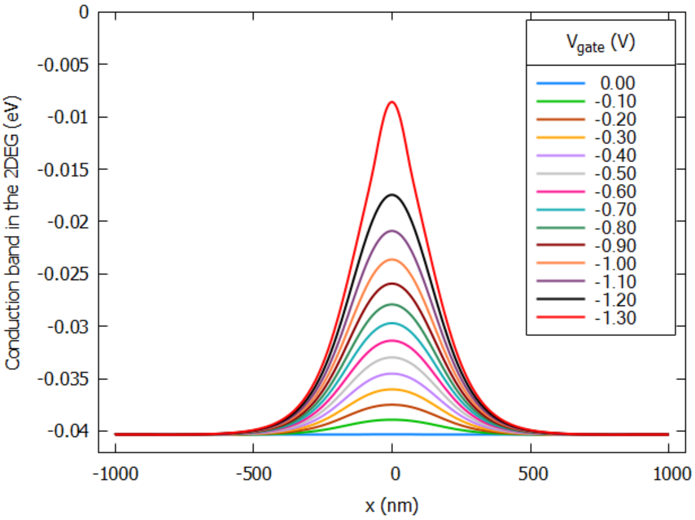


Figure 4.13.4.6: Slice of the computed conduction band in the 2DEG region at 111 nm under the surface as function of the applied bias to the gate. From the image the pinch-off voltage occurs around -1.30 and -1.20 V.

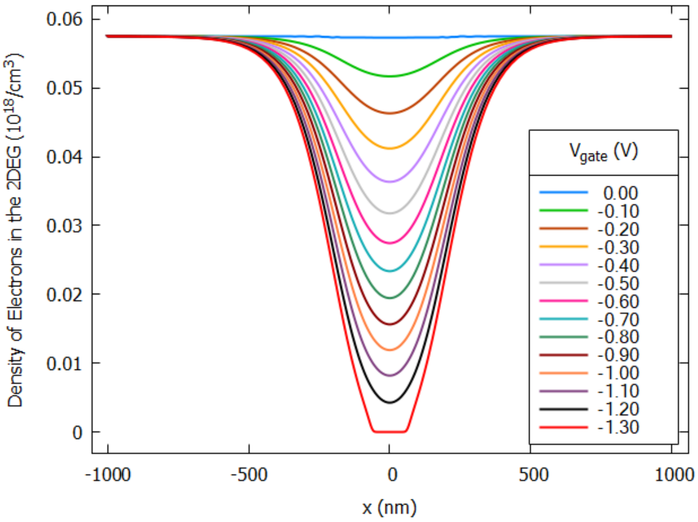


Figure 4.13.4.7: Slice of the computed density of electrons in the 2DEG region at 111 nm under the surface as function of the applied bias to the gate. From the image the pinch-off voltage occurs around -1.30 and -1.20 V.

This tutorial is based on the nextnano GmbH collaboration in the scope of the UltraFastNano Project aiming at development of the first Flying Electron Qubit at the picosecond scale, and it is funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 862683.

Last update: 2025/06/27

4.14 Transmission and Conductance (CBR method)

4.14.1 Transmission (CBR)

Header

Input Files:

- *transmission-barrier_1D_nnp.in*
- *transmission-step_1D_nnp.in*
- *transmission-quantum-well_1D_nnp.in*
- *transmission-double-barrier_Birner_JCEL_2009_1D_nnp.in*

Scope of the tutorial:

- Transmission coefficient

Relevant output files:

- *bias_00000\bandedge_Gamma.dat*
- *bias_00000\CBR\transmission_cbr_Gamma.dat*
- *bias_00000\Quantum\probabilities_shift_cbr_Gamma.dat*

Introduction

In this tutorial, we calculate the transmission coefficient $T(E)$ as a function of energy E. We consider the following pedagogical examples we learn in undergraduate quantum mechanics courses.

- Single potential barrier
- Step potential
- Quantum well
- Double potential barrier *[BirnerCBR2009]*

To calculate transmission spectra with *nextnano*++, we use Contact Block Reduction (CBR) method. This tutorial is an analog of here.

Single potential barrier

We first consider transmission through a finite quantum barrier. $10\ \mathrm{nm}$ barrier is located in a $50\ \mathrm{nm}$ sample. After running the input file *transmission-barrier_1D_nnp.in*, we obtain the following band edge profile. The barrier height is set to $E_{barrier} = 0.3$ eV.

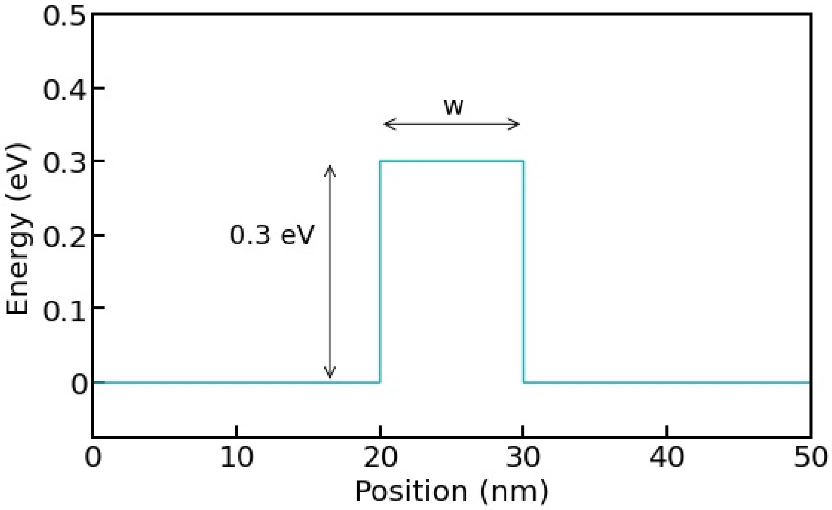
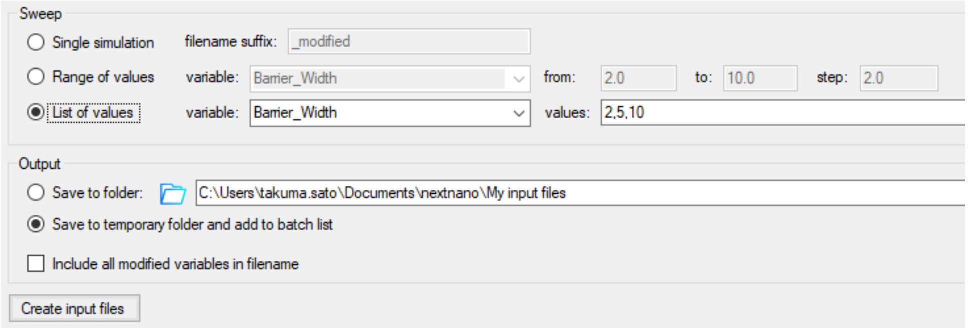


Figure 4.14.1.1: The conduction band edge profile (*bandedge_Gamma.dat*).

With *nextnano++*, one can calculate the transmission spectrum using the CBR method (*[BirnerCBR2009]*). The sample input file is generalized so that you can change the barrier width and alloy content (which determines the barrier height).

Here we look into the barrier width dependence. In *nextnanomat*, go to **'Template'** tab and select the input file. Then, you can select how to sweep the value at the bottom (`List of values`) and variable `Barrier_Width`. The list of values shows up automatically, as it is specified in the input file with the tag `ListOfValues`. Clicking the button *Create input files* generates multiple input files by sweeping variables. Please go to **'Simulation'** tab and run the simulation.



Sweep

○ Single simulation filename suffix: _modified

○ Range of values variable: Barrier_Width from: 2.0 to: 10.0 step: 2.0

● List of values variable: Barrier_Width values: 2,5,10

Output

○ Save to folder: C:\Users\takuma.sato\Documents\nextnano\My input files

● Save to temporary folder and add to batch list

☐ Include all modified variables in filename

Create input files

The result is written in *transmission_cbr_Gammma.dat*. The barrier width w affects the transmission coefficient as shown in Figure 4.14.1.2.

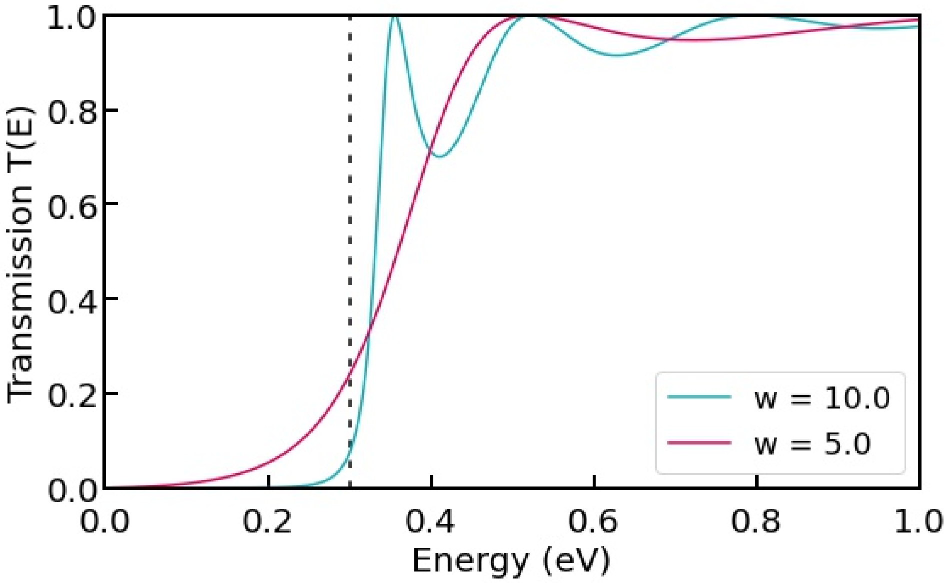


Figure 4.14.1.2: The transmission coefficient as a function of energy for different barrier width w (nm). The dashed line marks $E_{barrier}$.

Classical mechanics argues that the transmission is 0 below $E_{barrier}$ and abruptly increases to 1 at $E_{barrier}$. However, quantum mechanics allows electrons with energy below $E_{barrier}$ to go through the barrier. This effect becomes even larger when the barrier is thin ($w = 5$ nm in this example). Quantum mechanics also predicts a oscillatory behavior above $E_{barrier}$.

Step potential

For a step potential structure (*transmission-step_1D_nnp.in*) as shown in Figure 4.14.1.3 (a), the transmission of electrons with energy below $E_{barrier}$ is prohibited because the barrier is infinitely thick.

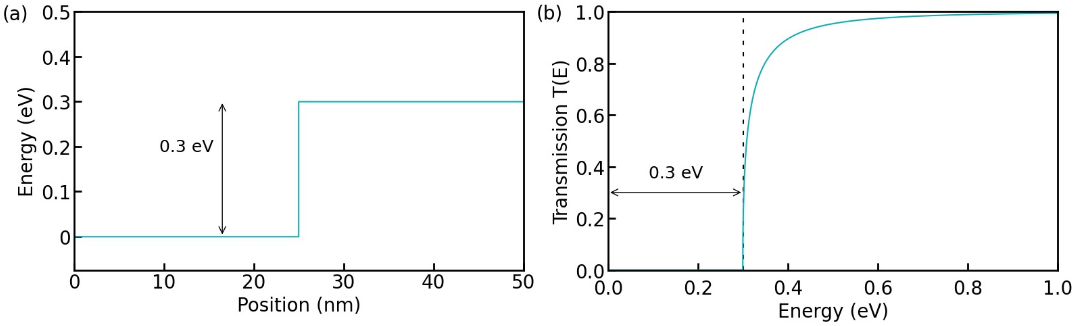


Figure 4.14.1.3: The conduction band edge profile is shown in (a). The transmission spectrum for a step potential is shown in (b). Transmission is only allowed above the step.

Quantum well

Similarly, a quantum well structure can be simulated with *transmission-quantum-well_1D_nnp.in*. The well width is $w = 10$ nm here. Again the transmission of electron within the barriers is impossible because the barrier is infinitely thick. Above 0 eV, the spectrum shows an oscillatory behavior.

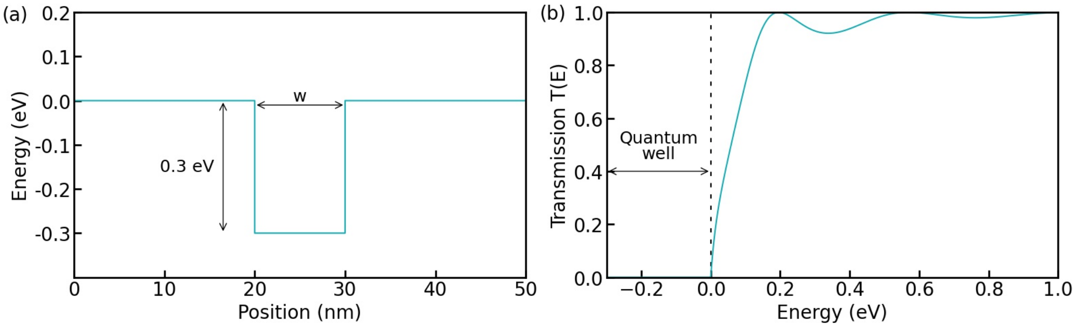


Figure 4.14.1.4: The conduction band edge profile is shown in (a). The transmission spectrum for a quantum well is shown in (b). The dashed line marks the top of the barrier.

Double potential barrier

Finally, we consider a double barrier structure with wall width 10 nm, *transmission-double-barrier_Birner_JCEL_2009_1D_nnp.in*. The barrier interval is 10 nm.

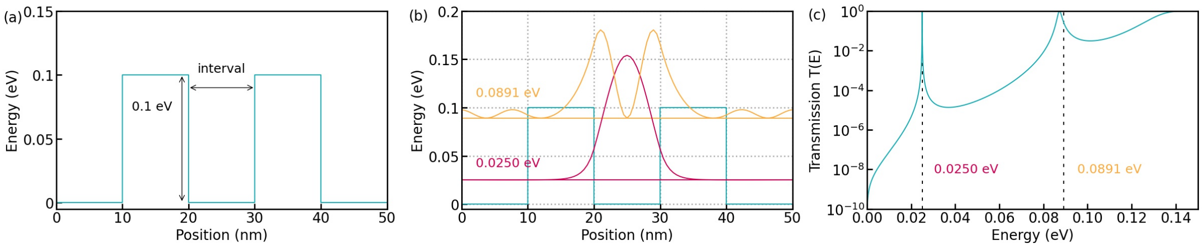


Figure 4.14.1.5: The conduction band edge profile is shown in (a). The probability distribution $|\psi(x)|^2$ of the two resonant modes. The transmission coefficient of the double barrier structure is shown in (c). The spectrum has two sharp peaks below the barrier height 0.1 eV, which corresponds to the resonant mode within the barriers.

This system has two resonant modes localized between the barriers. The band structure and wave functions are written in *bandedge_Gamma.dat* and *Quantumprobabilities_shift_cbr_Gamma.dat*, respectively. In the transmission spectrum, one can clearly see the sharp transmission at the energies of the resonant states in the quantum well. Note that the vertical axis is logarithmic scale.

A resonant tunneling diode (RTD) is an example of a device that exploits this δ-function-like behavior of transmission coefficient $T(E)$.

CBR efficiency assessment

Transmission_GaAs_AlAs_Birner_JCEL_2009_1D_Double_Barrier_nnp.in is used for this section. Figure 4 in *[BirnerCBR2009]* compares the transmission coefficient of a double barrier structure for different number of eigenstates considered in the CBR method. The following figure shows the result reproduced by *nextnano++* and demonstrates that the first resonant peak is accurately reproduced using incomplete set of eigenstates. The spectrum is not identical to the previous result because the barrier width here is 2 nm.

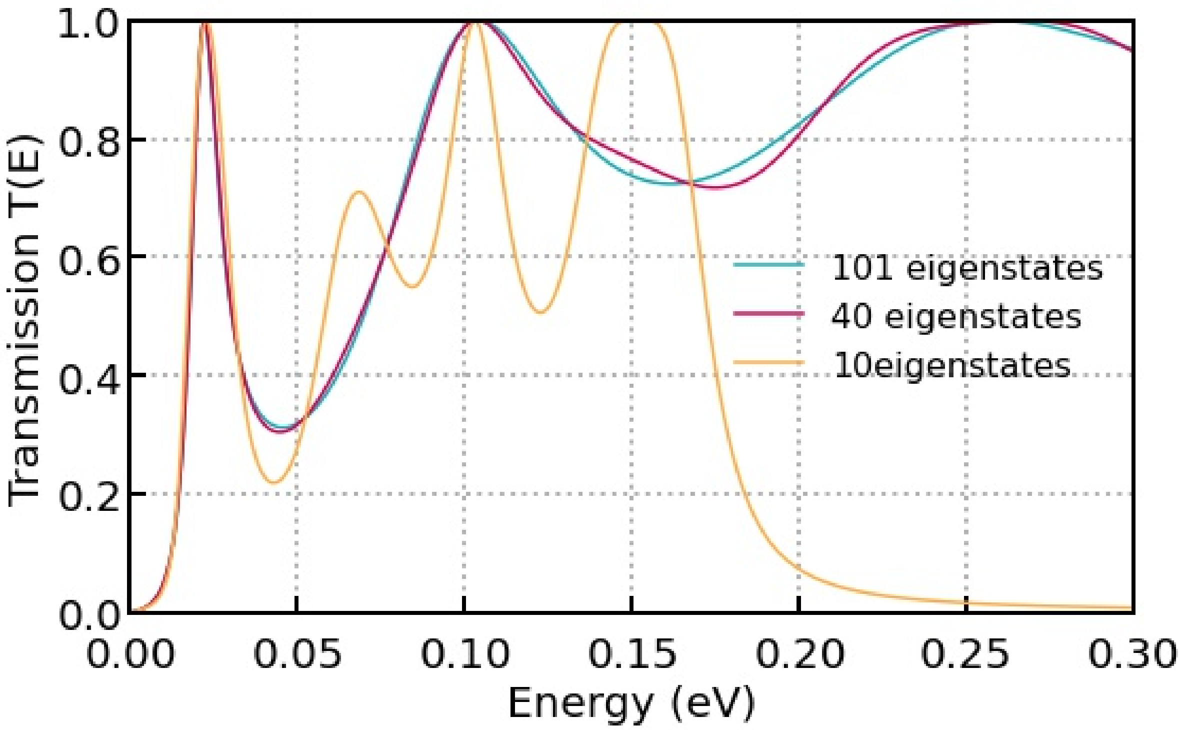


Figure 4.14.1.6: Transmission coefficient for three different CBR parameters. The blue curve is the result considering complete set of eigenstates, whereas violet and orange curves take into account only 40 $\%$ and 10 $\%$ of them, respectively.

See also for *3D case* for another CBR efficiency assessment.

Last update: nnnn/nn/nn

4.14.2 Landauer conductance and conductance quantization: from quantum wires to quantum point contacts

Header

Files for the tutorial located in *nextnano++\examples\transmission*

- *1D_GaAs_conductance-nnp.in* - simulations of 1D quantum wire in *nextnano++*
- *2D_transmission_QPC_nnp.in* - simulations of QPC in 2DEG
- *2D_transmission_QPC_potential_of_2DEG_1.fld* - numerically obtained energy profile of QPC
- *2D_transmission_QPC_potential_of_2DEG_2.fld* - numerically obtained energy profile of differently shaped QPC

Main adjustable parameters for 1D simulations (quantum wire):

- upper boundary for transmission energy - `%E_max`
- the barrier widths - `%Delta_x = %Barrier_max - %Barrier_min`
- the barrier heights - `%Barrier_Height`
- the temperature - `%Temperature`
- Fermi levels of left (xmin_contact < x < x_min) and right (x_max < x < xmax_contact) regions (leads) - `%Fermi_left` and `%Fermi_right`
- the effective mass of the electron - `%effective_mass`

Relevant output files of 1D simulations (quantum wire):

- *Results\BandEdges.dat* (energy profile)
- *Results\Transmission_cb_sg1_deg1.dat* (transmission)
- *Results\LocalDOS_sg1_deg1_Lead1.fld* and *Results\LocalDOS_sg1_deg1_Lead1.fld* (LDoS)
- *Results\IV_characteristics.dat* (currents)

Main adjustable parameters for 2D simulations (QPC):

- dimensions of the device - `$x_length` and `$y_length`
- grid spacing in x and y direction, `$grid_spacing`
- number of eigenvalues in the device and the leads - `$num_eigenstates_device` and `$num_eigenvalues_leads`
- the temperature - `$Temperature`
- energy range and resolution that the transmission will be computed - `$E_min`, `$E_max` and `$delta_energy`
- path of the file to be imported - `$pathPotentialFile`

Relevant output files of 2D simulations (QPC):

- *bias_00000\bandedges.fld* (energy profile)
- *Structure\contact.fld* (contacts)
- *bias_00000\CBR\transmission_sums_device_Gamma.dat* (transmission)

Introduction

Conductance, G, is the quantity which describes the relation between an electric current, J, and an applied voltage, V_{dc}, which causes this current. In this tutorial, we briefly review the analitical theory, which allows one to calculate the conductance, and compare it with the numerical approach implemented into the nextnano software. We discuss only the dc case with the main focus on the linear response regime where $J = GV_{\mathrm{dc}}$.

Unlike conductivity, which characterizes properties of a material, conductance describes a given sample. Therefore, geometry and size of the sample matter. We start below from an example of a quantum wire where the electric current is carried either by one (one-dimensional, 1D) or several (quasi-1D) propagating modes. Conductance of the quantum wire is described by the seminal Landauer theory. A simple introduction to the Landauer theory can be found in the book by S. Datta *[Datta]*, *section 2 "Conductance from Transmission"*.

The setup of the Landauer theory is shown in the upper panel of Figure 4.14.2.1. The device is connected via left and right ideal wires (ballistic conductors) to two leads with different chemical potentials. The current flows from the material with a larger chemical potential to that with a smaller one.

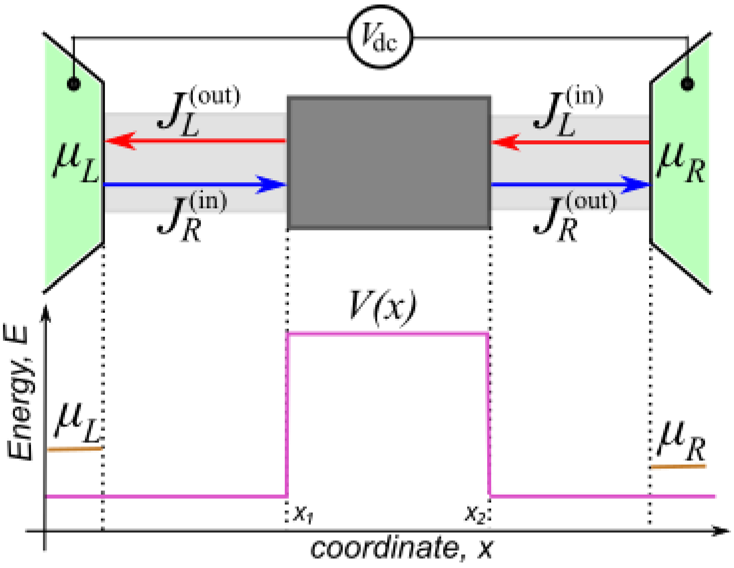


Figure 4.14.2.1: Upper panel: Landauer setup. Left and right leads (green regions) are connected to a semiconductor device (dark gray square) via connecting wires (light gray regions). Lower panel: chemical potentials of the leads (orange lines) and the energy of the potential barrier (magenta lines).

In the standard approach, the leads are two- or three-dimensional large conductors and contacts between the leads and the wires are reflectionless. This ensures that electrons supporting the current $J_{\mathrm{R}}^{(\mathrm{in})}$ are in equilibrium with the left lead and have the chemical potential μ_L. Similarly, the electrons supporting the current $J_{\mathrm{L}}^{(\mathrm{in})}$ are in equilibrium with the right lead and have the chemical potential μ_R.

Simulations of the current in 1D wires

Let us assume that all elements of the electric circuit are purely 1D, there is no temperature gradient, and the chemical potentials of the leads are shifted by the applied external voltage, $\mu_R - \mu_L = eV_{\mathrm{dc}}$.

⚠ **Attention**

The value of chemical potentials is not calculated in this tutorial but is set a kind of "artificially". Of course, this value must be in agreement with physics of a given material. For example, when the temperature (at $k_B = 1$) is smaller than the energy gap separating the conduction and valence bands, the chemical potential of an intrinsic unbiased semiconductor is close to the center of that gap, see e.g *section 3 The Fermi-Dirac Distribution* in *[Grahn]*.

Since the connecting wires are ballistic and the contacts are reflectionless, the backscattering of the electrons can occur ony inside the semiconductor device. We model this by including a potential scatterer (a square barrier) into the simulations. Hence, the scattering inside the device is elastic, the energy of the scattered electron is unchanged, and the electrons supporting the currents $J_{\mathrm{R,L}}^{(\mathrm{out})}$ are a mixture of the electrons with the chemical potentials $\mu_{R,L}$. The energy landscape of the device containing a square potential, $V(x < x_1) = V(x > x_2) = 0, V(x_1 < x < x_2) = V_0$, is shown in the lower panel of Figure 4.14.2.1. The electrons whose energy is small, $E < V_0$, can tunnel through the potential barrier. The electrons with large energies, $E > v_0$, can be reflected due to quantum effects. For the simple case of the rectangular barrier, the transmission in both cases is known:

$$\mathcal{T}(0 < E < V_0) = 1/\left(1 + [\kappa \sinh\left(\tilde{k}a\right)]^2\right), \tag{4.14.2.1}$$

$$\mathcal{T}(E > V_0) = 1/\Big(1 + [\kappa \sin\big(\tilde{k}a\big)]^2\Big); \tag{4.14.2.2}$$

Here $\tilde{k} = \sqrt{2m|V_0 - E|)|}/\hbar$, $\kappa = \sqrt{V_0^2/4E|V_0 - E|}$, m is the (effective) mass of the electron and $\hbar$ is the Planck constant. Transmission of the device is needed to calculate the current: The total current is the difference of currents flowing in opposite directions: $J = J_R^{(\mathrm{in})} - J_L^{(\mathrm{out})} = J_R^{(\mathrm{out})} - J_L^{(in)}$. Here, upper indices indicate whether a given current flows into or from the device. The Landauer formula allows one to express J via $\mathcal{T}$. In the purely 1D setup, the current reads:

$$J = 2e \int \frac{dk}{2\pi} v(k)\mathcal{T}(k)\Big(f_L(k) - f_R(k)\Big) = \frac{2e}{h} \int dE\, \mathcal{T}(E)\Big(f_L(E) - f_R(E)\Big); \tag{4.14.2.3}$$

where e and k are the electron charge and its wave-vector, respectively. The electrons in the left/right leads are described by the Fermi-Dirac distribution functions, $f_{L,R}$. The second equality in (4.14.2.3) has been obtained after changing the integration variable from the electron wave-vector to its energy.

If $V_0 = 0$, i.e. $\mathcal{T} = 1$, a simple calculation yields $J = G_0 V_{\mathrm{dc}}$ where $G_0 = 2e^2/h$ is the quantum of the conductance. The nextnano software reproduces this result with a very high accuracy, see Figure 4.14.2.2. The numerical simulations presented in this tutorial were done by using *Contact Block Reduction method [CBR]*, see also a tutorial on the CBR method in *nextnano*^{3}.

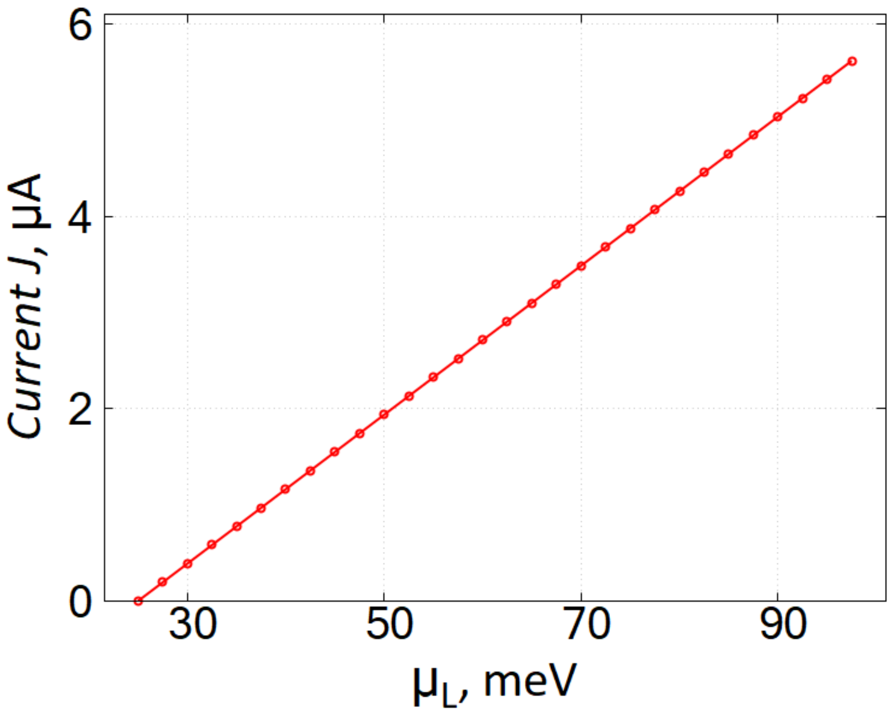


Figure 4.14.2.2: Numerically calculated IV-characteristics of a ballistic 1D conductor, $V_0 = 0$. We have chosen GaAs as the material of the conductor with the total length 32 nm at $\mu_R = 25\mathrm{meV}$; the temperature was set to $T = 50\mathrm{mK}$. Note that these parameters has no influence on the universal slope of the IV straight line which is equal to G_0. For chosen parameters of the numerical solver and the numerical integration procedure (cf. the sample input file), the difference between the numerically calculated slope and G_0 is $\simeq 4\%$.

The users of the nextnano software should pay attention that regions, which are called "leads" in the CBR-based sample input files, are actually interfaces between the devices and the connecting wires. These interfaces have minimal width of the space discretization. In the toy model which we discuss the chemical potential of each interface is equal to that of the corresponding lead. Such a simplification of the Landauer setup in natural in the CBR method. One may refer to the interfaces between the device and the connecting wires as "CBR-leads". An example of the CBR-leads is shown below for the case of the two-dimensional (2D) device.

Figure 4.14.2.3 and Figure 4.14.2.4 shows the transmission and the IV characteristics of the device which contains the square scattering potential of width 30 nm with $V_0 = 100\mathrm{meV}$.

Since transmission of the device is exponentially small at energies below 0.1 eV, the current become nonzero only

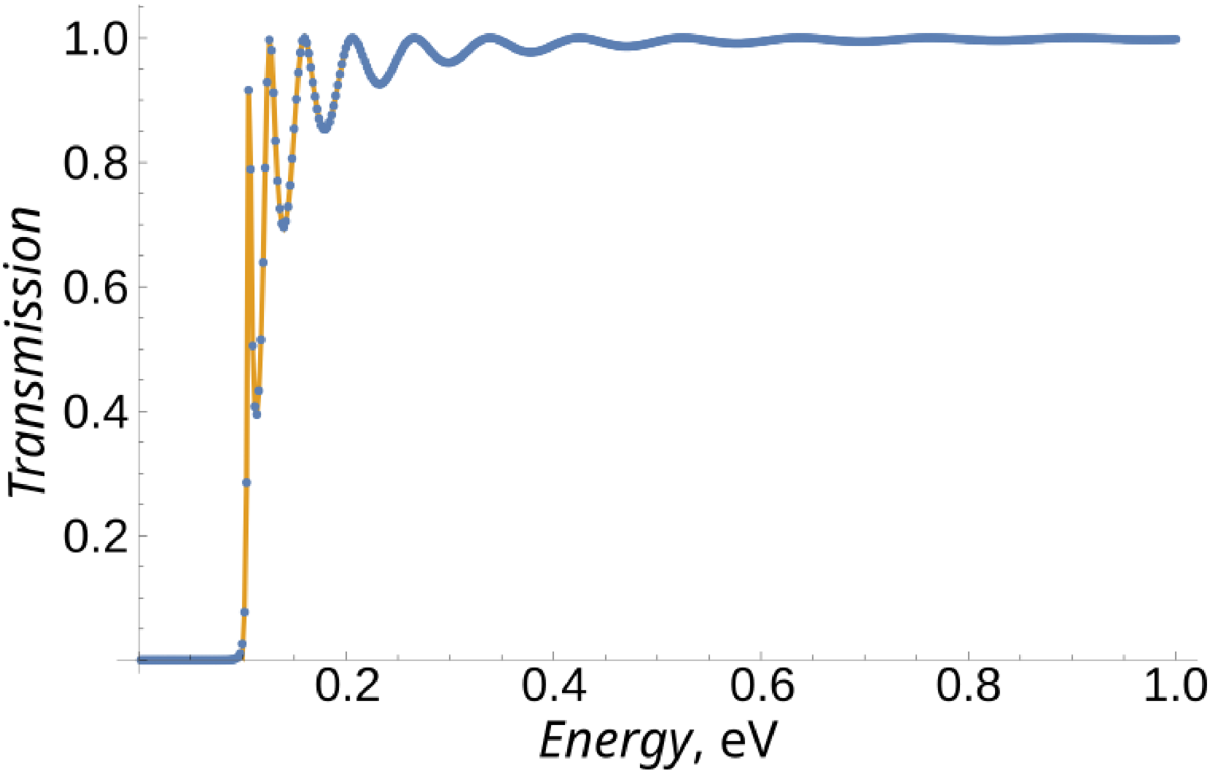


Figure 4.14.2.3: Transmission of a 1D conductor with $V_0 = 100\mathrm{meV}$ and width 30nm. Orange line and blue dots shows the exact analytical answer, Eqs. (4.14.2.1) and (4.14.2.2), and CBR calculations, respectively.

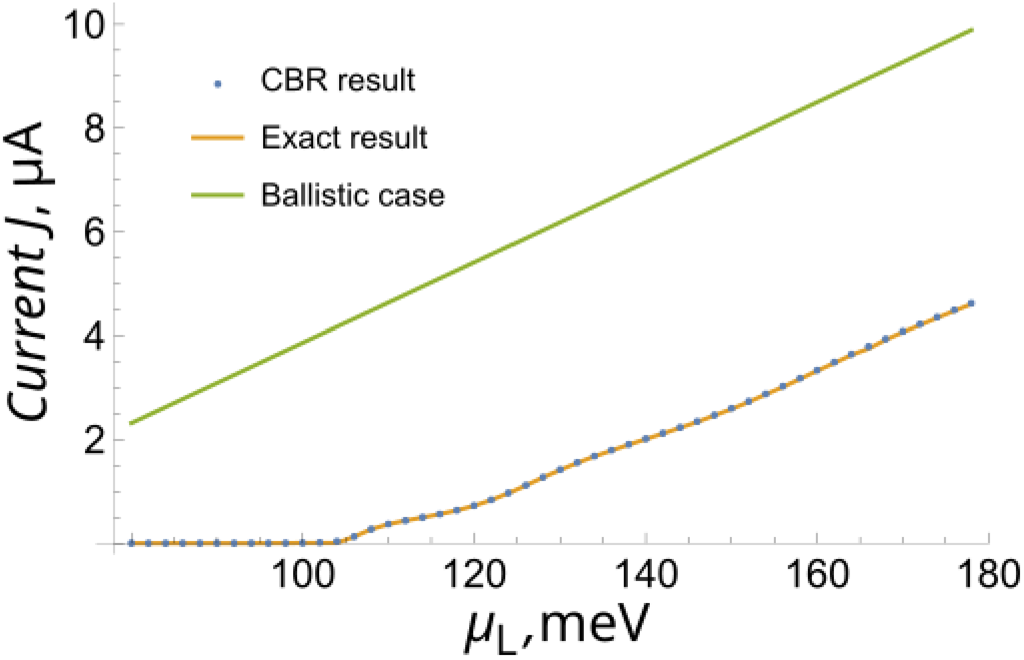


Figure 4.14.2.4: IV-characteristics of a 1D conductor with $V_0 = 100\mathrm{meV}$ and width 30 nm at $\mu_R = 50\mathrm{meV}$. Other parameters are the same as in Figure 4.14.2.2. Orange line and blue dots shows the exact analytical answer [obtained by using Eqs. (4.14.2.1) and (4.14.2.2)], and CBR calculations. Green line exemlifies the ballistic law $J = G_0 V_{\mathrm{dc}}$.

at $\mu_L > 0.1\text{eV}$ and, after some transient, the IV characteristics becomes again linear with the slope being close to G_0 with accuracy of several percents.

Exercise

- **Calculate numerically transmission and current through a biased potential which linearly**
  increases from the value $V(x_1) = V_1$ to $V(x_2) = V_2$ with $V_1 < V2$. Compare the result of simulations with that for the unbiased barrier.
- **Repeat the simulations for the inverted biased barrier:** $V(x_1) = V_2$ **to** $V(x_2) = V_1$
  keeping all other parameters the same as in the previous task. Do transmission and current change under spatial invertion of the barrier? Explain your answer.

Transmission and conductance of QPC, conductance quantization

The CBR method implemented in nextnano software allows one also to calculate conductance of more complicated semiconductor devices, for example, of a quantum point contact (QPC). QPC in a 2D electron gas (2DEG) can be created in a semiconductor heretostructure by a *voltage applied to a top gate*. In this case, the potential energy in the plane of the 2DEG can be obtained from the numerical *solution of the Poisson equation*. An example of such a profile of the potential energy is shown in Figure 4.14.2.5.

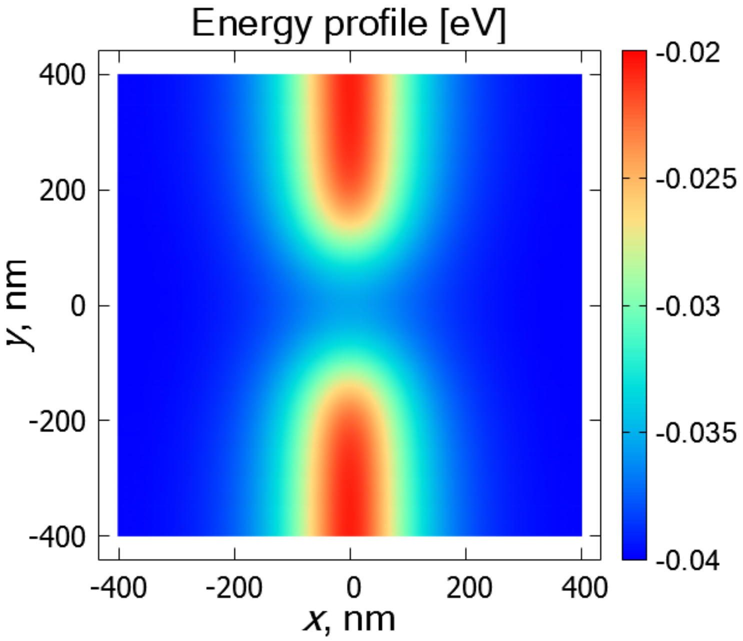


Figure 4.14.2.5: An example of the numerically obtained energy profile for a QPC in the plane of the 2D electron gas. The simulations were done for the 2D electron gas in GaAs at temperature 100mK.

The energy profile can be imported into the nextnano procedure which calculates transmission, e.g., from left to right boarder of the sample. The left CBR-lead used in this tutorial is illustrated in Figure 4.14.2.6. The right CBR-lead is attached at $x = 400\text{nm}$.

Numerically calculated energy dependence of the QPC transmission is shown in Figure 4.14.2.7. Temperature corrections to the transmission (due to the temperature-dependent gap) and to the conductance (due to the thermal broadening of the distribution functions) are negligibly small in the sub-Kelvin range ($\ll 1\text{K}$) and we neglect them in this tutorial.

The lowest modes with the energy $< -35\text{meV}$ are localized near the CBR-leads and do not contribute to transmission. A small plateau of $\mathcal{T} \simeq 1$ at $-34.5\text{meV} < E < -34\text{meV}$ corresponds to the energies where the first delocalized mode of the device yields its maximum contribution to the transmission. The second (slightly smeared) plateau, $\mathcal{T} \simeq 2$, signals that the second delocalized mode yields its maximum contribution to the transmission, etc.

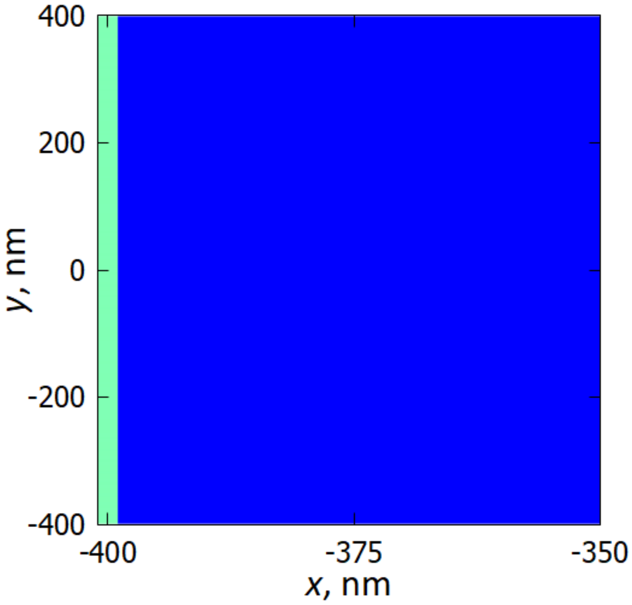


Figure 4.14.2.6: Illustration of how the left CBR-lead (light green region) is attached to the device (blue region). The width of the lead along x-axis is equal to the step of the space discretization. The width of the lead along y-axis has been chosen to be equal to the width of the device.

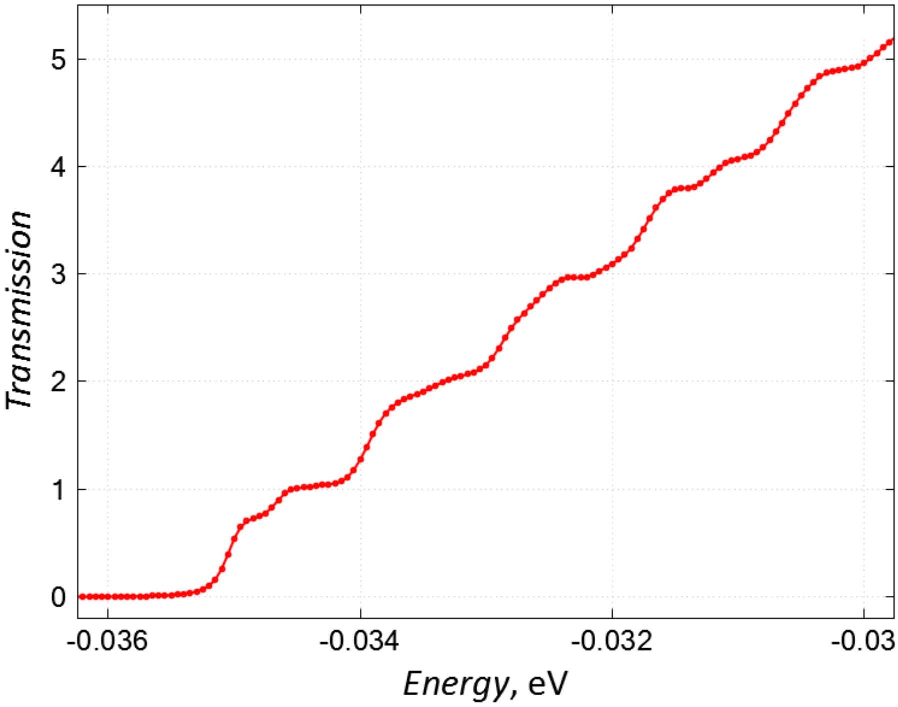


Figure 4.14.2.7: Numerically calculated energy dependence of the transmission via the QPC which is presented in Figure 4.14.2.5. The bottom of the conduction band, E_0, of the gated 2DEG is located at $\simeq -40\mathrm{meV}$. Hence, E_0 is the origin of the energy for this example.

The example of the gate-induced QPC is 2D and requires 2D simulations. However, the second equation in (4.14.2.3) still can be used. It suggests that, if temperature and V_{dc} are extremely small, then linear conductance is proportional to transmission: $G_{\mathrm{QPC}} = G_0\mathcal{T}(\mu)$. Negative values of the chemical potential, μ, of the gated semiconductor structure are related to the choice of the origin, which is explained above. To conclude, we note that plateaux in the energy dependent transmission correspond to those in the conductance which are called in the literature "conductance quantization".

Exercises

- **The above example was based on the QPC geometry taken from the file *2D_transmission_QPC_2D_potential-v1_of_2DEG.fld*.**

  File *2D_transmission_QPC_2D_potential-v2_of_2DEG.fld* contains another QPC geometry which results from a different shape of the top gate electrode. Use this file with the alternated QPC geometry, process it with the help of the nextnano input file, and calculate the QPC transmission.

  > **Attention**
  >
  > The minimal energy, above which transmission is finite (not zero), depends on the QPC geometry and on the applied gate voltage. Hence, one has to find an appropriate energy range where the plateaux of the quantized conductance are well visible.

- Compare the energy profile and the energy dependent transmission for the both shapes of the QPC.
- **Note that the second QPC shape does not possess "left ↔ right" inversion symmetry (inversion**

  with respect to the line $x = 0$). Compare transmissions from the left to right CBR leads with that from the right to left leads. Are they equal? Explain your observation.

This tutorial also exists for *nextnano³*.

Last update: 2024/07/17

4.14.3 Electron Flying Qubit

Input Files:

- *EPJQT2022_2D_TCW_nnp.in* (used for Figure 13 in *[Edlbauer2022]*)
- *EPJQT2022_2D_ABI_nnp.in* (used for Figure 13 in *[Edlbauer2022]*)
- *EPJQT2022_1D_slice_TCW_nnp.in* (used for Figure 13.c in *[Edlbauer2022]*)

In this tutorial, we discuss multi-terminal electron transport in various nanodevices. As an example, we focus on so-called electron flying qubits, which are solid-state counterparts of the quantum optics devices. Basic building blocks of these qubits are the following semiconductor-based nanodevices:

- Tunneling-coupled wires, TCW - the electronic counterpart of the optical beam splitter, see Figure 4.14.3.1;
- Aharonov-Bohm interferometer, ABI - the electronic counterpart of the optical interferometer, see the central region of Figure 4.14.3.2;
- Circuits containing these elements connected in a series, see Figure 4.14.3.2.

Left rectangular regions in Figure 4.14.3.1 and Figure 4.14.3.2 (with numbers 1 and 2) are incoming leads, where the electron can be injected into the nanodevice. We will assume that it is injected into the lead 1. Right rectangular regions (with numbers 3 and 4) are outgoing leads, where the electron can be detected after propagating through the entire nanodevice. The functionality of the electron flying qubits requires a reflection-free propagation of the electron. If the electron is reflected and returns to one of the incoming leads, a part of the quantum information is lost. The important task of numerical simulations is to identify regimes where reflection is reduced as much as possible.

The interior part of the nanodevices is assumed to be made from 2D GaAs-based semiconductor and includes regions with different electrostatic potentials and applied gate voltages that govern the energy profile through which the electron propagates. Colors in Figure 4.14.3.1 and Figure 4.14.3.2 reflect the strength of the electrostatic potential in different parts of the device, ranging from 0 eV (dark blue color) up to ≫ 1 eV (dark red color). All building blocks of the electron flying qubit can be realized in experiments with the help of properly tuned gated regions.

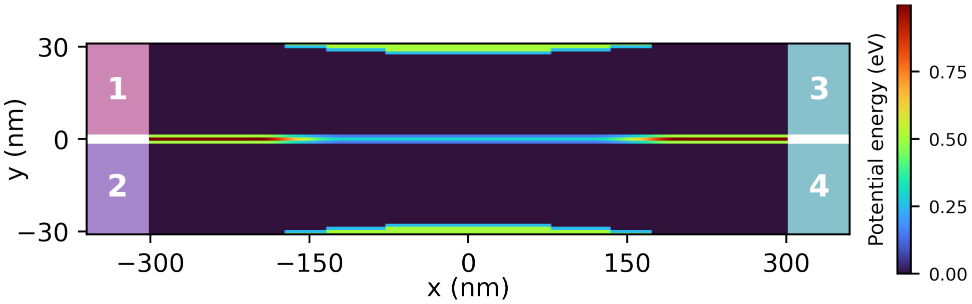


Figure 4.14.3.1: Geometry and potential landscape of TCW connected to four terminals (external leads marked by white numbers). Red and light blue separation regions denote impenetrable (very high with the height V_∞ = 10 eV) and penetrable (tunneling with the height V_T) potential barriers, respectively. Green regions mark those parts of the device where the gate voltages 0.5 eV and V_g are applied.

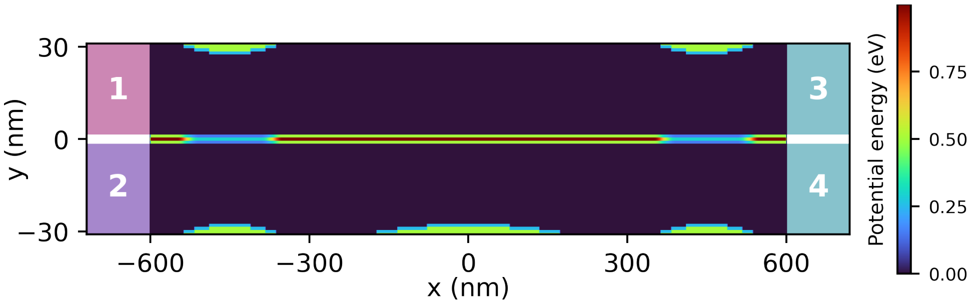


Figure 4.14.3.2: Geometry and potential of a circuit containing two TCWs and one ABI, also connected to four terminals. The additional barrier around x=0 in the lower path yields the electrostatic Aharonov-Bohm effect.

Let us first discuss transport in TCW. The horizontal line in Figure 4.14.3.1 shows a potential barrier separating two paths, along which the electron can move towards the outgoing leads. Red parts of the barrier are impenetrable for the electron while the electron can tunnel through the light blue segment. The latter is precisely the region where the quantum interference between the upper and lower paths takes place. Having experienced the interference, the electron wave function is split between the separated upper and lower paths. As a result, there is some probability to detect the electron in the outgoing leads 3 or 4, which depends both on the electron energy and on the parameters of the nanodevice, including the height of the tunneling barrier.

The TCW-ABI-TCW device shown in Figure 4.14.3.2 consists of two TCW (left and right outer) regions and the electrostatically induced ABI (central) region. In addition to the interference in the TCWs, the interference is influenced by the asymmetric gating in ABI: The electron trajectories traversing the lower and upper paths in ABI, which are separated by the impenetrable potential barrier, require different geometric phases governed by this asymmetric gating. This phase changes the interference and the transmmition through the entire device.

The nextnano software allows one to calculate the partial local density of states, Figure 4.14.3.3, and the transmission from the lead 1 to the leads 3 and 4, Figure 4.14.3.4 and Figure 4.14.3.5, in both, TCW and TCW-ABI-TCW, devices. The theoretical background involves the numerical solution of the Schrödinger equation by using the Contact Block Reduction method.

The partial local density of states, pLDoS, represents the probability of finding the propagating electron (that was injected with an energy E at the lead 1) at a certain position. The coordinate dependence of pLDoS illustrates how the electron with a given energy propagates through the device. The energy dependent transmission, $T_{ij}(E)$, is determined by the probability for the electron which is injected into lead i to reach lead j. Readers can find more information on these quantities in one of standard textbooks.

nextnano simulations of the pLDoS and of the transmission are discussed in detail our review *[Edlbauer2022]*. which presents the progress of the EU Ultrafastnano project. Let us emphasize here that these simulations are valuable tools to identify the parameter range where the reflection of the propagating electron, either to the lead no. 1 or to the lead no. 2, is minimized and, simultaneously, there is a pronounced manifestation of the quantum interference. Hence, one can find an optimal basic configuration for the realization of the electron flying qubit. Such a preliminary optimization saves a lot of experimental efforts and can substantially accelerate the overall progress.

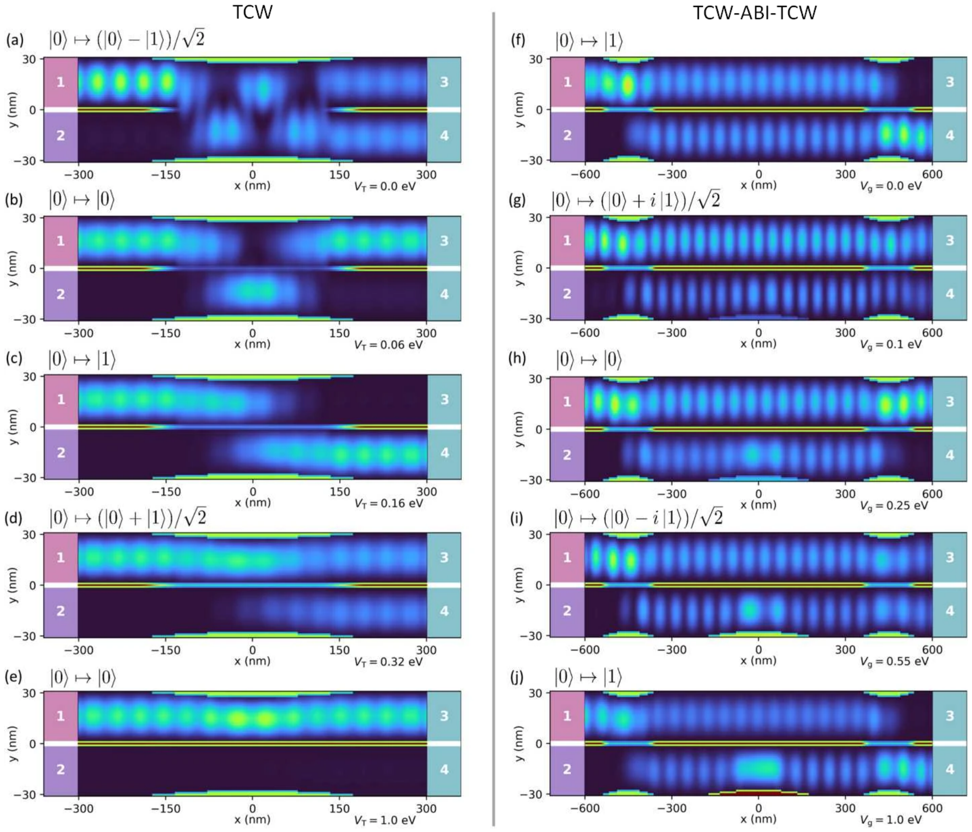


Figure 4.14.3.3: nextnano simulations of the electron partial local density of states in TCW [panels (a-e)] and the TCW - ABI - TCW [panels (f-j)] nanodevices. Both devices are connected to four terminals (marked by white numbers). The background shows the potential landscape defined by the voltage on the surface gates. The electron with a given energy (E = 9.2 meV for TCW and E = 7.5 meV for TCW-ABI-TCW) is always injected into the upper incoming channel from lead 1. The states at the output leads are indicated at the top of each plot, with the 0 and 1 qubit states corresponding to the densities at output leads 3 and 4. Panels (a-e): the pLDoS in TCW for increasing the tunneling barrier voltage (described by V_T). Panels (f-j): the pLDoS in TCW-ABI-TCW for increasing voltage on a side gate of the bottom path (described by V_g).

To conclude we note that this tutorial exemplifies the simulations done for a simple toy-model describing physics of the nanodevices. Nevertheless, the nextnano software can be used to simulate more realistic geometries whose potential profile can be obtained from electrostatic simulations. The restriction to 2D GaAs-based semiconductor materials is also not crucial, since input files can be easily adapted, e.g., for Si-based ones.

This tutorial is based on the nextnano GmbH collaboration in the scope of the UltraFastNano Project aiming at development of the first Flying Electron Qubit at the picosecond scale, and it is funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 862683.

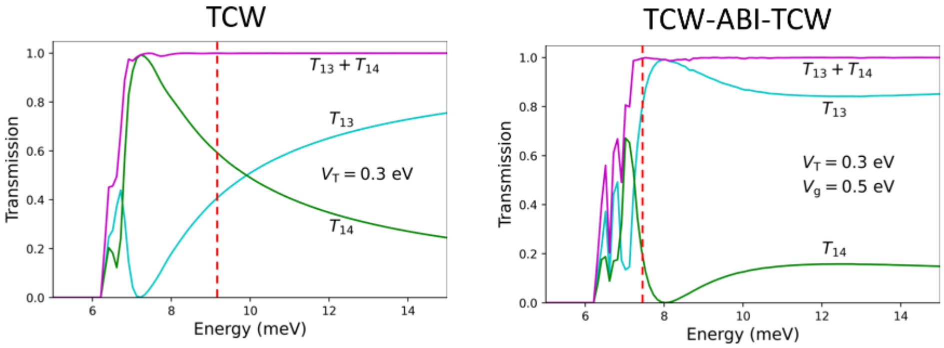


Figure 4.14.3.4: Energy-dependent transmission of the electron from the lead no. 1 into the leads no. 3 (T_{13}) and no. 4 (T_{14}). Red dashed lines mark some electron energies where the reflection is almost absent, T_{13} + $T_{14} \simeq 1$ (E = 9.2 meV for TCW and 7.5 meV for TCW-ABI-TCW).

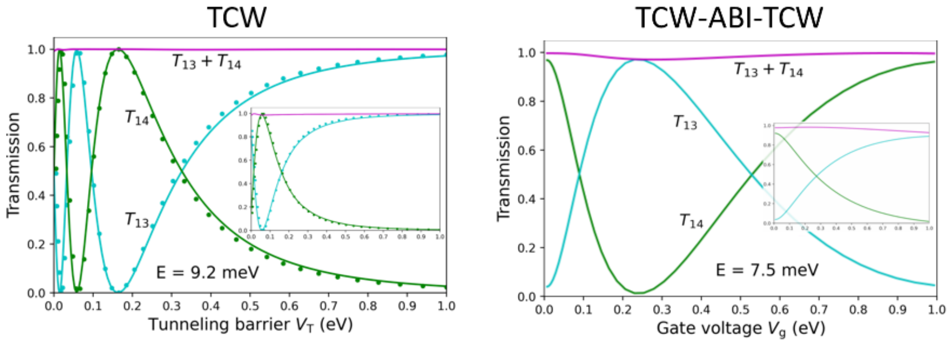


Figure 4.14.3.5: Almost reflectionless transmission of the electron with fixed energy as a function of V_T (TCW) and V_g (TCW-ABI-TCW). Dots in the left panel correspond to the semi-phenomenological theory supplied by the 1D simulation of the spectrum at the center of the device, x = 0. Insets: The same dependence as in the main figures but for devices with half-length, where the accessible number of quantum oscillations is much smaller.

Last update: 2025/06/27

4.14.4 — DEV — Efficient method for the calculation of ballistic quantum transport - The CBR method (2D example)

Attention

This tutorial is under construction

Header

Input Files:

- *Transmission_CBR_Mamaluy_JAP_2003_2D_nnp.in*
- *Transmission_CBR_Mamaluy_JAP_2003_2D_holes_nnp.in*

Scope of the tutorial:

-

Main adjustable parameters in the input file:

- parameter

Relevant output files:

- *bias_00000\bandedges.fld*
- *bias_00000\Quantum\probabilities_shift_device_Gamma.fld*
- *bias_00000\Quantum\probabilities_shift_lead_X_Gamma.dat*
- *bias_00000\CBR\transmission_device_Gamma.dat*

Introduction

In this tutorial, we apply the Contact Block Reduction (CBR) method to a Aharonov-Bohm-type structure with a large barrier in the middle of the device. This tutorial is based on *[MamaluyCBR2003]* and *[BirnerCBR2009]*. The input file *Transmission_CBR_Mamaluy_JAP_2003_2D_holes_nnp.in* simulates holes instead of electrons.

Simulation setup

First, we look into the structure of the device. Figure 4.14.4.1 shows the calculated conduction band edge of the device.

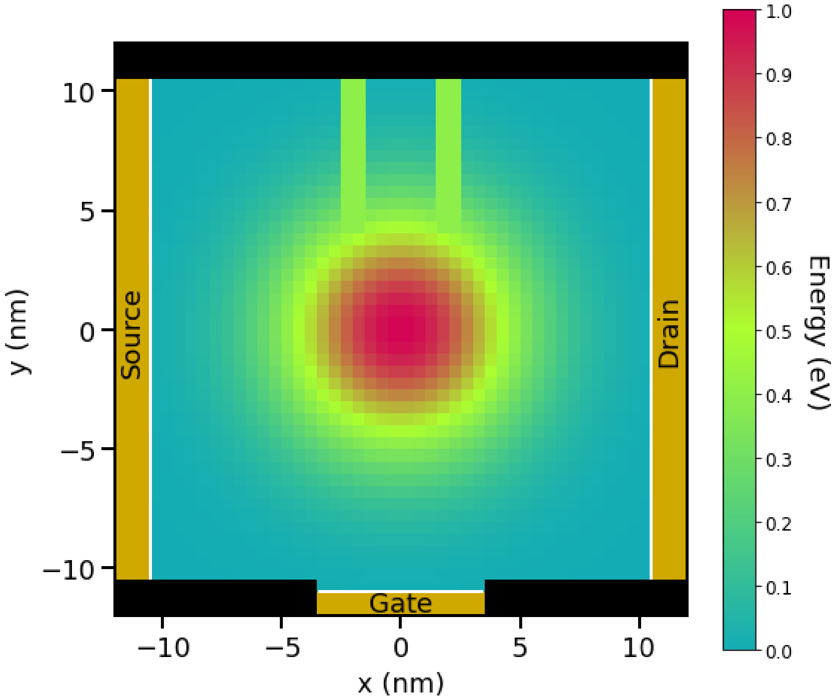


Figure 4.14.4.1: The calculated conduction band edge. The center of the device ($(x, y) = (0, 0)$ (nm)) is AlAs and the energy is 1.0 (eV). The vicinity of the edges of the device is GaAs and the energy is 0 (eV). The double potential barrier is set so that the energy is equivalent to 0.4 (eV). Note that the blacked out areas are set up with barriers of infinite height. *bias_00000\bandedges.fld*

The image below shows the 3-dimenional conduction band edge. Note that the height of the infinite potential barriers are set to 2.0 (eV) for convenience.

This device has some features.

The device consists of three contacts that are called ‘source’, ‘gate’ and ‘drain’. They also have leads adjacent to them, indicated by white lines in Figure 4.14.4.1

In the middle of the device a potential barrier of two-dimensional Gaussian shape effectively expels the electrons from the center. The energy profile is given by

$$E_c = E_{c,0} \exp\left(-\frac{x^2 + y^2}{a^2}\right),$$

where $E_{c,0} = 1.0$ (eV) so that the maximum height of the Gaussian barrier becomes 1.0 (eV) at the center of the device. In this tutorial, $a = 5$ (nm).

In the upper part of the device, a thin tunneling double barrier is present and the height is 0.4 (eV).

These conduction band profiles are achieved by adjusting the `database{ }` as below.

```
database{
    binary_zb{
        name = "GaAs"
        conduction_bands{ Gamma{ mass = 0.3 bandgap = 0 }   # effective mass 0.3m0
        valence_bands{
            bandoffset = 0.0   # artificially shifted so that (GaAs conduction␣
↪bandedge) = 0.0 eV
```

(continues on next page)

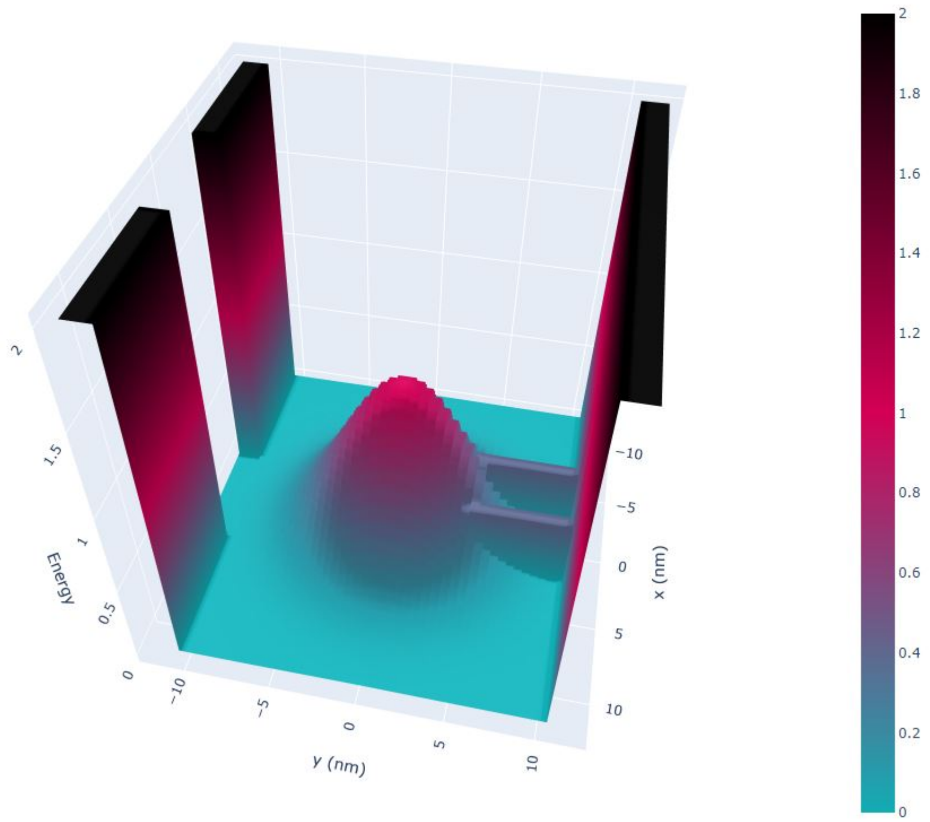


Figure 4.14.4.2: Potential landscape

(continued from previous page)

```
            delta_SO = 0.0
        }
    }

    binary_zb{
        name = "AlAs"
        conduction_bands{ Gamma{ mass = 0.3 bandgap = 0 }   # effective mass 0.3m0
        valence_bands{
            bandoffset = 1.00   # artificially shifted so that (AlAs conduction␣
↪bandedge) = 1.0 eV

            delta_SO = 0.0
        }
    }

    bowing_zb{
        name = "AlGaAs_Bowing_x"
        valence = III_V
        conduction_bands{ Gamma{ mass = 0.0 bandgap = 0.000 } }   # bowing is␣
↪switched off for this simulation
        valence_bands{
            bandoffset = 0.000   # artificially shifted so that (Al0.4Ga0.6As␣
↪conduction bandedge) = 0.4 eV
            delta_SO = 0
        }
    }
```

(continues on next page)

(continued from previous page)

```
    bowing_zb{
        name = "AlGaAs_Bowing_1_x"
        valence = III_V
        conduction_bands{ Gamma{ mass = 0.0 bandgap = 0.000 } }   # bowing is␣
↪switched off for this simulation
        valence_bands{
            bandoffset = 0.000   # artificially shifted so that (Al0.4Ga0.6As␣
↪conduction bandedge) = 0.4 eV
            delta_SO = 0
        }
    }

    ternary2_zb {
        name        = "Al(x)Ga(1-x)As"
        valence     = III_V
        binary_x    = AlAs
        binary_1_x  = GaAs
        bowing_x    = AlGaAs_Bowing_x
        bowing_1_x  = AlGaAs_Bowing_1_x
    }
}
```

In addition, the infinite potential barriers surround the device as shown as blacked out areas in Figure 4.14.4.1.

The effective electron mass is assumed to be constant throughout the device and equal to $0.3m_0$.

We set the boundary conditions as follows:

- If it is at the boundary, and if it is in contact to a lead, a **Neumann** boundary condition is set.
- If it is at the boundary, and if it is **not** in contact to a lead, a **Dirichlet** boundary condition is set.

```
quantum{
    region{
        name          = "device"
        no_density    = yes
        x             = [ $x_contact_left, $x_contact_right ]
        y             = [ $y_contact_bottom, $y_quantum_top ]
        boundary{ x = neumann y = neumann }   # boundary condition for CBR = Neumann␣
↪for propagation direction & Dirichlet for perpendicular direction.
        Gamma{ num_ev = $num_eigenstates_device cutoff = 4.0 }
        output_wavefunctions{
            probabilities   = yes
            max_num         = $num_output_wavefunctions_device
            in_one_file     = no
        }
    }

    # lead 1 is a 1D line (x = $contact_left).
    region{
        name          = "lead_1"
        no_density    = yes
        x             = [ $x_contact_left, $x_contact_left ]
        y             = [ $y_inf_barrier_bottom, $y_inf_barrier_top ]
        boundary{ x = neumann y = dirichlet }
        Gamma{ num_ev = $num_eigenstates_lead1 cutoff = 4.0 }
        output_wavefunctions{ probabilities = yes max_num = $num_eigenstates_lead1 }
```

(continues on next page)

(continued from previous page)

```
    }

    # lead 2 is a 1D line (y = $y_contact_bottom).
    region{
        name          = "lead_2"
        no_density    = yes
        x             = [ $bottom_contact_left, $bottom_contact_right ]
        y             = [ $y_contact_bottom, $y_contact_bottom ]
        boundary{ x = dirichlet y = neumann }
        Gamma{ num_ev = $num_eigenstates_lead2 cutoff = 4.0 }
        output_wavefunctions{ probabilities = yes max_num = $num_eigenstates_lead2 }
    }

    # lead 3 is a 1D line (x = $x_contact_right).
    region{
        name          = "lead_3"
        no_density    = yes
        x             = [ $x_contact_right, $x_contact_right ]
        y             = [ $y_inf_barrier_bottom, $y_inf_barrier_top ]
        boundary{ x = neumann y = dirichlet }
        Gamma{ num_ev = $num_eigenstates_lead3 cutoff = 4.0 }
        output_wavefunctions{ probabilities = yes max_num = $num_eigenstates_lead3 }
    }

    cbr{
        name = "device"
        lead{ name = "lead_1" }
        lead{ name = "lead_2" }
        lead{ name = "lead_3" }
        delta_energy    = 0.0005   # energy resolution
        min_energy      = 0.0      # minimum energy
        max_energy      = 0.5      # maximum energy
    }
}
```

Note the following points.

- To consistent with the results of *[MamaluyCBR2003]* and *[BirnerCBR2009]*, the quantum region is extended (1 grid point outside along x and y direction), respect to the device dimensions in the papers.

This is attributed to the difference in the way boundary conditions are set in *nextnano++*. The details are described below in attention.

- To set dirichlet boundary conditions at the top and bottom of the device that are no contact with leads, the quantum region is extended to the infinite potential barrier (1 grid point **further** outside along y direction), respect to the device dimensions in the papers.

The difference in the device dimensions from in *[MamaluyCBR2003]* and *[BirnerCBR2009]* arise from the reasons above.

For each energy E (energy step is equal to 0.0005) where the transmission coefficient $T(E)$ has to be calculated, a matrix of size 95 $\times$ 95 has to be inverted. The size of 95 is determined by the sum of the number of grid points in each lead that are in contact to the device.

- Lead 1 (Source): 41 grid points
- Lead 2 (Gate): 13 grid points
- Lead 3 (Drain): 41 grid points
    - in total: 95 grid points

  - The total CPU time for calculation of the transmission $T(E)$ in this example is about 5 seconds for 303 eigenstates.

Note that we do not take into account the increase in grid points due to the increase in the gate length.

Transmission

Figure 4.14.4.3 shows the calculated transmission coefficients of the various lead combinations T_{12}, T_{23}, and T_{13}. For the orange-dashed lines 100 % (1681 of 1681) of all eigenvectors were used whereas for the light-blue lines only 18 % (303 of 1681) had to be calculated. You can see that reducing the eigenvectors to 18 % or even 7 % (118 of 1681) of the total eigenvectors does not result in significant changes in $T(E)$, especially at lower energies. This means that one does not have to calculate all eigenvalues of the device Hamiltonian which grossly reduces CPU time. A small percentage of eigenvalues suffices for $T(E)$ in relevant energy range of interest.

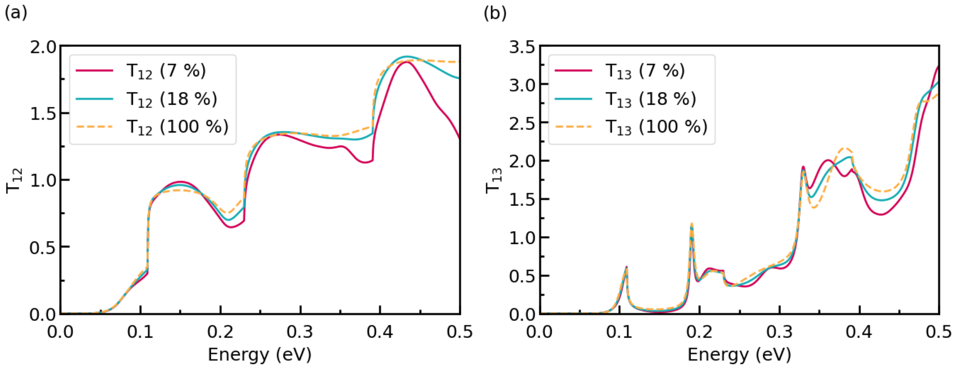


Figure 4.14.4.3: The transmission coefficient $T(E)$ of a 2D sample with 3 leads. T_{12} in (a), whereas T_{13} in (b). *bias_00000\CBR\transmission_device_Gamma.dat*

The *nextnano++* results differ slightly from the *[MamaluyCBR2003]* and *[BirnerCBR2009]*.

Reasons:

  - The potential energy profile in the device and in the leads is not identical, as well as the dimensions of the barriers.
  - The dimensions of the device are not identical as explained (See the attention below for further information).

Therefore, the eigenenergies and the wave functions in the device, and in the leads differ slightly which explains the small deviations.

The 16th eigenstate is a resonance state of the lower transmission path.

- 1st resonance: the 16th eigenstate: 0.119 (eV)

The square of the 16th wave function with the conduction band is shown below. (*bias_00000\Quantum\probabilities_shift_device_Gamma.fld*)

Note that the square of the wave function is rescaled so that you can see the shape clearly.

The 26th eigenstate and 29th eigenstate are resonance states of the double barrier.

- 1st resonance:
  - the 26th eigenstate: 0.177 (eV) (delocalized)
  - the 29th eigenstate: 0.193 (eV) (more localized)
- 2nd resonance:
  - the 56th eigenstate: 0.311 (eV) (delocalized)
  - the 59th eigenstate: 0.328 (eV) (more localized)
  - the 61th eigenstate: 0.336 (eV) (delocalized)

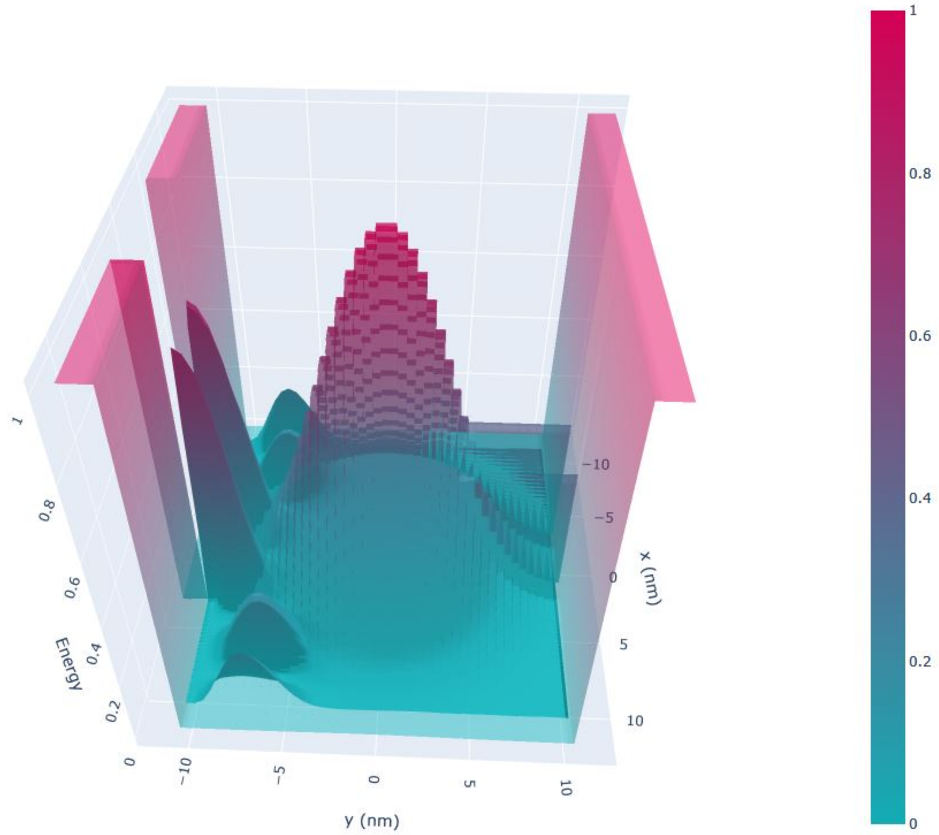


Figure 4.14.4.4: The 16th eigenstate

- the 63th eigenstate: 0.347 (eV) (delocalized)
- the 64th eigenstate: 0.352 (eV) (more localized)

TO BE CHECKED

The follow figure shows the square of the wave function of the 26th eigenstate with the conduction band. (*bias_00000\Quantum\probabilities_shift_device_Gamma.fld*) You can clearly see that it is a resonance state of the double barrier and corresponds to the second peak in the light-blue transmission curve T_{13} from source to draian around 190 (meV).

Note that the square of the wave function is rescaled so that you can see the shape clearly.

Lead modes

Figure 4.14.4.6 shows the lead modes of the gate, and the source (which is identical to the drain). In the transmission curve $T_{12}(E) = T_{23}(E)$, the transmisson shows a step-like behavior which is related to the energies of lead 2 (‘gate’).

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

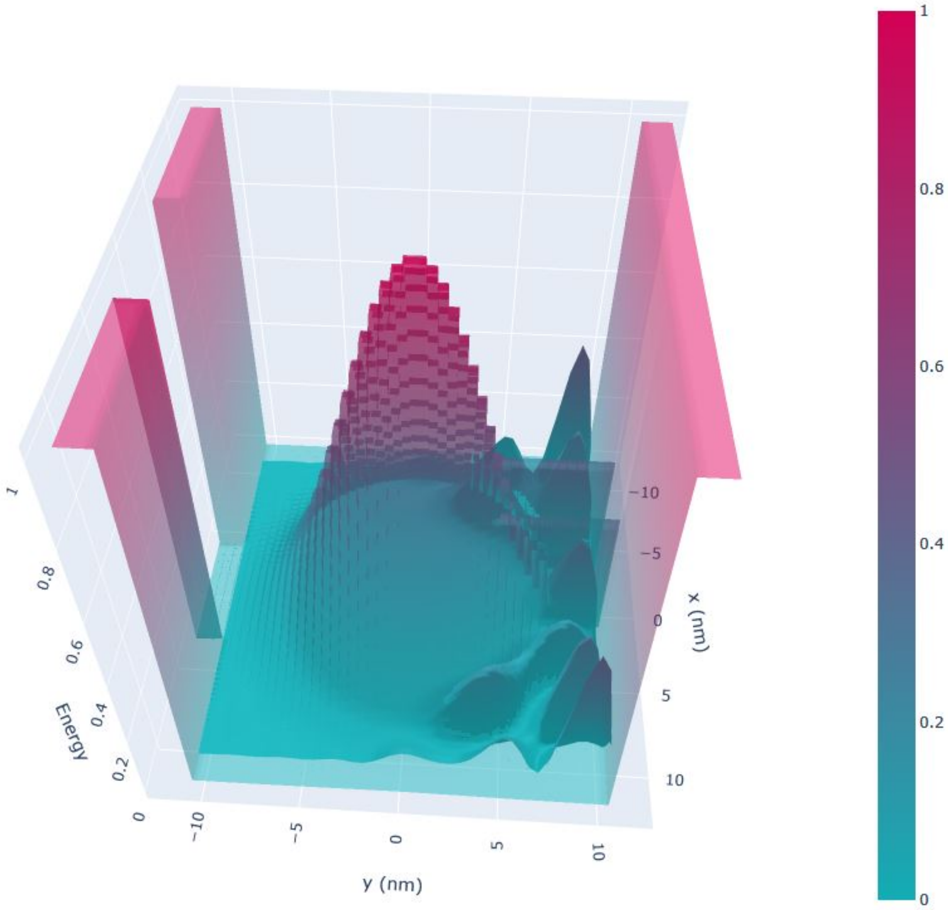


Figure 4.14.4.5: The 26th eigenstate

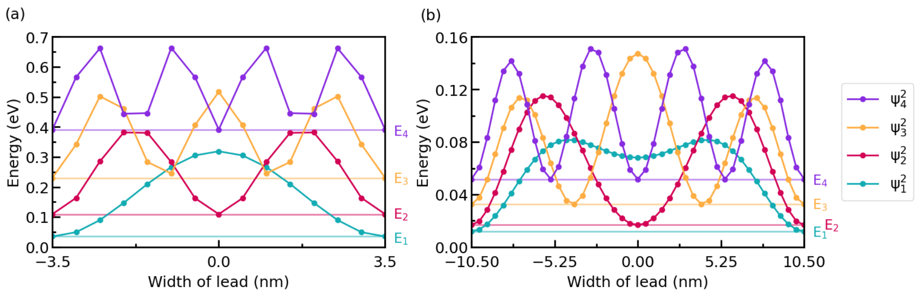


Figure 4.14.4.6: The lead modes of lead 2 (‘gate’) are shown in (a), whereas the lead modes of lead 1, 3 (‘source’, ‘drain’) are shown in (b). *bias_00000\Quantum\probabilities_shift_lead_X_Gamma.dat*

4.14.5 Transmission through a nanowire (CBR)

Header

Input Files:

- *transmission-nanowire_GaAs_3D_nnp.in*

Scope of the tutorial:

- transmission

We apply the Contact Block Reduction (CBR) method to a simple GaAs nanowire of cuboidal shape.

System

We consider a GaAs cuboidal tube of dimensions 10 nm × 10 nm × 20 nm. Two leads of 10 nm × 10 nm each are attached to the edge of the device. The grid spacing is 1 nm in all directions. The effective electron mass is assumed to be constant throughout the device and equal to 0.067 m_0.

Input file

To simulate 3D (or 2D) system with CBR method in *nextnano*++ correctly, The quantum regions have to be appropriately specified in the input file.

```
quantum{
    region{
        name = "lead_1"
        x    = [-6,6]
        y    = [-6,6]
        z    = [-0.1,0.1]
        boundary{ x=dirichlet y=dirichlet z=cbr }
        Gamma{ num_ev = $num_eigenstates_device }
    }
}
```

The perpendicular directions, i.e. x- and y-directions, of the system are elongated by one grid due to the treatment of edge points in nextano++. Since the simulation is three dimensional, the lead region specified here has to be two dimensional. The number ±0.1 is chosen to be smaller than the grid spacing, so that the region "lead_1" becomes a 2D sheet. CBR boundary condition has to be imposed in the propagation direction, i.e. z-direction, whereas Dirichlet boundary condition is set for perpendicular directions.

```
cbr{
    name = "device"
    lead{ name = "lead_1" }
    lead{ name = "lead_2" }
    delta_energy   = $delta_energy
    abs_min_energy = $E_min
    abs_max_energy = $E_max
}
```

Here we specify the device region and leads attached to the device. The program calculates transmission through the region "device", from "lead_1" to "lead_2". The resolution, minimum and maximum of the energy axis can be also tuned here.

CBR efficiency assessment

The biggest advantage of the CBR method is that it can correctly predict the spectrum without calculating all eigenmodes of the 3D device. That means that, for low energies, one can significantly reduce the simulation load for the calculation of transmission spectrum *Birner2009*. To demonstrate it we perform three different simulations, sweeping the number of modes considered in the calculation. In the input file, the variable `$CBR_case` switches the number of eigenmodes.

```
$CBR_case   = 1                                            # (ListOfValues:1,2,3)

$CBR_light  = iszero($CBR_case-1)
$CBR_medium = iszero($CBR_case-2)
$CBR_heavy  = iszero($CBR_case-3)

#if $CBR_light $num_eigenstates_device    =   200            #  5.6% of all device␣
↪modes
#if $CBR_light $num_eigenstates_lead      =    30            # 17.8% of all lead modes

#if $CBR_medium $num_eigenstates_device   =   400            # 11.3% of all device␣
↪modes
#if $CBR_medium $num_eigenstates_lead     =    50            # 30.0% of all lead modes

#if $CBR_heavy $num_eigenstates_device    =   600            # 16.9% of all device␣
↪modes
#if $CBR_heavy $num_eigenstates_lead      =    80            # 47.3% of all lead modes
```

Figure 4.14.5.1 shows the calculated transmission coefficient as a function of energy. The result of *nextnano³* is shown for reference. Arrows indicate the cutoff energies, namely the eigenenergy of the highest device mode considered in each simulation. The transmission coefficient drops when the energy exceeds the cutoff value. In the low energy, however, it is sufficient to calculate only a part of all eigenfunctions of the device Hamiltonian. Lower cutoff energy means lower dimension of matrices and vectors in the simulation, e.g. Eq.(36) in *Birner2009*, which reduces the calculation load. For example, a simulation performed at nextnano office took

- 42 sec for $CBR_case=1 (black)
- 3 min 14 sec for $CBR_case=2 (blue)
- 11min 17 sec for $CBR_case=3 (red)

Lead modes

The step-like increase of the transmission coefficient is attributed to the discrete energy levels of the lead modes. Let us have a close look at the first few steps. We can see that T(E) increases by integers.

The lead mode probability distribution $|\psi(x,y)|^2$ and corresponding eigenvalues are exported to the following files:

> `~\Quantum\wf_probabilities_lead_1_Gamma_0000.fld` `~\Quantum\wf_energy_spectrum_lead_1_Gamma_0000.dat`

To see the energy eigenvalues, it is convenient to switch to `Show Output File as Text` (marked yellow).

Once the energy reaches 76 meV, the first lead mode energy is reached and then this mode transmits perfectly, giving a transmission of 1.

As can be seen from `\Quantum\wf_probabilities_lead_1_Gamma_0000.fld`, the second and third lead mode states are degenerate due to the symmetry of the lead cross-section. Thus they have the same energy 190 meV. Consequently, the spectrum increases by 2 at the energy of 190 meV. In this fashion, the step-like behavior of the transmission coefficient is explained by lead eigenmodes.

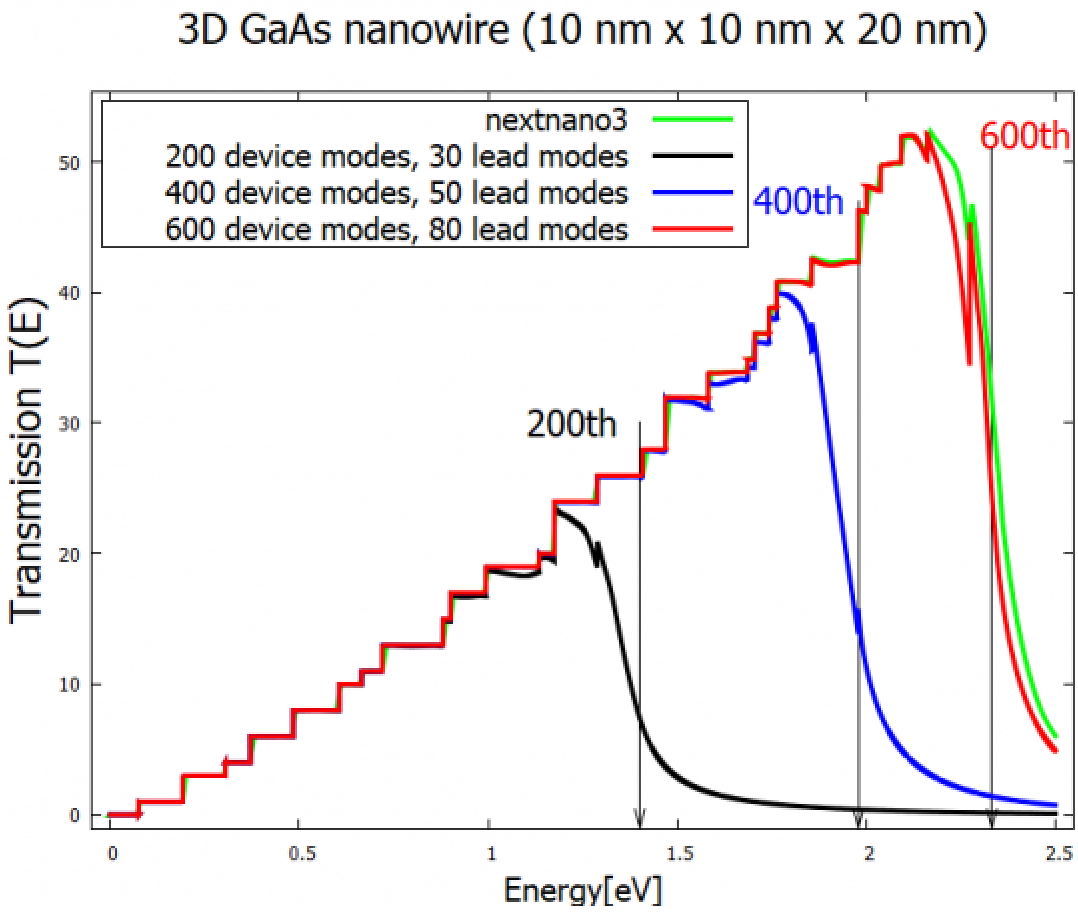


Figure 4.14.5.1: Transmission coefficient of a GaAs 3D nanowire simulated with three different CBR parameters. The *nextnano³* result is shown for reference. Arrows indicate the cutoff energies, namely the eigenenergy of the highest device eigenmode considered in each simulation.

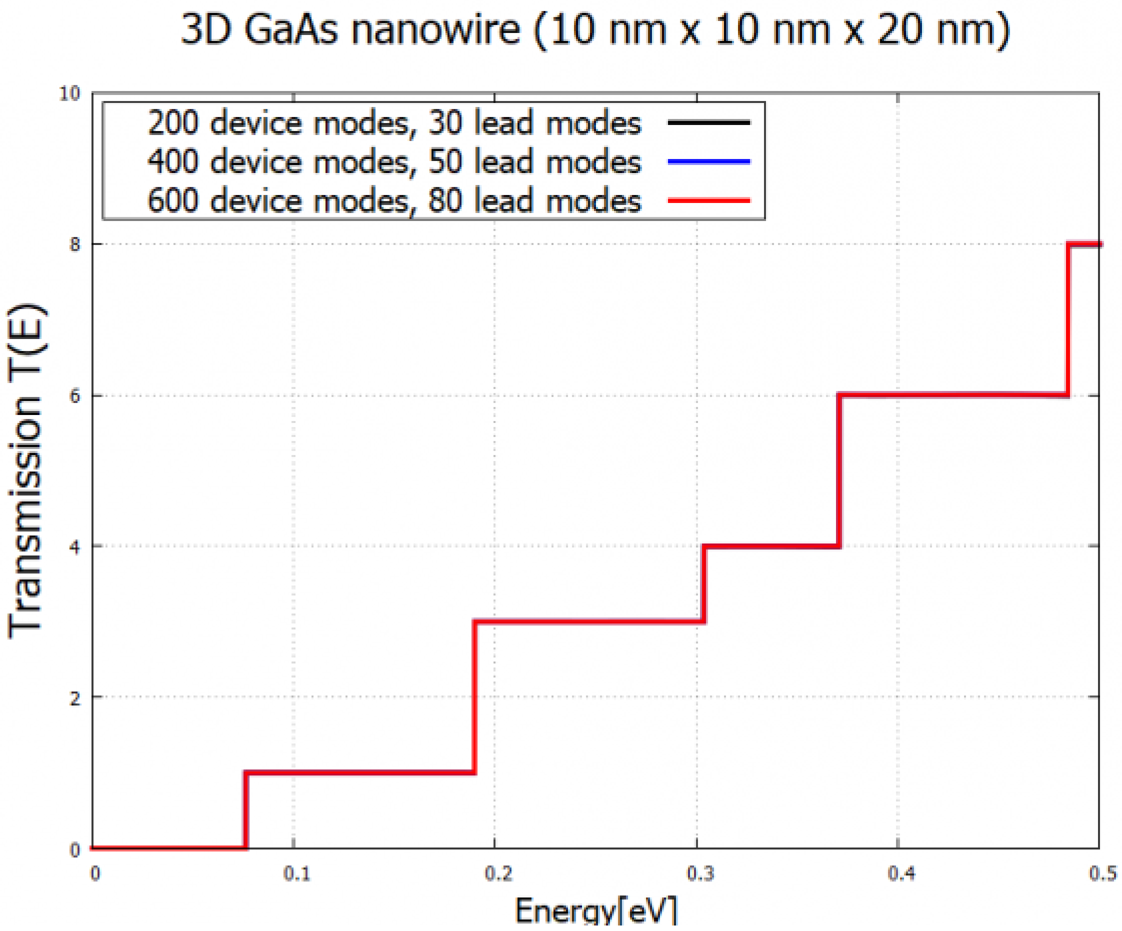


Figure 4.14.5.2: Zoom into the first few steps of T(E). The transmission increases by integer at the eigenenergies of the lead.

C:\Users\takuma.sato\Documents\nextnano\Output\Transmission_CBRtutorial_3Dnanowire_nnp_wider\bias_000_000\Quantum\wf_energy_spectrum_lead_1_Gamma_0000.dat

```
no.   Energy[eV]
1     0.076207147688
2     0.189824695700
3     0.189824695700
4     0.303442243812
5     0.370564113883
6     0.370564113883
7     0.484181661995
8     0.484181661995
9     0.606108309575
10    0.606108309575
```

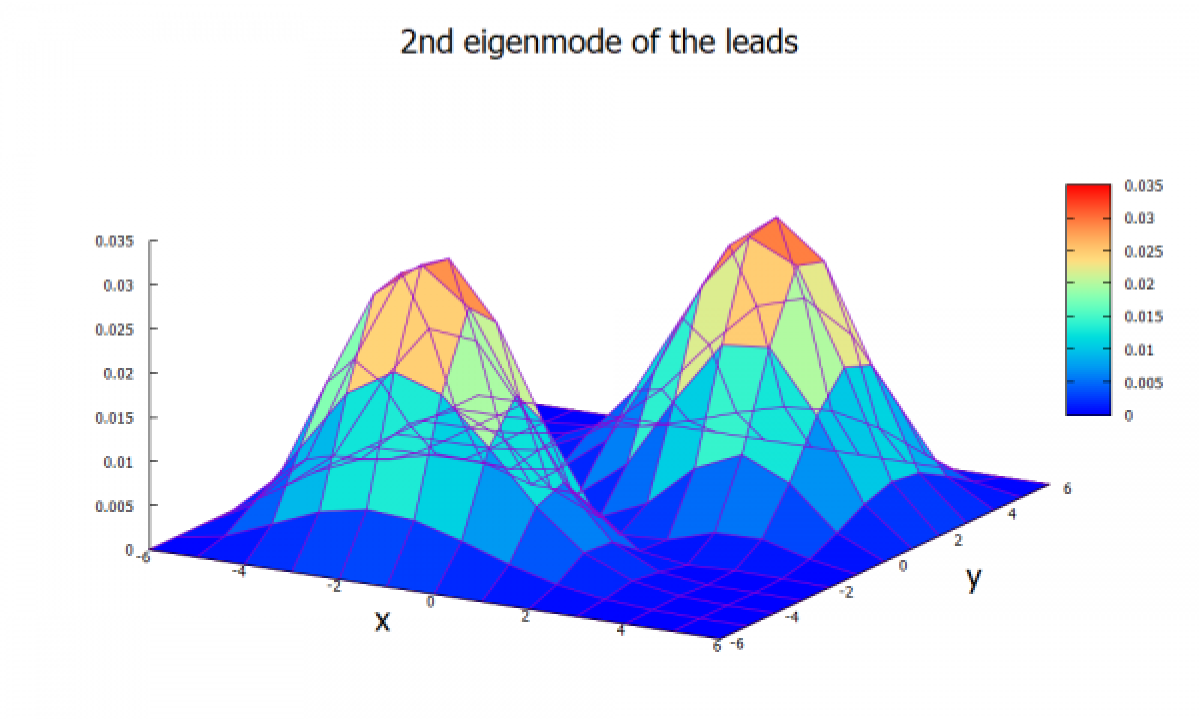


Figure 4.14.5.3: The probability distribution $|\psi(x, y)|^2$ of the 2nd lead mode.

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.14.6 Conductance of a quantum point contact (gated two-dimensional electron gas)

Attention

A tutorial on computing the conductance using CBR method can be found *here*

Related Files:

- *3D_conductance_in_top_gated_2DEG_nnp.in* - simulation of the potential in 2DEG
- *3D_conductance_in_top_gated_2DEG.py* - generates all plots
- *3D_conductance_in_top_gated_2DEG_verification.py* - does not generate conductance
- *3D_conductance_in_top_gated_2DEG_without_plot.py* - generates only conductance
- *3D_conductance_in_top_gated_2DEG_exercise.py* - semiclassical and quantum calculations (exercise)
- *3D_conductance_in_top_gated_2DEG.ipynb* - Jupyter Notebook for practicing the tutorial

The Python scripts and the Jupyter Notebook file are available on our GitHub

Scope of the tutorial:

- computing electrostatic potential using *nextnano++*
- interfacing *nextnano++* with Kwant, for computing the conductance between two leads

Main adjustable parameters in the input file:

- calculation with or without Schrödinger - `$solve_quantum`
- depth of the slice of the 2DEG region - `$slice_in_2DEG` (see lines 76 and 77)
- the widths of the gates - `$gate_width`
- the gap beetween the gates - `$gap_length`
- lowest bias on the top gate - `$top_gate_bias_min`
- highest bias on the top gate - `$top_gate_bias_max`
- number os bias sweeps of the top gate - `$top_gate_steps`
- bias of the bottom gate - `$bottom_gate_bias`

Relevant output files:

- *bias_xxxxx\bandedges_2d_2deg_slice.fld* (potential energy profile - semiclassical case)
- *bias_xxxxx\Quantum\energy_subbands_quantum_region_Gamma_2d_2deg_slice.fld* (potential energy profile - self-consistent quantum case)
- *bias_xxxxx\density_electron_1d_section_line_x_center.dat* (density of electrons in the growth direction)

Simulated Structure

Figure 4.14.6.1 presents the simulated structure, where a two-dimensional electron gas (2DEG) is formed at the interface of the AlGaAs and GaAs (the substrate) materials. The electron density in the 2DEG is enhanced by doping the region of the AlGaAs with n-type impurities only in the part close to the surface.

A GaAs layer over the n-AlGaAs region acts simply as a cap of the device. On the top of the surface metallic gates are deposited and can present different geometries. We will choose the gates in the Figure 4.14.6.2 as QPCs, to which negative bias will be applied in order to deplete electrons at the center of the 2DEG region. Although these gates pursue one of the simplest geometries, the method here described can also be used for gates with more complex shapes.

The dopant and surface charges concentrations used in this simulation are realistic, and were obtained by the calibration method described in *[Chatzikyriakou_PhysRevResearch_2022]*

The Simulation

The main objective of this tutorial is to simulate the conductance between two leads in the 2DEG region as a function of the applied bias in the gates deposited at the top of the structure.

Initially we will use *nextnano++* to obtain the conduction band in the device changing the applied bias to the top gate in the range of -1.5 V and 0.0 V. The applied bias to the bottom gate will be kept constant (-1.1V), through the whole set of simulations. For this first phase of this tutorial, we will use the input file: *3D_conductance_in_top_gated_2DEG_nnp.in*.

In order to obtain the trasmission coefficients between two leads in the 2DEG, we will import a slice of the conduction band in this region into the software Kwant, using the Python script: *3D_conductance_in_top_gated_2DEG.py*

Kwant is an open-source tool that performs numerical calculations on tight-binding models. For the installation of Kwant in your computer, please, follow the instructions on the Kwant webpage.

Phase 1: Obtaining the conduction band in the 2DEG region using nextnano++

The conduction band in the whole device can be obtained as a solution of the 3D-Poisson equation.

For realistic devices, a large number of nodes in the grid is required to evaluate with high accuracy the voltage that depletes electrons at the center of the 2DEG region. The *nextnano++* input file sweeps automatically the value of the top gate (V_{gate}) and generates 2D-slices of the band edges in the 2DEG plane that will be used in the next phase of the simulation.

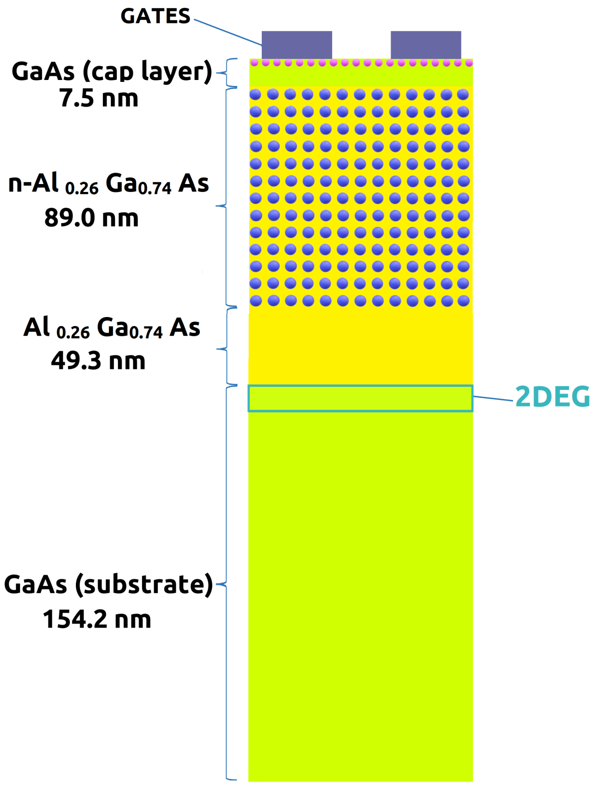


Figure 4.14.6.1: Schematics of a side view of the simulated device

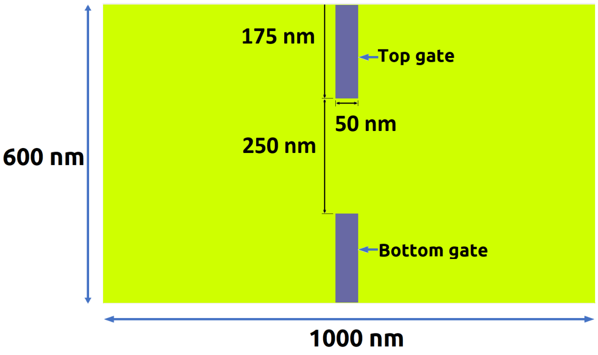


Figure 4.14.6.2: Top view of the gates deposited on the top of the simulations

Phase 2: Setting up Kwant

In order to setup Kwant in a consistent way with the configuration of *nextnano++* we need to define the next variables:

- the effective mass of electrons in the 2DEG region `ms = 0.067 * 9.109e-31`
- lattice constant of the tight-binding system (nm) `a = 1`
- conversion constant from eV (output of *nextnano++*) to Kwant energy unit `T = hbar*hbar/2/nm/nm/ms/e`

where:

- `e = 1.602e-19` is the electron charge (in C),
- `hbar = 6.626e-34/2/np.pi` is the Dirac constant (in Js),
- `h = 6.626e-34` is the Planck constant (in Js),
- `nm = 1e-9` is the conversion factor from 1 nanometer to 1 meter (in m),

Additionally, it is convenient to define a smaller portion of the slice of the potential obtained in the previous phase as the scattering region that will be used by Kwant. Here we will use a square scattering region with size of 400 nm x 400 nm, with the same center as before, the coordinates (0,0).

Phase 3: Computing the conductance coefficients with Kwant

Describing briefly the Kwant script *3D_conductance_in_top_gated_2DEG.py*, the program reads the file containing the potential in the 2DEG region (a 2D-slice at a depth of -146.8 nm under the surface), whose path is specified in the script through the variable `path_extracted_potential`. Through interpolation, Kwant maps the values of the potential into each node of the corresponding 2D-square lattice defined in the previous phase.

This is the basic element for building the system of equations to be solved under the tight-binding approach, whose the matrix elements and hoppings are set by discretization of the Hamiltonian:

$$H = -\frac{\hbar^2}{2ms}(\delta_x^2 + \delta_y^2) + V(x, y),$$

where $V(x, y)$ is the potential extracted from *nextnano++*. In this initial calculation we will start simulating the potential without computing Schrödinger equation.

The leads will be considered as ohmic contacts, and are attached to the left (lead 0) and to the right (lead 1) of the scattering region, as shown in Figure 4.14.6.3.

At this point it is convenient to verify the band edges of both leads, one of them plotted in the Figure 4.14.6.3. Finally the program solves the system of equations and the conductance from lead 0 to lead 1 is computed, for the especific potential imported. As example, when applying a voltage of -1.11 V to the upper gate of the structure, and -1.1 the the lower gate, the conductance between the two leads in the 2DEG is equal to 2.0074 $2e^2/h$

As we mentioned before, QPCs can be a very useful structure to control the conductance of electrons in a 2DEG region. In this example, we can verify how changes on the bias of one of the gates modifies the transport of electrons in the 2DEG region.

The Kwant script iteratively will import each potential simulated in nextnano and compute the correspondent conductance. This script requires that you have *nextnanopy* installed in your machine, that can be downloaded for free in our nextnanopy repository. In the script it will be required to modify variable `path_extracted_potential` with the path where the simulation results of *nextnano++* will be stored. As this process will process 101 files, it could take some minutes to perform the calculations. At the end of the process, a plot will be generated in your screen.

The Figure 4.14.6.5 presents the channel conductance computed for each value of V_{gate}. The steps in the curve show the expected quantization for this device.

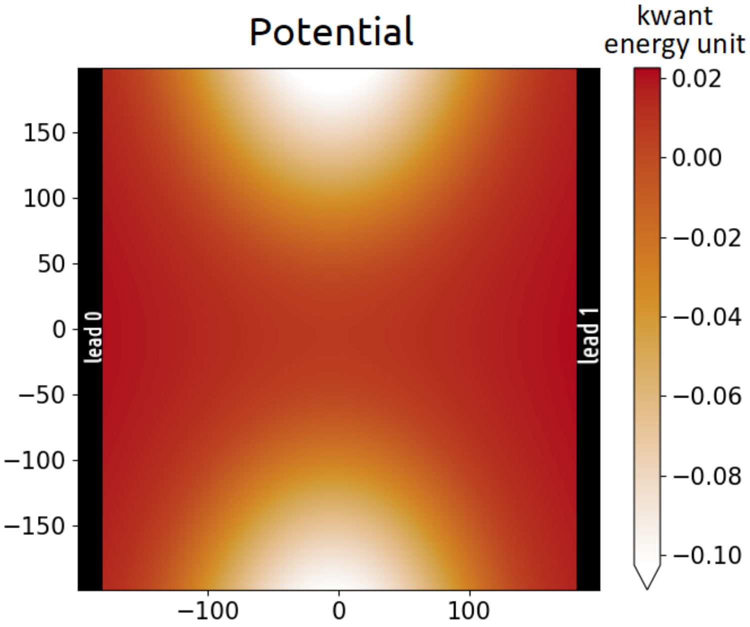


Figure 4.14.6.3: Imported conduction band when a bias of -1.11V is applied to the top gate.

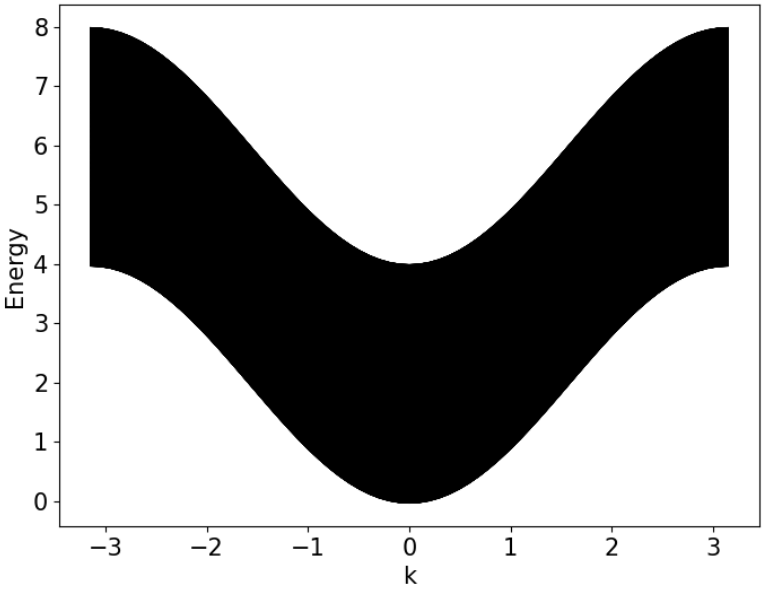


Figure 4.14.6.4: Band structure of the lead 0 for top gate voltage equal to -1.11 V.

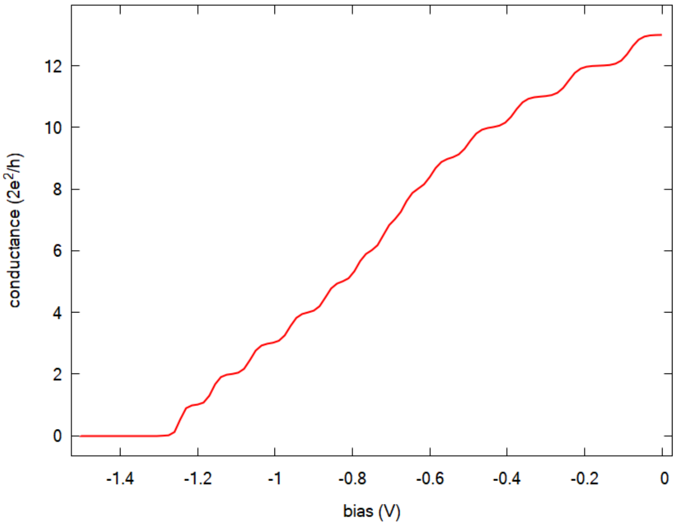


Figure 4.14.6.5: Conductance between lead 0 to lead 1 as function of the bias applied to the top gate

Phase 4: Computing conductance with potential from self-consistent Schrödinger-Poisson calculations

Until this point our potential has considered only the solutions of the Poisson equation for evaluation of the density of electrons in the 2DEG region. Nevertheless, it is expected that the density of states of the semiclassical potential be substantially different from the case when quantum effects are taken into account, especially at low energies.

Figure 4.14.6.6 presents the density distribution in the growth direction (perpendicular to the 2DEG plane) at the center of device ($x = 0$ nm and $y = 0$)for $V_{gate} = -1.17V$. They correspond to the results from *nextnano++* simulations with and without quantum calculations.

First we observe that both distributions present their maxima at different depths of the 2DEG. This result is expected because the confined states are discrete and present their maxima not so close to the interface. The integration of the density of states over a triangular-shaped potential for the semiclassical case generates distributions closer to the deepest part of the potential (close to the interface) when compared with the case including quantization.

Last but not least, we can observe that the peak of the electron distribution for the same value of V_{gate} is higher when quantum solution is not taken into account. This practically means that for the semiclassical solution it is required to apply more negative bias in order to deplete electrons that are accumulated close to the interface. In another words, it is expected that neglecting quantum effects the depletion of the electrons show occur at higher values than predicted from the semiclassical approach.

In order to analyse the impact of including quantum effects in the conductance calculations we need to import the final results from *nextnano++* values of the eigenstate of the ground state (E1) from the file *energy_subbands_quantum_region_Gamma_2d_2deg_slice.fld* in the *Quantum* folder. The imported potentials used both cases (with and without quantization) were obtained for a 2D-slice 161.8 nm below the surface, where the density of electrons for the quantum solution is maximum.

We can observe that at this plane the depletion of electrons in both simulations occurs at the same bias (around -1.11 V), as discussed and predicted above.

As a final conclusion, for accurate determination of the pinch-off voltages, obtaining the potential from self-consistent simulations of Schrödinger-Poisson are required.

Exercise:

In order to reproduce the figures of the last section, modify and run the *nextnano++* input file for both cases:

- `$solve_quantum = 0` and use the option `$slice_in_2DEG = 161.8` at the line 77 (save the input file with the name *3D_conductance_in_top_gated_2DEG_exercise_nnp.in*)
- `$solve_quantum = 1` and use the option `$slice_in_2DEG = 161.8` at the line 77 (save

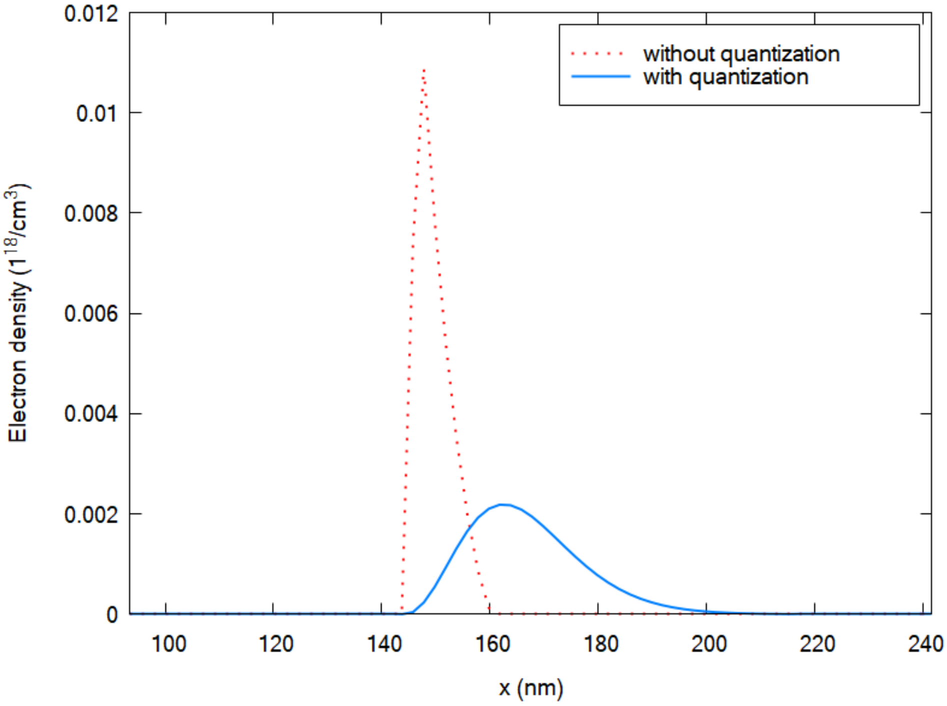


Figure 4.14.6.6: Density of electrons in the growth direction at the center of the device ($y = 0$ nm and $z = 0$) for $V_{gate} = -1.17V$ for semiclassical computation (without quantization) and for self-consistent Schrödinger-Poisson calculations (with quantization)

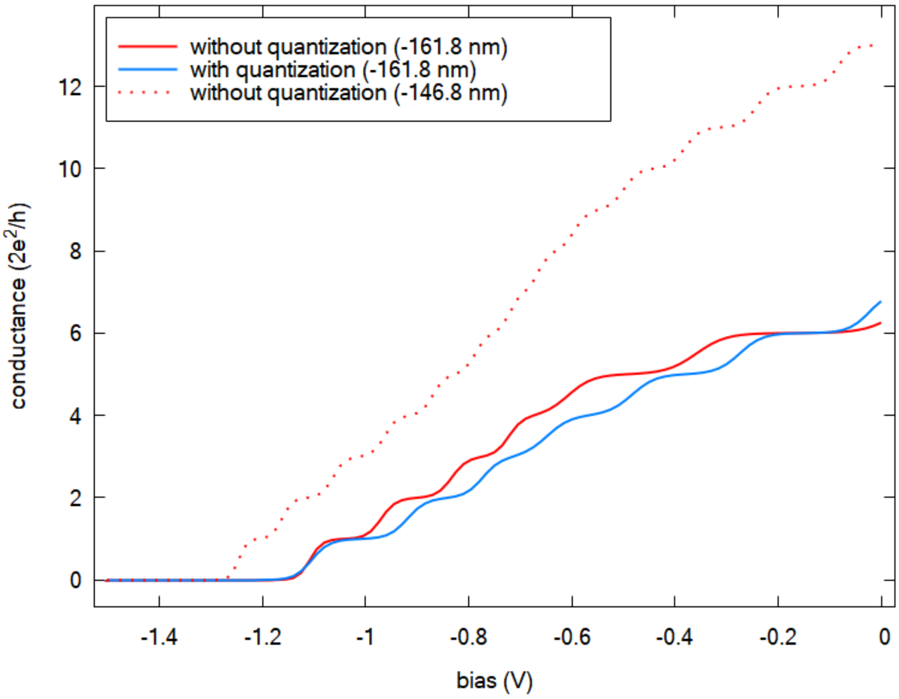


Figure 4.14.6.7: Conductance between lead 0 to lead 1 as function of the bias applied to the top gate at the plane z = 161.8 nm in the 2DEG region with and without quantization along the growth direction (in solid lines). In dotted lines the conductance without quantization is shown at the depth where the electron density is higher in the 2DEG (146.8 nm below the surface, as shown in Figure 4.14.6.5)

the input file with the name *3D_conductance_in_top_gated_2DEG_QM_exercise_nnp.in*)

> Edit the path of the output folders of both simulations in the script *3D_conductance_in_top_gated_2DEG_exercise.py* (variables `path_extracted_potential_Poisson` and `path_extracted_potential_QM`), and compute the transmission.

This tutorial is based on the nextnano GmbH collaboration in the scope of the UltraFastNano Project aiming at development of the first Flying Electron Qubit at the picosecond scale, and it is funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 862683.

Last update: 2025/06/27

4.15 Transistors

4.15.1 HEMT structure (High Electron Mobility Transistor)

Input files:

- *HEMT_1D_nnp.in*
- *HEMT_2D_nnp.in*
- *HEMT_3D_nnp.in*

Scope:

This tutorial demonstrates how High Electron Mobility Transistors can be modelled with *nextnano++*.

HEMT structure

Input file: *HEMT_1D_nnp.in*

The structure consists of the following material layers:

	width [nm]	material
1		Schottky barrier 0.2 eV
2	10.0	$In_{0.532}Ga_{0.468}As$
3	25.0	$Al_{0.477}In_{0.523}As$
4	50.0	$In_{0.532}Ga_{0.468}As$
5	300.0	$Al_{0.477}In_{0.523}As$
6	300.0	InP

The conduction band edge profile without doping is plotted in Figure 4.15.1.1.

Now we add at x = 35 nm a silicon delta doping of $4.5 \cdot 10^{12}$ cm^{-2} which leads to band bending. Instead of choosing a delta doping we specify a constant doping of $1.5 \cdot 10^{20}$ cm^{-3} that extends over 0.3 nm. ($1.5 \cdot 10^{20}$ cm^{-3} $\cdot$ $3 \cdot 10^{-8}$ cm = $4.5 \cdot 10^{12}$ cm^{-2})

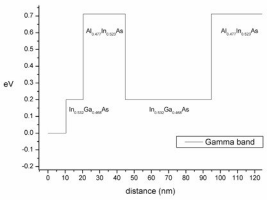


Figure 4.15.1.1: Calculated conduction band edge profile.

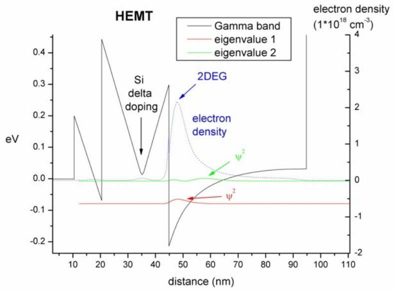


Figure 4.15.1.2: Calculated conduction band edge profile and probability densities.

We obtain two eigenstates and their corresponding wave functions inside the HEMT channel which leads to a two-dimensional electron gas (2DEG), see Figure 4.15.1.2. The electron density is plotted in blue.

In the file *bias_00000/total_charges.txt* we can find the integrated electron and hole densities. The total integrated density (from 10 nm to 100 nm) which can be measured experimentally is $1.87 \cdot 10^{12}$ cm^{-2} in agreement with the experiment. Most of the density is located between 45 nm and 95 nm.

2D/ 3D simulations

Input files: *HEMT_2D_nnp.in*, *HEMT_3D_nnp.in*

Input files for the same HEMT structure as in 1D, this time for a 2D and 3D simulations, are also available.

- 2D: rectangle of dimension 250 nm x 10 nm
- 3D: cuboid of dimension 250 nm x 10 nm x 10 nm

Last update: nnnn/nn/nn

4.15.2 Two-dimensional electron gas in an AlGaN/GaN FET

Input files:

- *Jogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig4Fig1Fig7_1D_nnp.in*
- *Jogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig2Fig3_1D_nnp.in*
- *Jogai_AlGaNGaN_FET_JAP2003_GaNcap_Fig4_1D_nnp*
- *Jogai_AlGaNGaN_FET_JAP2003_GaNcap_Fig6Fig5_1D_nnp.in*

Note

The input files are also available as 2D input file.

Scope:

This tutorial tries to reproduce the results of *[Jogai2003]*.

Introduction

For this one-dimensional simulation of an $AlGaN$/ GaN heterojunction field effect transistor (HFET) we are solving self-consistently the Schrödinger-Poisson equation taking into account strain, and piezo- and pyroelectric charge densities.

At the left boundary we use a Schottky contact boundary condition with a Schottky barrier height of ϕ_B = 1.4 eV. Note that in Fig. 1 of *[Jogai2003]*, the Schottky barrier height corresponds to

$$\mathrm{e}\phi_B = E_c - E_F$$

which fixes the conduction band edge energy E_c above the Fermi energy E_F, where e is the elementary charge. Alternative boundary conditions such as a fixed surface charge density or surface states based on incomplete ionization of donor or acceptor states are described in the *— FREE — Surface Charges*.

Our simulated structure is undoped. Note that the 2DEG is present even in the absence of doping due to piezo- and pyroelectric interface charge densities. The temperature is set to 300 K in all simulations. We only consider cation-faced structures, i.e. we have rotated the crystal so that our [000-1] direction points along the positive x direction.

Figure 4.15.2.1 shows the results of the input file *1DJogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig4Fig1Fig7_nnp.in*.

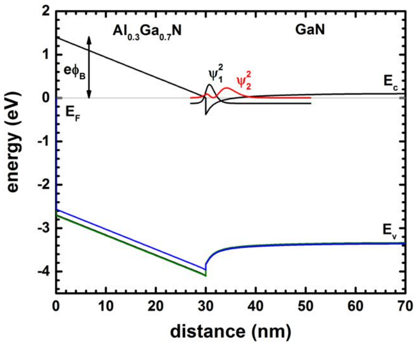


Figure 4.15.2.1: Calculated conduction and three valence band edges with the probability densities of the two lowest subbands of a 30 nm $Al_{0.3}Ga_{0.7}N$ / 40 nm GaN heterostructure.

Variation of the $Al_xGa_{1-x}N$ layer thickness and alloy content x (Fig. 2 and Fig. 3 of [Jogai2003])

Now we try to reproduce Fig. 2 and Fig. 3 of *[Jogai2003]*, with the input file *Jogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig2Fig3_1D_nnp.in*. We are calculating the variation of the 2DEG density with the

- $Al_xGa_{1-x}N$ layer thickness and
- mole fraction (alloy content x).

Within the *nextnano++* input file, we can perform a sweep over the alloy concentration very conveniently:

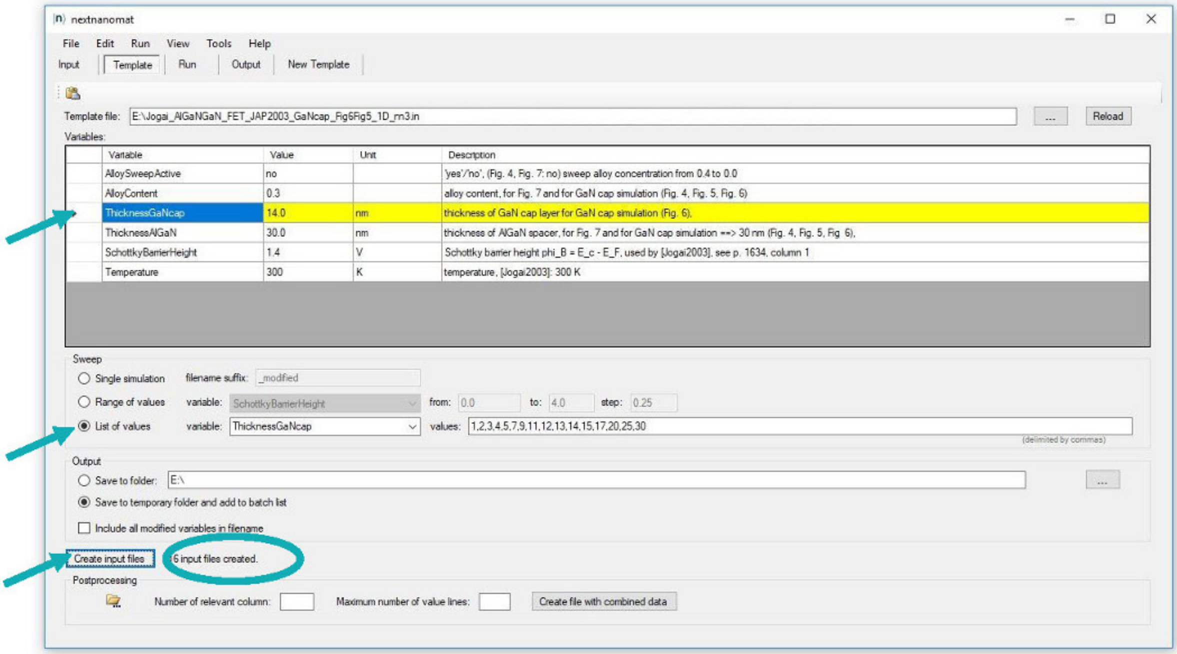
```
$AlloySweepActive = yes   # sweep alloy concentration from 0.4 to 0.0␣
↪(HighlightInUserInterface)
```

The thickness of the $Al_xGa_{1-x}N$ barrier is defined as a variable.

```
$ThicknessAlGaN   = 30.0  # thickness of AlGaN spacer (ListOfValues:6,10,14,18,22,26,
↪30,34,38)(DisplayUnit:nm)(HighlightInUserInterface)
```

We use *nextnanomat*'s Template feature in order to sweep over the $Al_xGa_{1-x}N$ barrier thickness. This is shown in the following screenshot. The input files are created automatically and are added to the "Run" tab.

The 2DEG sheet carrier concentration can be found in this file: *bias_00000\total_charges.txt*. This file contains the integrated electron density for the whole simulation region.



Template file: E:\Jogai_AlGaNGaN_FET_JAP2003_GaNcap_Fig6Fig5_1D_nn3.in

Variables:

Variable	Value	Unit	Description
AlloySweepActive	no		'yes'/'no', (Fig. 4, Fig. 7: no) sweep alloy concentration from 0.4 to 0.0
AlloyContent	0.3		alloy content, for Fig. 7 and for GaN cap simulation (Fig. 4, Fig. 5, Fig. 6)
ThicknessGaNcap	14.0	nm	thickness of GaN cap layer for GaN cap simulation (Fig. 6),
ThicknessAlGaN	30.0	nm	thickness of AlGaN spacer, for Fig. 7 and for GaN cap simulation ==> 30 nm (Fig. 4, Fig. 5, Fig 6),
SchottkyBarrierHeight	1.4	V	Schottky barrier height phi_B = E_c - E_F, used by [Jogai2003], see p. 1634, column 1
Temperature	300	K	temperature, [Jogai2003]: 300 K

Sweep
○ Single simulation filename suffix: _modified
○ Range of values variable: SchottkyBarrierHeight from: 0.0 to: 4.0 step: 0.25
● List of values variable: ThicknessGaNcap values: 1,2,3,4,5,7,9,11,12,13,14,15,17,20,25,30
(delimited by commas)

Output
○ Save to folder: E:\
● Save to temporary folder and add to batch list
☐ Include all modified variables in filename

Create input files 16 input files created.

Postprocessing
Number of relevant column: Maximum number of value lines: Create file with combined data

The following figure shows the total integrated electron density as a function of alloy concentration for various $AlGaN$ thicknesses. Note that these results were obtained by using one input file template only: *1DJogai_AlGaNGaN_FET_JAP2003_nn3_Fig2Fig3.in*.

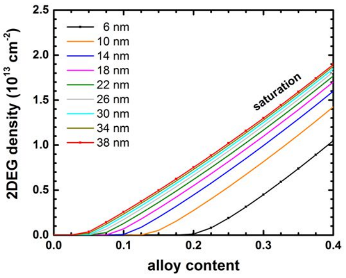


Figure 4.15.2.2: Calculated 2DEG density for different layer widths of $Al_xGa_{1-x}N$ as a function of alloy content x.

For a given barrier thickness, the 2DEG sheet carrier concentration varies almost linearly with alloy concentration x. The 2DEG density approaches saturation as the barrier thickness is increased. This fact can be better seen Figure 4.15.2.3 where we show exactly the same data.

Our results seem to be in reasonable agreement to the simulations of *[Jogai2003]* (Fig. 2 and Fig. 3).

Variation of the Schottky barrier height (Fig. 7 of [Jogai2003])

Using te input file *Jogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig4Fig1Fig7_1D_nnp.in*, we vary the Schottky barrier height phi_B and calculate for each value the 2DEG density:

```
$SchottkyBarrierHeight = 1.4    # Schottky barrier height phi_B = E_c - E_F, used by␣
↪[Jogai2003], see p. 1634, column 1 (ListOfValues:1.40,1.42,1.
↪65)(RangeOfValues:From=0.0,To=4.0,Step=0.25)(DisplayUnit:V)
```

This situation is equivalent to fixing the surface potential to

$$ephi_B = E_c - E_F.$$

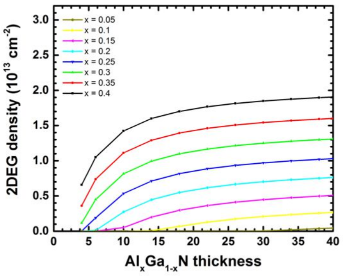
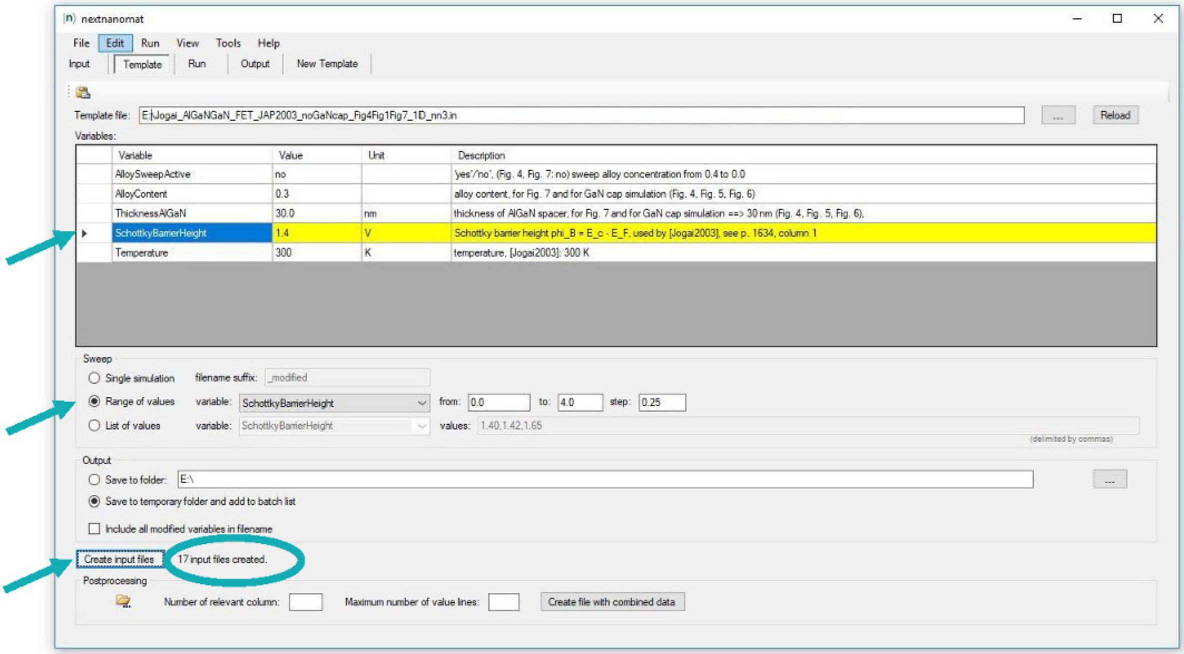


Figure 4.15.2.3: Calculated 2DEG density for different alloy contents x of $Al_xGa_{1-x}N$ as a function of layer widths.



Template file: E:\Jogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig4Fig1Fig7_1D_nn3.in

Variables:

Variable	Value	Unit	Description
AlloySweepActive	no		'yes'/'no', (Fig. 4, Fig. 7: no) sweep alloy concentration from 0.4 to 0.0
AlloyContent	0.3		alloy content, for Fig. 7 and for GaN cap simulation (Fig. 4, Fig. 5, Fig. 6)
ThicknessAlGaN	30.0	nm	thickness of AlGaN spacer, for Fig. 7 and for GaN cap simulation ==> 30 nm (Fig. 4, Fig. 5, Fig. 6),
SchottkyBarrierHeight	1.4	V	Schottky barrier height phi_B = E_c - E_F, used by [Jogai2003], see p. 1634, column 1
Temperature	300	K	temperature, [Jogai2003]: 300 K

Sweep

○ Single simulation filename suffix: _modified

● Range of values variable: SchottkyBarrierHeight from: 0.0 to: 4.0 step: 0.25

○ List of values variable: SchottkyBarrierHeight values: 1.40,1.42,1.65 (delimited by commas)

Output

○ Save to folder: E:\

● Save to temporary folder and add to batch list

☐ Include all modified variables in filename

17 input files created.

Postprocessing

Number of relevant column:

Maximum number of value lines:

Figure 4.15.2.4 shows the calculated 2DEG density as a function of Schottky barrier height, i.e. surface potential. We used a 30 nm $Al_{0.3}Ga_{0.7}N$ barrier. Again, the 2DEG sheet carrier concentration can be found in this file: *bias_00000\total_charges.txt*. The results are in reasonable agreement to Fig. 7 of *[Jogai2003]*.

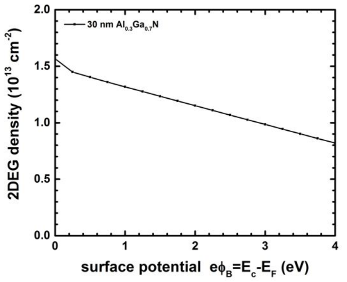


Figure 4.15.2.4: Calculated 2DEG density as a function of surface potential $e\phi_B$.

AlGaN/GaN FET including a GaN cap layer

Now we compare HFET structures **with** and **without** a GaN cap layer by using the input files *Jogai_AlGaNGaN_FET_JAP2003_GaNcap_Fig4_1D_nnp.in* and *Jogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig4Fig1Fig7_1D_nnp.in*. GaN-capped HFETs have a lower 2DEG density compared to uncapped structures. For the case of a 30 nm $Al_{0.3}Ga_{0.7}N$ barrier, introducing a GaN cap layer reduces the density of the 2DEG:

- 5 nm cap: The calculated 2DEG density is n = 1.03 · 10^{13} cm^{-2} (n = 1.20 · 10^{13} cm^{-2} *[Jogai2003]*).
- without cap: The calculated 2DEG density is n = 1.25 · 10^{13} cm^{-2} (n = 1.47 · 10^{13} cm^{-2} *[Jogai2003]*).

Figure 4.15.2.5 compares the band edges of capped and uncapped HEMT structure.

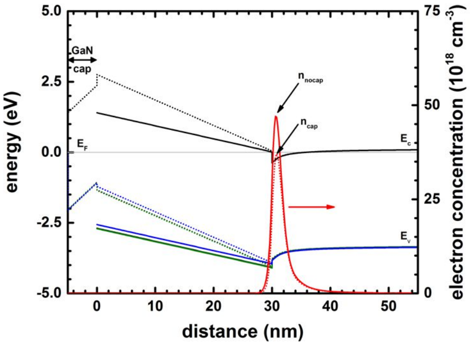


Figure 4.15.2.5: Calculated conduction and valence band edges of a $Al_{0.3}Ga_{0.7}N$/ GaN FET **with** (solid lines) and **without** (dotted lines) a 5 nm GaN cap layer.

Figure 4.15.2.6 shows the band edges and the electron and hole densities for a 14 nm GaN cap layer. The $Al_{0.3}Ga_{0.7}N$ barrier thickness is 30 nm. For GaN cap layers thicker than 14 nm, a 2DHG forms. The density of the 2DHG screens the surface potential so that the density of the 2DEG is maintained at a constant level even if the GaN cap layer thickness increases further.

- The calculated 2DHG density is p = 0.513 · 10^{12} cm^{-2} (p = 1.77 · 10^{12} cm^{-2} *[Jogai2003]*).
- The calculated 2DEG density is n = 0.839 · 10^{13} cm^{-2} (n = 1.009 · 10^{13} cm^{-2} *[Jogai2003]*).

Variation of the GaN cap layer thickness (Fig. 5 of [Jogai2003])

Input file: *Jogai_AlGaNGaN_FET_JAP2003_GaNcap_Fig6Fig5_1D_nnp.in*

Now we are going to vary the GaN cap layer thickness.

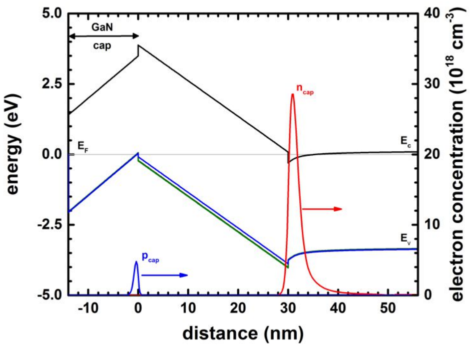
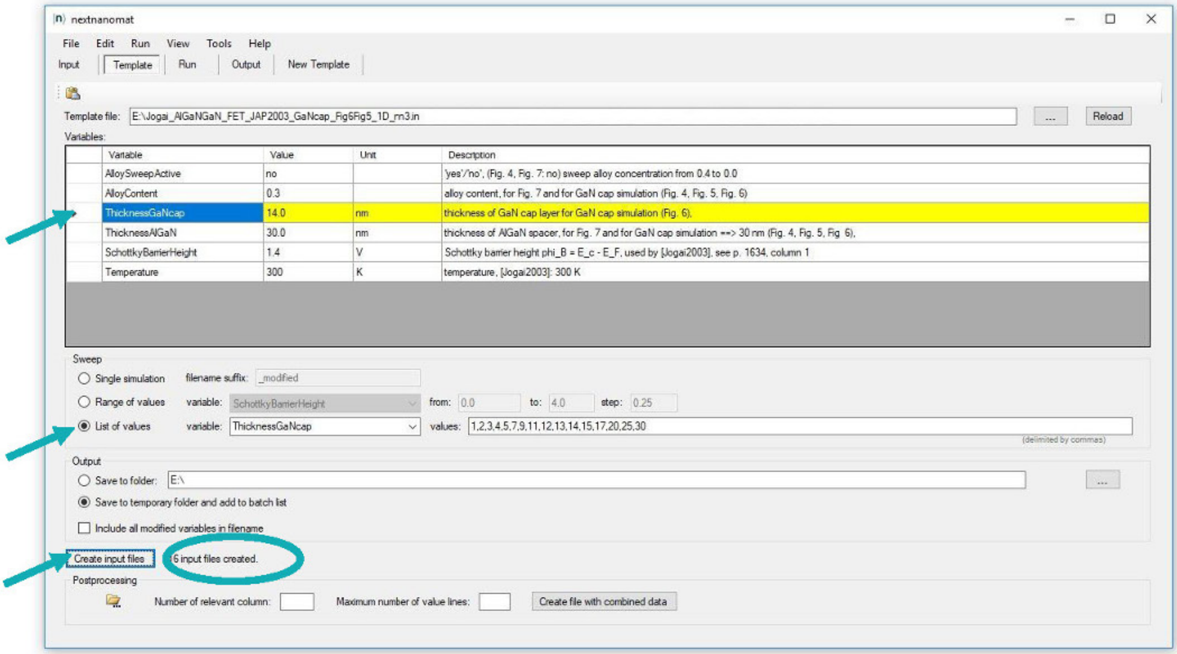


Figure 4.15.2.6: Calculated conduction and valence band edges of a $Al_{0.3}Ga_{0.7}N$/ GaN FET with 14 nm GaN cap.

```
$ThicknessGaNcap = 5.0      # thickness of GaN cap layer for GaN cap simulation (Fig. 4, Fig. 5), (ListOfValues:1,2,3,4,5,7,9,11,12,13,14,15,17,20,25,30)(DisplayUnit:nm)(HighlightInUserInterface)
```



Template file: E:\Jogai_AlGaNGaN_FET_JAP2003_GaNcap_Fig6Fig5_1D_nn3.in

Variables:

Variable	Value	Unit	Description
AlloySweepActive	no		'yes'/'no', (Fig. 4, Fig. 7: no) sweep alloy concentration from 0.4 to 0.0
AlloyContent	0.3		alloy content, for Fig. 7 and for GaN cap simulation (Fig. 4, Fig. 5, Fig. 6)
ThicknessGaNcap	14.0	nm	thickness of GaN cap layer for GaN cap simulation (Fig. 6).
ThicknessAlGaN	30.0	nm	thickness of AlGaN spacer, for Fig. 7 and for GaN cap simulation ==> 30 nm (Fig. 4, Fig. 5, Fig 6),
SchottkyBarrierHeight	1.4	V	Schottky barrier height phi_B = E_c - E_F, used by [Jogai2003], see p. 1634, column 1
Temperature	300	K	temperature, [Jogai2003]: 300 K

Sweep

○ Single simulation filename suffix: _modified

○ Range of values variable: SchottkyBarrierHeight from: 0.0 to: 4.0 step: 0.25

● List of values variable: ThicknessGaNcap values: 1,2,3,4,5,7,9,11,12,13,14,15,17,20,25,30 (delimited by commas)

Output

○ Save to folder: E:\

● Save to temporary folder and add to batch list

☐ Include all modified variables in filename

Create input files 16 input files created.

Postprocessing

Number of relevant column:

Maximum number of value lines:

Create file with combined data

Figure 4.15.2.7 shows the 2DEG density vs. GaN cap layer thickness for a 30 nm $Al_{0.3}Ga_{0.7}N$ barrier. Beyond a GaN cap layer thickness of ~13 nm (12 nm *[Jogai2003]*) the 2DEG density saturates.

Additional comments

In contrast to the article of *[Jogai2003]*, we did not include exchange-correlation effects and we used a single-band model for the 2DHG rather than a 6-band k.p model.

Last update: nnnn/nn/nn

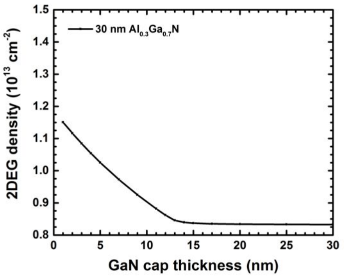


Figure 4.15.2.7: Calculated 2DEG density as a function of GaN cap thickness.

4.15.3 MOS Capacitor & MOSFET

Section author: Daryoush Nosraty Alamdary

The purpose of this tutorial is to show how the results of our simulation software (which solves the Poisson and drift-diffusion current equations numerically) compare with analytical equations given in standard text books on MOSFETs. The analytical equations use certain approximations and assumptions which limit their applicability. Nevertheless, in most cases the agreement is very good as demonstrated in this tutorial.

Contents

Part 1: Capacitance-voltage characteristics of a 2D MOS capacitor

In the first part of this tutorial we discuss the capacitance-voltage (C-V) characteristics of the MOS capacitor in a 2D simulation. (For a 1D simulation of the C-V characteristics, see also this tutorial: "Capacitance-Voltage curve of a "metal"-insulator-semiconductor (MIS) structure"). Our MOS has the same dimensions and properties (channel length, doping profiles and gate contact type) as the corresponding MOSFET discussed in Part 2.

Part 2: Current-voltage characteristics of a 2D n-Channel MOSFET

In the second part of the tutorial, we start with the design of the MOSFET based on its 2D MOS capacitor, and then discuss its input and output characteristics and their respective conductances, namely transconductance and channel conductance.

Part 3: Mobility models and pinch-off in a 2D n-Channel MOSFET

In this part we discuss and compare the effect of different mobility models on the output characteristics of the MOSFET and how they affect properties such as pinch-off, saturation, etc.

References

1. [Goetzberger] A. Goetzberger, M. Schulz, Fundamentals of MOS Technology, In: H. J. Queisser (eds) Festkörperprobleme 13, Advances in Solid State Physics 13, Springer, Berlin, Heidelberg, 309-336 (1973), https://doi.org/10.1007/BFb0108576
2. [Wu] Y.-C. Wu, Y.-R. Jhan, 3D TCAD Simulation for CMOS Nanoeletronic Devices, Springer, Singapore (2018)
3. [Sze] S. M. Sze, K. K. NG, Physics of Semiconductor Devices (3rd ed.), John Wiley, New York (2007)
4. [Brews] J. R. Brews, W. Fichtner, E. H. Nicollian, S. M. Sze, Generalized guide for MOSFET miniaturization, IEEE Electron Device Letters 1, 2 (1980) https://doi.org/10.1109/EDL.1980.25205
5. [Miura-Mattausch] M. Miura-Mattausch, H. J. Mattausch, N. D. Arora, C. Y. Yang, MOSFET modeling gets physical, IEEE Circuits and Devices Magazine 17, 29 (2001) https://doi.org/10.1109/101.968914

2D MOS Capacitor

Input files:

- *MOS_CV_5 nmSiO2_5 nmCont_Dop1e16_QM_1D_fine_grid.in*
- *MOS_CV_5 nmSiO2_5 nmCont_Dop1e16_QM_1D.in* (nonuniform grid)
- *MOS_CV_5 nmSiO2_5 nmCont_Dop1e16_QM_2D.in*
- *MOS_CV_5 nmSiO2_5 nmCont_Dop1e16_QM_2D_periodic_x.in* (uniform grid along x direction with periodic boundary conditions, quasi-1D simulation)

In this tutorial we illustrate the simulation and analysis of an N-channel MOSFET (Metal-Oxide-Semiconductor Field-Effect Transistor) in 2D as implemented in CMOS technologies and nanodevice fabrication. The first step in simulating the MOSFET is the construction and the simulation of the corresponding MOS capacitor, i.e. the Metal-Oxide-Semiconductor device, which can act as a capacitor on its own, and is an integral part of the MOSFET. The gate contact on this capacitor is the same gate contact as of the MOSFET, and it underlies the same physics in both the MOS and the MOSFET. The 2D sketch of the MOS capacitor is illustrated in the following figure Figure 4.15.3.1

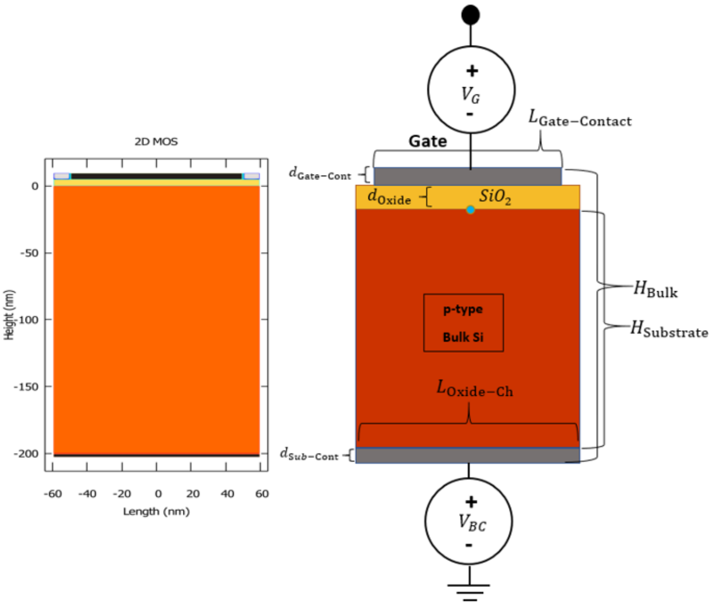


Figure 4.15.3.1: The geometry of the 2D MOS design, and its equivalent geometry from the output file *regions.vtr* (colored differently in post-processing). The blue circle indicates the position of the origin of our (x, y) coordinate system.

In this tutorial we use a p-doped bulk-Si MOS with a Schottky contact at the gate (instead of a poly-Si contact), and ohmic contact at the substrate. Therefore, the effect of poly-Si depletion at the gate is not present in either of the devices in order to produce the C-V characteristics of our capacitor, which then is the same MOS device used within the N-Ch MOSFET. The bulk p-doping level is $1 \times 10^{16}\mathrm{cm}^{-3}$, and the oxide layer which consists of SiO:sub:*2* has a thickness of $d_{\mathrm{ox}} = 5\mathrm{nm}$. The length of the channel is $L_{\mathrm{G}} = 100\mathrm{nm}$, the substrate has a height of $H_{\mathrm{Substrate}} = 200\mathrm{nm}$. The importance of the C-V characteristics of the MOS device derives from the fact, that the charge inversion layer, that is responsible for conduction in the MOSFET, is generated by the capacitive properties of the MOS devices.

Low-Frequency Capacitance

In what follows are the results of our numerical calculations. Concretely, we solve the coupled Schrödinger, Poisson and current equations in two dimensions. We compare our results with the analytic formulas given in standard text books.

The low-frequency capacitance of a MOS capacitor can be measured experimentally with a low frequency signal. In the simple case scenario, the interface trapped charges (charges trapped in the oxide) usually play no role in the capacitance of the device and are not considered in our simulations. Therefore the total capacity of the device is a series connection of the oxide capacitance and the depletion layer capacitance,

$$C_{\mathrm{tot}} = \frac{C_{\mathrm{ox}} C_{\mathrm{D}}}{C_{\mathrm{ox}} + C_{\mathrm{D}}}. \tag{4.15.3.1}$$

The oxide capacitance is the capacitance of the oxide layer, which is independent of the bias, and is simply calculated according to $C_{\mathrm{ox}} = \varepsilon_{\mathrm{ox}}/d_{\mathrm{ox}}$. This gives a capacitance per unit area $(\mathrm{F/cm^2})$. Multiplying this value with the length L_{G} and width W of the gate gives a capacitance in units of F.

The depletion layer capacitance is calculated using the charge in the depletion layer as defined in equation (4.15.3.2),

$$Q_{\mathrm{D}} = qW_{\mathrm{D}}N_{\mathrm{Sub}}, W_{\mathrm{D}} = \sqrt{\frac{\varepsilon_{\mathrm{s}}^2}{C_{\mathrm{ox}}^2} + \frac{2\varepsilon_{\mathrm{s}}V}{qN_{\mathrm{Sub}}}} - \frac{\varepsilon_{\mathrm{s}}}{C_{\mathrm{ox}}}, \tag{4.15.3.2}$$

where W_{D} is the width of the depletion layer, ε_{s} is the dielectric constant of the semiconductor and $\varepsilon_{\mathrm{ox}}$ the dielectric constant of the oxide. The depletion layer capacitance is then given by the derivative $\partial Q_{\mathrm{D}}/\partial\psi_{\mathrm{s}}$, where ψ_{s} is the surface potential. Further details on the surface potential can be found in the appendix section. Therefore, the total capacitance calculated according to these formulas would approximately approach the C_{ox} at its maximum, would have a flat-band capacitance C_{FB} given by the expression in equation (4.15.3.3) , i.e. the capacitance at the voltage, which creates the flat-band condition in the MOS band structure,

$$C_{\mathrm{FB}}(\psi_{\mathrm{s}} = 0) = \frac{\varepsilon_{\mathrm{s}}\varepsilon_{\mathrm{ox}}}{\varepsilon_{\mathrm{s}} d_{\mathrm{ox}} + \varepsilon_{\mathrm{ox}} L_{\mathrm{D}}}, L_{\mathrm{D}} = \sqrt{\frac{k_{\mathrm{B}}T\varepsilon_{\mathrm{s}}}{q^2 N_{\mathrm{Sub}}}}, \tag{4.15.3.3}$$

with L_{D} as the Debye screening length. The Debye length for our MOS capacitor amounts to $\approx 40.8\mathrm{nm}$, and with that the flat-band capacitance is calculated to be $C_{\mathrm{FB}} \approx 1.85\mathrm{mF/m}^2$, the equivalent of $1.85\mathrm{pF/cm}$ if the channel length is $100\mathrm{nm}$. The C-V curve of the MOS, taking the entire substrate for charge integration, with $\partial Q_{\mathrm{Sub}}/\partial V_{\mathrm{Bias}}$ is shown in figure Figure 4.15.3.3. Note that the output of the simulations, however, is only the total charge (per cm in 2D), as shown in figure Figure 4.15.3.2, which needs to be (first multiplied with the elementary charge $|q|$, and then) derived with respect to the bias voltage:

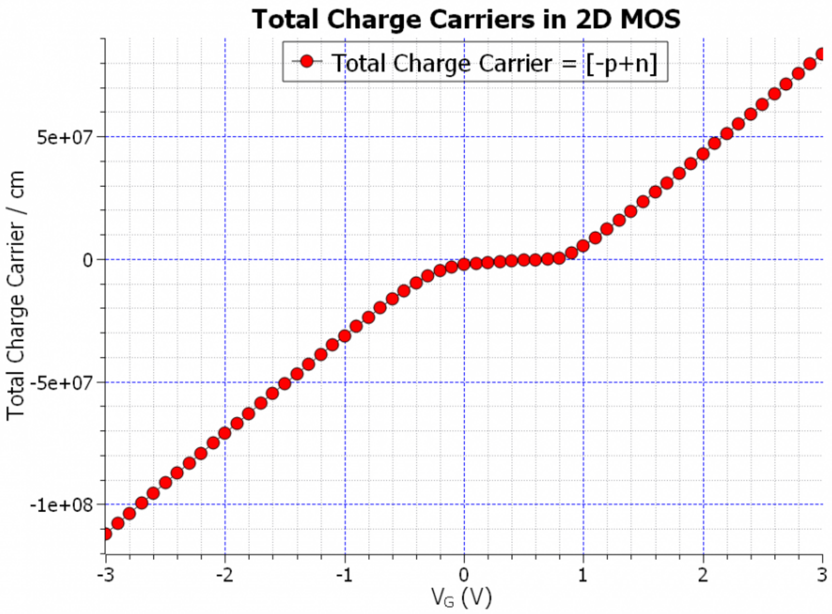


Figure 4.15.3.2: The total charge carriers per cm of the MOS, integrated in the substrate, vs. the applied gate bias.

In the above figure the C^*_{FB} is marked with * because the value measured differs from the calculated value. Later we will show how the C-V curve could be measured, so that the value of the flat-band capacitance is consistent with (4.15.3.3).

There are three values which we read from the graph (actually four but since we have the band edges here in the simulation output, we just need three). The first is the oxide capacitance C_{ox}, which is approximately the ceiling

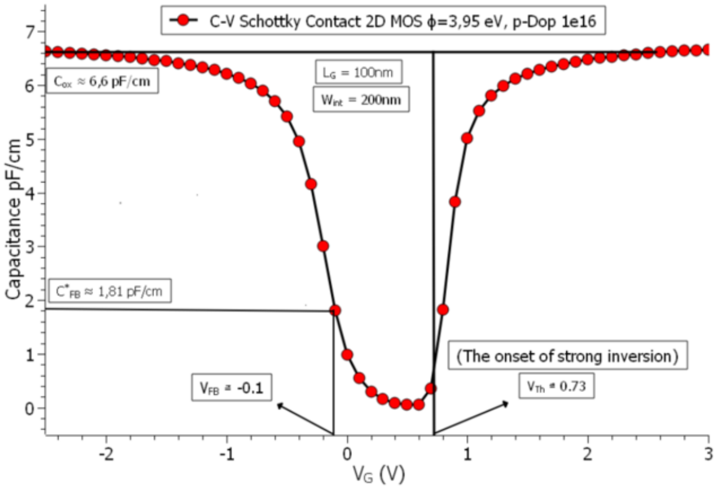


Figure 4.15.3.3: The C-V characteristics of the 2D MOS with $N_{\mathrm{Sub}} = 10^{16}\mathrm{cm}^{-3}$ doping concentration in the p-doped silicon substrate, channel length of 100 nm, a Schottky barrier of $\phi_{\mathrm{B}} = 3.95\mathrm{eV}$, and a charge integration region equal to the entire substrate. (Note that the flat-band voltage has been chosen from the observation of the band edges in the simulation output, which are flat for the bias value of $-0.1\mathrm{V}$).

of the curve. The second is the flat-band capacitance C_{FB}, corresponding to the value of the flat-band voltage V_{FB} (read from the status of the band edges in the simulation output). And the third is the threshold voltage V_{Th}, which is the onset of the strong inversion. The flat-band condition in the 1-dimensional band edges output is illustrated in figure Figure 4.15.3.4:

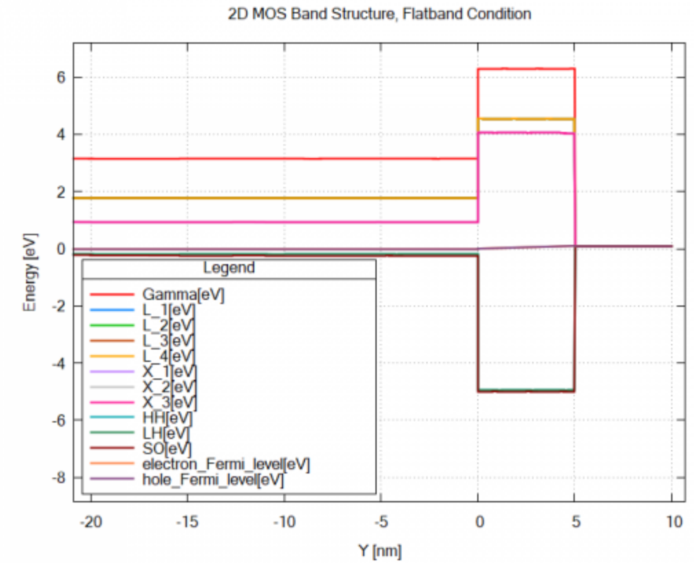


Figure 4.15.3.4: The alignment of conduction and valence band edges with respect to the Fermi levels of the 2D MOS under the flat-band condition along a one-dimensional slice along the y direction. (The lowest conduction band edge is labeled with X.

The bias voltage that results in a band structure in the figure Figure 4.15.3.4, is called the flat-band voltage V_{FB}. This voltage is related to, and is a part of the definition of the threshold voltage,

$$V_{\mathrm{Th}} = V_{\mathrm{FB}} + 2\psi_{\mathrm{B}} + \frac{\sqrt{4\varepsilon_{\mathrm{Si}} q N_{\mathrm{Sub}} \psi_{\mathrm{B}}}}{C_{\mathrm{ox}}}. \tag{4.15.3.4}$$

The ψ_{B} is the distance of the semiconductor Fermi level to the mid-point of the band gap, and it is estimated that the onset of the strong inversion is at the point when the surface potential $\psi_{\mathrm{s}} \approx 2\psi_{\mathrm{B}}$. This surface potential is estimated to be

$$\psi_{\mathrm{s}}(\mathrm{stronginversion}) \approx \frac{2k_{\mathrm{B}}T}{q}\ln\left(\frac{N_{\mathrm{Sub}}}{n_i}\right) \tag{4.15.3.5}$$

Calculating this expression for our system, the surface potential amounts to $\approx\ 0.713\mathrm{V}$, while the expression $\sqrt{4\varepsilon_{\mathrm{Si}}qN_{\mathrm{Sub}}\psi_{\mathrm{B}}}/C_{\mathrm{ox}} \approx 0.073\mathrm{V}$, which is actually the voltage drop across the oxide layer V_{ox}. Therefore taking the flat-band voltage $V_{\mathrm{FB}} = -0.1\mathrm{V}$, we arrive at a threshold voltage $V_{\mathrm{Th}} \approx 0.7\mathrm{V}$, which is somewhat lower than the $0.73\mathrm{V}$ read from the curve. Indeed the value of the threshold voltage is strongly affected by the value of the Schottky barrier.

The height of the Schottky barrier used here, however, has to reflect the metal-SiO:sub:*2* interface barrier, and not the metal-semiconductor barrier. This barrier depends on the metal and its work function that is used, and is therefore different for different metals. It is also mentioned in [Wu], that "**the work function of the metal gate has to be properly defined in order to achieve the expected threshold voltage $\mathbf{V_{\mathrm{Th}}}$**". Even though that the barrier heights for metals such as aluminum have been reported to be around $3.15\mathrm{eV}$, the barrier height of metals such as gold (Au), and silver (Ag), have been reported to be around $4.0\mathrm{eV}$ [Goetzenberger]. Here, in order to arrive at a threshold voltage of $0.7\mathrm{V}$, the barrier had to be chosen $3.95\mathrm{eV}$.

The Schottky Barrier, Doping Concentration, Depletion Region

In the following part we look at a set of figures, which illustrate various parameter changes, which then lead to variations in the three important values which we want to read from the C-V curve. First would be the threshold voltage, and the flatband voltage, both of which could be influenced by the height of the Schottky barrier, and the doping concentration in the bulk-semiconductor, as figure Figure 4.15.3.5 illustrates:

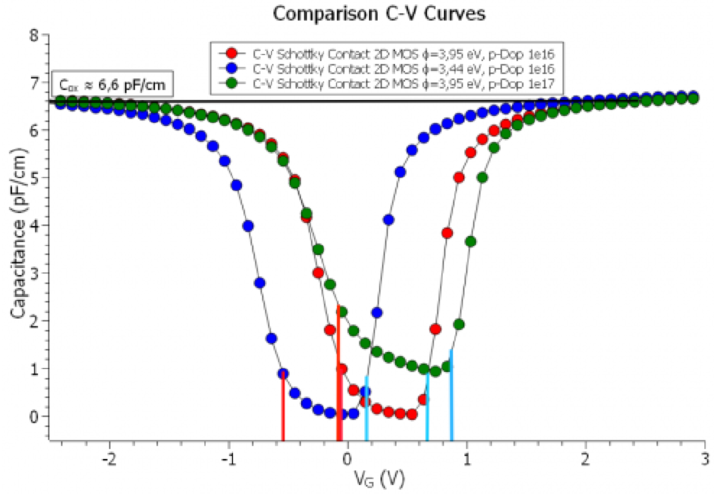


Figure 4.15.3.5: The comparison of the C-V characteristics of the 2D MOS for varying Schottky barrier and the substrate doping concentration, and their effects on the threshold voltage (vertical blue lines), and the flatband voltage (vertical red lines)

As it could be seen in the above figure Figure 4.15.3.5, both the barrier height and the doping concentration shift the threshold voltage V_{Th}, and the flatband voltage V_{FB}, however the flatband voltage is more affected by the barrier height rather than the doping concentration. It is also worth mentioning, that the doping concentration alone also affects the minimum capacitance in both low-frequency regime, and the high frequency regime, namely C_{min}, and $C^{'}_{\mathrm{min}}$, which are the bottom limits of the C-V curve ($C^{'}_{\mathrm{min}}$ is directly inversely related to the maximum depletion region width, and apparently so is the C_{min}).

In the next set of figures we see, how changing the charge integration region can affect the C-V curve, which then would answer why the C^{*}_{FB} in our original curve did not exactly match the calculated flatband capacitance C_{FB}. The following figure Figure 4.15.3.6, illustrates the effect of changing the charge integration region on the flatband capacitance C_{FB}:

And figure Figure 4.15.3.7 shows the C-V curve of the MOS capacitor for a charge integration region of $W_{\mathrm{int}} = 300\mathrm{nm}$:

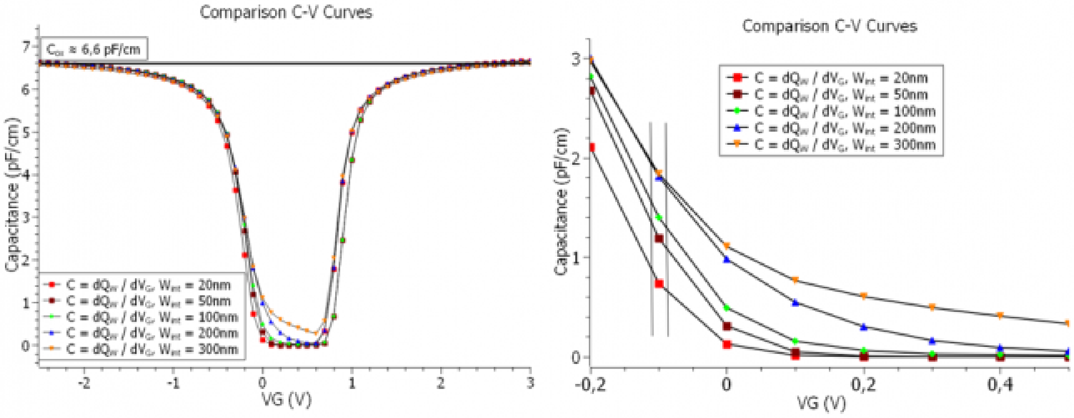


Figure 4.15.3.6: The comparison of the C-V characteristics of the 2D MOS for varying the width of the charge integration region.

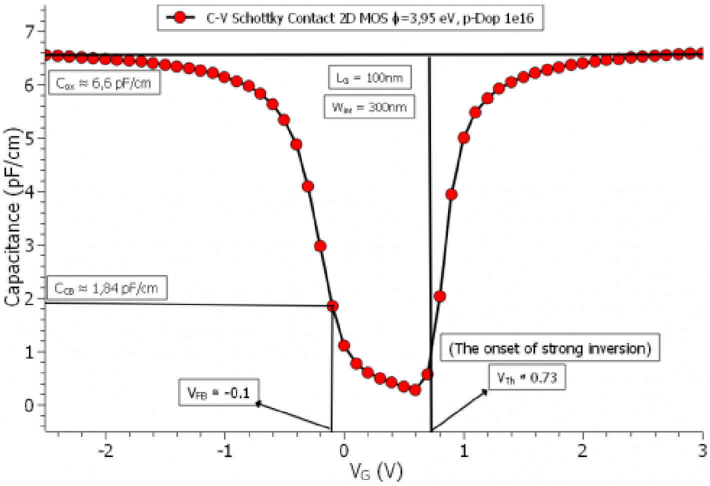


Figure 4.15.3.7: The comparison of the C-V characteristics of the 2D MOS for varying the width of the charge integration region.

Now it seems that the value of the flatband capacitance C_{FB} in the C-V curve ($1.84\mathrm{pF/cm}$) agrees very well with the calculated value. The reason for that is that, as mentioned in equation (4.15.3.2), the charge in the depletion region is directly proportional to the width of the depletion region. This width has a maximum which is given by:

$$W_{\mathrm{D},max} \approx \sqrt{\frac{2\epsilon_s\psi_s(\mathrm{strong\,inv.})}{qN_{\mathrm{A}}}} \approx \sqrt{\frac{2\epsilon_s k_{\mathrm{B}}T\ln(N_{\mathrm{A}}/n_i)}{q^2N_{\mathrm{A}}}} \tag{4.15.3.6}$$

which turns out to be $\approx 303\mathrm{nm}$ in our MOS capacitor. Therefore, it should be noted, that in order to be able to reach the flatband capacitance defined by the formalism, the charge integration region should be greater or equal to the maximum depletion region width $W_{\mathrm{D},max}$. Note that the charge carrier integration has to be specifically mentioned as a region with the following flags in the *<structure{ }_integrate>* group:

```
region{
    rectangle{ # Si Charge Carrier Integration Zone
       x = [-$L_Oxide_Ch/2 , :remove_enter:
       $L_Oxide_Ch/2]
       y = [-$H_Substrate, 0]
    }
    binary{
       name = "Si"
    }
 integrate{
       electron_density{}         # n-charge carriers
       hole_density{}             # p-charge carriers
       label = "Si_Substrate"
     }
 }
```

The total charge is then $q(-p_{\mathrm{tot}}+n_{\mathrm{tot}})$. The derivative of this charge with respect to the voltage bias sweep results in the C-V curve, as mentioned before.

Appendix: 2D MOS

The MOS capacitor is a 2D device in its correct form for simulations (with the optional 3rd dimension if need be…). The width of the substrate needs to be somewhat larger than the channel length, so that the depletion layer charges have enough space to expand, also the boundary conditions have to be set to non-periodic in the simulation. That is because even though the channel length is set by the length of the gate-contact, and the inversion layer is bounded by this length, this is not the case for the charges in the depletion layer. Figure Figure 4.15.3.8 illustrates this phenomenon:

If we set the substrate width to the length of the channel, which basically would mean that the MOS could also be simulated in 1D, the C-V curve would look like the following in figure Figure 4.15.3.9

As seen in the C-V curve, not only the oxide capacitance C_{ox} is somewhat less than what it should be, the flatband capacitance C_{FB} ($1.57\mathrm{pF/cm}$) does not agree, within an acceptable margin of error, with the calculated value.

With regards to the surface potential ψ_{s}, it is worth mentioning, that this potential can be measured by measuring the electrostatic potential at the semiconductor-oxide interface, as function of the gate-voltage. For that in *nextnano*++, one needs to perform a bias sweep at the gate-contact using the template, and make a 1D-section slice of the simulation in the *section{ }* group, mentioning a range in y-direction around $y = 0$, so that **exactly one** grid point falls within this range:

```
output{
  section1D{                 # output a 1D section of the simulation area (1D slice)
    name = "surface_potential"              # name of section enters file name
    x = 0
    range_y = [-0.2, 0.0]    # 1D slice at x = 0 through the middle of the channel
                                            # however limited to the range in y
  }
}
```

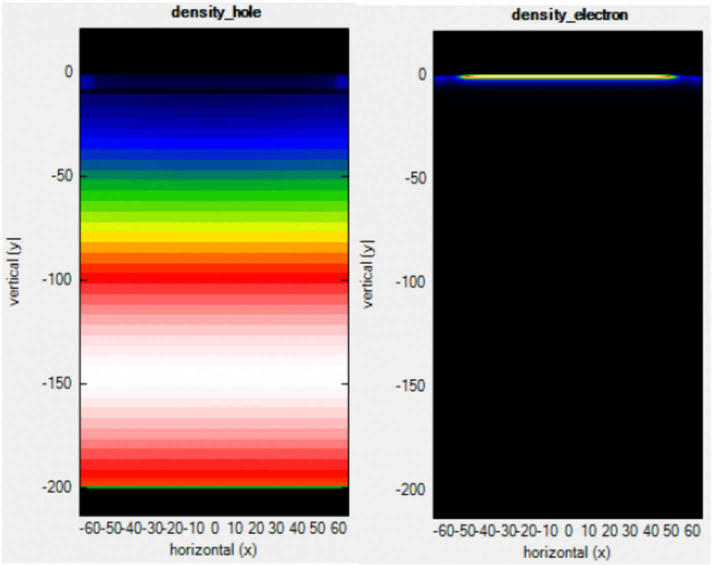


Figure 4.15.3.8: The spatial distribution of charge carriers (electrons) in the inversion layer during inversion, compared to the ones (holes) in the depletion region during depletion.

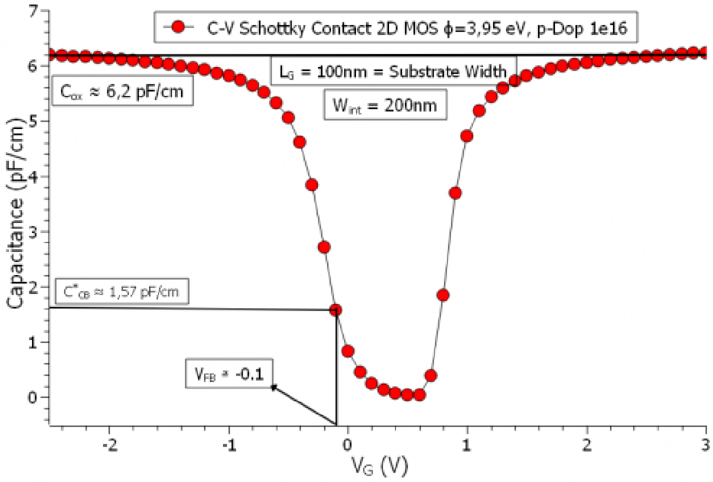


Figure 4.15.3.9: The C-V curve of the quasi 1-D Simulations of the MOS (this is when we set the length of the oxide and the channel-length equal in a 2D simulation and set the boundary condition in x-direction as periodic).

Using the post-processing in the template, one can then construct a curve, which should look like the one shown in figure Figure 4.15.3.10

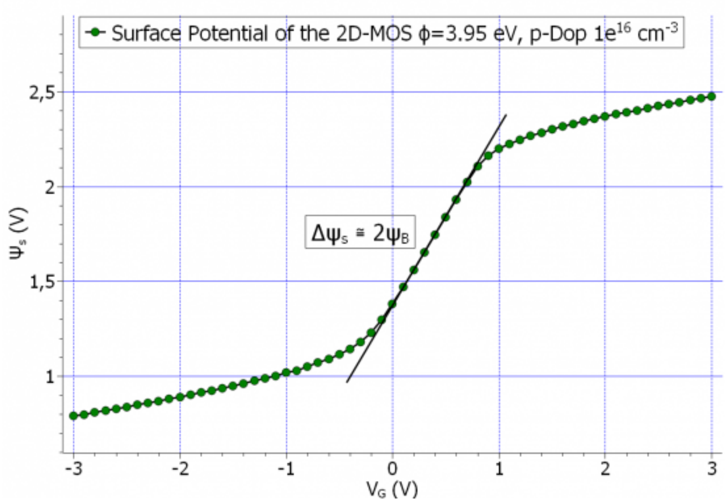


Figure 4.15.3.10: The surface potential, at the semiconductor-oxide interface ψ_s, as a function of the gate.voltage V_G

Such a curve would go through the origin for an ideal MOS device, however depending on how the electrostatic potential is calculated at the contacts, this curve could go higher or lower on the y-axis. The transition from accumulation to strong inversion of the total capacitance happens basically in the region of the potential, where the line is drawn, for which $\Delta\psi_s \approx 2\psi_B$.

The last remark regarding the capacitance of the MOS could be that, even though the classical formula of parallel plates capacitor is also here applied to the oxide capacitance, in small dimensions and in few nanometer regime, other effects such as tunneling current, and thermionic emissions could play a significant role. Additionally, since the quantum mechanical charge distribution distances itself from the semiconductor-oxide interface (as we shall see in the inversion layer comparison of the MOSFET), these effects would significantly reduce the maximum capacitance of the MOS. As we could see from the C-V curve the flatband capacitance is less than 30% of the oxide capacitance, even though one would expect that the C_FB be somewhere around 80% of the C_ox. Therefore if the aforementioned effects be taken under consideration, it could very well be that the C_ox fall to half of its parallel-plate value.

2D N-Ch MOSFET

Input files:

- *nMOSFET_2D_Dop-16-20_Schottky_noQM.in*
- *nMOSFET_2D_Dop-16-20_Schottky_QM.in*
- *nMOSFET_2D_Dop-16-20_Schottky_QM_decomposition.in*

The MOSFET is a transistor, which is made of a MOS capacitor in the middle and a source-drain channel for conduction. The channel of the MOSFET, which is probably the most important aspect of the MOSFET, extends from source to drain, and is created by a charge carrier inversion layer in the MOS. In this tutorial we simulate an **N-channel MOSFET** based on the proposed model in [Wu]. As parameters, we vary the oxide thickness, channel length and the doping profiles and investigate how these changes affect the simulation results. These quantities are defined as follows:

$d_\mathrm{oxide} = t_\mathrm{ox} = 5\mathrm{nm}$, $L_\mathrm{Ch} = 100\mathrm{nm}$, $N^+ = 10^{20}\mathrm{cm}^{-3}$, $P = 10^{16}\mathrm{cm}^{-3}$.

The overall geometry of the simulated N-Ch MOSFET in this tutorial is illustrated in the following figure Figure 4.15.3.11:

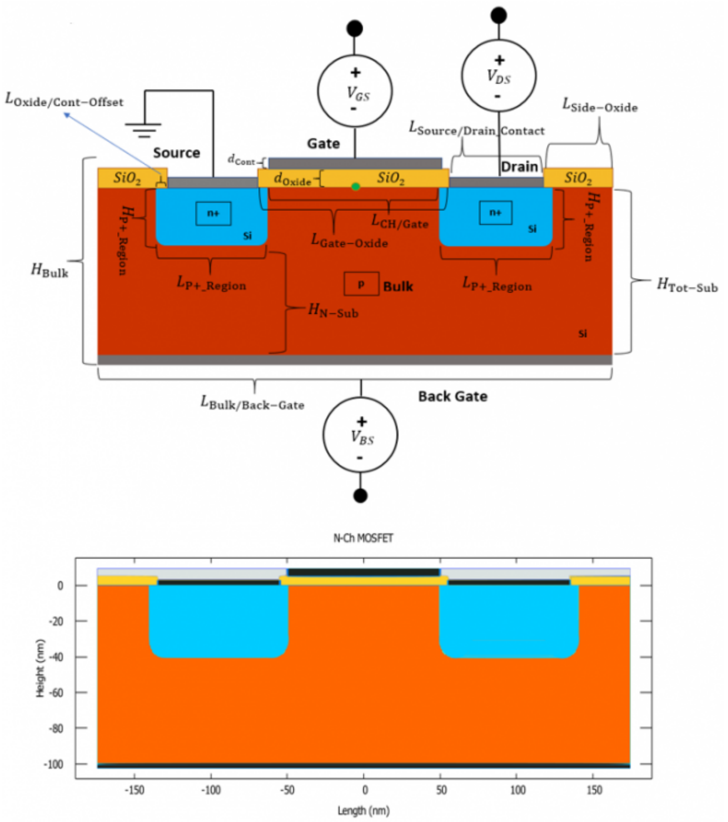


Figure 4.15.3.11: The geometry of the N-Ch MOSFET design, and its corresponding geometry from the output file user_index.vtr. The individual regions can also be found in the output file regions.vtr.

The drain-source current of the MOSFET is given by equation (4.15.3.7)

$$I_{\text{DS}} = \frac{W}{L}\mu_n^{\text{eff}} C_{\text{ox}} \left\{ (V_{\text{GS}} - V_{\text{Th}})V_{\text{DS}} - \left[\frac{1}{2} + \frac{\sqrt{4\epsilon_{\text{Si}} q N_{\text{Sub}} \psi_{\text{B}}}}{C_{\text{ox}}}\right] V_{\text{DS}}^2 \right\} \tag{4.15.3.7}$$

where the threshold voltage V_{Th} is the same threshold voltage for the MOS as defined in equation (4.15.3.4). For the limit of $V_{\text{DS}} \ll (V_{\text{GS}} - V_{\text{Th}})$ equation (4.15.3.7) reduces to:

$$I_{\text{DS}} = \frac{W}{L}\mu_{\text{n}}^{\text{eff}} C_{\text{ox}} (V_{\text{GS}} - V_{\text{Th}} - \frac{V_{\text{DS}}}{2}) V_{\text{DS}} \tag{4.15.3.8}$$

For the **input characteristics**, this equation becomes a function of the gate voltage V_{GS} with the drain-source voltage V_{DS} kept constant. For the **output characteristics**, however, this current becomes a function of the drain-source voltage at constant gate voltage. (But rather for a set of gate voltages.) As can be seen the current is directly proportional to the effective mobility μ^{eff}, and the oxide capacitance of the MOS capacitor C_{ox}. Note that C_{ox} is the oxide capacitance per unit area in 3D (and per channel length in 2D), and therefore has the units of $F/(\text{length})^2$.

Input Characteristics

Using the Masetti mobility model, we have calculated the input characteristics of the MOSFET classically, which is shown in figure Figure 4.15.3.12 on a linear scale,

and in figure Figure 4.15.3.13, on a logarithmic scale:

The above input characteristics were calculated without the shift in the drain contact. This could modify the results in a certain way that is worth noting. More on this could be found in the *Appendix: MOSFET*. According to [Sze], the extrapolation of the linear region meets the x-axis at $V_{\text{Th}} + \frac{V_{\text{D}}}{2}$. Having set the V_{DS}, to 0.2V, for the calculation of the input characteristics, the value is very well expected to be $\approx 0.8\text{V}$, since the threshold voltage V_{Th} was calculated to be $\approx 0.7\text{V}$. However we also used a small backgate bias $V_{\text{BS}} = -0.1\text{V}$ in the above

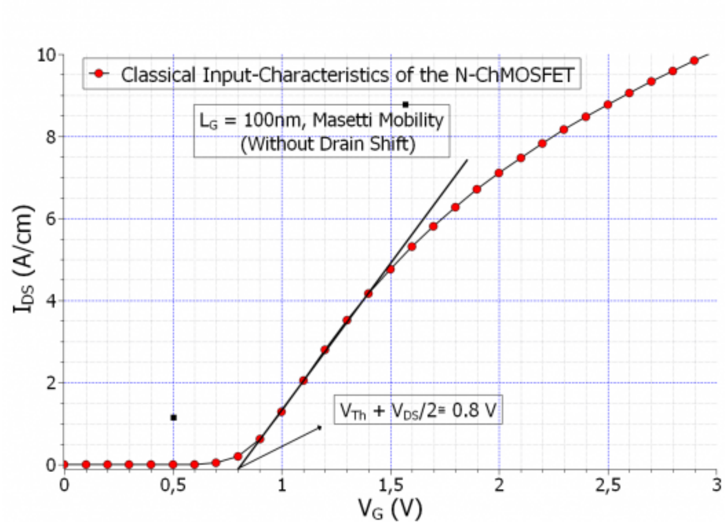


Figure 4.15.3.12: The input characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the position of the threshold voltage V_{Th}.

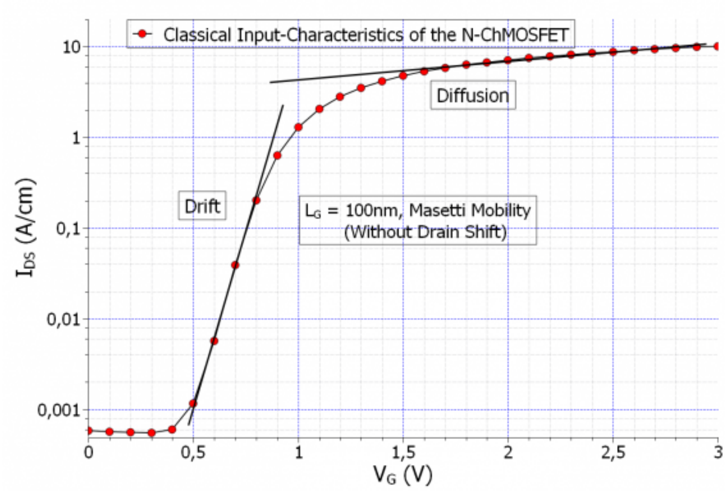


Figure 4.15.3.13: The input characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the drift and diffusion current regions on the logarithmic scale.

calculations, which slightly modifies the threshold voltage, by changing the voltage drop in the oxide to,

$$V_{\mathrm{ox}} = \frac{\sqrt{2\varepsilon_{\mathrm{Si}} q N_{\mathrm{Sub}}(2\psi_{\mathrm{B}} - V_{\mathrm{BS}})}}{C_{\mathrm{ox}}} \approx 0.08\mathrm{V},$$

compared to $V_{\mathrm{ox}} = 0.073\mathrm{V}$. However the difference is negligible in our case. Note that the above input characteristics were calculated without a shift in the drain contact. This can also modify the results to a certain degree as explained in the *Appendix: MOSFET* section.

However the input characteristics could also be calculated quantum mechanically, since we only have to define the inversion layer region as a quantum region. The prediction is that the charge carrier inversion layer would shift slightly away from the oxide, since the wave function amplitude would have to fall to zero at the oxide-semiconductor interface. This phenomenon is illustrated in figure Figure 4.15.3.14

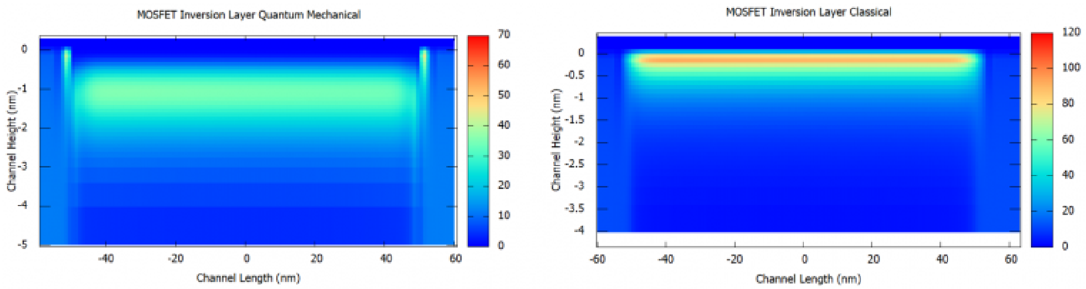


Figure 4.15.3.14: The comparison of the charge inversion layer of the N-Ch MOSFET calculated classically (right), and quantum mechanically (left) at $V_{\mathrm{GS}} > V_{\mathrm{Th}}$ and $V_{\mathrm{DS}} = 0.2\mathrm{V}$.

The following set of curves in figure Figure 4.15.3.15 are the comparison of the input characteristics calculated classically and quantum mechanically, with and without *quantum decomposition* method:

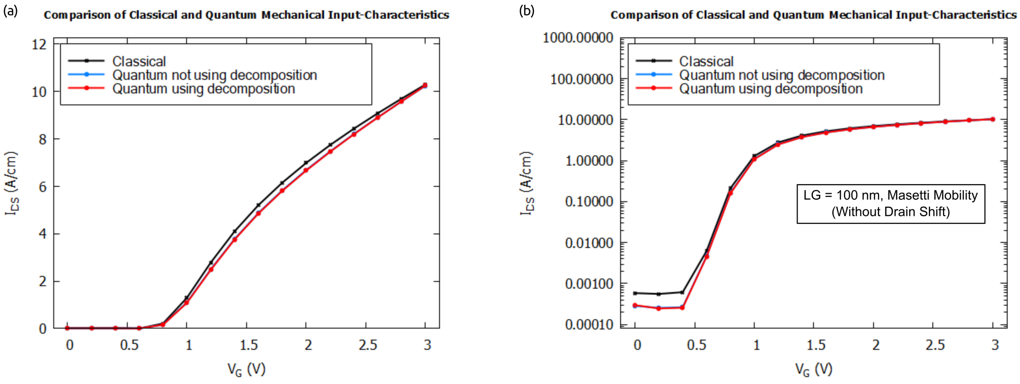


Figure 4.15.3.15: The comparison of the input characteristics of the MOSFET calculated classically and quantum mechanically wit (a) linear and (b) logarithmic scales.

As the simulations shows, there is a slight difference in the input characteristics, most importantly for the leakage current, the one below the threshold voltage. It turns out to be lower for the quantum mechanical input characteristics. Moreover, comparison above shows that using the *quantum decomposition* method triggered by a keyword **quantize_y{}** gives almost the same IV curves as in the case of solving the Schrödinger equation in 2D while notably reducing time and memory required for the computation.

Output Characteristics

The output characteristics of the MOSFET is the I-V characteristics of the drain current I_{DS} vs. the source drain voltage V_{DS}, for certain constant gate voltage. Therefore the output characteristics could be viewed as a double sweep, and considering the total simulation time, it is a heavy load on the simulator. With that in mind its worth mentioning that the issue of convergence becomes very important for the output characteristics, in the sense that if the simulation parameters are not chosen correctly the simulations may never converge. More on that could be found in the *Appendix: MOSFET*. The output characteristics of the MOSFET calculated with the Masetti mobility are shown in figure Figure 4.15.3.16:

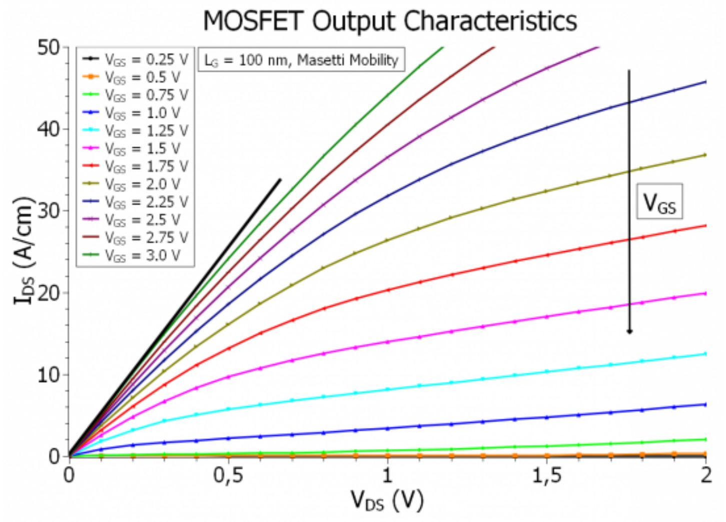


Figure 4.15.3.16: The output characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the linear and the saturation regions of the output characteristics.

The slope of the black line which covers the linear region of all the curves, can be used to calculate the channel specific resistivity. Now, if we take the width of the MOSFET to be $15\mathrm{nm}$, the output characteristics could be expressed in Amperes, as shown in figure Figure 4.15.3.17:

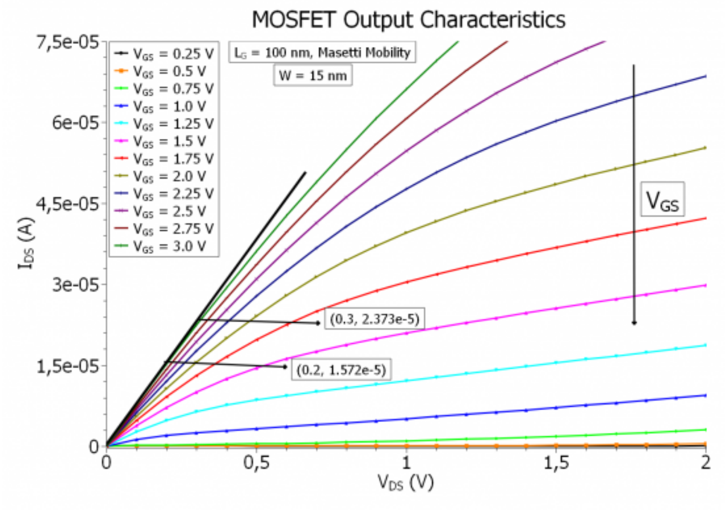


Figure 4.15.3.17: The output characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the linear and the saturation regions of the output characteristics for a width of $15\mathrm{nm}$.

From the readings on the curve we can estimate the specific channel resistivity,

$$\frac{1}{R_{\text{specific}}} = \frac{L}{W}\frac{I_{\text{DS}}}{V_{\text{DS}}} \rightarrow R_{\text{specific}} = 1.8\mathrm{k}\Omega.$$

As mentioned before, the output characteristics can be divided into two regions, the ohmic region and the saturation region. The transition to the saturation region happen at the $V_{\text{DS,sat}}$, which is give by equation (4.15.3.9):

$$V_{\text{DS,sat}} = \frac{V_{\text{GS}} - V_{\text{Th}}}{M}, M = 1 + \frac{K}{2\sqrt{\psi_{\text{B}}}}, K = \sqrt{\varepsilon_s q N_{\text{A}}}/C_{\text{ox}} \tag{4.15.3.9}$$

This value obviously is meaningful for $V_{\text{GS}} > V_{\text{Th}}$, as it is zero for $V_{\text{GS}} = V_{\text{Th}}$, and the M factor is a dimensionless factor equal to ≈ 1.051 for our system. The saturation current is then defined as the current that is measured at

$V_{\mathrm{DS,sat}}$, for each V_{GS} as defined in equation (4.15.3.10):

$$I_{\mathrm{DS,sat}} = \frac{W}{2ML}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}} (V_{\mathrm{GS}} - V_{\mathrm{Th}})^2 = \frac{WM}{2L}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}} V_{\mathrm{DS,sat}}^2 \tag{4.15.3.10}$$

and plotting this current over the output characteristics, the curve crosses each I_{DS}, exactly at the corresponding $V_{\mathrm{DS,sat}}$ for that output current, as shown in figure Figure 4.15.3.18

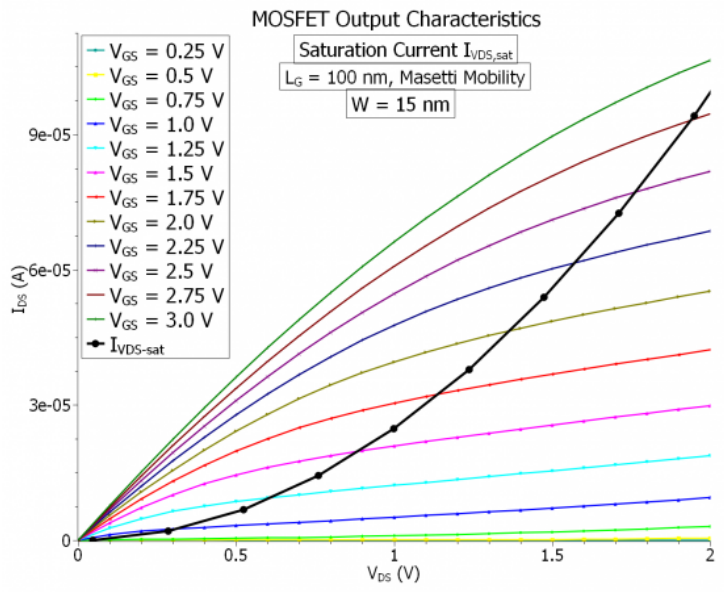


Figure 4.15.3.18: The output characteristics of the N-Ch MOSFET calculated classically with Masetti mobility for a width of $15\mathrm{nm}$, and the saturation current $I_{\mathrm{DS,sat}}$ plot.

If we take the effective mobility to be *field-independent* (which is the case in our simulations), the above $I_{\mathrm{DS,sat}}$ curve could be fitted with $I_{\mathrm{DS,sat}} = A \cdot V_{\mathrm{DS,sat}}^2$ formula, where A is estimated at $A \approx 2.475 \cdot 10^{-5}$. Note that, the quadratic curve does not meet the output current curves at the points, where they are supposed to meet (at $V_{\mathrm{DS,sat}}^2$ voltages), because, as we can see, the output charateristic curves do not really saturate after drain source voltage reaches $V_{\mathrm{DS,sat}}$. This is due to a short channel effect called drain-induced barrier lowering (or punch-through), which we will talk about in last section. When this effect diminishes (as we shall see), the quadratic curve meets the output-curves exactly at the saturation voltage point $V_{\mathrm{DS,sat}}$.

From the fit parameter estimate, and the rest of the known parameters, we can however estimated the effective mobility μ_n^{eff} independent of the field for the short channel case in an approximate way (and compared it later on with the long-channel variant). Taking the oxide capacitance to be $C_{\mathrm{ox}} \approx 6.6\mathrm{mF/m}^2$, the effective mobility of the electrons is then estimated to be

$$\mu_n^{\mathrm{eff}} \approx 525 \frac{\mathrm{cm}^2}{\mathrm{V \cdot s}},$$

The calculated bulk mobility from the simulations is given to be $\approx 933\mathrm{cm}^2/\mathrm{Vs}$ in the p-doped substrate, and $\approx 567\mathrm{cm}^2/\mathrm{Vs}$ at $y = 0$, which is the semiconductor-oxide interface.

Transconductance and Channel Conductance

In many cases, a MOSFET is used for signal amplification, as opposed to switching function, which is the case in CMOS, and digital logic circuits. For this purpose quantities such as transconductance and channel-conductance become important. The transconductance is defined as the derivative of the output current I_{DS} with respect to the gate voltage V_{GS}, for a constant source-drain voltage V_{DS}:

$$g_m = \left.\frac{\partial I_{\mathrm{DS}}}{\partial V_{\mathrm{GS}}}\right|_{V_{\mathrm{DS}}=\mathrm{const.}}$$

Figure Figure 4.15.3.19 shows the tranconductance curve and its maximum value:

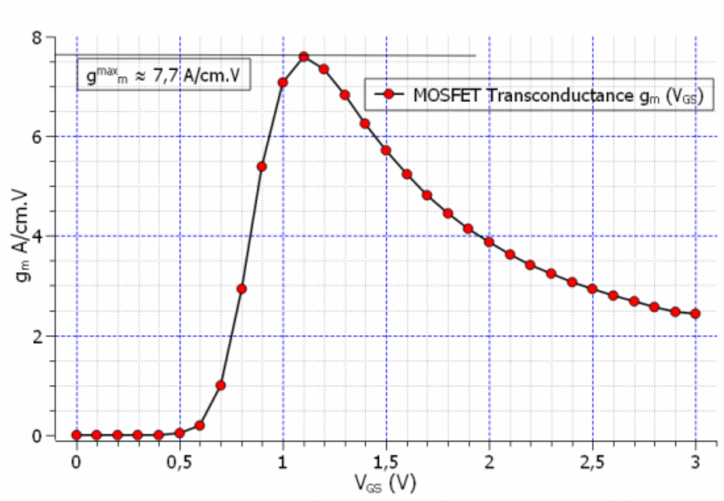


Figure 4.15.3.19: The transconductance of the MOSFET as a a derivative of the source-drain current I_{DS} with respect to the gate voltage V_{GS}.

The maximum value of the transconductance read from the curve amounts to $\approx 7.7 \mathrm{A/Vcm}$. However, it could also be calculated manually using the equation (4.15.3.11), since we now know the value of the effective mobility:

$$g_m = \left.\frac{\partial I_{\mathrm{DS}}}{\partial V_{\mathrm{GS}}}\right|_{V_{\mathrm{DS}}=\mathrm{const.}} = \frac{W}{L}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}} V_{\mathrm{DS}} \tag{4.15.3.11}$$

which amounts to $\approx 7.9 \mathrm{A/Vcm}$ for an eliminated W ($W = 1$). In contrast we have the channel conductance, which is the derivative of the source-drain current I_{DS} with respect to the source drain voltage V_{DS}, at a constant gate voltage V_{GS},as defined in equation (4.15.3.12):

$$g_{\mathrm{D}} = \left.\frac{\partial I_{\mathrm{DS}}}{\partial V_{\mathrm{DS}}}\right|_{V_{\mathrm{GS}}=\mathrm{const.}} = \frac{W}{L}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}} (V_{\mathrm{GS}} - V_{\mathrm{Th}}) \tag{4.15.3.12}$$

which is in turn a function of the gate voltage V_{GS}. Figure Figure 4.15.3.20 illustrates this conductance for a set of gate voltages:

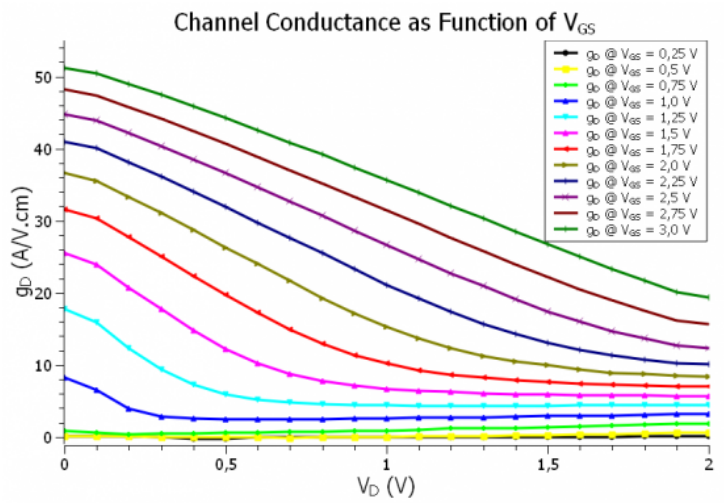


Figure 4.15.3.20: The channel conductance of the MOSFET as a derivative of the source-drain current I_{DS} with respect to the source-drain voltage V_{DS}.

Note that all of the curves in the above figure are from the same family. they are only stretched and displaced with respect to each other since the arguement $(V_{\mathrm{GS}} - V_{\mathrm{Th}})$ acts as a displacement and multiplication factor for the curves for each V_{GS}.

Finally we have for $V_{\mathrm{DS}} \geq V_{\mathrm{DS,sat}}$, the saturation transconductance which is derivative of the quadratic current output equation I_{DS} in (4.15.3.13) with respect to V_{GS}:

$$g_m = \frac{\partial I_{\mathrm{DS}}}{\partial V_{\mathrm{GS}}}\bigg|_{V_{\mathrm{DS}} \geq V_{\mathrm{DS,sat}}} = \frac{W}{ML}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}}(V_{\mathrm{GS}} - V_{\mathrm{Th}}) \qquad (4.15.3.13)$$

which would be straight line with respect to V_{DS}, and V_{GS}.

Comparison of Different Mobility Models

The effect of the correct mobility model for the simulations of such devices as MOSFETs cannot be overstated. It is an established fact, that the best mobility models used for simulating the current transport in the channel are those that are field dependent, and therefore are modified along the channel as a result of the perpendicular (and also parallel) field. The simplest of these models is the velocity saturation model which sets a maximum value for the drift velocity as the function of the field, and with that the mobility is limited by the maximum velocity. There are of course also more complicated models such as the **enhanced Lombardi** model, or **inversion layer mobility** models, which also take into account the scattering of the charge carriers at the semiconductor-oxide interface. These are very specialized models, specifically designed for the simulation of such devices as MOSFETs, and other field effect devices, and are implemented in specialized commercial TCAD tools used by industry. Here we are limited to the already implemented mobility models, which hopefully in the near future will expand. These are the **Masetti** model, **Arora** model, **Minimos** model, and **constant** mobility model. Figure Figure 4.15.3.21 illustrates the effect of different mobility models on the input characteristics of the MOSFET:

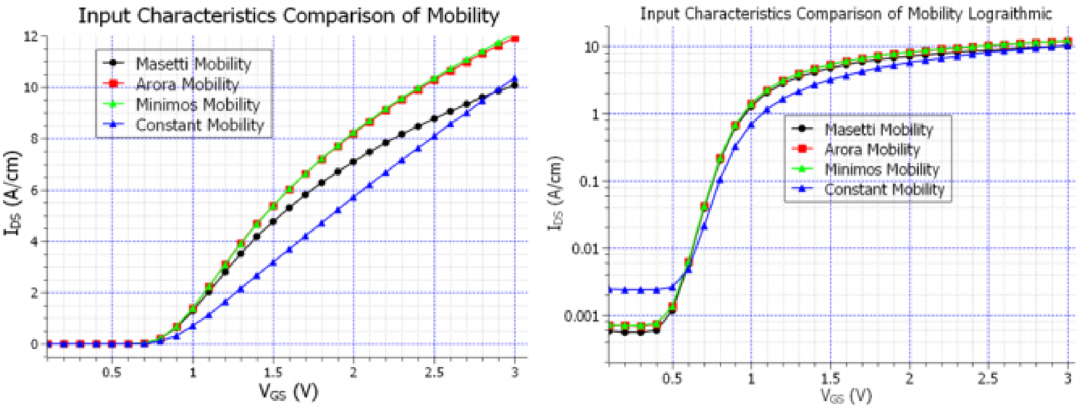


Figure 4.15.3.21: The input characteristics of the MOSFET calculated classically with different mobility models, in normal and logarithmic scales.

In the above curves, interestingly enough the Masetti model seems to reach the saturation much sooner than the other ones, and the constant mobility model seems to be a straight line, even though the value of the constant mobility is much lower in the inversion layer than the rest of the mobility models ($460\mathrm{cm}^2/\mathrm{Vs}$ compared to $900 - 1000\mathrm{cm}^2/\mathrm{Vs}$). The reason for that is that the constant mobility model defines the same electron mobility in the inversion layer, which is a p-doped region, as well as in the source and drain contact regions, which are heavily n-doped regions, whereas the other doping dependent mobility models have significantly different values for these regions, and the fact is that, in order for the current to flow, it must reach the contacts, which are the heavily n-doped regions. That is why the constant mobility produces a different input characteristics curve than the other mobility models. Also regarding the Masetti model, the reason that this model reaches the saturation faster could be attributed to the ratio of the mobility in the p-doped region with respect to the n-doped region, which for the Masetti model is ≈ 12, while it is ≈ 10 for the Minimos and Arora models. Obviously, this ratio is 1 for the constant mobility model.

The following figure Figure 4.15.3.22 shows the output characteristics calculated with the constant mobility model set at $\mu_0 \approx 460\mathrm{cm}^2/\mathrm{Vs}$:

We can now compare this to the Masetti mobility as the example of doping dependent models. Figure Figure 4.15.3.23 shows the comparison for a selection of the V_{GS} values:

As the curves suggest, the difference is negligible for very high and very low gate voltages. The difference becomes significant only for $1.5 \leq V_{\mathrm{GS}} \leq 2.5\mathrm{V}$.

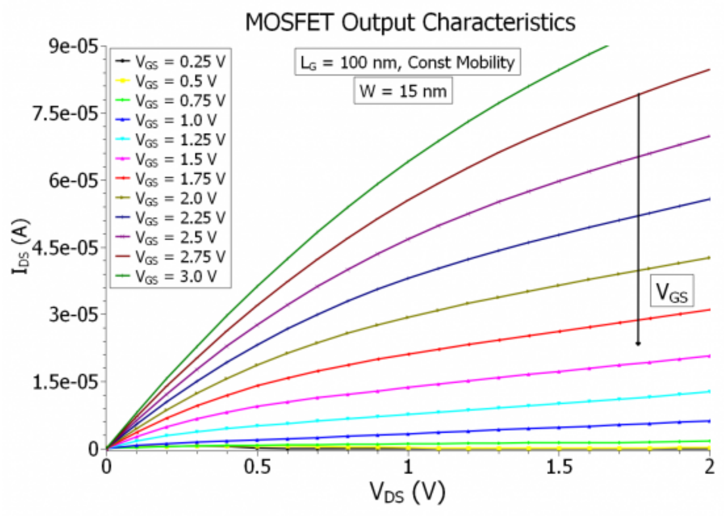


Figure 4.15.3.22: The output characteristics of the MOSFET calculated classically with the constant mobility model, taking the width W to be $15\mathrm{nm}$.

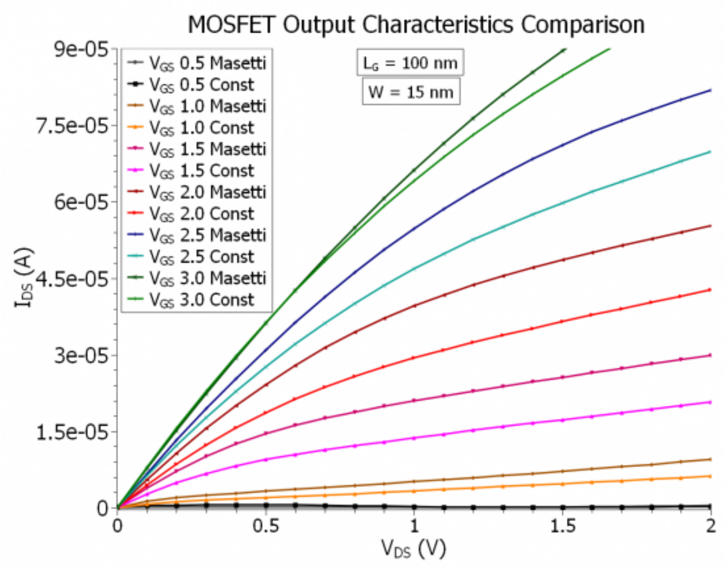


Figure 4.15.3.23: The comparison of the output characteristics of the MOSFET calculated classically with constant mobility and Masetti models, for a selection of gate voltages, and the width $W = 15\mathrm{nm}$.

Furthermore, it is worth mentioning, that a good mobility model for the inversion layer in the MOSFET should have two field dependencies, one being the perpendicular field originating from the gate, and the other one the parallel field coming from the source-drain bias. The velocity saturation method, which has recently been implemented would only have one of these components, namely the parallel field dependency, and since it is still at the experimental level, we did not put any results simulated with that. However the implementing velocity saturation would have a distinguishable effect on the output characteristics of the short channel MOSFET.

Channel Length Modulation and Pinch-Off effect

- *nMOSFET_2D_Dop-16-20_Schottky_Class_VG-2.0_Pinch-off.in*

One last effect that is worth talking about in the context of the output characteristics, is the pinch-off effect, i.e. the effective shortening of the channel length, which is known as the channel length modulation. It is said that the pinch-off effect steps in at the onset of saturation $V_{\mathrm{DS}} \approx V_{\mathrm{DS,sat}}$. Figure Figure 4.15.3.24 shows the electron density along the channel for 3 different source-drain voltages ($V_{\mathrm{DS}} = 0.0\mathrm{V}$, $V_{\mathrm{DS}} = 0.6\mathrm{V}$, $V_{\mathrm{DS}} = 1.5\mathrm{V}$) at a fixed gate voltage $V_{\mathrm{GS}} = 2.0\mathrm{V}$:

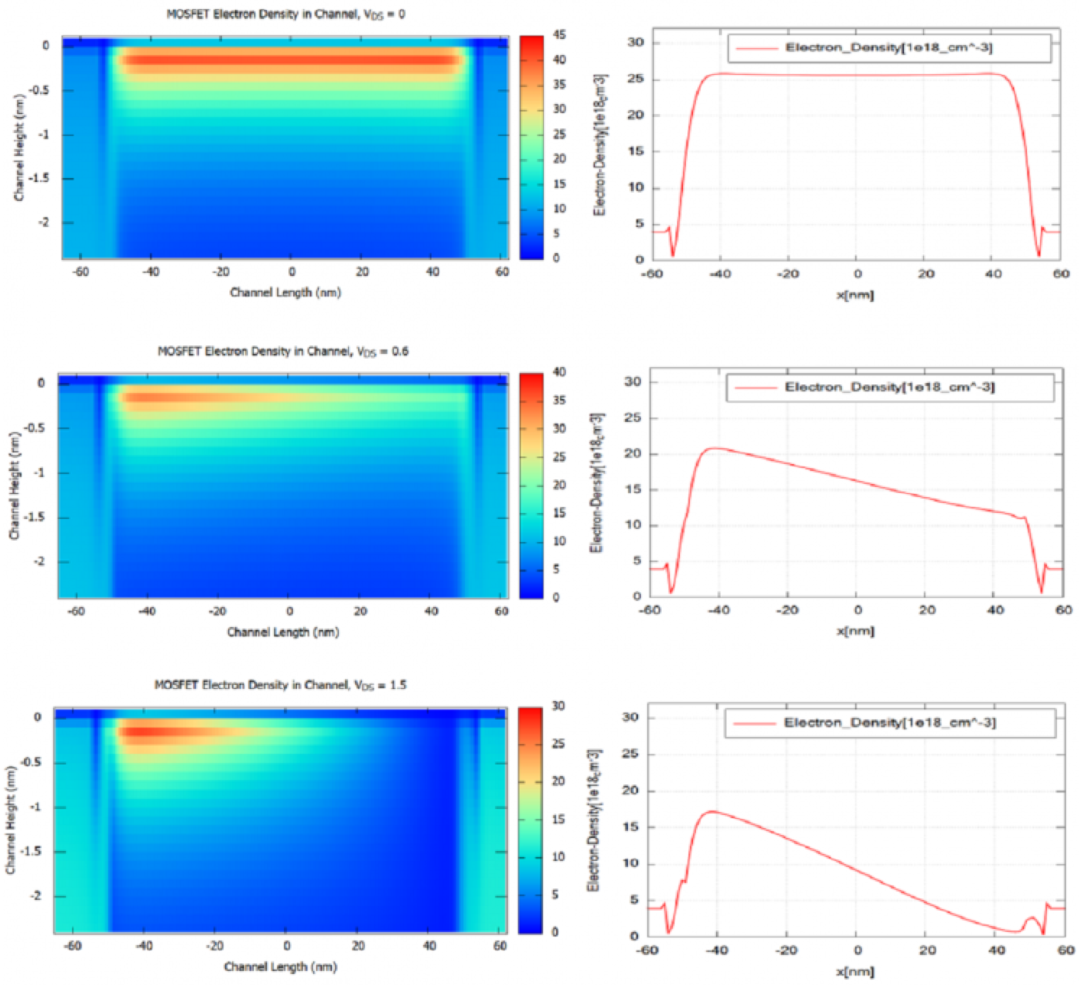


Figure 4.15.3.24: The comparison of the electron density distribution in the channel for $V_{\mathrm{DS}} = [0.0, 0.6, 1.5]\mathrm{V}$ at the gate voltage of $V_{\mathrm{GS}} = 2.0\mathrm{V}$, showing the pinch-off effect and the effective channel shortening. The 3 pictures of the left show the electron density n(x,y) which is contained in the file *density_electron.vtr*. The 3 pictures of the right show the content of the file *density_electron_1d_middle_line_x_direction.dat* which contains a slice along the x direction for constant y value where y lies in the channel for the pictures on the left.

Then the saturation current equation takes the following form:

$$I_{\mathrm{DS,sat}} = \frac{W}{2ML}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}} V_{\mathrm{DS,sat}}^2 (1 + \lambda V_{DS})$$

with $\lambda \approx \Delta L/L \cdot V_{\mathrm{DS}}$. However this is not an analytical approach, and can possibly lead to inconsistencies. There is a more precise way to calculate the effective channel length, if we take into consideration the depletion widths of the source and drain under potential difference. Figure Figure 4.15.3.25 illustrates these depletion widths:

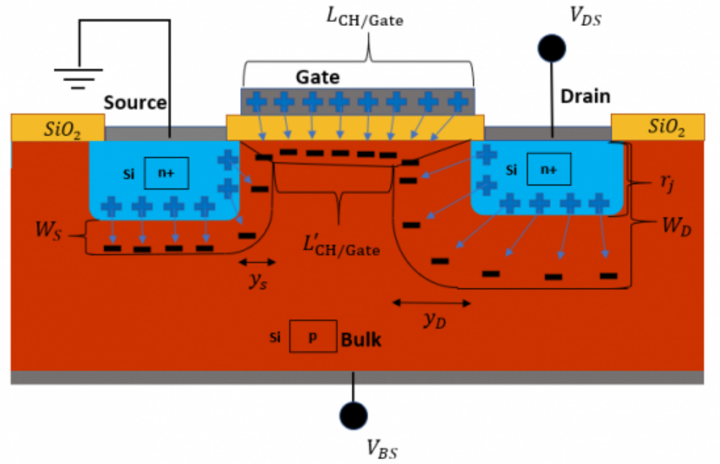


Figure 4.15.3.25: The illustration of the shortening of the effective channel length due to the expansion of the drain and source depletion widths.

Using the built-in potential of the p-n junction at the source and drain $\psi_{\mathrm{bi}} \approx 0.9\mathrm{V}$, and the surface potential $\psi_{\mathrm{s}} = 2\psi_{\mathrm{B}} \approx 0.713\mathrm{V}$, we can estimate the length of the effective channel, taking the depletion widths to be approximately equal to y_{S} and y_{D} for source and drain, within the inversion layer (meaning that the widths also include the surface potential at the semiconductor-oxide interface), as defined in equation (4.15.3.14),

$$y_{\mathrm{S}} \approx \sqrt{\frac{2\varepsilon_{\mathrm{s}}}{qN_{\mathrm{A}}}(\psi_{\mathrm{bi}} - \psi_{\mathrm{s}} - V_{\mathrm{BS}})}, y_{\mathrm{D}} \approx \sqrt{\frac{2\varepsilon_{\mathrm{s}}}{qN_{\mathrm{A}}}(\psi_{\mathrm{bi}} + V_{\mathrm{D}} - \psi_{\mathrm{s}} - V_{\mathrm{BS}})}. \tag{4.15.3.14}$$

From which then results the effective channel length (as also illustrated in figure Figure 4.15.3.25), as follows:

$$L_{\mathrm{eff}} = L^{'} = L - y_{\mathrm{S}} - y_{\mathrm{D}}$$

However, this analysis has an indirect implication with regards to the channel length. Namely, for given source and drain depletion regions there is a minimum channel length. And indeed there is such a consideration, which is said to separate the long channel scenario from the short channel one, meaning a channel above this minimum length is considered a long channel (and not a short channel), and the above considerations apply only to long channel MOSFETs. As we will later see there are also other effects and considerations which will apply to the case of short channels (together known as the **short channel effects**). The minimum channel length for the long channel case is then given by the following empirical formula in (4.15.3.15),

$$L_{\min} = C\Big[r_j d_{\mathrm{ox}}(W_{\mathrm{S}} + W_{\mathrm{D}})^2\Big]^{1/3}, \tag{4.15.3.15}$$

where C is a constant, and W_{S} and W_{D} are the depletion widths of source and drain,

$$W_S = \sqrt{\frac{2\epsilon_s}{qN_A}(\psi_{\mathrm{bi}} - V_{\mathrm{BS}})}, W_{\mathrm{D}} = \sqrt{\frac{2\varepsilon_{\mathrm{s}}}{qN_{\mathrm{A}}}(\psi_{\mathrm{bi}} + V_{\mathrm{D}} - V_{\mathrm{BS}})}. \tag{4.15.3.16}$$

If we take $V_{\mathrm{D}} = 0.2\mathrm{V}$, then we have $W_{\mathrm{S}} = 359\mathrm{nm}$, and $W_{\mathrm{D}} = 393\mathrm{nm}$, while for the same $V_{\mathrm{D}} = 0.2\mathrm{V}$, the $y_{\mathrm{S}} = 192\mathrm{nm}$, $y_{\mathrm{D}} = 198\mathrm{nm}$. It makes sense to claim, that a negative effective channel length makes no sense, therefore $L_{\min} \geq y_{\mathrm{S}} + y_{\mathrm{D}}$. In [Brews] it is mentioned, that the constant C for device parameters of: $d_{\mathrm{ox}} = 25\mathrm{nm}, r_j = 330\mathrm{nm}, N_{\mathrm{A}} = 10^{14}\mathrm{cm}^{-3}, V_{\mathrm{DS}} = 1\mathrm{V}, V_{\mathrm{BS}} = 0$, through single point fitting, was measured to be $0.41\mathrm{A}^{1/3}$. For this value of the constant, our $L_{\min}$ would have to be $198\mathrm{nm}$, which is almost half the value of $y_{\mathrm{S}} + y_{\mathrm{D}}$. However, for a value of $C = 0.8\mathrm{A}^{1/3}$, we would have a $L_{\min} = 390\mathrm{nm}$. Though if we take the fact, that we increase our drain source voltage all the way to $V_{\mathrm{DS}} = 2.0\mathrm{V}$, then y_{D} would go as high as $540\mathrm{nm}$. Then it would be safe to claim, that we need our channel to be at least $\approx 600\mathrm{nm}$. Now let us examine the consistency of the y_{S}, and y_{D} values, for a channel length of $L = 2000\mathrm{nm}$. The following figure Figure 4.15.3.26 illustrates the pinch-off effect and channel length modulation in the same MOSFET model with a $L_{\mathrm{G}} = 2\mu\mathrm{m}$:

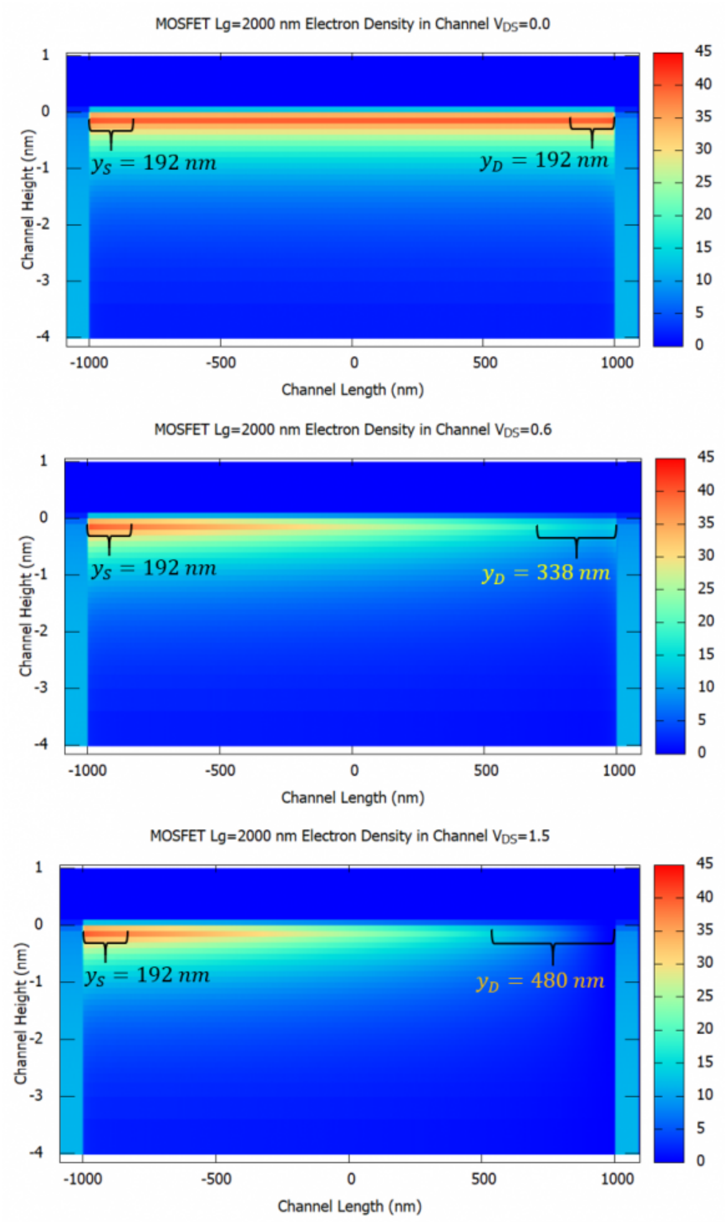


Figure 4.15.3.26: The illustration of the pinch-off effect, and the channel length modulation, in the N-Ch MOSFET with a channel length of $L_\mathrm{G} = 2\mu m$, calculated classically.The depletion widths at the source and drain, y_S and y_D, estimated from the analytical formulas given above, are indicated.

So therefore, according to the calculations in figure Figure 4.15.3.26, the effective channel length should be $L_{\mathrm{eff}} \approx 1330\mathrm{nm}$. Furthermore, it seems that the effects at the boundaries are not compatible with the calculations. However, the shortening of the boundaries due to the applied voltage at the drain is somehow in line with the depletion length y_{D}.

Short Channel Effects, DIBL and Punch-Through

So as we established in the previous section, our MOSFET, with a $100\mathrm{nm}$ channel, length would be below the long channel limit, and therefore would experience short channel effects. The most important of these effects is known as the drain induced barrier lowering (DIBL), which causes the injection of extra charge carriers, resulting in the increasing of the output current even after the saturation $I_{\mathrm{DS,sat}}(V_{\mathrm{DS,sat}})$. This phenomenon is known as the punch-trough effect and is present in our output characteristics in figures Figure 4.15.3.16 and Figure 4.15.3.17 of the output characteristics section. The DIBL effect is shown in figure Figure 4.15.3.27, comparing two channel lengths:

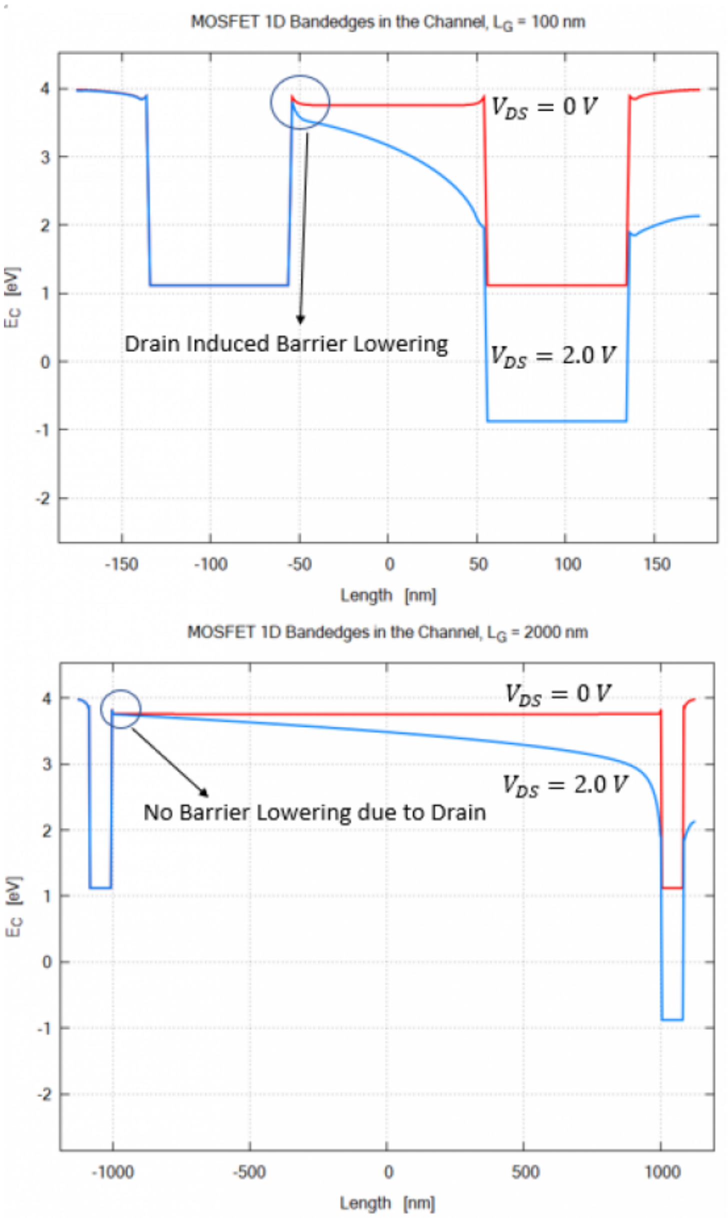


Figure 4.15.3.27: The illustration of the drain induced barrier lowering (DIBL) in $100\mathrm{nm}$ gate-length MOSFET, compared to the $2000\mathrm{nm}$ gate-length variant (where there are no barrier lowering).

In order to recognize the punch-through effect, the sweep of the gate-length should be performed at high drain-source voltages (for example $V_{\mathrm{DS}} = 2.0\mathrm{V}$) with the input characteristics on a logarithmic scale, which then show if the drift current is limited due to the gate length of the MOSFET. Figure Figure 4.15.3.28 shows this effect:

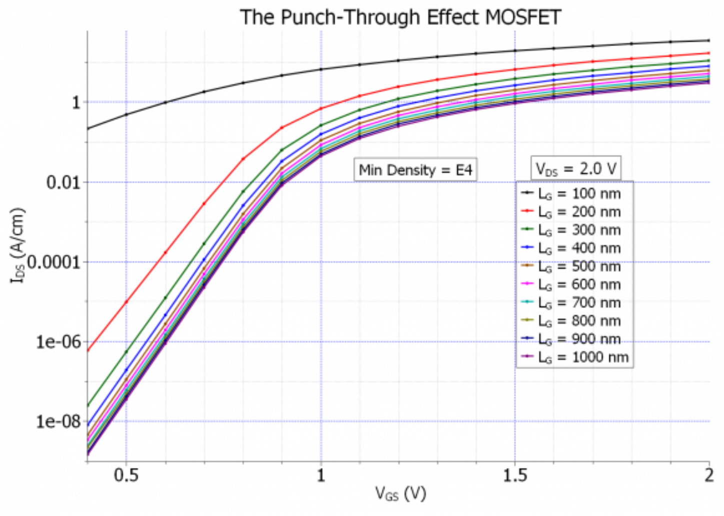


Figure 4.15.3.28: The punch-through effect for a set of channel lengths in MOSFET apparent in the input characteristics (calculated with minimum density of $10e4$).

As it could be seen in Figure 4.15.3.28, the MOSFET with gate-length of $L_G \leq 400\mathrm{nm}$ would definitely suffer from the punch-through effect. However, one could be safe with a channel length of $500\mathrm{nm}$ or $600\mathrm{nm}$. Let us now examine the effect of channel length on the normal input characteristics, namely at low drain source voltage. Using the Masetti mobility, the effect of increasing the channel length is illustrated in figure Figure 4.15.3.29:

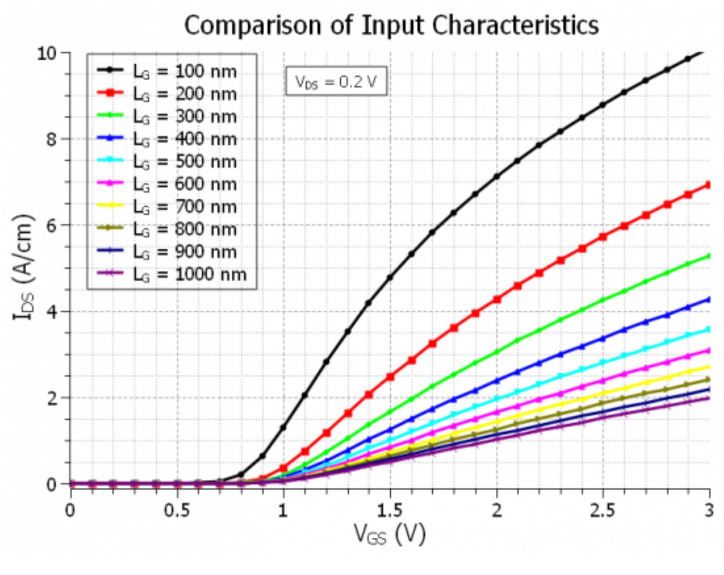


Figure 4.15.3.29: The effect of increasing the channel length on the input characteristics at $V_{\mathrm{DS}} = 0.2\mathrm{V}$.

So therefore we expect, that our input characteristics will be the same for a channel length of $400\mathrm{nm}$ or above using any of the mobilities (Masetti, or constant, or any other), as long as there is no field-dependent saturation in the mobility model. In the following figure Figure 4.15.3.30 let us estimate the threshold voltage for an ideally long channel MOSFET variant ($L_{\mathrm{G}} = 600\mathrm{nm}$):

From which it could be concluded, that the threshold voltage is $V_{\mathrm{Th}} \approx 0.87\mathrm{V}$. Consequently the output characteristics for the $L_{\mathrm{G}} = 600\mathrm{nm}$ MOSFET is shown in figure Figure 4.15.3.31:

As we can see in the above figure, the quadratic curve fits the output current curves exactly at the proper voltage point, which is $V_{\mathrm{DS,sat}}$. The fit factor for this MOSFET variant is $\approx 6.19 \times 10^{-6}$. Using this fitting factor, and

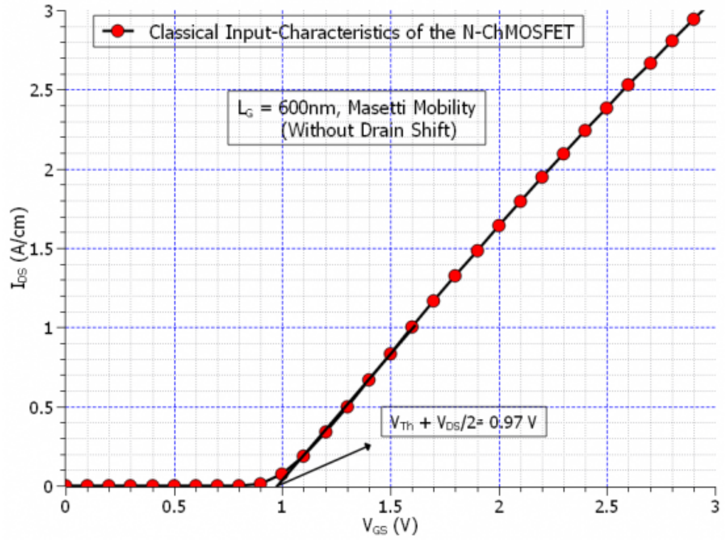


Figure 4.15.3.30: The input characteristics of the long-channel $L_G = 600\mathrm{nm}$ MOSFET, calculated with the Masetti mobility, showing the value of the threshold voltage V_{Th}.

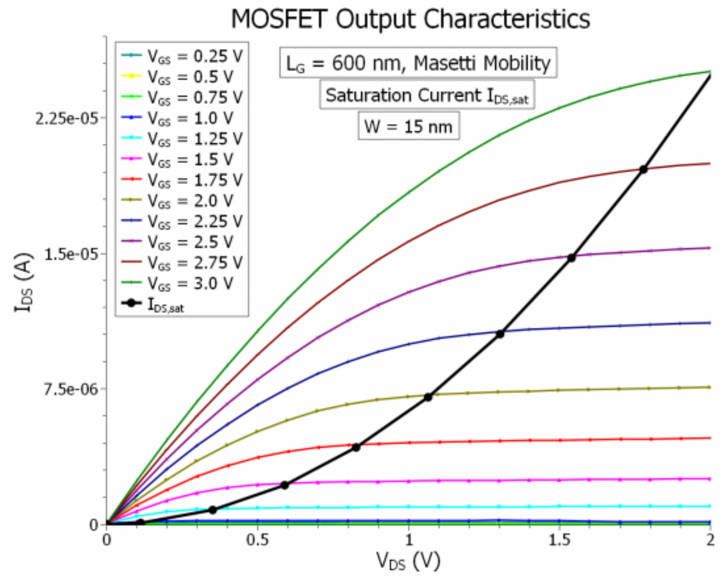


Figure 4.15.3.31: The output characteristics of the long-channel $L_G = 600\mathrm{nm}$ MOSFET, showing the diminishing of DIBL effect.

taking into consideration the new channel length $L_G = 600\text{nm}$, we get for the effective mobility:

$$\mu_n^{\text{eff}} \approx 788 \frac{\text{cm}^2}{\text{V} \cdot \text{s}}$$

The calculated mobility from the simulation is once again $933\text{cm}^2/\text{Vs}$ in the substrate, however it is $576\text{cm}^2/\text{Vs}$ at $y = 0$ coordinate.

Appendix: MOSFET

In the last section we found out, from the comparison of the input characteristics at high drain-source voltage $V_{\text{DS}} = 2\text{V}$, that the MOSFET device with a gate length of smaller than $L \leq 400\text{nm}$, would suffer from the punch-through effect. However, if we further shorten our gate length below 100nm, the situation would even be worse. Namely the leakage current would be so high, that even at very low source-drain voltages $V_{\text{DS}} = 0.2\text{V}$, the MOSFET would conduct, even at gate-voltages below the threshold voltage $V_{\text{GS}} < V_{\text{Th}}$, and therefore the switching capability of the MOSFET would be diminished and eliminated. Figure Figure 4.15.3.32 illustrates this phenomenon:

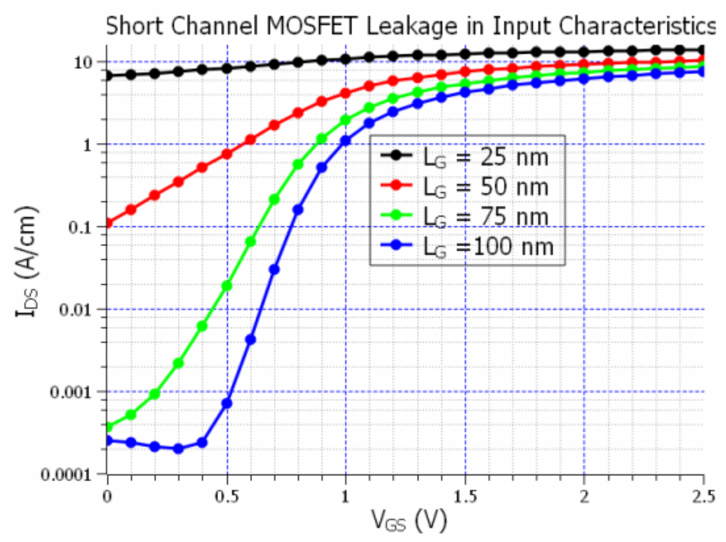


Figure 4.15.3.32: The comparison of input characteristics of the N-Ch MOSFET calculated quantum mechanically with the Masetti mobility, showing the leakage current in the input characteristics.

As the above input characteristics curves show, for gate-length below 100nm there is basically no valid switching function possible, as the drift current has already started at $V_{\text{GS}} = 0\text{V}$ for $L_G = 75\text{nm}$. This is basically to say that, at higher drain-source voltages the leakage curremt is actually more dominant to the channel inversion layer current, which can be switched on and off. It is also worth noting that the leakage current takes place inside the bulk of the MOSFET at the bottom of source drain doped region as figure Figure 4.15.3.33 shows:

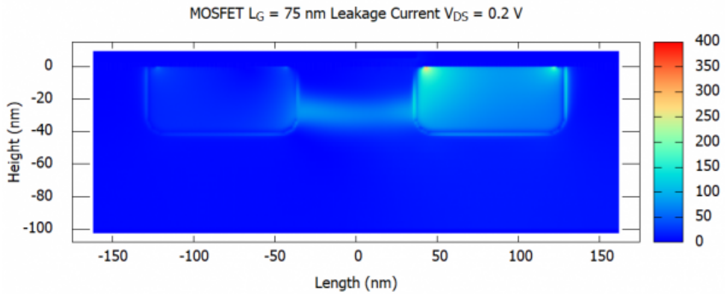


Figure 4.15.3.33: The norm of the leakage current in $L_G = 75\text{nm}$ MOSFET, at zero gate-voltage $V_{GS} = 0$, flowing within the bulk.

If we even consider the $L_G = 25\text{nm}$ MOSFET, there are certain quantum mechanical affects could be observed.

Using the *energy_resolved_density{ }*, one could observe spacial confinement within the channel at different energy levels. The code has to include the following lines:

```
classical{
 ...
 ...

   energy_distribution{
       min =  -0.5
       max =  1.0
       energy_resolution = 0.001
       only_density_quantum_regions = yes
   }

   energy_resolved_density{
       min =  -0.5
       max =  1.0
       energy_resolution = 0.001
       only_density_quantum_regions = yes
       output_energy_resolved_densities{}
   }

}
```

But to be able to see the quantum mechanical effects, lets us first take a look at the classical energy resolved densities in the channel and the source-drain doping regions (for that the `only_density_quantum_regions` flag has to be set to `no` in the `energy_resolved_density{}` group). The classical energy resolved densities are shown in figure Figure 4.15.3.34:

Now let us look at the same energy resolved densities in the MOSFET source and drain region, obtained using the quantum mechanics alone:

In the above figure we can clearly see that compared to the classical density, the quantum mechanical density indicate quantum confinement in the source drain doping regions. Furthermore, as we shall see in figure Figure 4.15.3.36, also the density in the inversion layer shows quantum confinement for different discrete energy levels:

As we can see there is clearly two different quantum confined modes in the inversion layer of the channel for this MOSFET.

With regards to the issue of convergence for the output characteristics, the convergence parameters become very relevant, since for the wrong set of parameters, the simulations may very well never converge and if so might take a significant amount of time. The key parameter to keep in mind is the ‘’alpha_fermi” parameter in current-poisson{ } calculations, which would decide the fate of the calculations. This parameter needs to be chosen corrently, and also since it will be dynamically reduced, the `alpha_scale` parameter also need to be set appropriately, with a relatively small `alpha_iterations` (default is 1000, which is very high!!!), so that a quick adjustment can be achieved if the parameter is too large. One also needs to significantly increase the number of iterations from the default 100, to a few thousand. This so called under-relaxation parameter for the quasi-Fermi level is important due to the fact that it decides the volume of the search for the solutions.

Last update: nnnn/nn/nn

4.15.4 Two-dimensional electron gas in a Si MOSFET

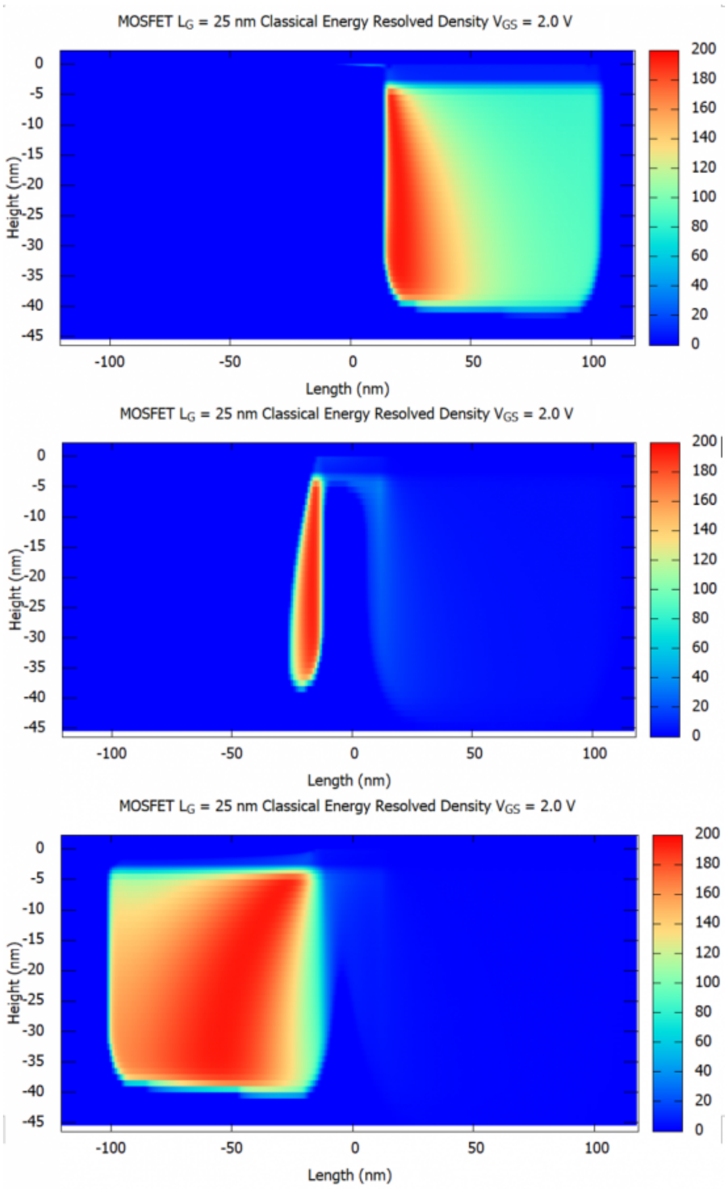


Figure 4.15.3.34: The classical energy resolved density in the $L_\mathrm{G} = 25\mathrm{nm}$ MOSFET at three different energy levels.

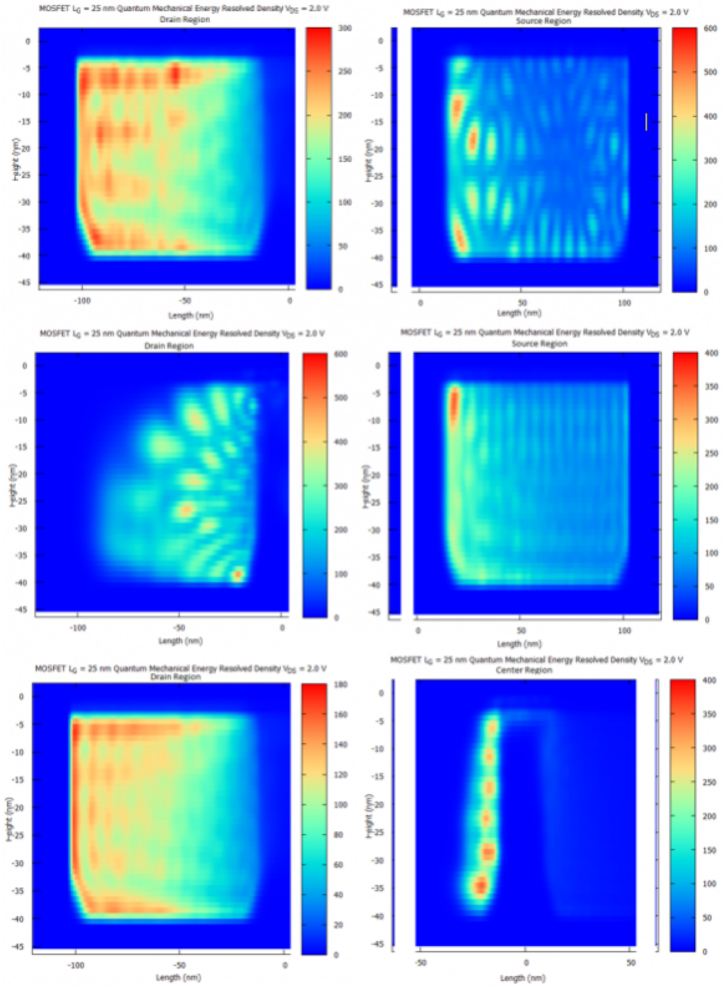


Figure 4.15.3.35: The quantum mechanical energy resolved density in the MOSFET source and drain regions, showing spacial quantum confinement at discrete energy levels.

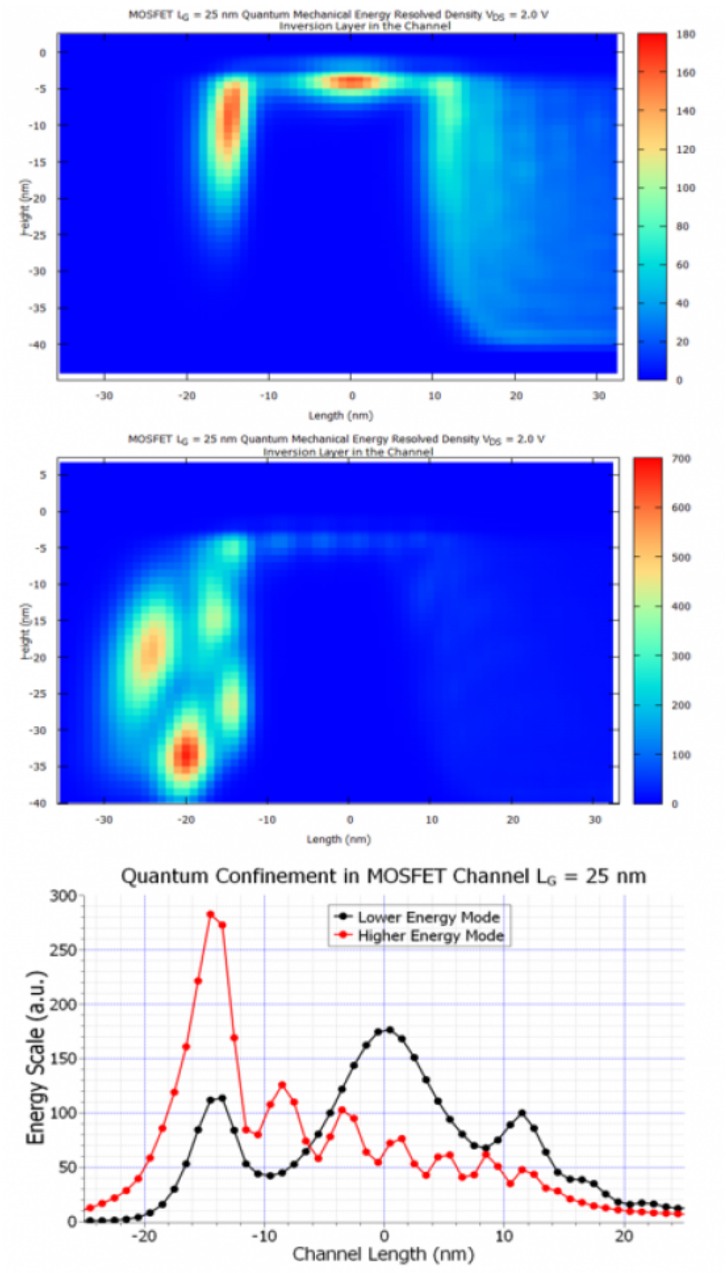


Figure 4.15.3.36: The quantum mechanical energy resolved density in the inversion layer of the MOSFET-channel, at two different energy levels, showing the standing wave pattern, which indicates quantum confinement.

Header

Files for the tutorial located in *nextnano++\examples*

- *2DEG_Si_MOSFET_1D_nnp.in*

Main adjustable parameters in the input file:

- parameter `$min_density_e`
- parameter `$max_density_e`
- parameter `$min_density_h`
- parameter `$max_density_h`

Relevant output files:

- *bias_*\bandedges.dat*
- *bias_*\Quantum\probabilities_shift_Quantum_region_X1.dat*
- *bias_*\Quantum\probabilities_shift_Quantum_region_X2.dat*
- *bias_*\Quantum\density_electron.dat*
- *integrated_density_electron.dat*

Introduction

In this tutorial, you can learn how to obtain carrier sheet densities in the inversion layer of MOSFET.

Layer sequence

The table below shows the materials, their widths, and their dopant concentrations for this tutorial.

material	width (nm)	doping
contact	10	
p-Si	99	5×10^{17} cm^{-3}
SiO$_2$	5	
n-Si (poly-Si)	54	3×10^{19} cm^{-3}
Gate contact	1	

The applied gate voltage leads to confined electron states at the p-Si/SiO$_2$ interface (**n-type inversion layer**) whereas the holes are repelled from the p-Si/SiO$_2$ surface towards the interior of the device (i.e. to the left side).

An applied source-drain voltage in the plane of the inversion layer will lead to a flow of current which depends on the sheet density in the inversion layer. The magnitude of the current is governed by the applied gate voltage, i.e. the gate controls the sheet density and thus switches the current on or off (MOSFET, metal-oxide-semiconductor field effect transistor).

Calculations

The temperature was set to 300 K. Self-consistent solution of the 1D-Schrödinger-Poisson equation within single-band effective-mass approximation (using ellipsoidal effective mass tensors) for the (Delta) conduction band edges.

We vary the gate voltage from 0 V to 2.5 V in steps of 0.1 eV.

Results

The following two figures show the band profiles and the electron density for two different gate voltages:

Figure 4.15.4.1: **0.7 V** (The electron ground state is above the electron Fermi level $E_{F,n}$)

Figure 4.15.4.2: **2.5 V** (The electron ground state is below the electron Fermi level $E_{F,n}$ and thus occupied, leading to a large quantum mechanical density)

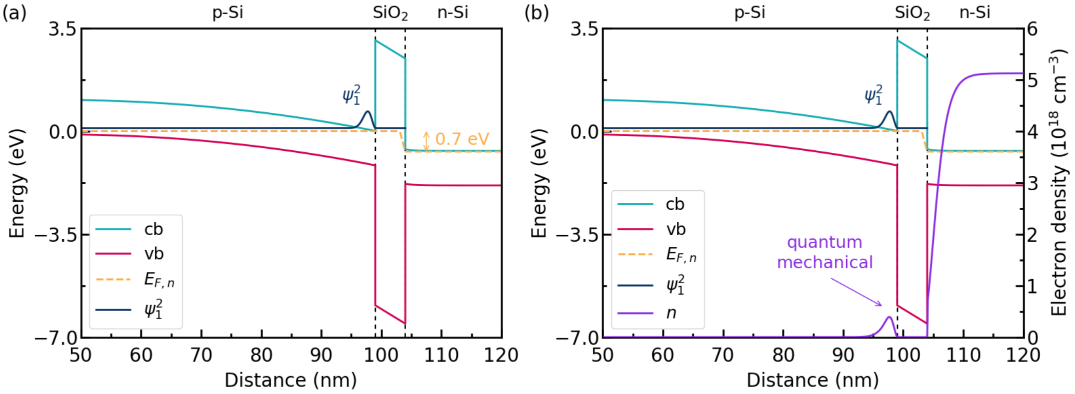


Figure 4.15.4.1: The calculated band edges are shown in (a). The quasi Fermi level of electrons $E_{F,n}$ drops 0.7 eV from p-Si to n-Si due to the gate bias. The calculated electron density n is overlaid on the band diagram in (b).

The amplitude of the ground state ψ_1^2 is **above** $E_{F,n}$ as you can see.

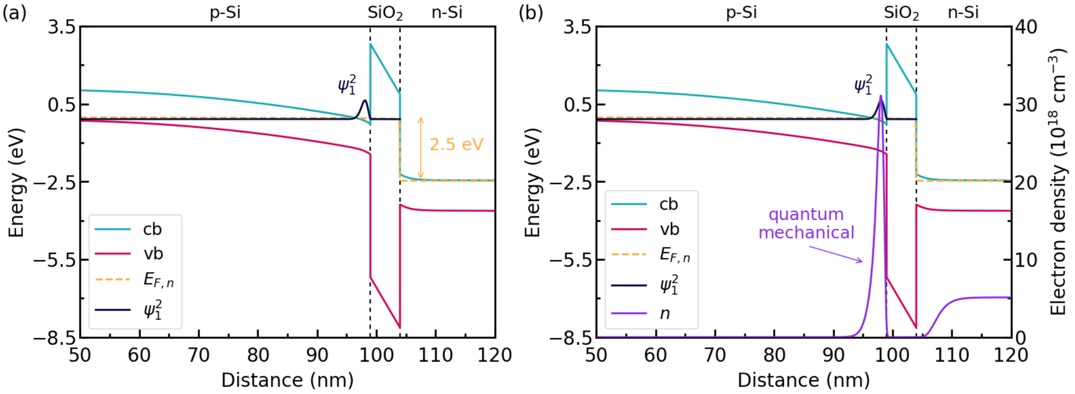


Figure 4.15.4.2: The calculated band edges are shown in (a). The quasi Fermi level of electrons $E_{F,n}$ drops 2.5 eV from p-Si to n-Si due to the gate bias. The calculated electron density n is overlaid on the band diagram in (b).

The amplitude of the ground state ψ_1^2 is **below** $E_{F,n}$ as you can see.

In the figures above, cb and vb represent the conduction band and the valence band, respectively.

In the poly-silicon on the right side of the SiO_2 barrier, the electrons get depleted from the oxide interface.

Due to the fact that the **quasi Fermi level** is nearly constant outside the SiO_2 barrier, almost no current is flowing. Inside the SiO_2 barrier, the quasi Fermi level has a step-like feature. However, as the electron density is close to zero inside the barrier, almost no current is eventually flowing.

The ground state electron level is associated with the longitudinal electron mass ($m_{\mathrm{longitudinal}} = 0.916\ m_0$). On the other hand, the second and the third eigenstate (which are degenerate) are associated with the transversal electron mass ($m_{\mathrm{transversal}} = 0.190\ m_0$). Due to this degeneracy, only **two** rather than three Schrödinger equations have to be solved: (a) $V(z), m = m_{\mathrm{transversal}} = 0.190\ m_0$ (b) $V(z), m = m_{\mathrm{longitudinal}} = 0.916\ m_0$ The potential $V(z)$ that enters into the Schrödinger equation is the same in these two cases.

The eigenvalues for $m_{\mathrm{longitudinal}}$ are contained in *bias_*\Quantum\probabilities_shift_Quantum_region_X1.dat*. The eigenvalues for $m_{\mathrm{transversal}}$ are contained in *bias_*\Quantum\probabilities_shift_Quantum_region_X2.dat*.

At $2.5\ \mathrm{eV}$, the energy spacing between the two lowest electron states is of the order $100\ \mathrm{meV}$ (in the case of the longitudinal effective mass). At $2.5\ \mathrm{eV}$, the energy spacing between the two lowest electron states is of the order $130\ \mathrm{meV}$ (in the case of the transversal effective mass). At $2.5\ \mathrm{eV}$, the energy spacing between the electron ground state of the longitudinal effective mass and the ground state of the transversal effective mass is of the order $70\ \mathrm{meV}$. Thus, in this case, one can safely assume that only **first subband** is occupied, i.e. the electron ground state with the longitudinal mass.

(to be fixed)

Electron sheet density in the inversion channel as a function of applied gate voltage

The file *bias_*\Quantum\density_electron.dat`* contains the electron density across the MOSFET. Since the p-Si region, where the inversion channel is located, extends from $x = 0\ \mathrm{nm}$ to $x = 99\ \mathrm{nm}$, you have to integrate the electron density over the region to obtain the sheet density. To do it on *nextnano*++, `structure{ region { integrate } }` is used as following (*structure{ region{ integrate{ } } }*).

```
region{
    line{ x = [ $itf_start_contact, $itf_p_Si_SiO2 ] }
    binary{ name = "Si" }
    doping{
        constant{
            name = "B_acceptor"
            conc = $acceptor_conc
        }
    }
    integrate{ electron_density{} }
}
```

The output is in the file *integrated_density_electron.dat*.

Figure 4.15.4.3 shows the electron sheet density of the p-Si inversion layer.

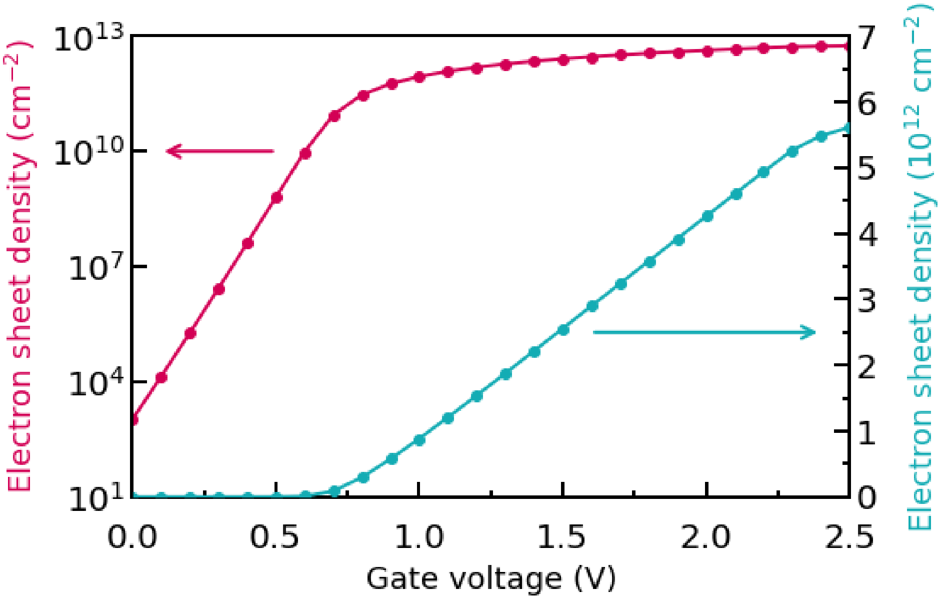


Figure 4.15.4.3: The electron sheet density of the p-Si inversion layer is shown.

To obtain the capacitance-voltage characteristics, you have to calculate the derivative of the sheet density.

Last update: 18/12/2024

4.15.5 Electron wave functions of a 2D slice of a Triple Gate MOSFET

In this tutorial we demonstrate the 2D simulation of a Triple Gate MOSFET. We solve the **two-dimensional Schrödinger and Poisson equations self-consistently** for a 2D slice. We would see the difference between the electron densities caluculated quantum mechanically and classically.

The relevant input files are as follows:

- *2DSi_TGMOS_2Dcut_atGate_cl_nnp.in*
- *2DSi_TGMOS_2Dcut_atGate_qm_nnp.in*
- *2DSi_TGMOS_2Dcut_atGate_qm_iso_nnp.in*
- *3DSi_TGMOS_5 nm_SD0V_G0V_qm.in*
- *3DSi_TGMOS_5 nm_SD0V_G05V_qm.in*

If you want to obtain the input files that are used within this tutorial, please contact support [at] nextnano.com.

2D Simulation

Structure

A Triple Gate MOSFET is a **nanowire** if the dimensions along the x and y directions are only a few nanometers, thus quantization effects have to be taken into account. The structure considered is as follows:

- The Si channel has a rectangular shape with a width of 5 nm and a height of 5 nm.
- The Si channel is surrounded by SiO2 (thickness 1.5 nm).

The Si/SiO2 nanowire is surrounded by a Gate (at the left and right side, and at the top).

The following schematic shows a 2D slice of a 3D Triple Gate MOSFET.

Simulation Details

In this tutorial we will only simulate this 2D slice and not the whole 3D structure.

We apply a voltage of 0.5 V to the Gates and solve the **two-dimensional Schrödinger and Poisson equations self-consistently** (including the SiO_2 region).

There are six equivalent conduction band minima in silicon (Delta valleys). Since the constant energy surfaces are ellipsoids, the mass tensor has the following two kinds of effective masses:

- The longitudinal mass is $0.916m_0$.
- The transversal mass is $0.190m_0$ (2 directions).

Therefore, we need to solve **three** 2D Schrödinger equations with different effective mass tensor orientations.

Our Schrödinger equations are numbered `X1`, `X2`, `X3`.

- `X1/deg1`: a) $m_{xx} = m_l = 0.916m_0$, $m_{yy} = m_t = 0.190m_0$
- `X2/deg2`: b) $m_{xx} = m_t = 0.190m_0$, $m_{yy} = m_l = 0.916m_0$
- `X3/deg3`: c) $m_{xx} = m_{yy} = m_t = 0.190m_0$

The potential $E_c(x, y)$ that enters the Schrödinger equation is the same in these three cases.

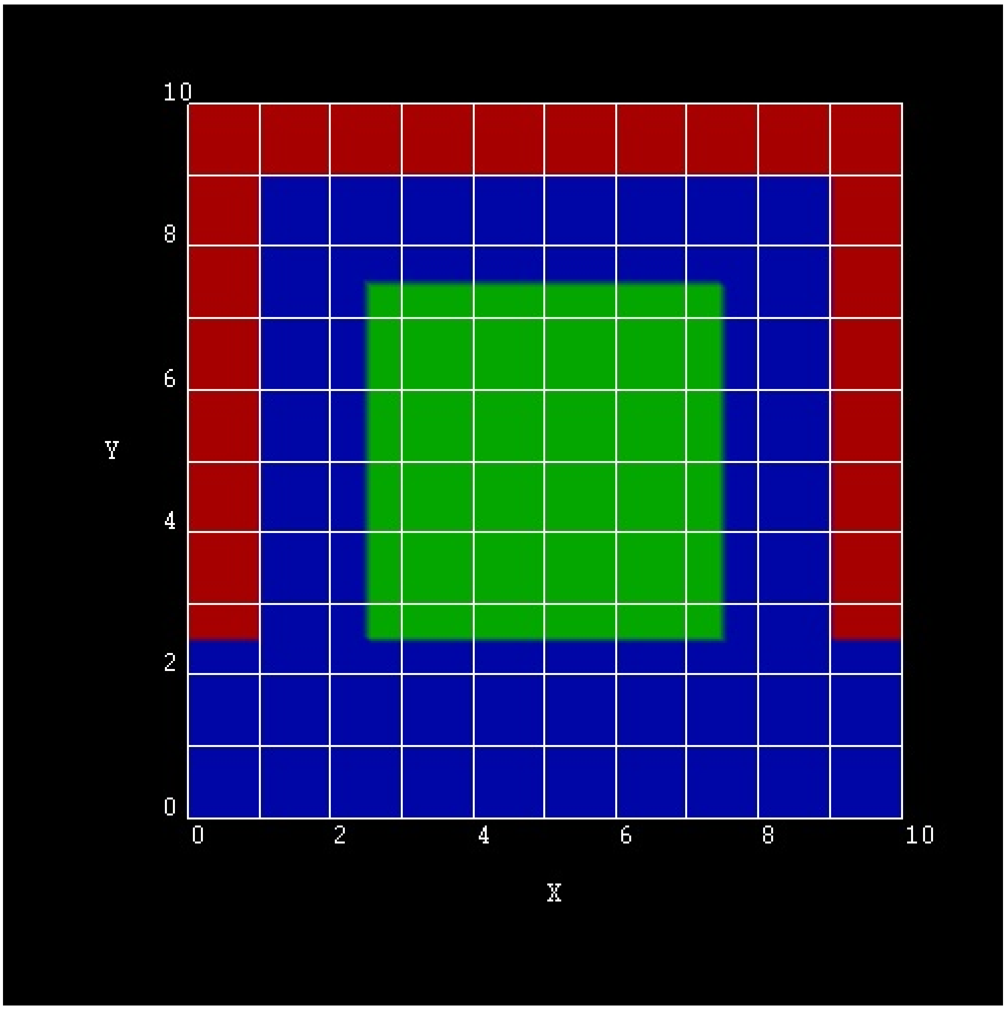


Figure 4.15.5.1: 2D slice of a 3D Triple Gate MOSFET

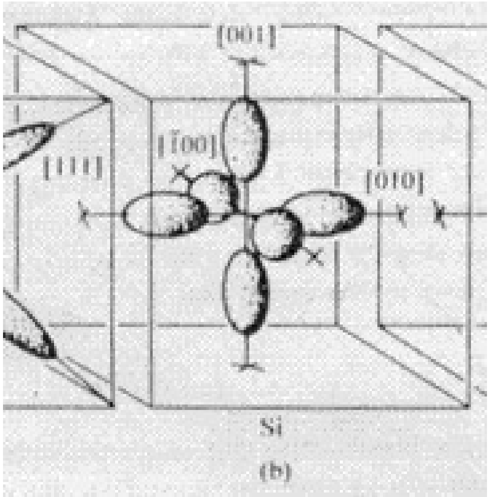


Figure 4.15.5.2: constant energy surface of Si conduction band

Note

The cases a) and b) are not identical (i.e. degenerate) because the potential is not symmetric with respect to exchanging x and y coordinates.

The following keyword and specifier can be used to output the effective mass tensors ($1/m_{ij}$).

```
# nextnano++
output{
    ...
    material_parameters{
        ...
        charge_carrier_masses{
            boxes = yes
        }
    }
}
```

Results

Electron wave functions $|\psi^2|$

- *2DSi_TGMOS_2Dcut_atGate_qm_nnp.in*, **_nn3.in*

The lowest eigenstates for the cases a), b) and c) are the following:

- `X1/deg1`: a) $m_{xx} = m_l = 0.916 m_0$, $m_{yy} = m_t = 0.190 m_0$

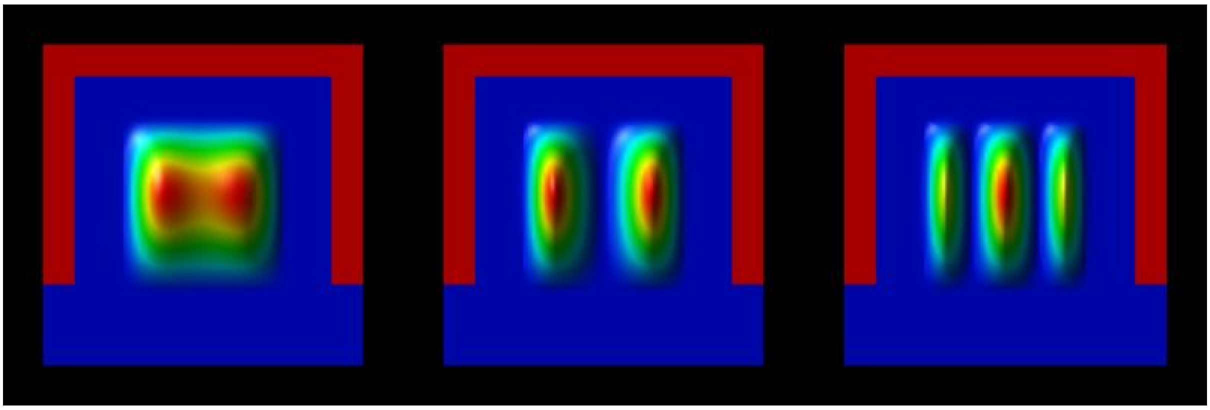


Figure 4.15.5.3: $E_{1,X1} = -26$ meV, $E_{2,X1} = -1$ meV, $E_{3,X1} = 77$ meV

  Here, the heavier mass is along the x direction, and the lighter mass along the y direction. The energy spacing between the two lowest subbands is about 24 meV. The eigenvalues are contained in *bias_00000/Quantum/energy_spectrum_quantum_region_X1_00000.dat/Schroedinger_1band/ev2D_cb003_qc001_sg0*

- `X2/deg2`: b) $m_{xx} = m_t = 0.190 m_0$, $m_{yy} = m_l = 0.916 m_0$

  Here, the lighter mass is along the x direction, and the heavier mass along the y direction. The energy spacing between the two lowest subbands is about 35 meV. The eigenvalues are contained in *bias_00000/Quantum/energy_spectrum_quantum_region_X2_00000.dat/Schroedinger_1band/ev2D_cb003_qc001_sg0*

- `X3/deg3`: c) $m_{xx} = m_{yy} = m_t = 0.190 m_0$

  These eigenvalues have the lighter mass in the x and y directions. Consequently, their energies are much higher than in the other two Schrödinger equations. The energy spacings between the lowest subbands is of the order 140-150 meV. The eigenvalues are contained in *bias_00000/Quantum/energy_spectrum_quantum_region_X3_00000.dat/Schroedinger_1band/ev2D_cb003_qc001_sg0*

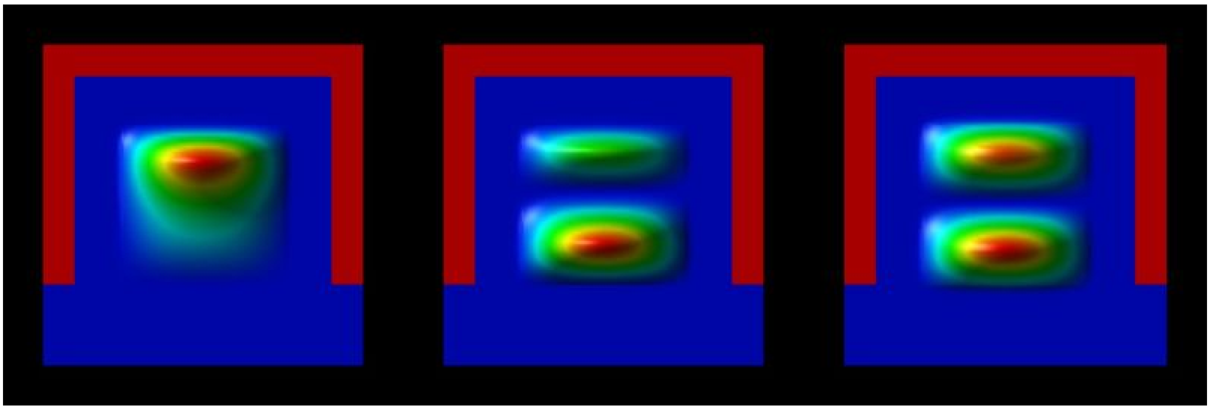


Figure 4.15.5.4: $E_{1,X2} = -28$ meV, $E_{2,X2} = 6$ meV, $E_{3,X2} = 82$ meV

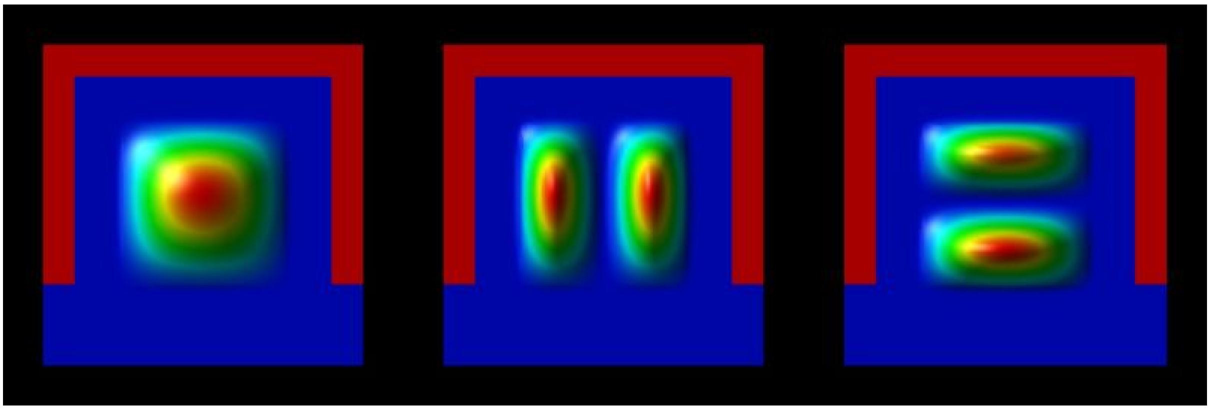


Figure 4.15.5.5: $E_{1,X3} = 16$ meV, $E_{2,X3} = 167$ meV, $E_{3,X3} = 173$ meV

(Compare the wave functions and the energies with the isotropic case as discussed further below.)

Electron density

The resulting electron density has the following shape, see Figure 4.15.5.6:

The units are 1×10^{18}cm^{-3}. The density has been calculated by occupying the eigenstates with respect to the Fermi level which is at 0 eV. Note that the quantum mechanical density is close to zero near the Si/SiO$_2$ interfaces because the wave functions tend to zero at the SiO$_2$ barriers.

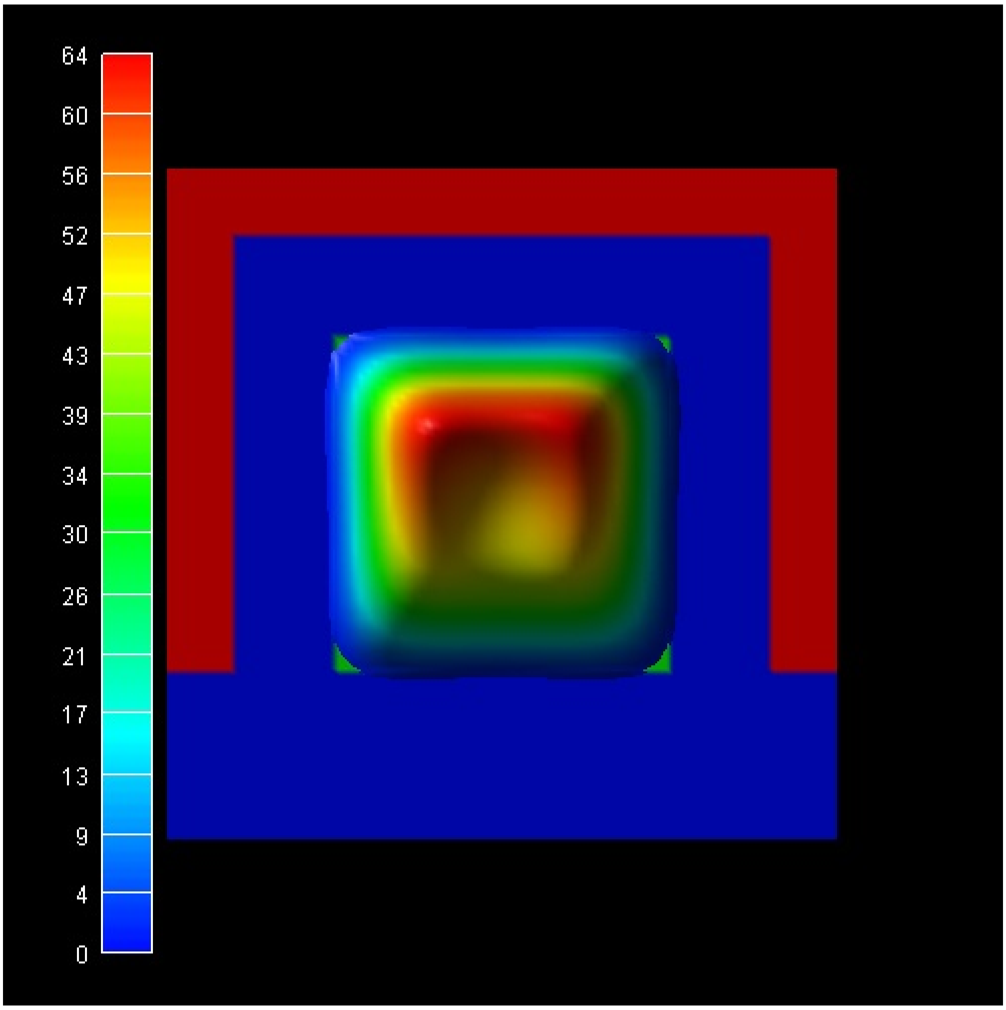


Figure 4.15.5.6: electron density

Figure 4.15.5.7 shows the same quantum mechanical electron density together with two slices through the conduction band edges. The units are in eV and the conduction band offset between SiO$_2$ and Si is 3.1 eV. At the gates, the conduction band edge is set to -0.5 eV, representing the applied bias of 0.5 eV. One can clearly see that **for silicon** in the middle of the nanowire the conduction band has its highest value and its lowest value close at the Si/SiO$_2$ interface.

If one had neglected the effect of quantum confinement, then the resulting **classical electron density** would have peaks near the Si/SiO$_2$ interfaces as is shown in Figure 4.15.5.8.

- *2DSi_TGMOS_2Dcut_atGate_cl_nnp.in*, **_mm3.in*

Obviously, a realistic calculation of such transistors cannot be based on classical densities. The full 2D (or better 3D) Schrödinger equations have to be solved. The IV characteristics of such a quantum-mechanically calculated Triple Gate MOSFET transistor will be discussed in another tutorial.

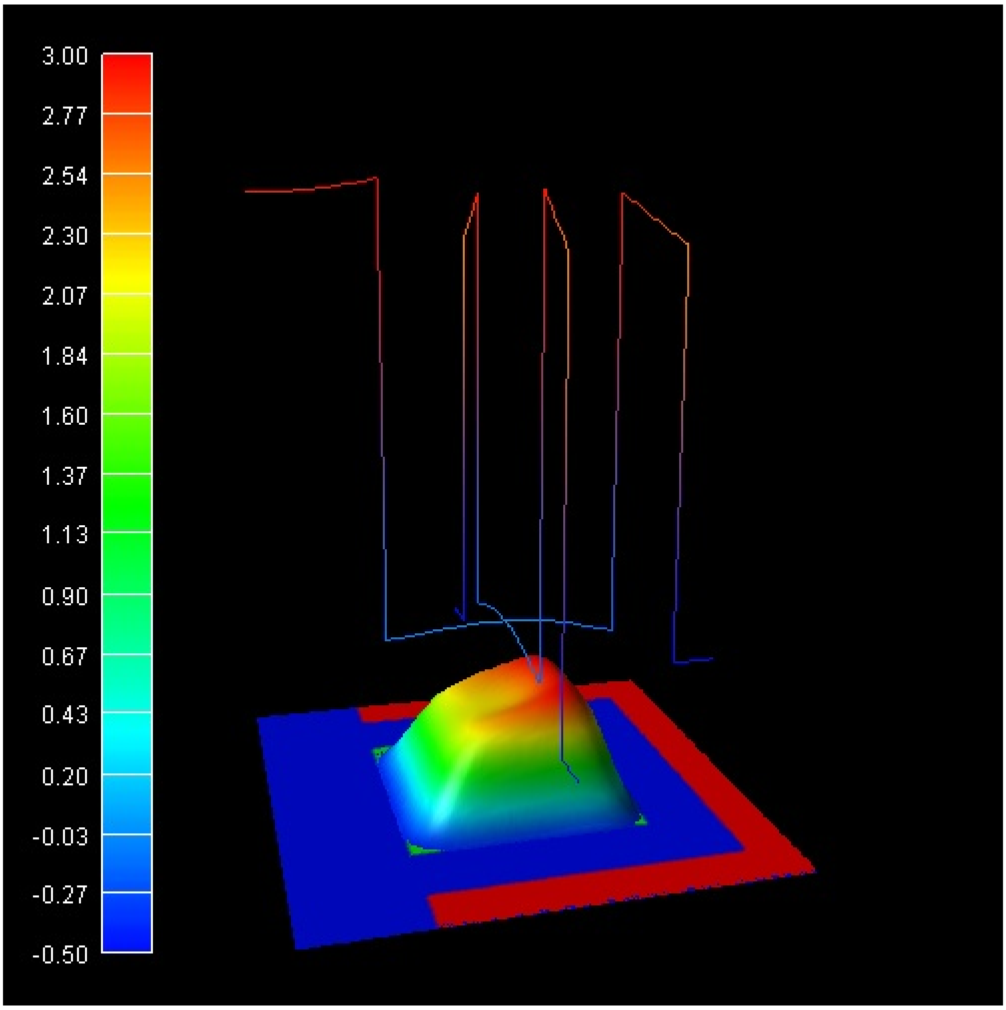


Figure 4.15.5.7: electron density and slices through the conduction band edges

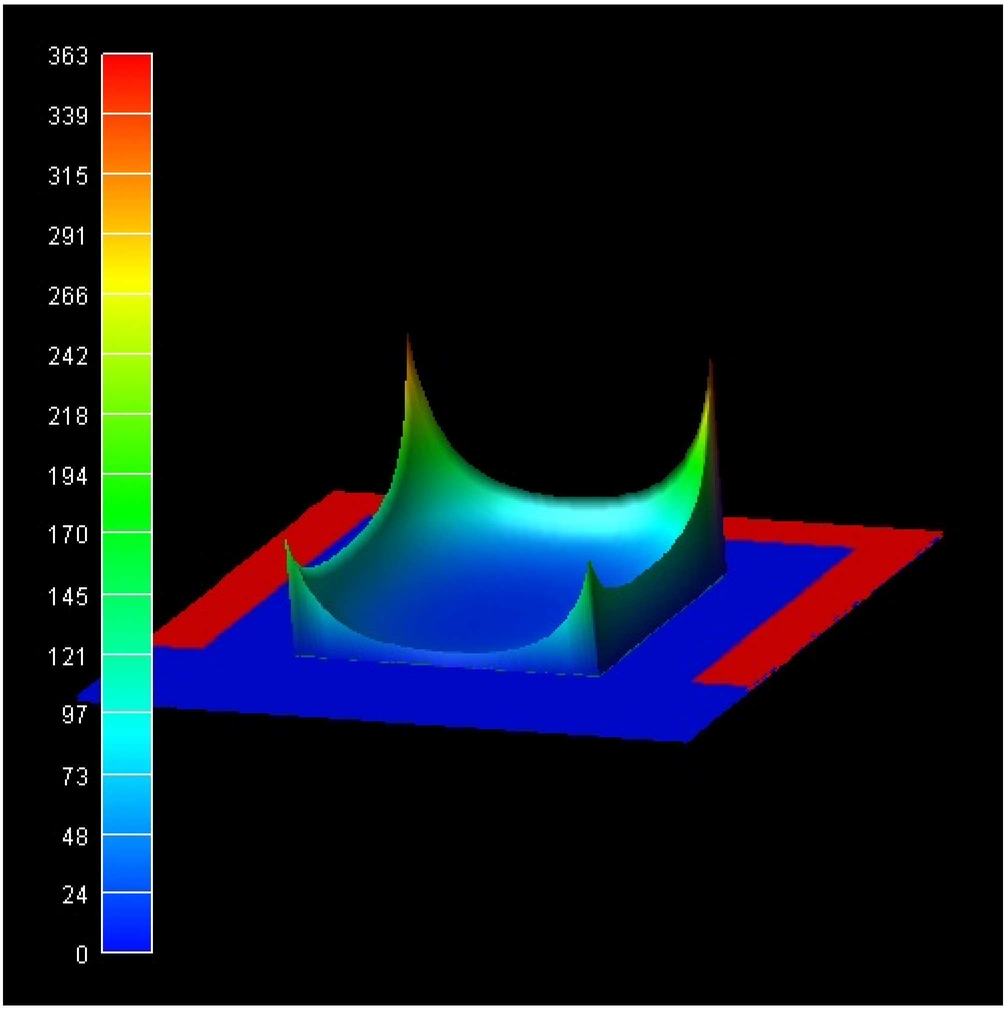


Figure 4.15.5.8: classical electron density calculated by *2DSi_TGMOS_2Dcut_atGate_cl.in*

Isotropic electron masses

Very often, for simplicity, an isotropic electron mass for the Schrödinger equation is assumed. E.g. the DOS (density of states) electron mass of Si in the Delta minima can be calculated as follows:

$$m^*_{e,DOS} = (m_l m_t^2)^{1/3} = (0.9160.19^2)^{1/3} m_0 = 0.321 m_0$$

In this case, only **one** Schrödinger equation has to be solved (in contrast to three equations as described above).

The wave functions and energies in this case are:

- $m_{xx} = m_{yy} = m_{DOS} = 0.321 m_0$

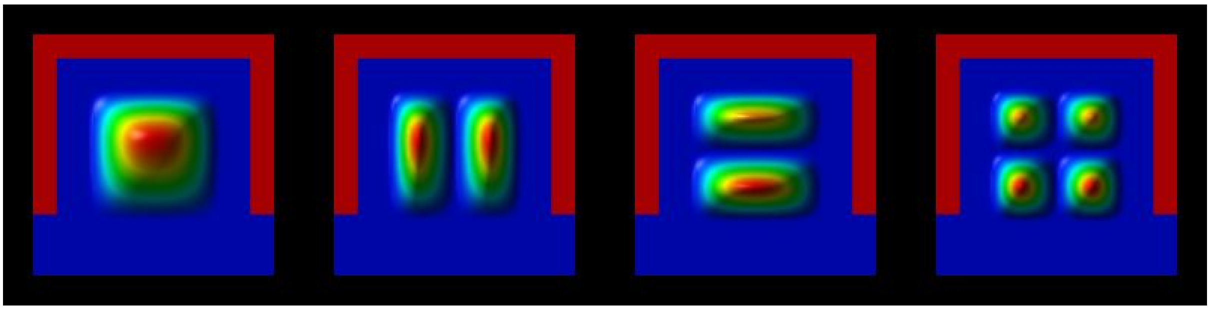


Figure 4.15.5.9: $E_1 = -21$ meV, $E_2 = 69$ meV, $E_3 = 75$ meV, $E_3 = 166$ meV

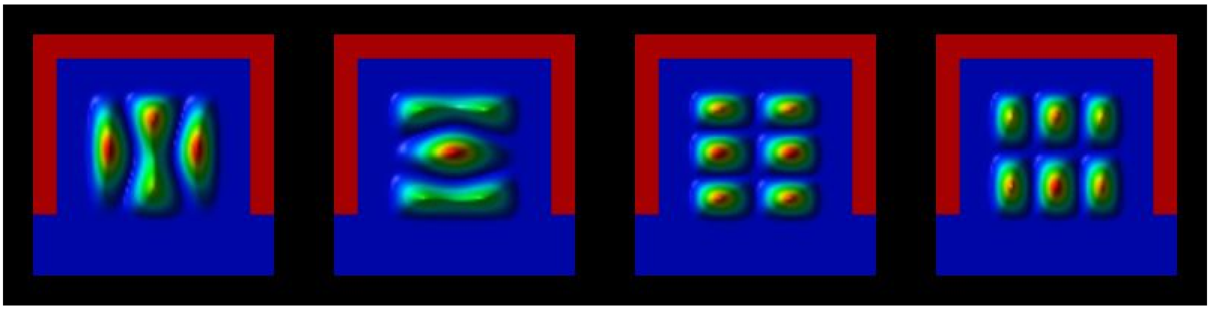


Figure 4.15.5.10: $E_5 = 262$ meV, $E_6 = 270$ meV, $E_7 = 360$ meV, $E_8 = 360$ meV

The wave functions $|\psi^2|$ look very similar as in the case of "`X3/deg3: c`)" (see above) where the masses are isotropic in the (x,y) plane but here, the energy spacings between different subbands are smaller (around 90-100 meV) because the DOS mass is larger than the transversal masses.

The eigenvalues are contained in *bias_00000/Quantum/energy_spectrum_quantum_region_X3_00000.dat/Schroedinger_1band/ev2*

3D simulation of the Triple Gate MOSFET

The following figures show the results of the self-consistent 3D Schrödinger-Poisson solution of this Triple Gate structure (Si channel length = 25 nm, source region length = 10 nm, drain region length = 10 nm, constant doping profile in source and drain region with a doping concentration of 1×10^{20} cm^{-3} (fully ionized)

The plots show the isosurfaces of the electron densities along 2D slices through the Triple Gate MOSFET. Figure 4.15.5.12 and Figure 4.15.5.13 also show 1D slices of the conduction band profiles and 1D slices of the electron densities in the middle of the device.

The classical densities would look similar to the classical densities of the 2D calculations shown above.

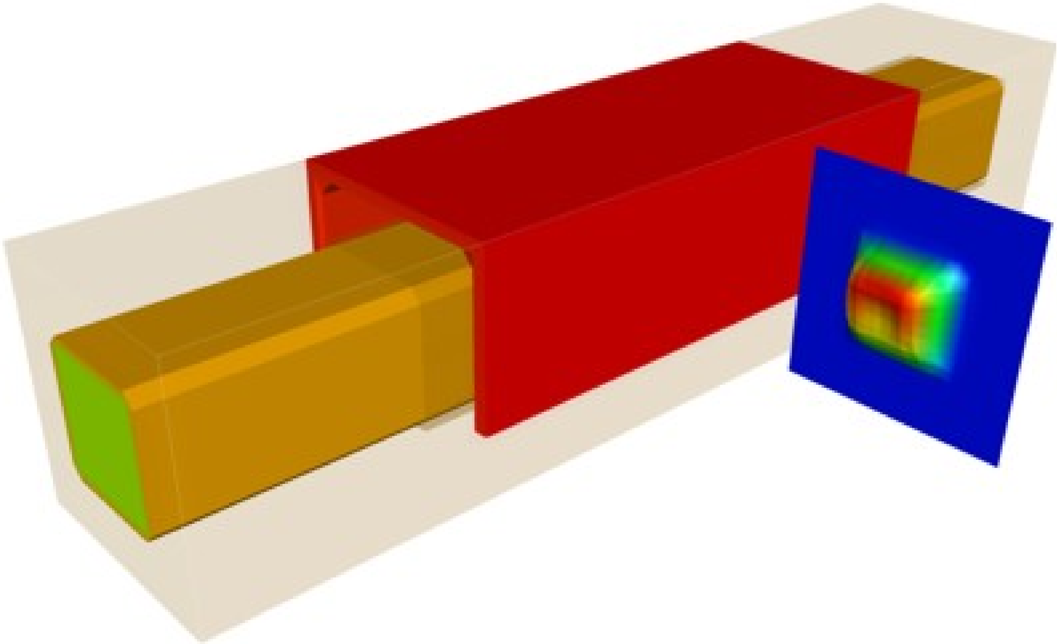


Figure 4.15.5.11: The whole 3D structure of this triple gate MOSFET and electron density through a 2D slice

This tutorial also exists for *nextnano^3*.

Last update: nnnn/nn/nn

4.15.6 Single-electron transistor - laterally defined quantum dot

Input files:

- *SET_Scholze_IEEE2000_1D_nnpp.in*
- *SET_Scholze_IEEE2000_3D_cl_nnpp.in*
- *SET_Scholze_IEEE2000_3D_top_gates_cl_nnpp.in*

Note

If you want to obtain the input files that are used within this tutorial, please check if you can find them in the installation directory. If you cannot find them, please submit a Support Ticket.

Scope:

In this tutorial, we simulate an $AlGaAs$/ $GaAs$ heterostructure grown along the z direction. The tutorial is based on *[Scholze2000]*.

Introduction

The $AlGaAs$/ $GaAs$ heterostructure leads to a two-dimensional electron gas (2DEG). By applying a gate voltage on top of the structure in the (x, y) plane, one is able to deplete the 2DEG and a laterally defined QD is formed. By adjusting the gate voltage, one is able to tune the number of electrons that are inside the QD.

Figure 4.15.6.1 shows the conduction band edge E_c (x, y) and the electron density $n(x, y)$ for the 2DEG plane, i.e. at z = 8 nm below the $GaAs$/ $AlGaAs$ heterojunction.

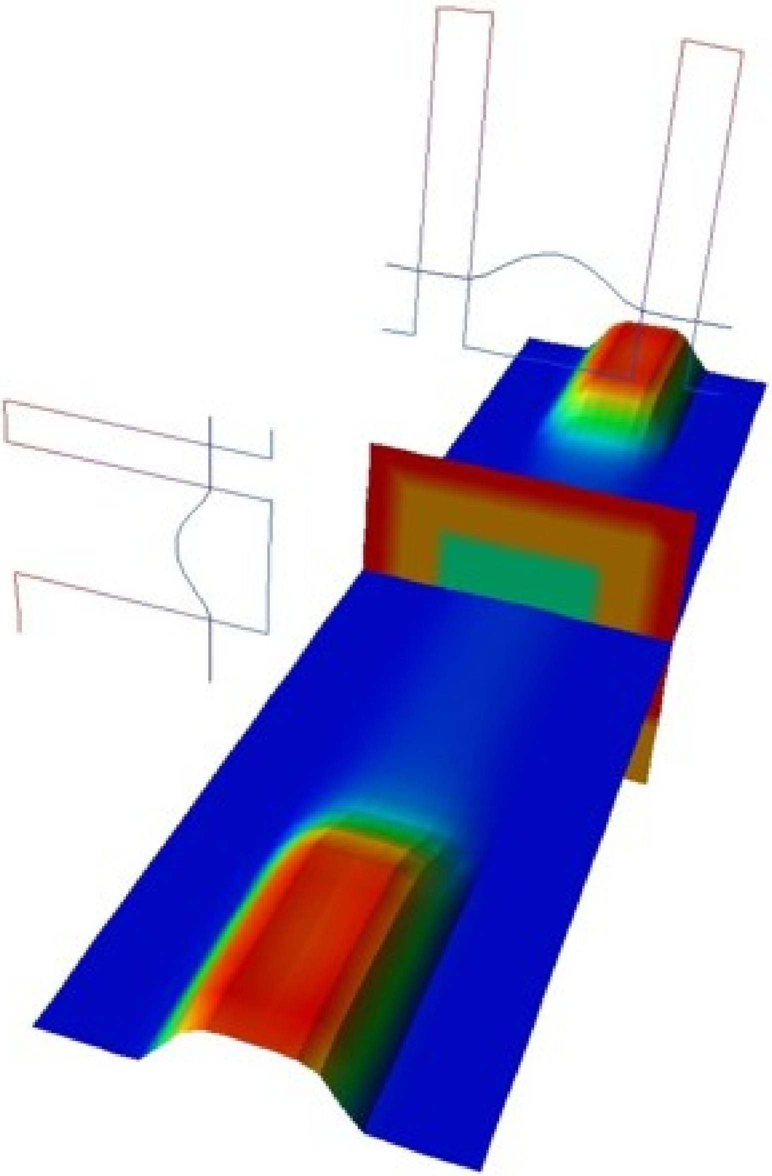


Figure 4.15.5.12: closed channel, $V_{SD} = 0.0\text{V}$, $V_{SG} = 0.0\text{V}$

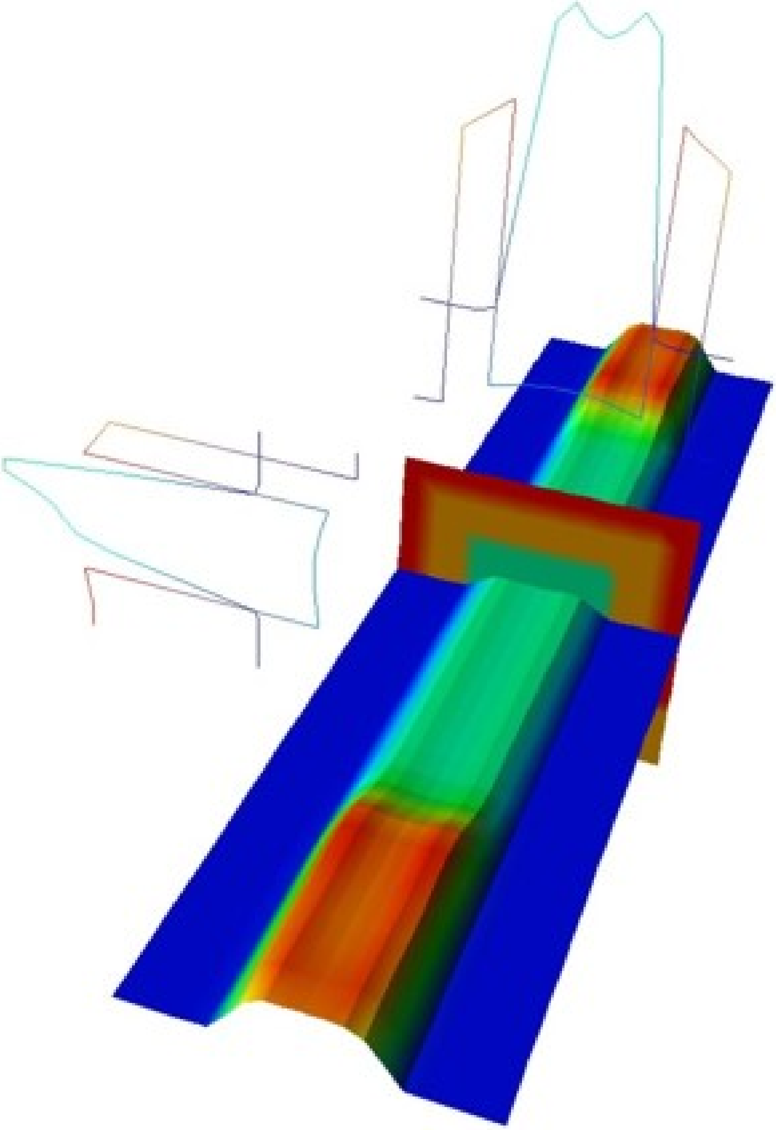


Figure 4.15.5.13: open channel, $V_{SD} = 0.0\text{V}$, $V_{SG} = 0.5\text{V}$

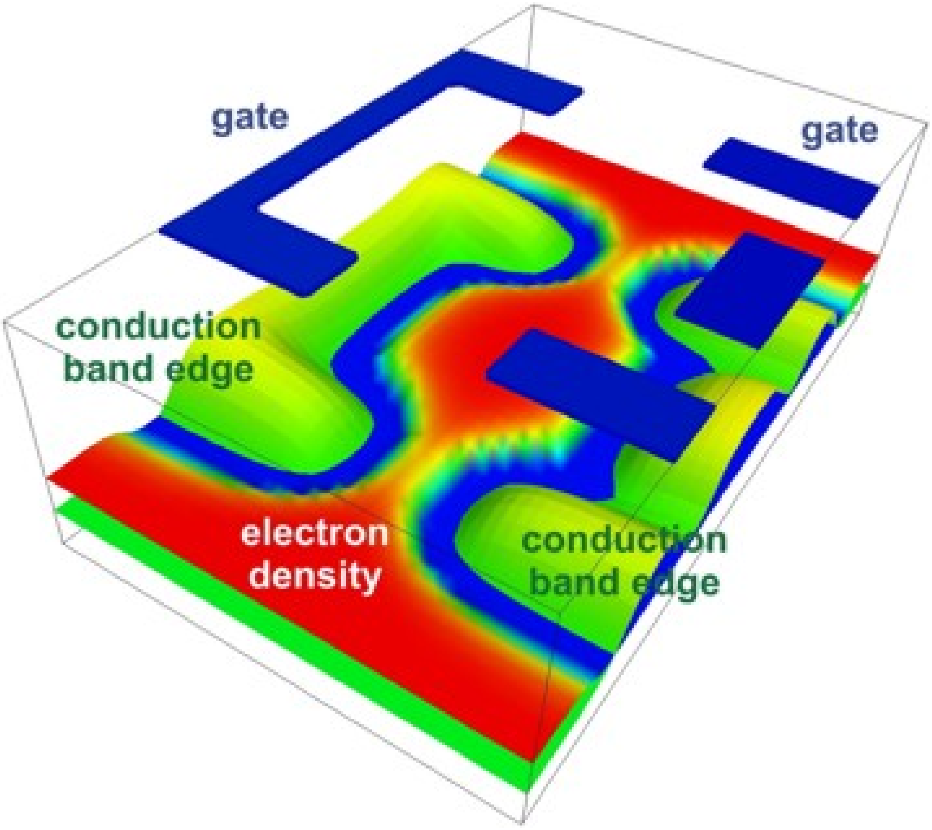


Figure 4.15.6.1: Conduction band edges (green), electron density (red) and geometry of the top gates (blue).

We divide the tutorial into two parts:

- In part 1, we simulate the heterostructure along the z direction and neglect the gates. (1D simulation: self-consistent Schrödinger-Poisson equation), *SET_Scholze_IEEE2000_1D_nnpp.in*
- In part 2, we solve the 3D Poisson equation to study the effect of the gates. (3D simulation: only Poisson equation using a classical density), *SET_Scholze_IEEE2000_3D_top_gates_cl_nnpp.in*

Part 1: 1D simulation (self-consistent Schrödinger-Poisson)

Figure 4.15.6.2 shows the calculated conduction band edge and the electron density of the heterostructure. The results are similar to Fig. 4 in *[Scholze2000]*.

At the left boundary, a Schottky barrier of 0.6 V has been assumed. At z = 20 nm, a δ-doping layer is present. The Fermi level is assumed to be constant at E_F = 0 eV. The ground state wave function (Ψ_1) is ~8 meV below the Fermi level and dominates the electron density. The first excited state (Ψ_2) is ~3 meV above the Fermi level, the second excited state (not shown) is 19 meV above the Fermi level.

Part 2: 3D simulation with top gates (Poisson equation only)

Figure 4.15.6.3 shows the 3D structure that we are going to simulate.

Figure 4.15.6.4 shows two 2D slices through the lateral (x, y) plane at a distance of 8 nm below the $AlGaAs$/ $GaAs$ interface. The results are similar to Fig. 5 in *[Scholze2000]*. At the top, the four gates are shown.

Last update: nnnn/nn/nn

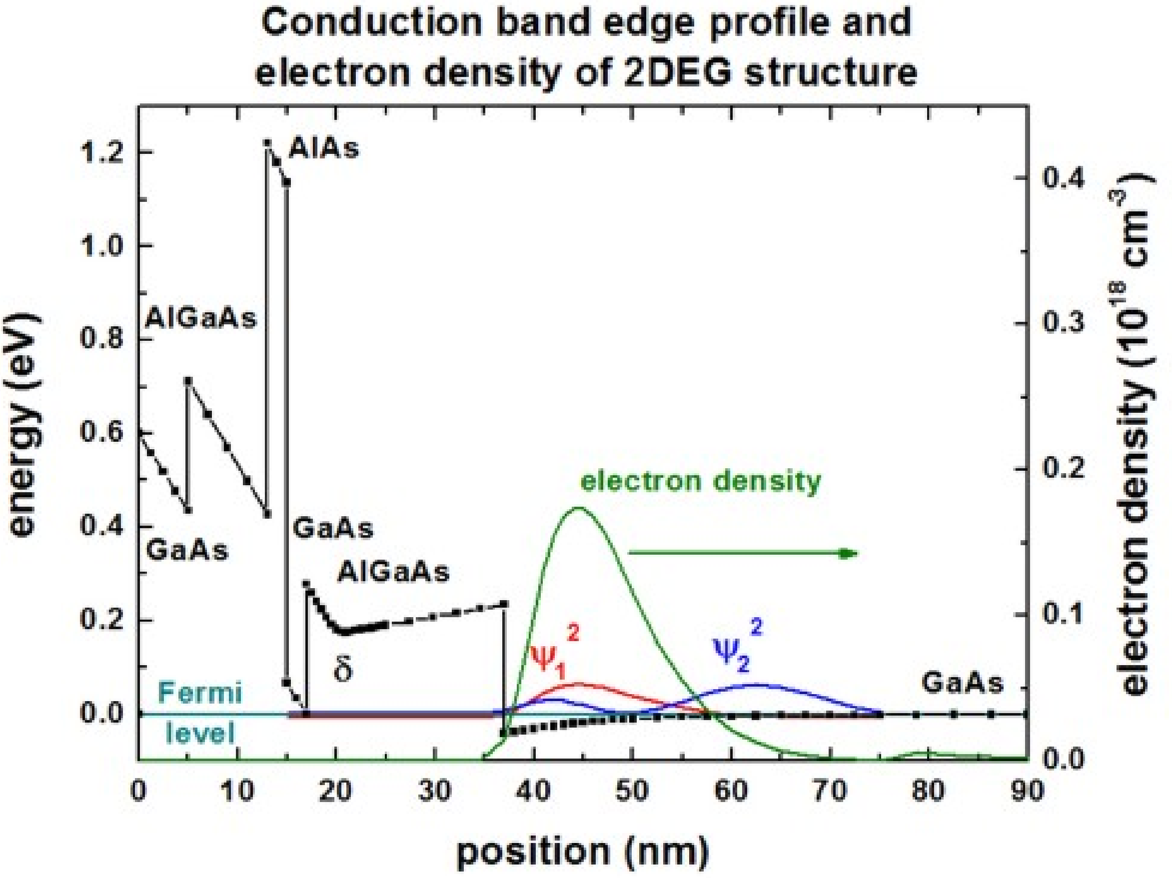


Figure 4.15.6.2: Conduction band edge profile (black), electron density (green) and Fermi levels (cyan) of 1D simulation.

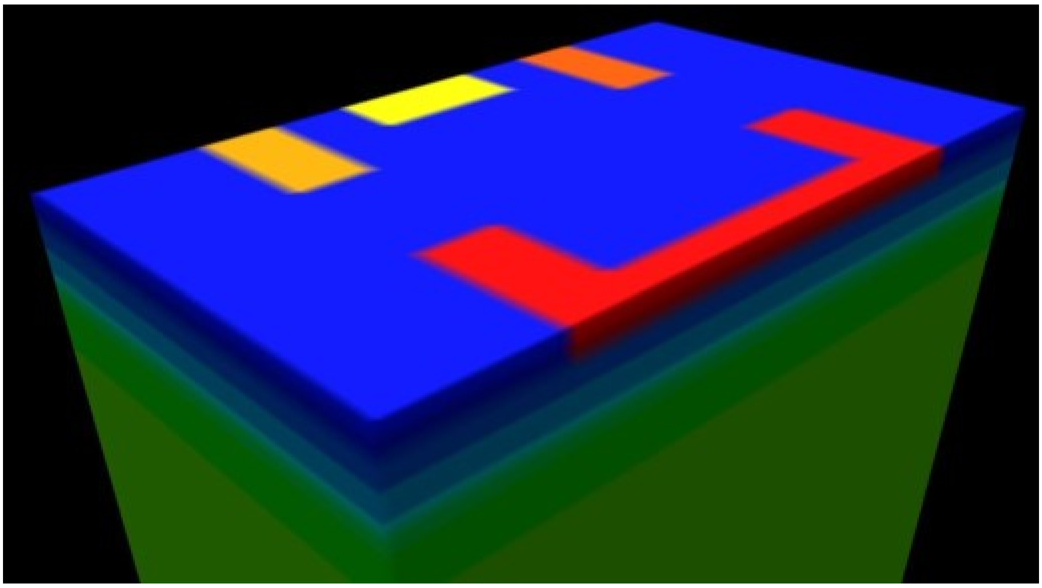


Figure 4.15.6.3: Device structure (3D)

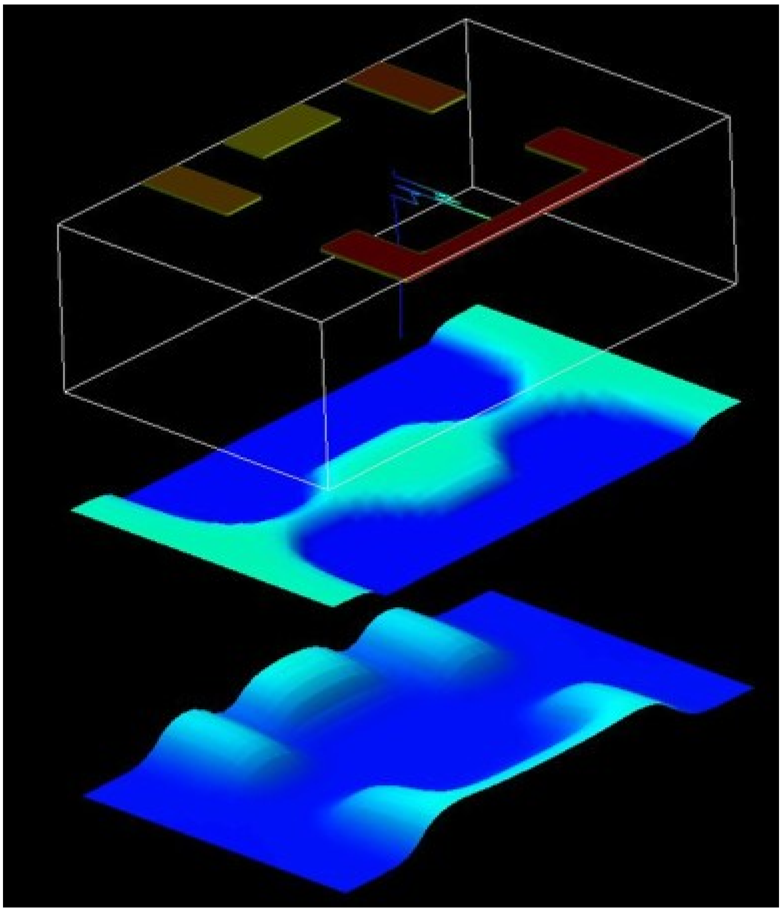


Figure 4.15.6.4: In the middle, the electron density is shown. The electron density has been calculated classically. At the bottom, the conduction band edge is shown.

4.15.7 — DEV — Ultrathin-body DG MOSFET with 2-nm channel

⚠ **Warning**

This tutorial is under development.

Tags: #FET

Contents

Files for the tutorial located in *nextnano++\examples\transistors*

- *DG-MOSFET-2-nm_zb_IV_2D_classical.in* (Figure 4.15.7.2, Figure 4.15.7.4, Figure 4.15.7.7)
- *DG-MOSFET-2-nm_zb_IV_2D_quantum.in* (Figure 4.15.7.2, Figure 4.15.7.6)
- *DG-MOSFET-2-nm_zb_IV_3D_classical.in* (Figure 4.15.7.2, Figure 4.15.7.4, Figure 4.15.7.7)

Parameters

- `$Temperature` – temperature of the crystal and electrons
- `$SourceDrainVoltage` – bias between the source and drain
- `$DopingConcentration` – doping concentration of the source and drain

Output files

- *Structure\last_region.avs.fld* (Figure 4.15.7.2)
- *Structure\contact.avs.fld* (Figure 4.15.7.2)
- *bias_xxxxx\density_electron.avs.fld* (Figure 4.15.7.4, Figure 4.15.7.5, Figure 4.15.7.6)
- *IV_characteristics.dat* (Figure 4.15.7.7)

Double Gate MOSFET

This tutorial aims to simulate the I-V characteristics of a double gate metal oxide semiconductor field effect transistor (DG MOSFET). The main idea of a DG MOSFET is to control the Si channel very efficiently by choosing the Si channel width to be very small and by applying a gate contact to both sides of the channel. This concept helps to suppress short channel effects and leads to higher currents as compared with a MOSFET having only one gate.

The geometry of the simulated Double Gate MOSFET structure is shown in Figure 4.15.7.1. The width of the Si channel is 2 nm. The distance between the two gates is 6 nm, i.e., the isolating SiO_2 is 2 nm thick on each side. The width of the two gates is 20 nm. The distance between source and drain is 60 nm. The widths and the lengths of source, drain, left, and right doped source regions are 10 nm x 10 nm each. The length of the 2 nm Si channel (without the square doped source and drain regions) is 40 nm.

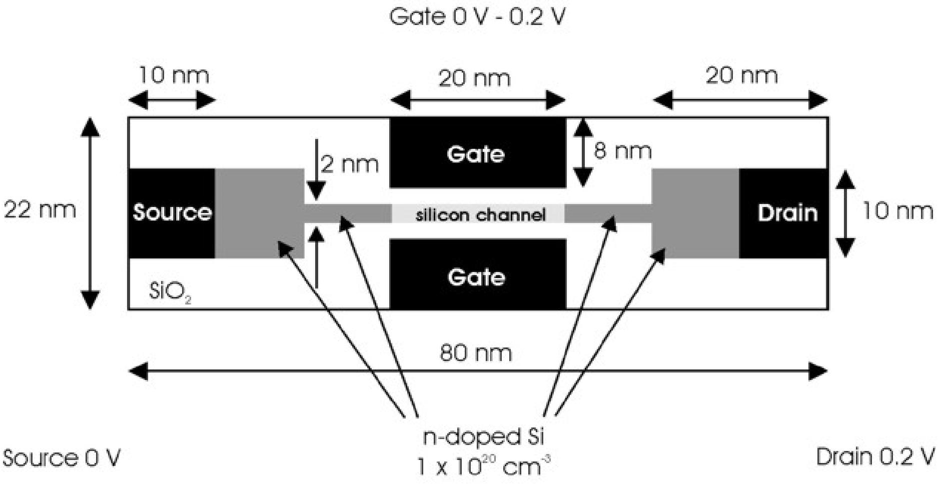


Figure 4.15.7.1: Geometry of the simulated Double Gate MOSFET.

The material regions defined in the input file for the *nextnano++* simulations are shown in Figure 4.15.7.2. The blue squares (Si) are n-doped with a concentration of $1 \cdot 10^{20}\ \mathrm{cm}^{-3}$. The 2 nm channel is n-doped with the same concentration from 20 nm to 30 nm and from 50 nm to 60 nm.

A constant bias of 0.0 V and 0.2 V is applied to source and drain, respectively. At the two gates we apply Schottky barriers of 3.443 eV, and sweep over the applied bias from 0 V to 1 V.

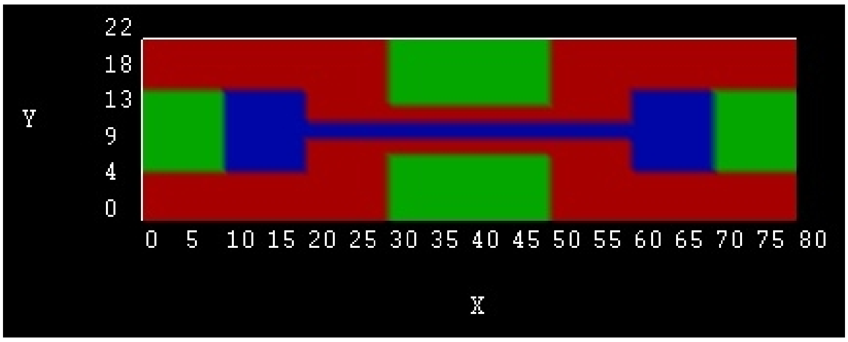


Figure 4.15.7.2: Schematic top view of the material regions defined in the *nextnano++* simulations.

The numerical grid employed in the simulations is shown in Figure 4.15.7.3.

Input file

For the simulations, the following parameters, which are specified in the corresponding input file *DG-MOSFET-2-nm_zb_IV_2D_classical.in*, are used:

- The lattice temperature is taken to be 300 Kelvin.
- The classical current and nonlinear Poisson equations are solved self-consistently without including the effect of strain.
- A two-dimensional simulation is performed. The overall simulation domain, that is the real space region in which the device is defined, is taken to be a rectangle having the size 22 nm x 80 nm.

Electron densities

In Figure 4.15.7.4 the electron density inside the MOSFET structure at 0 V is shown. The corresponding data is contained in the file *bias_00000\density_electron.avs.fld*.

In Figure 4.15.7.5 the electron density inside the MOSFET structure at 0.2 V is shown. The corresponding data

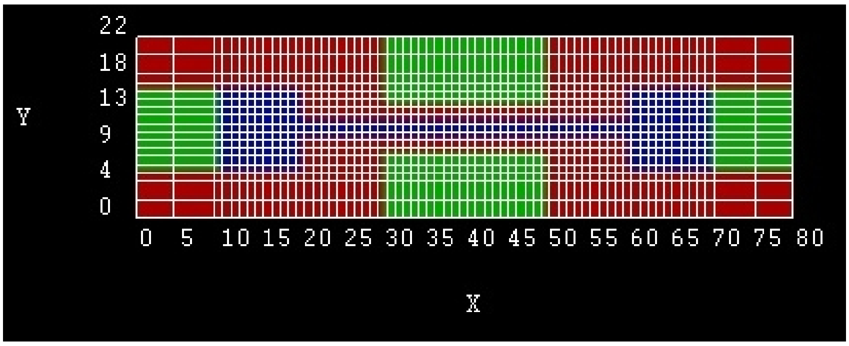


Figure 4.15.7.3: Grid lines of the Double Gate MOSFET

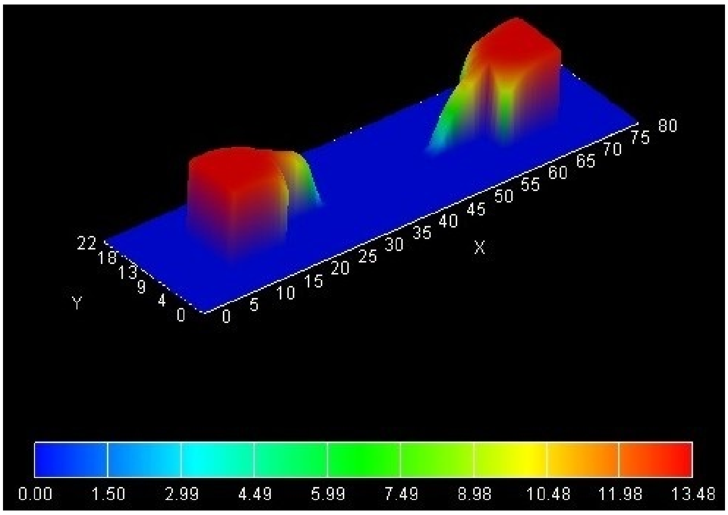


Figure 4.15.7.4: Electron density in units of $1 \cdot 10^{18}\mathrm{cm}^{-3}$ at $0\,\mathrm{V}$ gate voltage.

is contained in the file *bias_00002\density_electron.avs.fld*. One can clearly see that the electron density has the highest values at the $\mathrm{Si} - \mathrm{SiO_2}$ interfaces.

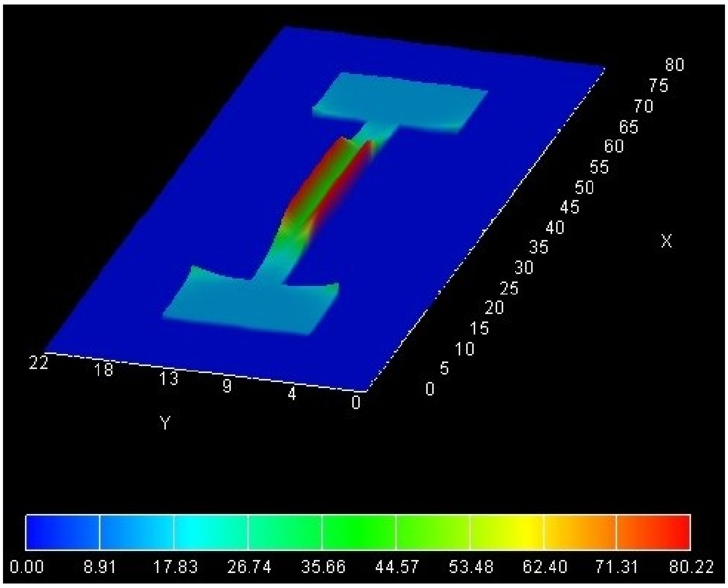


Figure 4.15.7.5: Electron density in units of $1 \cdot 10^{18}\mathrm{cm}^{-3}$ at $0.2\,\mathrm{V}$ gate voltage.

For comparison, Figure 4.15.7.6 shows the quantum mechanical electron density inside the MOSFET structure at $0.2\,\mathrm{V}$. The corresponding input which includes the quantum mechanical computation of the charge density is *DG-MOSFET-2-nm_zb_IV_2D_quantum.in*. One can clearly see that the electron density has the highest values in the middle of the channel and not at the $\mathrm{Si} - \mathrm{SiO_2}$ interfaces. This is because the wave functions tend to zero at the $\mathrm{Si} - \mathrm{SiO_2}$ interfaces. The peak values in the source and drain regions are due to classical densities because the quantum region did not extend over the whole source and drain regions.

I-V characteristics

In order to test the implementation of the three-dimensional drift-diffusion current, we performed a three-dimensional simulation of the Double Gate MOSFET. The corresponding input file is *IV_DG-MOSFET_Si_3D_classical_nnp*, where we assume complete ionization of the doping atoms. We further assume that the structure is homogeneous along the z-direction and assume the z-direction to be 10 nm long with grid spacing of 2 nm. The calculated current values in units of [A] can be found in *IV_characteristics.dat*. The current has to be divided by the length of the device along the z-direction, i.e. by 10 nm, in order to obtain it in units of [A/m]. Figure 4.15.7.7 confirms that the 3D results are in agreement with the 2D results.

This tutorial also exists for *nextnano³*.

Last update: 21/10/2024

4.15.8 Ultrathin-body DG MOSFET with 5-nm channel

Warning

This tutorial is under development.

Tags: #FET

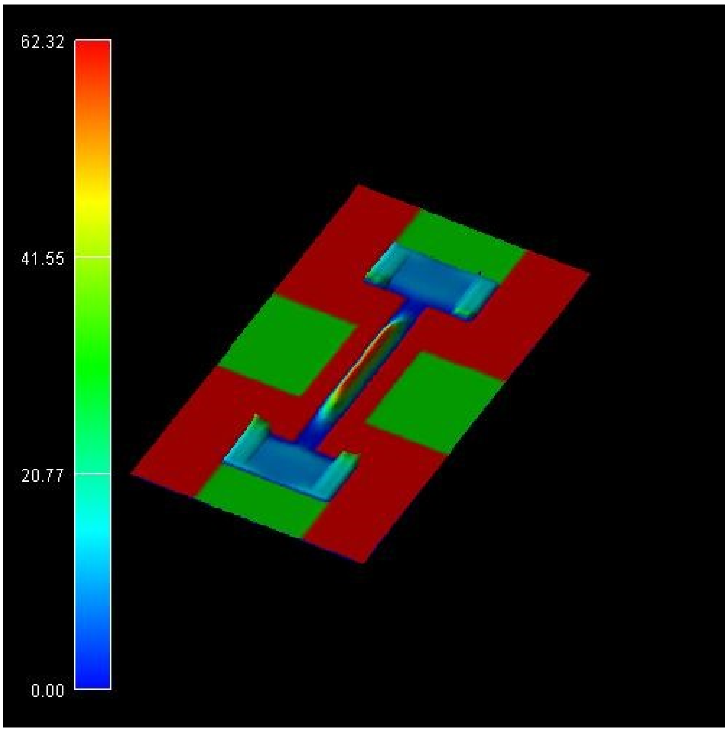


Figure 4.15.7.6: Quantum mechanical electron density in units of $1 \cdot 10^{18}\mathrm{cm}^{-3}$ at $0.2\,\mathrm{V}$ gate voltage.

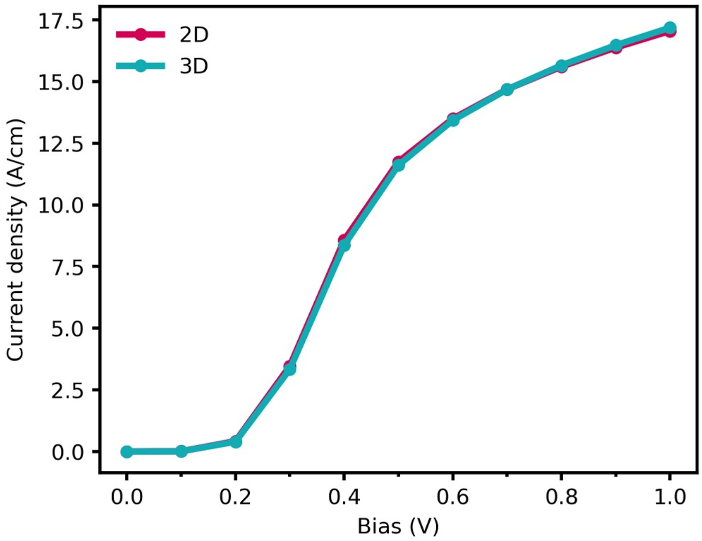


Figure 4.15.7.7: Comparison of the current-voltage characteristics between 2D and 3D simulations.

Contents

Files for the tutorial located in *nextnano++\examples\transistors*

- *DG-MOSFET-5-nm_zb_IV_Birner_APPA_2006_2D_cl.in* (Figure 4.15.8.1, Figure 4.15.8.2, Figure 4.15.8.3, Figure 4.15.8.4, Figure 4.15.8.7)
- *DG-MOSFET-5-nm_zb_IV_Birner_APPA_2006_2D_qm.in* (Figure 4.15.8.1, Figure 4.15.8.2, Figure 4.15.8.3, Figure 4.15.8.4, Figure 4.15.8.5, Figure 4.15.8.6, Figure 4.15.8.7)

Parameters

- `$temperature` – temperature of the crystal and electrons
- `$source_drain_voltage` – bias between the source and drain
- `$doping_conc` – doping concentration of the source and drain

Output files

- *Structure\last_region.avs.fld* (Figure 4.15.8.1)
- *Structure\contact.avs.fld* (Figure 4.15.8.1)
- *bias_xxxxx\bandedges_1d_along_y.dat* (Figure 4.15.8.2, Figure 4.15.8.3, Figure 4.15.8.4)
- *bias_xxxxx\density_electron_1d_along_y.dat* (Figure 4.15.8.2, Figure 4.15.8.3, Figure 4.15.8.4)
- *bias_xxxxx\Quantum\probabilities_quantum_region_Delta*.fld* (Figure 4.15.8.5, Figure 4.15.8.6)
- *IV_characteristics.dat* (Figure 4.15.8.7)

Introduction

This tutorial is related to the following publication: *[BirnerAPhys2006]* and it shows comparison of and quantum-mechanical simulations of an ultrathin-body double gate metal oxide semiconductor field effect transistor (DG MOSFET).

Structure

The main idea of a DG MOSFET is to control the Si channel very efficiently by choosing the Si channel width to be very small and by applying a gate contact to both sides of the channel. This concept helps to suppress short channel effects and leads to higher currents as compared with a MOSFET having only one gate. The structure in this tutorial consists of an intrinsic Si channel having the length 25 nm and the width 5 nm, as shown in Figure 4.15.8.1. The channel is connected to heavily n-type doped source and drain regions of length 10 nm each (constant doping profile with a concentration of $1 \cdot 10^{20}\,\mathrm{cm}^{-3}$, fully ionized). The gates have a length of 25 nm and are separated from the Si channel by a 1.5 nm thick SiO_2 layer with static dielectric constant $\epsilon = 3.9$.

In the simulations, a grid spacing of 1 nm and 0.5 nm are chosen for the x- and y-direction, respectively.

We apply a voltage of $V_{\mathrm{SD}} = 0.5\,\mathrm{V}$ to the drain contact. The gate voltage is varied from -0.3 V to 1.0 V in steps of 0.1 V. At the gate a Schottky barrier of 3.075 eV is chosen to mimic the gate electrode work function which has been assumed to be 4.1 eV.

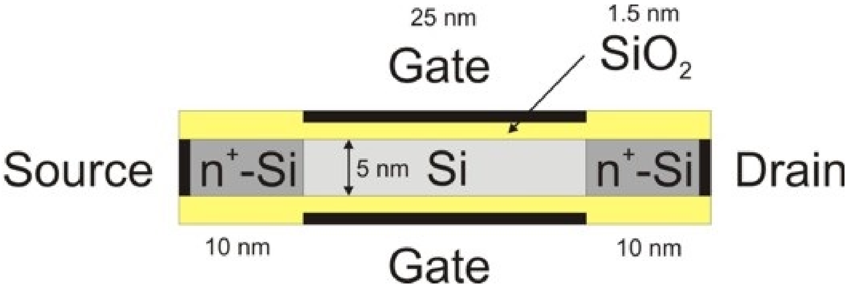


Figure 4.15.8.1: Geometry of the DG MOSFET, which consists of source contact, n-type doped source region (Si), Si channel (undoped), n-type doped drain region (Si), drain contact, SiO$_2$ insulator, top gate and bottom gate.

For the mobility we employ the arora mobility model. In this model, the mobility is assumed to depend on temperature ($T = 300\,\mathrm{K}$) and on the ionized dopants (N_D), but is independent of the electric field. Thus, we have two different electron mobilities:

- n-type doped Si region: $64.47\,\mathrm{cm}^2/\mathrm{Vs}$
- intrinsic Si region: $1429.2\,\mathrm{cm}^2/\mathrm{Vs}$

Electron density and conduction band profile

Figure 4.15.8.2 shows a slice through the middle of the device along the y-direction, i.e. through the gate contacts. The source drain voltage is $V_\mathrm{SD} = 0.5\,\mathrm{V}$, and the gate voltage is $V_\mathrm{G} = 0.7\,\mathrm{V}$. Two results are shown.

First, classical calculation with self-consistent solution of the two-dimensional Poisson and current equations. Here, the current equation is solved within a drift-diffusion model based on the classical density. For the classical calculation the input file *DG-MOSFET-5-nm_zb_IV_Birner_APPA_2006_2D_cl.in* should be used.

Second, quantum mechanical calculation with self-consistent solution of the two-dimensional Poisson, Schrödinger and current equations. Here, the current equation is solved within a drift-diffusion model based on the quantum mechanical density. For the quantum mechanical calculation the input file *DG-MOSFET-5-nm_zb_IV_Birner_APPA_2006_2D_qm.in* should be used.

The Fermi level is almost flat, i.e. constant (-0.249 eV) and very similar in both simulations. The conduction band edge in the Si channel is lower in the case of the quantum mechanical simulation. The main difference can be attributed to the electron density. The classical density has its maximum at the Si/SiO$_2$ interface, because $E_\mathrm{F,n} - E_\mathrm{C}$ has its maximum there. The quantum mechanical density is practically zero at the Si/SiO$_2$ interface, because the wave functions tend to zero due to the SiO$_2$ barrier. One can clearly see that the electron density has the highest values in the middle of the channel and not at the Si/SiO$_2$ interfaces.

Figure 4.15.8.3 and Figure 4.15.8.4 show the conduction band edge, charge densities and Fermi levels at the voltage of $V_\mathrm{G} = 0.3\,\mathrm{V}$ (closed channel) and $V_\mathrm{G} = 1.0\,\mathrm{V}$ (open channel), respectively. The quantum mechanical density has different shapes at different voltages (one maximum in the middle vs. two maxima off-the-center). Note that the axes for the electron density are scaled differently.

Electron wave functions

In our simulations we only consider electron states from the `Delta{}` conduction band. There are three Schrödinger equations that have to be solved each time having the following mass tensors that enter the Hamiltonian $H(x, y)$:

$$\begin{array}{lll}
1. & m_{xx} = m_\mathrm{longitudinal} \quad \text{and} \quad m_{yy} = m_{zz} = m_\mathrm{transversal}, \\
2. & m_{yy} = m_\mathrm{longitudinal} \quad \text{and} \quad m_{xx} = m_{zz} = m_\mathrm{transversal}, \\
3. & m_{zz} = m_\mathrm{longitudinal} \quad \text{and} \quad m_{xx} = m_{yy} = m_\mathrm{transversal},
\end{array}$$

with $m_\mathrm{longitudinal} = 0.916\,m_0$ and $m_\mathrm{transversal} = 0.190\,m_0$. Note that $m_{zz}(x, y)$ does not enter the Hamiltonian, but $m_{zz}(x, y)$ is used to calculate the quantum mechanical density ($k_\|$ dispersion). The quantum mechanical density for such a two-dimensional simulation is proportional to the square root of $m_{zz}(x, y)$. More precisely, the

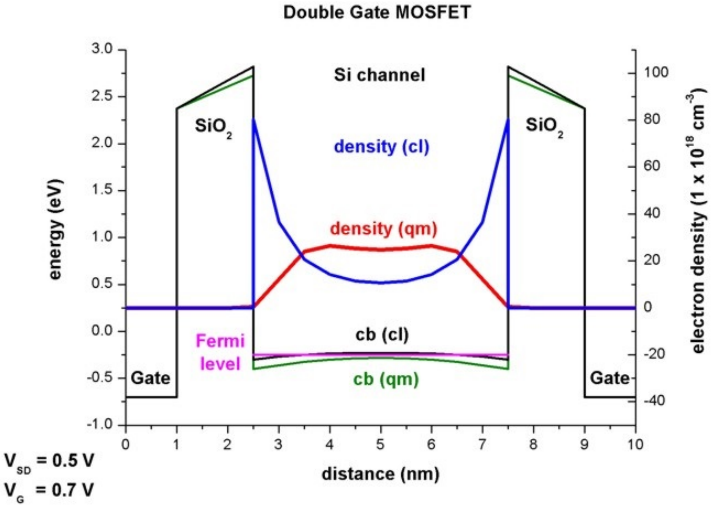


Figure 4.15.8.2: Conduction band profile, electron density and Fermi energy across the DB MOSFET structure at gate voltage $V_\mathrm{G} = 0.7\,\mathrm{V}$.

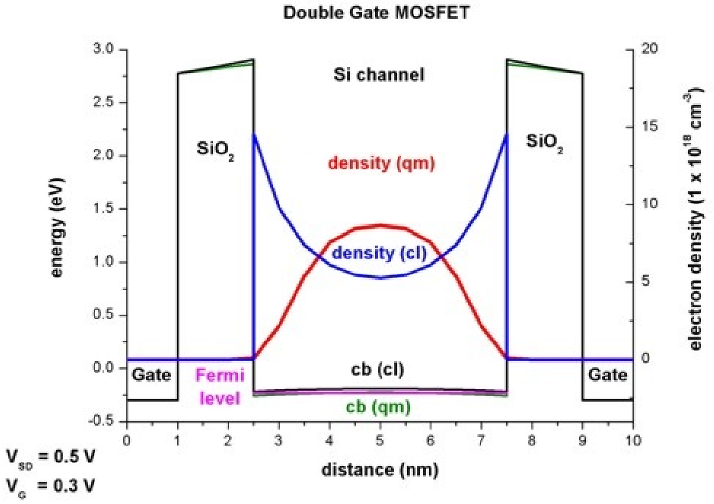


Figure 4.15.8.3: Conduction band profile, electron density and Fermi energy across the DB MOSFET structure at gate voltage $V_\mathrm{G} = 0.3\,\mathrm{V}$.

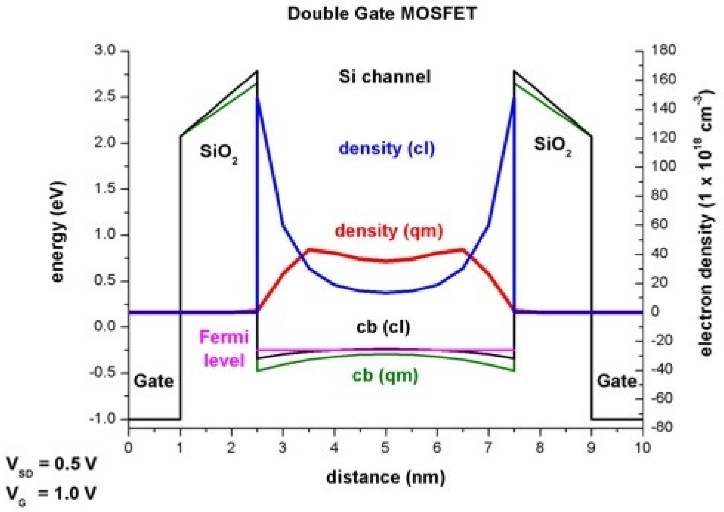


Figure 4.15.8.4: Conduction band profile, electron density and Fermi energy across the DB MOSFET structure at gate voltage $V_\mathrm{G} = 1\,\mathrm{V}$.

quantum mechanical density is obtained for each grid point by evaluating

$$n(x,y) = g_\mathrm{spin,valley}\sqrt{\frac{m_{xx}k_\mathrm{B}T}{2\pi\hbar^2}}\sum_i \left|\psi_i(x,y)\right|^2 \mathcal{F}_{-1/2}\left[(E_\mathrm{F}-E_i)/k_\mathrm{B}T\right],$$

which implies:

- summation over all eigenstates i
- evaluation of the square of the wave function $\left|\psi_i(x,y)\right|^2$
- weighting $\left|\psi_i(x,y)\right|^2$ with the Fermi-Dirac integral $\mathcal{F}_{-1/2}\left[(E_\mathrm{F}-E_i)/k_\mathrm{B}T\right]$, which includes the $\Gamma(1/2)$ pre-factor of the Fermi-Dirac integral
- multiplication by a factor which includes the square root of $m_{xx}k_\mathrm{B}T/(2\pi\hbar^2)$ and the spin and valley degeneracy $g_\mathrm{spin,valley}$.

Most of the wave functions are located in the source and drain region. In Figure 4.15.8.5 and Figure 4.15.8.6, the lowest wave functions ψ^2, which contribute to the quantum mechanical charge density in the region where the 1D slice was taken (i.e. in the middle of the device ($V_\mathrm{G} = 0.7\,\mathrm{V}$, $V_\mathrm{SD} = 0.5\,\mathrm{V}$)), are shown. The Fermi energy along the 1D slice through the middle of the device lies at -0.249 eV. The states are labelled from top to bottom:

- deg1: 35$^\mathrm{th}$ state with $E_{35} = -0.215\,\mathrm{eV}$ (ψ^2 is zero at the 1D slice which can be seen in Figure 4.15.8.6)
- deg1: 32$^\mathrm{nd}$ state with $E_{32} = -0.224\,\mathrm{eV}$ (25 meV above Fermi level)
- deg3: 13$^\mathrm{th}$ state with $E_{13} = -0.226\,\mathrm{eV}$ (23 meV above Fermi level)
- deg2: 32$^\mathrm{nd}$ state with $E_{32} = -0.250\,\mathrm{eV}$ (below Fermi level, corresponding to 2$^\mathrm{nd}$ subband)
- deg2: 25$^\mathrm{th}$ state with $E_{25} = -0.277\,\mathrm{eV}$ (below Fermi level, corresponding to 1$^\mathrm{st}$ subband)

Note

The states are sorted by eigenenergies, but their distance is not equivalent to their energy spacing.

Here, deg1 are the states originating from the valleys having the light, transversal mass perpendicular to the channel (i.e. these states have higher energies), deg3 are the states originating from the valleys having the light, transversal mass in the plane of the channel $m_{xx} = m_{yy} = m_\mathrm{transversal} = 0.190 m_0$ (high energies due to light masses) and deg2 are the states originating from the valleys having the heavy, longitudinal mass perpendicular to the channel as is the case in standard MOSFETs (i.e. these are the states that are occupied because the energies are the lowest).

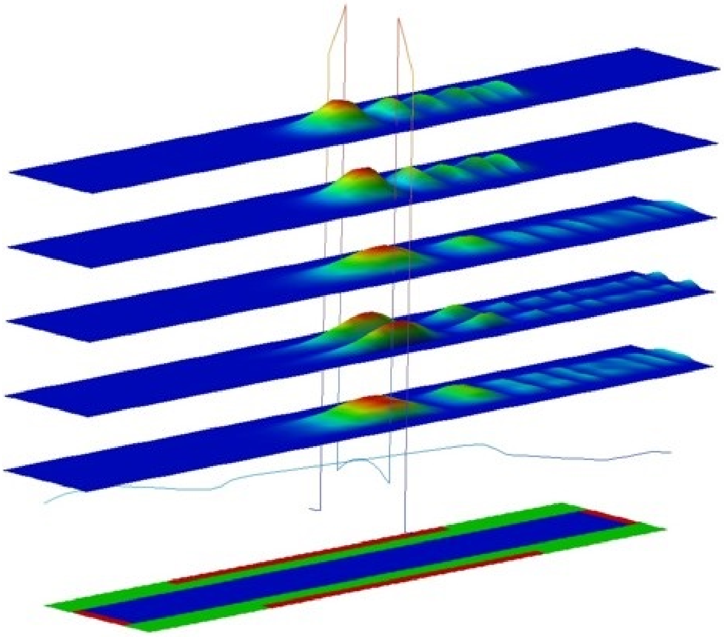


Figure 4.15.8.5: Wave functions located inside the Si-channel for $V_\mathrm{G} = 0.7\,\mathrm{V}$ and $V_\mathrm{SD} = 0.5\,\mathrm{V}$.

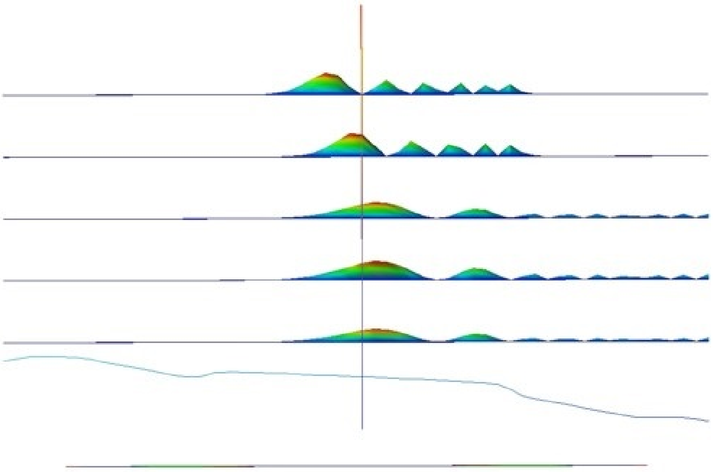


Figure 4.15.8.6: Wave functions located inside the Si-channel for $V_\mathrm{G} = 0.7\,\mathrm{V}$ and $V_\mathrm{SD} = 0.5\,\mathrm{V}$ (side-view).

I-V characteristics

The current-voltage (I-V) characteristic can be found in the following file: *IV_characteristics.dat*. The drain voltage has been kept constant at 0.5 V and the gate voltage varied from -0.3 V to 1.0 V. The resulting I-V curve is plotted in Figure 4.15.8.7. Due to the influence of quantum mechanics the current densities obtained from the quantum mechanical calculations are lower than from the classical calculations.

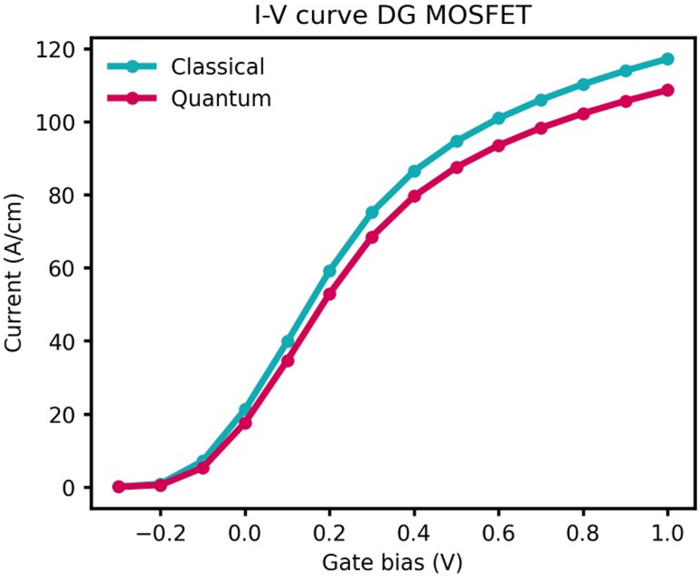


Figure 4.15.8.7: Comparison between classical and quantum mechanical calculation of the I-V characteristics.

Note that the absolute magnitude of the current is determined mostly by the mobility model. By using a more realistic mobility model that takes into account the dependency of the parallel and perpendicular electric fields, a smaller current would be obtained.

Last update: 21/10/2024

4.16 Magnetic Effects

4.16.1 Fock-Darwin states of a parabolic, anisotropic (elliptical) potential in a magnetic field

- *1D parabolic confinement along the x direction with* $\hbar\omega_0 = 4.6$ *meV (1D simulation)*
- *1D parabolic confinement along the y direction with* $\hbar\omega_0 = 6.1$ *meV (1D simulation)*
- *2D parabolic, anisotropic (elliptical) confinement with* $\hbar\omega_x = 4.6$ *meV and* $\hbar\omega_y = 6.1$ *meV - Fock-Darwin-like spectrum (2D simulation)*

In this tutorial we study the electron energy levels of a two-dimensional parabolic, anisotropic (elliptical) confinement potential that is subject to a magnetic field. Such a potential can be constructed by surrounding GaAs with an $Al_xGa_{1-x}As$ alloy that has a parabolic alloy profile in the (x,y) plane.

It is a good idea to get familiar with the results of a 2D parabolic and isotropic confinement beforehand: *Fock-Darwin states of a 2D parabolic potential in a magnetic field*

The input files used in this tutorial are the followings:

- *1DGaAs_ParabolicQW_infinite_4_6meV.in*
- *1DGaAs_ParabolicQW_infinite_6_1meV.in*
- *2DGaAs_BiParabolicEllipticQD_Austing_nnp.in*

First, it is necessary to study the energy states of a 1D parabolic confinement.

1D parabolic confinement along the x direction with $\hbar\omega_0 = 4.6$ meV (1D simulation)

- *1DGaAs_ParabolicQW_infinite_4_6meV.in*

For similar results and a discussion, we refer to this tutorial: Parabolic Quantum Well (GaAs / AlAs)

1D parabolic confinement along the y direction with $\hbar\omega_0 = 6.1$ meV (1D simulation)

- *1DGaAs_ParabolicQW_infinite_6_1meV.in*

For similar results and a discussion, we refer to this tutorial: Parabolic Quantum Well (GaAs / AlAs)

2D parabolic, anisotropic (elliptical) confinement with $\hbar\omega_x = 4.6$ meV and $\hbar\omega_y = 6.1$ meV - Fock-Darwin-like spectrum (2D simulation)

- *2DGaAs_BiParabolicEllipticQD_Austing.in/*_nnp.in*

The electron effective mass in GaAs is $m_e^* = 0.067m_0$. We assume this value for the effective mass in the whole region (i.e. also inside the AlGaAs alloy).

Ground state wave function (:math:`psi^2`)

The following figure shows the parabolic, anisotropic (elliptical) conduction band edge confinement potential, as well as the ground state wave function (ψ^2) at $B = 0$ T calculated in *nextnano++*. In the middle of the sample the conduction band edge is at 0 eV and at the boundary region the conduction band edge has the value 0.84 eV. The radii of the ellipse are 300 nm along the x axis and 226 nm along the y axis. The parabolic confinement along the x direction is: $\hbar\omega_x = 4.6$ meV

The parabolic confinement along the y direction is: $\hbar\omega_y = 6.1$ meV

Thus the ellipticity is roughly 4/3.

Fock-Darwin spectrum

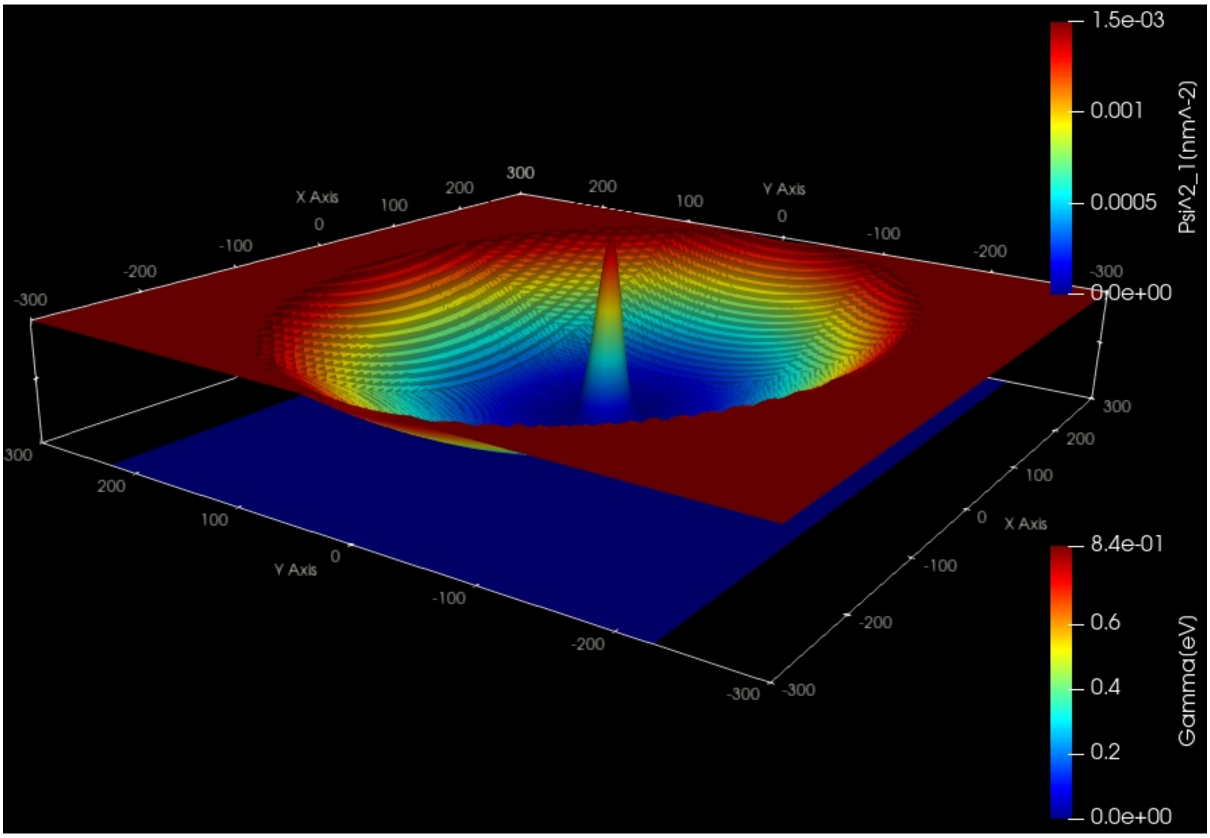
At zero magnetic field, the eigenvalues for such a system are given by:

$$E_{n_x,n_y} = (n_x + \frac{1}{2})\hbar\omega_x + (n_y + \omega\frac{1}{2})\hbar\omega_y n_x = n + \frac{1}{2}|l| - \frac{1}{2}|l| n_y = n + \frac{1}{2}|l| + \frac{1}{2}|l|$$

$$for\ n = 0, 1, 2, 3, ... \quad l = 0, \pm 1, \pm 2, ...$$

where n is a radial quantum number, l an angular momentum quantum number, ω_x and ω_y oscillator frequencies.

For more details, see A.V. Madhav, T. Chakraborty, Physical Review B 49, 8163 (1994).



The eigenvalue spectrum of a 2D parabolic and **isotropic** potential shows a shell-like structure: Energy levels of an "artificial atom" - 2D harmonic potential . For the **anisotropic elliptical** potential, this degeneracy at $B = 0$ T is lifted.

The following figure shows the calculated Fock-Darwin-like spectrum, i.e. the eigenstates as a function of magnetic field magnitude. This is the result of *nextnano*^{3} and each of these states is two-fold spin-degenerate. However, a magnetic field lifts this degeneracy (Zeeman splitting) but this effect is not taking into account in this tutorial.

Such a spectrum can be related to experimental transport measurements which give insight into the single-particle energy spectrum of a quantum dot.

The rectangles in the above figure are related to the figures of the following publications:

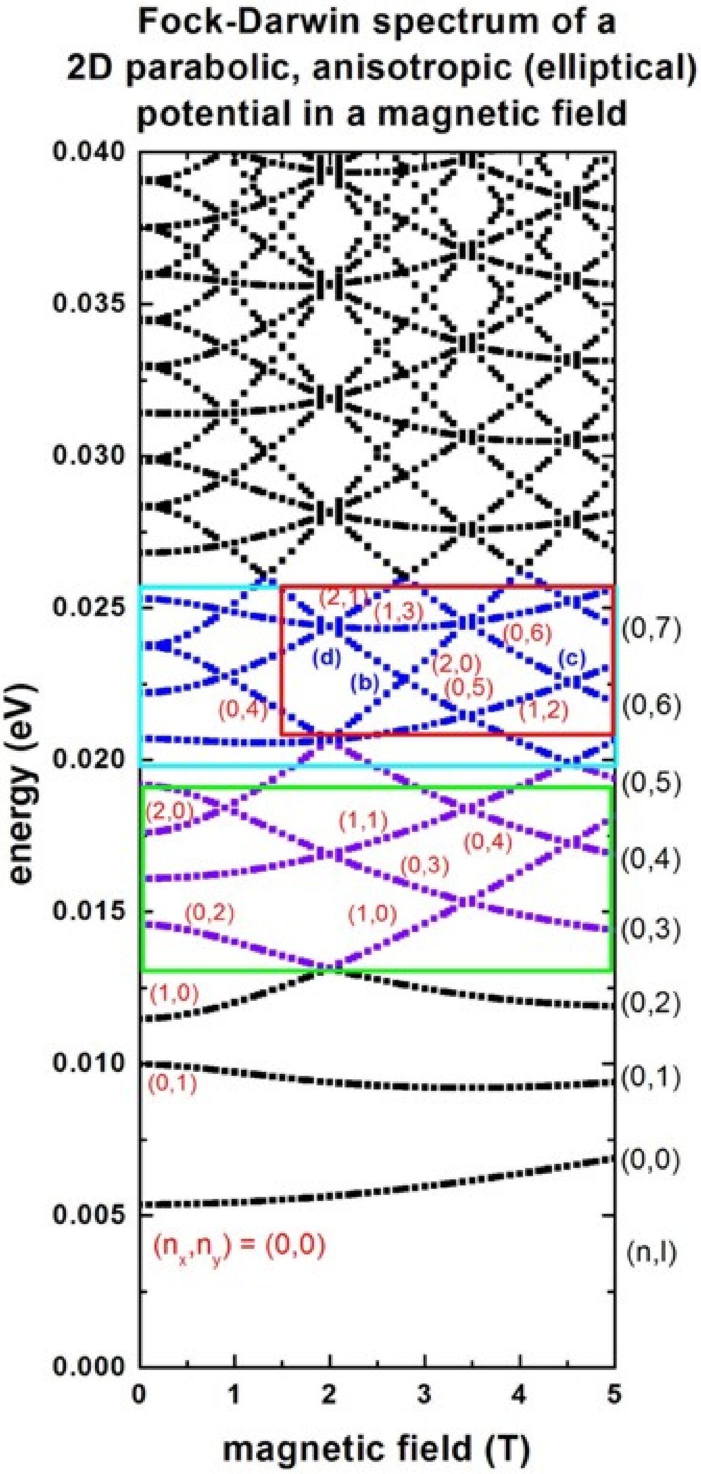
cyan rectangle: Fig.2 of

Two-level anti-crossings high up in the single-particle energy spectrum of a quantum dot
C. Payette, D.G. Austing, G. Yu, J.A. Gupta, S.V. Nair, B. Partoens, S. Amaha, S. Tarucha
arXiv:0710.1035v1 [cond-mat.mes-hall] (2007)

green rectangle and red rectangle: Fig.2(b) and Fig.3(a) of

Probing by transport the single-particle energy spectrum up to high energy of one quantum dot with the ground state of an adjacent weakly coupled quantum dot
D.G. Austing, G. Yu, C. Payette, J.A. Gupta, M. Korkusinski, G.C. Aers
physica status solidi (a), 508 (2007)

(Comments on red rectangle: In Fig. 3(a) of the publication by Austing et al., the ground state energy has been subtracted from the excited states. Thus the slope of the energy spectrum look slightly different.)


Fock-Darwin spectrum of a
2D parabolic, anisotropic (elliptical)
potential in a magnetic field
energy (eV)
magnetic field (T)
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
1
2
3
4
5
(2,1)
(1,3)
(0,6)
(d)
(2,0)
(c)
(b)
(0,5)
(0,4)
(1,2)
(2,0)
(1,1)
(0,4)
(0,3)
(0,2)
(1,0)
(1,0)
(0,1)
(n_x,n_y) = (0,0)
(0,7)
(0,6)
(0,5)
(0,4)
(0,3)
(0,2)
(0,1)
(0,0)
(n,l)

It is interesting to note that there are exact crossings in the calculated spectrum whereas the experiment reveals anti-crossings. In the first reference, this difference on crossings is regarded as a crue to investigate the deviations between the confining potential of realistic dots used in the experiment and the idealistic parabolic potential assumed in the calculation.

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.16.2 Fock-Darwin states of parabolic, isotropic potential in a magnetic field

Header

Input files

- *2DGaAs_BiParabolicQW_4meV_GovernalePRB1998_nnp.in*
- *2DGaAs_BiParabolicQW_3meV_FockDarwin_nnp.in*

Introduction

In this tutorial, we study the electron energy levels of a two-dimensional parabolic confinement potential that is subject to a magnetic field. Such a potential can be constructed by surrounding GaAs with an $Al_xGa_{1-x}As$ alloy that has a parabolic alloy profile in the (x,y) plane.

The magnetic field $\mathbf{B}$ is oriented along the z direction. $\mathbf{B}$ is the rotation of the vector potential $\mathbf{A}$ so, in this case, we can always take the z-component of the vector potential as 0. Thus the motion in the z direction is not influenced by the magnetic field and that of a free particle with energies and wave functions given by:

$$E_z = \frac{\hbar^2 k_z^2}{2m_e^*}$$

$$\psi(z) = \exp(\pm i k_z z)$$

For that reason, we do not include the z direction into our simulation domain, and thus only simulate in the (x,y) plane (two-dimensional simulation).

This tutorial consists of two parts. First we benchmark the *nextnano*++ code to the numerical calculation in *[GovernalePRB1998]*. Second we reproduce some figures of *[KouwenhovenRPP2001]*.

The figures provided in this tutorial is the results of *nextnano*++ input files.

Note

When `magnetic_field` is specified in a 2D or 3D simulation of *nextnano*++, the Pauli equation, in which both spin eigenfunctions are taken into consideration, is calculated instead of Schrödinger equation. Since the splitting of energy levels due to the spin is small compared to the difference of energy levels, we call the two states split from ith eigenstates as “ith eigenstates with up-spin” or “ith eigenstates with down-spin”.

2D parabolic confinement with $\hbar\omega_0 = 4$ meV

We want to benchmark the *nextnano++* code to the numerical calculation in *[GovernalePRB1998]*.

Input file *2DGaAs_BiParabolicQW_4meV_GovernalePRB1998.in/*_nnp.in* aims to reproduce the figures of eigenvalues, ground state and 14th excited state probability densities, and ground state energy as a function of magnetic field magnitude (Fig.1, 2, 3 and 4 of the paper).

The GaAs sample extends in the x and y directions (i.e. this is a two-dimensional simulation) and has the size of 240 nm x 240 nm. At the domain boundaries we employ Dirichlet boundary conditions to the Schrödinger equation, i.e. infinite barriers. The grid is chosen to be rectangular with a grid spacing of 2.4 nm, in agreement with *[GovernalePRB1998]*.

A two-dimensional parabolic confinement potential is constructed by surrounding GaAs with an $Al_xGa_{1-x}As$ alloy that has a parabolic alloy profile in the (x,y) plane. This is chosen so that the eclectron ground state has the energy: $E_1 = \hbar\omega_0 = 4$ meV (without magnetic field).

The magnetic field is oriented along the z direction, i.e. it is perpendicular to the simulation plane which is oriented in the (x,y) plane). (In *nextnano++*, the direction is automatically set to the direction perpendicular to the simulation plane.) We calculate the eigenstates for different magnetic field strengths (1 T, 2 T, . . . , 20 T), i.e. we make use of the magnetic field sweep. Since *nextnano++* does not have this feature for magnetic_field so far, please use the "Template" feature of *nextnanomat* (See the last section of *— FREE — Double Quantum Well* .)

```
global{
    ...
    magnetic_field{
        strength = $STRENGTH
        #direction = [,,] # We must not specify this in 1D or 2D simulation
    }
}
```

Magnetic length and cyclotron frequency

A useful quantitiy is the magnetic length (or Landau magnetic length) which is defined as:

$$l_B = \left(\frac{\hbar}{m_e^*\omega_c}\right)^{1/2} = \left(\frac{\hbar}{|e|B}\right)^{1/2}$$

It is independent of the mass of the particle and depends only on the magnetic field strength:

- 1 T: $l_B = 25.6556$ nm
- 2 T: $l_B = 18.1413$ nm
- 3 T: $l_B = 14.8123$ nm
- . . .
- 20T: $l_B = 5.7368$ nm

The electron effective mass in GaAs is $m_e^* = 0.067m_0$. We assume this value for the effective mass in the whole region (i.e. also inside the AlGaAs alloy). In the above formula, ω_c is the cyclotron frequency:

$$\omega_c = \frac{|e|B}{m_e^*}$$

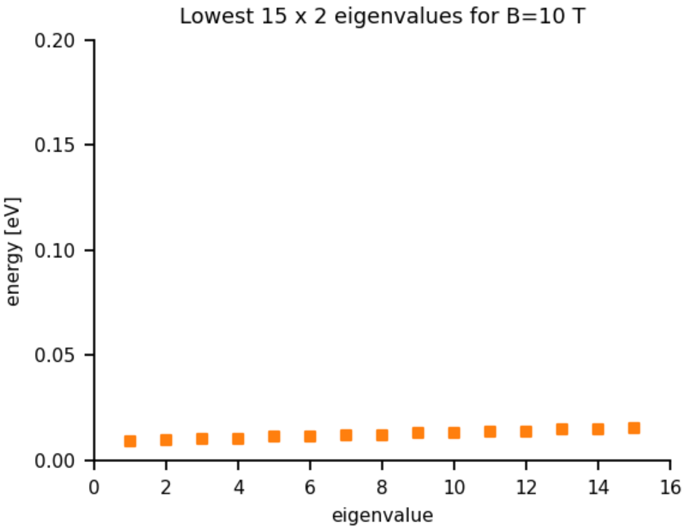
Thus for the electrons in GaAs, where $m_e^* = 0.067m_0$, it holds for the different magnetic field strengths:

- 1 T: $\hbar\omega_c = 1.7279$ meV
- 2 T: $\hbar\omega_c = 3.4558$ meV
- 3 T: $\hbar\omega_c = 5.1836$ meV
- . . .
- 20T: $\hbar\omega_c = 34.5575$ meV

Results

Lowest 15 eigenvalues

The following figure shows the lowest fifteen eigenvalues for a magnetic field magnitude of $B = 10$ T. It is in perfect agreement with Fig. 1 of *[GovernalePRB1998]*. The ground state has the energy $E_{1\uparrow} = 9.38$ meV and $E_{1\downarrow} = 9.55$ (at $B = 10$ T). The spin-split energy, $\frac{e\hbar B}{2m_e^*}$ is 0.174 meV, is calculated from our result as 0.174 meV which is constant in all of the pair of spin states.



Probability densities (ψ^2)

The following figure shows the probability density of the ground state with up-spin (ψ^2) for a magnetic field magnitude of $B = 10$ T. It is in perfect agreement with Fig. 2(a) of *[GovernalePRB1998]*. The ground states has the energy $E_{1,\uparrow} = 9.38$ meV and $E_{1,\downarrow} = 9.55$ (at $B = 10$ T).

The left, vertical axis shows ψ^2 in units of nm^{-2} (the peak value is 0.00267 nm^{-2}).

In the same figure, the parabolic conduction band edge confinement potential is also shown. The above axis shows the colormap of the conduction band edge values. In the middle of the sample the conduction band edge is 0 eV, and at the boundary region, the conduction band edge has the value 0.1014 eV.

The following figure shows the probability density (ψ^2) of the 14th excited state (up-spin) (i.e. $E_{15,\uparrow}$) for a magnetic field magnitude of $B = 10$ T. It is in perfect agreement with Fig. 3(a) of *[GovernalePRB1998]*. 14th excited states have the energy $E_{15,\uparrow} = 21.71$ and $E_{15,\downarrow} = 21.88$ meV (at $B = 10$ T). The left, vertical axis shows ψ^2 in units of nm^{-2} (the peak value is 0.000283 nm^{-2}).

In the same figure, parabolic conduction band edge confinement potential is also shown. The above axis shows the colormap of the conduction band edge values. In the middle of the sample the conduction band edge is 0 eV, and at the boundary region, the conduction band edge has the value 0.1014 eV.

Ground state energy vs. magnetic field magnitude

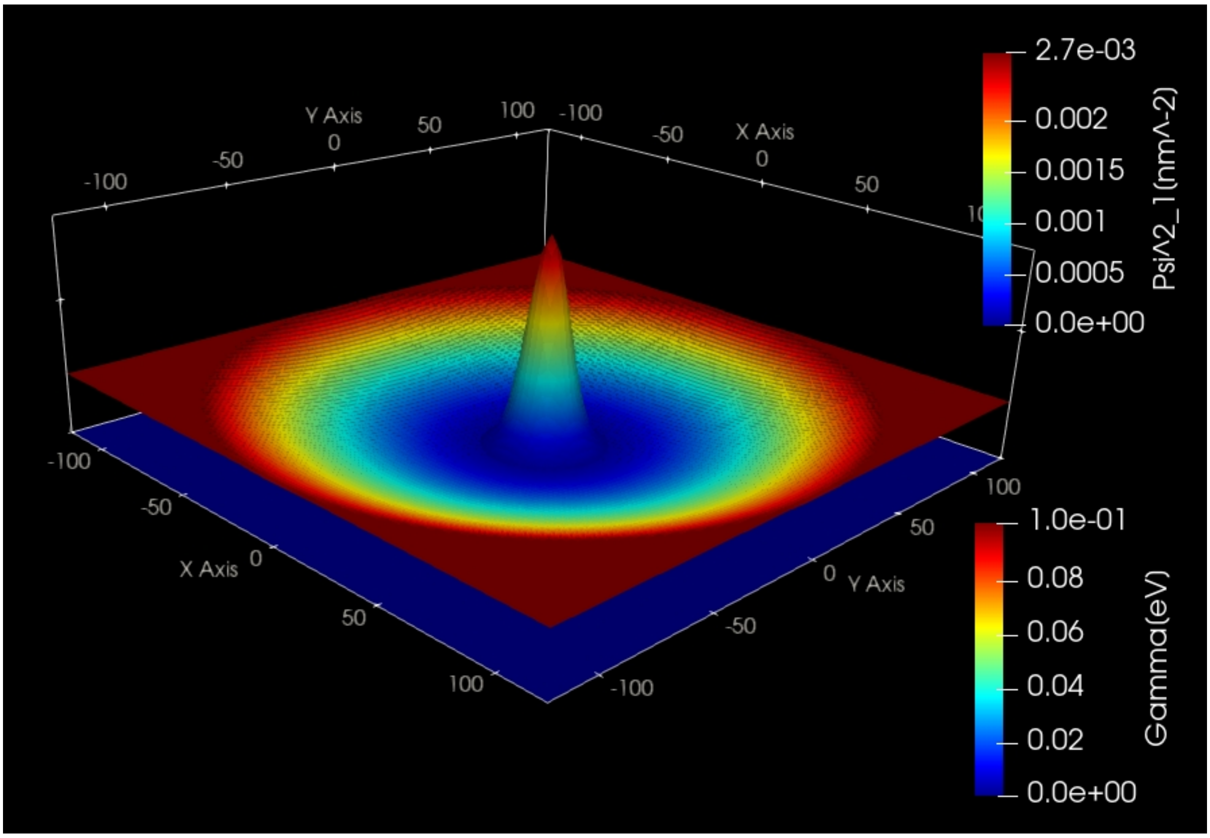
The following figure shows the ground state energy as a function of magnetic field magnitude. It is in perfect agreement with Fig.4 of *[GovernalePRB1998]*. The ground state has the energy $E_1 = 4.04$ meV (spin-degenerated).

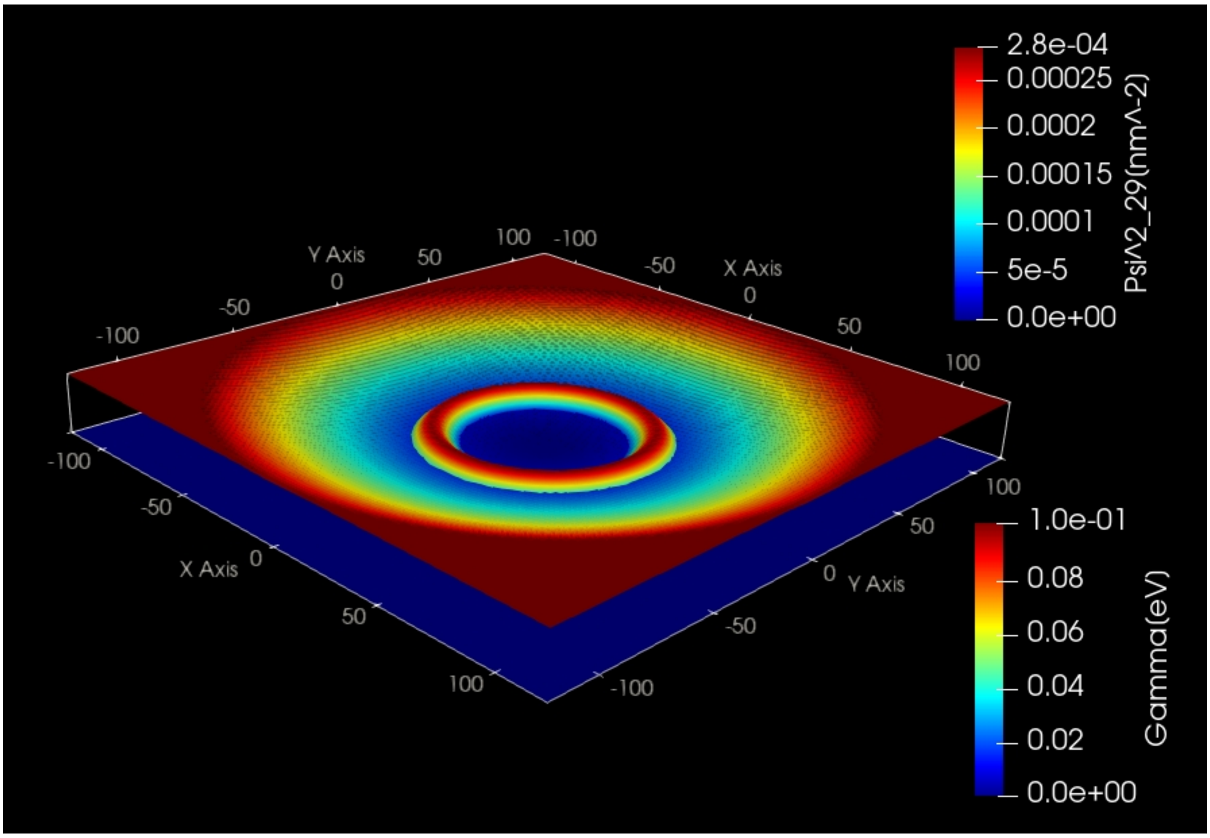
The following figure shows the magnetic field stlength dependence of the spin-split energy $(E_{1,\downarrow} - E_{1,\uparrow})$. The formula of the split energy in the Pauli equation is $\frac{e\hbar B}{2m_e^*}$. We can see the proportionality is reproduced in our calculation. The factor is calculated as 0.0174 [meV/T].

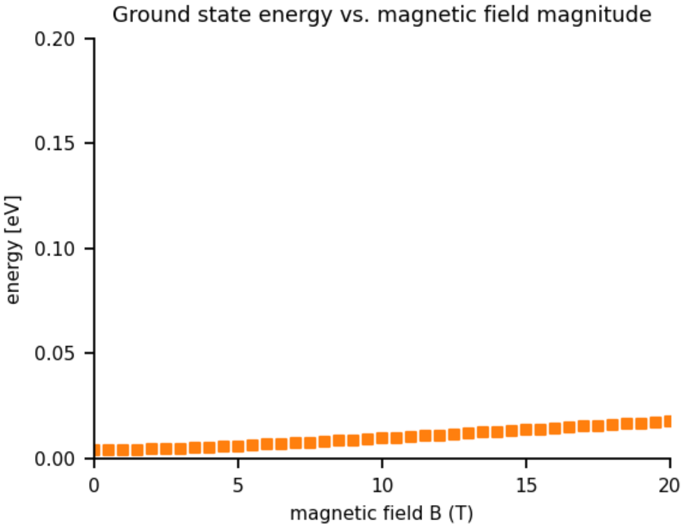
2D parabolic confinement with $\hbar\omega_0 = 3$ meV - Fock-Darwin spectrum

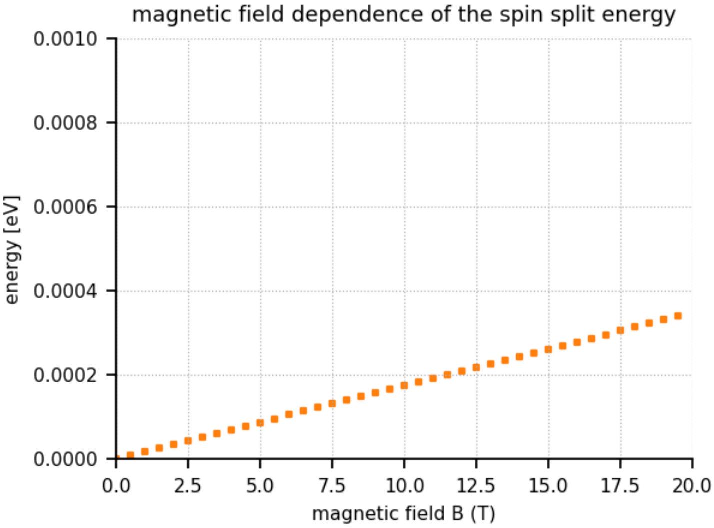
Next we reproduce some of the figures of *[KouwenhovenRPP2001]*.

Input file *2DGaAs_BiParabolicQW_3meV_FockDarwin.in/*_nnp.in* aims to reproduce the figures of the eigenvalues as a function of magnetic field magnitude and the probability densities of some of eigenstates (Figs. 5(a) and


Y Axis
X Axis
-100
-50
0
50
100
2.7e-03
0.002
0.0015
0.001
0.0005
0.0e+00
Psi^2_1(nm^-2)
1.0e-01
0.08
0.06
0.04
0.02
0.0e+00
Gamma(eV)


Y Axis
X Axis
-100
-50
0
50
100
2.8e-04
0.00025
0.0002
0.00015
0.0001
5e-5
0.0e+00
Psi^2_29(nm^-2)
1.0e-01
0.08
0.06
0.04
0.02
0.0e+00
Gamma(eV)


Ground state energy vs. magnetic field magnitude
energy [eV]
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
magnetic field B (T)


magnetic field dependence of the spin split energy
energy [eV]
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
magnetic field B (T)

6(a) (which are analytical results) of the paper).

The GaAs sample extends in the x and y directions (i.e. this is a two-dimensional simulation) and has the size of 600 nm x 600 nm. At the domain boundaries we employ Dirichlet boundary conditions to the Schrödinger equation, i.e. infinite barriers.

A two-dimensional parabolic confinement potential is constructed by surrounding GaAs with an $\mathrm{Al_xGa_{1\text{-}x}As}$ alloy that has a parabolic alloy profile in the (x,y) plane. This is chosen so that the eclectron ground state has the energy: $E_1 = \hbar\omega_0 = 3$ meV (without magnetic field) in agreement to the paper.

The eigenvalues of a two-dimensional parabolic potential that is subject to a magnetic field can be solved analytically. The spectrum of the resulting eigenstates is known as the **Fock-Darwin states** (1928):

$$E_{n,l} = (2n + |l| + 1)\hbar[w_0^2 + \frac{1}{4}\omega_c^2]^{1/2} - \frac{1}{2}l\hbar\omega_c \quad for\ n = 0, 1, 2, 3, ...\ and\ l = 0, \pm1, \pm2, ...$$

Note that the last term is ω_c and not ω_0 as in *[KouwenhovenRPP2001]*. ($\omega_c = \frac{|e|B}{m_e^*}$ = cyclotron frequency, as described before.)

Each of these states is two-fold spin-degenerate. A magnetic field lifts this degeneracy (Zeeman splitting). This effect is taking into account only in the input file of *nextnano*++ but this splitting is small compared to the scale of $E_{n,l}$.

The degeneracy of the eigenvalues for zero magnetic field is as follows:

- the ground state is not degenerate
- the second state is two-fold degenerate
- the third state is three-fold degenerate
- the forth state is four-fold degenerate
- …

Applying a magnetic field, these degeneracies are lifted as the following fugure.

Results

Fock-Darwin spectrum

The following figure shows the calculated Fock-Darwin spectrum, i.e. the eigenstates as a function of magnetic field magnitude. The figure is in excellent agreement with Fig. 5(a) of *[KouwenhovenRPP2001]*.

Probability densities (ψ^2)

The following figure show the probability densities (ψ^2) of some of these eigenstates for a magnetic field of $B = 0.05$ T. All of them are the up-spin states. The label of the colorbar shows the actual number of each eigenstates specified in the data file. For example, 5th state in this figure has the label “Psi^2_9[nm^-9]”.

The figures are in excellent agreement with Fig. 6(a) of *[KouwenhovenRPP2001]*.

The parabolic conduction band edges are also shown.

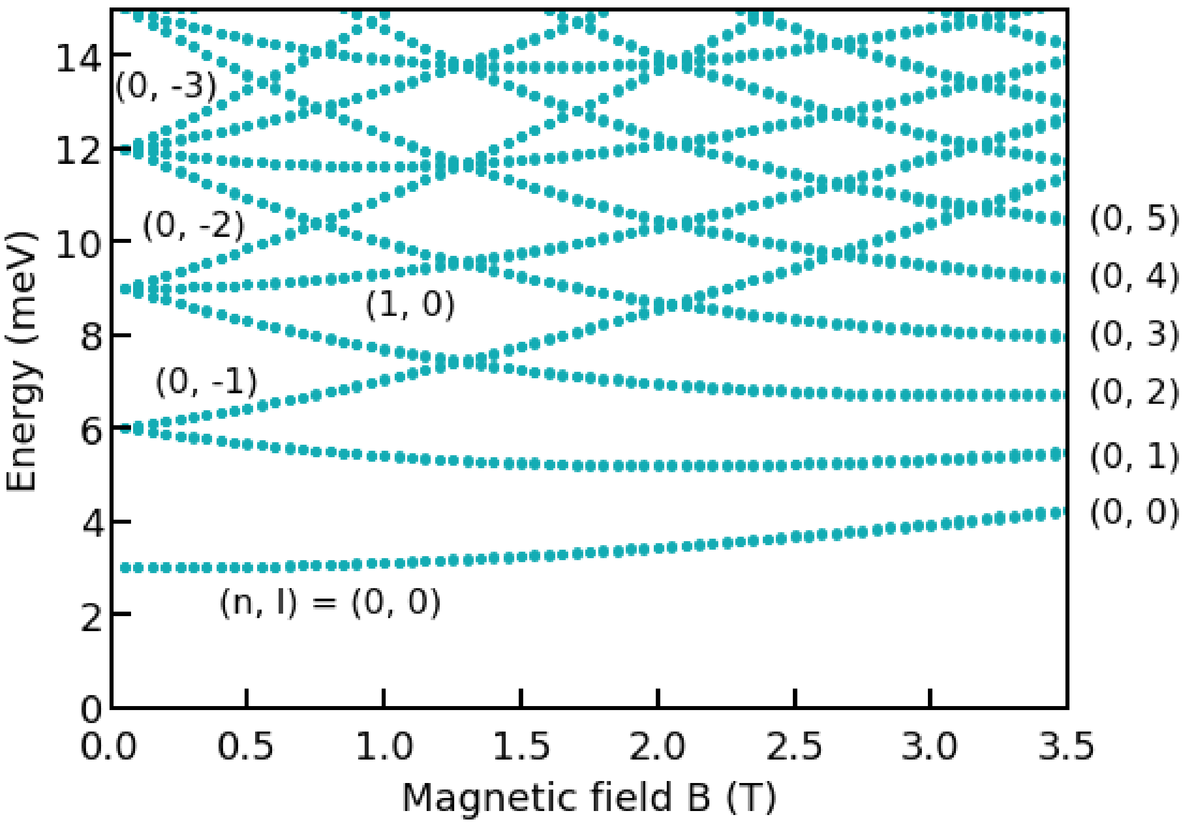
Fock-Darwin spectrum in a very high magnetic fields

The following figure shows the magnetic field dependence of the lowest 30 eigen values (0~4T) and lowest 60 eigenvalues (4~70T). We can see that eventually all states are becoming degenerate Landau levels for very high magnetic fields. The reason is that the electrons are confined only by the magnetic field and not any longer by the parabolic conduction band edge.

The red line shows the fan of the lowest Landau level at $1/2\hbar\omega_c$. The higher lying states (not shown) will collect in the second, third, …, and higher Landau fans (not shown).

The left part of the figure (black region) contains exactly the same Fock-Darwin spectrum that has been shown in the figure further above (from 0 T to 3.5 T).

This tutorial also exists for *nextnano³*.



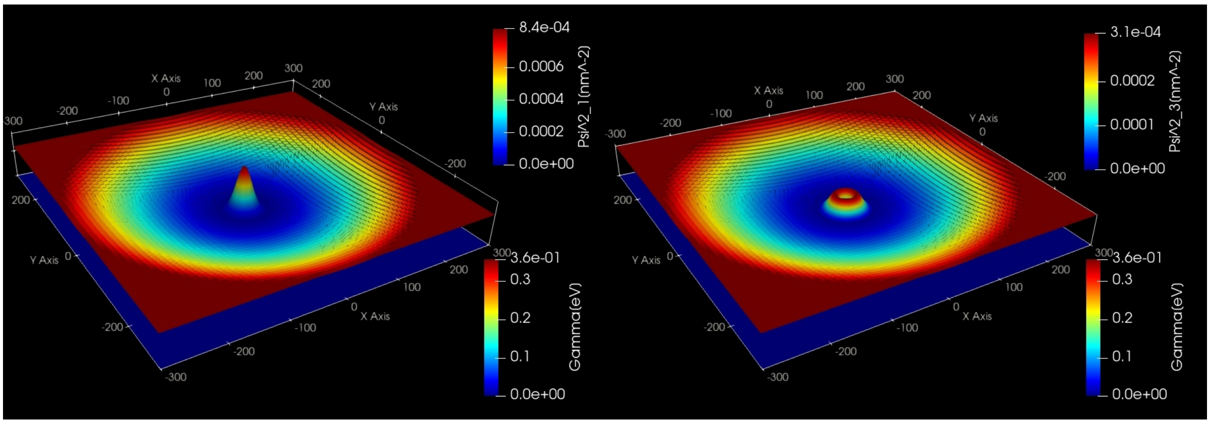


Figure 4.16.2.1: left: $(n,l) = (0,0)$ (1st), right: $(n,l) = (0,1)$ (2nd)

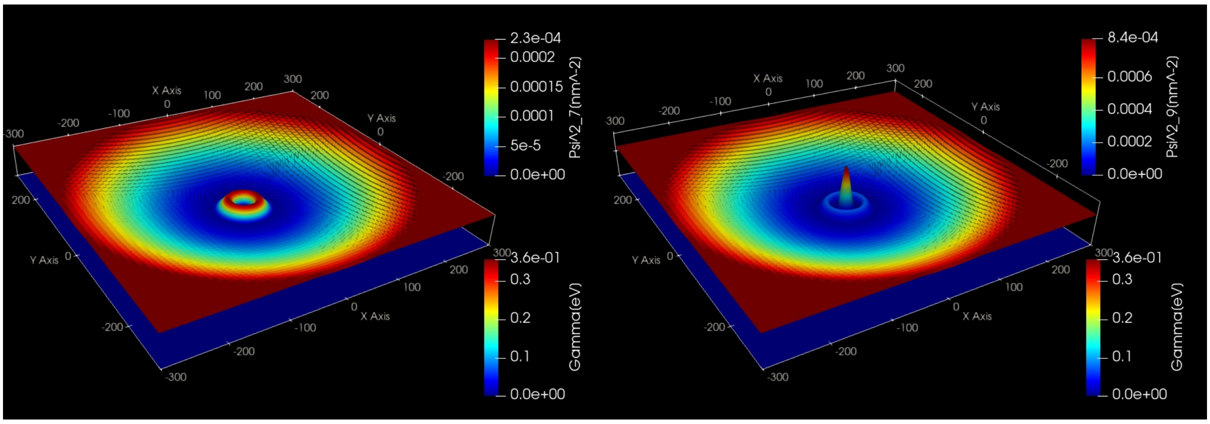


Figure 4.16.2.2: left: $(n,l) = (0,2)$ (4th), right: $(n,l) = (1,0)$ (5th)

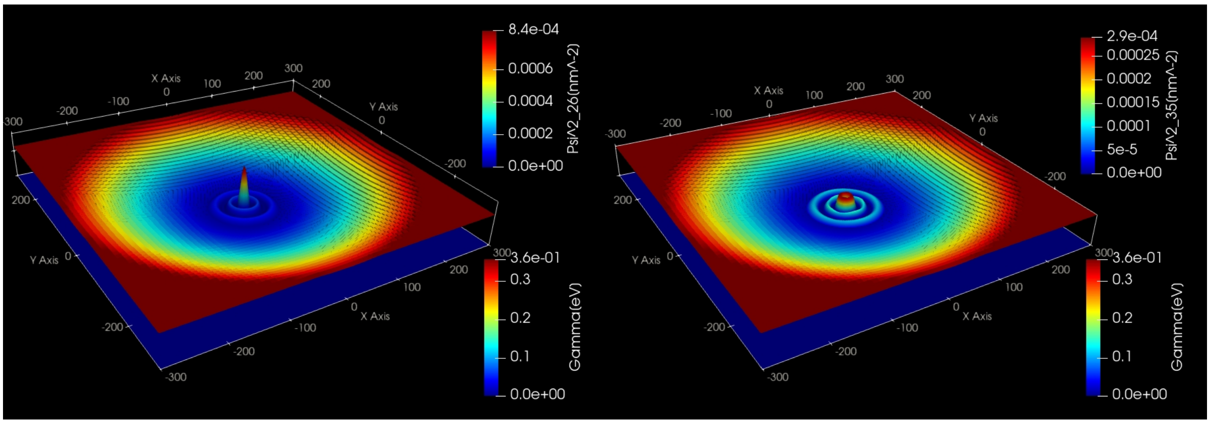
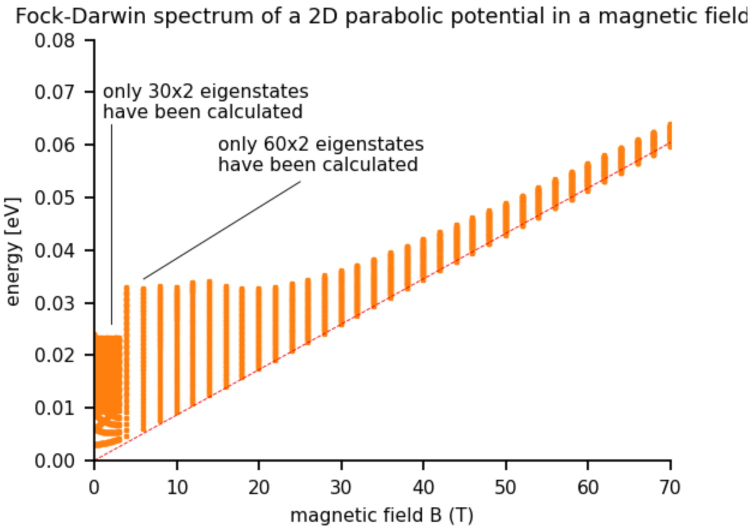


Figure 4.16.2.3: left: $(n, l) = (2, 0)$ (13th), right: $(n, l) = (2, 2)$ (18th)



Last update: nnnn/nn/nn

4.16.3 Landau levels of a bulk GaAs sample in a magnetic field

In this tutorial, we study the electron energy levels of a bulk GaAs sample that is subject to a magnetic field.

The input files are the followings:

- *2DBulkGaAs_LandauLevels_nn3.in / *_nnp.in*

Simulation details

The magnetic field is oriented along the z direction. The motion in the z direction is not influenced by the magnetic field and is thus that of a free particle with energies and wave functions given by:

$$E_z = \frac{\hbar^2 k_z^2}{2m^*}$$

$$\psi(z) = \exp(\pm i k_z z)$$

For that reason, we do not include the z direction into our simulation domain, and thus only simulate in the (x,y) plane (two-dimensional simulation).

This plane has the size of 300 nm × 300 nm and consists of GaAs. At the domain boundaries we employ Dirichlet boundary conditions to the Schrödinger equation, i.e. infinite barriers.

We calculate the eigenstates for different magnetic field strengths (1 T, 2 T, 3 T).

```
global{
    ...
    magnetic_field{
        strength = $B # [T]
    }
}
```

Magnetic length and cyclotron frequency

A useful quantitiy is the magnetic length (or Landau magnetic length) which is defined as:

$$l_B = \left(\frac{\hbar}{m_e^* \omega_c}\right)^{1/2} = \left(\frac{\hbar}{|e|B}\right)^{1/2}$$

It is independent of the mass of the particle and depends only on the magnetic field strength:

- 1 T: $l_B = 25.6556$ nm
- 2 T: $l_B = 18.1413$ nm
- 3 T: $l_B = 14.8123$ nm

In the above formula, ω_c is the cyclotron frequency:

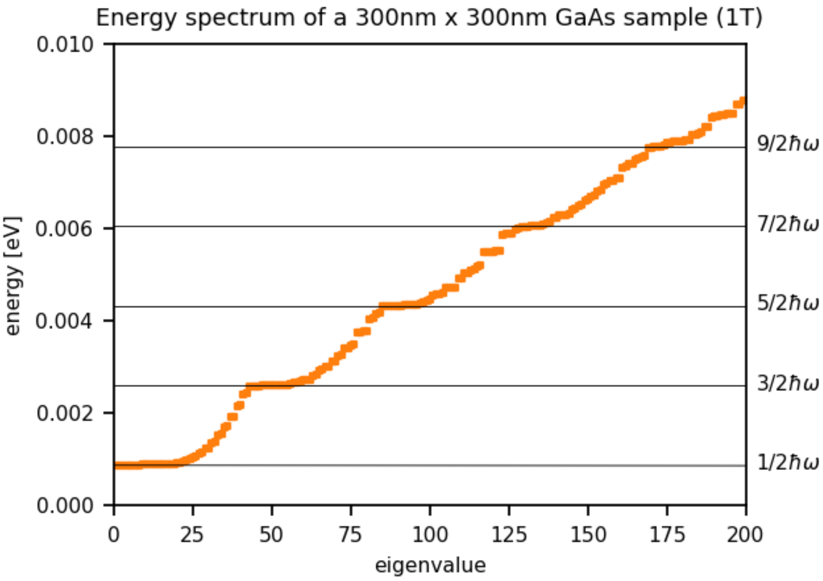
$$\omega_c = \frac{|e|B}{m_e^*}$$

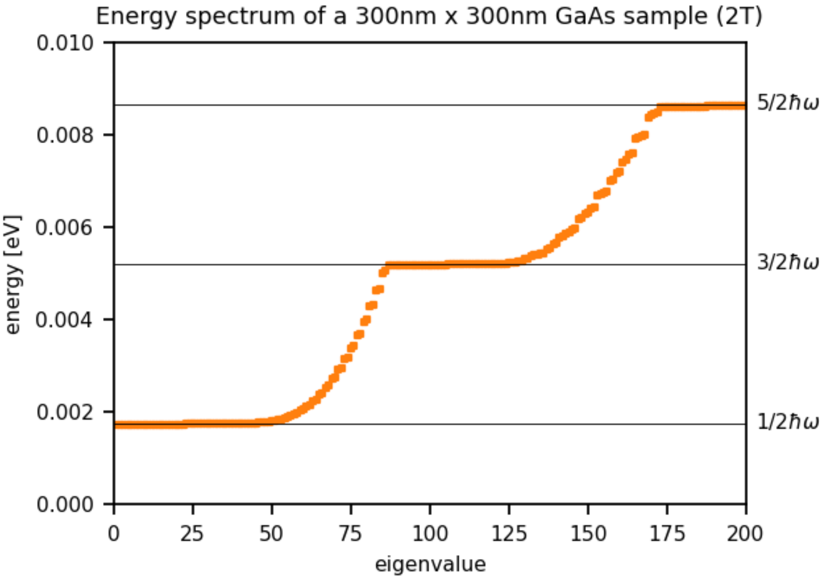
Thus for the electrons in GaAs, where $m_e^* = 0.067 m_0$, it holds for the different magnetic field strengths:

- 1 T: $\hbar\omega_c = 1.7279$ meV
- 2 T: $\hbar\omega_c = 3.4558$ meV
- 3 T: $\hbar\omega_c = 5.1836$ meV

Results

The calculated energy spectra for different magnetic fields (1 T, 2 T, 3 T) are as follows:



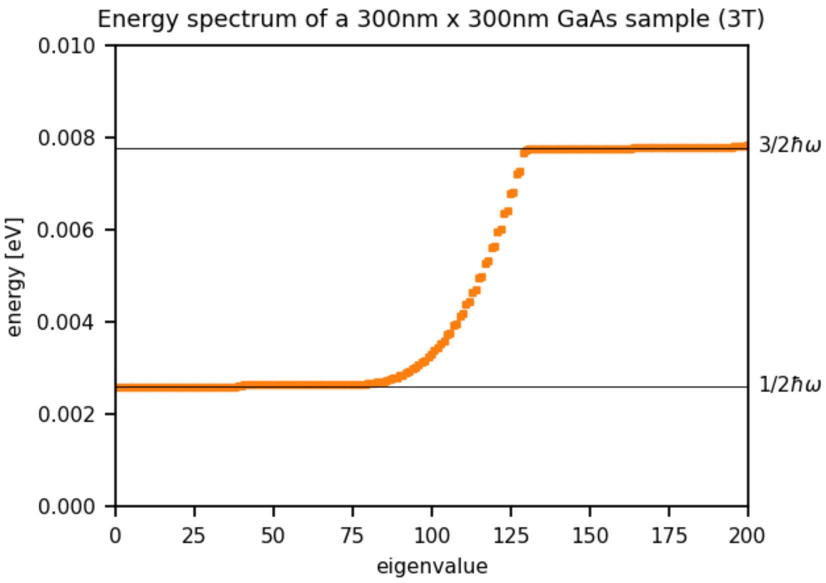


Landau levels

The Landau levels are analytically given by

$$E_n = \left(n - \frac{1}{2}\right) \hbar\omega_c$$

where $n = 1, 2, 3, ...$



The number of states for each Landau level can be calculated as follows (see P.Y. Yu, M. Cardona, Fundamentals of Semiconductors, p. 536, 3rd ed.):

$$N = L_x L_y \frac{|e|B}{h} = \frac{L_x L_y}{2\pi l_B^2}$$

where L_x and L_y are the lengths in the x and y directions (300 nm in this example) and l_B is the magnetic length. Here we ignore spin.

- N(1 T) = 21.76 ~ **22** states per Landau level (in the figure above: 42)
- N(2 T) = 43.52 ~ **44** states per Landau level (in the figure above: 86)
- N(3 T) = 65.29 ~ **65** states per Landau level (in the figure above: 130)

When `magnetic_field{}` is specified, *nextnano++* caculates the Schrödinger-Pauli equation, which takes into account the spin. Since the interaction energy between the spin and magnetic field is small compared to the separation of Landau levels, the number of states per Landau level calculated by *nextnano++* is almost double of the analytical result that ignores the spin.

Energy eigenvalues

For the calculations, we used the symmetric gauge $A = -\frac{1}{2} r \times B = \frac{1}{2} B \times r$ leading to the following energies (see J.H. Davies, The Physics of Low-Dimensional Semiconductors, p. 222):

$$E_{n,l} = \left(n + \frac{1}{2} l + \frac{1}{2} |l| - \frac{1}{2} \right) \hbar\omega_c$$

One can see that all states having a negative value of l are degenerate with the states with $l = 0$, i.e. the allowed energies are independent of l if $l < 0$ (for the same n). The energies increase if l increases (for $l > 0$ and for the same n).

This tutorial also exists for *nextnano³*.

Last update: nnnn/nn/nn

4.16.4 Hole wave functions in a quantum wire subjected to a magnetic field

Attention

This tutorial is under construction

Input files:

- *QWR-magnetic-field_InAs_2D_sg_nnp.in*
- *QWR-magnetic-field_InAs_2D_6kp_nnp.in*

Scope:

This tutorial aims to calculate the hole wavefunctions in a quantum wire, which is subject to an applied magnetic field.

Output files:

- *bias_00000/Quantum/energy_spectrum_quantum_region_HH_00000.dat*
- *bias_00000/Quantum/probabilities_quantum_region_HH.fld*
- *bias_00000/Quantum/energy_spectrum_quantum_region_kp6_00000.dat*
- *bias_00000/Quantum/probabilities_quantum_region_kp6_00000.fld*

Structure

Similar to the 1D confinement in a quantum well, it is possible to confine electrons or holes in two dimensions, i.e. in a quantum wire. The quantum wire structure which is simulated in this tutorial is depicted in Figure 4.16.4.1. The quantum wire consists of InAs (blue area) and is confined by GaAs barriers (red area). Its size is 10 nm x 10 nm whereas the whole simulation dimension is 30 nm x 30 nm.

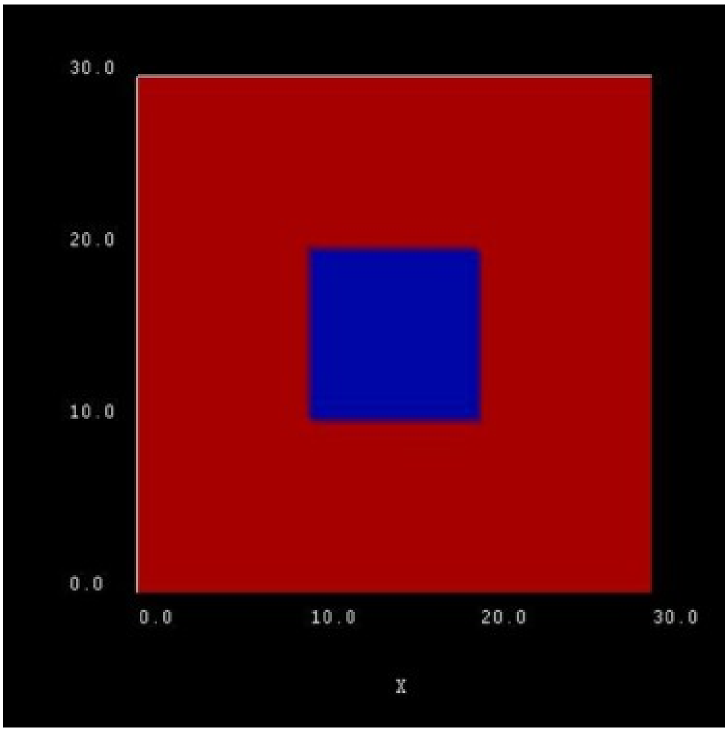


Figure 4.16.4.1: Simulated quantum wire (blue region) consisting of InAs surrounded by GaAs (red).

In our simulations we apply Dirichlet boundary conditions to the quantum region ($\psi = 0$ at the boundary). The quantum region is defined only in the area of the quantum wire, i.e. from 10 nm to 20 nm in both x and y direction. These two conditions lead to an infinite GaAs barrier, which forces the wave functions to zero at the InAs/GaAs

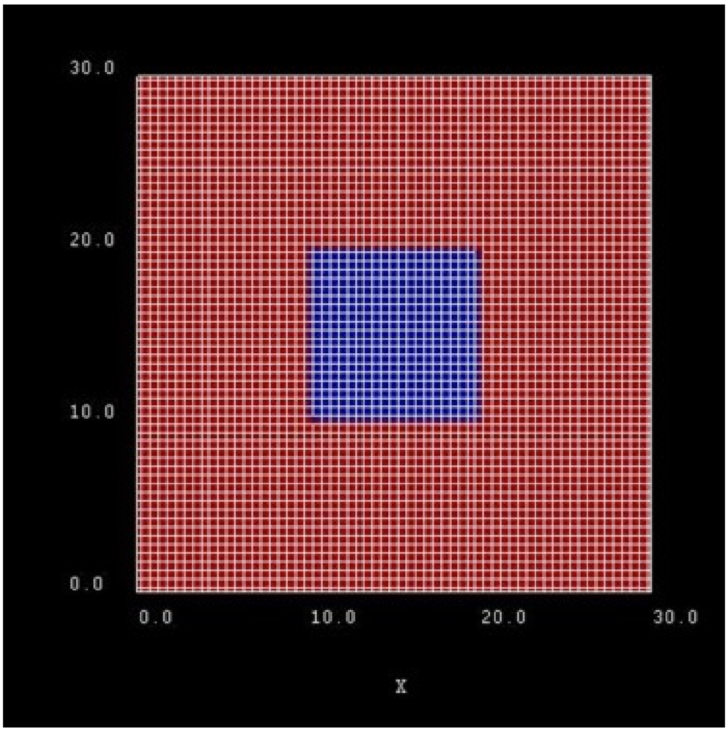


Figure 4.16.4.2: Possible configuration of rectangular grid lines. Here, the grid spacing is 0.5 nm, thus the quantum wire (blue area) consists of 21 x 21 = 400 grid points.

quantum wire boundaries. Of course, this is not a realistic assumption, but we simplify the sample to make the tutorial easier.

The energy levels and the wave functions of a rectangular quantum wire of length 10 nm with infinite barriers can be calculated analytically. This way we can compare our numerical calculations to analytical results. A discussion of the analytical solution of the 2D Schrödinger equation of a particle in a rectangle (i.e. quantum wire) with infinite barriers can be found in e.g. *[MitinKochelapStroscio1999]*.

The potential inside the quantum wire is assumed to be 0 eV. As effective mass we take the isotropic heavy hole effective mass of InAs, i.e. $m^*_{\mathrm{hh}} = 0.41 m_0$. The solution of the Schrödinger equation leads to the following eigenvalues (where m^*_{hh} is assumed to be negative):

$$E_{n_1,n_2} = \frac{\hbar^2\pi^2}{2m^*_{\mathrm{hh}}}\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}\right) = -9.17\,\mathrm{meV}\cdot\left(n_1^2+n_2^2\right),$$

where L_x and L_y (with $L_x = L_y = 10\,\mathrm{nm}$) are the lengths along the x and y direction, respectively. Here, E_{n_1,n_2} is the heavy hole energy in the two transverse directions, or the total heavy hole energy for $k_z = 0$. In the effective mass approximation, the total heavy hole energy is given by

$$E_{\mathrm{hh}} = E_{n_1,n_2} + \frac{\hbar^2 k_z^2}{2m^*_{hh}},$$

where k_z is the wavevector along z leading to a one-dimensional $E(k_z)$ dispersion, and n_1, n_2 are two discrete quantum numbers due to confinement in two directions.

Generally, the energy levels are not degenerate, i.e. all energies are different. However, some energy levels with different quantum numbers coincide, if the lengths along two directions are identical ($E_{n_1,n_2} = E_{n_2,n_1}$) or if their ratios are integers. In our quadratic quantum wire, the two lengths are identical. Consequently, we expect the following degeneracies:

$$\begin{aligned}
E_{11} &= -0.018343\,\mathrm{eV} \quad \text{(groundstate)}\\
E_{12} = E_{21} &= -0.045857\,\mathrm{eV}\\
E_{12} = E_{21} &= -0.045857\,\mathrm{eV}\\
E_{22} &= -0.073372\,\mathrm{eV}\\
E_{13} = E_{31} &= -0.091715\,\mathrm{eV}\\
E_{23} = E_{32} &= -0.119229\,\mathrm{eV}\\
E_{14} = E_{41} &= -0.155915\,\mathrm{eV}\\
&\dots\\
E_{18} = E_{81} = E_{47} = E_{74} &= -0.596145\,\mathrm{eV} \quad \text{(Here, the degeneracy is a coincidence.)}
\end{aligned}$$

The calculated eigenvalues for the 10 nm quadratic quantum wire can be found in the file *bias_00000/Quantum/energy_spectrum_quantum_region_HH_00000.dat*. The numerical results obtained by *nextnano*++ with 0.10 nm grid spacing are:

$$\begin{aligned}
E_{11} &= 0.018341\,\mathrm{eV}\\
E_{12} &= -0.045845\,\mathrm{eV} \quad \text{(two − fold degenerate)}\\
E_{21} &= -0.045845\,\mathrm{eV} \quad \text{(two − fold degenerate)}\\
E_{22} &= -0.073348\,\mathrm{eV}\\
E_{13} &= -0.091653\,\mathrm{eV} \quad \text{(two − fold degenerate)}\\
E_{31} &= -0.091653\,\mathrm{eV} \quad \text{(two − fold degenerate)}\\
E_{23} &= -0.119156\,\mathrm{eV} \quad \text{(two − fold degenerate)}\\
E_{32} &= -0.119156\,\mathrm{eV} \quad \text{(two − fold degenerate)}\\
E_{14} &= -0.155721\,\mathrm{eV} \quad \text{(two − fold degenerate)}\\
E_{41} &= -0.155721\,\mathrm{eV} \quad \text{(two − fold degenerate)}
\end{aligned}$$

The differences between the analytical and numerical results are highlighted in red.

Single-band effective-mass approximation

The corresponding input file is *QWR-magnetic-field_InAs_2D_sg_nnp.in*

Hole wave functions (without magnetic field)

To turn off the magnetic field in the simulation, the variable **$magnetic_field_on** should be set to **0** in the input file.

The following figures show the probability densities ψ^2 of the four lowest energy confined hole eigenstates in an infinitely deep 10 nm x 10 nm InAs quantum wire. Due to the symmetry of the quantum wire, the 2^{nd} and the 3^{rd} eigenstate are degenerate.

Note that these wave functions were obtained by using a single-band effective mass approximation for the holes. A more accurate and more realistic treatment would have been to use 6-band k.p. Note that the wire has been assumed to be unstrained (which is a rather unphysical situation) for the purpose to make this tutorial easier to understand.

Hole wave functions (with magnetic field)

To include the magnetic field in the simulation, the variable `$magnetic_field_on` should be set to 1 in the input file. Here, we assume a field strength of 1 T.

```
$magnetic_field_on        = 1      # choose 1 (magnetic field on) or 0 (magnetic field␣
↪off)
$magnetic_field_strength = 1.0    # Strength of the magnetic field [T]
```

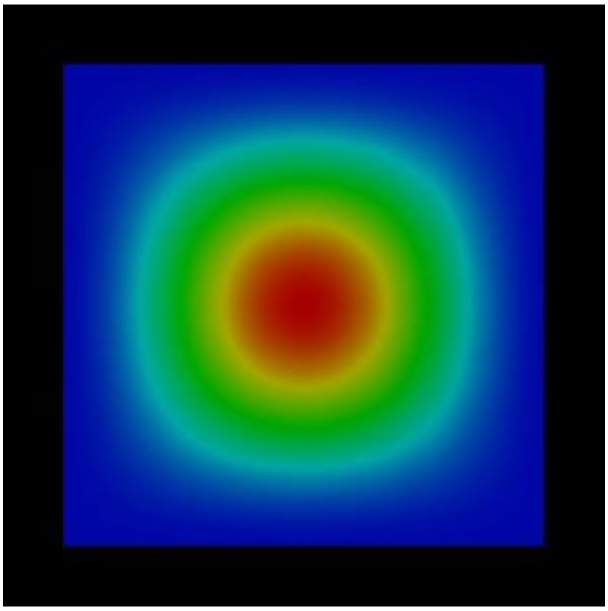


Figure 4.16.4.3: Probability density $\psi_{11}(x,y)^2$ of the 1st heavy hole state.

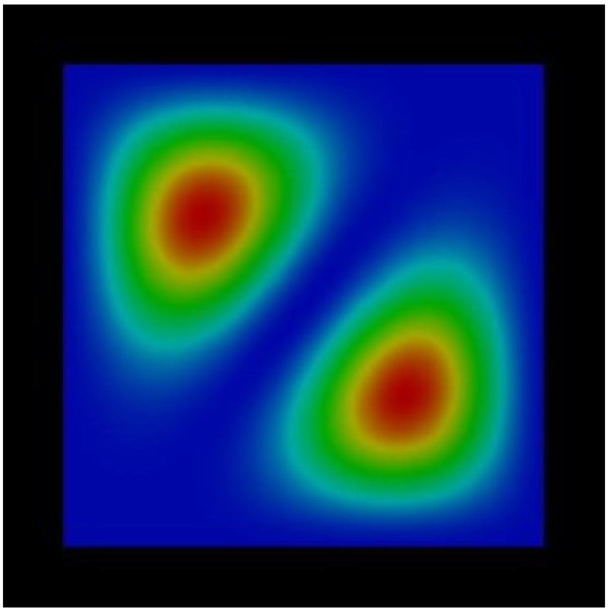


Figure 4.16.4.4: Probability density $\psi_{12}(x,y)^2$ of the 2nd heavy hole state.

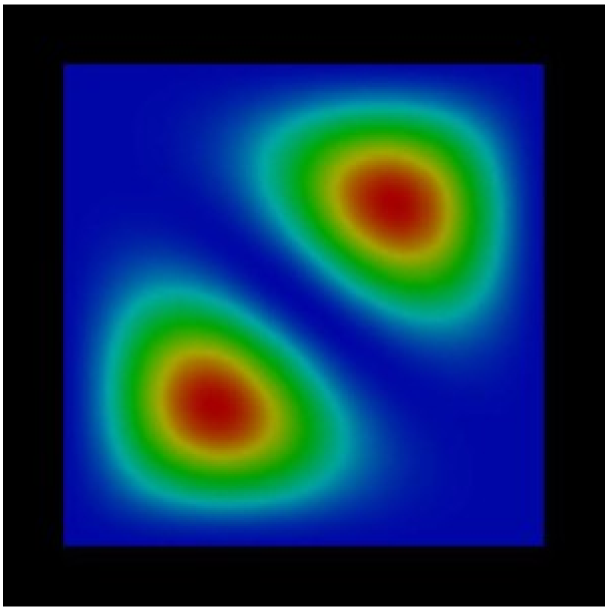


Figure 4.16.4.5: Probability density $\psi_{21}(x, y)^2$ of the 3^{rd} heavy hole state.

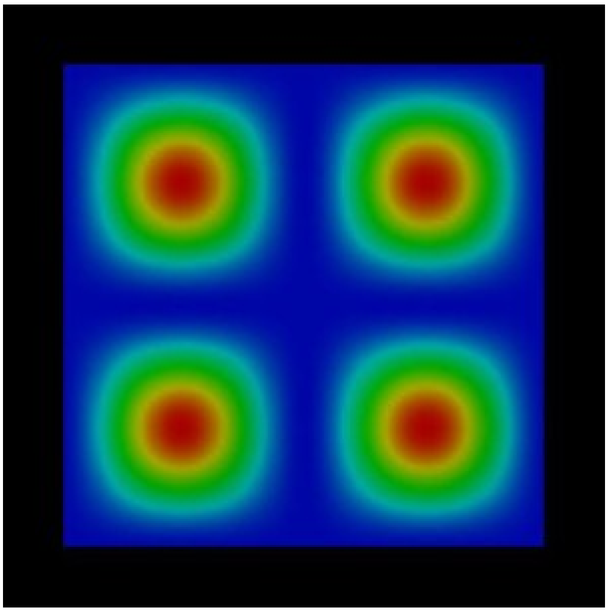


Figure 4.16.4.6: Probability density $\psi_{22}(x, y)^2$ of the 4^{th} heavy hole state.

The g-factor is explicitly set to 0 to avoid Zeeman splitting of the energy levels.

```
database{
    binary_zb {
        name = InAs
        valence_bands{
            HH{ mass = 0.41  g = 0}
        }
    }
}
```

In the following figures the probability densities ψ^2 of the four lowest energy confined hole eigenstates of the infinite InAs quantum wire under applied magnetic field are shown. The magnetic field leads to an additional confinement in addition to the wire potential. However, for the first and forth eigenstate, the confinement does not play an important role, whereas for the second and third it does. The effect is more dominant onto the wave functions but not so pronounced onto the values of the eigenenergies. We observe that the degeneracy of the 2^{nd} and 3^{rd} eigenstate is slightly lifted in comparison to the case where no magnetic field is applied.

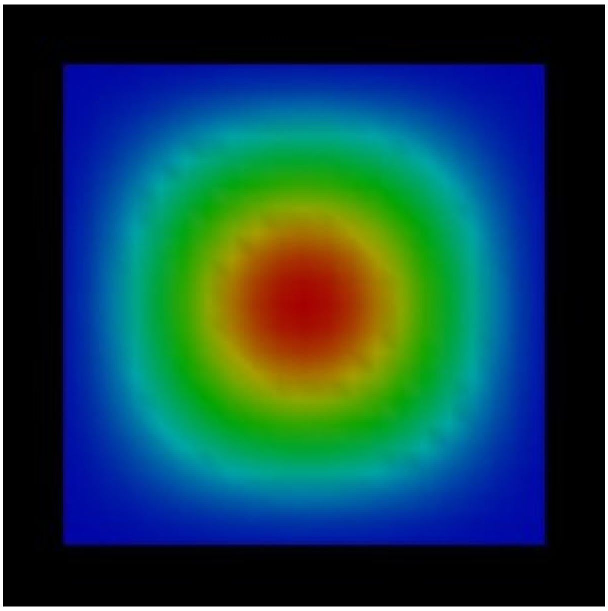


Figure 4.16.4.7: Probability density $\psi_{11}(x,y)^2$ of the 1^{st} heavy hole state with magnetic field applied.

In Figure 4.16.4.11, the probability density of the 2^{nd} eigenstate is plotted from a different perspective.

6-band k.p approximation

The corresponding input file is *QWR-magnetic-field_InAs_2D_6kp_nnp.in*. Here, we used the following Dresselhaus parameters for InAs: $L = -55.0$, $M = -4.0$ and $N = -55.2$.

Hole wave functions - (without magnetic field)

The following figures show the probabilities densities ψ^2 of the four lowest energy confined hole eigenstates in a finite 10 nm x 10 nm InAs quantum wire. This time we used 6-band k.p theory to describe the hole states. Here, the second and the third eigenstate are no longer degenerate.

Last update: nnnn/nn/nn

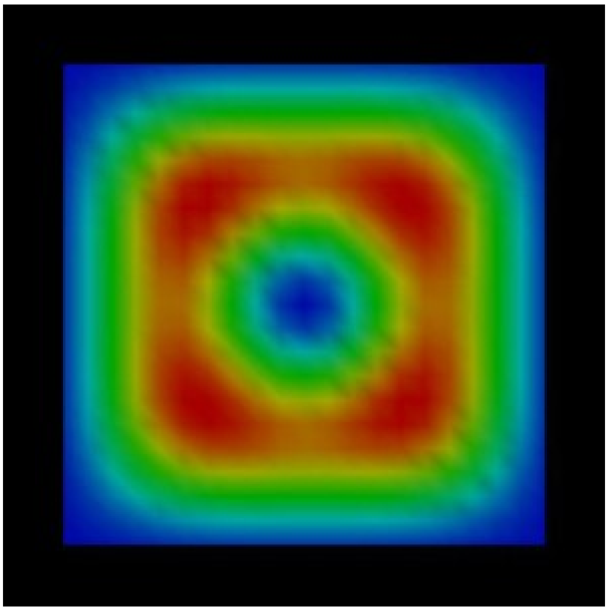


Figure 4.16.4.8: Probability density $\psi_{12}(x,y)^2$ of the 2^{nd} heavy hole state with magnetic field applied.

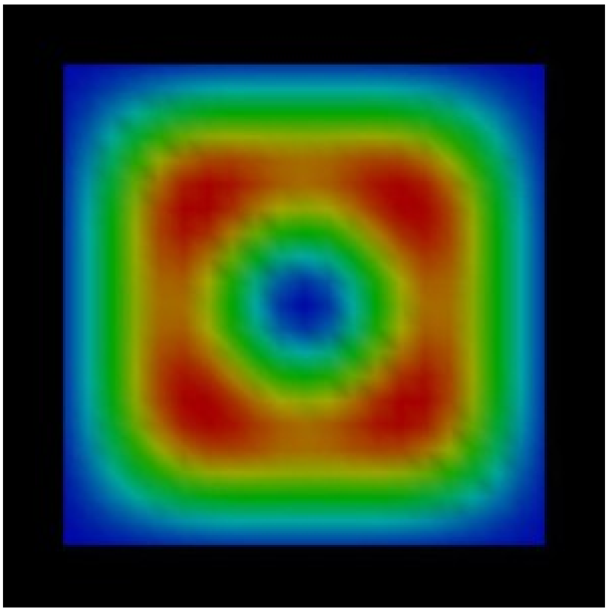


Figure 4.16.4.9: Probability density $\psi_{21}(x,y)^2$ of the 3^{rd} heavy hole state with magnetic field applied.

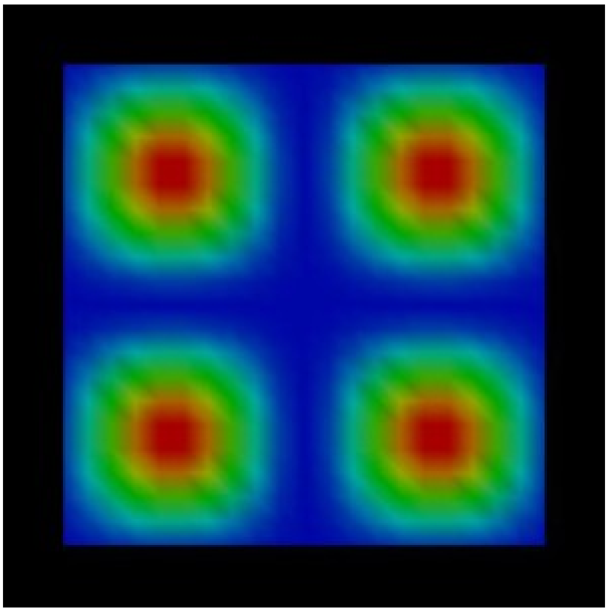


Figure 4.16.4.10: Probability density $\psi_{22}(x, y)^2$ of the 4^{th} heavy hole state with magnetic field applied.

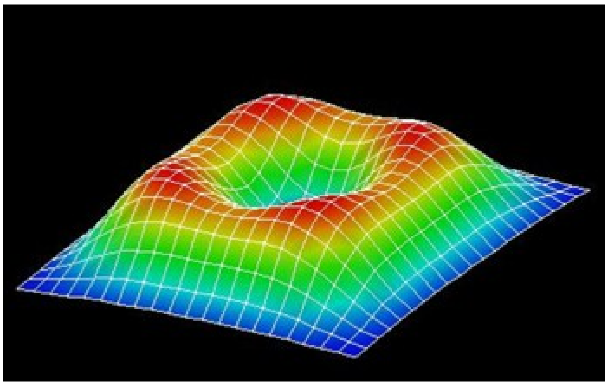


Figure 4.16.4.11: Probability density $\psi_{12}(x, y)^2$ of the 2^{nd} heavy hole state with magnetic field applied (viewed from a different perspective).

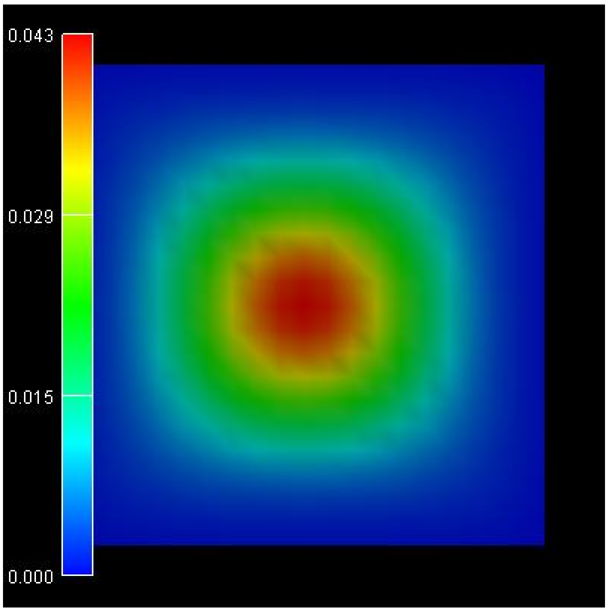


Figure 4.16.4.12: Probability density of the 1^{st}/2^{nd} heavy hole state with energy eigenvalue -0.0171 eV.

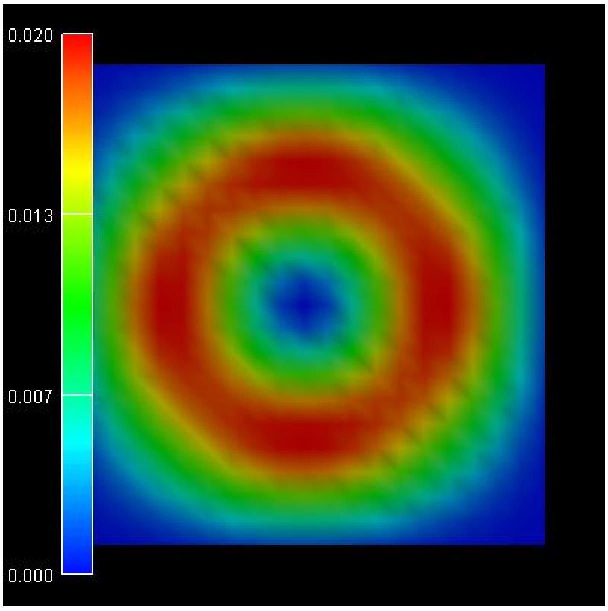


Figure 4.16.4.13: Probability density of the 3^{rd}/4^{th} heavy hole state with energy eigenvalue -0.0282 eV.

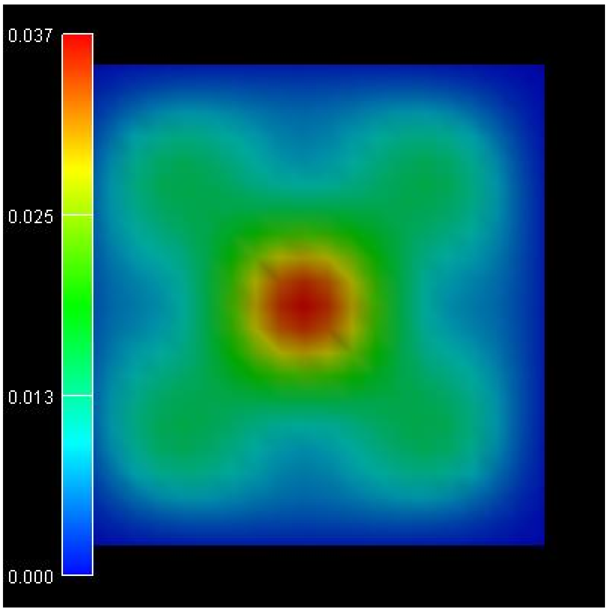


Figure 4.16.4.14: Probability density of the 5^{th}/6^{th} heavy hole state with energy eigenvalue -0.0294 eV.

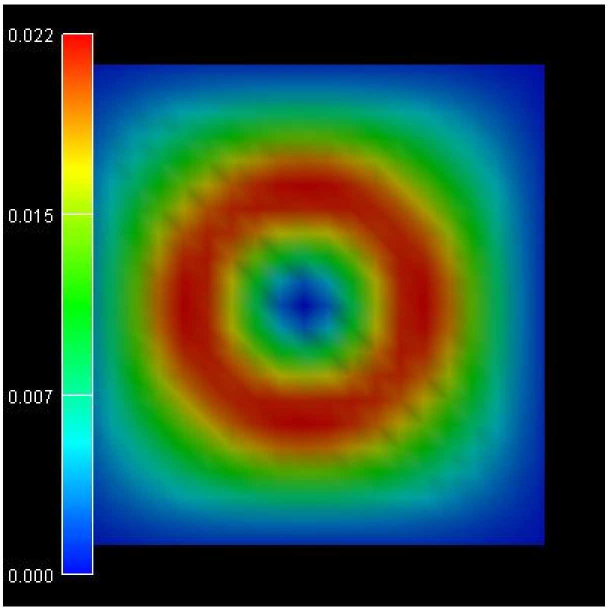


Figure 4.16.4.15: Probability density of the $7^{\mathrm{th}}/8^{\mathrm{th}}$ heavy hole state with energy eigenvalue -0.0367 eV.

4.16.5 — DEV — Vertically coupled quantum wires in a longitudinal magnetic field

Attention

This tutorial is under construction

Input files:

- *Double-QW_AlGaAs-GaAs_1D_nnp.in* - (double square well potential)
- *Parabolic-QW_1D_nnp.in* - (parabolic quantum well)
- *Coupled-QWRs_AlGaAs-GaAs_Mourokh_APL_2007_2D_nnp.in* - (quantum wire)

Scope:

In this tutorial we study the electron energy levels of two coupled quantum wires as a function of a longitudinal (i.e. perpendicular) magnetic field. We will compare our numerical results with analytical calculations published in *[Mourokh2007]*, as well as with experimental data published in *[Fischer2006]*.

Related output files:

- *\bias_00000\Quantum\energy_spectrum_quantum_region_Gamma_00000.dat* - (eigenstate energies)

Structure

The following figure shows the layout of the structure in the (x, z) plane. The blue regions are the barrier materials ($\mathrm{Al}_{0.32}\mathrm{Ga}_{0.68}\mathrm{As}$) and the red regions are 14.5 nm GaAs quantum wells that are stacked along the x direction and separated by a 1 nm thin $\mathrm{Al}_{0.32}\mathrm{Ga}_{0.68}\mathrm{As}$ tunnel barrier.

The confining potential along the y direction is assumed to be parabolic, i.e. of the form $V(y) = Cy^2$. The constant C is chosen such that the separation ΔE_y of the confined eigenstates is $10\,\mathrm{meV}$. From the analytical solution of Schrödinger's equation for a parabolic potential we know that the separation of the eigenstates is given by *[Davies1998]*

$$\Delta E_y = \hbar\omega_0 = \hbar\sqrt{\frac{2C}{m^*}}.$$

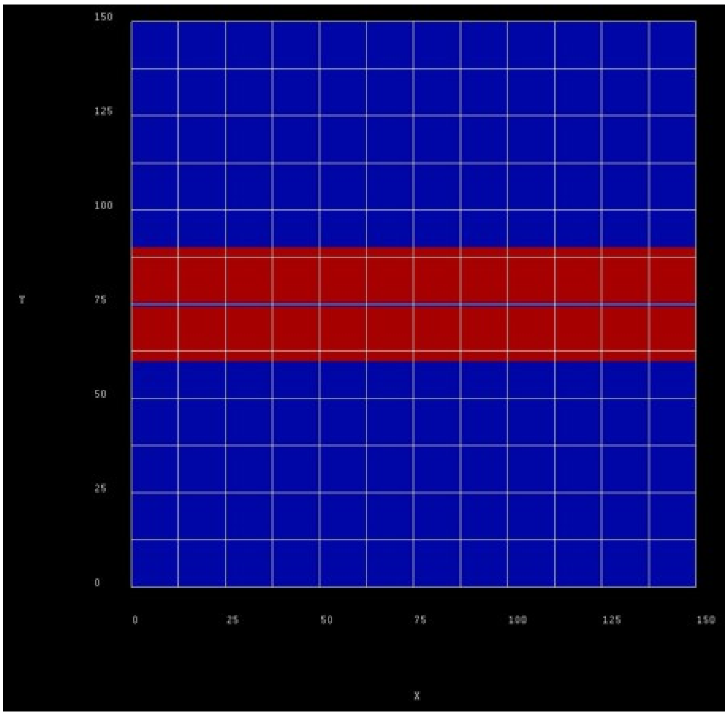


Figure 4.16.5.1: Quantum wire structure.

Therefore, we have

$$C = \frac{m^*}{2} \left(\frac{\Delta E_y}{\hbar} \right)^2 \approx 0.4396\,\mathrm{eV/m^2}.$$

In *nextnano*++ we can create the parabolic potential by using a ternary alloy with artificial material parameters which allows for quadratic interpolation of the conduction band edge energy.

Comparison with analytical results

The following figure shows the confined eigenstates E_z of the coupled, symmetric QW system (1D simulation along the x direction). Note that the states have bonding and antibonding character. The following material parameters were used:

- conduction band offset between GaAs and $\mathrm{Al_{0.32}Ga_{0.68}As}$: $\mathrm{CBO} = 0.27882\,\mathrm{eV}$
- electron effective mass GaAs: $m_\mathrm{e} = 0.067\,m_0$
- electron effective mass $\mathrm{Al_{0.32}Ga_{0.68}As}$: $m_\mathrm{e} = 0.09356\,m_0$

Magnetic field

The magnetic field is oriented along the z direction, i.e. it is perpendicular to the simulation plane which is oriented in the (x,:math:y) plane. We calculate the eigenstates for different magnetic field strengths ($0\,\mathrm{T}$, $0.5\,\mathrm{T}$, $1.0\,\mathrm{T}$, $\ldots$, $16\,\mathrm{T}$).

A useful quantity is the magnetic length (or Landau magnetic length) which is defined as

$$l_\mathrm{B} = \sqrt{\frac{\hbar}{m_\mathrm{e}\omega_\mathrm{c}}} = \sqrt{\frac{\hbar}{|e|B}}$$

It is independent of the mass of the particle and depends only on the magnetic field strength:

- 1 T: $l_\mathrm{B} = 25.6556\,\mathrm{nm}$
- 2 T: $l_\mathrm{B} = 18.1413\,\mathrm{nm}$
- 3 T: $l_\mathrm{B} = 14.8123\,\mathrm{nm}$

- ...
- 20 T: $l_\mathrm{B} = 5.7368\,\mathrm{nm}$

The electron effective mass in GaAs is $m_\mathrm{e} = 0.067\,m_0$. Another useful quantity is the cyclotron frequency:

$$\omega_\mathrm{c} = \frac{|e|B}{m_\mathrm{e}}$$

Thus, for the electrons in GaAs, it holds for the different magnetic field strengths:

- 1 T: $\hbar\omega_\mathrm{c} = 1.7279\,\mathrm{meV}$
- 2 T: $\hbar\omega_\mathrm{c} = 3.4558\,\mathrm{meV}$
- 3 T: $\hbar\omega_\mathrm{c} = 5.1836\,\mathrm{meV}$
- ...
- 20 T: $\hbar\omega_\mathrm{c} = 34.5575\,\mathrm{meV}$

The one-dimensional parabolic confinement (conduction band edge confinement) was chosen so that the electron ground state has the energy of $E_1 = \hbar\omega_0 = 5\,\mathrm{meV}$ in the 1D simulation. In the 2D simulation, the ground state has the energy: $E_1 = 18.64\,\mathrm{meV}$ (without magnetic field) which corresponds approximately to

$$E_1 \approx E_{y,1} + E_{z,1} = 5.03\,\mathrm{eV} + 13.86\,\mathrm{meV} = 18.89\,\mathrm{meV}.$$

(In 2D, we use a different grid resolution compared to 1D simulations.)

Comparison with experimental results

More realistic situation,

We introduce doping in the structure. Form of two delta peaks We apply a gate contact at the top of the device (which is intended to control the energy states of the electrons)

We solve the self-consistent Schrödinger Poisson equation self-consistently.

(In comparison to the analytical results/ calculation where we do not solve Poisson equation and therefore the effect of space charges is not included). Including the effect of space charges and the applied bias, leads to the vanishing alignment of the energy states. Non-zero anti-crossing between the tunneling states.

Last update: 17/07/2024

4.17 Numerics

4.17.1 General

This set of tutorials focus on explaining numerical side of simulations with *nextnano*++ from the practical point of view.

Convergence

Introduction

Simulations of Schrödinger-Poisson converge self-consistently, and almost automatically, thanks to an algorithm proposed some years ago by Alex Trellakis, one of our talented developers. This algorithm was implemented in *nextnano*++ and has been very successful for several devices. However, when the current equation is included in this system, the convergence to the solution becomes a challenge due to the nature of this equation. For some devices, the system of equations becomes very unstable and a certain ability to reach the convergence is required. Especially for systems where the carrier density fluctuates from large values to almost zero in certain regions or interfaces, the process of obtaining convergence becomes more critical and acting in a strategic way is very helpful.

Setting the input file for performing self-consistent current-Schrödinger-Poisson computations

Self-consistent current-Schrödinger-Poisson computations can be specified in the section run{ } of the input file, through the statements

- `current_poisson{ }`
- `quantum_current_poisson{ }`

The first statement is mandatory, and it provides a first estimate of the electrostatic potential and the (quasi-)Fermi levels, even before including the quantum calculations to the system. In principle, this is the minimum information required to start the simulations. All numerical parameters are adjusted automatically internally in the code until the solution is found or the maximum number of iterations is reached. Unfortunately, given the huge variety of devices the program can simulate, universal parameters are not possible to be predicted in advance. For this reason, in order to give to the user more control of the convergence process, some parameters can optionally be specified within the subsection `quantum_current_poisson{ }`. Some examples are the following parameters: `alpha_Fermi`, `residual`, `residual_fermi` and `iterations`.

It is not our purpose to describe each of these parameters in this document, but to provide some guidance how to control the numerical process with the minimum effort as possible. The list of all parameters, its description, range of values and default can be found on the section *run{ }*.

Talking about convergence

Before proceeding, it is important to discuss what the expression "to get convergence" means. Actually, *nextnano++* has to solve three groups of equations for electrons and for holes: current (also called, continuity) equation, Schrödinger equation and Poisson equation. As default the values of carrier densities, Fermi levels and potential are kept iteratively consistent from one step to the other. Internally the program computes for each equation a so-called cost function, that represents a metric of how close the obtained solutions are close to the "exact" one. For example, one way the cost function can be defined is by the difference of left and the right of each equation. Then, after each iteration the results of the cost function are called residuals.

Getting convergence means to find the conditions that minimize the cost functions. A good analogy of this process is the task of finding the deepest location of a valley in a mountain chain. In order to reach this valley, having some strategy concerning the necessary moves in some direction can reduce the time and the number of steps to conclude this task. If each step is too large, we can overfly the valley, if it is too small, we can take a long time to reach it. This is the role of the alpha_Fermi parameter in the `current_poisson` and `quantum_current_poisson` solvers: large values of alpha_Fermi can make the minimum invisible, and if it is too small can take a long time for simulations. Additionally, especially when the value of alpha_Fermi is small, it is possible that the number of iterations, given by the parameter iterations, is not enough to reach this minimum.

This analogy with a mountain chain is actually very simplistic, because the program deals with finding a minimum of cost functions in a multi-dimensional space and a non-linear system of equations, which makes this task more complex and, for this reason, provides more accurate results than any analytical model.

Keeping this in mind, setting the right parameters is usually an iterative process. One procedure that can be used for reducing the simulation time is by displaying the results "on-the-fly" within our graphical interface (*nextnanomat*) in two different ways.

The first method is to check the numerical values displayed in the "Simulation" tab of the graphical interface (*nextnanomat*). The evolution of the residuals is printed out as soon as they are computed. If some of the residuals are not reducing from one iteration to the next one after certain time, it is recommended to stop the simulation and restart a new one with different parameters.

The second method involves plotting the files `interation_current_poisson.dat` and `iteration_quantum_current_poisson.dat`. By default, these files are generated automatically by the program, unless "output_log = no" is specified in `current_poisson{ }` and `quantum_current_poisson{ }` subsections. They can be displayed in the browser menu for the "Output" tab of *nextnanomat*. As in the previous method, if the residuals are reducing too slow, it is recommended to restart a new simulation that can accelerate the process.

Recommended strategy

As mentioned before, for some devices, the value of the parameters appearing in `current_poisson{ }` and `quantum_current_poisson{ }` subsections that bring the algorithm into a quick convergence belong to a very small region of the parameter-space, and tuning these parameters can require certain ability and time. The program contains internally several default parameters that are suitable for many devices, but due to the huge variety of configurations that a device can present, it is possible that, for some devices they have to be adjusted manually.

We recommend beginning with the following steps which can assist you to control the simulation: they are not universal, but they can provide some ideas about the procedure.

1 - Simplify the system

Start finding a suitable electrostatic potential. In other words, comment out all the lines except `strain{ }` and `current_poisson{ }` subsections in the `run{ }` section of the input file.

2 - Set `minimum_density_*` or `maximum_density_*`

Set the `minimum_density_*` to a large value, i.e. 1e12 or even larger, if necessary. This parameter can be found within the `current{}` section of the input file. Nevertheless, for some conditions, where the density of carriers is expected to be low, the values for `minimum_density_*` and `maximum_density_*` should be reduced, for example to 1e-2 and 1e16. In this situation, the most critical value is the `maximum_density_*`. One typical example where the `maximum_density_*` should be reduced are simulations for which the current in expected be almost zero, like in a diode or transistor operating under the threshold bias.

3 - Adjust parameters of `current_poisson` simulation

A complete control of the simulation can be obtained by choosing new target residuals (`residual` and `residual_fermi`) and the number of iterations (`iterations`). The smaller the residuals, the larger the runtime will be. Choose a certain number of iterations, and after the simulation verify, by reading the log file, if it is necessary to increase this number.

In the latest versions of *nextnano*++, a new method was developed that can reduce the simulation time. Please set `fast_poisson = yes` inside `current_poisson{ }` in order to activate the new method.

After each simulation it is recommended to gradually reduce the value of the `minimum_density_*`, for example, by a factor of ten until the system again does not converge. At this point, change the values of `alpha_fermi` and `current_iterations` until the code converge again. In the next section an intuitive approach of how these parameters can be smartly changed is presented.

For fast simulations, choose the value of `alpha_Fermi` as large as possible (the maximum value is 1.0). If this value generates overflow, a message will appear in the graphical interface and the simulation will stop. In this case, it is recommended to reduce the value of `alpha_Fermi` for example to: 0.5, 0.1, 0.05, and so on. Simulation rarely converges for values close to 1 (the default value).

If the simulation is still not converging, increase the `minimum_density_*`, and simulate again using `alpha_Fermi` equal to 0.5 or less. There is no recipe valid for all devices.

At the end of this process, for certain values of the residuals, the value of the `minimum_density_*` should be as small as possible. Taking some time to find a larger value of `alpha_Fermi` bringing the system to convergence will speed up the rest of the simulations.

4 - Self-consistent quantum calculations

Follow the same procedure as before, but do not change the parameters of the `current_poisson{ }` subsection. Usually, it is a good strategy to start with larger residuals within the `quantum_current_poisson{ }` subsection than the one used in `current_poisson{ }`.

Having obtained some initial results, even before reaching convergence, it is always helpful to check if the occupation number of all bands decays to zero, or at least, several orders of magnitude from the initial values. If necessary, increase the number of events in the specific band where the occupation number is not small enough. Keep in mind that the self-consistent solution contains all information about the states that can be populated for the system under certain conditions (for example, a certain applied bias).

5 - Alternative solution

The self-consistent simulation of the three groups of equations can result in a numerically unstable solution for some systems under certain conditions. For this reason, the option of limiting how far the quasi-Fermi levels can move above the highest contact of below the lowest one has been implemented. This is still a new feature under development and it is only recommended in the case of devices presenting materials with huge band gaps and extreme photogeneration. Set `fermi_limit` to a value in the range 0 and 10 eV for this kind of simulations. The default value is 2 eV.

Getting some intuition…

As mentioned above, depending on the nature of the device and the specific operation conditions (temperature or bias), it is necessary to guide the tool to get convergence. Let us see some practical examples.

Here we will illustrate how the evolution of the residuals in a current-Schrödinger-Poisson can evolve during the convergence process for two different devices. The images correspond to the plot of the data from `interation_current_poisson.dat`, and `iteration_quantum_current_poisson.dat` files, that can be found in the output folder of the simulation.

Figure 4.17.1.1 corresponds to the residual evolution of a system that converges faster: all residuals drop around one order of magnitude every ten iterations. The default parameters within the code brings the system almost automatically to the minimum of the residuals.

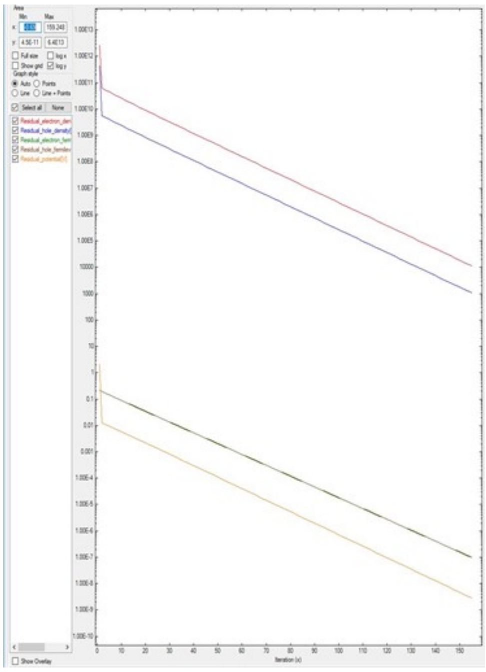


Figure 4.17.1.1: Residual evolution for a `system A` exhibiting quick convergence.

In contrast, Figure 4.17.1.2 shows the final result for a different device after the system gets convergence. In this case, in the input file were especified that `residual_fermi` is equal to 10^{-7} eV and residual (density) as $10^5/cm^3$. The value of `alpha_Fermi` is 0.01. Although it was specified a total of 2000 iterations, the convergence was achieved in around 400 steps. It is important to notice that only after 180 iterations the system starts reducing the residuals in several orders of magnitude.

For some devices, setting the values of `alpha_iterations` and `alpha_scale` can result in a better performance. The value of `alpha_iterations` is related to the moment where the `alpha_Fermi` shall start to gradually reduce, and the value `alpha_scale` is the rate of reduction between two successive iterations. There is no rule for the direction they should be changed. It is necessary to test some cases and look at the effect on the residuals.

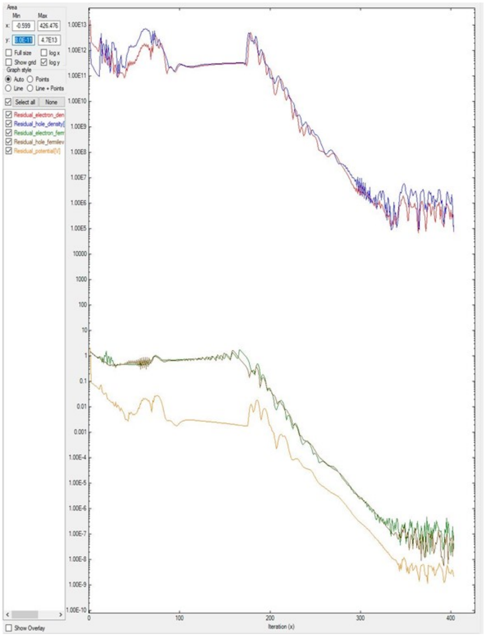


Figure 4.17.1.2: Residual evolution for a `system B` with slow convergence. In the input file were specified `residual_fermi = 10^-7 eV`, `residual (density) = 10^5 /cm^3`, and `iterations = 2000`. The `alpha_Fermi` parameter was set to 0.01.

Sometimes the number of iterations is not enough to reach the convergence. Figure 4.17.1.3 and Figure 4.17.1.2 plot the results of the same `system B` but differ in their number of iterations. Figure 4.17.1.3 is simulated with only 150 iterations. As it was shown in Figure 4.17.1.2, only after 180 iterations the residuals start to decrease. Hence Figure 4.17.1.3 does not show converging behavior. In this kind of simulations, there are no criteria for knowing at which point this will happen: it requires experience or can be done by trial and error.

A pseudo-non-convergence can also happen when small residuals are specified in the input file. Returning to the Figure 4.17.1.2 it can be observed that, choosing `residual_fermi` as 10-^10 eV would probably result in a non-convergence: the `residual_fermi` does not decrease at a high rate after 350 iterations. Then, increasing the number of iterations in this case would not solve the problem.

Another situation is when the value of `alpha_Fermi` is too small: it looks like the residuals do not decrease, like in Figure 4.17.1.4. In this example, `alpha_Fermi` was reduced from 0.01 (value used for Figure 4.17.1.2 and Figure 4.17.1.3) to 0.0001, and after 2000 iterations the system does not converge. Here we used the `system B` of the previous two images.

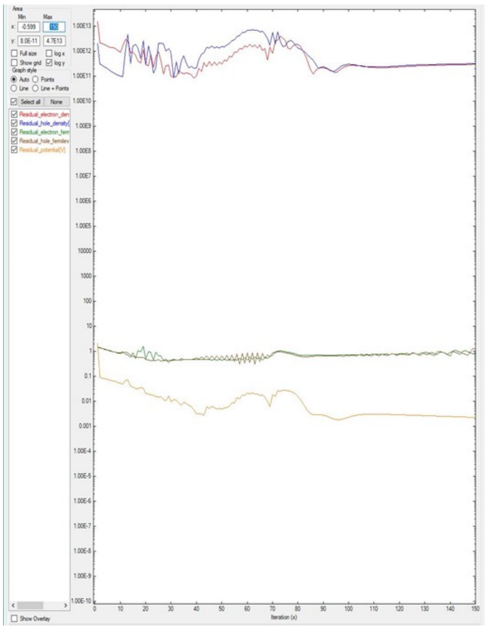
There are other patterns for finding convergence, but here only the most relevant ones have been shown.

Sweeping parameters

It is very common to use a sweep of specific variables within the input file, for example bias or any other user defined parameter.

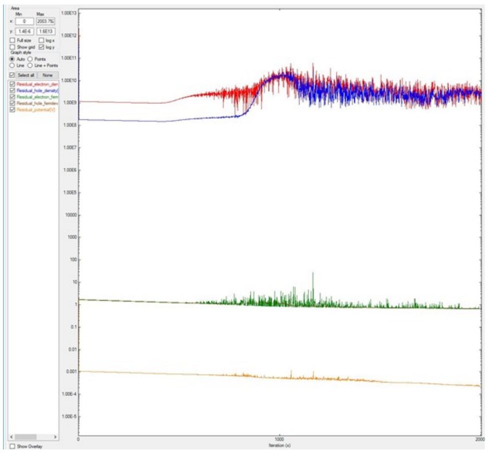
It is important to have in mind that any change in the input file is equivalent to a simulation of a new system (for example when modifying doping), or the operation condition (temperature or bias). There is no mathematical reason that the solutions of two systems should be similar. In other words, it is not expected that all solutions using different conditions will converge under the same criteria, for the entire range of variation of the sweep parameters. Eventually, for example, a sweep of bias from 0 to 8 Volts can use the same parameters for the whole simulation, but this is not the most common case.

A good strategy is to start the sweep of the parameters and verify at which value the solution does not longer converge. For saving time it is recommended to split the range of variation in two parts, and to follow the simulation only using the values of the parameter (for example, bias) that have still not converged. Trying to make the solution converge for a wide range of values for the sweep variable, using with a unique set of residuals and `alpha_fermi`, can become a very hard task, without the recommended range splitting.



Area
Min Max
x: -0.599 150
y: 8.0E-11 4.7E13
☐ Full size ☐ log x
☐ Show grid ☑ log y
Graph style
● Auto ○ Points
○ Line ○ Line + Points
☑ Select all None
☑ Residual_electron_den
☑ Residual_hole_density(
☑ Residual_electron_ferm
☑ Residual_hole_fermilev
☑ Residual_potential(V)
☐ Show Overlay

Figure 4.17.1.3: Residual evolution of `system B` with 150 iterations, exhibiting a pseudo-non-convergence behavior. Specifications in the input file: `residual_fermi = 10^-7 eV`, `residual` (density) of 10^5 /cm^3, and `iterations = 150`. The value of `alpha_Fermi` is 0.01.



Area
Min Max
x: 0 2003.79
y: 1.4E-6 1.6E13
☐ Full size ☐ log x
☐ Show grid ☑ log y
Graph style
● Auto ○ Points
○ Line ○ Line + Points
☑ Select all None
☑ Residual_electron_den
☑ Residual_hole_density(
☑ Residual_electron_ferm
☑ Residual_hole_fermilev
☑ Residual_potential(V)
☐ Show Overlay

Figure 4.17.1.4: Residual evolution for a system exhibiting pseudo-non-convergence. Specifications in the input file: `residual_fermi =10^-7 eV`, `residual (density) = 10^5 /cm^3`, and `iterations = 2000`. The `alpha_Fermi` parameter was reduced to 0.0001.

. . . and when nothing works

Our concern, in the development of our code, is to make it as accurate and fast as possible. Some simulations can be performed in a simple notebook, especially for 1D simulations.

Unfortunately, for some devices under specific conditions, making the system of Current-Schrödinger-Poisson converge in few iterations is a very specialized and time-consuming task. Observing the needs of our customers, nextnano is offering our customers the opportunity to perform this task on demand. Please consult our **schedules** and **fees** when an extra assistance is required. Our experts in simulation can assist you to boost your project!

Residuals

The residuals specified in the input file are numbers defining accuracy of the simulation; we refer to them further as the **desired residuals**. The convergence process is terminated when all the residuals reach the values of the desired residuals or lower. Reaching lower values of the residuals provides more accurate solution, however, at expense of longer runtime.

The evolution of residuals is stored in real-time of the simulation run in files *iteration_current_poisson.dat* or *iteration_quantum_current_poisson.dat*. They can be directly monitored in *nextnanomat* during the simulation.

Our suggestion is to begin simulations using the default values, defined according to the dimensionality of the simulation domain. To obtain a compromise between the accuracy of the solutions and the simulation time, one can adjust the desired residuals.

Quasi-Fermi Levels

The residuals of each quasi-Fermi level are computed as a maximum norm of the difference of values obtained in two consecutive iterations at every grid point. Therefore, this value directly corresponds to the highest local change of the quasi-Fermi levels in the simulated structure after each iteration of our algorithm.

The desired residual for the quasi-Fermi levels in the input file can be specified by assigning a value to the variable `residual_fermi` within the group `run{ }`.

Once having the simulation done, the accuracy of the solution can be estimated by investigating the file *bandedges.dat* where both quasi-Fermi levels are outputted. Changes of the levels and related changes of the carrier concentrations, especially in the region of interest of the modeled structure, can be used to decide whether the simulation reached the desired accuracy or it should be refined.

Carrier Densities

The residuals of each carrier densities are computed as a 1-norm of the difference of values of the carrier densities (multiplied by volumes assigned to each grid point) obtained in consecutive iterations at every grid point. Therefore, this value corresponds to a cumulative change of entire carrier distributions between two consecutive iterations. In other words, it is an integrated absolute value of a difference of carrier distributions computed in two consecutive iterations.

The desired value for the residuals of the densities can be specified in the input file using a variable `residual` within the section `run{ }`.

In order to evaluate the accuracy of the solutions for your needs, it is convenient to verify the final densities and charges in the output files:

- *density_electron.dat*

- *density_holes.dat*
- *total_charges.txt*

Changes of orders of magnitude of integrated carriers in various important regions of the simulation should be taken into account based on these files to decide if more accurate solutions are required by reducing the respective residuals.

Electric Potential

The residual for potential is computed in the exact same way as for the quasi-Fermi levels. It is computed as a maximum norm of the differences of values obtained in consecutive iterations at every grid point. Therefore, this value directly corresponds to the highest change of the electric potential after each iteration of our algorithm.

The desired value for the residual of the electric potential is only available internally in the code and is well controlled by the algorithm.

It is recommended to see the file *potential.dat* to estimate the accuracy of the computed electrostatic potential.

Self-Consistent Simulations

The guidelines described above should be treated as a basic example aiming at developing intuition and understanding of how the simulation behaves from the numerical point of view. As all the residuals are interdependent in a specific way, it is important to have at least basic understanding of the dependencies between currents, carrier concentrations, electrostatic potential, and quasi-Fermi levels for the structure design of the interest for every simulation run, especially within a self-consistent algorithm.

4.17.2 Big 3D systems

These tutorials cover topic of practical approach to simulations of big 3D systems aiming at specified accuracy within possibly short time.

Approaching large 3D designs with Schrödinger-Poisson self-consistent solver

Large memory consumption and long runtimes are usually the challenge when performing 3D-simulations of large devices with high accuracy.

Based in our experience simulating large number of devices, we created a methodology that will assist you to set up the input files in a very efficient way. Figure 4.17.2.1 summarizes the three phases in the development of these files:

- reduction of the dimensionality
- optimization of the grid for electrostatics problems
- setting up the input file for the quantum computations

The main idea in all steps is to define the necessary grid in the shorter time as possible. We will focus on the use coarse grids for identifying regions that are more relevant from our simulations.

Reducing the dimensionality of the problem by creating 1D- and 2D- versions of the system are generally very useful to identify which regions do not require a fine grid. Additionally, by convenient application of boundary conditions, some regions can be completely eliminated from the simulation domain. A typical example is the substitution of substrate by an adequate boundary condition, that in *nextnano*++ we denominate `contact`.

It is important to optimize the grid always step by step: first one dimension, and then, the next.

Even for self-consistent solution of the Schrödinger-Poisson equations we always suggest to set up the input file solving only the Poisson equation, even when not accurate enough. These solutions can be very useful for identifying unnecessary regions to be eliminated from the simulation domain, and to refine the grid only where is actually necessary.

Our focus will be the evolution of the residuals at the beginning of the convergence process. Then, as we mentioned above, it is not expected to obtain accurate results, but only the trends of these residuals.

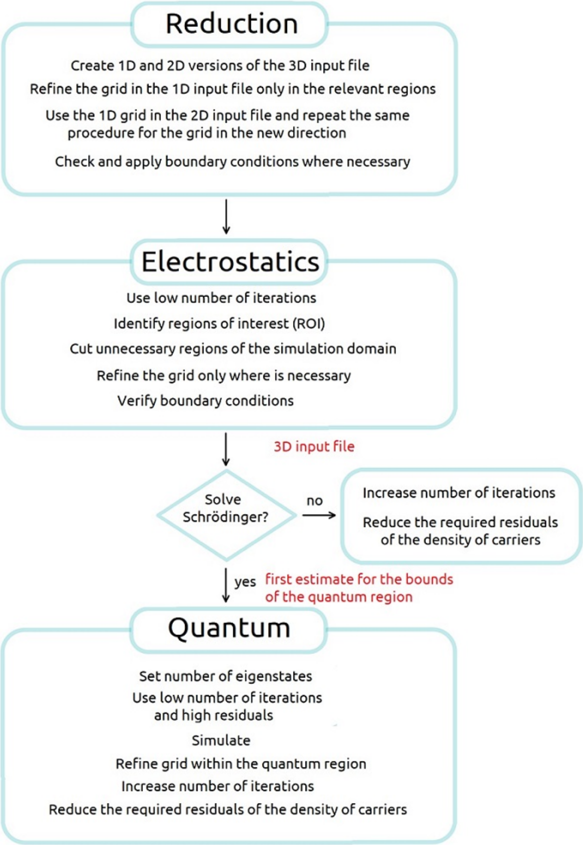


Figure 4.17.2.1: Methodology for 3D-simulation of large devices.

If no quantum computations are required this would be the point to reduce the residuals in the convergence process for obtaining the results with the accuracy desired.

Similarly, as done in the two previous steps, the definition of the quantum region can be the secret to the final tuning of the 3D-input file. Starting with the results of the electrostatic problem we can identify the regions of interest for such simulations where the grid has to be refined. The identification of a suitable number of eigenvalues for the self-consistent simulations is a crucial procedure that must be performed. It is also important to be aware of the boundary conditions that are adequate at the bounds of the quantum region.

We can take advantage of the one symmetry that the device can present for making a first exploration of these issues. This will save you memory and time.

Each of these procedures are explained in details and with a practical example in three independent tutorials:

Reducing dimensionality of large 3D designs

Optimizing electrostatics simulation for large 3D designs

Optimizing Schrödinger-Poisson self-consistent solver for electrostatic quantum dots

where the other guidelines concerning how to simulate large devices in three dimensions efficiently.

Last update: 15/07/2024

Reducing dimensionality of large 3D designs

Header

Files for the tutorial located in *nextnano++\examples\numerics*

- *large-3D-systems-reduction_1D_nnp.in*
- *large-3D-systems-reduction_2D_nnp.in*
- *large-3D-systems-reduction_3D_nnp.in*

Scope of the tutorial:

- Guidelines for reducing dimensionality of 3D-input files
- Refining the grid line spacing efficiently
- Impact of the grid resolution and the number of nodes in the grid on the simulation time

Introduced Keywords:

- `global{ simulate1D }`
- `global{ simulate2D }`
- `global{ simulate3D }`
- `grid{ xgrid{ } ygrid{ } zgrid{ } }`
- `quantum{ region{boundary_conditions{}} }`
- `strain{ growth direction }}`
- `structure{ line{} }`
- `structure{ rectangle{} }`
- `structure{ cuboid{} }`

Relevant output Files:

- *\bias_00000\bandedges.dat*
- *\bias_00000\bandedges_1d_xz_Si_2DEG.dat*
- *large-3D-systems-reduction_2D_nnp.log*

Accurate simulations depend on finding a compromise between a very fine grid, the memory consumption and the corresponding runtime. Nevertheless tuning the grid resolution for 3D simulations of large devices can become highly time expensive, when a methodological approach is missing.

The purpose of this tutorial is to provide some suggestions with the aim of reducing the time for choosing a suitable grid and of its impact on the solutions. It is part of the methodology *Approaching large 3D designs with Schrödinger-Poisson self-consistent solver*, that we strongly recommend being followed.

In this first step we will show what we can learn from simulations in 1D and 2D of the device, for building a suitable grid when modeling the most important regions on it.

To make it very practical, we will introduce in the next section a structure that can be used in a semiconductor-based quantum computer as an example. The quantum operations are performed by handling the bias of gates on the top of the device, that controls the transport of the carriers through the active region. This is a typical device where all transport of carriers is electrostatically dominated. For this reason, a consistent simulation of the charge distribution and the potential in the device is imperative to reach accuracy enough to identify the most important modes of operation at each position.

Most of these devices can present hundreds of nanometers than represent a heavily time-consuming procedure when performing 3D simulations. The suggestions presented below will assist you to define the grid that can reduce the bottlenecks of larger simulations. There is not a unique way to do it, but it has been used for numerous cases, not only for quantum computing, and provided very good results in most of them.

Device to be simulated

Figure 4.17.2.2 presents a simplified version of a device that consists basically of a 7 nm-Si layer buried in a silicon dioxide structure *[Kriekouki2022]*. This silicon layer will be used as the channel where electrons can transit through.

Gates (FGS, FGD, LG1, LG2, LG3) are deposited at few nanometers of top of the interface of the Si channel with the surrounding oxide gates. By applying specific combinations of biases to these gates it is possible to change the electrostatic potential and, in this way, to control the states present in the structure for each configuration. The source and drain contacts can be seen as the reservoirs that will provide the carriers that will propagate in the channel.

Additionally, applying bias to a back gate under the thick layer of oxide under the Si-channel can allow or prevent the transport through the device.

The dimensions of this device to be simulated is the order of 400 nm x 800 nm x 70 nm. The last dimension (70 nm) does not include the back gate and substrate regions that, as we will see soon, can be removed from the simulation domain. Nevertheless, the relevant results in the active regions are very localized and can require grid resolutions of order of few nanometers or smaller.

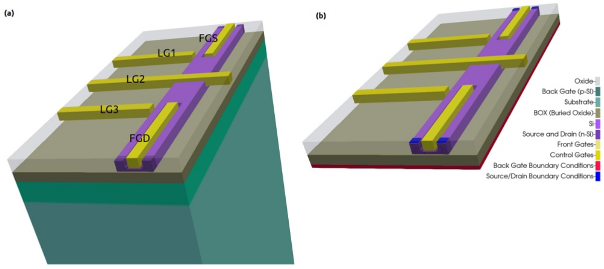


Figure 4.17.2.2: Device to be simulated. The Si-channel is buried in the oxide. FGS, FGD, LG1, LG2, and LG3 are used to shape the electrostatic potential.The back gate is used to allow or to interrupt the transport of electrons through the channel. The source and drain are the reservoirs of carriers.

Reducing the dimensionality of the problem

Before setting up input files for 3D simulations we recommend to start with 1D or 2D computations. Even when quantum computations are necessary, use only semiclassical models (Poisson), just enough to identify the most relevant aspects of the transport in some critical regions.

You can either start designing the 3D version and reduce it to the 1D and 2D versions, or to develop first the 1D version and expand it to the final 3D structure.

For making the design more flexible, use variables to represent the most important coordinates of the structure. Name the variables according its 3D representation in the device reference frame, in contrast to the simulation reference frame. The simulation system is defined in the `global{ }` section of the input file. Figure 4.17.2.3 presents the most important coordinates in the device coordinate system, used in all versions of the input files of our example.

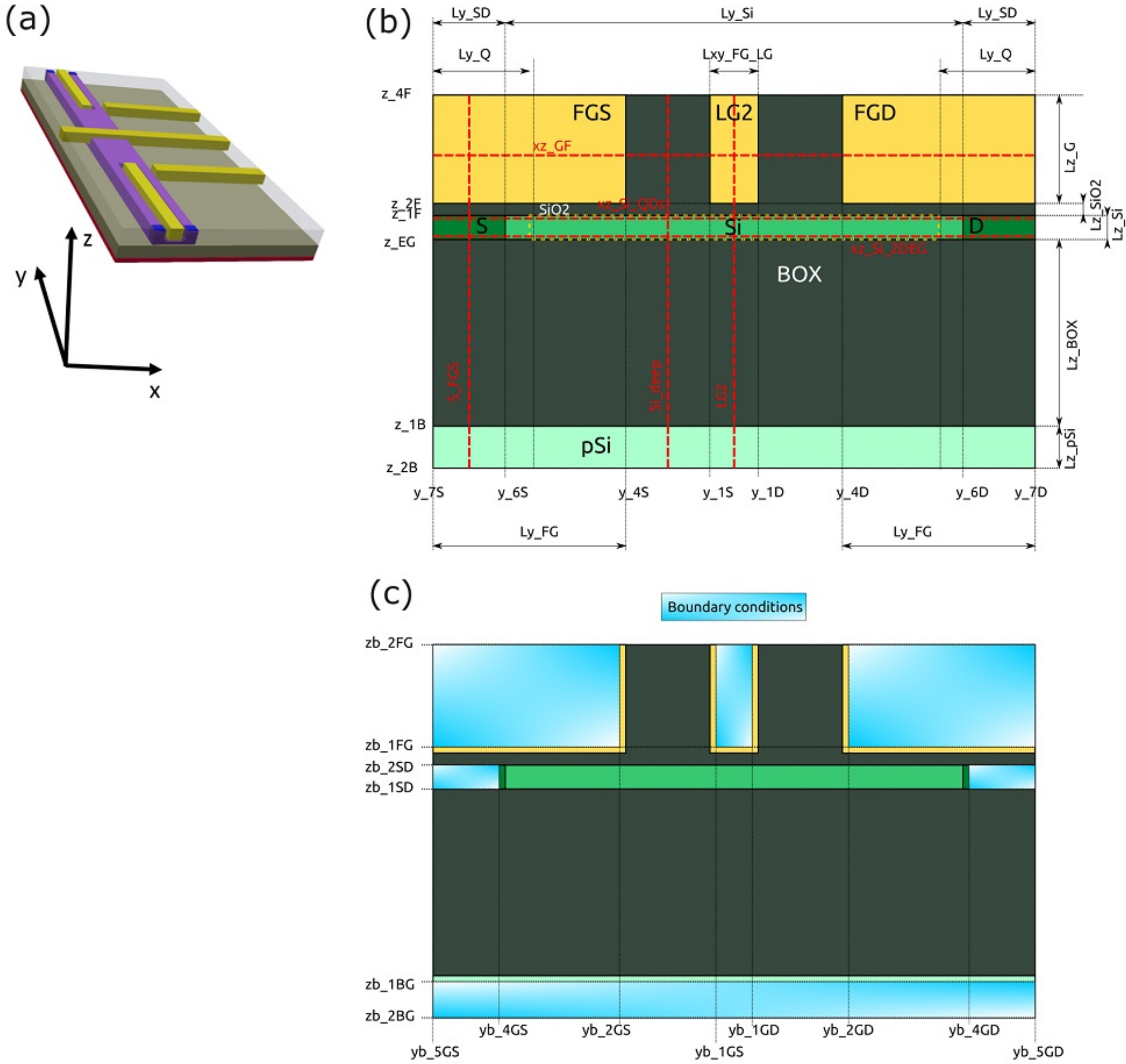


Figure 4.17.2.3: Device reference system and most important coordinates used for 1D and 2D simulations: (a) the 3D representation, (b) structure definition, and (c) structure after applying boundary conditions to the contacts and gates. Dotted lines (in red) represent sections defined in the input files.

Here is an example how to perform the modification from 3D to 2D input file. Suppose that one region is defined in the 3D input file by:

```
cuboid{
     x = [$x_3F, $x_3L]
     y = [$y_4GS, $y_4GD]
     z = [$z_EG, $z_2F]   # growth direction in the simulation reference system for␣
↪3D simulations
}
```

where the growth direction is along the z-axis (vertical) in the device coordinate system.

This has to be translated to a 2D-input file as:

```
rectangle{
     x = [$y_4GS, $y_4GD]
     y = [$z_EG, $z_2F]   # growth direction in the simulation reference system for␣
↪2D simulations
}
```

and to a 1D-input file as:

```
line{
     x = [$z_EG, $z_2F]   # growth direction in the simulation reference system for␣
↪1D simulations
}
```

Avoid renaming variables when changing from one dimension to another.

Why this is important?

In *nextnano++* the growth direction is aligned to different axis, depending on the dimensionality of the simulation. For 1D simulations, the x-axis of the simulation system is the growth direction. Nevertheless, when we change to the 2D version, the code interprets that the y-axis as the growth direction. Finally, 3D simulations assumes (implicitly) that the growth direction is aligned to the z-axis of the simulation system.

In the general case, the crystal orientation in the simulation system shall be changed every time we make a change of dimensionality, in the `global{ }` section of the input file. This shall be also be taking into account concerning the strain{ } section of the input file, when strain calculations are necessary (that in this not the case in this example).

Then, reducing or expanding the input files to another dimensions will require changes in the next sections of the input file:

- in `global`: `simulate1D{}`, `simulate1D{}`, `simulate1D{}`, and changing the crystal orientation (when necessary)
- in `grid`: `xgrid{ }`, `ygrid{ }`, `zgrid{ }`
- in `quantum` (when present): `region{}`, `boundary_conditions`
- in `strain` (when present): `growth direction`
- in `structure`: `line{}`, `rectangle{}`, `cuboid` or another shapes
- in `contacts`

Last but not least, also regions that must not appear in the plane (for 2D) or line (for 1D) of the simulations must be eliminated from the section `structure{ }`, `quantum{ }` and `contacts{ }`.

As example, *large-3D-systems-reduction_1D_nnp.in* and *large-3D-systems-reduction_2D_nnp.in* are input files for 1D and 2D simulations of the same device respectively. We recommend comparing these two versions with the corresponding 3D version.

Learning from 1D Simulations

The most frequent simplification that can be made when modeling the device is the substitution of extensive regions at the bottom of the structure, mainly the substrate and back contacts, or even buffer layers. For this device this procedure is adequate, because of the wide buried oxide layer that separates the back gate and the Si channel, our main area of interest. Figure 4.17.2.4 illustrates the final device to be simulated where the substrate and the back gate (green in Figure 4.17.2.2) were substituted by boundary conditions at the bottom of the structure (red). This is the equivalent to set this last layer as a point or plane of reference for the electrical potential or the Fermi level to a certain value.

Additionally, gates and vias that connect the external environment with the source and drain regions can be substituted by convenient boundary conditions. We will skeep this discussion concerning how to set boundary conditions that can be explored in another tutorials of our documentation related to this very important topic. What is important to mention is that 2D or even 1D versions can become valuable for modeling the eliminated regions through use of suitable boundary conditions.

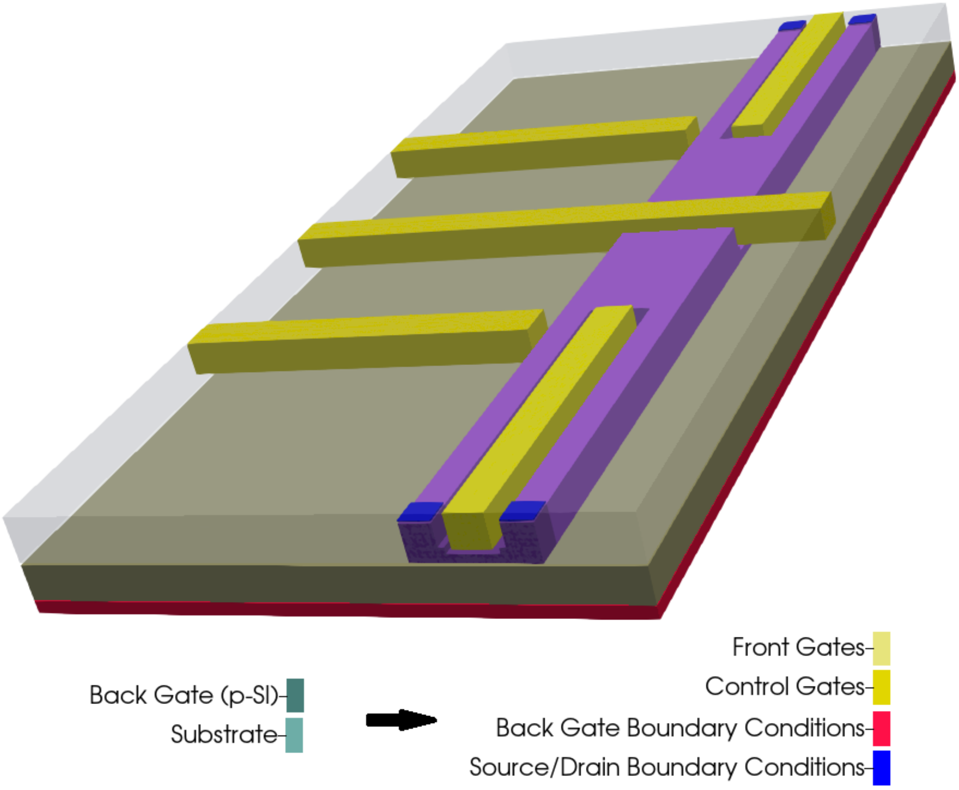


Figure 4.17.2.4: Regions substituted by adequate boundary conditions and final device representation

In 1D simulations it is required to choose the direction to be simulated that depends on the geometry of the specific device. In our example, the structure consists basically of a stack of layers where the Si layer is embedded, and is biased at the top and at the bottom. Then, a natural choice for 1D simulations of devices with this characteristic is along the growth direction that, by convention in *nextnano*++, is aligned in this case to the x-axis of the simulation system, as discussed before.

Depending on the complexity of the device it may be required to choose different points for the 1D simulations. Figure 4.17.2.5 illustrates some of these points that could be explored for the device of our example. From a quick analysis of our example we can observe that the line A is the most relevant for the first tuning of the grid, because it contains the most important coordinates of the interfaces to be examined.

The input file *large-3D-systems-reduction_1D_nnp.in* presents the device as a stack of layers passing through one of the gates over the Si channel (line A). This can be used to set up and/or verify the parameters used to model each material of the structure. After simulation, we can easily identify, for example, the conduction band across this direction as shown in Figure 4.17.2.6.

These plots were obtained by running this input file for different homogeneous grid line spacings in the growth

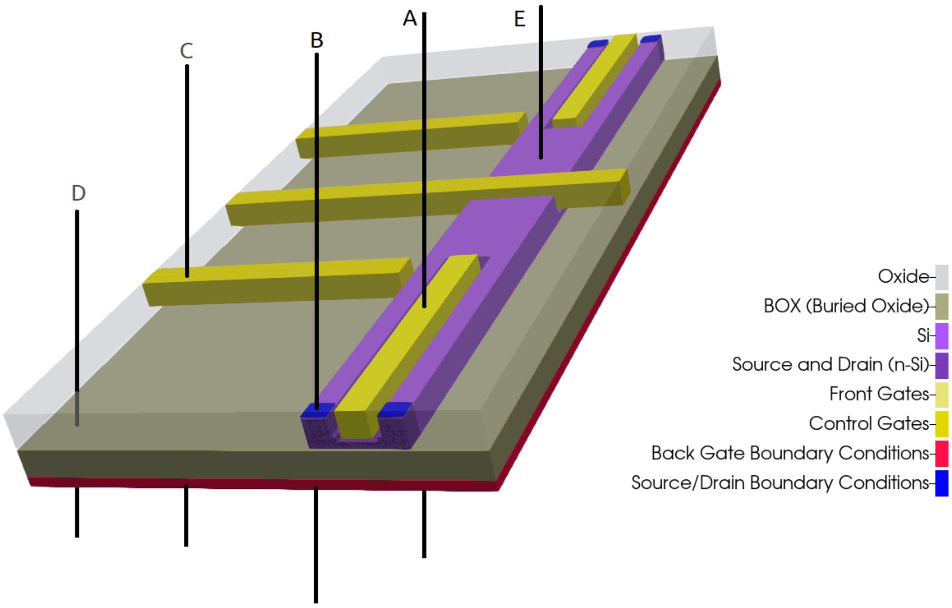


Figure 4.17.2.5: Representation of possible regions of study for 1D simulation in the growth direction.

direction (from a to d). We can easily identify the most important regions: the back-gate, the buried oxide, the channel (surrounded by oxide) and some of the top gates. Here, the most important region is the Si-channel (the active region), whose grid resolution can be increased.

Such input file runs very quickly, and it is a very good starting point for choosing a suitable grid resolution. From these plots we can observe that the conduction band is not too sensitive to the choice of the grid resolution in this direction. An ideal situation is to define a finer grid spacing in the active region and a coarse grid for the remaining parts of the device. It is recommended to make the final refining of the growth direction only in the last steps of the 2D or 3D grid tuning, for saving more runtime. In our example for the next simulations it will be used 1 nm and 5 nm grid as fine and coarse grid spacing for the growth direction, respectively (plot e in Figure 4.17.2.6).

Hint

Visualize the grid lines selecting `Simulation grid` in *nextnanomat* menu.

Refining grid in 2D Simulations

Now it is time to perform the 2D simulations, using our input file *large-3D-systems-reduction_2D_nnp.in*. It represents a slice of the device passing through the center of both front gates (FGS and FGD), parallel to the growth direction and the propagation direction, as shown in Figure 4.17.2.7.

This kind of representation can be very useful for defining the more convenient boundary conditions at equilibrium conditions for the gates and for the contacts. The device of our example requires these gates be modeled as highly-doped quasi-metallic regions at low temperatures. How to set them properly we invite you to visit our tutorial about *contacts{ }*.

At this point we will freeze the grid resolution in the growth direction, and will refine the grid spacing along the propagation direction. In this way, when talking about grid resolution or spacing we will be referring to the propagation direction.

Our main goal of these 2D simulations is the identification of the most important regions where the grid must be refined in the propagation direction. We will focus in the conduction band computed with different grid resolutions, that are presented in Figure 4.17.2.8. The data is stored in *\bias_00000\bandedges.dat* of the output folder.

As soon we decrease the grid line spacing it becomes difficult to distinguish the results from the 2D plots. For this reason, it is recommended to include in the input file some 1D sections for both directions, that makes easier to compare the results. You will find several of these sections defined in the 2D input file of our example.

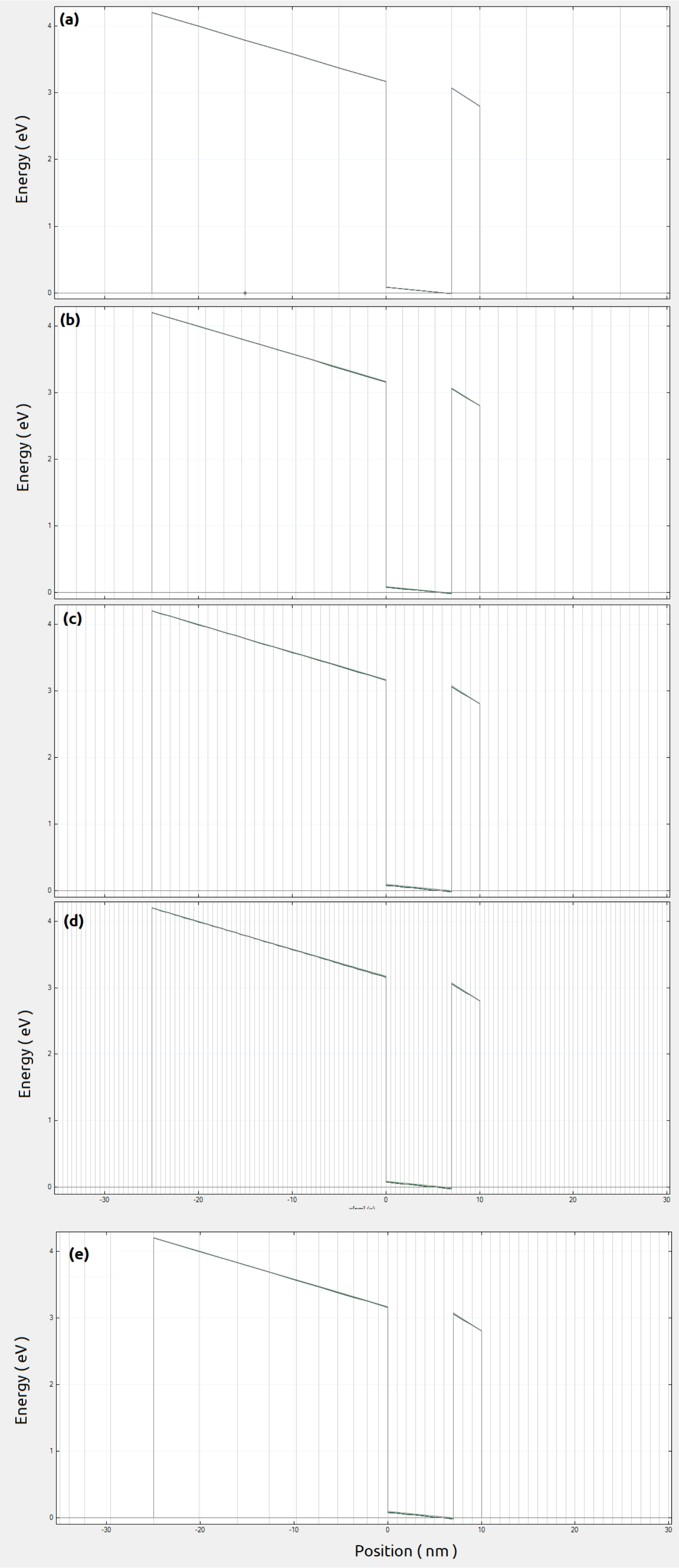


Figure 4.17.2.6: Conduction band resulting from 1D simulations in the growth direction along the line A for homogeneous grid resolution: (a) 5 nm, (b) 2 nm, (c) 1 nm and (d) 0.5 nm. The gray vertical lines represent the grid lines used in this simulation. (e) corresponds to a grid resolution of 1 nm inside the active region and 5 nm in the remaining parts of the structure (in the growth direction).

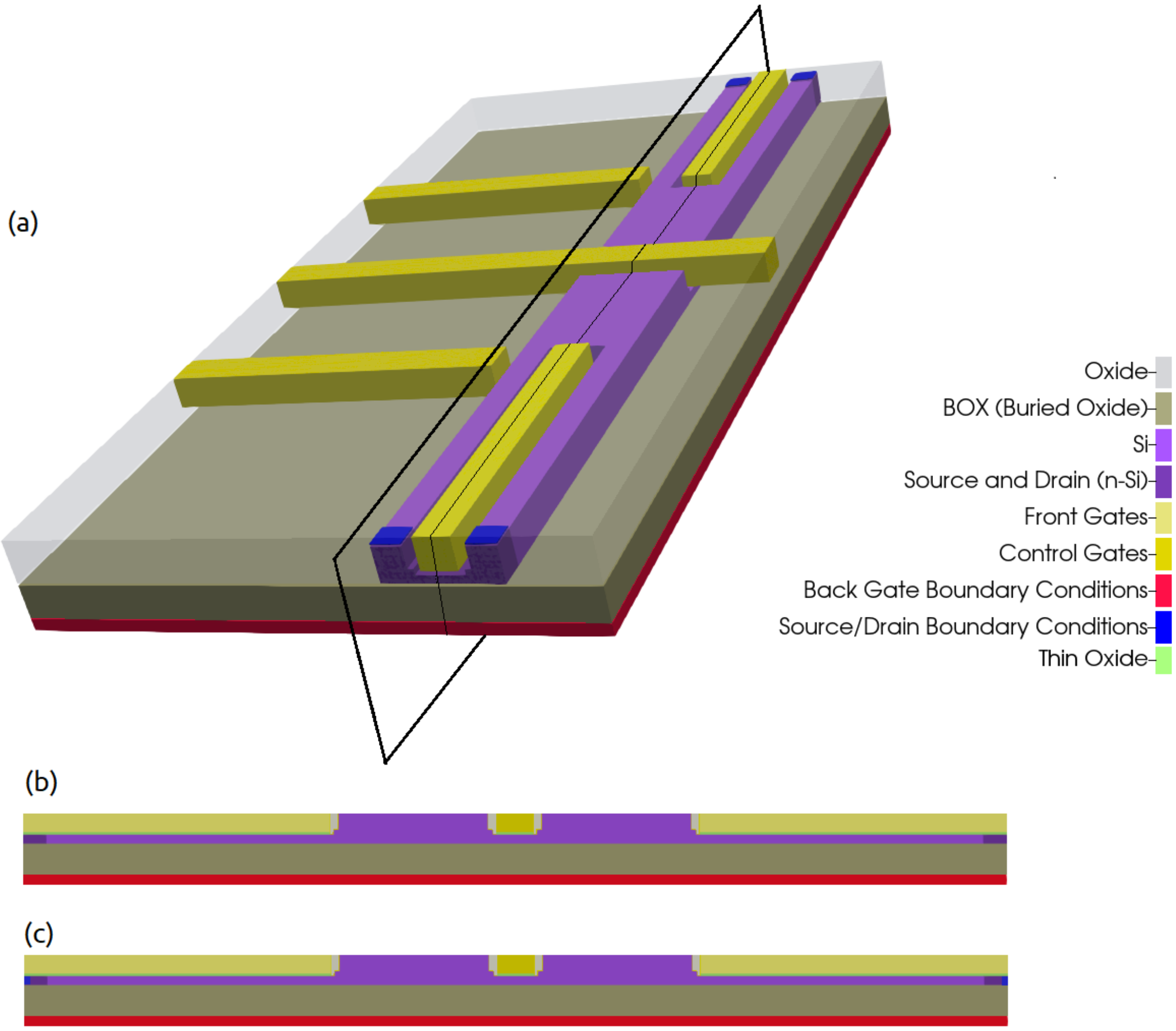


Figure 4.17.2.7: Slice simulated in our example.

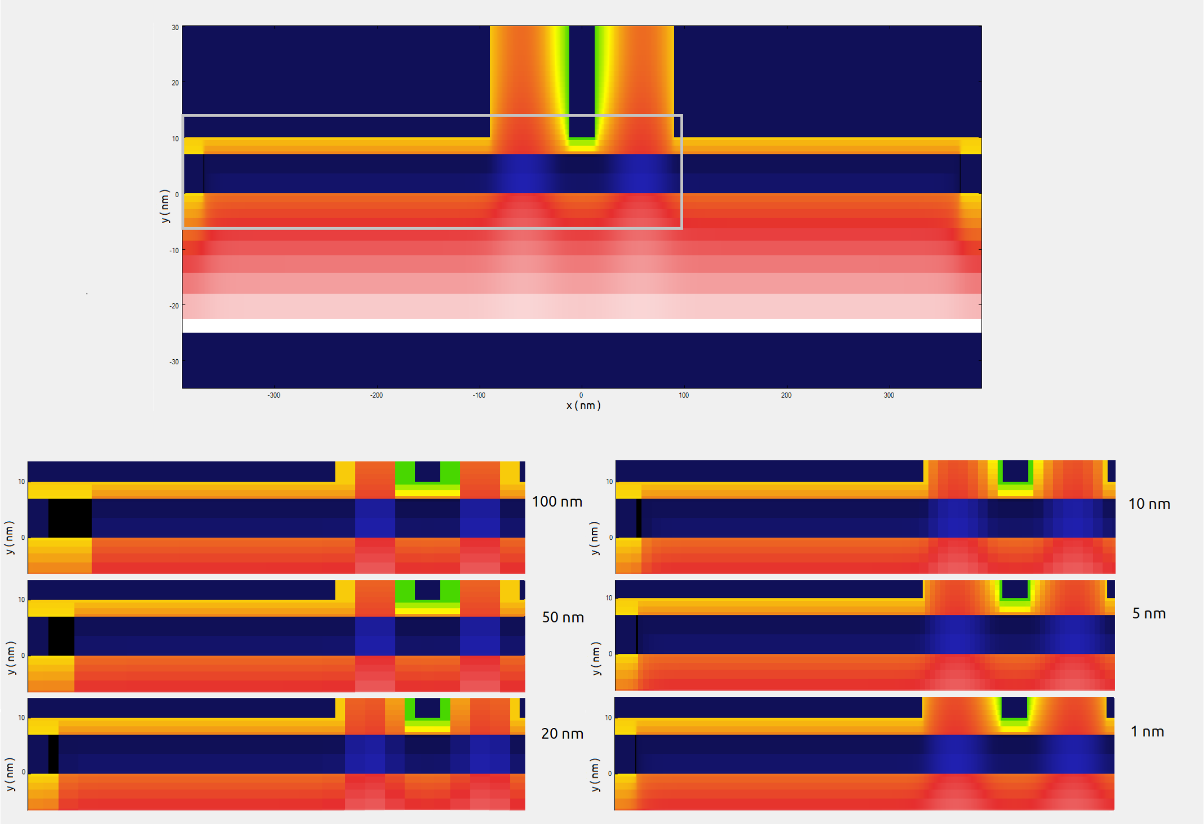


Figure 4.17.2.8: Conduction band along a plane containing the growth direction and the center of the front gates. This result was obtaining grounding all gates and contacts, except the front gates that were biased at 0.8 V. The upper image corresponds to the full simulation domain simulated. The region inside the gray rectangle is presented below for different grid resolutions.

Hint

It is highly recommended to include the coordinates of all interfaces and the one used for specifying output sections and slices in the grid definition on your input file this avoids unnecessary interpolation of the results.

Figure 4.17.2.9 presents the comparison of the conduction band just 1 nm above the interface between the buried oxide and the Si-channel (section `xz_Si_2DEG` of Figure 4.17.2.3) from 2D simulations with the different grid spacing. The corresponding results can be found in the output files *\bias_00000\bandedges_1d_xz_Si_2DEG.dat*. From the image we identify that the central region from -150 and 150 nm at the most relevant for controlling the transit of carriers from one side to the other of the channel.

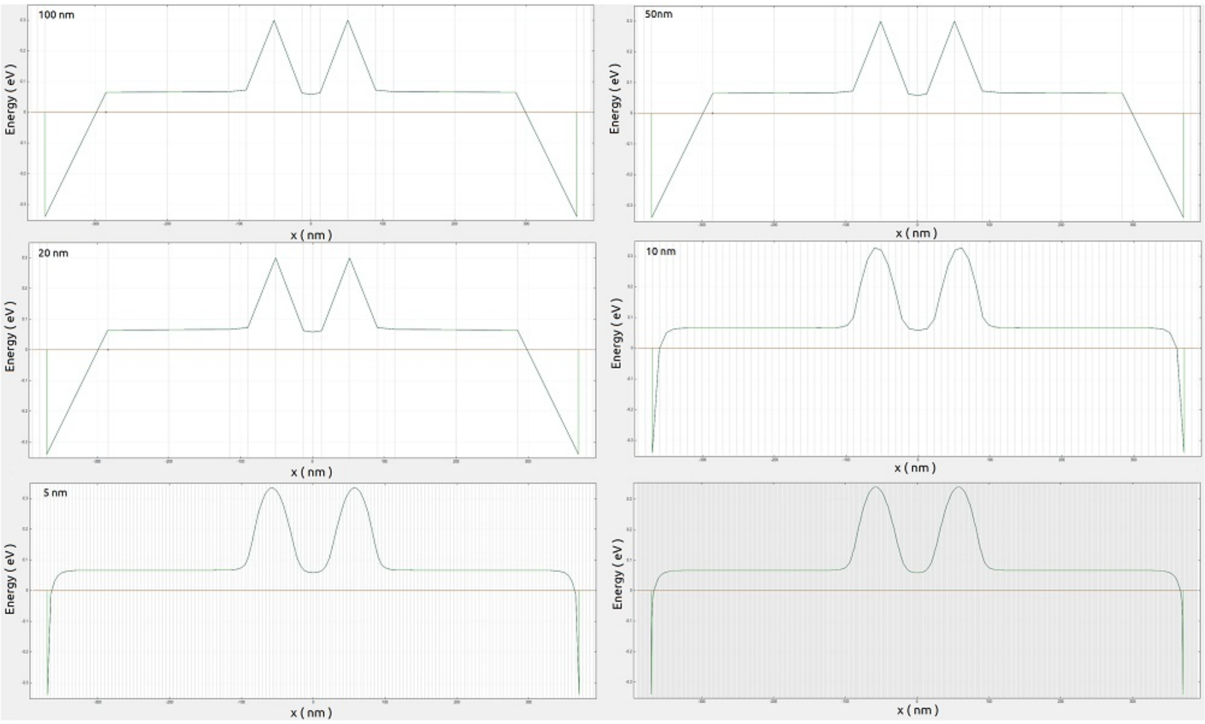


Figure 4.17.2.9: Conduction band at 1 nm above the interface between the buried oxide and the Si-channel (section `xz_Si_2DEG` of Figure 4.17.2.3) from 2D simulations with the different grid spacing. The gray lines correspond to the grid lines.

In Figure 4.17.2.10 we can observe in detail these regions for resolutions of 1, 5, 10 and 20 nm. The central region presents similar results using fine grids, while at the borders of the simulation region, a good model of the potential requires resolutions higher than 20 nm.

The first temptation is to use the minimum resolution as possible (1 nm), but this is not necessary and not recommended: we have not started the 3D simulations yet. Figure 4.17.2.11 shows how the simulation time scales with the number of nodes and the grid resolution. We observe that for coarse grid (grid line spacing around 20 and 100 nm) the time for simulation does not change too much. Nevertheless, as soon it becomes fine the time starts to increase dramatically.

A good strategy is to define different grid spacings in the x direction: small for the relevant regions (central and the contact) and larger for the ones that does not change (the remaining).

Last but not least, this simulation was performed for a specific combination of biases to the gates (0.8 V to the front gates, and 0 to the other gates and contacts). It is not necessary to simulate all bias combinations, but it is useful to check some of them that can result in larger modifications of the potential at least in active region.

Exercise:

> Run the input file *large-3D-systems-reduction_2D_nnp.in* for several grid resolutions and obtain the plot of Figure 4.17.2.11 for your system. All information required for this exercise (number

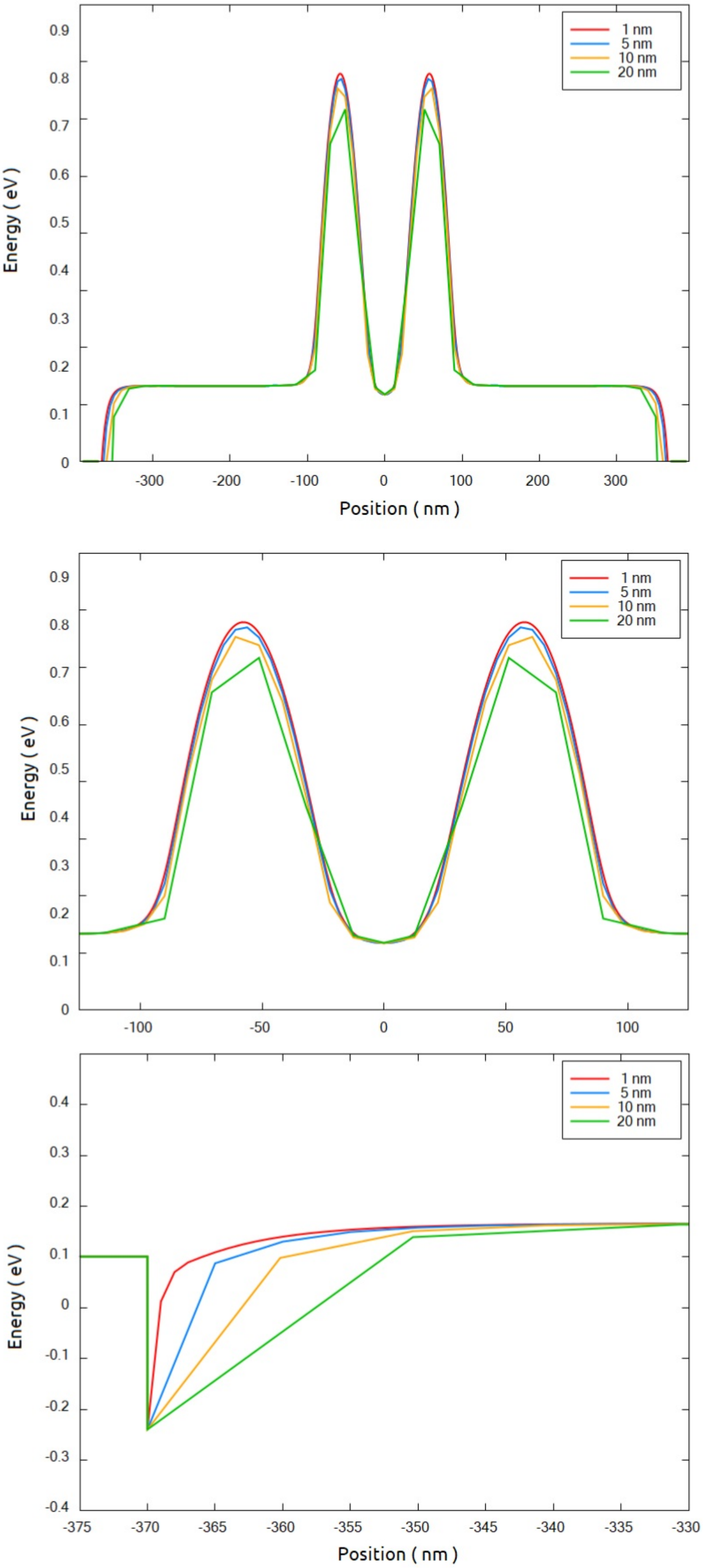


Figure 4.17.2.10: Comparison of the conduction band at a 1 nm above the interface between the buried oxide and the Si-channel (section `xz_Si_2DEG` of Figure 4.17.2.3) from 2D simulations. The central region and the source contact regions are also shown with more details.

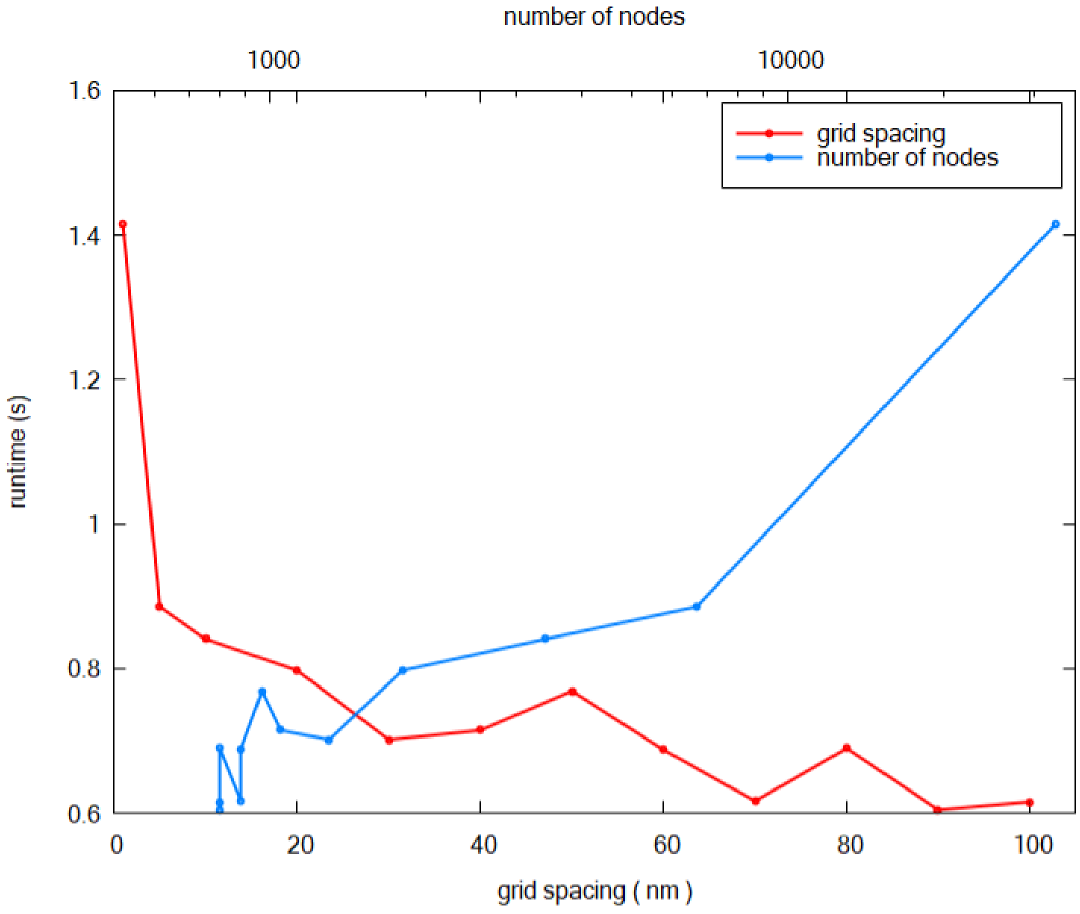


Figure 4.17.2.11: Runtime for 2D simulations as function of the number of nodes in the grid and the grid spacing.

of nodes and runtime) you can find in the file *large-3D-systems-reduction_2D_nnp.log* in the output folder of each simulation.

Hint

The performance of the simulations can be improved setting the number of threads for a single simulation in the menu `Tools` >> `Options` >> `Simulation` of *nextnanomat*.

It is also recommended to set the tab `Tools` >> `Options` >> `Executable` the command

```
-b <number the cores of your system>
```

as additional parameter passed to the executable (field `Command line` of this menu). For example, if you are a user of a 6- cores-processor, write `-b 6`.

Using the grid defined in the growth and propagation directions, we can expand to the third dimension. The result is shown in the *large-3D-systems-reduction_3D_nnp.in* that still we require further optimization, but with less effort.

We also recommend visiting our tutorials:

Optimizing electrostatics simulation for large 3D designs

Optimizing Schrödinger-Poisson self-consistent solver for electrostatic quantum dots

where we present another guidelines concerning efficient simulations of large devices in three dimensions.

Last update: 15/07/2024

Optimizing electrostatics simulation for large 3D designs

Header

Files for the tutorial located in *nextnano++\examples\numerics*

- *large-3D-systems-poisson_2D_nnp.in*
- *large-3D-systems-poisson_3D_nnp.in*
- *large-3D-systems-poisson_3D_nnp_reduced.in*

Scope of the tutorial:

- Guidelines for refining the grid in 3D-input files
- Performing electrostatic calculations efficiently

Relevant output Files:

- *\bias_00000\bandedges_1d_xz_Si_QDs.dat*
- *\bias_00000\bandedges_1d_yz_Si_QDs.dat*

- *\bias_00000\density_electron_1d_xz_Si_QDs.dat*
- *\bias_00000\density_electron_1d_yz_Si_QDs.dat*

For structures that strain computations are not needed, obtaining the electrostatic potential is the first step even when more complex computations are required.

Nevertheless, self-consistent computations of the landscape potential with the charge distribution for large devices demanding high accuracy generally consume a huge amount of memory and long execution time.

This tutorial is the second part of a methodology for reducing the time in the development of the input files for modeling such 3D structures, that can be found in *Approaching large 3D designs with Schrödinger-Poisson self-consistent solver*. In this methodology we suggest to start by tuning the grid of the simulation using 2D versions of the correspondent 3D input file. Although it is not mandatory following this first step for implementing the suggestions in this tutorial, we strongly recommend its reading at *Reducing dimensionality of large 3D designs*, for understanding of the main concepts also used here.

We will take as an example a structure that can be used in a semiconductor-based quantum computer, that we introduced in the first tutorial of the methodology and quickly summarized below.

Device to be simulated

Figure 4.17.2.12 presents a simplified version of a device found in the literature *[Kriekouki2022]* that consists basically of a 7 nm-Si layer buried in a silicon dioxide structure. This silicon layer corresponds to the channel where the quantum operations are performed.

The transport of the carriers depends on the combination of the voltage applied to the gates (FTS, FTD, LG1, LG2, LG3) at the top of the structure isolated from the silicon channel by a thin layer of oxide. At the bottom of the structure, just below the thick buried oxide layer, a back gate plays also an important role in the definition of the landscape potential. The source and drain contacts in this scenario act as the reservoirs that will provide the carriers that will propagate in the channel.

Applying adequate boundary conditions, the device to be simulated can be simplified as shown in the Figure 4.17.2.12 (shown in (b)).

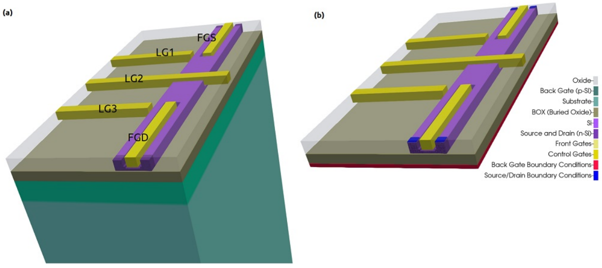


Figure 4.17.2.12: Device to be simulated. The Si-channel is buried in the oxide. FTS, FTD, LG1, LG2, and LG3 at the top of the structure and the back-gate, between the thick oxide layer under of Si layer (BOX) and the substrate, are gates used to shape the electrostatic potential. Source and drain act as reservoirs of carriers propagating through the channel. Device (a) before and (b) after applying adequate boundary conditions.

Starting simulations in the semiclassical domain

The first thing we have to keep in mind is the goal of our simulation: which equations have to be solved, the accuracy we want to achieve, and other post-processing tasks that will be necessary. In our practical example is expected that under certain bias combinations a quantum dot is formed in the channel close to the lateral gates LG1 and/or LG3. Then, an accurate electrostatic potential self-consistently solved with the Schrödinger equation is required for obtaining a good estimate of the wave functions in the device, that will also be used in coherent transport calculations.

Nevertheless, self-consistent quantum computations with Poisson equation means that we need to a sufficient number of eigenvalues enough to reproduce the carrier densities that will be used in the next Poisson iterations. For this reason, the runtime of the whole simulation does not only scale with the number of nodes of the structure for the electrostatic potential calculations, but also depends on the size of the quantum region and the number of eigenvalues that has to be solved.

Then, as a general rule, setting the grid for 3D simulations is more efficient when started with semiclassical calculations, where only the Poisson equation, or even the coupled current-Poisson equations, is solved. Additionally, *nextnano++* always uses the resulting potential as a first estimate for the next steps of the quantum computation and other calculations. As a rule of thumb, run and verify the results step by step. In other words, perform the next step of the computations when the previous step (the electrostatic problem) properly converged. In this tutorial we will focus only in the solution of the Poisson equation self-consistent with the semiclassical densities of electrons, for refining the grid of 3D input files. Hints for optimizing the performance of quantum simulations will be provided in a separated tutorial (*Optimizing Schrödinger-Poisson self-consistent solver for electrostatic quantum dots*).

Refining the grid of 3D-input files

It is more efficient following the first step of our methodology, where the grid is progressively refined in the growth and in the propagation directions, that results in the file *large-3D-systems-poisson_2D_nnp.in* for 2D simulations. Then, it follows that the 3D version of this input file is simply the extension of the refined 2D version, that now includes the lateral gates LG1 and LG3 in the structure. The growth direction for 3D simulations is aligned to the z-axis of the simulation domain (see file *large-3D-systems-poisson_3D_nnp.in*). Figure 4.17.2.13 shows the nomenclature of the most important points in the device coordinate system, sections and boundary conditions used in this input file.

It is clear that the previous grid tuning in 1D or 2D simulations is not a mandatory procedure: simultaneous tuning of the three axes in the 3D input file could be also performed. The disadvantage of this approach is that the execution time of each simulation depends on the number of nodes on the grid, that it is in higher number for the 3D case.

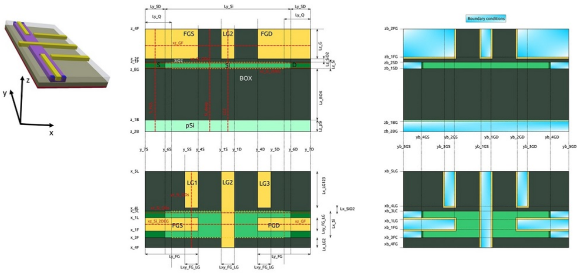


Figure 4.17.2.13: Device reference system and most important coordinates: (a) the 3D representation, (b) structure definition, and (c) structure after applying boundary conditions to the contacts and gates. Dotted lines (in red) represent sections defined in the input files.

Now it is time to start the simulations. Refining of the grid in the last dimension (x-axis) does not require, initially, high accuracy. In this way, the criteria that define the end of the convergence process (residuals, for example) can be "relaxed" in these first estimates. The idea is to identify regions of interest (ROI) where a fine grid has to be necessary to a suitable description of the density of electrons or holes in the simulated domain.

This is an iterative process where, looking at the conduction bands or the density of carriers in the ROI, we will try to refine the x-grid that the resulting density presents a smooth decay. For this task, it is recommended to define 1D slices in the most important ROIs and to overlay different plots in the same image within our graphical interface (*nextnanomat*). In our practical example, our objective it to capture the results in the region where the quantum dots are expected to be formed and different points where the density of carriers or the potential will be analyzed. Slices where the potential presents the steepest slopes are also important to be included in this analysis.

Figure 4.17.2.14 shows the conduction bands of two important ROIs, for a particular combination of biases (0.8V to both front gates, 4 V at LG1 and LG3, 1.7 V at the central gate (LG2) and 0 V for the remaining gates and contacts). In the image, the `xz_Si_QDs` section corresponds to a slice at the region on Si-channel close to the interface with the gate LG1 (for x = 35 nm) and 1 nm above its interface with the buried oxide (BOX), where the quantum dot is expected to be formed. `xz_Si_2DEG` is the slice of the conduction band along the y direction at 1 nm below the oxide under the front gates (x = 0 nm). Also important to observe is the slice `yz_Si_QDs` defined by the intersection of the plane along the longitudinal axis of the LG1 gate and the plane passing at 1 nm above the interface between the Si-channel and the BOX. These sections are shown in Figure 4.17.2.12 using dotted lines (in red).

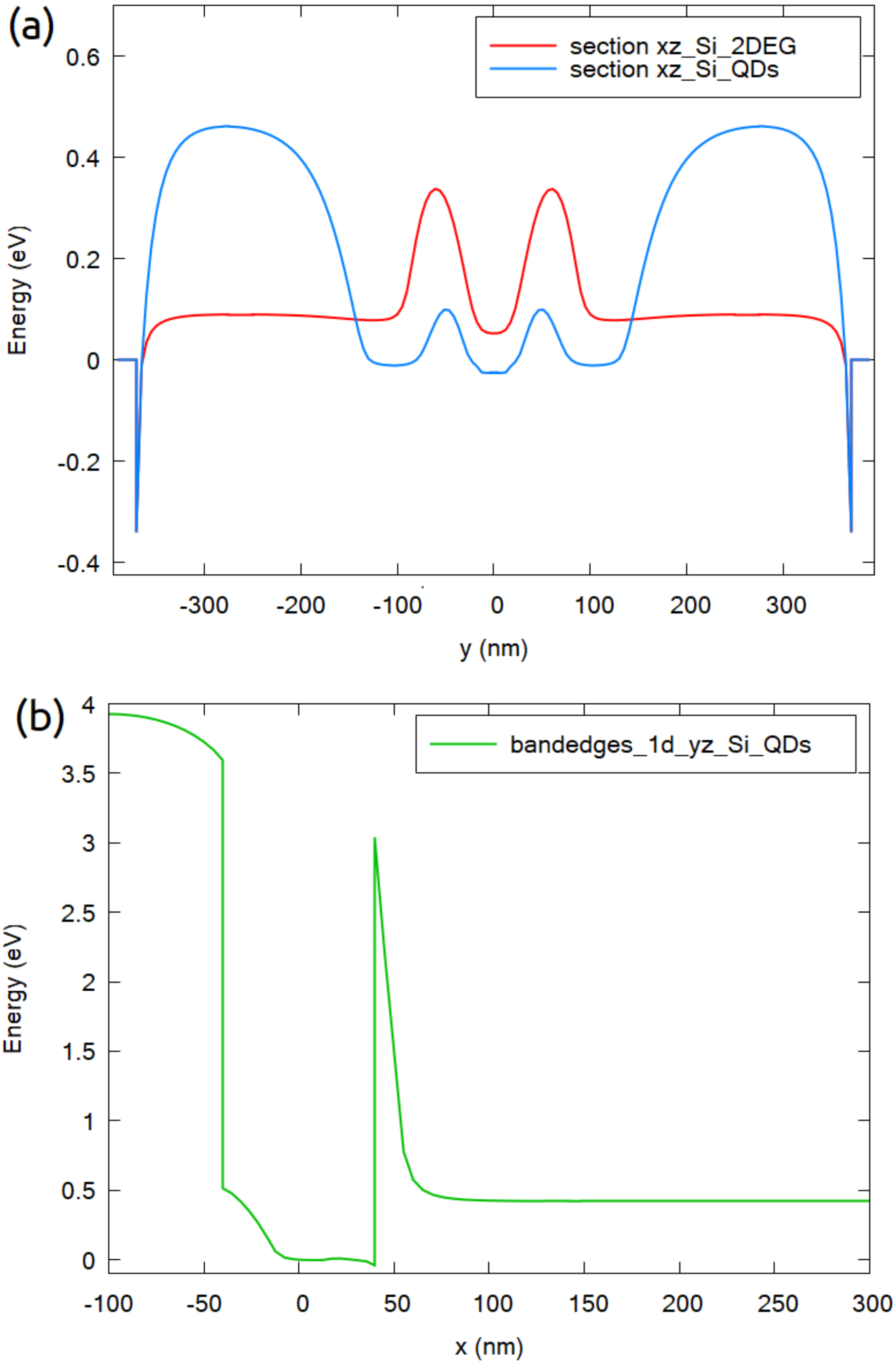


Figure 4.17.2.14: Slices of the conduction band in the two most relevant regions of interest for a particular combination of bias applied to the gates (see text): (a) `xz_Si_QDs`, at 1 nm above the interface between the Si-channel and the BOX, close to the lateral gate LG1 (x = 35 nm), and `xz_Si_2DEG`, at 1 nm below the oxide under one of the front gates (x = 0 nm), (b) `yz_Si_QDs`, at 1 nm above the interface between the Si-channel and the BOX, in the plane containing the longitudinal axis of the gate LG1.

The first step of the grid definition in the third axis consists in the elimination of unnecessary areas of the device. We need to distinguish two situations: solutions of quantum mechanics problems, or solutions of the electrostatics of the device only.

The first situation, when the semiclassical computations will be followed by computation of the wave functions, will demand more attention when eliminating or even reducing areas from the simulation domain. In this case we recommend that the final size of the device be defined only when the first quantum simulations be performed. A reduction of several undesired nodes at this moment still will bring benefits, but it is important a future evaluation of the impact of these cuts in the boundary conditions for the quantum calculations. Keep in mind that the wave functions can penetrate certain interfaces. Then preserve certain margin around the interfaces in order to allow a priori that some tail of the wave function can be properly calculated.

For the other situation, when we are only interested in the electrostatic solutions, this is the appropriate moment to cut these regions from the simulation domain.

Our particular example is in the first situation, and the elimination of some unnecessary areas can be valuable. We can expect a priori that the potential of the lateral gates LG1 and LG3 close to the Si-channel does not depend on the length of these gates, because of the large potential barrier between the Si-channel and the surrounding oxide of each gate, that practically results in vanishing of the wave functions at the interface of both materials. Then, this simplifies a lot our 3D simulations, because the lateral gates can be reduced and substituted by the convenient boundary conditions.

Figure 4.17.2.15 presents the impact of the changes of the lateral gate lengths on the results of the conduction band for the section `xz_Si_QDs`. The results are practically equal, if the gate lengths are larger than 100 nm. Then, we will use 100 nm as the length of the lateral gates for the reduced version of the 3D input file. This value is also reasonable when we analyze the results for the section `yz_Si_QDs` (the growth direction), that practically independent of the choice of the level of this reduction.

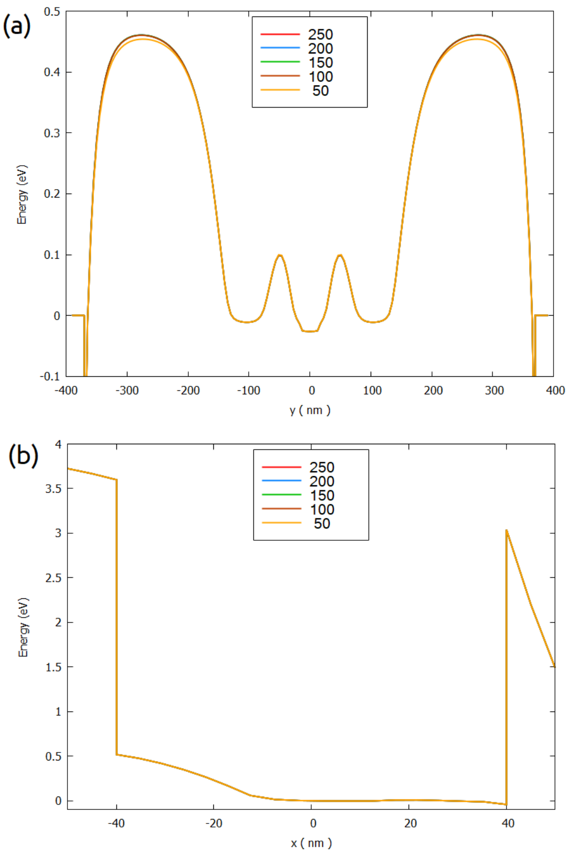


Figure 4.17.2.15: Conduction band in the quantum dot region as function of the length of the gates LG1 and LG3 in the simulation: (a) slice `xz_Si_QDs`, and (b) slice `yz_Si_QDs`

Another natural candidate to be eliminated is the region from the start of the simulation system (`$x_min` = `$x_4F`).

Figure 4.17.2.16 shows the results of the conduction band for different values of `$x_min`, for the same slices of the previous image. In this case, the extension of the negative axis of the simulation domain plays an important role in the definition of the electrostatic potential at the left border of the Si-channel (at `x = -40 nm`), while the region close to the lateral gates practically does not change (at `x = 40 nm`).

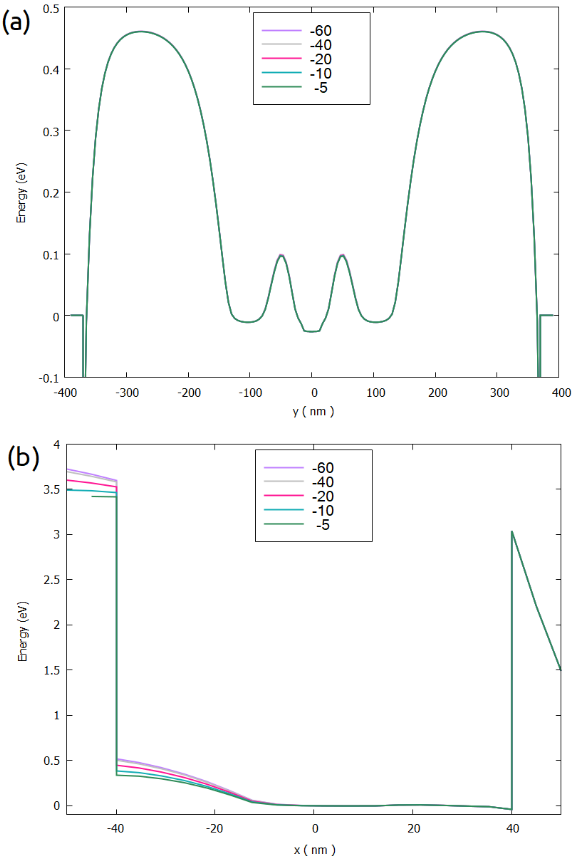


Figure 4.17.2.16: Conduction band in the quantum dot region as function of the value of `$xmin`: (a) slice `xz_Si_QDs`, and (b) slice `yz_Si_QDs`.

Looking at to the conduction band results not enough to decide what it is an optimal value for `$xmin` to be used in the next simulations. When this happens, more careful evaluation of the impact of these cuts have to be done. Our suggestion is to verify the goals of the simulations and to combine results. In our example, we can overlap to the conduction bands the corresponding density of electrons (our goal) and observe the differences using different cuts in the ROIs, as illustrated in the Figure 4.17.2.17.

From this figure we can observe that in terms of electron density, they are not affected by the value of `$x_min` chosen. Similar analysis must be performed for all relevant results of the calculations.

large-3D-systems-poisson_3D_nnp_reduced.in is the resulting input file after these reductions, and it will be used in the next computations. As we can observe, we are using a very conservative approach concerning the cuts around the Si-channel and the lateral gates, in order to give an example that would be done in a more general way.

If no quantum computations are necessary, this would be the moment of increasing the accuracy of the simulations by requiring lower residuals for the density and fermi levels, until the results (for example, density of electrons) does not change within a certain precision from one simulation to the other. If necessary, you can include some more lines in the positions of the grid for getting better results.

Once the grid is completely defined, make a final check concerning the sensitivity of the calculations with changes in the grid resolution.

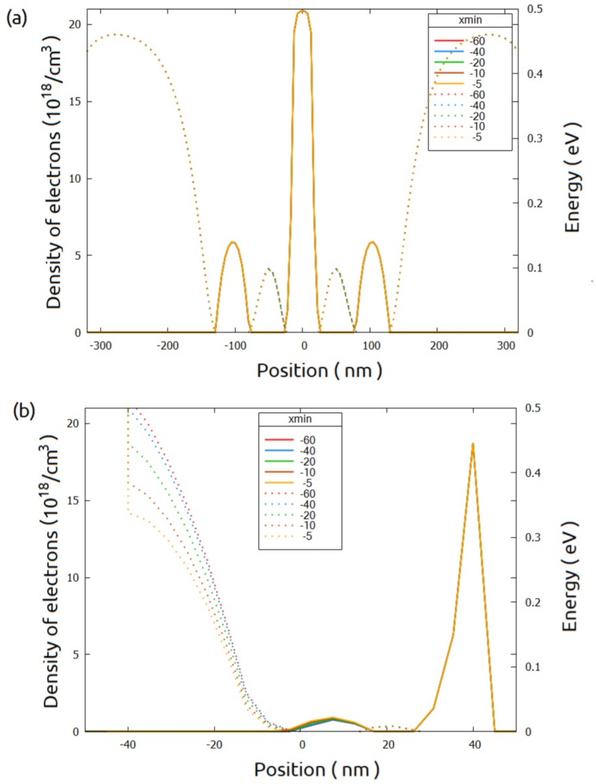


Figure 4.17.2.17: Conduction band (dotted lines) and density of electrons (solid lines) in the `xz_Si_QDs` and `yz_Si_QDs`.

Considerations if quantum computations will be required

Semi-classical computations of the density of electrons are very useful to identify how wide is actually the region where the carriers can be observed. Specially for self-consistent calculations, this evaluation is tremendously valuable because allows us to estimate the minimum size of the quantum domain to be simulated. It is always relevant to keep in mind that the total execution time in this case will also be affected by the number of the nodes in the quantum region and the number of eigenvalues to be used for self-consistent calculations, as we mentioned before. We will discuss in the more detail in our next tutorial of the presented methodology.

From Figure 4.17.2.17 we could identify the bounds of the region where most of the electrons are present. Then it is natural to choose them as first good estimate for the quantum region. Nevertheless, we must not forget that this result was obtained for one a specific combination of biases applied to the gates and the contacts. Then, it is convenient to make a quick check for some other combinations in order to verify if this region need to be extended.

Figure 4.17.2.18 shows the density of electrons overlapped with the respective conduction band for another bias combinations. Starting from the one presented above (0.8V to both front gates, 4 V at LG1 and LG3, 1.7 V at the central gate (LG2)), we changed either the bias on the back gate, in the central gate (LG2), or simultaneously in the other lateral gates (LG1 and LG3).

We can observe that applying bias to the back gate greater than 1.0 V, will require an extension of the quantum region from `[-150, 150]` to `[-200, 200]` in the y-direction. Changes in the bias of the lateral gates does not change too much the semi-classical density of electrons distribution. Then, in the next step we suggest to start defining the quantum region limited to the smaller interval (`[-150, 150]`), at least for the first setup of the input file including quantum calculations.

We also recommend visiting our tutorials:

Optimizing electrostatics simulation for large 3D designs

Optimizing Schrödinger-Poisson self-consistent solver for electrostatic quantum dots

where we present the first step of the methodology (the first one) and how to proceed for the case of the quantum

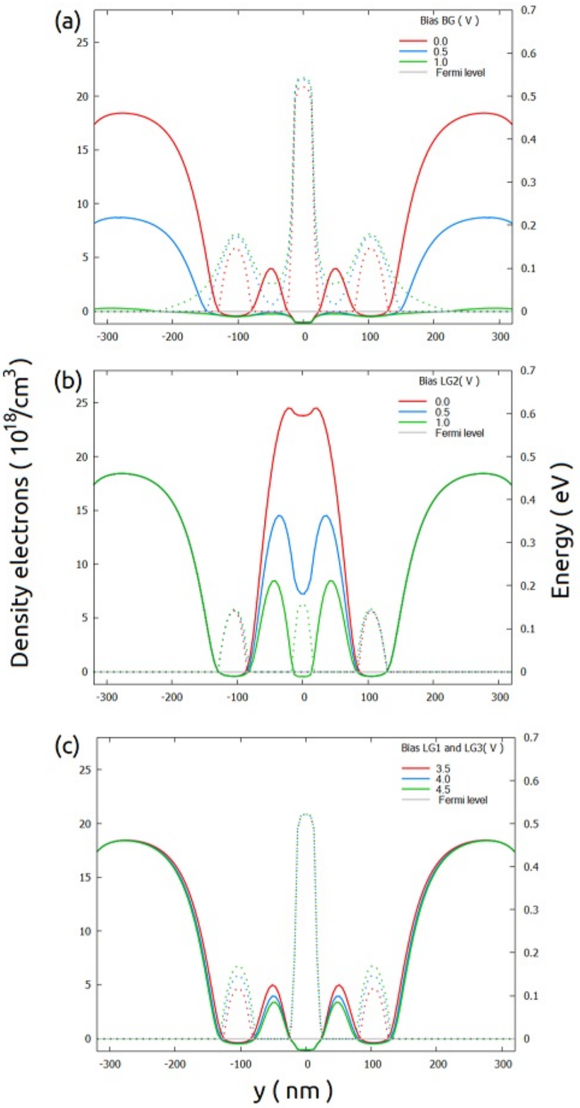


Figure 4.17.2.18: Conduction band (solid lines) and density of electrons (dotted lines) in the `xz_Si_QDs` sections changing only one of the bias of the combination discussed above: (a) the back gate, (b) the central lateral gate (LG2), and (c) simultaneously to the lateral gates LG1 and LG3. Here the full length of the lateral gate LG2 (250 nm) was used.

calculations (the second one).

Last update: 15/07/2024

Optimizing Schrödinger-Poisson self-consistent solver for electrostatic quantum dots

Header

Files for the tutorial located in *nextnano++\examples\numerics*

- *large-3D-systems-schroedinger_3D_nnp_initial.in*
- *large-3D-systems-schroedinger_3D_nnp_final.in*

Scope of the tutorial:

- Guidelines for setting the quantum calculations in 3D-input files of large devices
- Dimensioning the quantum region

Introduced Keywords:

- `quantum{ }`
- `grid{ xgrid{ } ygrid{ } }`

Relevant output Files:

- *\bias_00000\bandedges_1d_xz_Si_QDs.dat*
- *\bias_00000\bandedges_1d_yz_Si_QDs.dat*
- *\bias_00000\iteration_quantum_poisson.dat*
- *\bias_00000\quantum\probabilities_shift_QuantumRegion_Delta3_1d_xz_Si_2DEG.dat*
- *\bias_00000\quantum\probabilities_shift_QuantumRegion_Delta3_1d_yz_Si_2DEG.dat*
- *\bias_00000\quantum\occupation_QuantumRegion_Delta1.dat*
- *\bias_00000\quantum\occupation_QuantumRegion_Delta2.dat*
- *\bias_00000\quantum\occupation_QuantumRegion_Delta3.dat*
- *nn_Large_Devices_3D_initial_version_quantum_nnp.log*

Setting up input files for 3D-simulations of the self-consistent Schrödinger-Poisson or self-consistent Schrödinger-current-Poisson system of equations can demand some effort in terms of memory allocation and time consumption, if a systematic approach is missing. This development can become a real challenge when the dimensions of the devices are large (some can be of order of microns) and a fine grid (few nanometers) is required.

This tutorial aims to assist you to reduce such effort, and it is the third part of the methodology *Approaching large 3D designs with Schrödinger-Poisson self-consistent solver*, that we strongly recommend being followed.

The input file *large-3D-systems-schroedinger_3D_nnp_initial.in* was obtained in the first two steps of this methodology for the structure that we will very briefly summarize in the next section. This file presents a suitable grid resolution (only sufficient, but not optimally, refined) for obtaining a first estimate of the bounds of the region where the quantum computations will be performed. Unnecessary regions on the devices were eliminated or replaced by convenient boundary conditions.

Following these previous steps are not mandatory for the discussion in the tutorial, but it is very advantageous avoiding grid refinement or performing such tasks directly on 3D-simulations. We remark that there is not a unique way to do it, but it has been used for numerous cases, and provided very good results in most of them.

Device to be simulated

Figure 4.17.2.19 presents a simplified version of a device that is proposed as a possible semiconductor-based implementation of a quantum computer found in the literature *[Kriekouki2022]* with dimensions of 400 nm x 800 nm x 70 nm. It consists basically of a 7 nm-Si layer buried in a silicon dioxide layer. By applying bias to the gates deposited at the top of the structure (FTS, FTD, LG1, LG2, and LG3) and at the bottom of the oxide (the back gate) the electrostatic potential can be modified, in order to control the transport of carriers through the silicon layer (the channel of the system). The source and drain are the reservoirs of carriers.

Applying adequate boundary conditions, the simulation domain can be reduced as shown in the Figure 4.17.2.19 (shown in (b)). The nomenclature of the most important coordinates and sections defined in the input file are summarized in the same image in (c).

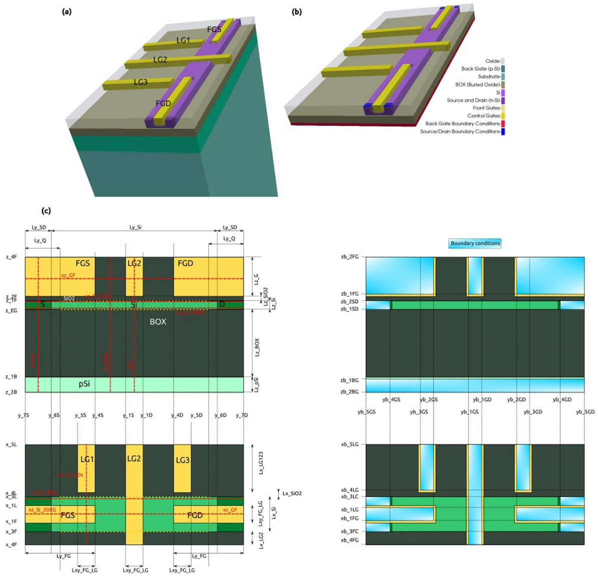


Figure 4.17.2.19: Device to be simulated. The Si-channel is buried in the oxide. The electrostatic potential is shaped by applying bias to the gates (FTS, FTD, LG1, LG2,LG3 and the back-gate). Source and drain act as reservoirs of carriers propagating through the channel. Device (a) before and (b) after applying adequate boundary conditions. The most important coordinates and sections (dotted lines in red) are shown in (c).

Setting input files for self-consistent calculations of Schrödinger-Poisson equations

As we mentioned before, self-consistent solution of Schrödinger and Poisson equations demands a good strategy in order to reduce the simulation time when tuning the grid. Usually smooth wavefunctions in some region of interest (ROI) require a fine grid resolution and enough number of states to compute the quantum mechanical density of carriers that iteratively will also be used in the solution of the Poisson equation.

Another important issue that show be addressed is the choice of the boundary conditions at the borders of the quantum region. It has to be constantly observed if they are consistent with the models used in the simulation.

Below we present some hints that may be explored for designing an efficient input file for 3D simulations.

Define the goals of the quantum computations

The simulation time of self-consistent Schrödinger-Poisson simulations depends on the time expended in the solution of the Poisson equation and the time for obtaining the quantum solution.

As we showed in previous tutorials, the time for solving the Poisson equation scales with the number of nodes in the grid of the simulation domain. On the other hand, the solution of the Schrödinger equation demands runtimes scaling with the number of nodes in the quantum region, the number of eigenvalues to be computed and also the model and corresponding solver to be used.

Below we will provide some tricks related to these aspects for getting excellent results with less effort.

Optimizing the grid within the quantum regions

1. Defining the bounds of the quantum region: at the beginning does not need to be perfect!

The nodes in the quantum region consist on a subset of the grid points of the simulation domain that are within and at the borders of are region where the Schrödinger equation will be solved. In other words, limiting the size of the quantum region of interest (QROI) and its corresponding grid resolution in the first phase of the quantum simulations will boost the input file development.

Any previous understanding of the physical phenomena in the device may be used to introduce simplifications in the QROI design. Let us present one simplification from our practical example. A quantum dot in the Si channel is expected to be present just in the channel, close to one or both lateral gates (LG1 and LG3) depending on the bias applied to these and the other gates. In this way, if our goal is to compute the density of carriers in the region where the quantum dots appear, the number of nodes of the QROI will represent a very small subset compared with the number of nodes in the whole domain.

A trick for estimating the bounds of the quantum region is to look at the density of electrons from the semi-classical calculations (solving only the Poisson equation). Please, refer to our tutorial *Optimizing electrostatics simulation for large 3D designs* concerning some considerations that may be taken into account. Figure 4.17.2.20 presents the results of such simulations for the conduction band overlapped with the semi-classical density of electrons for the sections `xz_Si_QDs` and `yz_Si_QDs` under a particular combination of biases (0.8V to both front gates, 4 V at LG1 and G3, 1.7 V at the central gate (LG2) and the remaining gates and contacts are grounded). These sections, shown with red dotted lines in Figure 4.17.2.20, correspond to slices at the region on Si-channel where the quantum dot is expected to be formed.

Although the electrostatic potential is shaped by each specific combination of bias applied to the gates, the bounds of the QROI estimated by the semi-classical electron distribution does not change too much if the biases are around the first operation point, as we showed in the tutorial concerning the electrostatic calculations mentioned above. The bounds of the QROI resulting from this analysis are `x = [-40, 40]`, `y = [-150, 150]` and `z = [0, 7]`.

The device of our example presents a geometrical symmetry related to the plane y = 0. We can take advantage of this property by reducing even more the QROI for the first tuning of the parameters concerning quantum computations. Our main objective here is not even to get good results, but to have a first idea about the convergence process of the system of equations, the required grid resolution within the quantum region, and to verify if the boundary conditions at the borders are satisfied.

In this way we can weak a little the criteria of the convergence of the quantum_poisson solver, requiring a low number of iterations (for example 10 iterations, or `$quantum_iterations = 10`). Keeping these requirements in mind we can start defining a reduced QROI with `y = -150` and `y= -50` as the bounds in the y-direction.

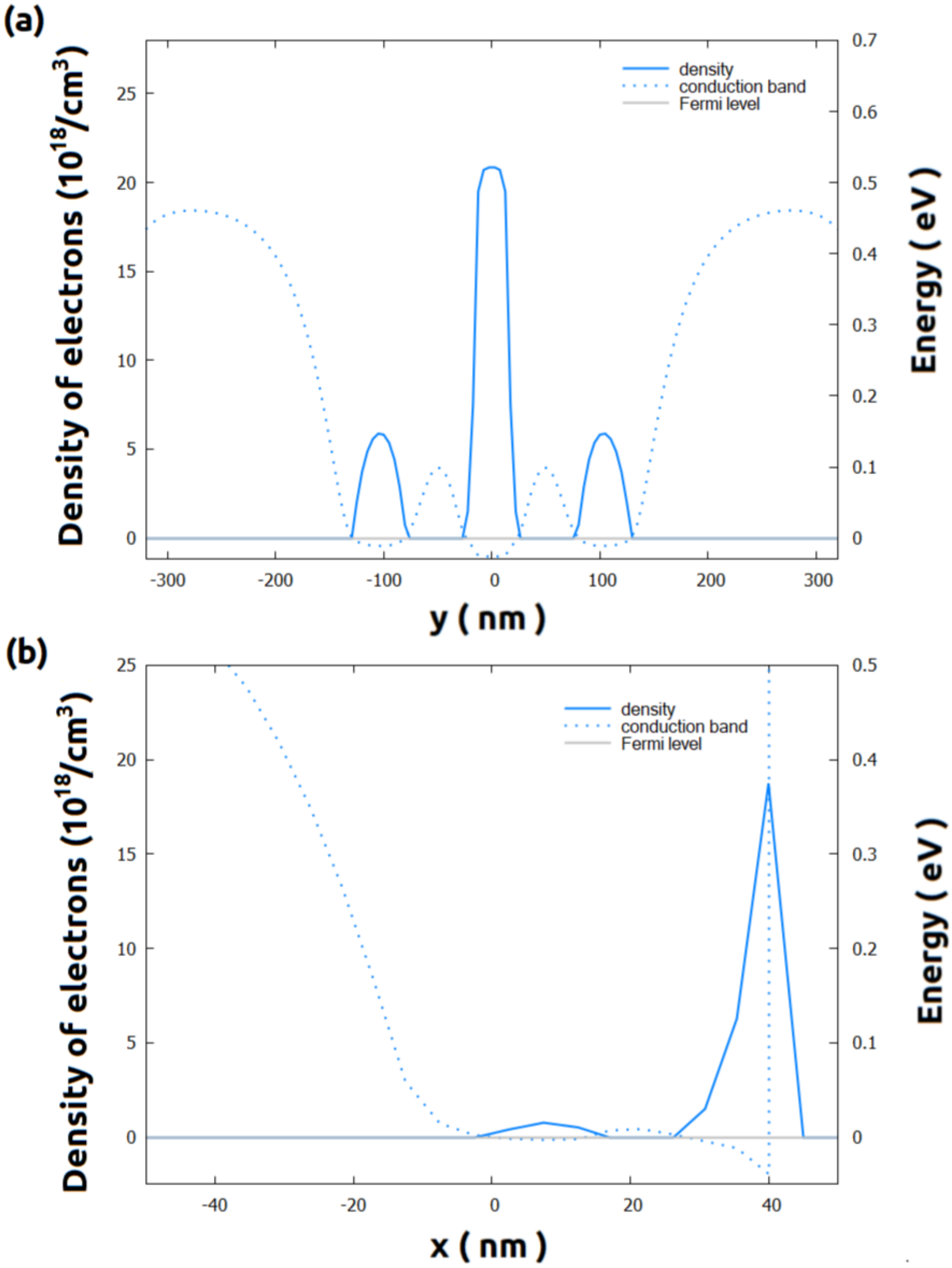


Figure 4.17.2.20: Conduction band (dotted lines) and semi-classical density of electrons (solid lines) in the slices `xz_Si_QDs` and `yz_Si_QDs` (see red dotted lines in Figure 4.17.2.20), when 0.8 V is applied to both front gates, 4 V to LG1 and LG3 and 1.7 V to the central gate LG2. The remaining gates and contacts are kept grounded.

Hint

You can save some time and storage disabling all outputs files that are not relevant or did not change from one run to the other, like contacts, intrinsic density, and material.

2. Finding a suitable number of eigenvalues

The Hamiltonian to be solved in the Schrödinger equation is specified in the section `quantum{ }` of the input file. In the section *region{ }* of our documentation you will find the models currently implemented in *nextnano++*. Independent of your choice we recommend to use at this point the computationally lighter one: the single-band. The relevant bands to be taken into account in these calculations must be defined in the input file. In our example, the band gap of silicon, the material of the region of interest, is defined by the minimum of the Delta band.

As we mentioned above, we need to choose a number of eigenvalues enough to compute the density of carriers from the wavefunctions obtained after each quantum iteration. This quantity will be injected in the Poisson equation in the next iteration, and a new electrostatic potential will be computed. Then, here we need to do a trade-off: the number of states can not be too small, but also not too large.

Low number of computed states generates truncated quantum density that, when included in the next iteration of the solver of Poisson equation, may change the electrostatic potential in another direction, and more frequently may not converge. On other hand larger number of states will require more computational effort unnecessary for this first tuning.

How to choose a suitable number of eigenvalues? The answer is simple: guess, compare and improve. Remember: our grid is still coarse. Then, this is the best moment to explore a first guess. We recommend that, starting with 10 states, for example, to increase this number and compare some relevant results iteratively, instead of simply sweeping the variable `$N_states` in our input file.

Now it is time to perform the first calculations. Remember: what it is important to observe here is how the residuals behave during the convergence process when new states are added. Figure 4.17.2.21 shows the evolution of the residual of the density of electrons as function of the number of eigenvalues. We can see that the residual decay faster when more states are included in the computation. The resulting conduction band in two relevant sections does not change substantially for number of states larger than 20. The reason for this can be inferred from the occupation number for one of the Delta bands: it is required computing at least 20 states in order to converge that 4 states are actually occupied.

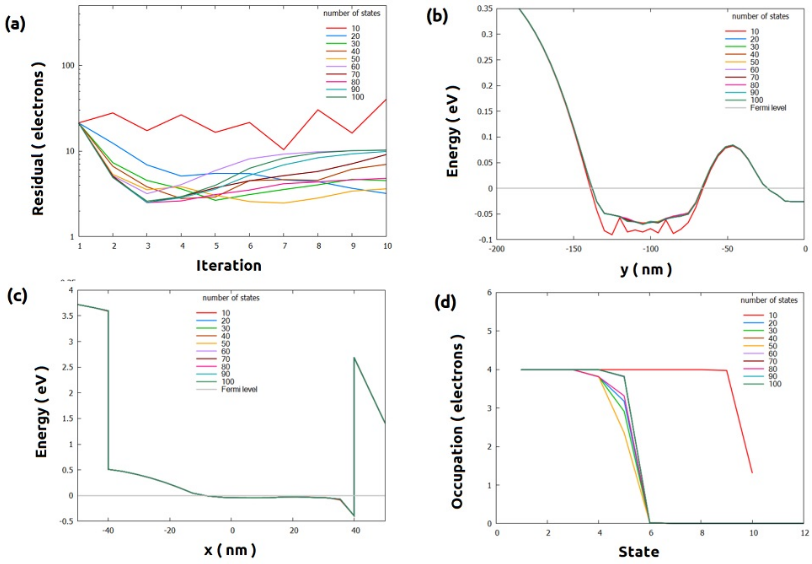


Figure 4.17.2.21: Results of the self-consistent Schrödinger-Poisson simulations, in the reduced QROI as function of the number of eigenstates computed, after 10 iterations. (a) The evolution of the residuals, (b) and (c) the conduction band for the sections `xz_Si_QDs` and `yz_Si_QDs`, respectively, and (d) the occupation number after only 10 iterations. These are intermediate results: the convergence process was still not completed.

Please, be aware: we still are not getting the solutions of the system (look at the log files in the output folder, *large-3D-systems-schroedinger_3D_nnp_initial.log*). The system is still coarse, and probably we are still very far from the minimum residuals to be reached, for stopping the process. Nevertheless, this behavior of the residuals tells us that we are in the right direction.

3. Making the grid fine in the quantum region

Let us take a look at the wavefunctions in the computations using 20 eigenstates (`$N_states = 20`) shown in Figure 4.17.2.22 for the same sections of Figure 4.17.2.21. It is more convenient to use the results of the output file *\bias_00000\quantum\probabilities_shift_QuantumRegion_Delta3_1d_xz_Si_2DEG.dat* or *\bias_00000\quantum\probabilities_shift_QuantumRegion_Delta3_1d_xz_Si_2DEG.dat* that represent the values of the density probabilities in a section, shifted by the correspondent eigenvalue. From this reason, from this point to the end of this tutorial "wavefunction" actually means shifted probability density. The first observation is that the boundary conditions for the quantum conditions looks being suitable for the band edges in this region. Nevertheless, the grid resolution in the x- and y-directions is actually too coarse, as expected.

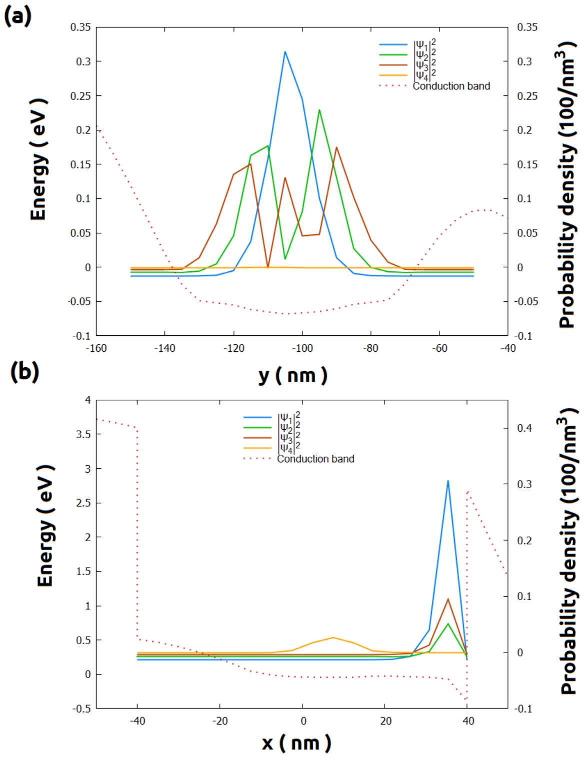


Figure 4.17.2.22: Wavefunctions overlapped to the conduction band from self-consistent quantum-Poisson simulations for the sections (a) `xz_Si_QDs` and (b) `yz_Si_QDs`. The "quantum" density of electrons were computed considering 20 states and a grid resolution of 5 nm in the y-direction. These are intermediate results: the convergence process was stopped after 10 iterations.

To avoid explosion of the number of nodes to be simulated, we suggest modifying the grid definition by introducing new variables for the control of the grid space only within the quantum region. Including the bounds of the quantum region in the grid is also highly recommended. In our example, our previous definition of the grid in x- and y-direction:

```
xgrid{
    line{ pos = $x_4F spacing = $space_x_4F }
    line{ pos = $x_3F spacing = $space_x_Si }
    line{ pos = $x_1F spacing = $space_x_Si }

    line{ pos = $x_1L spacing = $space_x_Si }
    line{ pos = $x_3L spacing = $space_x_Si }
```

(continues on next page)

(continued from previous page)

```
        line{ pos = $x_4L spacing = $space_x_4L }
        line{ pos = $x_5L spacing = $space_x_5L }
    }

    ygrid{
        line{ pos = $y_7S spacing = $space_y_SD }
        line{ pos = $y_6S spacing = $space_y_SD }
        line{ pos = $y_5S spacing = $space_y_LG }
        line{ pos = $y_4S spacing = $space_y_LG }
        line{ pos = $y_1S spacing = $space_y_LG }
        line{ pos = $y_1D spacing = $space_y_LG }
        line{ pos = $y_4D spacing = $space_y_LG }
        line{ pos = $y_5D spacing = $space_y_LG }
        line{ pos = $y_6D spacing = $space_y_SD }
        line{ pos = $y_7D spacing = $space_y_SD }
    }
```

will be changed to

```
    xgrid{
        line{ pos = $x_4F spacing = $space_x_4F }
        line{ pos = $x_3F spacing = $space_x_Si }           # bound of the quantum region
        line{ pos = $x_1F spacing = $space_x_QR }

        line{ pos = $x_1L spacing = $space_x_QR }
        line{ pos = $x_3L spacing = $space_x_QR }           # bound of the quantum region
        line{ pos = $x_4L spacing = $space_x_4L }
        line{ pos = $x_5L spacing = $space_x_5L }
    }

    ygrid{
        line{ pos = $y_7S spacing = $space_y_SD }
        line{ pos = $y_6S spacing = $space_y_SD }

        line{ pos = $yq_min spacing = $space_y_QR }           # bound of the quantum region
        line{ pos = $y_5S   spacing = $space_y_QR }
        line{ pos = $y_4S   spacing = $space_y_QR }
        line{ pos = $y_1S   spacing = $space_y_QR }
        line{ pos = $y_1D   spacing = $space_y_QR }
        line{ pos = $y_4D   spacing = $space_y_QR }
        line{ pos = $y_5D   spacing = $space_y_QR }
        line{ pos = $yq_max spacing = $space_y_QR }           # bound of the quantum region

        line{ pos = $y_6D spacing = $space_y_SD }
        line{ pos = $y_7D spacing = $space_y_SD }
    }
```

where `$space_x_QR` and `$space_y_QR` will control the grid resolution within the quantum region, and `$yq_min` and `$yq_max` are the bounds of this region in the y-direction.

Instead of refining both axes at the same time, let us reduce the grid resolution in the y-direction first. Figure 4.17.2.23 presents the wavefunctions overlapped with the band edges in the section `xz_Si_QDs` for different grid spacing in the y-direction controlled by `$space_y_QR`. We can also observe the corresponding residual evolution in the first 10 iterations. The grid in the x-direction was kept 5 nm.

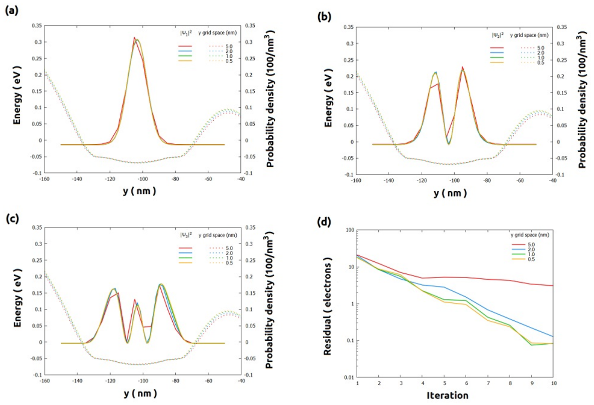


Figure 4.17.2.23: Results of the self-consistent Schrödinger-Poisson simulations using different grid spacing in the y-direction only within the reduced QROI (section `xz_Si_QDs`). (a) -(c) wavefunctions (solid lines) for the three lowest states overlapped with the conduction band (dotted lines) (d) residual evolution. These are intermediate results after only 10 iterations: the convergence process was still not completed.

The evolution of the residuals are very similar, except for the case of 5 nm. Additionally, for `$space_y_QR` of 1 nm or less the wavefunctions are smooth and do not present relevant changes.

Repeating a similar procedure for different grid resolutions in the x-direction (`$space_x_QR`) we obtain the wavefunctions of the Figure 4.17.2.24. In the y-direction the grid resolution in this region was considered equal to 1 nm (`$space_y_QR = 1` in our input file).

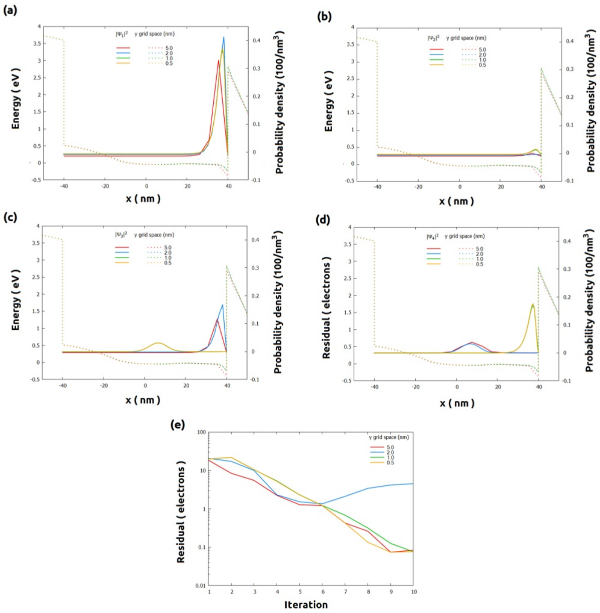


Figure 4.17.2.24: Results of the self-consistent Schrödinger-Poisson simulations for section `yz_Si_QDs` using different grid spacing in the x-direction within the QROI: (a)-(d) wavefunctions (solid lines) for the four lowest states overlapped with the conduction band (dotted lines), and (e) residual evolution. These are intermediate results after only 10 iterations: the convergence process was still not completed. `$space_y_QR` was kept 5 nm.

From analysis of these plots, we observe that decreasing the grid resolution in the x-direction from 1 nm to 0.5 nm does not introduce significant improvement in the computation of the density of probabilities from the wavefunctions in the first 10 iterations. For this reason, we infer that values of 1 nm or less for `$space_x_QR` and `$space_y_QR` will be required for more accurate simulations.

4. Expanding the Quantum Region: time to get beautiful plots (and accurate results)!

Using the results obtained for reduced QROI, we can now design the whole quantum region, now extended from -150 nm to 150 nm for the y-direction. Until now the central valley of the conduction band in Figure 4.17.2.20 (around `y = 0`) was not part of the reduced QROI. Therefore, it may require to increase the number of the eigenvalues in the self-consistent computations in order to fill also this region with carriers.

How to estimate the minimum number of eigenvalues required (`$Nstates`)?

Our hint is to use, once again, our lower resolution grid within the quantum region (for example, with 5 nm) and few iterations (10) for this tuning. We will reserve the finer grid (1 nm) we previously obtained, only to get the final accurate results. Figure 4.17.2.25 illustrates a sequence of simulations where the grid resolution and the number of states were iteratively changed.

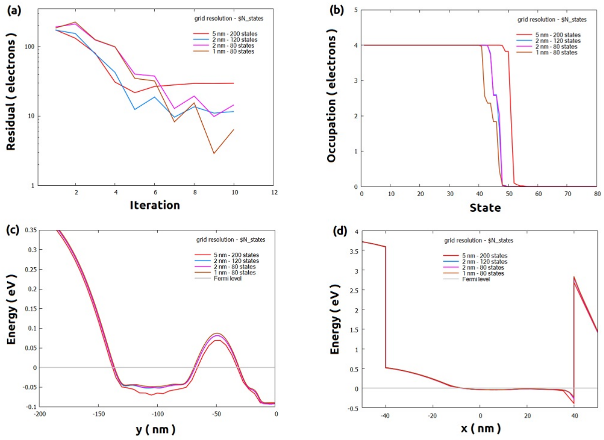


Figure 4.17.2.25: Sequence of simulations for defining a suitable value for `$Nstates`. (a) residual evolution, (b) occupation number of the most populated band, (c) and (d) conduction band for the sections `xz_Si_QDs` and `yz_Si_QDs`, respectively. These are intermediate results (only 10 iterations of the coupled solvers were taken into account).

From the coarser grid we observed that the occupation number in the most populated band it is no more than 80 states. The residual of the density of electrons decreases as the grid gets finer and the number of states is around 80. The conduction band in the more relevant sections, shown in the image, does not change too much, for grid spacing less than 2 nm in the quantum region.

Now it is time to obtain more accurate results. As we mentioned before we will compute 80 states in a quantum grid with `$space_x_QR = 1` and `$space_y_QR = 1`. The convergence process will be controlled by the maximum number of iterations (`$quantum_iterations = 100`) and the accuracy desired (`$CRes`) for the residual of the quantum densities. The solutions converge when the quantum density of electrons is smaller than `$Cres` before ending the total number of iterations of the self-consistent calculations.

We start, for example, with a constraint `$CRes = 1` and, gradually we decrease until the solutions do not change.

Figure 4.17.2.26 shows an example how to choose a suitable value for `$CRes`. This corresponds to simulations that converged in less than 100 iterations. The curves correspond to some wavefunctions for the section `xz_Si_QDs` when requesting accuracy of 0.1, 0.01 and 1.0. We observe that the lowest states (like shown in (a)) are more requires more deep constraints in the value of `$CRes` than the highest states. Nevertheless, decreasing this parameter from 0.10 to 0.01 does not present a significant improvement in the results. What you need to keep in mind is which of both values to use: using `$CRes = 0.01` will produce, in thesis, better results, but it will result in longer runtimes and more memory for storing the results.

The solutions for the section `yz_Si_QDs` are even more robust than the previous one (see Figure 4.17.2.27): the wavefunctions do not present relevant variation even when the value of `$Cres` is higher.

Hint

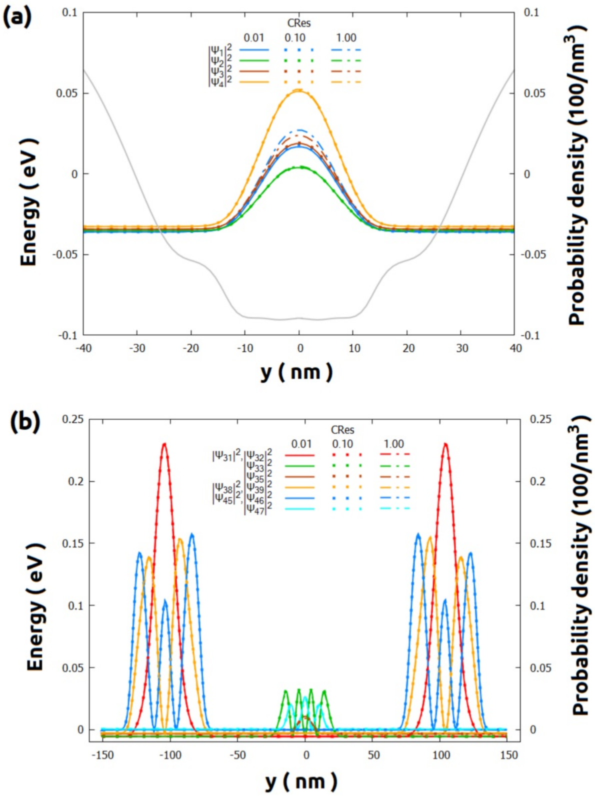


Figure 4.17.2.26: Some wavefunctions in the section `xz_Si_QDs` as function of the residual used in the convergence process `$CRes`. (a) in the central, and (b) in the whole quantum region. All solutions converged before reaching the maximum number of iterations (100).

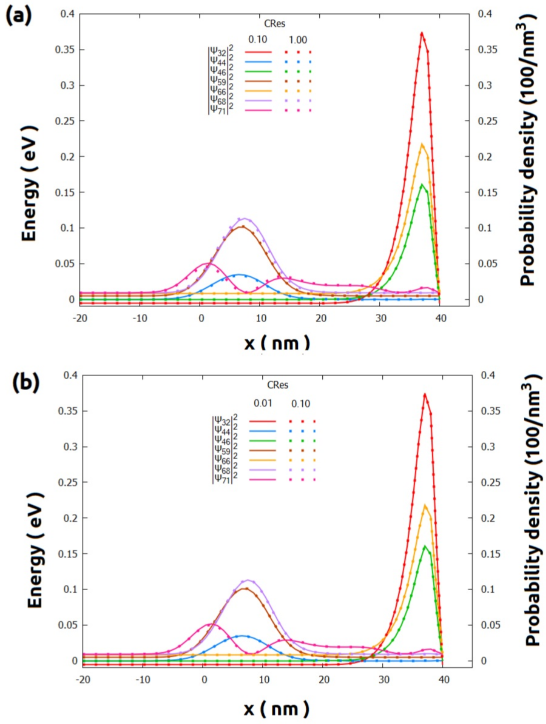


Figure 4.17.2.27: Comparison of some wavefunctions in the section `yz_Si_QDs` for convergence residual (`$Cres`) (a) from 1.00 to 0.10, and (b) from 0.10 to 0.01. All solutions converged before reaching the maximum number of iterations (100).

Requiring higher accuracy of the solutions may result in large runtime, when the decrease of residuals are too slow, or even the process does not converge within the chosen maximum number of iterations. Therefore, it is a good practice tracking the residual evolution during the simulations. If they are taking too long (compared with another previous one) for decreasing, you always can interrupt the calculations pressing the key `F11` or `F12`.

Final considerations

Last but not least, we will simply mention here some important topics are worth to be discussed in separated tutorials.

For some problems that requires really fine grids in very large regions the memory may become the bottleneck of the simulations: the system to be solved may not fit in your RAM. For these situations we implemented in *nextnano*++ the decomposition method, that converts the 3D-Schrödinger-Poisson problem in multiples 1D problems. Additionally, our implementation results very fast. Nevertheless, this algorithm has intrinsic assumptions, that may not apply to all devices and shall be carefully used. For more detail look at in our page *quantum{ region{ quantize_x{ }, … } }* of our documentation.

Nevertheless, this algorithm has intrinsic assumptions, that may not apply to all devices and shall be carefully used.

It is also important to mention that, coherent quantum transport calculations can be performed using the *nextnano*++ implementation of the CBR method. The performance of these computations can be improved implementing small changes in the final input file from this tutorial. The most important modification consists on importing the file with the final result of the electrostatic potential from your self-consistent simulations, instead of being solved directly. Our tutorial *Landauer conductance and conductance quantization: from quantum wires to quantum point contacts* presents this method in detail and the corresponding input files than can be easily extended to 3D devices.

One again, we remind you that in this tutorial we considered only a combination of biases applied to the gates. It is always convenient to check the constraints (boundary conditions for the quantum region, occupation number, residual evolution, etc.) to another scenarios.

We recommend visiting our documentation, where we present the whole methodology *Approaching large 3D designs with Schrödinger-Poisson self-consistent solver* and its first two steps:

> *Reducing dimensionality of large 3D designs*
>
> *Optimizing electrostatics simulation for large 3D designs*

Last update: 15/07/2024

4.18 Tricks and Hacks

This set of tutorials focus on non-standard simulations with *nextnano*++, therefore, on overcoming difficulties and limitations of models and numerics often arising from the general complexity of simulations of semiconductor devices.

This group of tutorials also covers topics related to extracting additional information from the output of *nextnano*++ by post-processing it with *nextnanopy* and Python programming language.

4.18.1 C-V curve calculation for general structures (Post-processing by python)

Header

Files for the tutorial located in *nextnano++\examples\tricks_and_hacks*

- *MIS_CV_1nmSiO2_1D_nnp.in*
- *MIS_CV_3nmSiO2_1D_nnp.in*
- *MIS_CV_3nmSiO2_metal_1D_nnp.in*
- *CapacitanceBySplines_2021_Nov.py*
- *calculate_CV.py*

Important output files:

- *integrated_density_electron.dat*
- *integrated_density_hole.dat*

Introduction

The *nextnano++* tool can calculate many fundamental quantities like potentials, carrier densities, wave functions and so on. By processing the results of *nextnano++* using the calculation tools such as python, we can calculate further advanced characteristics required for some specific devices.

C-V curve is one of the example of such characteristics. This curve is used for the analysis of the devices that could have a depletion region such as metal-insulator-semiconductor, p-n junction, MOSFET and so on.

Specifically, the C-V characteristic is obtained by calculating the capacitance as

$$C = \frac{dQ}{dV}.$$

When the bias sweep and spacial integration are specified in the input file, the electron and hole densities integrated over the region are output in *integrated_density_electron.dat* and *integrated_density_hole.dat* with respect to each bias. The C-V curve can be calculated by taking a derivative of the Q-V curve that is obtained from these data file.

(For the details of bias sweep and spacial integration, please refer to the input file of the tutorial in *Example*.)

In this tutorial we provide python scripts that calculate and plot the C-V curve. They are applied to our MIS tutorial here, but they can also be applied for the general structures that output *integrated_density_electron.dat* and *integrated_density_hole.dat*. The second script uses our post-processing tool *nextnanopy*.

- *Post-processing without nextnanopy*
- *Post-processing with nextnanopy*

Post-processing without nextnanopy

CapacitanceBySplines_2021_Nov.py first calculates the Q-V curve interpolating the total integrated charges obtained from the data files and calculates the C-V curve from that Q-V curve.

Example command:

```
python C:\Users\user.name\Documents\CapacitanceBySplines_2021_Nov\
↪CapacitanceBySplines_2021_Nov.py -o C:\Users\user.name\Documents\nextnano\Output\ -p
```

The commandline options are followings:

- `-o` : Path of the output folder where *integrated_density_hole.dat* and *integrated_density_electron.dat* are stored follows. (required)
- `-p` : if present in the command line, the total integrated charge and interpolated C-V curves will be plotted using Matplotlib (optional)
- `-b1` : Substring of the contact that will be used as reference follows. When not specified the first common contact of both integrated_density files will be used. (optional)
- `-b2` : Substring of a second contact that will be used as reference follows. The final C-V will be calculated as function of the voltage given by bias1 - bias 2. If bias1 was not specified, bias2 will be ignored. (optional)

Example

Here we have a MIS tutorial: "Capacitance-Voltage curve of a "metal"-insulator-semiconductor (MIS) structure".

After running the *nextnano++* input file of this tutorial *MIS_CV_1nmSiO2_1D_nnp.in*, we can find *integrated_density_electron/hole.dat* in the output folder.

By executing *CapacitanceBySplines_2021_Nov.py* in the following command,

```
python C:\Users\user.name\Documents\CapacitanceBySplines_2021_Nov\
↪CapacitanceBySplines_2021_Nov.py -o C:\Users\user.name\Documents\nextnano\Output\
↪MIS_CV_1nmSiO2_1D_nnp\ -p
```

we get the Q-V curve and C-V curve as follows.

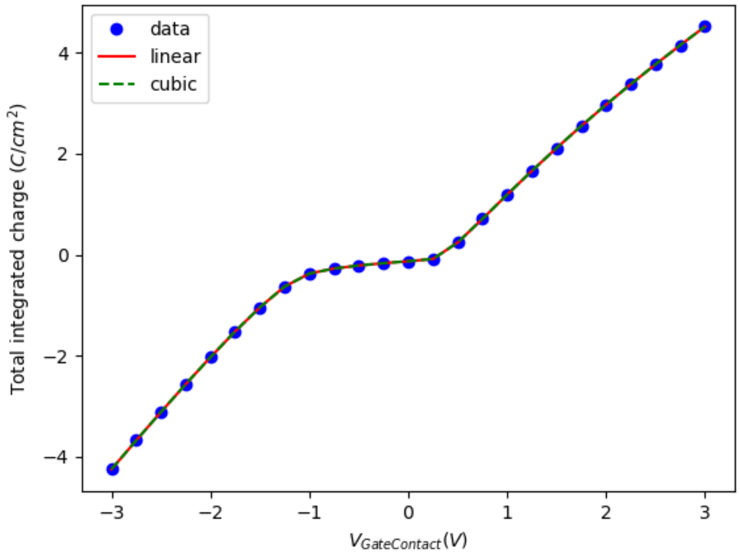


Figure 4.18.1.1: Q-V characteristics obtained by post-processing the result of *MIS_CV_1nmSiO2_1D_nnp.in* by *CapacitanceBySplines_2021_Nov.py*. Linear and cubic interpolation are done to the output data.

Post-processing with nextnanopy

In order to use the CV calculation with *nextnanopy*, import the CV calculation function from postprocess modul.

```
from nextnanopy.postprocess import CV_calculation
```

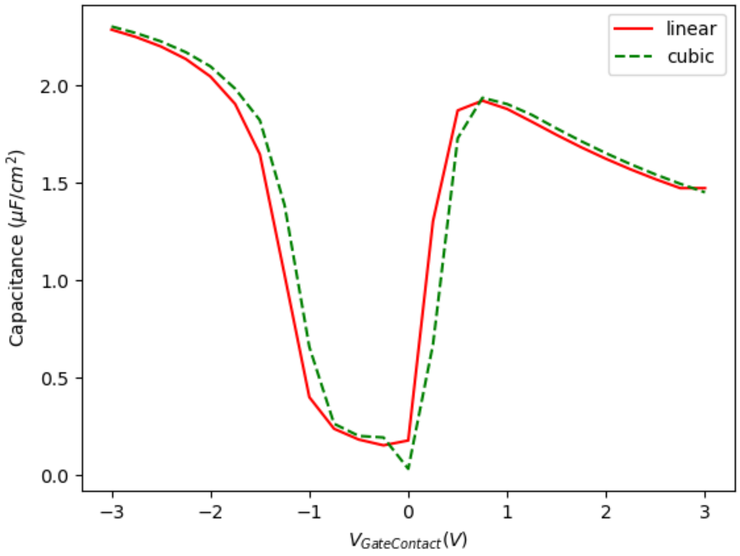


Figure 4.18.1.2: C-V characteristics obtained by post-processing the result of *MIS_CV_1nmSiO2_1D_nnp.in* by *CapacitanceBySplines_2021_Nov.py*.

`nextnanopy.postprocess.(output_directory_path, bias1 = None, bias2 = None, total = False, net_charge_sign = -1) -> voltage, C_regions`

Calculates the CV curve of the device. The voltage is defined based on the following criteria:

- If the values for bias1 and bias2 are not given, the voltage is set to the value of the first bias column in the densities file.
- If the value for bias1 is given and bias2 is not given, the voltage is set to the value of bias1.
- If both bias1 and bias2 are given, the voltage is set to the difference between bias2 and bias1.

Parameters

- **`output_directory_path`** (*`str`*) – output directory path of the simulation
- **`bias1`** (*`str`*) – name of bias1
- **`bias2`** (*`str`*) – name of bias2
- **`total`** (*`bool`*) – if True, capacitance is calculated for total charge

Returns

voltage(numpy array) and capacitances (list of capacities for each computed region)

To calculate the capacitance vs voltage using linear interpolation, use this function as following

```
voltage, C_regions = calculate_CV(output_directory_path)
```

To plot the output it is recommended to use

```
import matplotlib.pyplot as plt
for region in C_regions:
    plt.plot(voltage, region)
```

The example, which runs the simulation and plots the CV curve with *nextnanopy* can be found here: Python template to run CV calculation

---

Last update: 17/07/2024

4.18.2 Interband tunneling current in a highly-doped nitride heterojunction

Header

Files for the tutorial located in *nextnano++\examples\tricks_and_hacks*

- *InterbandTunneling_Duboz2019_nnp.py*
- *InterbandTunneling_Duboz2019_nnp.in*

Important output files:

- *bias_xxxxx/integrated_density_electron.dat*
- *bias_xxxxx/integrated_density_hole.dat*
- *bias_xxxxx/mobility_electron.dat*

Introduction

We compute interband tunneling current through a highly-doped heterojunction by *nextnano*++ simulation and Python post-processing. We follow the methods in the following publication of Jean-Yves Duboz and Borge Vinter *[Duboz2019]*, using fewer approximations wherever possible:

This tutorial uses the Python script *nextnanopy/templates/InterbandTunneling_Duboz2019_nnp.py* to automate the simulation of the *nextnano*++ input file *InterbandTunneling_Duboz2019_nnp.in* and post-calculation of interband tunneling current.

The script

The Python script does the following while sweeping the bias:

1. Runs the *nextnano*++ simulations based on the user-defined parameters
2. From the simulation output folder, load the envelopes $F_{\mathrm{vj,z1}}(z)$, $F_{\mathrm{vj,z2}}(z)$, and $F_{\mathrm{ci}}(z)$ together with the electrostatic potential $\phi(z)$. The units are 1/nm$^{1/2}$ and V, respectively.
3. Differentiates the potential.
4. Calculates the dipole matrix elements using the position-dependent material parameters.
5. Plots the matrix elements as a function of position.
6. Integrates the product over the device.
7. Calculates tunneling current density for individual transitions in units A/cm^2.
8. Sums up the tunnel current density for all possible transitions.

After all simulations and post-calculations, the Python script exports the tunnel I-V curve in the following formats:

1. Image file with the format specified by the user

2. *.dat file

The output folders are indicated in the console log. The *.dat format is useful if you compare I-V curves using the *nextnanomat* overlay feature.

Options in the script

Effective ffield

If the Boolean variable `CalculateEffectiveField_fromOutput = True` *(the default)*, then the script calculates the position-dependent effective field

$$M_{ij}^{\sigma} = \alpha_{Z\sigma}^{j*} \int \frac{P_1}{E_g} F_{vj,z\sigma}^{*}(z) F_{ci\sigma}(z) q \frac{\partial \phi(z)}{\partial z} dz$$

based on the computed electrostatic potential. When `CalculateEffectiveField_fromOutput = False`, the assumption in the paper is used.

$$\frac{\partial \phi(z)}{\partial z} = 1 \frac{\mathrm{V}}{\mathrm{nm}}$$

Kane's parameter

If the Boolean variable `KaneParameter_fromOutput = True` *(the default)*, then the script reads in the Kane's parameter P in from the *nextnano*++ output to evaluate

$$\langle Z|\, z \,|S\rangle = \frac{1}{E_g} \langle Z|\, p_z \,|S\rangle = \frac{P}{E_g}$$

In this case, an 8-band $\mathbf{k} \cdot \mathbf{p}$ simulation with exactly the same device geometry will be performed so that *nextnanopy* can extract the Kane parameter.

If `KaneParameter_fromOutput = False`, then P is calculated from the assumption in *[Duboz2019]* (E_P = 15 eV).

Reduced mass

If the Boolean variable `CalculateReducedMass_fromOutput = True`, then the script calculates the position-dependent reduced mass m_r in

$$I_{ij} = \frac{2\pi q}{\hbar} \sum_{\sigma} |M_{ij}^{\sigma}|^2 \cdot \frac{m_r}{2\pi \hbar^2} = \frac{q m_r}{\hbar^3} \sum_{\sigma} |M_{ij}^{\sigma}|^2$$

using the *nextnano*++ outputs of the effective masses.

When `CalculateReducedMass_fromOutput = False` *(the default)*, then the assumption as in *[Duboz2019]* is used.

Results

The structure is an AlGaN/GaN p-i-n junction with 2 nm GaN interlayer.

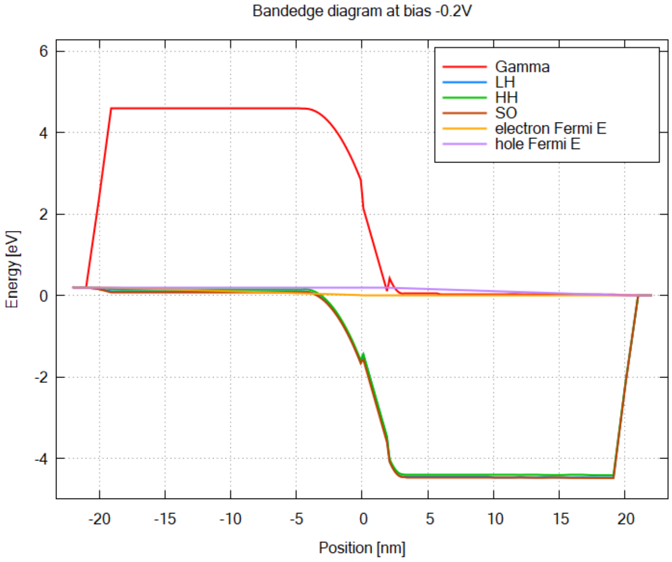
The energy overlap between the hole states and electron states increases as the bias, leading to larger tunnel current.

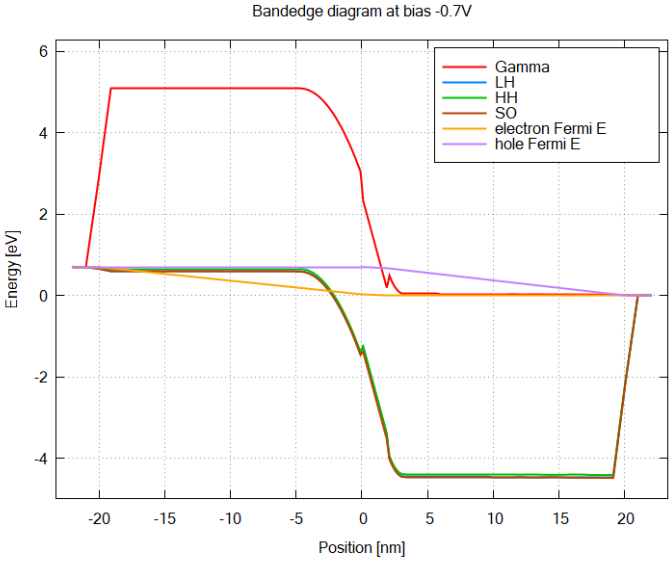
The Python script calculates dipole matrix elements from the simulation results:

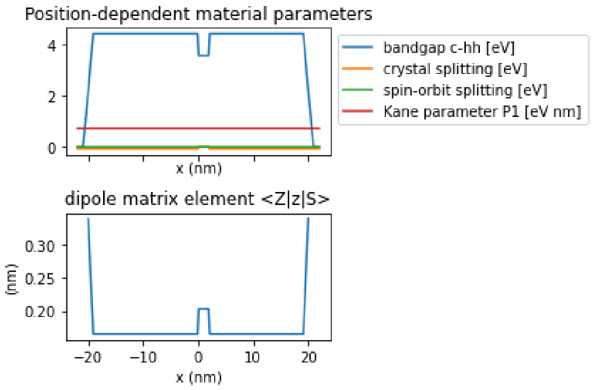
from which we obtain the tunnel current as a function of bias:

Last update: 17/07/2024

4.18.3 Optical generation in InGaAs/GaAs QW


Bandedge diagram at bias -0.2V
Energy [eV]
6
4
2
0
-2
-4
-20
-15
-10
-5
0
5
10
15
20
Position [nm]
Gamma
LH
HH
SO
electron Fermi E
hole Fermi E


Bandedge diagram at bias -0.7V
Energy [eV]
6
4
2
0
-2
-4
-20
-15
-10
-5
0
5
10
15
20
Position [nm]
Gamma
LH
HH
SO
electron Fermi E
hole Fermi E


Position-dependent material parameters
4
2
0
x (nm)
bandgap c-hh [eV]
crystal splitting [eV]
spin-orbit splitting [eV]
Kane parameter P1 [eV nm]
dipole matrix element <Z|z|S>
(nm)
0.30
0.25
0.20
−20
−10
0
10
20
x (nm)

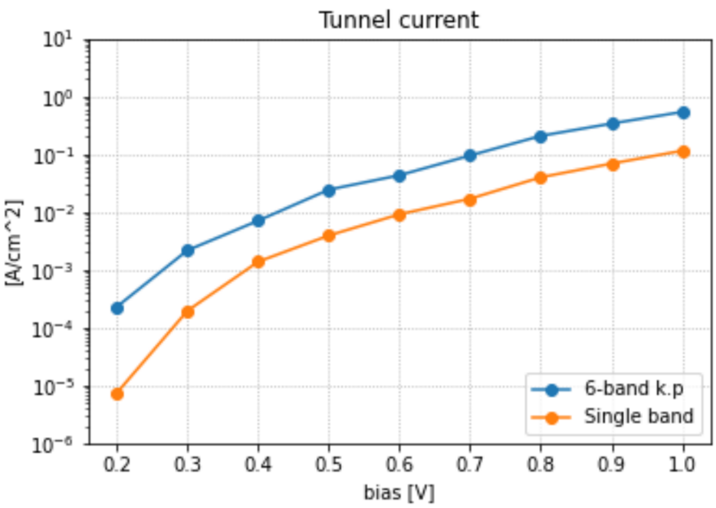


Figure 4.18.2.1: Interband tunneling current in a nitride p-i-n junction. Following the paper, backward bias is taken to be positive in this plot.

Header

Files for the tutorial located in *nextnano++\examples\tricks_and_hacks*

- *1D_optical_generation_ingas_gaas_qw.in*
- *1D_optical_generation_ingas_gaas_2qw.in*
- *1D_optical_generation_ingas_gaas_qw.py*
- *1D_optical_generation_ingas_gaas_2qw.py*

Scope:

In this tutorial, a procedure for simulating photogeneration inside quantum wells is described.

Important keywords:

- `optics{ irradiation{} quantum_spectra{} }`
- `import{ }`
- `region{ generation{} }`

Relevant output files:

- *bias_00000\Optics\absorption_quantum_region_TE_eV.dat*
- *Irradiation\illumination_spectrum_power_eV.dat*
- *bias_00000\recombination.dat*
- *bias_00000\bandedges.dat*

Introduction

We consider a simple 1D single QW (In$_{0.2}$Ga$_{0.8}$As/ GaAs) structure under illumination along the QW growth direction. The photon energy is little above the absorption edge of the GaAs QW. The In$_{0.2}$Ga$_{0.8}$barriers are transparent for the incident photons, because the band gap in these regions exceeds the energy of the photons. Thus generation of charge carriers only occurs inside the QW.

Simulation Scheme

Based on the current implementation of photogeneration in *nextnano*++, the simulation procedure shown in Figure 4.18.3.1 is employed.

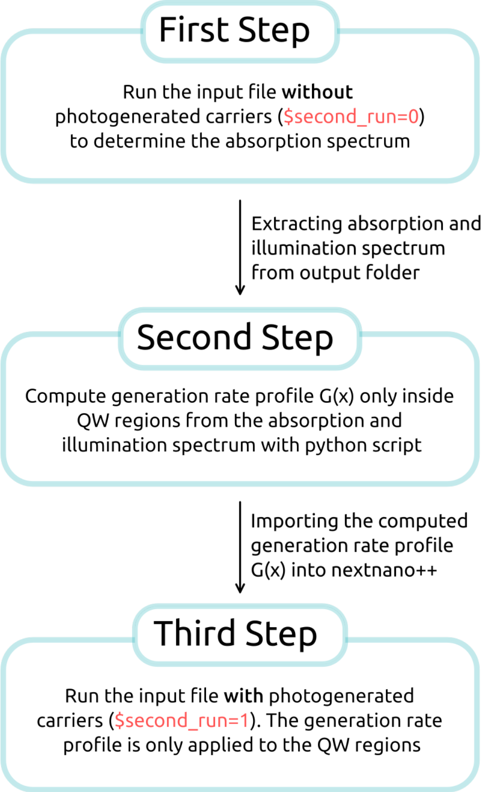


Figure 4.18.3.1: Visualization of the Simulation Procedure

Each step of the procedure is further elaborated in the sections below.

First Step

In the first step, data files for the absorption spectrum and the illumination spectrum are created, which are going to be used to determine the generation profile $G(x)$, in a later step.

Before running the input file, the user should specify the properties of the light source inside the group `optics{ irradiation{} }`.

```
optics{
    irradiation{
        min_energy = 1.0            # minimum energy of the light source spectrum
        max_energy = 1.8            # maximum energy of the light source spectrum
        energy_resolution = 0.001   # resolution of the light source spectrum

        global_illumination{
            direction_x=1
            gaussian_spectrum{      # lineshape
```

(continues on next page)

(continued from previous page)

```
                irradiance = 1e5    # total intensity [W/m^2]
                energy = 1.25       # peak energy [eV]
                gamma = 0.01        # FWHM [eV]
            }
        }
        output_spectra{}            # create light source spectrum in the output␣
↪folder
    }

    ...

}
```

When running the input file, *nextnano++* computes the absorption spectrum quantum mechanically based on the settings inside `optics{ quantum_spectra{} }`.

```
optics{

    ...

    quantum_spectra{
        name = "quantum_region"
        polarization{ name = "TE" re = [0,1,0] }
        k_integration{
            relative_size = 0.2
            num_subpoints = 6
            num_points = 8
        }
        output_spectra{
            spectra_over_energy = yes       # output spectrum dI/dE
            emission = yes
        }
        output_occupations = yes

        energy_broadening_lorentzian= 1.0e-2
        spontaneous_emission = yes

        # Note: the following settings should be the same as in irradiation{}
        energy_min  = 1.0                   # minimum energy of the absorption␣
↪spectrum
        energy_max  = 1.8                   # maximum energy of the absorption␣
↪spectrum
        energy_resolution = 0.001           # resolution of the absorption spectrum
    }
}
```

The computed absorption and illumination spectra are located in the output folder at:

- *<input file name>\bias_00000\Optics\absorption_quantum_region_TE_eV.dat*
- *<input file name>\Irradiation\illumination_spectrum_power_eV.dat*

Warning

Depending on the settings in *nextnanomat*, <input file name> could contain, in addition to the actual input file name, the current date or a counting index if the input file is run several times. It has to be checked that the path name of the simulation results is consistent with the path name which is used later in the python script when extracting the files.

Second Step

With the python script, the generation rate profile $G(x)$ is calculated as follows:

$$G(x) = \int G(x, E)dE,$$

where $G(x, E)$ is given by

$$G(x, E) = \alpha(E) \cdot \phi(E)e^{-\alpha(E)x},$$

with the spectral photon flux $\Phi(E)$ and absorption coefficient $\alpha(E)$. Reflectance is neglected in this case. The factor $\phi(E)e^{-\alpha(E)x}$ represents the light field which attenuates exponentially along the propagation direction.

The spectral photon flux is determined by the spectral properties of the light source, i.e. the light source spectrum dI/dE, as follows:

$$\Phi(E) = \frac{dI/dE}{E}$$

with energy E. For $\alpha(E)$ the absorption spectrum which was computed in the first step is rescaled by a factor f. This scaling factor is necessary, because the absorption spectrum, as computed by the tool, scales with the chosen quantum region L_q and well width L_w, c.f. Figure 4.18.3.2 (left). Multiplying the absorption spectra by

$$f = \frac{L_w}{L_q}$$

compensates the dependency on L_q around the absorption edge of the QW, which lies around 1.225eV in the case of GaAs-QW, as shown in Figure 4.18.3.2 (right).

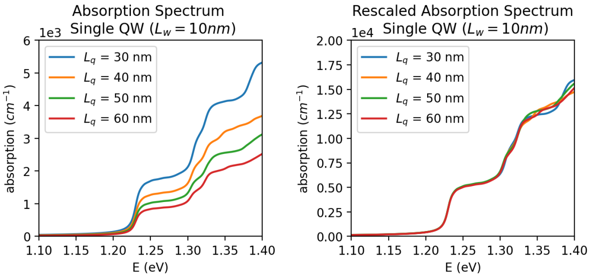


Figure 4.18.3.2: Computed absorption spectrum of a single InGaAs/GaAs quantum well for different quantum region widths L_q, unscaled (left) and rescaled by the factor L_w/L_q (right)

The rescaling factor for multiple quantum well structures becomes:

$$f = \frac{L_q}{\sum_i L_w^{(i)}}.$$

Third Step

The generation rate profile can now be imported from the data file. The file should contain values for position and generation rate as separate columns.

```
import{
    file{
        name = "my_generation_profile"               # reference name
        filename = "Generation_profile.dat"          # relative path to generation␣
```

(continues on next page)

(continued from previous page)

```
↪rate profile
        format = DAT
    }
}
```

The imported generation profile is then applied to the QW region:

```
structure{

    ...

    region{
        ternary_constant{ name = "Ga(1-x)In(x)As" alloy_x = 0.2 }  # material GaInAs␣
↪alloy
        line{ x = [ $well_start, $well_end  ] }                    # overwriting␣
↪previously defined GaAs

    !IF($second_run)
        generation{                                                 # generation␣
↪profile G(x) applied to QW region
            import{ import_from = "my_generation_profile" }         # refer to␣
↪imported data file with name "my_generation_profile"
        }
    !ENDIF
    }

    ...

}
```

Results

Generation Rate Profile

Figure 4.18.3.3 shows the generation rate profile calculated according to the above described methodology.

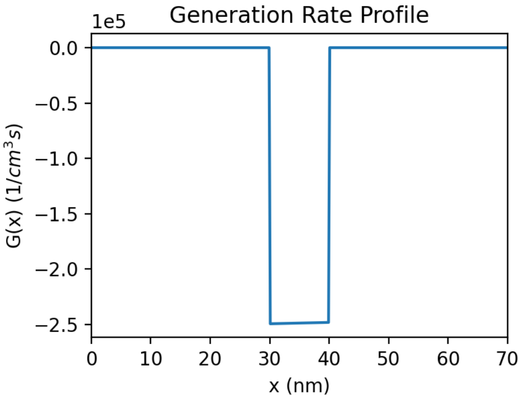


Figure 4.18.3.3: Computed generation rate profile G(x) for single QW structure

Bandedges and Fermi Levels

Figure 4.18.3.4 compares the band edges and Fermi levels without photogeneration (left) and with photogeneration based on the imported generation rate profile (right).

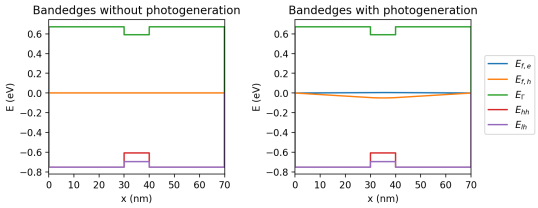


Figure 4.18.3.4: Band edge profile of 1D QW (L_w = 10 nm) structure without photogeneration (left) and with photogeneration (right)

Last update: 17/07/2024

4.18.4 Photoluminescence of Quantum Well

Header

Files for the tutorial located in *nextnano++\examples\tricks_and_hacks*

- *1D_PL_of_QW_absorption_nnp.in*
- *1D_PL_of_QW_nnp.in*
- *1D_PL_of_QW_nnp_absorption_spectrum.dat*

Output Files:
bias_00000\Optics\spont_emission_power_region_longitudinal_nm.dat

Scope:
In this tutorial, we show an approach how to model photoluminescence (PL) in 1D QW structures. The following is covered:

- Short overview of the most essential groups which are needed in the input file for PL simulations
- How to compute the absorption spectrum, when no experimental data is available
- Results: photoluminescence spectra
- Limitations of the simulation

Important keywords:

- `classical{ energy_resolved_density{} energy_distribution{} }`
- `optics{ irradiation{} semiclassical_spectra{} quantum_spectra{} }`
- `quantum`, `current{}`

Introduction

What are we modelling? In short, light impinges on the surface of the structure parallel to the growth direction. A certain fraction of the total photon flux penetrates into the material and is absorbed, which leads to generation of mobile charge carriers (*photogeneration*). These carriers are lifted into excited states. If an excited electron recombines radiatively with a hole, light is emitted (*spontaneous emission*). As depicted in Figure 4.18.4.1, the recombination process happens mainly in the quantum well.

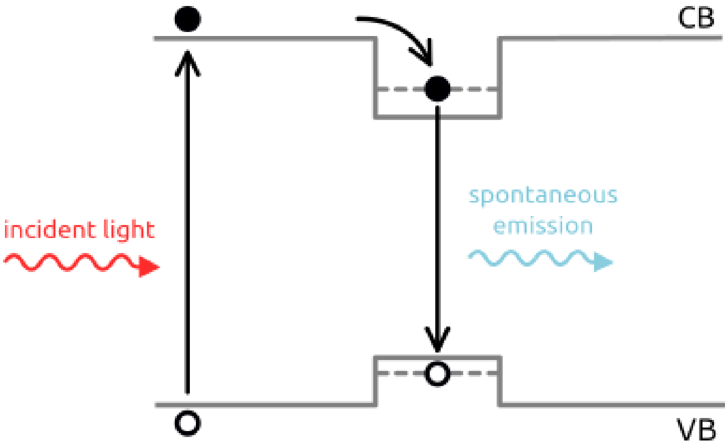


Figure 4.18.4.1: Visualization of involved processes: 1) light absorption and generation of electron - hole pairs, 2) trapping of carriers inside the quantum well, 3) recombination and spontaneous emission of light

The quantum well structure under consideration in this tutorial consists of the following material layers:

Layer	Material	Thickness (nm)
1	$Al_{0.36}Ga_{0.64}$ As	500
2	GaAs	7
3	$Al_{0.36}Ga_{0.64}$ As	500
Substrate	GaAs	1000

Simulation scheme

In our model we treat the absorption and generation of charge carriers within a semiclassical approach. The current equation is calculated self-consistently within the Schrödinger and Poisson equations in order to get accurate charge carrier densities. Afterwards, the luminescence spectra are calculated quantum mechanically based on the occupied states.

General approach

One of the most important process in our simulation is the generation of charge carriers, which is governed by the generation rate $G(x, E)$. The dependency on energy is described by the absorption spectra $\alpha(E)$. Since we assume not having experimental data for the absorption spectra available, we have to calculate $\alpha(E)$.

Figure 4.18.4.2 visualizes the idea of our procedure. The **1. step** is running the input file *1D_PL_of_QW_absorption_nnp.in*, which does not include any optical phenomena (photogeneration, emission, …). Then the **2. step** is to run the normal input file *1D_PL_of_QW_nnp.in*, which includes generation of carriers, using the imported absorption spectrum from the **1. step**. Normally, one has to repeat the whole cycle, until the absorption spectra fully converge. For simplicity, we assume that no additional repetition is needed.

Input file

The optical phenomena related to the irradiation, absorption and spontaneous emission processes, which should be taken into account in the simulation, have to be specified in the `optics{ }` block. The absorption process is modelled within a semiclassical approach calling `irradiation{}` and `semiclassical_spectra{}`. The spontaneous emission is treated quantum mechanically inside the block `quantum_spectra{}`:

```
optics{

    irradiation{

        global_illumination{ # specification of the light source, i.e. illumination␣
↪spectrum
            direction_x=1
```

(continues on next page)

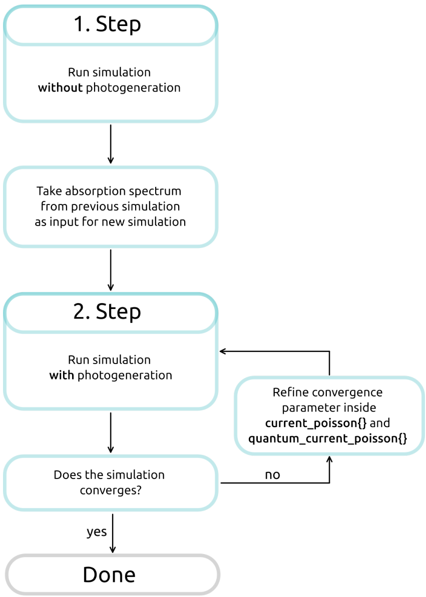


Figure 4.18.4.2: Iterative procedure calculating absorption spectrum until convergences is reached

(continued from previous page)

```
            gaussian_spectrum{
                irradiance = $irradiance*1e4
                wavelength = $peak_wavelength
                gamma = 0.01
        }

        global_absorption_coeff{ # specification of absorption spectrum
            import_spectrum{# choice of imported file with previously calculated abs␣
↪spectra
                import_from = "my_abs"
                cutoff = yes
            }
        }

        photo_generation{ # enabling photogeneration
            output_energy_resolved = yes
        }

        output_spectra{ # output options
            illumination = yes
            absorption = yes
        }
    }

    semiclassical_spectra{ # important group for absorption spectrum
        refractive_index = 3.14768486
        output_spectra{
            absorption = yes
            emission = yes
    }

    quantum_spectra{ # calculate emission spectrum quantum mechanically for the␣
↪quantum region
```

(continues on next page)

(continued from previous page)

```
        name = "quantum_region"

        intraband = no
        interband = yes
        polarization{ name = "longitudinal" re = [1,0,0] }

        k_integration{
            relative_size = 0.3
            num_subpoints = 5
            num_points = 10
            #force_k0_subspace = yes
        }

        output_spectra{
            spectra_over_energy = yes
            spectra_over_wavelength = yes
            emission = yes
            power_spectra = yes
        }

        # settings for output spectra
        energy_min  = 0.001
        energy_max  = 5.0
        energy_resolution = 0.001
        spontaneous_emission = yes
        energy_broadening_lorentzian= 1.0e-2

    }
}
```

The absorption spectrum used in the group `irradiation{}` should be imported from *1D_PL_of_QW_absorption_nnp.in*.

```
import{
    directory = "...1D_PL_of_QW_absorption_nnp\bias_00000\Optical\" # location of the␣
↪file with absorption spectrum - it should be changed accordingly
    file{
        name = "my_abs"                                 # rename filename
        filename = "computed_absorption_spectrum_nm.dat"    # reference desired␣
↪filename
        format = DAT
    }
}
```

Inside the group `classical{ }` one has to specify **energy resolved densities** n(x,E) and p(x,E), which are required for the semiclassical absorption and emission spectra. More information on the underlying equations can be found *here*

```
classical{

    Gamma{} HH{} LH{} # bands involved in 1 band calculation

    energy_resolved_density{
        # calculate position and energy resolved electron and hole densities: n(x,E),␣
↪p(x,E)
        # required for calculation of semiclassical emission and absorption spectra
        min = -5.0
```

(continues on next page)

(continued from previous page)

```
        max = 5.0
        energy_resolution = 0.001
        only_density_quantum_regions = no
    }

    energy_distribution{
        # settings for energy resolved density
        min = -4.0
        max = 4.0
        energy_resolution = 0.001
        only_density_quantum_regions = no
    }

}
```

To calculate the quantum mechanical emission spectra, one has to include the group **quantum{ }**. The group **quantum{ }** as well as **current{}** and **poisson{ }** are also required for self-consistent quantum-current-poisson calculations. Inside these group proper convergence parameters have to be chosen. In this part, one has to think about proper convergence parameters for the solvers.

```
poisson{ ... }
currents{ ... }
quantum{ ... }
```

Note that proper boundary conditions are needed for Poisson and current equation. These are imposed by contact regions. In our simulation, we apply `ohmic{}` contacts only to the bottom of the substrate, i.e. to the not illuminated side of the structure.

```
contacts{
    ohmic{ name = "whatever" bias = 0.0 }
}
```

Simulation

In the simulation a light source with Gaussian spectrum with central wavelength λ_{peak} = 530 nm (2.34 ev) and linewidth of 10 meV is used. The intensity $\Phi_{\text{intensity}}$ is varied between the two values $0.5 \cdot 10^4$ W/cm^2 and $0.05 \cdot 10^4$ W/cm^2. The temperature in this simulation is swept between the three values 200 K, 250 K and 300 K.

Results

First, we have to calculate suitable absorption spectra with the input file *1D_PL_of_QW_absorption_nnp.in*. Figure 4.18.4.3 shows the calculated absorption spectrum at each temperature. For all temperatures, the absorption coefficient at λ = 530 nm is of the order of 10^6.

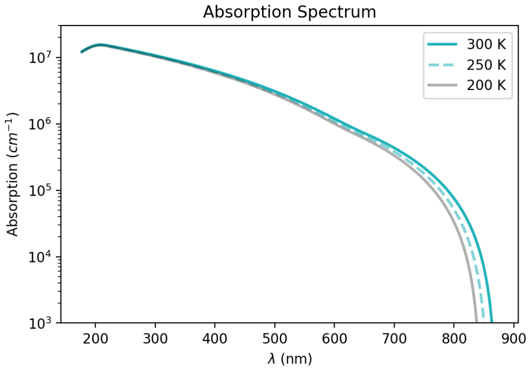


Figure 4.18.4.3: Calculated absorption spectrum

Now we can run the main input file *1D_PL_of_QW_nnp.in*, which imports and uses the computed absorption spectrum.

1) **Band edges**

Figure 4.18.4.4 shows the energy profiles with electron- and hole-Fermi levels. It is visible that boundary conditions (contacts) are only imposed on the right side of the structure. This set up was found to have better convergence behavior.

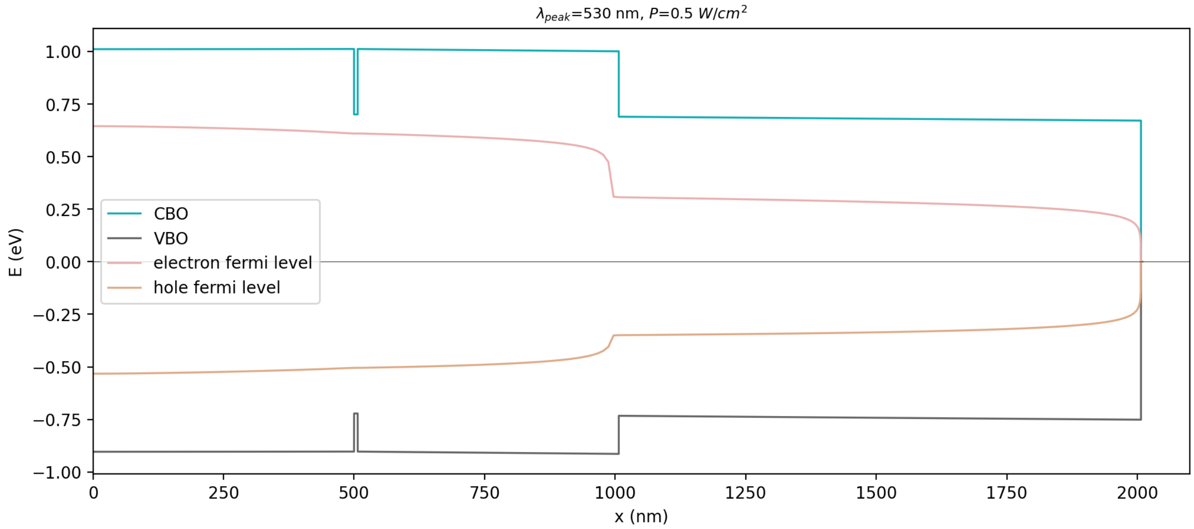


Figure 4.18.4.4: Energy profiles of conduction band (CBO) and valence band (VBO), with electron- and hole-Fermi levels across the structure

2) **Electron/ hole density**

Figure 4.18.4.5 illustrates the spatial and energy distribution of electrons and holes with respect to the band edges for case: $P_{\text{illumination}}$ $0.5 \cdot 10^4$ W/cm^2 at 300 K. Both, electrons and holes, are localized inside the quantum well, thus exhibit discrete energy levels. The occupation of the energy levels gives us insight about possible transitions (*recombination*) between electron states in the conduction band and hole states in the valence band. From Figure 4.18.4.5 we can deduce that most transition energies are in the interval 1.4eV-1.6eV of magnitude. For the emission spectrum, we assume to find its peak energy in this energy interval.

3) **Photogeneration**

Figure 4.18.4.6 depicts the spatial and energy resolved generation rate inside the structure for the case: $P_{\text{illumination}}$ $0.5 \cdot 10^4$ W/cm^2 at 300 K.

4) **Spontaneous emission spectrum**

Figure 4.18.4.7 shows the normalized spontaneous emission spectra at different temperatures. The peak of the emission spectra are primarily attributed to the E_{e1} - E_{h1} transition inside the quantum well. Due to band gap shrinking the peak shifts to higher wavelengths with increasing temperatures. At higher temperatures the contribution from other transitions, such as E_{e2} - E_{h1} to the spectra becomes visible. Thus, the spectrum exhibit a broadening.

5) **Temperature dependence of emitted intensity**

Last update: 17/07/2024

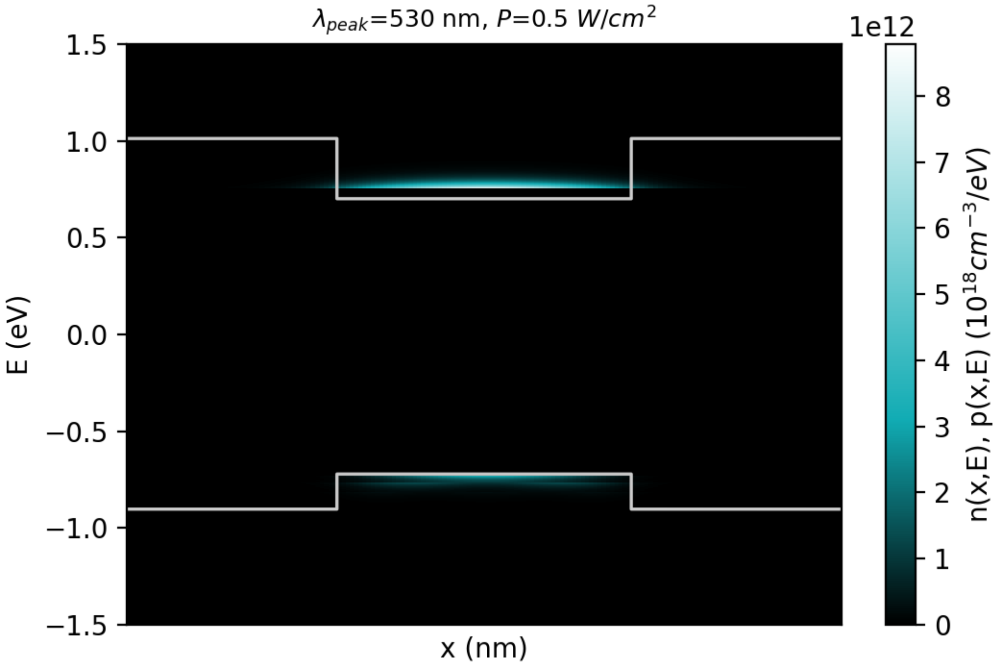


Figure 4.18.4.5: Electron density n(x,E) and hole density p(x,E) with conduction and valence band edges at 300K.

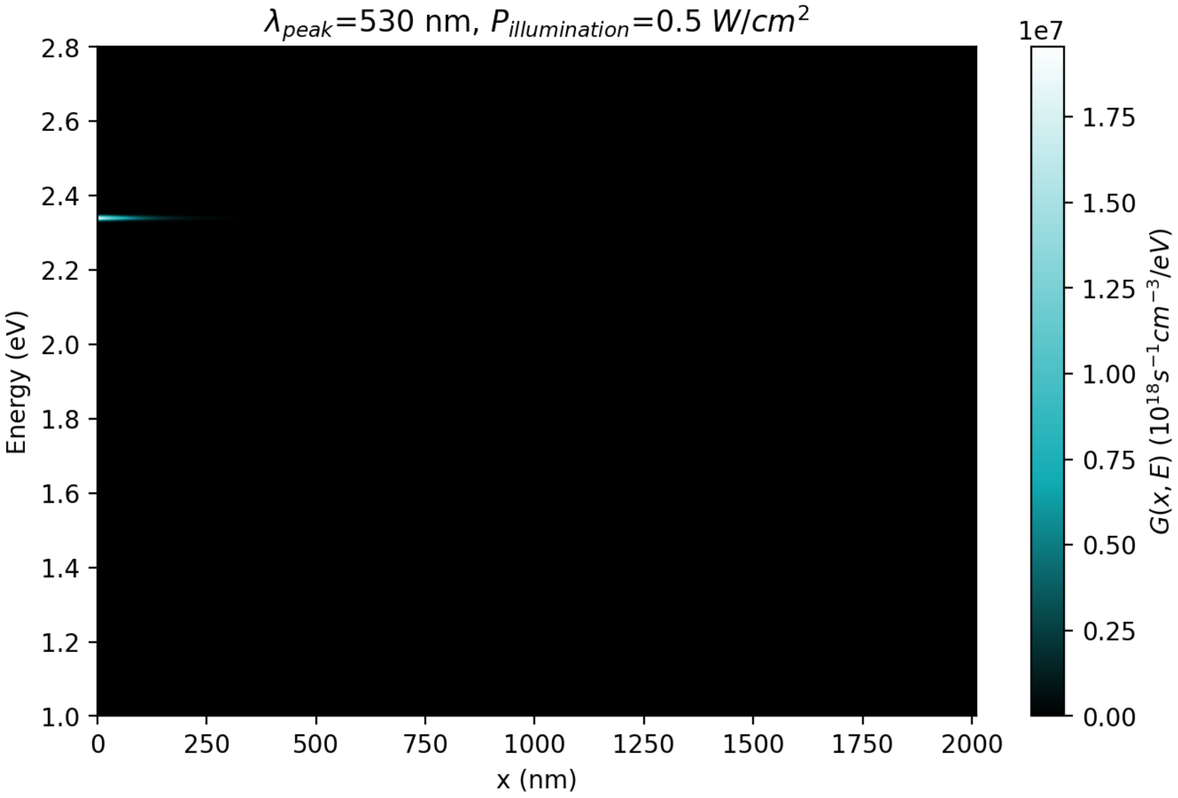


Figure 4.18.4.6: Photogeneration rate $G(x, E)$ at $T = 300K$ and $P = 0.5 \cdot 10^{\ 4}$ W/cm^2

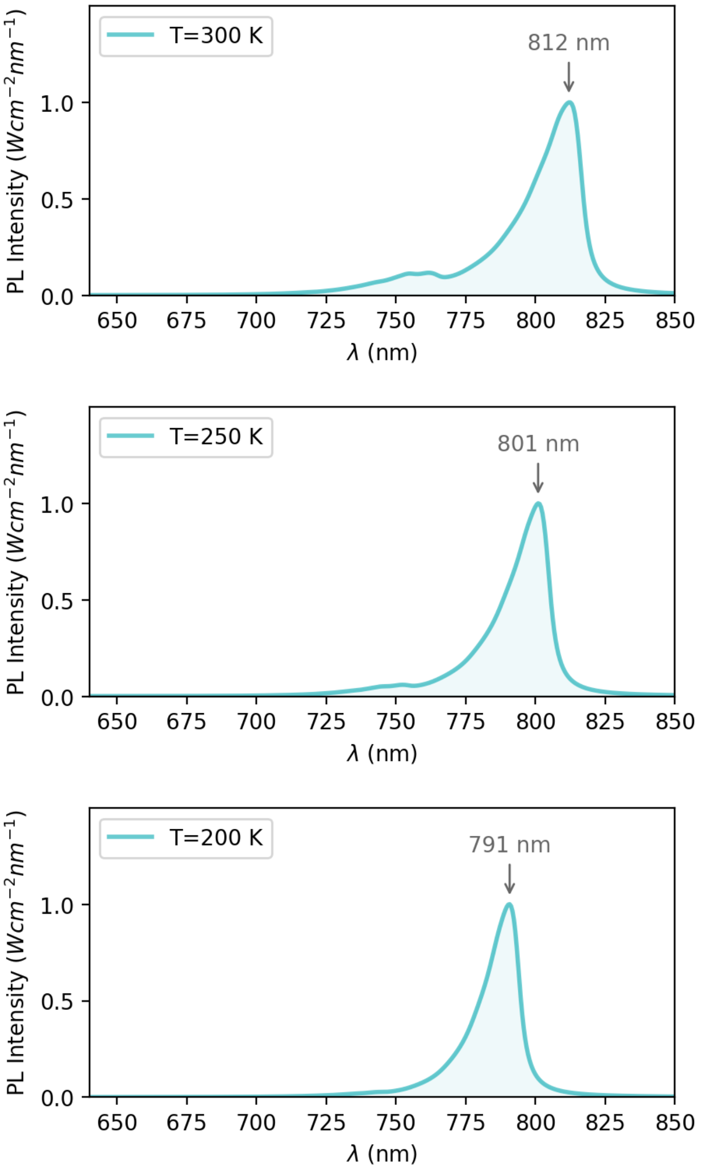


Figure 4.18.4.7: Normalized luminescence spectra with highlighted peak wavelength at each temperature (200 K, 250 K and 300 K), when illuminated by P = $0.5 \cdot 10^4$ W/cm^2.

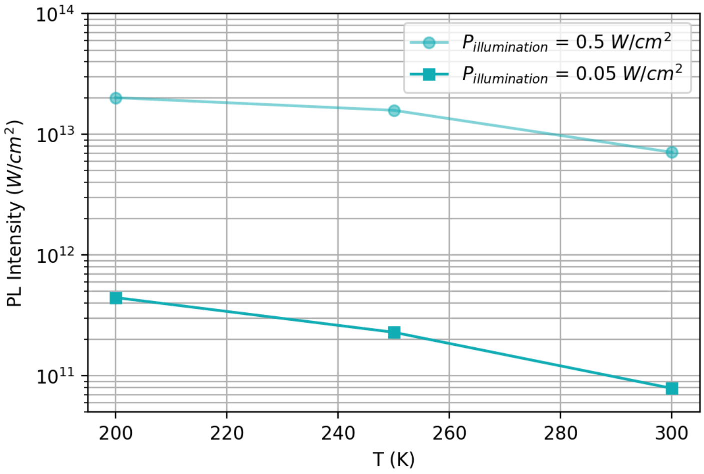


Figure 4.18.4.8: Total emitted intensity as a function of temperature.

4.18.5 From GDSII to Transmission Workflow

Header

Files for the tutorial located in *nextnano++\examples\tricks_and_hacks*

- *3D_GDS_workflow_template_nnp.in* - template for 3D Simulations without gates
- *3D_GDS_workflow_gdsfile.gds* - GDS file
- *2D_GDS_workflow_transmission_in_2DEG_nnp.in* - 2D Simulations
- *3D_GDS_workflow_script.py* - script for importing GDS

Scope of the tutorial:

- Importing layout of gates to a *nextnano*++ 3D input file
- Generating an input file for 3D simulation for a certain bias
- Importing a slice of the potential in the 2DEG to a 2D input file
- Computing the transmission between two leads.

Required python packages

- *nextnanopy*
- gdspy
- shapely

Relevant Keywords (to be updated):

- `structure{ }`
- `quantum{ quantize_x{}}`

Important output Files:

- *input files\3D_GDS_input_file_npp.in*
- *simulations\3D_GDS_Workflow_Results_V_-1.03_npp.in*
- *outputs\3D_GDS_Workflow_Results_V_-1.03_npp\bias_00000\potential_2d_2DEG.fld*
- *outputs\2D_GDS_workflow_transmission_in_2DEG_nnp\bias_00000\CBR\transmission_sums_device_Gamma.dat*

Within this tutorial we present a convenient methodology of simulating transmission in top-gated structure focusing on the geometrical design of the gates. As transmission of such structures highly depend on the geometry of the gates we propose approach involving usage GDSII files to make the definition of the gates more comfortable. The workflow is presented on the example of the Electron Flying Qubit.

The simulated structure : Electron flying qubit

The implementation of an electron Flying Qubit requires to estimate the changes in the electrostatic potential in the 2DEG region as a function of the applied bias to the QPCs. Depending on the shape of the gates 3D simulations of the gates are required, demanding huge runtime for obtaining the potential at each point of the two-dimensional electron gas formed at a certain depth of the structure. A critical value of the bias that depletes the electrons in this region (the pinch-off voltage) shall be computed with high accuracy. Knowing the pinch-off the transport of electrons in this layer of the structure can be fully controlled.

Nevertheless, the effort for computing the transmission in the 2DEG region can be reduced, if we restrict the simulation to a plane in this region.

This tutorial illustrates this workflow using the structure presented in Figure 4.18.5.1.

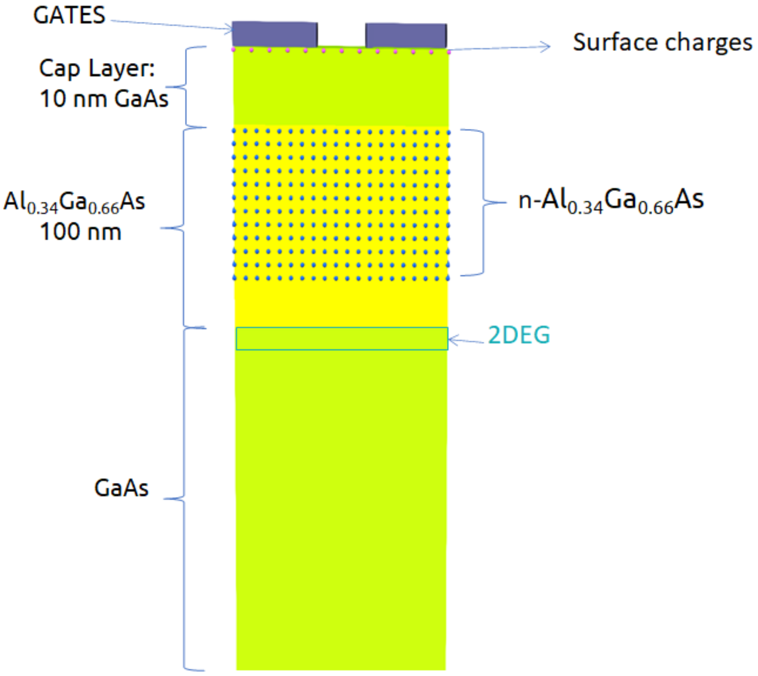


Figure 4.18.5.1: Device under simulation

The two-dimensional electron gas is formed at the interface of the AlGaAs and GaAs (the substrate) materials. Doping the AlGaAs with n-type impurities at a certain distance of this interface improves the confinement of electrons in the 2DEG region. A GaAs layer over the n-AlGaAs region acts as a cap of the device. Finally metallic gates with different geometries are directly deposited on the top of surface. Figure 4.18.5.2 presents several geometries used for testing this methodology.

Work flow

Figure 4.18.5.3 presents a possible methodology for computing the transmission in a channel in the 2DEG plane that provides accurate results and is less workload intensive.

The main idea is to split the structure in two parts: a stack of layers (the wafer) and the parts with more complex geometries.

The wafer specifications and experimental measurements provide the most relevant information for modeling the structure to be simulated. As wafer specifications we mean the different material, alloy composition, doping and thickness of each layer of the device before deposition of the metallic gates on its surface. Experimental measurements, when available, can become a key component for estimation of the surface charges and reduction of the incertainty or the doping concentration of the n-AlGaAs layer.

The combination of these two elements (wafer specification and experimental measurements) provides the required information to calibrate the model. If the experimental measurements are translation invariant on the each layer

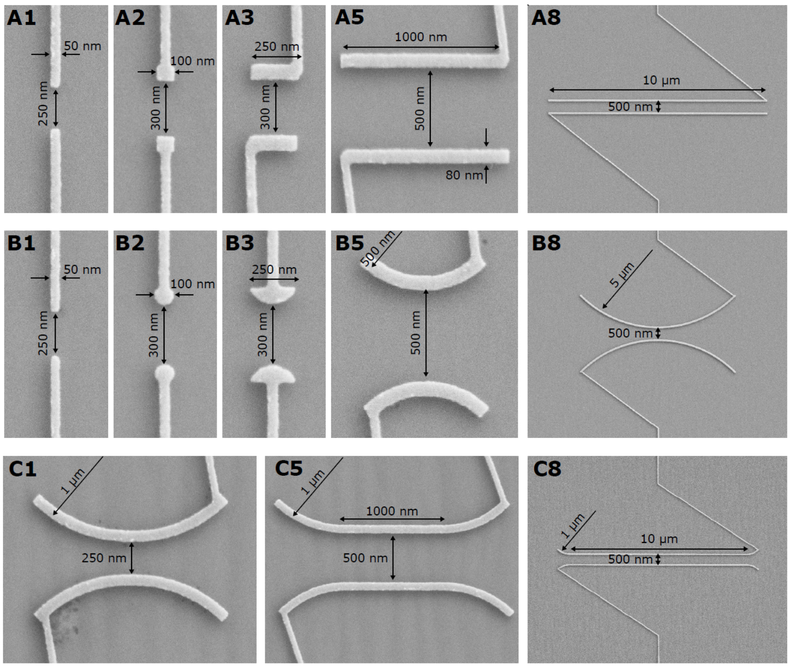


Figure 4.18.5.2: Geometry of the gates used to test this methodology. Source: *[Chatzikyriakou_PhysRevResearch_2022]*

of the sample, a simple 1D simulation can be used. Nevertherless it is important to validate and to perform a sensitivity analysis of the new grid that will be used in the simulation of the device in three dimensions.

In the case of UltraFastNano we successfully adopted a methodology that provided high accuracy in the estimate of the charge distribution of the manufactured device. This methodology for the calibration was tested for hundreds of geometries of the gates, and it detailed described in *[Chatzikyriakou_PhysRevResearch_2022]*

As a final result, we can obtain an input file of the calibrated structure without the gates, that we will use as template.

In this tutorial we will not discuss the previous steps in detail, because they are very dependent on the manufacturing and modeling of each specific device under test. The dashed region in the workflow shown in the figure is the one we will discuss in the following sections.

Once we build the template, we can import the geometric information of the gates from a file, for example in GDS format, as we will use as example. We will present a script to perform this operation in a simple way. The resulting input file will be used in 3D simulations, generating automatically the potential in the whole structure, and the corresponding slice in plane in the 2DEG region.

This slice of the electrostatic potential can be exported to a 2D-input file that will compute the transmission between two leads for a certain bias in the gates.

1. Implementing the structure without gates

It is always convenient to start defining an input file that will contains all information of the calibrated wafer with the model that will be used for the whole set of simulations. Our suggestion is to prepare this input file without the region of the QPCs. This will provide more flexibility for simulating gates with different geometries.

In this tutorial our template is the file *3D_GDS_workflow_template_nnp.in* that implements the stack of layers of the Figure 4.18.5.1 without the gates.

In *nextnano++*, the order of the layers in the section `structure{ }` of the input file it is important. Each new layer overwrites the previous one. Another important detail it is that the doping is additive, by default. For this reason, for importing the geometry of the gates to the right position in the new input file, it is necessary to use two identifiers as delimiters of the beginning and the end of the gate region. In our example, we used as identifiers the next labels in the template file:

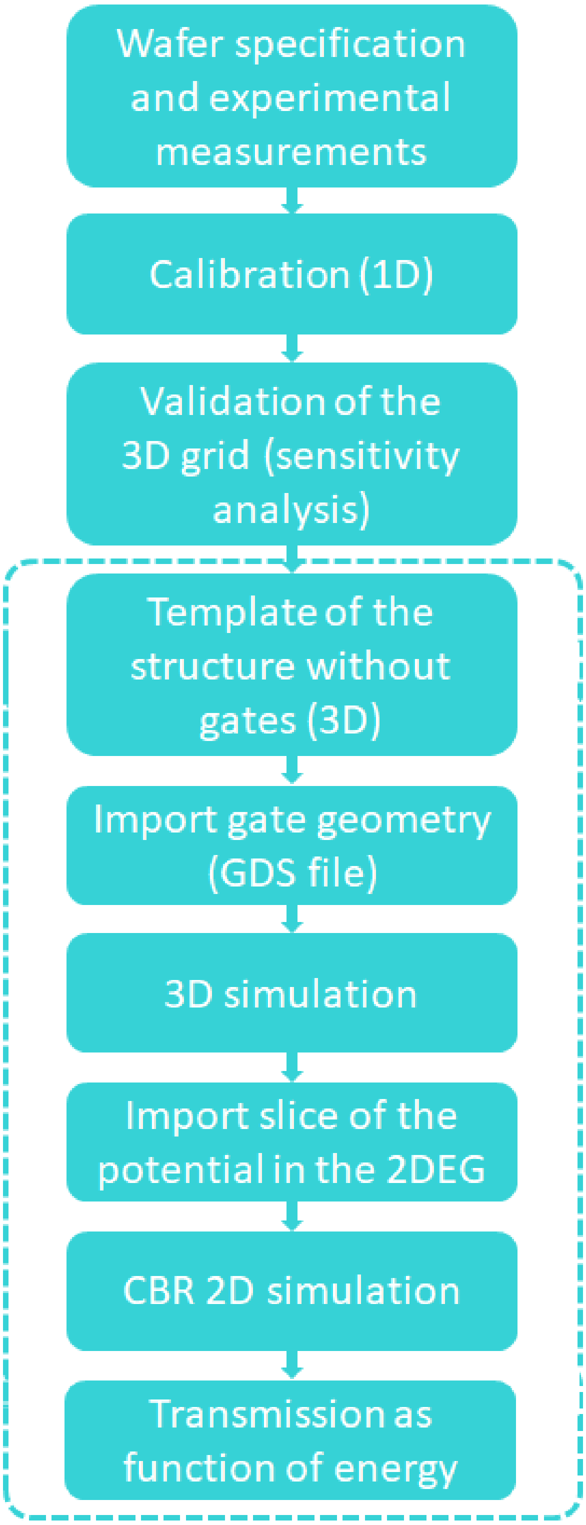


Figure 4.18.5.3: Workflow proposed by this methodology

```
# --- BEGINNING OF THE GATE REGION --- #
```

and

```
# --- END OF THE GATE REGION --- #
```

as in this snapshot of the template file:

```
# --- BEGINNING OF THE GATE REGION --- #


# ANY LINE BETWEEN THESE TWO IDENTIFIERS WILL BE REPLACED BY THE
# GATE SPECIFICATION

# --- END OF THE GATE REGION --- #
```

It is important that will be exactly two, in order to identify lines in the input file from previous simulations, like calibration, that shall be replaced by the specification of the gates.

2. Importing the geometry of the gates

For this implementation, the GDS file provides 2D polygons that shall be extruded to represent the gates in the 3D representation of the structure. In this particular example, the gates are extended from the coordinates `zi = 0` and `zf = 17` (nm), where z is the growth direction and `z = 0` corresponds to the surface of the device.

From the calibration we can estimate the surface charges and specify them in the input file in terms of a volumetric surface charge concentration, over the whole region of the structure between `zis= -1` and `zfs= -1` nm.

In the case the gates are defined as schottky contacts, as illustrated in this tutorial, the removal of surface charge concentration just under these gates is necessary.

The script presented above illustrates how to use *nextnanopy* to import the polygons corresponding to each gate and generates prisms by extrusion of them from the coordinate `zi` to `zf`. Additionaly it removes the surface charge concentration only under the gates as mentioned above (for z in the interval `[zis,zfs]`). Additional information is added, like the boundary conditions and material adjacent to these gates, when necessary.

For running the script execute `python 3D_GDS_workflow_script.py` in the command line.

The script assumes that the template file and the GDS files are stored in the folders “templates” and “GDS files” in the same directory where this script is.

Basically it will recognize the identifiers of the gate region in the template and will replace all content between these lines by the imported and processed content from the GDS file as discussed above.

At this point we encourage you to use of *nextnanopy* for performing the import of the GDS file, although this is not mandatory. Another advantage of using this package is that input files can be automatically modified and executed, and the scripts can be used for documenting each step of your simulation. We remind you that you can find *nextnanopy* in our GitHub repository at https://github.com/nextnanopy/nextnanopy: it is open source and free!

We prepared a nice Jupyter notebook at docs/examples folder concerning the import of GDS files to a *nextnano*++ input file.

3. Setup of the input file for 3D simulations

> After running the script two different inputs files will be generated:and verify the resulting input file that will be used in the 3D Simulation. It will be stored in the folder *input files* in the same directory of the script.
>
> - *\input files\3D_GDS_input_file_npp.in*
> - *\simulations\3D_GDS_Workflow_Results_V_-1.03_npp.in*

The second is one example for simulating the input file *3D_GDS_Workflow_Results_V_-1.03_npp.in* for one specific bias. In this example we will simulate for $V_{gate} = -1.03\ V$.

In the most general case, 3D simulations can be required for more accurate estimation of the pinch-off voltage. Additionally, in the development of a Electron Flying Qubit building block computation of the conduction band through the whole device is necessary, in order to reproduce the transport phenomena in the 2DEG layer.

As the simulation time depends on the number of the nodes on the grid, for more complex forms and for large devices (of order of microns) with required fine grid (of order of nm), some computers shall have not memory enough for the numerical solution of a self-consistent calculation of the Schrödinger and Poisson equations, with a minimum number of wave functions required for such operation.

In this case, a new algorithm was developed within *nextnano*++ that decomposes the 3D-problem in multiple 1D-problems. In this example, the Schrödinger-Poisson system is solved along the growth direction independently for each pair of coordinates of the nodes of the corresponding perpendicular plane. This decomposition method can be perfect applied to this structure because it is expected that the electrostatic potential does not present any abrupt variation in the any plane perpendicular to the quantization direction. For the application of this algorithm is only required to include the line `quantize_x{}`, `quantize_y{}` or `quantize_z{}` in the `quantum{ }` section of the input file. In this tutorial the quantum calculations are decomposed in solutions over the growth direction (the z-axis) and, therefore, we use quantize_z{}.

The most important result that will be used in the next steps is the electrostatic potential of the whole structure when a certain bias is applied to both gates. It also generates one slice (a plane) within the 2 DEG region.

For purposes of this tutorial it will be required to simulate the input file *\simulations\3D_GDS_Workflow_Results_V_-1.03_npp.in* using *nextnanomat*, for example.

4. Setup of the input file for 2D simulations

The next step in the workflow correspond to the calculation of the transmission of the electrons in a plane in the 2DEG region for a defined bias applied to the gates over the surface (here -1.03V).

We will perform this simulation importing the corresponding slice of the electrostatic potential obtained from the previous section, and will use the Contact Block Reduction (CBR) method, defining two leads in the simulation domain: one at the left border (lead 0) and other at the right border (lead 1). The input file *\outputs\2D_GDS_workflow_transmission_in_2DEG_nnp.in* is prepared to perform these tasks automatically.

Figure 4.18.5.4 presents the imported slice of the potential. The dashed lines represent the leads of the structure.

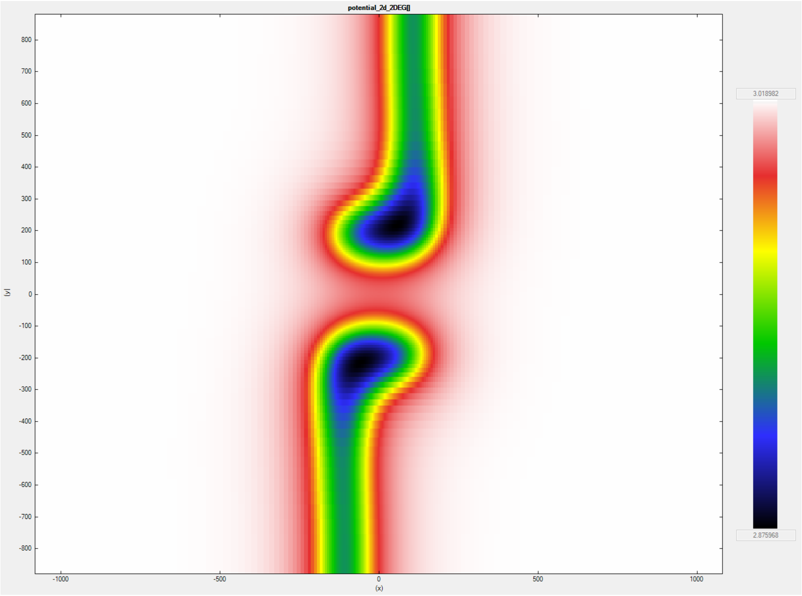


Figure 4.18.5.4: One slice of the potential in the 2DEG region

5. Plotting the transmission through the channel

Figure 4.18.5.4 shows the part of the imported slice of the potential that will actually be simulated when running *\outputs\2D_GDS_workflow_transmission_in_2DEG_nnp.in*. The image also shows the position of the leads we are considering to computing the transmission. The slice obtained from the 3D simulation at 111 nm under the surface. This results corresponds to the case Vgate = -1.03 V that is still far from the pinch-off voltage for this device, where it is expected several modes can be transmitted through the channel in the 2DEG.

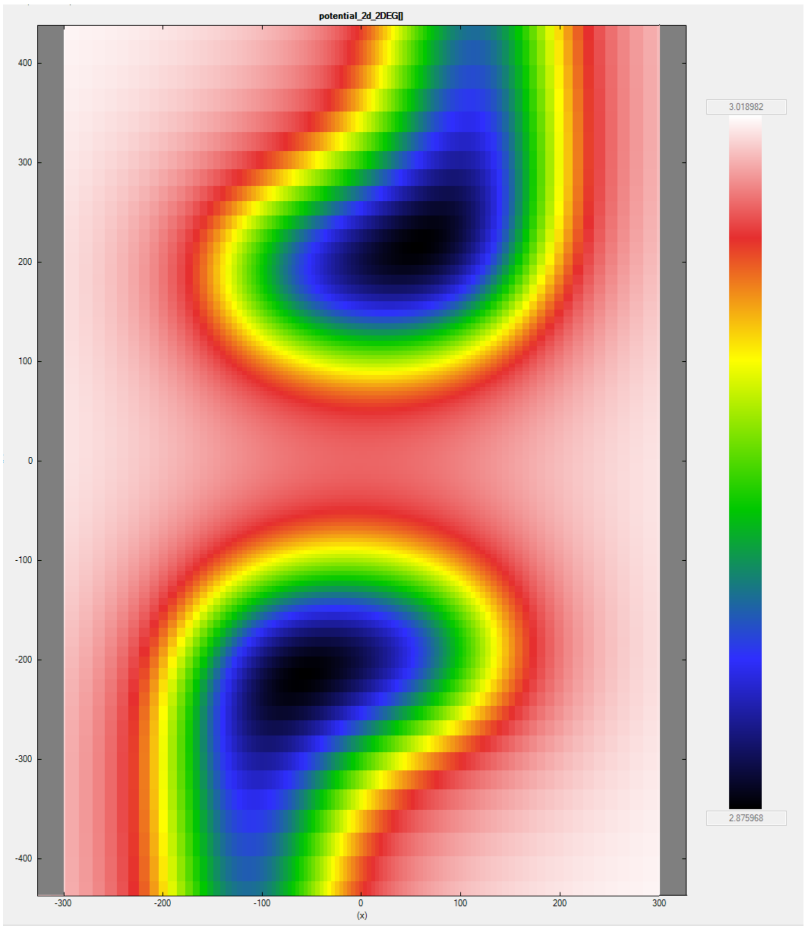


Figure 4.18.5.5: Portion of the slice of the imported potential shown in Figure 4.18.5.4 that will actually be used in the computation of the conductance in the channel in the 2DEG.

As result of the simulation of this input file, we can observe in the folder *2D_GDS_workflow_transmission_in_2DEG_nnp\bias_00000\CBR\transmission_sums_device_Gamma.dat* the transmission as function of the energy, shown in Figure 4.18.5.6.

The stepwise behavior of the transmission is consequence of the fact that the conductance is quantized.

This tutorial is based on the nextnano GmbH collaboration in the scope of the UltraFastNano Project aiming at development of the first Flying Electron Qubit at the picosecond scale, and it is funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 862683.

Ultra Fast Nano

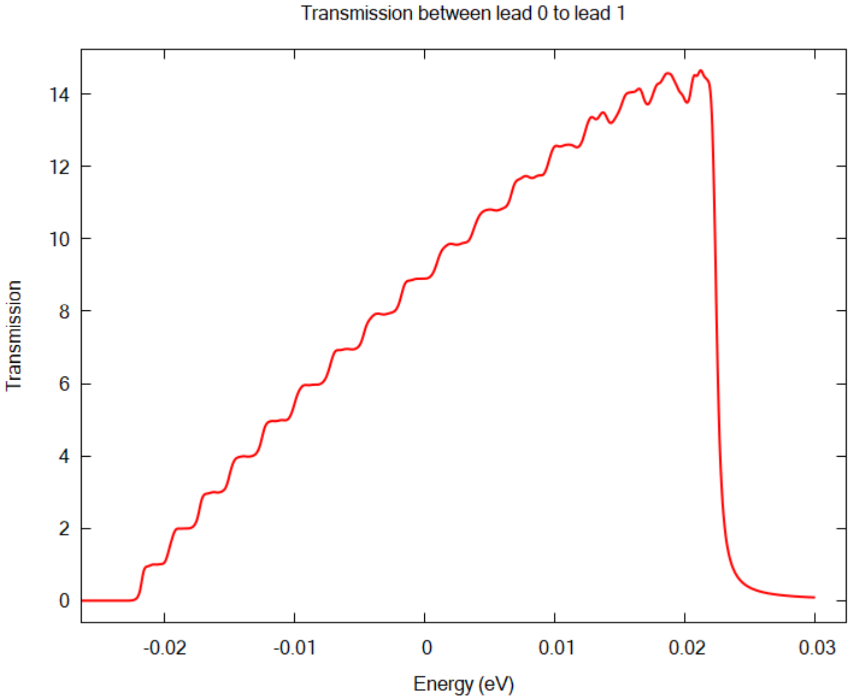


Figure 4.18.5.6: Transmission in the 2DEG region between two leads for $V_{gate} = -1.03\ V$.

Last update: 2025/06/27

4.18.6 Wurtzite GaN/AlN/GaN on Si(111)

Header

Files for the tutorial located in *nextnano++\examples\tricks_and_hacks*

- *zb-substrate-in-wz-system_GaN-AlN-Si_1D_nnp.in*

Scope of the tutorial:

- Strain
- Database

Introduction

This tutorial presents how to model wurtzite heterostructures grown on zincblende (111) substrates. Here the presented example is GaN/AlN/GaN heterostructure (see *— NEW/EDU — Piezo- and Pyroelectric charges in GaN/AlN/GaN wurtzite heterostructure*) with the substrate replaced by equivalence of Si(111).

As *nextnano*++ does not support simulations containing materials of different symmetries, it is natively not possible to define wurtzite heterostructure with zincblende substrate, and vice versa. However, one can use certain simple workaround for special cases.

Solution for a special case

Let us consider a substrate made of any material having zincblende or diamond structure with the surface (111) being prepared for the epitaxial growth. Let us also assume that the heterostructure deposited on that surface has wurtzite symmetry oriented respective to the substrate in a way that the [0001] being perpendicular to the substrate surface (parallel to the growth direction).

Based on basic geometrical considerations (see Figure 4.18.6.1), the last monolayer of the substrate can be modelled as the last layer of some artificial material with wurtzite symmetry of proper lattice constant, somehow related to the real material of the substrate. In other words, it can be seen that the last monoatomic layer in the plane (111) of zincblende or diamond crystals have exactly the same symmetry as a monoatomic layer in the place (0001) of wurtzite crystals.

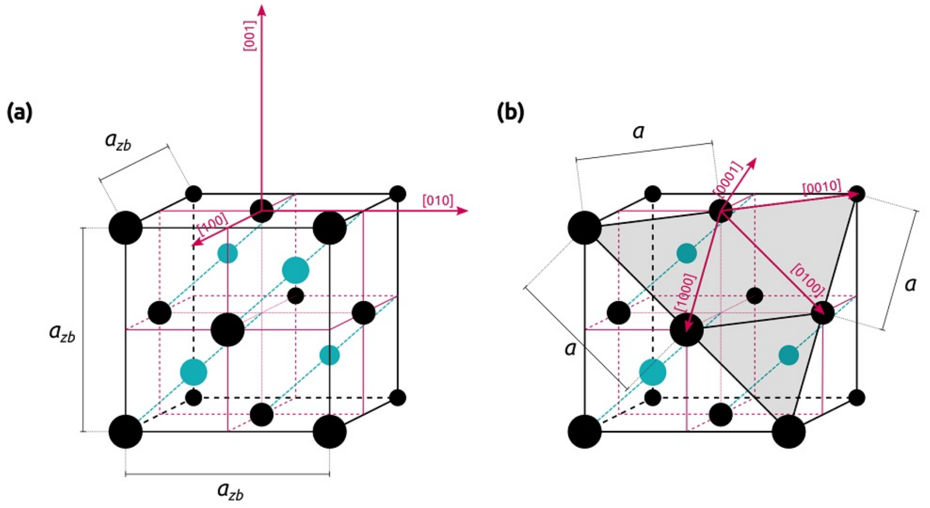


Figure 4.18.6.1: Conventional unit cell of a zincblende crystal or diamond. (a) Crystallographic directions of cubic crystals are plotted. Lattice constant of the crystal is denoted as a_{zb} (b) The unit cell is sliced through one of (111) planes. Crystallographic directions of wurtzite crystals which can be grown on such a plane are plotted. Lattice constant experienced by potentially deposited wurtzite material is denoted by a.

Distances between the atoms in that monolayer constitute a lattice constant a of the artificial wurtzite crystal that can be used to define the substrate for the simulation. It's lattice constant c does not matter here. Based on forementioned geometrical considerations, if the lattice constant of the zincblende substrate is a_{zb}, then

$$a = \frac{a_{zb}}{\sqrt{2}} \tag{4.18.6.1}$$

for the artificial, corresponding wurtzite material.

Respective thermal expansion coefficients follow the same transformation.

Therefore, if the substrate is made of Silicon, which has lattice constant $a_{Si} = 5.4304$ Å at room temperature and expansion coefficient $a_{exp,Si} = 1.8138 \times 10^{-5}$ Å/K, then the corresponding artificial wurtzite crystal will have the lattice constant

$$a = \frac{a_{Si}}{\sqrt{2}} = \frac{5.4304 \text{ Å}}{\sqrt{2}} = 3.8399 \text{ Å} \tag{4.18.6.2}$$

at room temperature and the expansion coefficient

$$a_{exp} = \frac{a_{exp,Si}}{\sqrt{2}} = \frac{1.8138 \times 10^{-5} \text{ Å/K}}{\sqrt{2}} = 1.2826 \times 10^{-5} \text{ Å/K}. \tag{4.18.6.3}$$

The other lattice constant c and the related expansion coefficient can be chosen arbitrary.

Implementation

To implement this solution in the simulation one needs to do only two things:

1. define the corresponding artificial wurtzite material,
2. use it as a substrate.

Attention

Presented approach is valid only when the zincblende substrate is not included in the simulation domain.

The easiest way to define the artificial material for the substrate is to follow suggestions from *Defining New Materials*. In the input file *zb-substrate-in-wz-system_GaN-AlN-Si_1D_nnp.in* we did that by - copy-pasting definition of GaN from our database (*database_nnp.in*) under `database{ }` group - modifying the name, one lattice constant and one expansion coefficient

Note

We have also removed unnecessary comments and not required definitions for simplicity. The code examples are also simplified. Compare them with the input file to this tutorial.

```
database{
    binary_wz{
        name = "Si_wz_substrate_only"
        lattice_consts{
            a = 3.8399
            a_expansion = 1.2826e-5
        }
    }
}
```

To use this material for the simulation it is used only as a substrate in the `global{ }` group.

```
global{
    substrate{ name = "Si_wz_substrate_only" }
}
```

Last update: 17/07/2024

4.18.7 Automatically running processes after simulation

Header

This tutorial shows a couple of examples of how to use *postprocessor{ }* group.

Properties of the input file

```
postprocessor{
    datafile = "query.bat"
    call = "query.bat"
    goto_output = yes
}

!DATA

@echo off

@echo:

FOR %%? IN (*.*) DO (
    ECHO File Name Only       : %%~n?
    ECHO Name in 8.3 notation : %%~sn?
    ECHO File Extension       : %%~x?
    ECHO File Attributes      : %%~a?
    ECHO Located on Drive     : %%~d?
    ECHO File Size            : %%~z?
    ECHO Last-Modified Date   : %%~t?
    ECHO Parent Folder        : %%~dp?
    ECHO Fully Qualified Path : %%~f?
    ECHO FQP in 8.3 notation  : %%~sf?
    @echo:
)

@echo:
```

Deleting excess output files

The script below moves the *bias_00000Quantumamplitudes_quantum_region_Gamma.dat* outside of the *bias_00000Quantum* directory (to *bias_00000*) and deletes the:guilabel:*bias_00000Quantum* directory with the entire content.

Note

The removing command (rmdir) is called in the quiet mode (/q) such that no prompts occur and the script can be executed automatically.

```
postprocessor{
    datafile = "query.bat"
    call = "query.bat"
    goto_output = yes
}

!DATA

move "bias_00000\Quantum\amplitudes_quantum_region_Gamma.dat" "bias_00000\amplitudes_
↪quantum_region_Gamma.dat"
rmdir /s /q "bias_00000\Quantum"
```

Last update: 17/07/2024

OTHER SIMULATIONS

5.1 Christmas HEMT (2021/12)

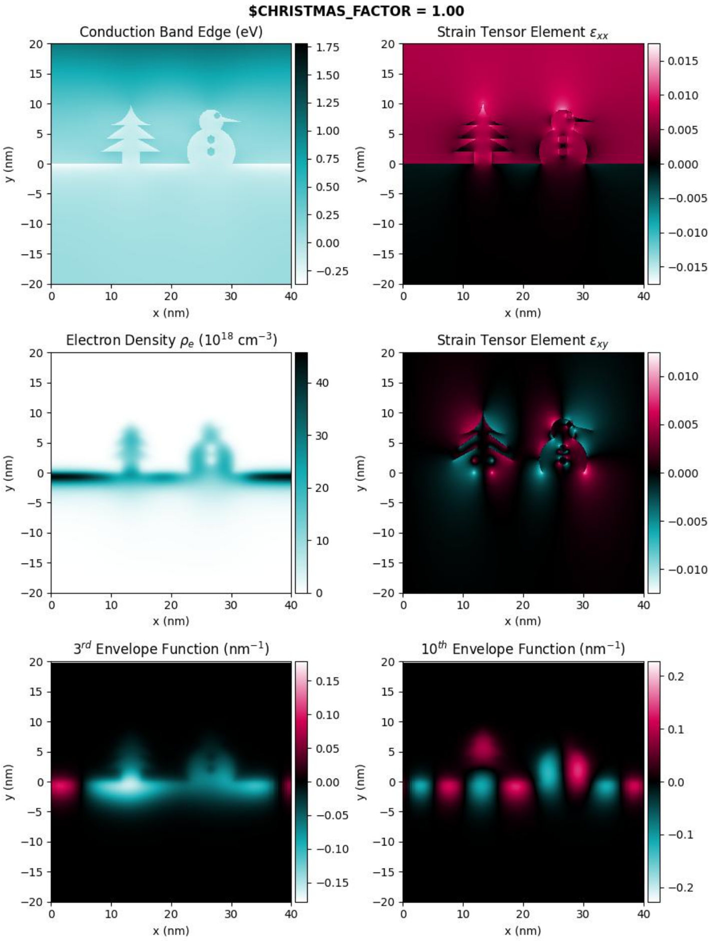


Figure 5.1.1: 2021/12 - Christmas HEMT with *nextnano++* . Get `2024_christmas_HEMT.in` here.

CHAPTER

SIX

MATERIAL DATABASE

Note

This section is under construction

6.1 Introduction to Material Database

As *nextnano++* is a general tool for simulations of semiconductor devices, we have structured our database to handle numerous materials and alloys. The database is prepared to hold any materials which can be described within one of two models currently implemented in the *nextnano++* code. These are models for crystals with zincblende and wurtzite symmetries. The database is not limited, which means that you can modify it and extend it adding new materials as much as you need for your purposes. Below, you can find a pedestrian guide to the material database of *nextnano++*.

6.1.1 Parameters of Elements & Binary Compounds

The first step to learn how to modify the material parameters is to open *nextnano++\Syntax\database_nnp.in* in your installation folder to see how it is structured. You can open it with any text editor.

You will quickly notice that every single material is contained in a separate top-level group `binary_zb{}`, if it follows model for zincblende crystals, or `binary_wt{}`, if it is described within models for wurtzite symmetry. These are the most important groups in the database, and most likely, these you want to edit as they contain all material parameters for pure components (mostly binaries) your materials which can further form alloys. Every such group has similar structure, namely they begin with two attributes with some value assigned `name` and `valence`, and other groups containing multiple parameters describing given material. See a few examples below.

```
binary_zb{
    name     = Si
    valence = IV_IV

    # Some other groups
}
```

```
binary_zb {
    name     = GaAs
    valence = III_V

    # Some other groups
}
```

```
binary_wz {
    name     = GaN
    valence = III_V

```

(continues on next page)

(continued from previous page)

```
    # Some other groups
}
```

The `name` attribute defines the name of defined material, which you use to refer to the set of parameters. The `valence` attribute specifies families of elements forming the binaries.

Task

Find these groups in your database. Depending on the release, they may be present at different lines.

Next, groups containing all the parameters are listed in some order, e.g., `lattice_consts{}`, `dielectric_consts{}`, and so on.

```
binary_wz {
    name    = ZnO
    valence = II_VI

    lattice_consts{
        # Some parameters
    }

    dielectric_consts{
        # Some parameters
    }

    # Some other groups
}
```

In principle, you can modify these parameters and run *nextnano*++ with them. However, other approach might be better for you. If you need to change the database only a bit or only for some of your simulations then better do not change it here.

Attention

Every single parameter for the model must be defined in these groups.

6.1.2 Bowing Parameters and Ternary Alloys

The next are definitions of ternary alloys. They begin separately for each kind of alloys, following definitions of respective binaries and elements. You can find the definitions of ternary alloys beginning with big comments. A few examples below for materials with zincblende symmetry.

```
############################################################################
↪##############
#          T E R N A R Y     A L L O Y S     --    IV - IV       V A L E N␣
↪C E
############################################################################
↪#############
```

```
############################################################################
↪##############
#          T E R N A R Y     A L L O Y S     --    III - V       V A L E N␣
↪C E
############################################################################
↪##############
```

```
#############################################################################
↪##############
#          T E R N A R Y     A L L O Y S     --     II - VI       V A L E N␣
↪C E
#############################################################################
↪##############
```

These comments are directly followed by lists of available alloys definitions, which you can use in your input files. Then, similarly o the definitions of the binary compounds, bowing parameters of specific ternaries and definitions of the alloys are coded within three top-level groups.

Constant Bowing Parameters

The simplest definition, for ternaries with bowing parameters not-dependent on the mole fraction, is done with a group `ternary_zb{}` like in the case of SiGe.

```
1433 ternary_zb {
1434     name        = "Si(1-x)Ge(x)"
1435     valence     = IV_IV
1436     binary_x    = Ge
1437     binary_1_x  = Si
1438 
1439     conduction_bands{
1440         Delta{
1441         bandgap = 0.206
1442         }
1443     }
1444 
1445     kp_6_bands{
1446         L = 0   M = 0   N = 0
1447     }
1448 
1449 } : {
1450     name        = "Ge(x)Si(1-x)"
1451     valence     = IV_IV
1452     binary_x    = Ge
1453     binary_1_x  = Si
1454 } : {
1455     name        = "Si(x)Ge(1-x)"
1456     valence     = IV_IV
1457     binary_x    = Si
1458     binary_1_x  = Ge
1459 } : {
1460     name        = "Ge(1-x)Si(x)"
1461     valence     = IV_IV
1462     binary_x    = Si
1463     binary_1_x  = Ge
1464 }
```

The main body of `ternary_zb{}` (lines 1434-1448) is structured very similarly to what can be found in `binary_zb{}`. First, the reference name of the alloy is specified by setting some string to the attribute `name`. Then the attribute `valence` is set to element families the same as for binaries or elements. Besides these attributes, another two are introduced: `binary_x` and `binary_1_x`. Names of already defined binary materials must be assigned to these attributes. Having

```
1434     name        = "Si(1-x)Ge(x)"
1435     valence     = IV_IV
```

(continues on next page)

(continued from previous page)

```
    binary_x   = Ge
    binary_1_x = Si
```

means that material parameters of the material with a name "Si(1-x)Ge(x)", categorized as IV-IV alloy, are computed based on material parameters of materials named "Ge" and "Si" in the database, where a mole fraction x specified in the input file corresponds to the mole fraction of "Ge", while a value $(1 - x)$ is a mole fraction of "Si" in this alloy.

Definition of bowing parameters for the alloy is following these four attributed. Syntax related to the parameters is exactly the same as in previously described group `binary_zb{}` with a difference, such that if some bowing parameters are not defined, then they are set to **zero** by default.

Attention

Not defined bowing parameters are set to **zero** by default.

The three extra sections following the top part of the group `ternary_zb{}` are clones of the group with the top-level attributes redefined. This allows to create equivalent definitions of the same alloy, with different names.

```
} : {
    name       = "Ge(x)Si(1-x)"
    valence    = IV_IV
    binary_x   = Ge
    binary_1_x = Si
} : {
    name       = "Si(x)Ge(1-x)"
    valence    = IV_IV
    binary_x   = Si
    binary_1_x = Ge
} : {
    name       = "Ge(1-x)Si(x)"
    valence    = IV_IV
    binary_x   = Si
    binary_1_x = Ge
}
```

As a result multiple equivalent definitions of the same alloy are available allowing you to pick your favorite convention to express the alloy. Similar approach applies to materials with wurtzite sammetry, the difference are parameters and name of groups containing "_wz" instead of "_zb".

Fraction-Dependent Bowing Parameters

Two other groups, `bowing_zb{}` and `ternary2_zb{}` are needed to define an alloy with bowing parameter dependent on the mole fraction. The definition of such ternary alloy begins with defining the bowing parameters for mole fractions **zero** and **one**, which will further be linearly interpolated for each intermediate composition.

An example of such alloy is AlGaSb, for which two groups of bowing parameters, `bowing_zb{}`, are specified in the database.

```
bowing_zb {
    name       = "AlGaSb_Bowing_Al"
    valence    = III_V

    # Some parameters
}
```

```
bowing_zb {
    name        = "AlGaSb_Bowing_Ga"
    valence     = III_V

    # Some parameters
}
```

These groups do not contain any definition of what are the constituent materials and how they should be interpreted. They only contain reference name, family assigned and sets of bowing parameters.

The groups `ternary2_zb{}` are used just after to define such dependencies, e.g.,

```
ternary2_zb {
    name        = "Al(x)Ga(1-x)Sb"
    valence     = III_V
    binary_x    = AlSb
    binary_1_x  = GaSb
    bowing_x    = AlGaSb_Bowing_Al
    bowing_1_x  = AlGaSb_Bowing_Ga
}
```

defines a ternary alloy named “Al(x)Ga(1-x)Sb”, classified as III-V material, constructed based on materials “AlSb” and “GaSb”, with the bowing parameters linearly interpolated from these listed in “AlGaSb_Bowing_Al” to these listed in “AlGaSb_Bowing_Ga” when mole fraction x goes from 1 to 0.

Ternaries, Quaternaries, & Quinternaries

Other types of alloys are defined similarly to ternaries, the same rules and syntax applies. They require proper definition of binaries and ternaries beforehand.

Related Documentation

- *Interpolation Schemes*
- *Default Materials and Alloys*
- *Definition of Band Offsets (zincblende)*
- *Input File Syntax - database{ }*

Last update: nnnn/nn/nn

6.2 Defining New Materials

How to define new materials for simulations with *nextnano*++? You may have multiple reasons for modifying your parameter database. You may like to tune some parameters to adjust the simulation to some experimental results. You are simulation for a new material or a material with not very well established parameters, so you need to explore results in the space of various values of the parameters. Your technological process produces “the same” material with slightly different parameters in various regions of your simulation, so you need to have a duplicate behaving slightly different in different areas of your simulation.

To address all of this issues you need one of two solutions either to use a keyword group *database{ }* in your input file or modify the database file *nextnano++\Syntax\database_nnp.in*. Both methods requires you to get familiar with already existing database. If you didn’t read it yet, get familiar with our *Introduction to Material Database* first.

6.2.1 Database or Input File?

Before introducing any modifications to the material parameters, let us answer an important question: Should the modification be done in the database file (by default *nextnano++\Syntax\database_nnp.in*) or in the input file?

Here are our advices on this matter. If you want to have the change for all your simulations, then this is a good approach. If you loose the original database, then you can always download it again. Be sure that you know what you are doing. If the change is meant only for one of your simulations, then you should do it in the input file. If the change is small, and you are not sure if correct, then do it in the input file. If you need the change for multiple of your simulations but not all of them, then either create a second database or add your own modified materials independently of the default ones.

The nicest practical thing about our definition of the database is that we introduced *here*, that it is fully consistent with the syntax of the input file. It means that everything what you write inside the *database{ }* group in the input file, will behave exactly as written in the database file, e.g., in *nextnano++\Syntax\database_nnp.in*. For this purpose, the modification of the *nextnano++\Syntax\database_nnp.in* is not discussed in this site, as whatever you can script inside the `database{ }` group in your input file can be copy-pasted to the database file, e.g., at the end of the file.

6.2.2 Modifying an Existing Material

Let us assume that you would like to modify energy band gap of GaAs to a value of 1.42 eV and the average energy of valence bands to 1.26 eV. Assuming also, that you are not familiar with the syntax yet, the best approach is to open the default database file *nextnano++\Syntax\database_nnp.in* and find definition of the binary compound GaAs; It begins at the line **1596**. Below there is a simplified piece of the referred to database.

```
1596 binary_zb {
1597     name    = GaAs
1598     valence = III_V
1599 
1600     lattice_consts{
1601         a           = 5.65325
1602         a_expansion = 3.88e-5
1603     }
1604 
1605     # Some other parameters
1606 
1607     conduction_bands{
1608         Gamma{
1609             mass                  = 0.067
1610             bandgap               = 1.519
1611             bandgap_alpha         = 0.5405e-3
1612             bandgap_beta          = 204
1613             defpot_absolute       = -9.36
1614             g                     = -0.30
1615         }
1616 
1617     # Some other parameters of conduction bands
1618 
1619     }
1620 
1621     valence_bands{
1622         bandoffset        = 1.346
1623 
1624         HH{ mass          = 0.51  g = -7.86 }
1625         LH{ mass          = 0.082 g = -2.62 }
1626         SO{ mass          = 0.172 }
1627 
1628         defpot_absolute   = -1.21
```

(continues on next page)

(continued from previous page)

```
        defpot_uniaxial_b = -2.0   defpot_uniaxial_d = -4.8

        delta_SO          = 0.341
    }

    # The rest of parameters
}
```

Then, you need to copy the `name` attribute specifying which material is edited and all attributes relating to the parameters of interest, together with the groups and nested groups to which they belong to. I this case, aiming only at modification of the band gap represented by an attribute `bandgap` and average valence band energy represented by an attribute `bandoffset`, you need to write following script in your input file.

```
database{
    binary_zb {
        name = GaAs
        conduction_bands{
            Gamma{
                bandgap = 1.49
            }
        }

        valence_bands{
            bandoffset = 1.26
        }
    }
}
```

Note that all the copy-pasted script is additionally enclosed in the `database{ }` group for the input file.

Exactly the same approach can be applied to modify bowing parameters. Having definition in the database as follows

```
ternary_zb {
    name       = "In(x)Ga(1-x)As"
    # Here you can add or edit:
    # - other attributes
    # - ternary bowing parameters
} : {
    name       = "Ga(1-x)In(x)As"
    # Other Attributes
} : {
    name       = "Ga(x)In(1-x)As"
    # Other Attributes
} : {
    name       = "In(1-x)Ga(x)As"
    # Other Attributes
}
```

you should use only the top group, the one containing parameters, for redefinition with any of four names as reference. For example, assuming that you are aiming at changing bowing parameter for spin-orbit coupling energy, you need to have

```
database{
    ternary_zb {
        name = "In(x)Ga(1-x)As"
        valence_bands{
```

(continues on next page)

(continued from previous page)

```
            delta_SO    =  0.15
        }
    }
}
```

or

```
database{
    ternary_zb {
        name = "Ga(1-x)In(x)As"
        valence_bands{
            delta_SO    =  0.15
        }
    }
}
```

or with one of the remaining names, in your input file.

6.2.3 Defining a New Binary Compound or Element

Defining a new material is similarly simple as editing the existing one. The main difference is that you need to define all the parameters of the material. The best approach is, again, to begin with copy-pasting and existing material with crystal symmetry of your interest. After that, you can edit it such that it represents the material of your interest. It is important, that the new material is named differently than existing ones or the ones that you are using in your simulation. Otherwise you are risking overwriting materials that you do not want to overwrite. Use the new name to refer to this material. Let us assume that you are interested in having Silicon in wurtzite symmetry. The first step is to locate any wurtzite binary compound defined in the database, like the one below.

```
binary_wz {
    name     = GaN
    valence = III_V
    # Some parameters
}
```

The second step is to rename conveniently it and give it parameters of the wurtzite Silicon. You can also change the family to the group IV for consistency by modifying the `valence` attribute.

```
database{
    binary_wz {
        name     = Si_wz
        valence = IV_IV
        # All wurtzite Si parameters
    }
}
```

After you are satisfied with your definition of the new material, in most cases, it makes sense to copy-paste it back to the database file. Remember to remove the `database{ }` group while doing so.

6.2.4 Defining a New Alloy

It’s the same copy-pasting procedure as before. The best is to begin with finding a definition of the alloy that is qualitatively similar to yours - the same interpolation schemes and stoichiometric notation. Let us assume that the target alloy is Tl(x)Bi(1-x)Sb with zincblende symmetry. Then AlInAs is one of many perfect starting points to define this alloy.

```
ternary_zb {
    name        = "Al(x)In(1-x)As"
```

(continues on next page)

(continued from previous page)

```
    valence     = III_V
    binary_x    = AlAs
    binary_1_x = InAs

    # Some bowing parameters

} : {
    name        = "In(1-x)Al(x)As"
    valence     = III_V
    binary_x    = AlAs
    binary_1_x = InAs
} : {
    name        = "In(x)Al(1-x)As"
    valence     = III_V
    binary_x    = InAs
    binary_1_x = AlAs
} : {
    name        = "Al(1-x)In(x)As"
    valence     = III_V
    binary_x    = InAs
    binary_1_x = AlAs
}
```

The first step is to create ternary definition, so it is clear how many and which components are required. All bowing parameters are taken equal **zero** for simplicity in the example below.

```
database{
    ternary_zb {
        name        = "Tl(x)Bi(1-x)Sb"
        valence     = III_V
        binary_x    = TlSb
        binary_1_x = BiSb

        # No ternary bowing parameters here if all are assumed to be zero

    } : {
        name        = "Bi(1-x)Tl(x)Sb"
        valence     = III_V
        binary_x    = TlSb
        binary_1_x = BiSb
    } : {
        name        = "Tl(1-x)Bi(x)Sb"
        valence     = III_V
        binary_x    = BiSb
        binary_1_x = TlSb
    } : {
        name        = "Bi(x)Tl(1-x)Sb"
        valence     = III_V
        binary_x    = BiSb
        binary_1_x = TlSb
    }
}
```

The clones are not necessary, but useful to have. The next step, is to define binaries. Therefore, any zinc-blend binary compound needs to be copied and pasted twice with names “BiSb” and “TlSb” as these have been used in the definition of the ternary “Tl(x)Bi(1-x)Sb”.

```
database{
    binary_zb {
        name     = BiSb
        valence = III_V
        # All BiSb parameters
    }
    binary_zb {
        name     = TlSb
        valence = III_V
        # All TlSb parameters
    }
}
```

Related Documentation

- *Interpolation Schemes*
- *Default Materials and Alloys*
- *Definition of Band Offsets (zincblende)*
- *Input File Syntax - database{ }*

Last update: nnnn/nn/nn

6.3 Interpolation Schemes

6.3.1 Introduction

As our software addresses simulations for a broad range of semiconductor materials, these based on binary compounds (like GaAs) and single elements (like Si), a unified naming of the alloys becomes problematic if one tries to follow standards in the literature. For example, $\mathrm{Si_xGe_{1\text{-}x}}$is a binary alloy (two elements), while $\mathrm{Ga_xIn_{1\text{-}x}As}$ is a ternary alloy (three elements), even though their parameters are interpolated using exactly the same schemes.

On the other hand side, in both cases there are only two component materials (pure materials) involved in formation of the alloy, Si and Ge in the first case, and GaAs and InAs in the second case. Therefore, in this documentation, we will refer to all pure materials (which parameters are typically tabulated in literature, like: GaAs, InN, ZnO, Si, etc.) as component materials and naming of interpolation schemes will be based on the number of these components materials involved in them. With such formalism, both $\mathrm{Si_xGe_{1\text{-}x}}$ and $\mathrm{Ga_xIn_{1\text{-}x}As}$ are two-component alloys.

Attention

Syntax of the database is consistent with standard naming for III-V and II-VI material systems. Therefore, regardless of the number of elements forming component materials, they are referred to as binaries; and the simplest available alloys are ternary alloys.

6.3.2 Two-component alloys

Two-component alloys are typically called binary alloys when group-IV are mixed (IV-IV) and ternary alloys in the case of III-V or II-VI binary compounds (III-V-V, III-III-V, II-VI-VI, and II-II-VI). Examples of such alloys are: Si_xGe_{1-x}, $Ga_xIn_{1-x}N$, and $GaAs_xSb_{1-x}$.

Material parameters of two-component alloys are interpolated based on material parameters of two components and a proper bowing parameter b_{AB} for the alloy, if defined. Three interpolation schemes are available in *nextnano++* for this type of alloys: *Linear*, *Quadratic*, and *Cubic*.

Linear - no bowing

If only parameters of the component materials are defined then a linear interpolation is used to evaluate values of the parameters for the alloy.

IV-IV

For alloys of type A_xB_{1-x}, the scheme reads

$$P_{\mathrm{AB}}\left(x\right) = x \cdot P_{\mathrm{A}} + \left[1 - x\right] \cdot P_{\mathrm{B}},$$

where $P_{\mathrm{AB}}\left(x\right)$ is an interpolated material parameter of a two-component alloy A_xB_{1-x} based on parameters P_{A} and P_{B} describing pure components A and B, respectively.

III-III-V and II-II-VI

For alloys of type $A_xB_{1-x}C$, the scheme reads

$$P_{\mathrm{ABC}}\left(x\right) = x \cdot P_{\mathrm{AC}} + \left[1 - x\right] \cdot P_{\mathrm{BC}},$$

where $P_{\mathrm{ABC}}\left(x\right)$ is an interpolated material parameter of a two-component alloy $A_xB_{1-x}C$ based on parameters P_{AC} and P_{BC} describing pure components AC and BC, respectively.

III-V-V and II-VI-VI

For alloys of type AB_xC_{1-x}, the scheme reads

$$P_{\mathrm{ABC}}\left(x\right) = x \cdot P_{\mathrm{AB}} + \left[1 - x\right] \cdot P_{\mathrm{AC}},$$

where $P_{\mathrm{ABC}}\left(x\right)$ is an interpolated material parameter of a two-component alloy AB_xC_{1-x} based on parameters P_{AB} and P_{AC} describing pure components AB and AC, respectively.

Syntax

Let us consider an alloy GaInAs with AC being GaAs and BC being InAs. All the parameters of GaAs and InAs needs to be defined within *binary_zb{}* or *binary_wz{}*. To recognize the alloy and relate names of component materials, one needs to also define *ternary_zb{}* or *ternary_wz{}*, but no bowing parameters needs to be defined there, zeroes are assumed.

```
binary_zb{
    name    = GaAs
    valence = III_V

    # All the parameters of GaAs here (P_A)
}

binary_zb{
    name    = InAs
```

(continues on next page)

(continued from previous page)

```
    valence = III_V

    # All the parameters of InAs here (P_B)
}

ternary_zb{
    name       = "Ga(x)In(1-x)As"
    valence    = III_V
    binary_x   = GaAs
    binary_1_x = InAs

    # No bowing parameters specified here
}
```

Quadratic - constant bowing

If bowing parameters are specified in the database using keywords If linear interpolation is not sufficient, quadratic interpolation with a bowing parametr can be used instead.

IV-IV

For alloys of type A_xB_{1-x}, the scheme reads

$$P_{\mathrm{AB}}\left(x\right) = x \cdot P_{\mathrm{A}} + \left[1-x\right] \cdot P_{\mathrm{B}} - x\left[1-x\right] \cdot b_{\mathrm{AB}},$$

where $P_{\mathrm{AB}}\left(x\right)$ is an interpolated material parameter of a two-component alloy A_xB_{1-x} based on parameters P_{A} and P_{B} describing pure components A and B, respectively, and b_{AB} is a bowing parameter for the alloy.

III-III-V and II-II-VI

For alloys of type $A_xB_{1-x}C$, the scheme reads

$$P_{\mathrm{ABC}}\left(x\right) = x \cdot P_{\mathrm{AC}} + \left[1-x\right] \cdot P_{\mathrm{BC}} - x\left[1-x\right] \cdot b_{\mathrm{ABC}},$$

where $P_{\mathrm{ABC}}\left(x\right)$ is an interpolated material parameter of a two-component alloy $A_xB_{1-x}C$ based on parameters P_{AC} and P_{BC} describing pure components AC and BC, respectively, and b_{ABC} is a bowing parameter for the alloy.

III-V-V and II-VI-VI

For alloys of type AB_xC_{1-x}, the scheme reads

$$P_{\mathrm{ABC}}\left(x\right) = x \cdot P_{\mathrm{AB}} + \left[1-x\right] \cdot P_{\mathrm{AC}} - x\left[1-x\right] \cdot b_{\mathrm{ABC}},$$

where $P_{\mathrm{ABC}}\left(x\right)$ is an interpolated material parameter of a two-component alloy AB_xC_{1-x} based on parameters P_{AB} and P_{AC} describing pure components AB and AC, respectively, and b_{ABC} is a bowing parameter for the alloy.

Syntax

For quadratic interpolation of a certain material paramter, one has to specify a bowing paramter b_{AB} inside the groups *ternary_zb{}* or *ternary_wz{}*.

```
binary_zb{
    name    = GaAs
    valence = III_V

    # All the parameters of GaAs here (P_A)
}

binary_zb{
    name    = InAs
```

(continues on next page)

(continued from previous page)

```
    valence = III_V

    # All the parameters of InAs here (P_B)
}

ternary_zb{
    name       = "Ga(x)In(1-x)As"
    valence    = III_V
    binary_x   = GaAs
    binary_1_x = InAs

    # Some bowing parameters  (b_AB)
}
```

Cubic - composition-dependent bowing

If a constant bowing paramter b_{AB} is not sufficient for interpolation of the parameters, like for **highly-mismatched alloys** or **dilute nitrides**, one can use a scheme where the bowing parameter is assumed to be linearly dependent on the mole fraction x, $b_{\mathrm{AB}}(x)$.

IV-IV

For alloys of type A_xB_{1-x}, the scheme reads

$$P_{\mathrm{AB}}(x) = x \cdot P_{\mathrm{A}} + [1-x] \cdot P_{\mathrm{B}} - x[1-x] \cdot b_{\mathrm{AB}}(x)$$

$$b_{\mathrm{AB}}(x) = x \cdot b_{\mathrm{AB}\to\mathrm{A}} + [1-x] \cdot b_{\mathrm{AB}\to\mathrm{B}},$$

where $P_{\mathrm{AB}}(x)$ is an interpolated material parameter of a two-component alloy A_xB_{1-x} based on parameters P_{A} and P_{B} describing pure components A and B, respectively. The $b_{\mathrm{AB}\to\mathrm{A}} = b_{\mathrm{AB}}(1)$ is a bowing parameter for nearly pure A, while the $b_{\mathrm{AB}\to\mathrm{B}} = b_{\mathrm{AB}}(0)$ is a bowing parameter for nearly pure B.

III-III-V and II-II-VI

For alloys of type $A_xB_{1-x}C$, the scheme reads

$$P_{\mathrm{ABC}}(x) = x \cdot P_{\mathrm{AC}} + [1-x] \cdot P_{\mathrm{BC}} - x[1-x] \cdot b_{\mathrm{ABC}},$$

$$b_{\mathrm{ABC}}(x) = x \cdot b_{\mathrm{ABC}\to\mathrm{AC}} + [1-x] \cdot b_{\mathrm{ABC}\to\mathrm{BC}},$$

where $P_{\mathrm{ABC}}(x)$ is an interpolated material parameter of a two-component alloy $A_xB_{1-x}C$ based on parameters P_{AC} and P_{BC} describing pure components AC and BC, respectively. The $b_{\mathrm{ABC}\to\mathrm{AC}} = b_{\mathrm{ABC}}(1)$ is a bowing parameter for nearly pure AC, while the $b_{\mathrm{ABC}\to\mathrm{BC}} = b_{\mathrm{ABC}}(0)$ is a bowing parameter for nearly pure BC.

III-V-V and II-VI-VI

For alloys of type AB_xC_{1-x}, the scheme reads

$$P_{\mathrm{ABC}}(x) = x \cdot P_{\mathrm{AB}} + [1-x] \cdot P_{\mathrm{AC}} - x[1-x] \cdot b_{\mathrm{ABC}},$$

$$b_{\mathrm{ABC}}(x) = x \cdot b_{\mathrm{ABC}\to\mathrm{AB}} + [1-x] \cdot b_{\mathrm{ABC}\to\mathrm{AC}}$$

where $P_{\mathrm{ABC}}(x)$ is an interpolated material parameter of a two-component alloy AB_xC_{1-x} based on parameters P_{AB} and P_{AC} describing pure components AB and AC, respectively. The $b_{\mathrm{ABC}\to\mathrm{AB}} = b_{\mathrm{ABC}}(1)$ is a bowing parameter for nearly pure AB, while the $b_{\mathrm{ABC}\to\mathrm{AC}} = b_{\mathrm{ABC}}(0)$ is a bowing parameter for nearly pure AC.

Example and Syntax

Let us consider the bowing paramters of energy gaps in $Al_xGa_{1-x}As$ based on the Table XII. in [*vurgaftmanjap2001*]. The direct gap has a bowing parameter given by the formula

$$b_{\mathrm{AlGaAs}}(x) = -0.127 + 1.310 \cdot x$$

while indirect gaps to the points L and X have bowing parameters 0 and 0.055, respectively. Therefore, two bowing parameters needs to be included in the database, the one at mole fraction x=0 to describe the interpolation for small amounts of Al, near GaAs:

$$b_{\mathrm{AlGaAs}\rightarrow\mathrm{GaAs}} = b_{\mathrm{AlGaAs}}\left(0\right) = -0.127 + 1.310 \cdot 0 = -0.127,$$

and at x=1 to describe the interpolation for small amounts of Ga, near AlAs:

$$b_{\mathrm{AlGaAs}\rightarrow\mathrm{AlAs}} = b_{\mathrm{AlGaAs}}\left(1\right) = -0.127 + 1.310 \cdot 1 = 1.183.$$

Finally, the fraction-dependent bowing parameter is given by

$$b_{\mathrm{AlGaAs}}\left(x\right) = x \cdot b_{\mathrm{AlGaAs}\rightarrow\mathrm{AlAs}} + \left[1 - x\right] \cdot b_{\mathrm{AlGaAs}\rightarrow\mathrm{GaAs}}$$

To use this of interpolation, one **should not** use *ternary_zb{}* or *ternary_wz{}* groups to define bowing parameters. Instead, groups *bowing_zb{}* or *bowing_wz{}* should be used to define valued of the bowing for extrememal concentrations, x=0 and x=1. The groups *ternary2_zb{}* and *ternary2_wz{}* should be used to relate all the bowing parameters and component materials for the alloy.

```
binary_zb{
    name    = AlAs
    valence = III_V

    # All the parameters of GaAs here (P_A)
}

binary_zb{
    name    = GaAs
    valence = III_V

    # All the parameters of InAs here (P_B)
}

# Al(x)Ga(1-x)As: (x=1)
bowing_zb{
    name       = "AlGaAs_Bowing_AlAs"
    valence    = III_V

    conduction_bands{
        Gamma{ bandgap = -0.127 + 1.310 * 1 }  # b_AB(x=1)
        X    { bandgap = 0.055              }  # b_AB(x=1)
    }
}

# Al(x)Ga(1-x)As: (x=0)
bowing_zb{
    name       = "AlGaAs_Bowing_GaAs"
    valence    = III_V

    conduction_bands{
        Gamma{ bandgap = -0.127 + 1.310 * 0 }  # b_AB(x=0)
        X    { bandgap = 0.055              }  # b_AB(x=0)
    }
}

ternary2_zb{
    name       = "Al(x)Ga(1-x)As"
```

(continues on next page)

(continued from previous page)

```
    valence     = III_V

    binary_x    = AlAs
    binary_1_x  = GaAs
    bowing_x    = AlGaAs_Bowing_AlAs    # b_AB(x=1)
    bowing_1_x  = AlGaAs_Bowing_GaAs    # b_AB(x=0)
}
```

Note, that there is no bowing parameter specified for the indirect band gap to the L valley, which is equivalent to using linear interpolation (the bowing equal zero).

Hint

An alternative approach can be to use *analytical formulas* to define the bowing parameter with the mole fraction as a variable.

6.3.3 Three-component alloys

Three-component alloys are typically called ternary alloys when group-IV are mixed (IV-IV-IV) and quaternary alloys in the case of III-V or II-VI binary compounds (III-V-V-V, III-III-III-V, II-VI-VI-VI, and II-II-II-VI).

Examples of such alloys are: $\mathrm{Si_xGe_y\,Sn_{1\text{-}x\text{-}y}}$, $\mathrm{Al_xGa_yIn_{1\text{-}x\text{-}y}N}$, and $\mathrm{GaP_xAs_ySb_{1\text{-}x\text{-}y}}$.

IV-IV-IV

For alloys of type $\mathrm{A_xB_yC_{1\text{-}x\text{-}y}}$, having $w = 1 - x - y$, the scheme reads

$$\begin{aligned} P_{\mathrm{ABC}}(x,y) &= P'_{AB}(x,y) + P'_{AC}(x,w) + P'_{BC}(y,w) \\ &- xy \cdot b'_{\mathrm{AB}}(x,y) - xw \cdot b'_{\mathrm{AC}}(x,w) - yw \cdot b'_{\mathrm{BC}}(y,w) \\ &- xyw \cdot b_{\mathrm{ABC}}. \end{aligned}$$

The $P'_{AB}(x,y)$, $P'_{AC}(x,w)$, and $P'_{BC}(y,w)$ are linear combinations of parameters P_{A}, P_{B}, and P_{C}.

$$\begin{aligned} P'_{AB}(x,y) &= x \cdot P_{\mathrm{A}} + y \cdot P_{\mathrm{B}} \\ P'_{AC}(x,w) &= x \cdot P_{\mathrm{A}} + w \cdot P_{\mathrm{C}} \\ P'_{BC}(y,w) &= y \cdot P_{\mathrm{B}} + w \cdot P_{\mathrm{C}} \end{aligned}$$

The $b'_{\mathrm{AB}}(x,y)$, $b'_{\mathrm{AC}}(x,w)$, and $b'_{\mathrm{BC}}(y,w)$ are two-component bowing parameters. They can be equal zero, constant or dependent on mole fraction as:

$$\begin{aligned} b'_{\mathrm{AB}}(x,y) &= \frac{x \cdot b_{\mathrm{AB\to A}} + y \cdot b_{\mathrm{AB\to B}}}{x+y} \\ b'_{\mathrm{AC}}(x,w) &= \frac{x \cdot b_{\mathrm{AC\to A}} + w \cdot b_{\mathrm{AC\to C}}}{x+w} \\ b'_{\mathrm{BC}}(y,w) &= \frac{y \cdot b_{\mathrm{BC\to B}} + w \cdot b_{\mathrm{BC\to C}}}{y+w}. \end{aligned}$$

The b_{ABC} is a three-component bowing parameter.

III-III-III-V and II-II-II-VI

For alloys of type $\mathrm{A_xB_yC_{1\text{-}x\text{-}y}D}$, having $w = 1 - x - y$, the scheme reads

$$\begin{aligned} P_{\mathrm{ABCD}}(x,y) &= P'_{ABD}(x,y) + P'_{ACD}(x,w) + P'_{BCD}(y,w) \\ &- xy \cdot b_{\mathrm{ABD}}(x) - xw \cdot b_{\mathrm{ACD}}(x) - yw \cdot b_{\mathrm{BCD}}(x) \\ &- xyw \cdot b_{\mathrm{ABCD}}, \end{aligned}$$

The $P'_{ABD}\left(x,y\right)$, $P'_{ACD}\left(x,w\right)$, and $P'_{BCD}\left(y,w\right)$ are linear combinations of parameters P_{AD}, P_{BD}, and P_{CD}.

$$\begin{aligned} P'_{ABD}\left(x,y\right) &= x\cdot P_{\mathrm{AD}} + y\cdot P_{\mathrm{BD}} \\ P'_{ACD}\left(x,w\right) &= x\cdot P_{\mathrm{AD}} + w\cdot P_{\mathrm{CD}} \\ P'_{BCD}\left(y,w\right) &= y\cdot P_{\mathrm{BD}} + w\cdot P_{\mathrm{CD}} \end{aligned}$$

The $b'_{\mathrm{ABD}}\left(x,y\right)$, $b'_{\mathrm{ACD}}\left(x,w\right)$, and $b'_{\mathrm{BCD}}\left(y,w\right)$ are two-component bowing parameters. They can be equal zero, constant or dependent on mole fraction as:

$$\begin{aligned} b'_{\mathrm{ABD}}\left(x,y\right) &= \frac{x\cdot b_{\mathrm{ABD}\to\mathrm{AD}} + y\cdot b_{\mathrm{ABD}\to\mathrm{BD}}}{x+y} \\ b'_{\mathrm{ACD}}\left(x,w\right) &= \frac{x\cdot b_{\mathrm{ACD}\to\mathrm{AD}} + w\cdot b_{\mathrm{ACD}\to\mathrm{CD}}}{x+w} \\ b'_{\mathrm{BCD}}\left(y,w\right) &= \frac{y\cdot b_{\mathrm{BCD}\to\mathrm{BD}} + w\cdot b_{\mathrm{BCD}\to\mathrm{CD}}}{y+w}. \end{aligned}$$

The b_{ABCD} is a three-component bowing parameter.

III-V-V-V and II-VI-VI-VI

For alloys of type $\mathrm{AB_xC_yD_{1\text{-}x\text{-}y}}$, having $w = 1 - x - y$, the scheme reads

$$\begin{aligned} P_{\mathrm{ABCD}}\left(x,y\right) = {} & P'_{ABC}\left(x,y\right) + P'_{ABD}\left(x,w\right) + P'_{ACD}\left(y,w\right) \\ & - xy\cdot b_{\mathrm{ABC}}\left(x\right) - xw\cdot b_{\mathrm{ABD}}\left(x\right) - yw\cdot b_{\mathrm{ACD}}\left(x\right) \\ & - xyw\cdot b_{\mathrm{ABCD}}, \end{aligned}$$

The $P'_{ABC}\left(x,y\right)$, $P'_{ABD}\left(x,w\right)$, and $P'_{ACD}\left(y,w\right)$ are linear combinations of parameters P_{AB}, P_{AC}, and P_{AD}.

$$\begin{aligned} P'_{ABC}\left(x,y\right) &= x\cdot P_{\mathrm{AB}} + y\cdot P_{\mathrm{AC}} \\ P'_{ABD}\left(x,w\right) &= x\cdot P_{\mathrm{AB}} + w\cdot P_{\mathrm{AD}} \\ P'_{ACD}\left(y,w\right) &= y\cdot P_{\mathrm{AC}} + w\cdot P_{\mathrm{AD}} \end{aligned}$$

where b_{ABC}, b_{ABD}, and b_{ACD} are two-component bowing parameters and b_{ABCD} is a three-component bowing parameter.

The $b'_{\mathrm{ABC}}\left(x,y\right)$, $b'_{\mathrm{ABD}}\left(x,w\right)$, and $b'_{\mathrm{ACD}}\left(y,w\right)$ are two-component bowing parameters. They can be equal zero, constant or dependent on mole fraction as:

$$\begin{aligned} b'_{\mathrm{ABC}}\left(x,y\right) &= \frac{x\cdot b_{\mathrm{ABC}\to\mathrm{AB}} + y\cdot b_{\mathrm{ABC}\to\mathrm{AC}}}{x+y} \\ b'_{\mathrm{ABD}}\left(x,w\right) &= \frac{x\cdot b_{\mathrm{ABD}\to\mathrm{AB}} + w\cdot b_{\mathrm{ABD}\to\mathrm{AD}}}{x+w} \\ b'_{\mathrm{ACD}}\left(y,w\right) &= \frac{y\cdot b_{\mathrm{ACD}\to\mathrm{AC}} + w\cdot b_{\mathrm{ACD}\to\mathrm{AD}}}{y+w}. \end{aligned}$$

The b_{ABCD} is a three-component bowing parameter.

Syntax

As the two-component bowing parameters can be linearly dependent on composition, constant, or equal zero, one needs to begin with defining the parameters for all material components with the bowing parameters for two-component alloys, following the syntax described before in sections *Linear*, *Quadratic*, and *Cubic*.

The three-component bowing parameter can be specified in the groups *quaternary_zb{}* or *quaternary_wz{}*. The role of these groups is to associate all component-materials, two-component bowing parameters with a name of the three-component alloy and to define the three-component bowing parameters if some of them are non-zero.

Let'consider $\mathrm{Si_xGe_y\ Sn_{1\text{-}x\text{-}y}}$. The parameters for three material components, Si, Ge, and, Sn need to be defined, as well as up to three sets of constant bowing parameters (or up to six sets of composition dependent bowing parameters), for SiGe, GeSn, and SiSn. The structure of database for this alloy with constant bowing parameters can be as follows.

```
binary_zb{
    name     = Si
    valence = IV_IV

    # All the parameters of Si here (P_A)
}

binary_zb{
    name     = Ge
    valence = IV_IV

    # All the parameters of Ge here (P_B)
}

binary_zb{
    name     = Sn
    valence = IV_IV

    # All the parameters of Sn here (P_C)
}

ternary_zb{
    name        = "Si(x)Ge(1-x)"
    valence     = IV_IV
    binary_x    = Si
    binary_1_x = Ge

    # Optional bowing parameters  (b_AB)
}

ternary_zb{
    name        = "Si(x)Sn(1-x)"
    valence     = IV_IV
    binary_x    = Si
    binary_1_x = Sn

    # Optional bowing parameters  (b_AC)
}

ternary_zb{
    name        = "Ge(x)Sn(1-x)"
    valence     = IV_IV
    binary_x    = Ge
    binary_1_x = Sn

    # Optional bowing parameters  (b_BC)
}

quaternary_zb {
    name        = "Si(x)Ge(y)Sn(1-x-y)"
    valence     = IV_IV
    binary1     = Si
    binary2     = Ge
    binary3     = Sn
    ternary12   = "Si(x)Ge(1-x)"
    ternary13   = "Si(x)Sn(1-x)"
    ternary23   = "Ge(x)Sn(1-x)"
```

(continues on next page)

(continued from previous page)

```
    # Optional bowing parameters  (b_ABC)
}
```

Attention

The following sections are not finished.

6.3.4 Four-component alloys

Four-component alloys with a stoichiometry $A_xB_{1\text{-}x}C_yD_{1\text{-}y}$are typically used only for III-V and II-VI material systems (III-III-V-V and II-II-VI-VI). They are typically called quaternary alloys.

An exemplary alloy is $Ga_xIn_{1\text{-}x}P_xAs_{1\text{-}y}$.

III-III-V-V and II-II-VI-VI

For alloys of type $A_xB_{1\text{-}x}C_yD_{1\text{-}y}$, having $u = 1 - x$ and $v = 1 - y$, the scheme reads

$$\begin{aligned}P_{\mathrm{ABCD}}(x,y) &= xy \cdot P_{\mathrm{AC}} + uy \cdot P_{\mathrm{BC}} + xv \cdot P_{\mathrm{AD}} + uv \cdot P_{\mathrm{BD}} \\ &\quad - xuy \cdot b'_{\mathrm{ABC}}(x,u) - xuv \cdot b'_{\mathrm{ABD}}(x,u) - xyv \cdot b'_{\mathrm{ACD}}(y,v) - uyv \cdot b'_{\mathrm{BCD}}(y,v) \\ &\quad - xuyv \cdot b_{\mathrm{ABCD}}\end{aligned}$$

Groups required like for three-component alloys but instead of using `quaternary_zb{}` one should use `quaternary4_zb{}`.

```
### Indium Aluminum Arsenide Antimonide (InAlAsSb) ###

quaternary4_zb {
    name      = "In(x)Al(1-x)As(y)Sb(1-y)"
    valence   = III_V
    binary1   = InAs
    binary2   = AlAs
    binary3   = AlSb
    binary4   = InSb

    ternary12 = "In(x)Al(1-x)As"   # Note: In(x)Al(1-x)As and In(1-
↪x)Al(x)As are equivalent
    ternary23 = "AlAs(x)Sb(1-x)"   #      as can be seen in the above␣
↪equation.
    ternary34 = "Al(x)In(1-x)Sb"   #      So one has to use the name␣
↪that is already defined in the database.
    ternary14 = "InAs(x)Sb(1-x)"
}
```

IV-IV-IV-IV

For alloys of type $A_xB_yC_zD_{1\text{-}x\text{-}y\text{-}z}$, having $w = 1 - x - y - z$, the scheme reads

$$\begin{aligned}P_{\mathrm{ABCD}}(x,y) &= x \cdot P_{\mathrm{A}} + y \cdot P_{\mathrm{B}} + z \cdot P_{\mathrm{C}} + w \cdot P_{\mathrm{D}} \\ &\quad - xy \cdot b'_{\mathrm{AB}}(x,y) - xz \cdot b'_{\mathrm{AC}}(x,z) - xw \cdot b'_{\mathrm{AD}}(x,w) \\ &\quad - yz \cdot b'_{\mathrm{BC}}(y,z) - yw \cdot b'_{\mathrm{BD}}(y,w) - zw \cdot b'_{\mathrm{CD}}(z,w) \\ &\quad - xyz \cdot b'_{\mathrm{ABC}} - xyw \cdot b'_{\mathrm{ABD}} - xzw \cdot b'_{\mathrm{ACD}} - yzw \cdot b'_{\mathrm{BCD}} \\ &\quad - xyzw \cdot b_{\mathrm{ABCD}}\end{aligned}$$

```
quinternary_zb : _alloy_zb{ TYPE=group OPT=1
```

(continues on next page)

(continued from previous page)

```
    binary_a{ TYPE=string }             #  A
    binary_b{ TYPE=string }             #  B
    binary_c{ TYPE=string }             #  C
    binary_d{ TYPE=string }             #  D

    ternary_ab{ TYPE=string }           #  A(x)B(1-x)
    ternary_ac{ TYPE=string }           #  A(x)C(1-x)
    ternary_ad{ TYPE=string }           #  A(x)D(1-x)
    ternary_bc{ TYPE=string }           #  B(x)C(1-x)
    ternary_bd{ TYPE=string }           #  B(x)D(1-x)
    ternary_cd{ TYPE=string }           #  C(x)D(1-x)

    quaternary_abc{ TYPE=string }       #  A(x)B(y)C(1-x-y)
    quaternary_abd{ TYPE=string }       #  A(x)B(y)D(1-x-y)
    quaternary_acd{ TYPE=string }       #  A(x)C(y)D(1-x-y)
    quaternary_bcd{ TYPE=string }       #  B(x)C(y)D(1-x-y)

    # from base group, optional quinternary bowing parameters
}
```

6.3.5 Six-component alloys

III-III-III-V-V and II-II-II-VI-VI

For alloys of type $\mathrm{A_xB_yC_{1\text{-}x\text{-}y}D_zE_{1\text{-}z}}$, having $u = 1 - x - y$ and $w = 1 - z$, the scheme reads

$$\begin{aligned}
P_{\mathrm{ABCDE}}\left(x,y,z\right) = {} & xz \cdot P_{\mathrm{AD}} + yz \cdot P_{\mathrm{BD}} + uz \cdot P_{\mathrm{CD}} \\
& + xw \cdot P_{\mathrm{AE}} + yw \cdot P_{\mathrm{BE}} + uw \cdot P_{\mathrm{CE}} \\
& - xyz \cdot b'_{\mathrm{ABD}}\left(x,y\right) - xuz \cdot b'_{\mathrm{ACD}}\left(x,u\right) - yuz \cdot b'_{\mathrm{BCD}}\left(y,u\right) \\
& - xyw \cdot b'_{\mathrm{ABE}}\left(x,y\right) - xuw \cdot b'_{\mathrm{ACE}}\left(x,u\right) - yuw \cdot b'_{\mathrm{BCE}}\left(y,u\right) \\
& - xzw \cdot b'_{\mathrm{ADE}}\left(z,w\right) - yzw \cdot b'_{\mathrm{BDE}}\left(z,w\right) - uzw \cdot b'_{\mathrm{CDE}}\left(z,w\right) \\
& - xyzw \cdot b'_{\mathrm{ABDE}} - xuzw \cdot b'_{\mathrm{ACDE}} - yuzw \cdot b'_{\mathrm{BCDE}} \\
& - xyuz \cdot b'_{\mathrm{ABCD}} - xyuw \cdot b'_{\mathrm{ABCE}} \\
& - xyuzw \cdot b_{\mathrm{ABCDE}}
\end{aligned}$$

```
quinternary6_zb : _alloy_zb{ TYPE=group OPT=1

    binary_a_d{ TYPE=string }           #  AD
    binary_b_d{ TYPE=string }           #  BD
    binary_c_d{ TYPE=string }           #  CD
    binary_a_e{ TYPE=string }           #  AE
    binary_b_e{ TYPE=string }           #  BE
    binary_c_e{ TYPE=string }           #  CE

    ternary_ab_d{ TYPE=string }         #  A(x)B(1-x)D
    ternary_ac_d{ TYPE=string }         #  A(x)C(1-x)D
    ternary_bc_d{ TYPE=string }         #  B(x)C(1-x)D
    ternary_ab_e{ TYPE=string }         #  A(x)B(1-x)E
    ternary_ac_e{ TYPE=string }         #  A(x)C(1-x)E
    ternary_bc_e{ TYPE=string }         #  B(x)C(1-x)E
    ternary_a_de{ TYPE=string }         #  AD(x)E(1-x)
    ternary_b_de{ TYPE=string }         #  BD(x)E(1-x)
    ternary_c_de{ TYPE=string }         #  CD(x)E(1-x)

    quaternary_abc_d{ TYPE=string }     #  A(x)B(y)C(1-x-y)D
```

(continues on next page)

(continued from previous page)

```
    quaternary_abc_e{ TYPE=string }      #  A(x)B(y)C(1-x-y)E
    quaternary_ab_de{ TYPE=string }      #  A(x)B(1-x)D(y)E(1-y)
    quaternary_ac_de{ TYPE=string }      #  A(x)C(1-x)D(y)E(1-y)
    quaternary_bc_de{ TYPE=string }      #  B(x)C(1-x)D(y)E(1-y)

    # from base group, optional quinternary bowing parameters
}
```

6.3.6 Eight-component alloys

$A_xB_{1-x}C_yD_{1-y}E_zF_{1-z}$

```
quaternary8_zb : _alloy_zb{ TYPE=group OPT=1

    binary_a_c_e{ TYPE=string }          #  ACE
    binary_b_c_e{ TYPE=string }          #  BCE
    binary_a_d_e{ TYPE=string }          #  ADE
    binary_b_d_e{ TYPE=string }          #  BDE
    binary_a_c_f{ TYPE=string }          #  ACF
    binary_b_c_f{ TYPE=string }          #  BCF
    binary_a_d_f{ TYPE=string }          #  ADF
    binary_b_d_f{ TYPE=string }          #  BDF

    ternary_ab_c_e{ TYPE=string }        #  A(x)B(1-x)CE
    ternary_ab_d_e{ TYPE=string }        #  A(x)B(1-x)DE
    ternary_ab_c_f{ TYPE=string }        #  A(x)B(1-x)CF
    ternary_ab_d_f{ TYPE=string }        #  A(x)B(1-x)DF
    ternary_a_cd_e{ TYPE=string }        #  AC(x)D(1-x)E
    ternary_b_cd_e{ TYPE=string }        #  BC(x)D(1-x)E
    ternary_a_cd_f{ TYPE=string }        #  AC(x)D(1-x)F
    ternary_b_cd_f{ TYPE=string }        #  BC(x)D(1-x)F
    ternary_a_c_ef{ TYPE=string }        #  ACE(x)F(1-x)
    ternary_b_c_ef{ TYPE=string }        #  BCE(x)F(1-x)
    ternary_a_d_ef{ TYPE=string }        #  ADE(x)F(1-x)
    ternary_b_d_ef{ TYPE=string }        #  BDE(x)F(1-x)

    quarternary_ab_cd_e{ TYPE=string }   #  A(x)B(1-x)C(y)D(1-y)E
    quarternary_ab_cd_f{ TYPE=string }   #  A(x)B(1-x)C(y)D(1-y)F
    quarternary_ab_c_ef{ TYPE=string }   #  A(x)B(1-x)CE(y)F(1-y)
    quarternary_ab_d_ef{ TYPE=string }   #  A(x)B(1-x)DE(y)F(1-y)
    quarternary_a_cd_ef{ TYPE=string }   #  AC(x)D(1-x)E(y)F(1-y)
    quarternary_b_cd_ef{ TYPE=string }   #  BC(x)D(1-x)E(y)F(1-y)

    # from base group, optional quinternary bowing parameters
}
```

Note

If you need other interpolation schemes for your research, raise a support ticket attaching formulas of your interest, related references, and explanation why it's valuable.

A brief introduction to quaternaries is shown in this Powerpoint presentation (Quaternaries.pptx , Quaternaries.pdf).

Last update: nnnn/nn/nn

6.4 Default Materials and Alloys

Following zincblende (cubic crystal structure) and wurtzite (hexagonal crystal structure) materials are parametrized in our defyult material database *database_nnp.in*:

Note

Synonyms are supported.

Examples:

- Si(1-x)Ge(x) ≡ Ge(x)Si(1-x) ≡ Si(x)Ge(1-x) ≡ Ge(1-x)Si(x)
- Sapphire ≡ Al2O3

6.4.1 Insulators and Metals

- SiO_2
- HfO_2
- Air
- Air_wz
- Al_2O_3 (sapphire)

6.4.2 Binary alloys

IV - IV

Elements

- C
- Si
- Ge
- Sn

Silicon-based

- SiC
- SiC-4H
- SiC-6H

III - V

Arsenides

- GaAs

- AlAs
- InAs

Phophides

- GaP
- AlP
- InP

Antimonides

- GaSb
- AlSb
- InSb

Nitrides

- ScN
- YN
- GaN
- GaN_zb
- AlN
- AlN_zb
- InN
- InN_zb

Other

- GaBi

II - VI

Oxides

- ZnO_wz
- ZnO
- CdO_wz
- MgO_wz

Tellurides

- HgTe
- MgTe
- ZnTe
- BeTe
- MnTe
- MnTe_zb
- CdTe

Selenides

- ZnSe
- MgSe
- CdSe

- BeSe
- MnSe
- MnSe_zb

Sulfides

- ZnS
- CdS

6.4.3 Ternary alloys

IV - IV

- $Si_{1-x}Ge_x$
- $Ge_{1-x}Sn_x$
- $Si_{1-x}Sn_x$

III - V Valence

Arsenides

- $Al_xGa_{1-x}As$
- $In_xGa_{1-x}As$
- $Al_xIn_{1-x}As$

Phosphides

- $Ga_xIn_{1-x}P$
- $Al_xIn_{1-x}P$
- $Al_xGa_{1-x}P$

Antimonides

- $Ga_xIn_{1-x}Sb$
- $Al_xIn_{1-x}Sb$
- $Al_xGa_{1-x}Sb$

Nitrides

- $In_xGa_{1-x}N$
- $In_xGa_{1-x}N$_zb
- $Al_xGa_{1-x}N$
- $Al_xGa_{1-x}N$_zb
- $Al_xIn_{1-x}N$
- $Al_xIn_{1-x}N$_zb
- $Al_xSc_{1-x}N$
- $Al_xY_{1-x}N$
- $Sc_xGa_{1-x}N$
- $Y_xGa_{1-x}N$
- $Sc_xIn_{1-x}N$
- $Y_xIn_{1-x}N$
- $Y_xSc_{1-x}N$

Arsenides - Antimonides

- $GaAs_{1-x}Sb_x$
- $InAs_xSb_{1-x}$
- $AlAs_xSb_{1-x}$

Arsenides - Phosphides

- $GaAs_{1-x}P_x$
- $InAs_xP_{1-x}$
- $AlAs_xP_{1-x}$

Phosphides - Antimonides

- GaP_xSb_{1-x}
- InP_xSb_{1-x}
- AlP_xSb_{1-x}

Dilude Nitrides

- $GaAs_{1-x}N_x$
- $InAs_{1-x}N_x$
- $AlAs_{1-x}N_x$
- $GaP_{1-x}N_x$
- $InP_{1-x}N_x$
- $AlP_{1-x}N_x$
- $GaSb_{1-x}N_x$
- $InSb_{1-x}N_x$
- $AlSb_{1-x}N_x$

Others

- $GaAs_{1-x}Bi_x$
- Zn_{1-x} Mg_xS

II - VI

Selenides

- $Be_xZn_{1-x}Se$
- $Be_xCd_{1-x}Se$
- Zn_{1-x} Mg_xSe
- $Cd_xZn_{1-x}Se$
- $Be_{1-x}Mn_xSe$
- $Cd_{1-x}Mn_xSe$
- $Zn_{1-x}Mn_xSe$

Tellurides

- $Be_xZn_{1-x}Te$
- $Cd_{1-x}Mg_xTe$
- $Cd_xZn_{1-x}Te$
- $Hg_{1-x}Cd_xTe$

- $Cd_{1-x}Mn_xTe$
- $Zn_{1-x}Mn_xTe$

Oxides

- $Mg_xZn_{1-x}O$
- $Cd_xZn_{1-x}O$

Others

- ZnS_xSe_{1-x}
- $Zn_xCd_{1-x}S$

6.4.4 Quaternary alloys

IV - IV

- $Si_{1-x-y}Ge_xSn_y$

III - V

III-III-III-V materials

- $Al_xGa_yIn_{1-x-y}N$ (wz)
- $Al_xGa_yIn_{1-x-y}N$ (zb)
- $Al_xGa_yIn_{1-x-y}P$
- $Al_xGa_yIn_{1-x-y}As$
- $Al_xGa_yIn_{1-x-y}Sb$
- $Al_xSc_yGa_{1-x-y}N$
- $Al_xSc_yIn_{1-x-y}N$
- $Sc_xIn_yGa_{1-x-y}N$
- $Al_xY_yGa_{1-x-y}N$
- $Al_xY_yIn_{1-x-y}N$
- $Y_xIn_yGa_{1-x-y}N$
- $Y_xSc_yGa_{1-x-y}N$
- $Y_xSc_yAl_{1-x-y}N$
- $Y_xSc_yIn_{1-x-y}N$

III-V-V-V materials

- $AlAs_xSb_yP_{1-x-y}$
- $GaAs_xSb_yP_{1-x-y}$
- $InAs_xSb_yP_{1-x-y}$
- $GaAs_xSb_yN_{1-x-y}$

III-III-V-V materials

- $Ga_xIn_{1-x}As_yP_{1-y}$
- $Al_xGa_{1-x}As_yP_{1-y}$
- $In_xAl_{1-x}As_yP_{1-y}$
- $In_xGa_{1-x}As_yN_{1-y}$
- $Ga_xIn_{1-x}As_ySb_{1-y}$

- $Al_xGa_{1-x}As_ySb_{1-y}$
- $In_xAl_{1-x}As_ySb_{1-y}$

II - VI

- $Zn_{1-x-y}Be_xMn_ySe$

6.4.5 Quinternary alloys

III - V

- $Al_xGa_yIn_{1-x-y}As_zSb_{1-z}$
- $Al_xGa_yIn_{1-x-y}As_zP_{1-z}$
- $Sc_xIn_yAl_zGa_{1-x-y-z}N$
- $Y_xIn_yAl_zGa_{1-x-y-z}N$

Last update: nnnn/nn/nn

6.5 Definition of Band Offsets (zincblende)

This section explains how band offsets are evaluated in *nextnano*++. It begins with showing connection between parameters used in **database{ …{ conduction_bands{} } }** (see *valence_bands{}* and *conduction_bands{}*) and band energies at their extrema. Then, various band alignments and exemplary interpolations, with and without strain, are presented. All plots are computed for materials at **300 K**.

Schematics of band structure in vicinity of the Γ point is shown in Figure 6.5.1. Energies of band extrema in the case of lack of strain are depicted with gray lines and labels, while the parameters stored in database are plotted in black color.

The starting point of defining the offset in *nextnano*++ is the average valence band energy $E_{v,av}$ which can be modified by using *bandoffset* attribute. Formally, it is defined as the average energy of three top valence bands

$$E_{v,av} = \frac{1}{3}(E_{hh} + E_{lh} + E_{so})$$

where E_{hh}, E_{lh}, and E_{so} are energies of heavy-hole, light-hole, and split-off bands at Γ point, respectively. In the case without strain, $E_{v,av}$ is located $\frac{1}{3}\Delta_{so}$ below the top of the valence band. The spin-orbit splitting energy Δ_{so} and the energy gap at the Γ point E_g^Γ are available through attributes *delta_SO* and *bandgap*. Depending on the group to which the *bandgap* attribute belongs to, it may refer to energy differences involving conduction band minima at Γ, $L(\Lambda)$, or $X(\Delta)$ points (lines).

One has to specify all three parameters (including Varshni's parameters for temperature dependence of E_g^Γ and other gaps) for every material of interest to define whole band alignments. Our *database* contains and provide space to contain these parameters and related bowing parameters for all specified materials listed *here*.

It is important to keep in mind that offsets of bands are not easy-to-measure parameters, so their values are typically provided by simulations within *ab-initio* approaches. Therefore, for fine simulations, we advise to always verify all the material parameters and adjust them. Our database already consists of numerous published material parameters resulting in the offsets as visible in Figure 6.5.2 and Figure 6.5.3.

To obtain band alignments for alloys, the three parameters ($E_{v,av}$, Δ_{so}, and E_g^Γ) are properly *interpolated* and further used to provide minima of conduction bands (E_c^Γ, E_c^X, E_c^L, …) and maxima of valence bands (E_{hh}, E_{lh}, and E_{so}) according to formulas:

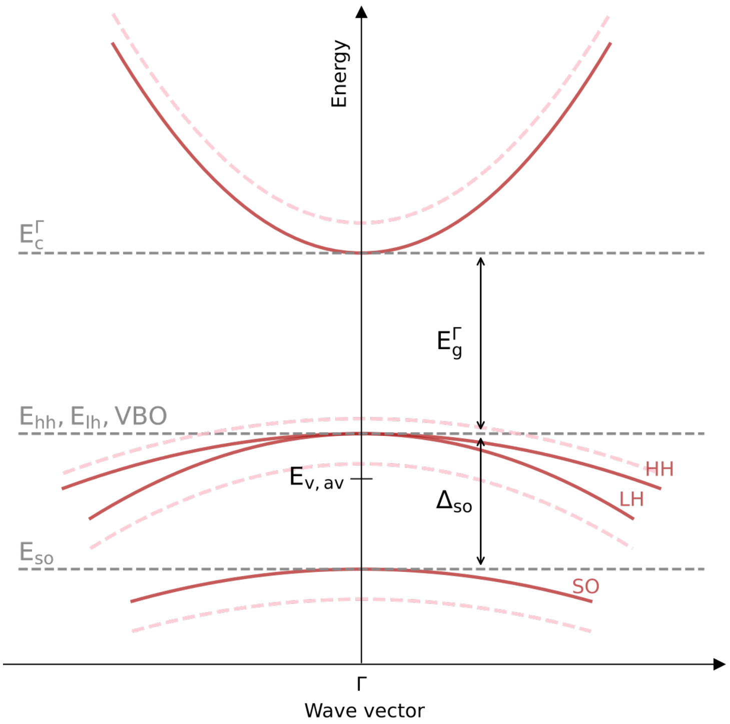


Figure 6.5.1: Band structure of freestanding (red solid lines) and compressively strained (pink dashed lines) in vicinity of Γ point.

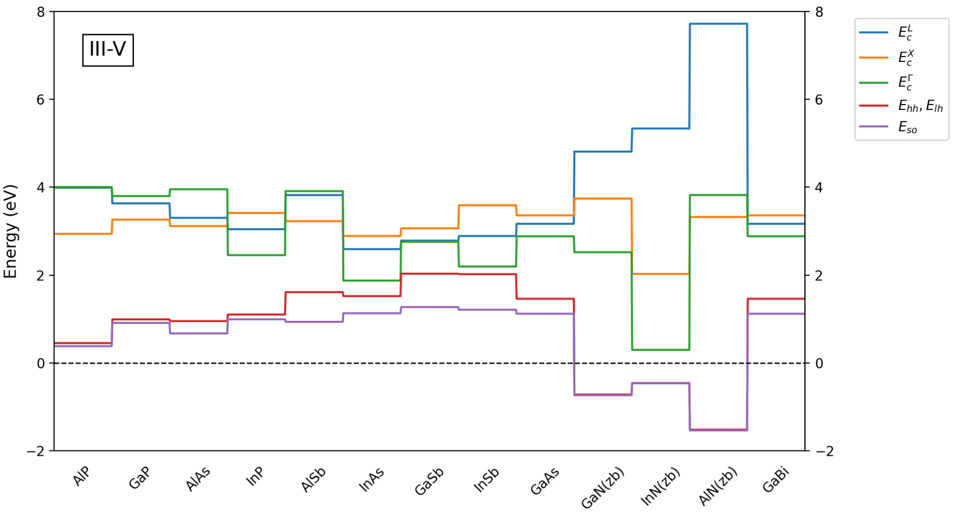


Figure 6.5.2: Band offsets of III-V zincblende binary compounds calculated with default parameters predefined in our database

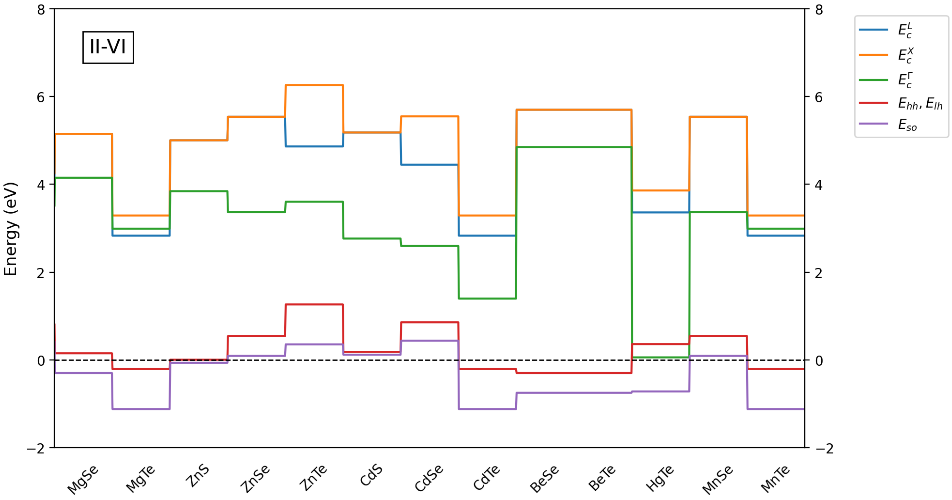


Figure 6.5.3: Band offsets of II-VI zincblende binary compounds calculated with default parameters predefined in our database

$$\begin{aligned}
E_c^X &= E_{v,av} + \frac{1}{3}\Delta_{so} + E_g^X \\
E_c^L &= E_{v,av} + \frac{1}{3}\Delta_{so} + E_g^L \\
E_c^\Gamma &= E_{v,av} + \frac{1}{3}\Delta_{so} + E_g^\Gamma \\
E_{hh} = E_{lh} &= E_{v,av} + \frac{1}{3}\Delta_{so} \\
E_{so} &= E_{v,av} - \frac{2}{3}\Delta_{so}
\end{aligned}$$

Attention

The parameters in the database, such as $E_{v,av}$, Δ_{so}, and E_g^Γ, are defined for freestanding bulk crystals (without any strain), while final band energies, like E_{hh}, E_{lh}, E_{so}, E_c^Γ, do include strain effects if proper conditions are met. Therefore, for example, in strained structures one should expect that $E_g^\Gamma \neq E_c^\Gamma - E_{hh}$ and $E_g^\Gamma \neq E_c^\Gamma - E_{lh}$.

Plots of resulting band energies for three chosen alloys ($\mathrm{Ga_xIn_{1-x}As}$, $\mathrm{Al_xGa_{1-x}As}$, and $\mathrm{In_xAl_{1-x}As}$) within full mole fraction ranges are shown in Figure 6.5.4. As visible for $\mathrm{Al_xGa_{1-x}As}$, content-dependent bowing parameters are also available in our routines. All the parameters necessary to compute strain effects are included in the algorithm in the similar manner. They are interpolated first and then applied to evaluate energy shifts of band energies.

Last update: nnnn/nn/nn

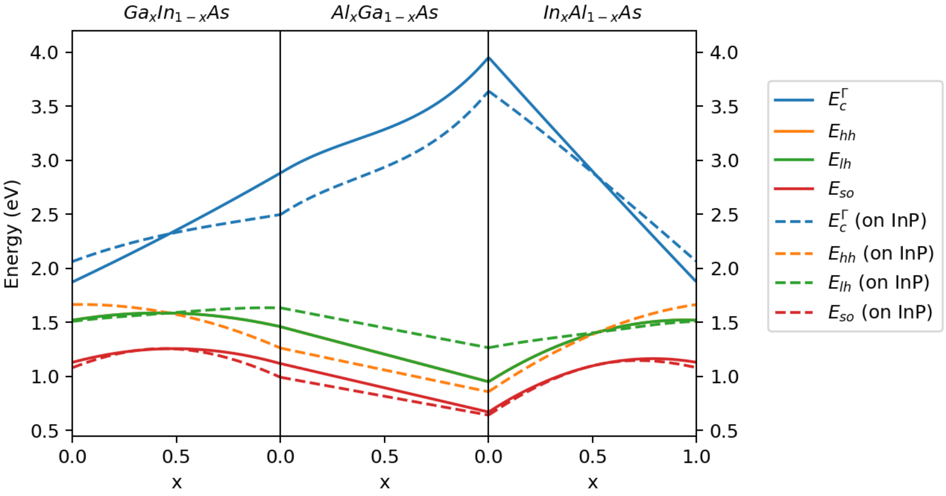


Figure 6.5.4: Interpolated band edges of $\mathrm{Ga_xIn_{1-x}As}$, $\mathrm{Al_xGa_{1-x}As}$, and $\mathrm{In_xAl_{1-x}As}$ without strain (solid lines) and strained as grown on [1 0 0] plane of InP (dashed lines).

CHAPTER

SEVEN

KEYWORDS

7.1 postprocessor{ }

Calling sequence

`postprocessor{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- At least one of *postprocessor{ datafile }* and *postprocessor{ call }* must be specified within this group.

Functionality

A group allowing to run post-processing automatically after the simulation is done.

Attention

This group is ignored by default. It is active only when *nextnano*++ is run with an option `--postprocessor`.

Examples

```
postprocessor{
    datafile = "query.bat"
    call = "query.bat"
    goto_output = yes
}

!DATA
# some list of commands here
```

Nested keywords

- *datafile*
- *goto_output*
- *call*

7.1.1 datafile

Calling sequence

`postprocessor{ datafile }`

Properties

- **usage:** optional
- **type:** character string

Functionality

If `datafile` is defined, then a file `datafile` is created in the output directory. The content of the `!DATA` section, if it exists, will be written into this file. Possible content in the `!DATA` section could be, e.g., comments, copyright or user info, or scripts in Python, Julia, Bash, Cmd, etc.

Example

```
postprocessor{
    datafile = "query.bat"
}

!DATA

dir
```

7.1.2 goto_output

Calling sequence

`postprocessor{ goto_output }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If `goto_output = yes` then the shell command defined by `call` will be launched from within the output directory. Otherwise, the directory from where *nextnano++* has been launched will be used.

> ⚠ **Warning**
>
> Setting `goto_output = no` may cause conflicts between jobs when running multiple jobs in parallel e.g. in *nextnanomat* or through a batch system such as HTCondor or Slurm.

Example

```
postprocessor{
    goto_output = no
    call = dir
}
```

7.1.3 call

Calling sequence

`postprocessor{ call }`

Properties

- **usage:** optional
- **type:** character string

Functionality

If `call` is defined, then it is used as a shell command line, typically cmd on Windows and bash on Linux, which will be launched. This command line can, but does not have to, refer to a file defined by `datafile`.

> ⚠ **Attention**
>
> Calling GUI based programs such as ParaView is also possible but may interfere with operation of job control tools such as *nextnanomat* or *nextnanopy*, as the job will only be considered finished once also all the post-processing tasks are finished.

> ℹ **Note**
>
> If *nextnano*++ is running through a batch system such as HTCondor or Slurm, the postprocessing is executed on the respective destination computer using the file systems available there.

Example

```
postprocessor{
    goto_output = yes
    call = dir
}
```

7.2 import{ }

Calling sequence

`import{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- At least one of *analytic_function{ }* and *file{ }* must be present if *output_imports{ }* is defined.

Functionality

Specifications for importing data from a file or generating them from an analytic function, e.g. electrostatic potential, alloy profile, strain profile, doping profile, generation rate profile, electron or hole Fermi level profile.

Once a file has been imported or a function has been defined, it can be used several times, e.g. the same file could include the alloy concentration of a ternary for different region objects.

Data with dimensionality deviating from the simulation dimension can also be imported, e.g. an absorption spectrum for solar cell modeling.

Examples

```
import{
    file{...}
    output_imports{}
}
```

```
import{
    analytic_function{...}
    output_imports{}
}
```

Nested keywords

- *directory*
- *file{ }*
- *file{ name }*
- *file{ filename }*
- *file{ format }*
- *file{ scale }*
- *file{ number_of_dimensions }*
- *analytic_function{ }*
- *analytic_function{ name }*
- *analytic_function{ function }*
- *analytic_function{ label }*
- *analytic_function{ component{ } }*
- *analytic_function{ component{ function_i } }*
- *analytic_function{ component{ label } }*
- *output_imports{ }*

7.2.1 directory

Calling sequence

`import{ directory }`

Properties

- **usage:** optional
- **type:** character string
- **default:** `empty`

Functionality

Name of directory where files to be imported are located (if data are imported from files)

Example

```
import{
    directory = "D:\\import_files\\"
    file{...}
}
```

7.2.2 file{ }

Calling sequence

`import{ file{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

—

Example

```
import{
    file{...}
}
```

7.2.3 file{ name }

Calling sequence

`import{ file{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Name for referencing the imported data in the input file, e.g. “imported_potential_profile_2D”

Example

```
import{
    file{
        name = "1D_import"
        ...
    }
}
```

7.2.4 file{ filename }

Calling sequence

`import{ file{ filename } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Name of file which is imported. Three ways of using are available.

One can define an absolute path to a file, e.g., `"D:\\precious_data.dat"`. If so then *directory* is ignored if specified.

If the path is not specified here, e.g., `"precious_data.dat"` then the file must be located in the directory specified by *directory*.

When neither path is specified here, e.g., `"precious_data.dat"`, nor the *directory* is defined, then the file must be located in the directory of the input file

Examples

```
import{
    file{
        name = "1D_import"
        filename = "D:\\precious_data.dat"
        ...
    }
}
```

```
import{
    directory = "D:\\"
    file{
        name = "1D_import"
        filename = "precious_data.dat"
        ...
    }
}
```

```
import{
    file{
        name = "1D_import"
        filename = "precious_data.dat"
        ...
```

(continues on next page)

(continued from previous page)

```
    }
}
```

7.2.5 file{ format }

Calling sequence

`import{ file{ format } }`

Properties

- **usage:** required
- **type:** choice
- **values:** `AVS` or `DAT`

Functionality

Format of the file to be imported. Formats `.fld` and `.dat` are supported for options `AVS` and `DAT`, respectively.

Example

```
import{
    directory = "D:\\"
    file{
        name = "1D_import"
        filename = "precious_data.dat"
        format = DAT
    }
}
```

7.2.6 file{ scale }

Calling sequence

`import{ file{ scale } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 1.0$
- **unit:** $-$

Functionality

A factor used to multiply the imported data. Can be used to change units of imported data for consistency with *nextnano*++, e.g., conversion from `J` to `eV`.

Examples

```
import{
    directory = "D:\\"
    file{
        name = "1D_import"
        filename = "precious_data.dat"
        format = DAT
        scale = 1.6022e-19
    }
}
```

```
import{
    directory = "D:\\"
    file{
        name = "1D_import"
        filename = "precious_data.dat"
        format = DAT
        scale = -1
    }
}
```

7.2.7 file{ number_of_dimensions }

Calling sequence

`import{ file{ number_of_dimensions } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 3$
- **default:** `simulation dimension`
- **unit:** –

Functionality

Explicit specification of the number of dimensions of the space onto which the data is defined. Can be only used for `.dat` files.

Example

```
import{
    directory = "D:\\"
    file{
        name = "1D_import"
        filename = "precious_spectra.dat"
        format = DAT
        number_of_dimensions = 1
    }
}
```

7.2.8 analytic_function{ }

Calling sequence

`import{ analytic_function{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Dependencies

- At least one of *analytic_function{ component{ } }* and *analytic_function{ function }* must be defined.
- *analytic_function{ component{ } }* and *analytic_function{ function }* cannot be defined together.
- *analytic_function{ label }* cannot be defined if *analytic_function{ component{ } }* as already present.

Functionality

Defines analytic functions to be imported here. Does not need to be defined if data are imported from files.

Example

```
import{
    analytic_function{
        name = "function_1"
        component{...}
    }
    analytic_function{
        name = "function_2"
        function = ...
    }
    analytic_function{
        name = "function_3"
        function = ...
        label = ...
    }
}
```

7.2.9 analytic_function{ name }

Calling sequence

`import{ analytic_function{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Name for referencing the imported function in the input file.

Example

```
import{
    analytic_function{
        name = "Distribution_FD"
        function = ...
    }
}
```

7.2.10 analytic_function{ function }

Calling sequence

`import{ analytic_function{ function } }`

Properties

- **usage:** optional
- **type:** character string

Functionality

String defining the function in case only one component needs to be defined, otherwise use component.

Attention

One should use the syntax allowed for functions:

- white spaces are ignored
- valid operators are "+", "-", "*", "/" and "^"
- multiplication signs always have to be spelled out (i.e. "5*x" is valid, "5x" is not)
- variable names are fixed to "x", "y" and "z" (capital letters are also allowed)
- additional functions also available (e.g. "exp" , "sqrt", "sin", see full list below), have to be followed by brackets ("exp(x)" is valid, "exp x" is not)
- global variables are allowed if preceded by "$" (e.g. "$PI")
- exponential notation ("2e-3" or "4E10") is allowed

See also table at the bottom of this site.

Example

```
import{
    analytic_function{
        name = "Distribution_FD"
        function = 1/(exp(x) + 1)
    }
}
```

7.2.11 analytic_function{ label }

Calling sequence

`import{ analytic_function{ label } }`

Properties

- **usage:** optional
- **type:** character string

Functionality

Label to be displayed in legend in case only one component is defined. If it's not defined then, *analytic_function{ name }* is displayed.

Example

```
import{
    analytic_function{
        name = "Distribution_FD"
        function = 1/(exp(x) + 1)
        label = "Fermi Dirac"
    }
}
```

7.2.12 analytic_function{ component{ } }

Calling sequence

`import{ analytic_function{ component{ } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

In case multiple components are needed, define one component group for each component.

Example

```
import{
    analytic_function{
        name = "Distributions"
        component{...}
        component{...}
    }
}
```

7.2.13 analytic_function{ component{ function_i } }

Calling sequence

`import{ analytic_function{ component{ function_i } } }`

Properties

- **usage:** optional
- **type:** character string

Functionality

String defining the function for this component.

Example

```
import{
    analytic_function{
        name = "Distributions"
        component{
            function_i = 1/(exp(x) + 1)
        }
        component{
            function_i = 1/(exp(x) - 1)
        }
    }
}
```

7.2.14 analytic_function{ component{ label } }

Calling sequence

`import{ analytic_function{ component{ label } } }`

Properties

- **usage:** optional
- **type:** character string

Functionality

Label to be displayed in legend for this component.

Example

```
import{
    analytic_function{
        name = "Distributions"
        component{
            function_i = 1/(exp(x) + 1)
            label = "Fermi-Dirac"
        }
        component{
            function_i = 1/(exp(x) - 1)
            label = "Bose-Einstein"
        }
    }
}
```

7.2.15 output_imports{ }

Calling sequence

`import{ output_imports{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output all imported data including scale factor. The filenames correspond to the entry given in `name = ...`. The files will be written to a folder called `Imports/`.

Example

```
import{
    file{...}
    analytic_function{...}
    output_imports{}
}
```

Operators and Functions supported by `analytic_function{}` group, sorted with decreasing precedence:

Operators

power (exponentiation)	`^`
multiplication, division	`*` `/`
plus and minus	`+` `-`
round arithmetic brackets	`( )`

Functions

`sqrt()`	square root $\sqrt{}$
`cbrt()`	cubic root $\sqrt[3]{}$
`exp()`	exponential function $\exp(\,)$
`log()`	natural logarithm $\log$
`ln()`	natural logarithm $\ln$
`log2()`	decadic logarithm (base 2) $\log_2$
`log10()`	decadic logarithm (base 10) $\log_{10}$
`sin()`	sine $\sin(\,)$
`cos()`	cosine $\cos(\,)$
`tan()`	tangent $\tan(\,)$
`asin()`	acrsine $\sin^{-1}(\,)$
`acos()`	arccosine $\cos^{-1}(\,)$
`atan()`	arctangent $\tan^{-1}(\,)$
`sinh()`	hyperbolic sine $\sinh(\,)$
`cosh()`	hyperbolic cosine $\cosh(\,)$
`tanh()`	hyperbolic tangent $\tanh(\,)$
`asinh()`	inverse hyperbolic sine $\sinh^{-1}(\,)$
`acosh()`	inverse hyperbolic cosine $\cosh^{-1}(\,)$
`atanh()`	inverse hyperbolic tangent $\tanh^{-1}(\,)$
`erf()`	error function $\mathrm{erf}(\,)$

continues on next page

Table 7.2.15.1 – continued from previous page

`erfc()`	complementary error function erfc( )
`gamma()`	Gamma function Γ( )
`fdm3half()`	complete Fermi–Dirac integral $F_{-3/2}()$ of order -3/2 (includes the $1/\Gamma(-1/2)$ prefactor)
`fdmhalf()`	complete Fermi–Dirac integral $F_{-1/2}()$ of order -1/2 (includes the $1/\Gamma(1/2)$ prefactor)
`fdzero()`	complete Fermi–Dirac integral $F_0()$ of order 0 (includes the $1/\Gamma(1) = 1$ prefactor)
`fdphalf()`	complete Fermi–Dirac integral $F_{1/2}()$ of order 1/2 (includes the $1/\Gamma(3/2)$ prefactor)
`fdp3half()`	complete Fermi–Dirac integral $F_{3/2}()$ of order 3/2 (includes the $1/\Gamma(5/2)$ prefactor)
`abs()`	absolute value | |
`floor()`	floor function floor(x): largest integer $\leq x$
`ceil()`	ceiling function ceil(x): smallest integer $\geq x$
`round()`	rounds the number to the nearest integer
`sign()`	sign function
`heaviside()`	Heaviside step function (corresponds to `isnotnegative()`)
`ispositive()`	check if value is positive
`isnegative()`	check if value is negative
`iszero()`	check if value is zero
`isnotpositive()`	check if value is not positive
`isnotnegative()`	check if value is not negative (corresponds to `heaviside()`)
`isnotzero()`	check if value is not zero

7.3 output{ }

Calling sequence

`output{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Sets options for the output data and controls additional output of material parameters.

Example

```
output{...}
```

Nested keywords

- *directory*
- *mandatory_path*
- *set_origin{ }*
- *set_origin{ x }*
- *set_origin{ y }*
- *set_origin{ z }*
- *format2D*
- *format3D*

- *silent*
- *write_avs_v*
- *write_origin_plt*
- *write_gnuplot_plt*
- *use_gnuplot_one_file*
- *only_sections*
- *section{ }*
- *section{ name }*
- *section{ range_x }*
- *section{ range_y }*
- *section{ range_z }*
- *section1D{ }*
- *section1D{ name }*
- *section1D{ x }*
- *section1D{ y }*
- *section1D{ z }*
- *section1D{ range_x }*
- *section1D{ range_y }*
- *section1D{ range_z }*
- *section2D{ }*
- *material_parameters{ }*
- *material_parameters{ kp_parameters{ } }*
- *material_parameters{ kp_parameters{ boxes } }*
- *material_parameters{ spin_orbit_coupling_energies{ } }*
- *material_parameters{ spin_orbit_coupling_energies{ boxes } }*
- *material_parameters{ charge_carrier_masses{ } }*
- *material_parameters{ charge_carrier_masses{ boxes } }*
- *material_parameters{ static_dielectric_constants{ } }*
- *material_parameters{ static_dielectric_constants{ boxes } }*
- *material_parameters{ deformation_potentials{ } }*
- *material_parameters{ deformation_potentials{ boxes } }*

7.3.1 directory

Calling sequence

```
output{ directory }
```

Properties

- **usage:** optional
- **type:** character string

Functionality

Defines alternative output directory. Using this path is controlled by *mandatory_path*

Example

```
output{
    directory = "../output/the_best_simulation"
}
```

7.3.2 mandatory_path

Calling sequence

`output{ mandatory_path }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `mandatory_path = yes` then the (relative or absolute) output directory specified by *directory* is used, and any directory specified in the command line is ignored (as, e.g., done by *nextnanomat*).

If `mandatory_path = no` then the directory specified in the command line is used as base path to which a relative path specified in directory then is appended. In this case an absolute path specified in *directory* is ignored.

In all cases, a subdirectory named as the input file is further appended to the output path, unless `-n` or `--noautooutdir` is set as *command line option* (*nextnanomat* sets this option automatically).

Also note that the location of the log (`*.log`) file is not affected by these settings.

Warning

Please make sure that a mandatory output directory is set such that no important files (or the input directory) are overwritten. Be especially careful when accepting input files from others, and do not run simulations using administrative privileges.

7.3.3 set_origin{ }

Calling sequence

`output{ set_origin{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Defines origin of coordinate system of the output files within the coordinate system of the simulation. If the origin of the output coordinate system is set to r_{ori}, then every vector in the simulation coordinate system r_{sim} is transformed to

$$r_{\mathrm{out}} = r_{\mathrm{sim}} - r_{\mathrm{ori}}$$

for every output file with results dependent on position.

7.3.4 set_origin{ x }

Calling sequence

`output{ set_origin{ x } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** nm
- **default:** $z = 0$

Functionality

Defines x-coordinate of the origin of the output coordinate system r_{ori} within the coordinate system of the simulation.

7.3.5 set_origin{ y }

Calling sequence

`output{ set_origin{ y } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** nm
- **default:** $z = 0$

Functionality

Defines y-coordinate of the origin of the output coordinate system r_{ori} within the coordinate system of the simulation.

7.3.6 set_origin{ z }

Calling sequence

`output{ set_origin{ z } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** nm
- **default:** $z = 0$

Functionality

Defines z-coordinate of the origin of the output coordinate system r_{ori} within the coordinate system of the simulation.

7.3.7 format2D

Calling sequence

`output{ format2D }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **values:** `AvsBinary`; `AvsAscii`; `AvsBinary\_one\_file`; `AvsAscii\_one\_file`; `VtkAscii`; `VtkAscii\_AvsAscii`; `VtkAscii\_AvsAscii\_one\_file`; `VtkAscii\_AvsBinary`; `VtkAscii\_AvsBinary\_one\_file`; `Origin`
- **default:** `AvsBinary_one_file`

Functionality

Sets format of output files with data defined on 2-dimensional spaces of any kind.

Note

Instead of `Vtk` one can write `VTK`. Likewise, `Avs` can be replaced by `AVS`.

Table 7.3.7.1: Output file format for data on N-dimensional spaces

Chosen option	Format
`AvsBinary` ...	AVS/Express file format (AVS steering files `*.v`, and `*.fld`, `*.coord`, `*.dat` data files) - data files in binary format
`AvsAscii` ...	AVS/Express file format (AVS steering files `*.v`, and `*.fld`, `*.coord`, `*.dat` data files) - data files in ASCII format
`AvsBinary_one_file` ...	AVS/Express file format - header (ASCII), coordinates and variables (both binary) are written into a single `.fld` file
`AvsAscii_one_file` ...	AVS/Express file format - header (ASCII), coordinates and variables (both ASCII) are written into a single `.fld` file
`VTKAscii` ...	VTK XML ASCII format (`.vtr`, `r` = rectilinear grid)
`VTKAscii_AvsAscii` ...	`VTKAscii` + `AvsAscii`
`VTKAscii_AvsAscii_one_` ...	`VTKAscii` + `AvsAscii_one_file`
`VTKAscii_AvsBinary` ...	`VTKAscii` + `AvsBinary`
`VTKAscii_AvsBinary_one` ...	`VTKAscii` + `AvsBinary_one_file`
`Origin`	Origin file format (Origin steering files `*.plt`, data files `*.dat`)

7.3.8 format3D

Calling sequence

`output{ format3D }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **values:** `AvsBinary`; `AvsAscii`; `AvsBinary\_one\_file`; `AvsAscii\_one\_file`; `VtkAscii`; `VtkAscii\_AvsAscii`; `VtkAscii\_AvsAscii\_one\_file`; `VtkAscii\_AvsBinary`; `VtkAscii\_AvsBinary\_one\_file`; `Origin`
- **default:** `AvsBinary_one_file`

Functionality

Sets format of output files with data defined on 3-dimensional spaces of any kind.

Note

Instead of `Vtk` one can write `VTK`. Likewise, `Avs` can be replaced by `AVS`.

Table 7.3.8.1: Output file format for data on N-dimensional spaces

Chosen option	Format
`AvsBinary` …	AVS/Express file format (AVS steering files `*.v`, and `*.fld`, `*.coord`, `*.dat` data files) - data files in binary format
`AvsAscii` …	AVS/Express file format (AVS steering files `*.v`, and `*.fld`, `*.coord`, `*.dat` data files) - data files in ASCII format
`AvsBinary_one_file` …	AVS/Express file format - header (ASCII), coordinates and variables (both binary) are written into a single `.fld` file
`AvsAscii_one_file` …	AVS/Express file format - header (ASCII), coordinates and variables (both ASCII) are written into a single `.fld` file
`VTKAscii` …	VTK XML ASCII format (`.vtr`, `r` = rectilinear grid)
`VTKAscii_AvsAscii` …	`VTKAscii` + `AvsAscii`
`VTKAscii_AvsAscii_one_` …	`VTKAscii` + `AvsAscii_one_file`
`VTKAscii_AvsBinary` …	`VTKAscii` + `AvsBinary`
`VTKAscii_AvsBinary_one` …	`VTKAscii` + `AvsBinary_one_file`
`Origin`	Origin file format (Origin steering files `*.plt`, data files `*.dat`)

7.3.9 silent

Calling sequence

`output{ silent }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If set to `no` then prints additional warnings concerning output.

7.3.10 write_avs_v

Calling sequence

`output{ write_avs_v }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs AVS steering file `.v`.

7.3.11 write_origin_plt

Calling sequence

`output{ write_origin_plt }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs Origin steering file `.plt`.

7.3.12 write_gnuplot_plt

Calling sequence

`output{ write_gnuplot_plt }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs gnuplot file `.plt`.

> **Attention**
>
> Currently, gnuplot format is only implemented for energy resolved densities in 1D, energy resolved photo generation in 1D, and light field and may generate huge files.

7.3.13 use_gnuplot_one_file

Calling sequence

`output{ use_gnuplot_one_file }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `yes` then all information (metadata and data) necessary for the gnuplot figure is contained in one file.

7.3.14 only_sections

Calling sequence

`output{ only_sections }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If `only_sections = yes` then outputs only sections of 2D and 3D fields defined by *output{ }* will be generated. Thus, if no sections are defined then also no fields will be outputted. These files can be used to restrict field output to the actual regions of interest, or also to suppress most file I/O (if no sections are defined).

Note

Quantities living on, e.g., an energy grid, integrative quantities like I-V curves, or files needed for resuming operation are not influenced by this setting.

Attention

This setting has no effect on RAM usage or on the fields used in the calculation, it just affects what is written into output files.

7.3.15 section{ }

Calling sequence

`output{ section{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Generates outputs from selected range of the simulation domain. The range is defined by *section{ range_x }*, *section{ range_y }*, and *section{ range_z }*.

Attention

All section commands are ignored for energy resolved densities, energy resolved photo generation, and light field.

Examples

```
output{
    section{
        name = "part"       # name of section enters file name
        range_x = [0, 20]   # range in x direction [nm]
        range_y = [-5, 5]   # range in y direction [nm] (2D or 3D only)
        range_z = [2, 10]   # range in z direction [nm] (3D only)
    }
}
```

```
output{
    directory = "../output/mosfet_2D"
    section{
        name    = "zoom"
        range_x = [0,20]          # range in x direction from 0 nm to 20 nm
        range_y = [-5,5]          # range in y direction from -5 nm to 5 nm
    }
}
```

7.3.16 section{ name }

Calling sequence

`output{ section{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Defines a suffix to a name of the generated output file.

7.3.17 section{ range_x }

Calling sequence

`output{ section{ range_x } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **unit:** nm

Functionality

Defines a range interval along the x-direction of the simulation domain for the additional output. The first number defines the beginning of the interval and the second defines its end.

> **Note**
>
> Ranges in sections must contain at least one grid point. If no point is found inside the range then the closest grid point is used. Zero-length intervals, such as `[50.1, 50.1]`, are allowed.

7.3.18 section{ range_y }

Calling sequence

`output{ section{ range_y } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **unit:** nm

Functionality

Defines a range interval along the y-direction of the simulation domain for the additional output. The first number defines the beginning of the interval and the second defines its end.

> **Note**
>
> Ranges in sections must contain at least one grid point. If no point is found inside the range then the closest grid point is used. Zero-length intervals, such as `[50.1, 50.1]`, are allowed.

7.3.19 section{ range_z }

Calling sequence

`output{ section{ range_z } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **unit:** nm

Functionality

Defines a range interval along the z-direction of the simulation domain for the additional output. The first number defines the beginning of the interval and the second defines its end.

> **Note**
>
> Ranges in sections must contain at least one grid point. If no point is found inside the range then the closest grid point is used. Zero-length intervals, such as `[50.1, 50.1]`, are allowed.

7.3.20 section1D{ }

Calling sequence

`output{ section1D{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Outputs a 1D section of the simulation area, a 1D slice, from 2D or 3D simulation.

> **Note**
>
> - **2D usage:**
>   - **`x`, `range_y`**
>
>   1D slice at `x` = … nm within the range from `y` = … nm to `y` = … nm or
>   - **`y`, `range_x`**
>
>   1D slice at `y` = … nm within the range from `x` = … nm to `x` = … nm
> - **3D usage:**
>   - **`x`, `y`, `range_z` or**
>
>   1D slice at `x` = … nm and `y` = … nm within the range from `z` = … nm to `z` = … nm
>
>   …
>
> If range is left out, the section extends over the whole simulation area.

Examples

```
output{
    section1D{
        name = "x"          # name of section enters file name

        x = 10.0            # 1D slice at x = 10 nm
        y = 10.0            # 1D slice at y = 10 nm
        z = 10.0            # 1D slice at z = 10 nm (3D only)

        range_x = [0, 20]   # (optional) range in x direction [nm]
        range_y = [-5, 5]   # (optional) range in y direction [nm]
        range_z = [2, 10]   # (optional) range in z direction [nm] (3D only)
    }
}
```

```
output{
    directory = "../output/mosfet_3D"

    section1D{
        name = "x"
        y    = 10
        z    = 10
    }
}
```

```
output{
    directory = "../output/mosfet_2D"
    section1D{
        name    = "y"
        y       = 10                  # 1D slice at y = 10 nm
        range_x = [-20, 220.5]    # range in x direction from -20 nm to 220.5 nm
    }
}
```

7.3.21 section1D{ name }

Calling sequence

`output{ section1D{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Defines a suffix to a name of the generated output file.

7.3.22 section1D{ x }

Calling sequence

`output{ section1D{ x } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

Defines position along the x-direction of the simulation domain at which the section of generated data is created and added to the output.

7.3.23 section1D{ y }

Calling sequence

`output{ section1D{ y } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

Defines position along the y-direction of the simulation domain at which the section of generated data is created and added to the output.

7.3.24 section1D{ z }

Calling sequence

`output{ section1D{ z } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

Defines position along the z-direction of the simulation domain at which the section of generated data is created and added to the output.

7.3.25 section1D{ range_x }

Calling sequence

`output{ section1D{ range_x } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **unit:** nm

Functionality

Defines a range interval along the x-direction of the simulation domain for the additional output. The first number defines the beginning of the interval and the second defines its end.

Note

Ranges in sections must contain at least one grid point. If no point is found inside the range then the closest grid point is used. Zero-length intervals, such as `[50.1, 50.1]`, are allowed.

7.3.26 section1D{ range_y }

Calling sequence

`output{ section1D{ range_y } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **unit:** nm

Functionality

Defines a range interval along the y-direction of the simulation domain for the additional output. The first number defines the beginning of the interval and the second defines its end.

Note

Ranges in sections must contain at least one grid point. If no point is found inside the range then the closest grid point is used. Zero-length intervals, such as `[50.1, 50.1]`, are allowed.

7.3.27 section1D{ range_z }

Calling sequence

`output{ section1D{ range_z } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **unit:** nm

Functionality

Defines a range interval along the z-direction of the simulation domain for the additional output. The first number defines the beginning of the interval and the second defines its end.

Note

Ranges in sections must contain at least one grid point. If no point is found inside the range then the closest grid point is used. Zero-length intervals, such as `[50.1, 50.1]`, are allowed.

7.3.28 section2D{ }

Calling sequence

`output{ section2D{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Outputs a 2D section of the simulation area, a 2D slice, from 3D simulation.

Note

- **3D usage:**
  - **`x`, `range_y`, `range_z`**

    2D slice at `x` = … nm within the range from `y` = … nm to `y` = … nm and from `z` = … nm to `z` = … nm or
  - **`y`, `range_x`, `range_z`**

    2D slice at `y` = … nm within the range from `x` = … nm to `x` = … nm and from `z` = … nm to `z` = … nm or
  - **`z`, `range_x`, `range_y`**

    2D slice at `z` = … nm within the range from `x` = … nm to `x` = … nm and from `y` = … nm to `y` = … nm

Examples

```
output{
    section2D{
        name = "center"         # name of section enters file name

        x = 10.0                # 2D slice at x = 10 nm
        y = 20.0                # 2D slice at y = 20 nm
        z = 10.0                # 2D slice at z = 10 nm

        range_x = [0, 20]       # (optional) range in x direction [nm]
        range_y = [-5, 5]       # (optional) range in y direction [nm]
        range_z = [2, 10]       # (optional) range in z direction [nm]
    }
}
```

```
output{
    directory = "../output/mosfet_3D"

    section2D{
        name    = "y"
        y       = 10                # 2D slice at y = 10 nm
        range_x = [-20, 220.5]     # range in x direction from -20 nm to 220.5 nm
        range_z = [-20, 220.5]     # range in z direction from -20 nm to 220.5 nm
    }
}
```

7.3.29 material_parameters{ }

Calling sequence

`output{ material_parameters{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Defines additional outputs.

7.3.30 material_parameters{ kp_parameters{ } }

Calling sequence

`output{ material_parameters{ kp_parameters{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs

- $\mathbf{k}\cdot\mathbf{p}$ parameters of materials in quantum regions where 6-band or 8-band $\mathbf{k}\cdot\mathbf{p}$ Hamiltonian was solved,
- the Dresselhaus-Kip-Kittel (DKK) parameters (`L`, `M`, `N`), which are used internally in the code,
- the Luttinger parameters (`gamma1`, `gamma2`, `gamma3`, `kappa`) (for zinc blende) or Rashba-Sheka-Pikus (`A1`, `A2`, . . . , `A6`) parameters (for wurtzite),
- the `S`, `E_P`, `P` and `B` parameters for 8-band $\mathbf{k}\cdot\mathbf{p}$ calculations.

For further information, consult Chapter 3 of *[BirnerPhD2011]*.

7.3.31 material_parameters{ kp_parameters{ boxes } }

Calling sequence

`output{ material_parameters{ kp_parameters{ boxes } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

7.3.32 material_parameters{ spin_orbit_coupling_energies{ } }

Calling sequence

`output{ material_parameters{ spin_orbit_coupling_energies{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs spin-orbit coupling energy for zinc blende (1 parameter) or crystal-field splitting and spin-orbit coupling energies for wurtzite (3 parameters) in `(eV)`.

7.3.33 material_parameters{ spin_orbit_coupling_energies{ boxes } }

Calling sequence

`output{ material_parameters{ spin_orbit_coupling_energies{ boxes } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

7.3.34 material_parameters{ charge_carrier_masses{ } }

Calling sequence

`output{ material_parameters{ charge_carrier_masses{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs effective masses of all energy bands used in the simulations in `(m0)`.

7.3.35 material_parameters{ charge_carrier_masses{ boxes } }

Calling sequence

`output{ material_parameters{ charge_carrier_masses{ boxes } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

7.3.36 material_parameters{ static_dielectric_constants{ } }

Calling sequence

`output{ material_parameters{ static_dielectric_constants{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs static relative dielectric constants for zinc blende (1 parameter) and wurtzite (3 parameters).

7.3.37 material_parameters{ static_dielectric_constants{ boxes } }

Calling sequence

`output{ material_parameters{ static_dielectric_constants{ boxes } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

7.3.38 material_parameters{ deformation_potentials{ } }

Calling sequence

`output{ material_parameters{ deformation_potentials{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output the deformation potentials for zinc blende and wurtzite in `(eV)`.

7.3.39 material_parameters{ deformation_potentials{ boxes } }

Calling sequence

`output{ material_parameters{ deformation_potentials{ boxes } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

7.4 run{ }

Calling sequence

`run{ }`

Properties

- **usage:** required
- **items:** exactly 1

Dependencies

- Up to one of *poisson{ }* and *current_poisson{ }* can be defined.
- Up to one of *quantum{ }*, *quantum_density{ }*, *quantum_poisson{ }*, and *quantum_current_poisson{ }* can be defined.
- Exactly one of *quantum{ }*, *quantum_density{ }*, *quantum_poisson{ }*, or *quantum_current_poisson{ }* must be defined if *quantum_optics{ }* is defined.
- None of *strain{ }*, *poisson{ }*, *current_poisson{ }*, *quantum{ }*, *quantum_density{ }*, *quantum_poisson{ }*, *quantum_current_poisson{ }*, and *quantum_optics{ }* are allowed to be defined if *structure_only{ }* is defined.

Functionality

This group defines the simulation flow, i.e., equations to be solved and degree of self-consistency.

Examples

```
run{}
```

```
run{
    structure_only{}
}
```

```
run{
    strain{}
    poisson{}
}
```

```
run{
    strain{}
    current_poisson{}
}
```

```
run{
    strain{}
    quantum{}
}
```

```
run{
    strain{}
    quantum_poisson{}
    quantum_optics{}
}
```

```
run{
    strain{}
    quantum_current_poisson{}
    quantum_optics{}
}
```

```
run{
    strain{}
    current_poisson{}
    quantum_current_poisson{}
    quantum_optics{}
}
```

Nested keywords

7.4.1 structure_only{ }

Calling sequence

`run{ structure_only{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

If this group is defined, then calculation is aborted after structure setup, similarly to when the command line flag `-s` or `--structure` is set. But differently from the command line flag, if `last_region` is present, partial structure initialization is performed. This is useful for debugging your structure definition, e.g. if you have a 2D or 3D simulation with many material regions, contact regions, doping regions and generation regions overlapping each other in a complicated way. The files in the output directory *Structure/* will then reflect this partial initialization. Note that in case not all regions are used here, some initialization and output steps related to strain, poisson, current, quantum, cbr, optics, etc. will be omitted in order to avoid inconsistencies.

Example

```
run{
    structure_only{ }
}
```

Nested keywords

- *last_region*

last_region

Calling sequence

`run{ structure_only{ last_region } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** `all regions`
- **unit:** $-$

Functionality

Defines the highest number of region printed in to the output file.

Example

```
run{
    structure_only{
        last_region = 5
    }
}
```

7.4.2 strain{ }

Calling sequence

`run{ strain{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The *strain{ }* must be defined.

Functionality

When this group is defined, the strain equation is solved at the beginning of the algorithm and the strain effects are included in further parts of the simulation.

Example

```
run{
    strain{}
}

strain{}
```

7.4.3 poisson{ }

Calling sequence

`run{ poisson{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The *poisson{ }* must be defined.

Functionality

When this group is defined, the Poisson equation is solved using semiclassical (bulk-material) densities of sates and without any self-consistency with the other equations. The major result here is the electrostatic potential.

Example

```
run{
    poisson{}
}

poisson{}
```

7.4.4 current_poisson{ }

Calling sequence

`run{ current_poisson{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- Global groups *poisson{ }* and *currents{ }* must be defined.

Functionality

When this group is defined, the system of coupled current and Poisson equations is solved self-consistently using semiclassical (bulk-material) densities of sates. The major results here are the electrostatic potential and quasi-Fermi levels.

Example

```
run{
    current_poisson{}
}

poisson{}
currents{}
```

Nested keywords

- *fermi_limit*
- *multi_stage_solve*
- *fast_poisson*
- *system_solve*
- *iterations*
- *current_repetitions*
- *limit_repetitions*
- *residual*
- *residual_fermi*
- *alpha_fermi*
- *alpha_iterations*
- *alpha_scale*
- *minimum_density_electrons*
- *minimum_density_holes*
- *maximum_density_electrons*
- *maximum_density_holes*
- *smooth_currents*
- *output_log*
- *output_local_residuals*

---

fermi_limit

Calling sequence

`run{ current_poisson{ fermi_limit } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.0 \leq r \leq 10.0$
- **default:** $r = 2.0$
- **unit:** eV

Example

```
run{
    current_poisson{
        fermi_limit = 0.5
    }
}
```

(continues on next page)

(continued from previous page)

```

poisson{}
currents{}
```

Functionality

This keyword defines the energy range within which the quasi-Fermi levels are allowed in the simulation, and during the runtime of related algorithms. The maximum is defined as the highest Fermi level at contacts plus the `fermi_limit` while the minimum is defined as the lowest Fermi level at contacts minus the `fermi_limit`.

> **Note**
>
> Except in case of huge band gaps and extreme photogeneration, the default value should not require any change.
>
> At the same time, in the absence of any externally induced photogeneration, this value can be set to zero in order to stabilize the solver.

---

multi_stage_solve

Calling sequence

`run{ current_poisson{ multi_stage_solve } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

When `multi_stage_solve = yes`, then the current equation is solved in two stages. First, without recombination and generation processes. Second, with the recombination and generation processes included using the solutions from the first run as initial conditions, if any recombination or generation models are turned on.

> **Hint**
>
> This keyword can be used to improve convergence in some cases, but may also increase the simulation runtime.

Example

```
run{
    current_poisson{
        multi_stage_solve = yes
    }
}

poisson{}
currents{}
```

---

fast_poisson

Calling sequence

`run{ current_poisson{ fast_poisson } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `fast_poisson = yes`, then Newton iterations of the Poisson solver in the within the classical current-Poisson iteration will be limited to 1. Note that enabling this setting may also influence stability of convergence or change the optimal value for `alpha_fermi`. Typically, `fast_poisson = yes` increases the number of iterations but significantly reduces the overall execution time.

Example

```
run{
    current_poisson{
        fast_poisson = yes
    }
}

poisson{}
currents{}
```

---

system_solve

Calling sequence

`run{ current_poisson{ system_solve } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- Defining this attribute requires presence of *current_repetitions*.

Functionality

Alternative new iteration method for classical current-Poisson. This Newton method may provide better convergence for some systems (but may require different values of convergence parameters). Setting `system_solve = yes` results in Fermi levels and potential being simultaneously updated as a system of unknowns during the iteration. Irrespective of its value, `system_solve` always takes the value of `current_repetitions` into account.

Example

```
run{
    current_poisson{
        system_solve = yes
    }
}

poisson{}
currents{}
```

---

iterations

Calling sequence

`run{ current_poisson{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 100$
- **unit:** $-$

Functionality

Maximum number of iterations for current-Poisson solver

Example

```
run{
    current_poisson{
        iterations = 200
    }
}

poisson{}
currents{}
```

---

current_repetitions

Calling sequence

`run{ current_poisson{ current_repetitions } }`

Properties

- **usage:** conditional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 1$
- **unit:** $-$

Dependencies

- This attribute is required if *system_solve* or *limit_repetitions* is defined.

Functionality

Number of current-density iterations. The current equations are repeatedly solved for the quasi-Fermi levels with the densities fixed. The current equation for the electrons and for the holes are solved independently with a common and fixed recombination term. For each iteration, the densities are adjusted according to the new quasi-Fermi levels of the previous iteration. `current_repetitions` defines number of these repetitions. If generation/recombination is present, using a value > 1 (e.g. 5) may stabilize the iteration and sometimes enable faster convergence (larger `alpha_fermi` may also be possible then).

Example

```
run{
    current_poisson{
        current_repetitions = 5
    }
}

poisson{}
currents{}
```

---

limit_repetitions

Calling sequence

`run{ current_poisson{ limit_repetitions } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- Defining this attribute requires presence of *current_repetitions*.

Functionality

If enabled, the current-density loop is exited early as soon as `residual_fermi` is reached by the quasi-Fermi levels.

Example

```
run{
    current_poisson{
        current_repetitions = yes
    }
}

poisson{}
currents{}
```

residual

Calling sequence

`run{ current_poisson{ residual } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 10^5$ for 1D; $r = 10^3$ for 2D; $r = 10^{-3}$ for 3D
- **unit:** cm^{-2} for 1D; cm^{-1} for 2D; none for 3D

Functionality

Residual occupation changes.

Example

```
run{
    current_poisson{
        residual = 1e4
    }
}

poisson{}
currents{}
```

---

residual_fermi

Calling sequence

`run{ current_poisson{ residual_fermi } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 1e-5$
- **unit:** eV

Functionality

Residual Fermi level changes, see *Residuals* for more details. This value is also used during `quantum_current_poisson{ }`

Example

```
run{
    current_poisson{
        residual_fermi = 1e-6
    }
```

(continues on next page)

(continued from previous page)

```
}

poisson{}
currents{}
```

---

alpha_fermi

Calling sequence

`run{ current_poisson{ alpha_fermi } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-5} \leq r \leq 1.0$
- **default:** $r = 1.0$
- **unit:** $-$

Functionality

Dimensionless under-relaxation parameter for Fermi level. The final quasi-Fermi level for electrons after each iteration is calculated as follows:

$E_{F,n}$ = ($E_{F,n}$ of previous iteration) * (1 - `alpha_fermi`) + ($E_{F,n}$ of actual iteration) * `alpha_fermi`

This Fermi level is then input to the next iteration. The same holds for the Fermi level $E_{F,p}$ for holes. The value of `alpha_fermi` will change due to `alpha_scale` during the iterations. The actually used `alpha_fermi` is now included in *iteration_current_poisson.dat* and *iteration_quantum_current_poisson_details.dat*.

Example

```
run{
    current_poisson{
        alpha_fermi  =  0.5
    }
}

poisson{}
currents{}
```

---

alpha_iterations

Calling sequence

`run{ current_poisson{ alpha_iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 1000$
- **unit:** $-$

Functionality

Iteration at which `alpha_fermi` begins to be rescaled by `alpha_scale` at each following iteration.

Example

```
run{
    current_poisson{
        alpha_iterations = 200
    }
}

poisson{}
currents{}
```

---

alpha_scale

Calling sequence

`run{ current_poisson{ alpha_scale } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.1 \leq r \leq 1.0$
- **default:** $r = 0.998$
- **unit:** $-$

Functionality

A factor rescaling `alpha_fermi` starting at the iteration `alpha_iterations`, both for classical and quantum stages of simulation. The `alpha_fermi` is overwritten by: `max( alpha_fermi * alpha_scale , 1e-5)` at each iteration step once the number of iterations exceeds `alpha_iterations`.

Use this feature to improve convergence (particularly convergence of Fermi levels) towards the end of the iteration.

Warning

Decreasing `alpha_fermi` too fast (a problem with older versions) will result in the iteration stalling, (only the residuals of the densities but none of the Fermi levels decrease). The total current equation may then not be properly conserved.

Example

```
run{
    current_poisson{
        alpha_scale = 0.995
    }
}

poisson{}
currents{}
```

minimum_density_electrons

Calling sequence

`run{ current_poisson{ minimum_density_electrons } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{20}$
- **default:** $r = 1.0$
- **unit:** cm^{-3}

Functionality

—

---

minimum_density_holes

Calling sequence

`run{ current_poisson{ minimum_density_holes } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{20}$
- **default:** $r = 1.0$
- **unit:** cm^{-3}

Functionality

—

---

maximum_density_electrons

Calling sequence

`run{ current_poisson{ maximum_density_electrons } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{30}$
- **default:** $r = 1e30$
- **unit:** cm^{-3}

Functionality

—

---

maximum_density_holes

Calling sequence

`run{ current_poisson{ maximum_density_holes } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{30}$
- **default:** $r = 1e30$
- **unit:** cm^{-3}

Functionality

—

---

smooth_currents

Calling sequence

`run{ current_poisson{ smooth_currents } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then current equation is additionally solved at the very end of the algorithm.

---

output_log

Calling sequence

`run{ current_poisson{ output_log } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

—

Example

```
run{
    current_poisson{
        output_log = no
    }
}

poisson{}
currents{}
```

---

output_local_residuals

Calling sequence

`run{ current_poisson{ output_local_residuals } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs residuals as functions of position when `output_local_residuals = yes`. In case the attribute is enabled for both classical and quantum iterations, the quantum iteration overwrites the respective files of the classical iteration.

> **Attention**
>
> Both conditions specified by `residual` and `residual_fermi` must hold in order to consider a calculation as converged.

Example

```
run{
    current_poisson{
        output_local_residuals = yes
    }
}

poisson{}
currents{}
```

Last update: 02/04/2025

7.4.5 quantum{ }

Calling sequence

`run{ quantum{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The *quantum{ }* must be defined.

Functionality

Solves the Schrödinger equation. Exchange–correlation effects (optional) can be included and are calculated from the quantum density. Then the Schrödinger equation is solved again but this time including the exchange-correlation potential energy.

Example

```
run{
    quantum{}
}

quantum{}
```

7.4.6 quantum_density{ }

Calling sequence

`run{ quantum_density{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The *quantum{ }* must be defined.
- The *exchange_correlation{ }* must be defined.

Functionality

Includes exchange correlation effects into solutions of Schrödinger equation in a self-consistent manner.

Example

```
run{
    quantum_density{}
}

quantum{
    exchange_correlation{}
}
```

Nested keywords

- *residual*
- *iterations*
- *use_subspace*
- *subspace_iterations*
- *subspace_residual_factor*
- *output_log*
- *output_local_residuals*

---

residual

Calling sequence

`run{ quantum_density{ residual } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 10^5$ for 1D; $r = 10^3$ for 2D; $r = 10^{-3}$ for 3D
- **unit:** cm^{-2} for 1D; cm^{-1} for 2D; none for 3D

Functionality

Defines requested residual of the integrated total charge carrier density changes. Note that this is **dimension dependent** and default is: 1e5/cm^2 (1D), 1e3/cm (2D), 1e-3[dimensionless] (3D). This applies to exact Schrödinger equation, not to subspace Schrödinger equation

Note

If you do not include enough eigenstates, the convergence behavior might be affected as the occupation of the eigenstates is not considered in a useful way.

Example

```
run{
    quantum_density{
        residual = 1e4
    }
}

quantum{
    exchange_correlation{}
}
```

---

iterations

Calling sequence

`run{ quantum_density{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 0$
- **default:** $z = 30$
- **unit:** $-$

Functionality

Maximum number of iterations, i.e. self-consistency cycles

Example

```
run{
    quantum_density{
        iterations = 50
    }
}

quantum{
    exchange_correlation{}
}
```

---

use_subspace

Calling sequence

`run{ quantum_density{ use_subspace } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Solve Schrödinger equation within subspace of eigenvectors of previous iteration as long as achieved residual is larger than desired `residual * residual_factor` and at least in every second iteration

Example

```
run{
    quantum_density{
        use_subspace = no
    }
}
```

(continues on next page)

(continued from previous page)

```
quantum{
    exchange_correlation{}
}
```

---

subspace_iterations

Calling sequence

`run{ quantum_density{ subspace_iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 1000$
- **default:** $z = 1$
- **unit:** $-$

Functionality

Number of subspace iterations

Example

```
run{
    quantum_density{
        subspace_iterations = 5
    }
}

quantum{
    exchange_correlation{}
}
```

---

subspace_residual_factor

Calling sequence

`run{ quantum_density{ subspace_residual_factor } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[2.0, ...)`
- **default:** $r = 1e12$
- **unit:** $-$

Functionality

Residual factor for subspace iterations

Example

```
run{
    quantum_density{
        subspace_residual_factor = 1e10
    }
}

quantum{
    exchange_correlation{}
}
```

---

output_log

Calling sequence

`run{ quantum_density{ output_log } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Output of convergence of Schrödinger-Poisson equation (residuals for `quantum_density`) into the logfile *iteration_quantum_density.dat*

Example

```
run{
    quantum_density{
        output_log = no
    }
}

quantum{
    exchange_correlation{}
}
```

---

output_local_residuals

Calling sequence

`run{ quantum_density{ output_local_residuals } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs residuals as functions of position when `output_local_residuals = yes`. In case the attribute is enabled for both a classical and quantum iterations, the quantum iteration overwrites the respective files of the classical iteration.

Example

```
run{
    quantum_density{
        output_local_residuals = yes
    }
}

quantum{
    exchange_correlation{}
}
```

Last update: 03/04/2025

7.4.7 quantum_poisson{ }

Calling sequence

`run{ quantum_poisson{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The *quantum{ }* and *poisson{ }* must be defined.

Functionality

Triggers solving of the Schrödinger and Poisson equations self-consistently for the defined system.

Example

```
run{
    quantum_poisson{}
}

poisson{}
quantum{}
```

Nested keywords

- *residual*
- *iterations*
- *use_subspace*
- *subspace_iterations*
- *subspace_residual_factor*
- *alpha_potential*
- *output_log*
- *output_local_residuals*

---

residual

Calling sequence

`run{ quantum_poisson{ residual } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 10^5$ for 1D; $r = 10^3$ for 2D; $r = 10^{-3}$ for 3D
- **unit:** cm^{-2} for 1D; cm^{-1} for 2D; none for 3D

Functionality

Defines requested residual of the integrated total charge carrier density changes. Note that this is **dimension dependent** and default is: 1e5/cm^2 (1D), 1e3/cm (2D), 1e-3[dimensionless] (3D). This applies to exact Schrödinger equation, not to subspace Schrödinger equation

Note

If you do not include enough eigenstates, the convergence behavior might be affected as the occupation of the eigenstates is not considered in a useful way.

Example

```
run{
    quantum_poisson{
        residual = 1e4
    }
}

poisson{}
quantum{}
```

---

iterations

Calling sequence

`run{ quantum_poisson{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 0$
- **default:** $z = 30$
- **unit:** $-$

Functionality

Maximum number of iterations, i.e. self-consistency cycles

Example

```
run{
    quantum_poisson{
        iterations = 50
    }
}

poisson{}
quantum{}
```

---

use_subspace

Calling sequence

`run{ quantum_poisson{ use_subspace } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Solve Schrödinger equation within subspace of eigenvectors of previous iteration as long as achieved residual is larger than desired `residual * residual_factor` and at least in every second iteration

Example

```
run{
    quantum_poisson{
        use_subspace = no
    }
}
```

(continues on next page)

(continued from previous page)

```
poisson{}
quantum{}
```

---

subspace_iterations

Calling sequence

`run{ quantum_poisson{ subspace_iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 1000$
- **default:** $z = 1$
- **unit:** $-$

Functionality

Number of subspace iterations

Example

```
run{
    quantum_poisson{
        subspace_iterations = 5
    }
}

poisson{}
quantum{}
```

---

subspace_residual_factor

Calling sequence

`run{ quantum_poisson{ subspace_residual_factor } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[2.0, ...)`
- **default:** $r = 1e12$
- **unit:** $-$

Functionality

Residual factor for subspace iterations

Example

```
run{
    quantum_poisson{
        subspace_residual_factor = 1e10
    }
}

poisson{}
quantum{}
```

---

alpha_potential

Calling sequence

`run{ quantum_poisson{ alpha_potential } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-3} \leq r \leq 1.0$
- **default:** $r = 1.0$
- **unit:** $-$

Functionality

In case of stubborn convergence problems which do not appear to have any root cause such as not enough eigenvalues and which appear not to respond to any change in other parameters, try using a mildly smaller value than 1.0 such as 0.5.

Using values smaller than 1.0 per default is not recommended, as the run time is expected to increase as 1/`alpha_potential` for normally converging input files.

Example

```
run{
    quantum_poisson{
        alpha_potential = 0.5
    }
}

poisson{}
quantum{}
```

---

output_log

Calling sequence

`run{ quantum_poisson{ output_log } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Output of convergence of Schrödinger-Poisson equation (residuals for `quantum_poisson`) into the logfile *iteration_quantum_poisson.dat*

Example

```
run{
    quantum_poisson{
        output_log = no
    }
}

poisson{}
quantum{}
```

---

output_local_residuals

Calling sequence

`run{ quantum_poisson{ output_local_residuals } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs residuals as functions of position when `output_local_residuals = yes`. In case the attribute is enabled for both a classical and quantum iterations, the quantum iteration overwrites the respective files of the classical iteration.

Example

```
run{
    quantum_poisson{
        output_local_residuals = yes
    }
}

poisson{}
quantum{}
```

7.4.8 quantum_current_poisson{ }

Calling sequence

`run{ quantum_current_poisson{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The *quantum{ }* must be defined.
- The *currents{ }* must be defined.
- The *poisson{ }* must be defined.

Functionality

It solves the **Schrödinger-Current-Poisson** equations self-consistently. When `quantum_current_poisson{ }` is desired, note that additionally either `poisson{ }` or `current_poisson{ }` is required and `current_poisson` must be defined in the input file..

Example

```
run{
    quantum_current_poisson{}
}

poisson{}
currents{}
quantum{}
```

Nested keywords

- *residual*
- *iterations*
- *use_subspace*
- *subspace_iterations*
- *subspace_residual_factor*
- *fermi_limit*
- *current_repetitions*
- *limit_repetitions*
- *residual_fermi*
- *alpha_fermi*
- *alpha_iterations*
- *alpha_scale*
- *alpha_potential*
- *minimum_density_electrons*

- *minimum_density_holes*
- *maximum_density_electrons*
- *maximum_density_holes*
- *smooth_currents*
- *output_log*
- *output_local_residuals*

---

residual

Calling sequence

`run{ quantum_current_poisson{ residual } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 10^5$ for 1D; $r = 10^3$ for 2D; $r = 10^{-3}$ for 3D
- **unit:** cm^{-2} for 1D; cm^{-1} for 2D; none for 3D

Functionality

Defines requested residual of the integrated total charge carrier density changes. Note that this is **dimension dependent** and default is: 1e5/cm^2 (1D), 1e3/cm (2D), 1e-3[dimensionless] (3D). This applies to exact Schrödinger equation, not to subspace Schrödinger equation

Note

If you do not include enough eigenstates, the convergence behavior might be affected as the occupation of the eigenstates is not considered in a useful way.

Example

```
run{
    quantum_current_poisson{
        residual = 1e4
    }
}

poisson{}
currents{}
quantum{}
```

---

iterations

Calling sequence

`run{ quantum_current_poisson{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 0$
- **default:** $z = 30$
- **unit:** –

Functionality

Maximum number of iterations, i.e. self-consistency cycles

Example

```
run{
    quantum_current_poisson{
        iterations = 50
    }
}

poisson{}
currents{}
quantum{}
```

---

use_subspace

Calling sequence

`run{ quantum_current_poisson{ use_subspace } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Solve Schrödinger equation within subspace of eigenvectors of previous iteration as long as achieved residual is larger than desired `residual * residual_factor` and at least in every second iteration

Example

```
run{
    quantum_current_poisson{
        use_subspace = no
    }
}

poisson{}
currents{}
quantum{}
```

subspace_iterations

Calling sequence

`run{ quantum_current_poisson{ subspace_iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 1000$
- **default:** $z = 1$
- **unit:** –

Functionality

Number of subspace iterations

Example

```
run{
    quantum_current_poisson{
        subspace_iterations = 3
    }
}

poisson{}
currents{}
quantum{}
```

---

subspace_residual_factor

Calling sequence

`run{ quantum_current_poisson{ subspace_residual_factor } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[2.0, ...)`
- **default:** $r = 1e12$
- **unit:** –

Functionality

Residual factor for subspace iterations

Example

```
run{
    quantum_current_poisson{
        subspace_residual_factor = 1e11
    }
```

(continues on next page)

(continued from previous page)

```
}

poisson{}
currents{}
quantum{}
```

---

fermi_limit

Calling sequence

`run{ quantum_current_poisson{ fermi_limit } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.0 \leq r \leq 10.0$
- **default:** $r = 2.0$
- **unit:** eV

Functionality

—

Example

```
run{
    quantum_current_poisson{
        fermi_limit = 0.7
    }
}

poisson{}
currents{}
quantum{}
```

---

current_repetitions

Calling sequence

`run{ quantum_current_poisson{ current_repetitions } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 2$
- **unit:** $-$

Functionality

number of current-density iterations. The current equation is repeatedly solved for the quasi-Fermi levels. For each iteration, the densities are adjusted according to the new quasi-Fermi levels of the previous iteration. `current_repetitions` defines number of these repetitions. If generation/recombination is present, using a value > 1 (e.g. 5) may stabilize the iteration and sometimes enable faster convergence (larger `alpha_fermi` may also be possible then).

Example

```
run{
    quantum_current_poisson{
        current_repetitions = 4
    }
}

poisson{}
currents{}
quantum{}
```

---

limit_repetitions

Calling sequence

`run{ quantum_current_poisson{ limit_repetitions } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If enabled, the current-density loop is exited early as soon as `residual_fermi` is reached by the quasi-Fermi levels.

Example

```
run{
    quantum_current_poisson{
        limit_repetitions = yes
    }
}

poisson{}
currents{}
quantum{}
```

---

residual_fermi

Calling sequence

`run{ quantum_current_poisson{ residual_fermi } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 1e - 5$
- **unit:** eV

Functionality

—

Example

```
run{
    quantum_current_poisson{
        residual_fermi = 1e-6
    }
}

poisson{}
currents{}
quantum{}
```

---

alpha_fermi

Calling sequence

`run{ quantum_current_poisson{ alpha_fermi } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-5} \leq r \leq 1.0$
- **default:** $r = 1.0$
- **unit:** $-$

Functionality

The Fermi level is under-relaxed between repetitions using an under-relaxation parameter for the Fermi levels. It should be used once an oscillation of residuals is observed while self-consistently solving the Poisson and Schrödinger (and current) equations to improve convergence. For further information, please read comments on `alpha_fermi` parameter above

Example

```
run{
    quantum_current_poisson{
        alpha_fermi = 0.2
    }
}
```

(continues on next page)

(continued from previous page)

```
poisson{}
currents{}
quantum{}
```

---

alpha_iterations

Calling sequence

`run{ quantum_current_poisson{ alpha_iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 1000$
- **unit:** $-$

Functionality

number of alpha iterations

Example

```
run{
    quantum_current_poisson{
        alpha_iterations = 100
    }
}

poisson{}
currents{}
quantum{}
```

---

alpha_scale

Calling sequence

`run{ quantum_current_poisson{ alpha_scale } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.1 \leq r \leq 1.0$
- **default:** $r = 0.998$
- **unit:** $-$

Functionality

Both for classical and for quantum iterations, `alpha_fermi` will be reduced further as `alpha_fermi <- max( alpha_fermi * alpha_scale , 1e-5)` at each iteration step once the number of iterations exceeds alpha_iterations. Use this feature to improve convergence (particularly convergence of Fermi levels) towards the end of the iteration. Note that decreasing `alpha_fermi` too fast (a problem with older versions) will result in the iteration stalling (only the residuals of the densities but none of the Fermi levels decrease). The total current equation may then not be properly conserved.

Example

```
run{
    quantum_current_poisson{
        alpha_scale = 0.995
    }
}

poisson{}
currents{}
quantum{}
```

---

alpha_potential

Calling sequence

`run{ quantum_current_poisson{ alpha_potential } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-3} \leq r \leq 1.0$
- **default:** $r = 1.0$
- **unit:** $-$

Functionality

In case of stubborn convergence problems which do not appear to have any root cause such as not enough eigenvalues and which appear not to respond to any change in other parameters, try using a mildly smaller value than 1.0 such as 0.5.

Using values smaller than 1.0 per default is not recommended, as the run time is expected to increase as 1/ `alpha_potential` for normally converging input files.

Example

```
run{
    quantum_current_poisson{
        alpha_potential = 0.5
    }
}

currents{}
quantum{}
```

---

minimum_density_electrons

Calling sequence

`run{ quantum_current_poisson{ minimum_density_electrons } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{20}$
- **default:** $r = 1.0$
- **unit:** cm^{-3}

Functionality

—

minimum_density_holes

Calling sequence

`run{ quantum_current_poisson{ minimum_density_holes } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{20}$
- **default:** $r = 1.0$
- **unit:** cm^{-3}

Functionality

—

maximum_density_electrons

Calling sequence

`run{ quantum_current_poisson{ maximum_density_electrons } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{30}$
- **default:** $r = 1e30$
- **unit:** cm^{-3}

Functionality

—

maximum_density_holes

Calling sequence

`run{ quantum_current_poisson{ maximum_density_holes } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{30}$
- **default:** $r = 1e30$
- **unit:** cm^{-3}

Functionality

—

smooth_currents

Calling sequence

`run{ quantum_current_poisson{ smooth_currents } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then current equation is additionally solved at the very end of the algorithm.

output_log

Calling sequence

`run{ quantum_current_poisson{ output_log } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Output of convergence of (quantum) current-Poisson equation (residuals for `quantum_current_poisson`) into the logfile *iteration_quantum_current_poisson.dat*

Example

```
run{
    quantum_current_poisson{
        output_log = no
    }
}

currents{}
quantum{}
```

---

output_local_residuals

Calling sequence

`run{ quantum_current_poisson{ output_local_residuals } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs residuals as functions of position when `output_local_residuals = yes`. In case the attribute is enabled for both classical and quantum iterations, the quantum iteration overwrites the respective files of the classical iteration.

> **Note**
>
> Both conditions specified by `residual` and `residual_fermi` are only checked between iterations but not between repetitions.

Example

```
run{
    quantum_current_poisson{
        output_local_residuals = yes
    }
}

currents{}
quantum{}
```

7.4.9 quantum_optics{ }

Calling sequence

`run{ quantum_optics{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The *optics{ }* must be defined.
- The *optics{ quantum_spectra{ } }* must be defined.
- The *quantum{ }* must be defined.

Functionality

Calculates optical properties based on solutions of the Schrödinger equation defined within the *quantum{ }* group. Optical spectra are controlled within *optics{ }*, which is also additional redefining selected setting from the *quantum{ }* group.

Example

```
run{
    quantum_optics{}
}

quantum{}
optics{
    quantum_spectra{}
}
```

Remarks

- **Poisson**: Only maximally one of `poisson{ }` and `current_poisson{ }` can be defined, which defines the classical equation to be solved (also as first stage before possibly solving any quantum mechanics). If neither is set, only fixed potentials will be used.
- **Quantum**: If quantum mechanics is desired, one of `quantum{ }`, `quantum_density{}`, `quantum_poisson{ }`, and `quantum_current_poisson{ }` must be set.
- The quantum equations to be solved - only quantum, quantum with self-consistent density/exchange, self-consistent quantum-Poisson, and self-consistent quantum-current-Poisson - are only defined by the choice of `quantum{ }`, `quantum_density{}`, `quantum_poisson{ }`, and `quantum_current_poisson{ }`, irrespective of the choice of the classical solution method. Note that one of `poisson{ }` and `current_poisson{ }` must be set when `quantum_poisson{ }` or `quantum_current_poisson{ }` is desired. Use `poisson{ }` in conjunction with `quantum_current_poisson{ }` to skip classical current calculations.
- Quantum with self-consistent density/exchange is solved by selection of `quantum_density{}` (users can change parameters in there as needed).

Further Remarks

2019-01-24: At the end of `current_poisson{ }`, Poisson is now solved once to make the band structure consistent with the Fermi levels. In case of incomplete convergence, the partly converged output is then more in line with physical intuition.

Input residuals and tolerances are rescaled to various internal units (often in a dimension-dependent manner, i.e. they are different for 1D, 2D and 3D simulations) before being passed to low-level numerical routines like ARPACK, LAPACK, BLAS, nonlinear solvers, etc. Therefore, diagnostic output from low-level numerical solvers usually contains values which are completely different from those which are output by the high-level physics routines or output into files.

There are log files that track the convergence behavior of the iterations during the simulation. The convergence information for the respective self-consistent equations can be plotted. It is best to use a logarithmic scale.

- *iteration_quantum_density.dat*

  `quantum_density{}` | Convergence of Schrödinger equation with self-consistent density/exchange

- *iteration_quantum_poisson.dat*

  `quantum_poisson{ }` | Convergence of outer iteration loop for Schrödinger-Poisson

- *iteration_quantum_current_poisson.dat*

  `quantum_current_poisson{ }` | Convergence of outer iteration loop, i.e. for Current-Poisson-Schrödinger with quantum

- *iteration_quantum_current_poisson_details.dat*

  `quantum_current_poisson{ }` | Convergence of current equation, i.e. for Current-Poisson with quantum densities

7.5 global{ }

Calling sequence

`global{ }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

Contains global settings of the simulation domain.

Example

```
global{
    simulate1D{}
    crystal_zb{
        x_hkl = [1, 0, 0]
        x_hkl = [0, 1, 0]
    }
    substrate{ name = "GaAs" }
    temperature = 300
}
```

- *simulate1D{ }*
- *simulate2D{ }*
- *simulate3D{ }*

- *crystal_zb{ }*
- *crystal_zb{ x_hkl }*
- *crystal_zb{ y_hkl }*
- *crystal_zb{ z_hkl }*
- *crystal_wz{ }*
- *crystal_wz{ x_hkl }*
- *crystal_wz{ y_hkl }*
- *crystal_wz{ z_hkl }*
- *crystal_wz{ rotation_c_a_ratio_use_substrate }*
- *crystal_wz{ rotation_c_a_ratio }*
- *substrate{ }*
- *substrate{ name }*
- *substrate{ alloy_x }*
- *substrate{ alloy_y }*
- *substrate{ alloy_z }*
- *temperature*
- *temperature_dependent_bandgap*
- *temperature_dependent_lattice*
- *magnetic_field{ }*
- *magnetic_field{ direction }*
- *magnetic_field{ strength }*
- *periodic{ }*
- *periodic{ x }*
- *periodic{ y }*
- *periodic{ z }*

7.5.1 simulate1D{ }

Calling sequence

`global{ simulate1D{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- Exactly one out of *simulate1D{ }*, *simulate2D{ }*, and *simulate3D{ }* must be defined.

Functionality

Instructs the solver that the simulation will be held in 1D, along the x-direction.

Example

```
global{
    simulate1D{}
    ...
}
```

7.5.2 simulate2D{ }

Calling sequence

`global{ simulate2D{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- Exactly one out of *simulate1D{ }*, *simulate2D{ }*, and *simulate3D{ }* must be defined.

Functionality

Instructs the solver that the simulation will be held in 2D, within the (x,y)-plane.

Example

```
global{
    simulate2D{}
    ...
}
```

7.5.3 simulate3D{ }

Calling sequence

`global{ simulate3D{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- Exactly one out of *simulate1D{ }*, *simulate2D{ }*, and *simulate3D{ }* must be defined.

Functionality

Instructs the solver that the simulation will be held in 3D, within the (x,y,z)-volume.

Example

```
global{
    simulate3D{}
    ...
}
```

7.5.4 crystal_zb{ }

Calling sequence

`global{ crystal_zb{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Instructs the tool that models and routines for zincblende (including diamond) materials should be used within the simulation. Organizes keywords to define orientation of the crystal coordinate system with respect to simulation coordinate system.

Example

```
global{
    crystal_zb{...}
    ...
}
```

7.5.5 crystal_zb{ x_hkl }

Calling sequence

`global{ crystal_zb{ x_hkl } }`

Dependencies

- Exactly two out of *crystal_zb{ x_hkl }*, *crystal_zb{ y_hkl }*, and *crystal_zb{ z_hkl }* must be defined.

Properties

- **usage:** conditional
- **type:** vector of 3 integers: (z_1, z_2, z_3)
- **values:** no constraints
- **unit:** $-$

Functionality

Miller indices specifying a lattice plane that should be set perpendicular to the x-axis of the simulation coordinate system.

Note

See *Crystal coordinate systems* for more details.

Example

```
global{
    crystal_zb{
        x_hkl = [1, 1, 1]
        y_hkl = [-1, 2, -1]
    }
    ...
}
```

7.5.6 crystal_zb{ y_hkl }

Calling sequence

`global{ crystal_zb{ y_hkl } }`

Properties

- **usage:** conditional
- **type:** vector of 3 integers: (z_1, z_2, z_3)
- **values:** no constraints
- **unit:** $-$

Dependencies

- Exactly two out of *crystal_zb{ x_hkl }*, *crystal_zb{ y_hkl }*, and *crystal_zb{ z_hkl }* must be defined.

Functionality

Miller indices specifying a lattice plane that should be set perpendicular to the y-axis of the simulation coordinate system.

Note

See *Crystal coordinate systems* for more details.

Example

```
global{
    crystal_zb{
        x_hkl = [0, -1, 1]
        y_hkl = [1, 1, 0]
    }
    ...
}
```

7.5.7 crystal_zb{ z_hkl }

Calling sequence

`global{ crystal_zb{ z_hkl } }`

Properties

- **usage:** conditional
- **type:** vector of 3 integers: (z_1, z_2, z_3)
- **values:** no constraints
- **unit:** $-$

Dependencies

- Exactly two out of *crystal_zb{ x_hkl }*, *crystal_zb{ y_hkl }*, and *crystal_zb{ z_hkl }* must be defined.

Functionality

Miller indices specifying a lattice plane that should be set perpendicular to the z-axis of the simulation coordinate system.

> **Note**
>
> See *Crystal coordinate systems* for more details.

Example

```
global{
    crystal_zb{
        y_hkl = [1, -1, 0]
        z_hkl = [0, 1, -1]
    }
    ...
}
```

7.5.8 crystal_wz{ }

Calling sequence

`global{ crystal_wz{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Instructs the tool that models and routines for wurtzite materials should be used within the simulation. Organizes keywords to define orientation of the crystal coordinate system with respect to simulation coordinate system.

Example

```
global{
    crystal_wz{...}
    ...
}
```

7.5.9 crystal_wz{ x_hkl }

Calling sequence

`global{ crystal_wz{ x_hkl } }`

Properties

- **usage:** conditional
- **type:** vector of 3 integers: (z_1, z_2, z_3)
- **values:** no constraints
- **unit:** –

Functionality

Miller indices specifying a lattice plane that should be set perpendicular to the x-axis of the simulation coordinate system.

Note

See *Crystal coordinate systems* for more details.

Example

```
global{
    crystal_wz{
        x_hkl = [ 0, 0, 1]
        y_hkl = [ 1, 0, 0]
    }
    ...
}
```

7.5.10 crystal_wz{ y_hkl }

Calling sequence

`global{ crystal_wz{ y_hkl } }`

Properties

- **usage:** conditional
- **type:** vector of 3 integers: (z_1, z_2, z_3)
- **values:** no constraints
- **unit:** –

Functionality

Miller indices specifying a lattice plane that should be set perpendicular to the y-axis of the simulation coordinate system.

> **Note**
>
> See *Crystal coordinate systems* for more details.

Example

```
global{
    crystal_wz{
        x_hkl = [ 1, 0, 0]
        y_hkl = [ -1, 2, 0]
    }
    ...
}
```

7.5.11 crystal_wz{ z_hkl }

Calling sequence

`global{ crystal_wz{ z_hkl } }`

Properties

- **usage:** conditional
- **type:** vector of 3 integers: (z_1, z_2, z_3)
- **values:** no constraints
- **unit:** $-$

Functionality

Miller indices specifying a lattice plane that should be set perpendicular to the z-axis of the simulation coordinate system.

> **Note**
>
> See *Crystal coordinate systems* for more details.

Example

```
global{
    crystal_wz{
        x_hkl = [ 0, 0, 1]
        z_hkl = [ 1, 1, 0]
    }
    ...
}
```

7.5.12 crystal_wz{ rotation_c_a_ratio_use_substrate }

Calling sequence

`global{ crystal_wz{ rotation_c_a_ratio_use_substrate } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If `rotation_c_a_ratio_use_substrate = yes` then ratio of lattice constants a and c in wurtzite crystal to perform crystal coordination system rotation is computed based on the lattice constants of the substrate material. Otherwise the ratio *crystal_wz{ rotation_c_a_ratio }* is used.

Example

```
global{
    crystal_wz{
        rotation_c_a_ratio_use_substrate = no
        ...
    }
    ...
}
```

7.5.13 crystal_wz{ rotation_c_a_ratio }

Calling sequence

`global{ crystal_wz{ rotation_c_a_ratio } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $1.0 \leq r \leq 2.0$
- **default:** $r = \sqrt(8.0/3.0)$
- **unit:** $-$

Functionality

If the ratio for entering rotation matrix is not computed based on lattice constants of the substrate material `rotation_c_a_ratio_use_substrate = no`, then the default value or value assigned to this keyword is used.

Example

```
global{
    crystal_wz{
        rotation_c_a_ratio = 5.185 / 3.189
        ...
```

(continues on next page)

(continued from previous page)

```
    }
    ...
}
```

7.5.14 substrate{ }

Calling sequence

`global{ substrate{ } }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

Organizes keywords specifying parameters of the substrate material. The substrate enters the simulation as a reference unstrained material onto which the simulated structure is grown, and strained to, if the *strain{ }* is triggered. It also strongly impacts the symmetry of the first Brillouin zone, therefore also symmetry properties of all integrals over the space of the wave vector.

Example

```
global{
    substrate{...}
    ...
}
```

7.5.15 substrate{ name }

Calling sequence

`global{ substrate{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

The reference name of the material to be used as the substrate. The name must exist either in the linked database (see *Material Database*) file or in the *database{ }* group in the input file.

Example

```
global{
    substrate{
        name = "GaAs"
    }
    ...
}
```

7.5.16 substrate{ alloy_x }

Calling sequence

`global{ substrate{ alloy_x } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** $0.0 \leq r \leq 1.0$
- **unit:** $-$

Dependencies

- This group is required if any of the groups *substrate{ alloy_y }* and *substrate{ alloy_z }* is already present.

Functionality

If a name of at least *two-component alloy* is assigned to *substrate{ name }*, then this parameter defines the mole fraction "x" of the alloy.

Example

```
global{
    substrate{
        name = "Al(x)Ga(1-x)As"
        alloy_x = 0.3
    }
    ...
}
```

7.5.17 substrate{ alloy_y }

Calling sequence

`global{ substrate{ alloy_y } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** $0.0 \leq r \leq 1.0$
- **unit:** $-$

Functionality

If a name of at least *three-component alloy* is assigned to *substrate{ name }*, then this parameter defines the mole fraction "y" of the alloy.

Example

```
global{
    substrate{
        name = "Al(x)Ga(y)In(1-x-y)As"
        alloy_x = 0.3
        alloy_y = 0.1
    }
    ...
}
```

7.5.18 substrate{ alloy_z }

Calling sequence

`global{ substrate{ alloy_z } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.0 \leq r \leq 1.0$
- **unit:** $-$

Functionality

If a name of at least *six-component alloy* is assigned to *substrate{ name }*, then this parameter defines the mole fraction "z" of the alloy.

Example

```
global{
    substrate{
        name = "Al(x)Ga(y)In(1-x-y)As(z)P(1-z)"
        alloy_x = 0.3
        alloy_y = 0.1
        alloy_z = 0.9
    }
    ...
}
```

7.5.19 temperature

Calling sequence

`global{ temperature }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** K

Functionality

Specifies temperature of the crystal lattice and electrons.

Example

```
global{
    temperature = 300
    ...
}
```

7.5.20 temperature_dependent_bandgap

Calling sequence

`global{ temperature_dependent_bandgap }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If `temperature_dependent_bandgap = yes` then Varshni formula is used to calculate the band gap $E_\mathrm{g}(T)$ at given temperature T

$$E_\mathrm{g}\left(T\right) = E_\mathrm{g}(0\ \mathrm{K}) + \delta E_\mathrm{g}\left(T\right),$$

where $\delta E_\mathrm{g}\left(T\right)$ is the correction.

For pure materials, such as GaAs or Si, the correction is given as:

$$\delta E_\mathrm{g}\left(T\right) = \frac{-\alpha T^2}{T+\beta}$$

In case of alloys, the correction is interpolated consistently with our *Interpolation Schemes*. For instance, the correction for a two-component alloy with bowing parameters is computed as:

$$\delta E_{\mathrm{g,ABC}}\left(T,x\right) = x\frac{-\alpha_\mathrm{AC}\,T^2}{T+\beta_\mathrm{AC}} + (1-x)\frac{-\alpha_\mathrm{BC}\,T^2}{T+\beta_\mathrm{BC}} - x(1-x)\frac{-\alpha_\mathrm{ABC}\,T^2}{T+\beta_\mathrm{ABC}},$$

Where α_AC, α_BC, β_AC, and β_BC are Varshni parameters of binary compounds AC and AB, while α_ABC and β_ABC are related "bowing" parameters, all defined in the database. The latter ones are typically set to zero.

If `temperature_dependent_bandgap = no` then the energy gap from the database is taken without any corrections, assumed to be for 0 K.

Note

The temperature dependence impacts only the conduction bands, since the valence bands are used as reference energies for the band offsets.

Example

```
global{
    temperature_dependent_bandgap = no
    ...
}
```

7.5.21 temperature_dependent_lattice

Calling sequence

`global{ temperature_dependent_lattice }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If `temperature_dependent_lattice = yes` then the linear temperature dependence is included for the lattice constants. Otherwise, the lattice constant defined in the database as for 300 K is used.

$$a\left(T\right) = a(300\text{ K}) + a_{\text{exp}}\left(T - 300\right),$$

where a_{exp} is the expansion coefficient defined in the database and properly interpolated for alloys.

Example

```
global{
    temperature_dependent_lattice = no
    ...
}
```

7.5.22 magnetic_field{ }

Calling sequence

`global{ magnetic_field{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The groups *magnetic_field{ }* and *periodic{ }* cannot be defined simultaneously.

Functionality

Adds magnetic effects to the quantum solver (see. *quantum{ }*) for the single-band, the 6-band $\mathbf{k} \cdot \mathbf{p}$ and the 8-band $\mathbf{k} \cdot \mathbf{p}$ Hamiltonians, such that effectively the Pauli equation is solved.

Note

The single-band model is assumed to be two-fold spin degenerate without the magnetic field. This assumption is lifted when the *magnetic_field{ }* is defined.

Attention

The magnetic effects are not yet included directly in the drift-diffusion current equations, therefore the Hall effect is not covered by this model. Please contact us if you need this feature.

Attention

For the magnetic effects are not fully implemented for 1D simulations. While the Zeeman Splitting seem to be captured properly, other quantities, such as carrier densities, may be computed improperly. Please contact us if you need this feature.

Example

```
global{
    magnetic_field{...}
    ...
}
```

7.5.23 magnetic_field{ direction }

Calling sequence

`global{ magnetic_field{ direction } }`

Properties

- **usage:** conditional
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **default 1D:** $r_1 = 1.0$, $r_2 = 0.0$, $r_3 = 0.0$
- **default 2D:** $r_1 = 0.0$, $r_2 = 0.0$, $r_3 = 1.0$
- **unit:** $-$

Dependencies

- This group is not allowed if any of the groups *simulate1D{ }* and *simulate2D{ }* is already present.
- On the other hand, it is required if *simulate3D{ }* is defined.

Functionality

Defines orientation of constant homogenous magnetic field (magnetic induction $\mathbf{B}$) vector with respect to the simulation coordinate system.

Example

```
global{
    simulate3D{}
    magnetic_field{
        direction = direction = [3, 1, 1]
        strength  = 5.3
    }
    ...
}
```

7.5.24 magnetic_field{ strength }

Calling sequence

`global{ magnetic_field{ strength } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** $\mathrm{T} = \mathrm{Vs}/\mathrm{m}^2$

Functionality

Sets the strength of the constant magnetic field $\mathbf{B}$.

Hint

It is better to not define the group *magnetic_field{ }* instead of setting `strength = 0` for 1-band simulations, as including the magnetic effects is extending the 1-band model by spin. This extension results in longer runtime of the quantum solver.

Example

```
global{
    simulate1D{}
    magnetic_field{
        strength  = 5.3
    }
    ...
}
```

7.5.25 periodic{ }

Calling sequence

`global{ periodic{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The groups *magnetic_field{ }* and *periodic{ }* cannot be defined simultaneously.

Functionality

When defined, allows triggering periodic boundary conditions for the entire simulation domain along selected directions of the simulation coordinate system. These boundary conditions are applied to strain, electrostatic field (the Poisson equation), and wave functions (the Schrödinger equation) overwriting all the other possible definitions already present in the input file.

> **Note**
>
> The periodic boundary conditions will be imposed on the Schrödinger equation only if related quantum region extends over the entire simulation domain along the relevant direction.

> **Note**
>
> Shapes defining the layout of materials (*structure{ region{} } - shape objects*) which extends beyond the defined simulation domain are not automatically continued on the opposite side of the simulation domain.

Example

```
global{
    periodic{...}
    ...
}
```

7.5.26 periodic{ x }

Calling sequence

`global{ periodic{ x } }`

Properties

- **usage:** required
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `x = yes` then the periodic boundary conditions are applied along the x-axis of the simulation coordinate system to the most outer points of the grid. Otherwise, other default or defined elsewhere boundary conditions apply.

Example

```
global{
    simulate1D{}
    periodic{
        x = yes
    }
    ...
}
```

7.5.27 periodic{ y }

Calling sequence

`global{ periodic{ y } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- This group is required if *simulate2D{ }* or *simulate3D{ }* is specified.
- It is not allowed if *simulate1D{ }* is defined.

Functionality

If `y = yes` then the periodic boundary conditions are applied along the y-axis of the simulation coordinate system to the most outer points of the grid. Otherwise, other default or defined elsewhere boundary conditions apply.

Example

```
global{
    simulate2D{}
    periodic{
        x = no
        y = yes
    }
    ...
}
```

7.5.28 periodic{ z }

Calling sequence

`global{ periodic{ z } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- This group is required if *simulate3D{ }* is specified.
- It is not allowed if *simulate1D{ }* or *simulate2D{ }* is defined.

Functionality

If `z = yes` then the periodic boundary conditions are applied along the z-axis of the simulation coordinate system to the most outer points of the grid. Otherwise, other default or defined elsewhere boundary conditions apply.

Example

```
global{
    simulate3D{}
    periodic{
        x = yes
        y = no
        z = yes
    }
    ...
}
```

7.6 impurities{ }

Calling sequence

`impurities{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Specifies properties of impurities (donors, acceptor and fixed charges)

Example

```
impurities{
    donor{...}
    donor{...}
    acceptor{...}
}
```

Nested keywords

- *donor{ }*
- *donor{ name }*
- *donor{ degeneracy }*
- *donor{ energy }*
- *donor{ N_ref }*
- *donor{ c }*
- *acceptor{ }*
- *acceptor{ name }*
- *acceptor{ degeneracy }*
- *acceptor{ energy }*
- *acceptor{ N_ref }*
- *acceptor{ c }*
- *charge{ }*
- *charge{ name }*
- *charge{ type }*

7.6.1 donor{ }

Calling sequence

`impurities{ donor{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines properties of donors.

Example

```
impurities{
    donor{...}
    donor{...}
}
```

7.6.2 donor{ name }

Calling sequence

`impurities{ donor{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Name of the impurity for referencing during definition of the structure

Example

```
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

7.6.3 donor{ degeneracy }

Calling sequence

`impurities{ donor{ degeneracy } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $1 \leq z \leq 12$
- **unit:** $-$

Functionality

Degeneracy of the impurity. It affects the degree of ionization.

> **Note**
>
> The degeneracy of donors is usually assumed to be equal to 2 - degeneracy factor is 2. Outer s orbital is one-fold occupied (neutral state). There is one possibility to get rid of one electron, but there are two to incorporate one (spin up, spin down). More details on degenerate impurity levels can be found in e.g. *[ChuangOpto1995]*.

Example

```
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        ...
    }
}
```

7.6.4 donor{ energy }

Calling sequence

`impurities{ donor{ energy } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

Ionization (activation) energy of the impurity, $E_{ion}^{(0)}$. The positive value means that the donor level is located below the conduction band edge, while the negative value means that the level is located within the conduction band. See *Doping* for reference on typical activation energies.

> **Hint**
>
> The negative value can be used to force full ionization of donors despite the quasi-Fermi levels. The degeneracy factor effectively becomes irrelevant under the full ionization. This can be seen from eqs. $(1.4)-(1.7)$ in *[BirnerPhD2011]*.

Example

```
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
```

Available in the next release.

7.6.5 donor{ N_ref }

Calling sequence

`impurities{ donor{ N_ref } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[1e10, ...)`
- **default:** $r =$ infinity
- **unit:** cm^{-3}

Functionality

Reference doping N_{ref} for doping-density-dependent activation energy

$$E_{ion} = E_{ion}^{(0)} \times \left[1 - \left[\frac{N_{A,0} + N_{D,0}}{N_{ref}}\right]^c\right]$$

where $N_{D,0}$ and $N_{A,0}$ are donor and acceptor densities, and $E_{ion}^{(0)}$ is ionization energy of a dopant at low doping concentrations.

Available in the next release.

7.6.6 donor{ c }

Calling sequence

`impurities{ donor{ c } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.3 \leq r \leq 1.0$
- **default:** $r = 1.0/3.0$
- **unit:** $-$

Functionality

Exponent c for doping-density-dependent activation energy formula.

7.6.7 acceptor{ }

Calling sequence

`impurities{ acceptor{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines properties of acceptors.

Example

```
impurities{
    acceptor{...}
    acceptor{...}
}
```

7.6.8 acceptor{ name }

Calling sequence

`impurities{ acceptor{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Name of the impurity for referencing during definition of the structure

Example

```
impurities{
    acceptor{
        name = "p-C"
        ...
    }
}
```

7.6.9 acceptor{ degeneracy }

Calling sequence

`impurities{ acceptor{ degeneracy } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $1 \leq z \leq 12$
- **unit:** $-$

Functionality

Degeneracy of the impurity. It affects the degree of ionization.

> **Note**
>
> The degeneracy of acceptors is usually assumed to be equal to 4 - degeneracy factor is 4. The sp^3 orbital is threefold occupied. Thus, one possibility to incorporate an electron, four possibilities to get rid of one. More details on degenerate impurity levels can be found in e.g. *[ChuangOpto1995]*.
>
> The degeneracy factor may vary from 4 to 6 in nitride semiconductors crystallizing in the wurtzite structure because of a small valence band splitting.

Example

```
impurities{
    acceptor{
        name = "p-C"
        degeneracy = 4
        ...
    }
}
```

7.6.10 acceptor{ energy }

Calling sequence

`impurities{ acceptor{ energy } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

Ionization (activation) energy of the impurity. The positive value means that the acceptor level is located above the valence band edge, while the negative value means that the level is located within the valence band. See *Doping* for reference on typical activation energies.

Hint

The negative value can be used to force full ionization of acceptors despite the quasi-Fermi levels. The degeneracy factor effectively becomes irrelevant under the full ionization. This can be seen from eqs. (1.4) – (1.7) in *[BirnerPhD2011]*.

Example

```
impurities{
    acceptor{
        name = "p-C"
        degeneracy = 4
        energy = 0.027
    }
}
```

Available in the next release.

7.6.11 acceptor{ N_ref }

Calling sequence

`impurities{ acceptor{ N_ref } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[1e10, ...)`
- **default:** $r =$ infinity
- **unit:** cm^{-3}

Functionality

Reference doping N_{ref} for doping-density-dependent activation energy

$$E_{ion} = E_{ion}^{(0)} \times \left[1 - \left[\frac{N_{A,0} + N_{D,0}}{N_{ref}}\right]^{c}\right]$$

where $N_{D,0}$ and $N_{A,0}$ are donor and acceptor densities, and $E_{ion}^{(0)}$ is ionization energy of a dopant at low doping concentrations.

Available in the next release.

7.6.12 acceptor{ c }

Calling sequence

`impurities{ acceptor{ c } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.3 \leq r \leq 1.0$
- **default:** $r = 1.0/3.0$
- **unit:** $-$

Functionality

Exponent c for doping-density-dependent activation energy formula.

7.6.13 charge{ }

Calling sequence

`impurities{ charge{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines the type of charges which can be used to add positive or negative charges into the device, e.g., to describe interface charges.

Example

```
impurities{
    charge{...}
    charge{...}
}
```

7.6.14 charge{ name }

Calling sequence

`impurities{ charge{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A reference name.

Example

```
impurities{
    charge{
        name = "positive_charges"
        ...
    }
}
```

7.6.15 charge{ type }

Calling sequence

`impurities{ charge{ type } }`

Properties

- **usage:** required
- **type:** choice
- **values:** `positive` or `negative`

Functionality

Defines sign of the charge.

Example

```
impurities{
    charge{
        name = "positive_charges"
        type = positive
    }
}
```

7.7 contacts{ }

Calling sequence

`contacts{ }`

Properties

- **usage:** required
- **items:** exactly 1

Dependencies

- At least one of *fermi{ }*, *fermi_electron{ }*, *fermi_hole{ }*, *schottky{ }*, *ohmic{ }*, *zero_field{ }*, and *charge_neutral{ }* must be defined.

Functionality

Defines available boundary conditions for the **Current** and **Poisson** equations. These conditions can be assigned to specific regions by referring to assigned attribute **name**. We use the name **contacts** for these boundary conditions since typically they are chosen as the most outer regions of the structures aiming at simulating real contacts of some devices. It is, however, important to remember that whether these regions correspond to any contact in a real device or not depends on how semiconductors behave near the contact at specific conditions. To model the **contacts** properly, some knowledge about physics around contacts, specifically about Fermi levels and (or) electric potential, in the modeled device is required and should be applied as the boundary conditions for our solver.

All available groups for specifying boundary conditions for the **Current** and **Poisson** equations are described below. It is important to remember that, on top of them, the global boundary conditions are applied to the electrostatic potential $\phi(x)$ and quasi-Fermi levels $E_{F,e/h}(x)$ at the boundaries of the entire simulation. These are either, default, **Neumann** boundary conditions ($\frac{\mathrm{d}}{\mathrm{d}x}\phi(x) = 0$ and $\frac{\mathrm{d}}{\mathrm{d}x}E_{F,e/h}(x) = 0$) or *periodic* boundary conditions.

Attention

At each grid point, only one type of contact can exist. For overlapping contact regions, the last defined contact on this grid point is used.

Example

```
# In this example, there are three bias configurations computed
# 1) gate = 0.0 V, source = 0.0 V, drain = 0.0 V
# 2) gate = 0.5 V, source = 0.0 V, drain = 0.2 V
# 3) gate = 1.0 V, source = 0.0 V, drain = 0.2 V

contacts{
    vacuum_level = 6.2
    schottky{
        name = "gate"
        bias = [ 0.0, 0.5, 1.0]
        barrier = 0.2
    }
    ohmic{
        name = "source"
        bias = 0.0
    }
    ohmic{
        name = "drain"
        bias = [ 0.0, 0.2]
```

(continues on next page)

(continued from previous page)

```
    }
    bias_steps = 1
}
```

```
# In this example, there are three bias configurations computed
# 1) gate = 0.00 V, source = 0.0 V, drain = 0.0 V (as with bias_steps = 0)
# 2) gate = 0.00 V, source = 0.0 V, drain = 0.1 V
# 3) gate = 0.50 V, source = 0.0 V, drain = 0.2 V (as with bias_steps = 0)
# 4) gate = 0.75 V, source = 0.0 V, drain = 0.2 V
# 3) gate = 1.00 V, source = 0.0 V, drain = 0.2 V (as with bias_steps = 0)

contacts{
    vacuum_level = 6.2
    schottky{
        name = "gate"
        bias = [ 0.0, 0.5, 1.0]
        barrier = 0.2
    }
    ohmic{
        name = "source"
        bias = 0.0
    }
    ohmic{
        name = "drain"
        bias = [ 0.0, 0.2]
    }
    bias_steps = 2
}
```

Nested keywords

- *vacuum_level*
- *schottky{ }*
- *schottky{ name }*
- *schottky{ bias }*
- *schottky{ barrier }*
- *schottky{ work_function }*
- *ohmic{ }*
- *ohmic{ name }*
- *ohmic{ bias }*
- *ohmic{ shift }*
- *fermi{ }*
- *fermi{ name }*
- *fermi{ bias }*
- *fermi_electron{ }*
- *fermi_electron{ name }*

- *fermi_electron{ bias }*
- *fermi_hole{ }*
- *fermi_hole{ name }*
- *fermi_hole{ bias }*
- *charge_neutral{ }*
- *charge_neutral{ name }*
- *charge_neutral{ bias }*
- *zero_field{ }*
- *zero_field{ name }*
- *zero_field{ bias }*
- *long_directory_names*
- *bias_steps*
- *reuse_previous*
- *bias_output_level*

7.7.1 vacuum_level

Calling sequence

`contacts{ vacuum_level }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $-10^2 \leq r \leq 10^2$
- **default:** $r = 6.3$
- **unit:** eV

Functionality

Energy of vacuum level E_{vac}, used for *schottky{ }*. The value 6.3 eV is predefined in correspondence to the band offsets in the default database.

7.7.2 schottky{ }

Calling sequence

`contacts{ schottky{ } }`

Properties

- **usage:** conditional
- **items:** no constraints

Dependencies

- Exatly one of *schottky{ barrier }* and *schottky{ work_function }* must be defined within this group.

Functionality

This keyword applies **Dirichlet** boundary conditions to the Fermi levels $E_{F,e}(x)$ and $E_{F,h}(x)$

$$E_{F,e}(x) = E_{F,h}(x) = -qU,$$

where q is the elementary charge and U is an explicitly defined bias, and **Dirichlet** boundary conditions to the electrostatic potential $\phi(x)$

$$\phi(x) = \phi_0,$$

where ϕ_0 is determined numerically by requiring that the difference of the conduction band edge $E_c^\Gamma(x)$ and the Fermi level E_F is equal to the value of given Schottky barrier B,

$$E_c^\Gamma(x) - E_F = B,$$

or by requiring that the difference of the vacuum level E_{vac} and the Fermi level E_F is equal to the value of given work function W,

$$E_{vac} - E_F = W.$$

Attention

The Schottky contact with *schottky{ barrier }* defined requires paying attention to the material chosen for the region of the contact, as this material is used as a reference for the definition.

7.7.3 schottky{ name }

Calling sequence

`contacts{ schottky{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A name of the contact for referencing it in *contact{ }*

7.7.4 schottky{ bias }

Calling sequence

`contacts{ schottky{ bias } }`

Properties

- **usage:** required
- **type:** vector of up to 100 real numbers: (r_1), (r_1, r_2), ldots, $(r_1, r_2, \ldots, r_{100})$
- **values:** no constraints
- **unit:** V

Functionality

Explicitly defined set of values set to both electron and quasi-Fermi levels as minus `bias` multiplied by the electron charge. Often it corresponds directly to applied voltage.

7.7.5 schottky{ barrier }

Calling sequence

`contacts{ schottky{ barrier } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** $-10^2 \leq r \leq 10^2$
- **unit:** eV

Functionality

A Schottky barrier B - a difference between conduction band minimum and the Fermi level

---

7.7.6 schottky{ work_function }

Calling sequence

`contacts{ schottky{ work_function } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** $-10^2 \leq r \leq 10^2$
- **unit:** eV

Functionality

Work function W - a difference between vacuum level and the Fermi level The Schottky-Mott is be used to determine the barrier height of the contact.

Note

Due to Fermi level pinning, experimentally measured Schottky barrier heights may be quite different.

Hint

You can check the section about *Band Offsets* to estimate the energy of vacuum level in respect to band extrema of materials in your simulation.

Hint

This keyword can be successfully used to model the effect of Fermi level pinning due to surface states under equilibrium conditions.

7.7.7 ohmic{ }

Calling sequence

`contacts{ ohmic{ } }`

Properties

- **usage:** conditional
- **items:** no constraints

Functionality

This keyword applies **Dirichlet** boundary conditions to the electrostatic potential $\phi(x)$

$$\phi(x) = \phi_0,$$

where ϕ_0 is determined numerically by requiring local charge neutrality for each grid point of the contact if the **shift** parameter $\Delta U = 0$, and **Dirichlet** boundary conditions to the Fermi levels $E_{F,e}(x)$ and $E_{F,h}(x)$

$$E_{F,e}(x) = E_{F,h}(x) = -qU,$$

where q is the elementary charge and U is an explicitly defined bias. If $\Delta U \neq 0$ then, after the procedure described above, band edges are moved by the value $-q\Delta U$ and ϕ_0 is recalculated.

Attention

Material under the **ohmic** contact influences computing charge neutrality conditions. Therefore, one should pay attention to the material (and doping) chosen for the region of this type of contact.

Note

Check *bisection{ }* to learn about applied algorithm for definition of quasi-Fermi levels in this contact.

7.7.8 ohmic{ name }

Calling sequence

`contacts{ ohmic{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A name of the contact for referencing it in *contact{ }*.

7.7.9 ohmic{ bias }

Calling sequence

`contacts{ ohmic{ bias } }`

Properties

- **usage:** required
- **type:** vector of up to 100 real numbers: (r_1), (r_1, r_2), ldots, $(r_1, r_2, \ldots, r_{100})$
- **values:** no constraints
- **unit:** V

Functionality

Explicitly defined set of values set to both electron and quasi-Fermi levels as minus `bias` multiplied by the electron charge. Often it corresponds directly to applied voltage.

7.7.10 ohmic{ shift }

Calling sequence

`contacts{ ohmic{ shift } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** eV

Functionality

Shift potential energy of the bands ΔU.

Hint

You may find this keyword useful to calculate the energy levels in a quantum well (QW) or quantum cascade laser (QCL) as a function of applied bias.

Note

Check *bisection{ }* to learn about applied algorithm for definition of quasi-Fermi levels in this contact.

7.7.11 fermi{ }

Calling sequence

`contacts{ fermi{ } }`

Properties

- **usage:** conditional
- **items:** no constraints

Functionality

Applies **Dirichlet** boundary conditions to the Fermi levels $E_{F,e}(x)$ and $E_{F,h}(x)$

$$E_{F,e}(x) = E_{F,h}(x) = -qU,$$

where q is the elementary charge and U is an explicitly defined bias. No boundary conditions are specified for the electrostatic potential $\phi(x)$.

7.7.12 fermi{ name }

Calling sequence

`contacts{ fermi{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A name of the contact for referencing it in *contact{ }*

Attention

When triggered, both Poisson and Schrödinger equations are solved in the regions with these boundary conditions.

7.7.13 fermi{ bias }

Calling sequence

`contacts{ fermi{ bias } }`

Properties

- **usage:** required
- **type:** vector of up to 100 real numbers: (r_1), (r_1, r_2), ldots, $(r_1, r_2, \ldots, r_{100})$
- **values:** no constraints
- **unit:** V

Functionality

Explicitly defined set of values set to both electron and quasi-Fermi levels as minus `bias` multiplied by the electron charge. Often it corresponds directly to applied voltage.

7.7.14 fermi_electron{ }

Calling sequence

`contacts{ fermi_electron{ } }`

Properties

- **usage:** conditional
- **items:** no constraints

Functionality

This keyword applies only **Dirichlet** boundary conditions to the quasi-Fermi level for electrons $E_{F,e}(x)$

$$E_{F,e}(x) = -qU,$$

where q is the elementary charge and U is an explicitly defined bias. No boundary conditions are specified for the electrostatic potential $\phi(x)$ and for quasi-Fermi level for holes $E_{F,h}(x)$.

> ⚠ **Attention**
>
> As quasi-Fermi level for holes $E_{F,h}(x)$ is not defined within this group, other contacts are necessary to do so.

> ⚠ **Attention**
>
> When triggered, both Poisson and Schrödinger equations are solved in the regions with these boundary conditions.

> ⚠ **Warning**
>
> Using *fermi_electron{ }* and *fermi_hole{ }* for current calculations within biased structures may cause violation of charge conservation.

7.7.15 fermi_electron{ name }

Calling sequence

`contacts{ fermi_electron{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A name of the contact for referencing it in *contact{ }*

7.7.16 fermi_electron{ bias }

Calling sequence

`contacts{ fermi_electron{ bias } }`

Properties

- **usage:** required
- **type:** vector of up to 100 real numbers: (r_1), (r_1, r_2), ldots, $(r_1, r_2, \ldots, r_{100})$
- **values:** no constraints
- **unit:** V

Functionality

Explicitly defined set of values set to both electron and quasi-Fermi levels as minus `bias` multiplied by the electron charge. Often it corresponds directly to applied voltage.

7.7.17 fermi_hole{ }

Calling sequence

`contacts{ fermi_hole{ } }`

Properties

- **usage:** conditional
- **items:** no constraints

Functionality

This keyword applies only **Dirichlet** boundary conditions to the quasi-Fermi level for holes $E_{F,h}(x)$

$$E_{F,h}(x) = -qU,$$

where q is the elementary charge and U is an explicitly defined bias. No boundary conditions are specified for the electrostatic potential $\phi(x)$ and for quasi-Fermi level for electrons $E_{F,e}(x)$.

> ⚠ **Warning**
>
> Using *fermi_electron{ }* and *fermi_hole{ }* for current calculations within biased structures may cause violation of charge conservation.

7.7.18 fermi_hole{ name }

Calling sequence

`contacts{ fermi_hole{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A name of the contact for referencing it in *contact{ }*

Attention

As quasi-Fermi level for electrons $E_{F,e}(x)$ is not defined within this group, other contacts are necessary to do so.

Attention

When triggered, both Poisson and Schrödinger equations are solved in the regions with these boundary conditions.

7.7.19 fermi_hole{ bias }

Calling sequence

```
contacts{ fermi_hole{ bias } }
```

Properties

- **usage:** required
- **type:** vector of up to 100 real numbers: (r_1), (r_1, r_2), ldots, $(r_1, r_2, \ldots, r_{100})$
- **values:** no constraints
- **unit:** V

Functionality

Explicitly defined set of values set to both electron and quasi-Fermi levels as minus `bias` multiplied by the electron charge. Often it corresponds directly to applied voltage.

7.7.20 charge_neutral{ }

Calling sequence

```
contacts{ charge_neutral{ } }
```

Properties

- **usage:** conditional
- **items:** no constraints

Functionality

This keyword applies **Dirichlet** boundary conditions to the electrostatic potential $\phi(x)$

$$\phi(x) = \phi_0,$$

where ϕ_0 determined numerically by requiring local charge neutrality for each grid point of the contact, and **Dirichlet** boundary conditions to the Fermi levels $E_{F,e}(x)$ and $E_{F,h}(x)$

$$E_{F,e}(x) = E_{F,h}(x) = -qU,$$

where q is the elementary charge and U is an explicitly defined bias.

Attention

Material under the **Charge-Neutral** contact influences computing charge neutrality conditions. Therefore, one should pay attention to the material (and doping) chosen for the region of this type of contact.

Hint

You may find this keyword useful to calculate the energy levels in a quantum well (QW) or quantum cascade laser (QCL) as a function of applied bias.

Note

Check *bisection{ }* to learn about applied algorithm for definition of quasi-Fermi levels in this contact.

7.7.21 charge_neutral{ name }

Calling sequence

`contacts{ charge_neutral{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A name of the contact for referencing it in *contact{ }*.

7.7.22 charge_neutral{ bias }

Calling sequence

`contacts{ charge_neutral{ bias } }`

Properties

- **usage:** required
- **type:** vector of up to 100 real numbers: (r_1), (r_1, r_2), ldots, $(r_1, r_2, \ldots, r_{100})$
- **values:** no constraints
- **unit:** V

Functionality

Explicitly defined set of values set to both electron and quasi-Fermi levels as minus `bias` multiplied by the electron charge. Often it corresponds directly to applied voltage.

7.7.23 zero_field{ }

Calling sequence

`contacts{ zero_field{ } }`

Properties

- **usage:** conditional
- **items:** no constraints

Functionality

This keyword applies **Neumann** boundary conditions to the electrostatic potential $\phi(x)$

$$\frac{\mathrm{d}}{\mathrm{d}x}\phi(x) = 0,$$

and **Dirichlet** boundary conditions to the Fermi levels $E_{F,e}(x)$ and $E_{F,h}(x)$

$$E_{F,e}(x) = E_{F,h}(x) = -qU,$$

where q is the elementary charge and U is an explicitly defined bias.

Attention

Material under the **Zero-Field** contact influences computing charge neutrality conditions. Therefore, one should pay attention to the material (and doping) chosen for the region of this type of contact.

7.7.24 zero_field{ name }

Calling sequence

`contacts{ zero_field{ name } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A name of the contact for referencing it in *contact{ }*

Attention

Use of this group is typically not recommended. Quantum regions extending into **zero field** contacts will cause carrier densities higher than those in metals and Fermi levels in the range of keV. The cause of this is a nonphysical way in which **zero field** contacts are calculated, by enforcing a **Neumann** zero-field condition at the contact.

7.7.25 zero_field{ bias }

Calling sequence

`contacts{ zero_field{ bias } }`

Properties

- **usage:** required
- **type:** vector of up to 100 real numbers: (r_1), (r_1, r_2), ldots, $(r_1, r_2, \ldots, r_{100})$
- **values:** no constraints
- **unit:** V

Functionality

Explicitly defined set of values set to both electron and quasi-Fermi levels as minus `bias` multiplied by the electron charge. Often it corresponds directly to applied voltage.

7.7.26 long_directory_names

Calling sequence

`contacts{ long_directory_names }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

An attribute allowing to use longer names for bias subdirectories, dependent on the number of defined contacts. If `long_directory_names = no` then bias subdirectories are enumerated as `bias_*****` independently of the numbers of contacts defined.

If `long_directory_names = yes`: bias subdirectories are named `bias_000_001_***_...` which could result in issues with too long file paths for inputs with a large number of contacts.

7.7.27 bias_steps

Calling sequence

`contacts{ bias_steps }`

Properties

- **usage:** conditional
- **type:** integer
- **values:** $1 \leq z \leq 999$
- **default:** $z = 1$
- **unit:** $-$

Functionality

Defines the number N of bias steps ($N-1$ intermediate values of voltages applied to all the contacts) between explicitly defined values within the `bias` attribute defined separately for each of the contacts. These values are linear interpolations of the values defined within the `bias` vectors. For instance, having `contacts{ ohmic{ bias = [0, 10] } }` and `contacts{ bias_steps = 5 }` results in a sweep for biasses: 0, 2, 4, 6, 8, and 10 V (6 bias points) for this specific contact. If other contacts have more explicit bias points defined, e.g., `[0, 10, 20]` resulting in 11 bias points: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20. Then the contact with `bias = [0, 10]` will be swept multiple times repeating the last voltage to provide 6+5 biasses as well. Therefore, it will be swept through biasses: 0, 2, 4, 6, 8, 10, 10, 10, 10, 10, and 10 V.

> **Hint**
>
> See file *bias_points.log* to see the actual bias values used. This file contains the mapping between bias values and bias index for all bias points.

7.7.28 reuse_previous

Calling sequence

`contacts{ reuse_previous }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

It turns on the bias ramping when set to `yes`. As a result the electrostatic potential and quasi-Fermi levels obtained within simulations of preceding bias point are used to initialize simulations for the next bias point (set of voltages for all contacts). If this keyword is used in conjunctions with importing electrostatic potential, the potential is imported only to initialize the first iteration of the entire sweep.

7.7.29 bias_output_level

Calling sequence

`contacts{ bias_output_level }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 2$
- **default:** $z = 2$

Functionality

It controls output for the sweep. All regular outputs are always generated for the last bias point of the entire sweep. If set to 0, then only convergence outputs and data entering I-V characteristics are included in the output for non-last bias point. If set to 1, then also all densities, potentials, band edges, and currents are included for non-last bias points. If set to 2, then all regular outputs are generated for all bias points.

Last update: 02/04/2025

7.8 structure{ }

Calling sequence

`structure{ }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

definition of device structure (including doping{ })

Example

```
structure{
    region{...}
}
```

Nested keywords

7.8.1 region{ }

Calling sequence

`structure{ region{ } }`

Properties

- **usage:** required
- **items:** minimum 1

Functionality

Defines regions in the simulation domain and manages assigning materials, contacts (boundary conditions), impurities, fixed generation rates, and fixed injection rates. Each region is automatically indexed in the ascending manner as defined in the input file from top to bottom. Material regions, the regions assigning materials, contain additional indexing related to order of definition of materials in the database file used for the simulation.

Example

```
structure{
    region{...}
}
```

Nested keywords

user_index

Calling sequence

`structure{ region{ user_index } }`

Properties

- **usage:** optional
- **type:** integer
- **unit:** $-$
- **values:** $z \geq 0$

Functionality

Additional arbitrary index assigned to a region.

Example

```
structure{
    region{
        user_index = 1
        ...
    }
}
```

array_x{ }

Calling sequence

`structure{ region{ array_x } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Copies the `region` object along the x-direction.

Example

```
structure{
    region{
        array_x{...}
        ...
    }
}
```

Nested keywords

- *shift*
- *max*

- *min*

---

shift

Calling sequence

`structure{ region{ array_x{ shift } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines a shift distance in the x-direction used for creating the array of regions.

Example

```
structure{
    region{
        array_x{
            shift = 11.0
            ...
        }
        ...
    }
}
```

---

max

Calling sequence

`structure{ region{ array_x{ max } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of regions added in the positive direction of the x-axis.

Example

```
structure{
    region{
        array_x{
            shift = 11.0
            max = 3
        }
        ...
    }
}
```

---

min

Calling sequence

`structure{ region{ array_x{ min } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** $-$
- **default:** $z = 0$

Functionality

Number of regions added in the negative direction of the x-axis.

Example

```
structure{
    region{
        array_x{
            shift = 11.0
            max = 3
            min = 2
        }
        ...
    }
}
```

array_y{ }

Calling sequence

`structure{ region{ array_y } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Copies the `region` object along the y-direction.

Example

```
structure{
    region{
        array_y{...}
        ...
    }
}
```

Nested keywords

- *shift*
- *max*
- *min*

---

shift

Calling sequence

`structure{ region{ array_y{ shift } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines a shift distance in the y-direction used for creating the array of regions.

Example

```
structure{
    region{
        array_y{
            shift = 11.0
            ...
        }
        ...
    }
}
```

max

Calling sequence

`structure{ region{ array_y{ max } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of regions added in the positive direction of the y-axis.

Example

```
structure{
    region{
        array_y{
            shift = 11.0
            max = 3
        }
        ...
    }
}
```

---

min

Calling sequence

`structure{ region{ array_y{ min } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** $-$
- **default:** $z = 0$

Functionality

Number of regions added in the negative direction of the y-axis.

Example

```
structure{
    region{
        array_y{
            shift = 11.0
            max = 3
```

(continues on next page)

(continued from previous page)

```
            min = 2
        }
        ...
    }
}
```

array_z{ }

Calling sequence

`structure{ region{ array_z } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Copies the `region` object along the z-direction.

Example

```
structure{
    region{
        array_z{...}
        ...
    }
}
```

Nested keywords

- *shift*
- *max*
- *min*

---

shift

Calling sequence

`structure{ region{ array_z{ shift } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines a shift distance in the z-direction used for creating the array of regions.

Example

```
structure{
    region{
        array_z{
            shift = 11.0
            ...
        }
        ...
    }
}
```

---

max

Calling sequence

`structure{ region{ array_z{ max } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of regions added in the positive direction of the z-axis. . . . .

Example

```
structure{
    region{
        array_z{
            shift = 11.0
            max = 3
        }
        ...
    }
}
```

---

min

Calling sequence

`structure{ region{ array_z{ min } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** $-$
- **default:** $z = 0$

Functionality

Number of regions added in the negative direction of the z-axis.

Example

```
structure{
    region{
        array_z{
            shift = 11.0
            max = 3
            min = 2
        }
        ...
    }
}
```

array2_x{ }

Calling sequence

`structure{ region{ array2_x } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Copies the region of interest and its copies generated by *array_x{ }* along the x-direction.

Example

```
structure{
    region{
        array2_x{...}
        array_x{...}
        ...
    }
}
```

Nested keywords

- *shift*
- *max*

- *min*

---

shift

Calling sequence

`structure{ region{ array2_x{ shift } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines a shift distance in the x-direction used for creating the second level array of regions.

Example

```
structure{
    region{
        array2_x{
            shift = 11.0
            ...
        }
        array_x{...}
        ...
    }
}
```

---

max

Calling sequence

`structure{ region{ array2_x{ max } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of copies added in the positive direction of the x-axis.

Example

```
structure{
    region{
        array2_x{
            shift = 11.0
            max = 3
        }
        array_x{...}
        ...
    }
}
```

---

min

Calling sequence

`structure{ region{ array2_x{ min } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** $-$
- **default:** $z = 0$

Functionality

Number of copies added in the negative direction of the x-axis.

Example

```
structure{
    region{
        array2_x{
            shift = 11.0
            max = 3
            min = 2
        }
        array_x{...}
        ...
    }
}
```

array2_y{ }

Calling sequence

`structure{ region{ array2_y } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Copies the region of interest and its copies generated by *array_y{ }* along the y-direction.

Example

```
structure{
    region{
        array2_y{...}
        array_y{...}
        ...
    }
}
```

Nested keywords

- *shift*
- *max*
- *min*

---

shift

Calling sequence

`structure{ region{ array2_y{ shift } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines a shift distance in the y-direction used for creating the second level array of regions.

Example

```
structure{
    region{
        array2_y{
            shift = 11.0
            ...
        }
        array_y{...}
        ...
    }
}
```

max

Calling sequence

`structure{ region{ array2_y{ max } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of copies added in the positive direction of the y-axis.

Example

```
structure{
    region{
        array2_y{
            shift = 11.0
            max = 3
        }
        array_y{...}
        ...
    }
}
```

---

min

Calling sequence

`structure{ region{ array2_y{ min } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** $-$
- **default:** $z = 0$

Functionality

Number of copies added in the negative direction of the y-axis.

Example

```
structure{
    region{
        array2_y{
            shift = 11.0
```

(continues on next page)

(continued from previous page)

```
            max = 3
            min = 2
        }
        array_y{...}
        ...
    }
}
```

array2_z{ }

Calling sequence

`structure{ region{ array2_z } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Copies the region of interest and its copies generated by *array_z{ }* along the z-direction.

Example

```
structure{
    region{
        array2_z{...}
        array_z{...}
        ...
    }
}
```

Nested keywords

- *shift*
- *max*
- *min*

---

shift

Calling sequence

`structure{ region{ array2_z{ shift } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines a shift distance in the z-direction used for creating the second level array of regions.

Example

```
structure{
    region{
        array2_z{
            shift = 11.0
            ...
        }
        array_z{...}
        ...
    }
}
```

---

max

Calling sequence

`structure{ region{ array2_z{ max } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of copies added in the positive direction of the z-axis.

Example

```
structure{
    region{
        array2_z{
            shift = 11.0
            max = 3
        }
        array_z{...}
        ...
    }
}
```

---

min

Calling sequence

`structure{ region{ array2_z{ min } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** –
- **default:** $z = 0$

Functionality

Number of copies added in the negative direction of the z-axis.

Example

```
structure{
    region{
        array2_z{
            shift = 11.0
            max = 3
            min = 2
        }
        array_z{...}
        ...
    }
}
```

repeat_profiles

Calling sequence

`structure{ region{ repeat_profiles } }`

Properties

- **usage:** conditional
- **type:** enumerator
- **values:** `alloy`; `doping`; `generation`; `injection`; `other`
- **default:** `all`

Functionality

Specifies which profiles are repeated.

> **Note**
>
> To repeat various profiles independently of each other, one have to define separate regions for each of them.

Examples

```
structure{
    region{
        repeat_profiles = 'doping alloy other'
        array_x{...}
    }
}
```

contact{ }

Calling sequence

`structure{ region{ contact{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Allows assigning/removing boundary conditions for the Poisson equation and drift-diffusion model to/from the region.

Example

```
structure{
    region{
        contact{...}
        ...
    }
}
```

Nested keywords

- *name*
- *remove*

---

name

Calling sequence

`structure{ region{ contact{ name } } }`

Properties

- **usage:** conditional
- **type:** character string

Functionality

Refers to a set of boundary conditions defined within a global group *contacts{ }* and assigns them to the region.

Example

```
structure{
    region{
        contact{
            name = "my_boundary_conditions"
        }
        ...
    }
```

(continues on next page)

(continued from previous page)

```
}
contacts{
    schottky{
        name = "my_boundary_conditions"
        ...
    }
}
```

---

remove

Calling sequence

`structure{ region{ contact{ remove{ } } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Removes previously defined (if defined) boundary conditions for the Poisson equation and drift-diffusion model from the region.

Example

```
structure{
    region{
        contact{
            remove{ }
        }
        ...
    }
}
```

doping{ }

Calling sequence

`structure{ region{ doping{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Assigns dopants defined in the global group *impurities{ }* with selected concentrations.

Examples

```
structure{
    region{
        doping{...}
        ...
    }
}
impurities{
    ...
}
```

Note

See *— FREE — Schrödinger-Poisson - A comparison to the tutorial file of Greg Snider's code* as an example of use of *gaussian1D{ }* and *gaussian2D{ }* for donors and acceptors.

The Figure 7.8.1.1 shows a 3D doping profile that is defined inside a 20 nm x 20 nm x 50 nm cube where the 50 nm are the z direction. The doping rate profile is homogeneous with respect to the (x,y) plane, it only varies along the z direction.

The doping rate profile is constant between z = 10 nm and z = 25 nm with a rate of 1 x 1018 $[1/cm^3]$. It has Gaussian shape from z = 25 nm to z = 45 nm (gaussian1D). It is zero between z = 0 nm and z = 10 nm, as well as between z = 45 nm and z = 50 nm.

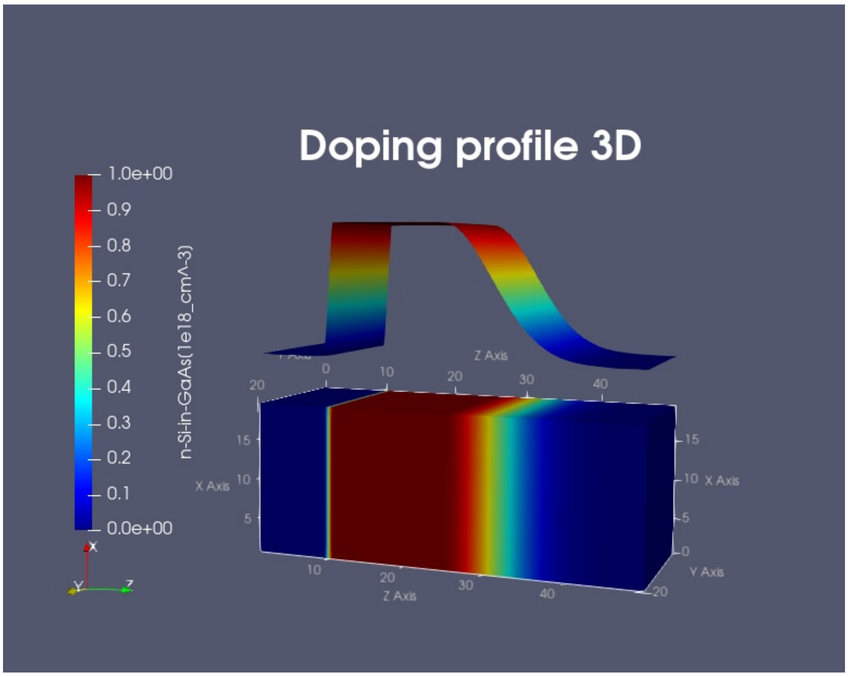


Figure 7.8.1.1: Three-dimensional doping profile (image generated by ParaView).

position along z-direction (nm)	generation rate (1/cm^3)
0 ~ 10 nm	0.0
10 ~ 25 nm	constant (1.0×10^{18})
25 ~ 45 nm	Gaussian (center = 25 nm, sigma_z = 6.0 nm)
45 ~ 50 nm	0.0

```
structure{
    region{
        everywhere{}
        binary{ name = GaAs }
        contact{ name = contact }
    }
    region{
        binary{ name = GaAs }
        cuboid{
            x = [0E0, 20E0]
            y = [0E0, 20E0]
            z = [0E0, 10E0]
        }
    }
    region{
        binary{ name = GaAs }
        cuboid{
            x = [0E0, 20E0]
            y = [0E0, 20E0]
            z = [10E0, 25E0]
        }
        doping{
            constant{
                name = "n-Si-in-GaAs"
                conc = 1.0E18
            }
        }
    }
    region{
        binary{ name = GaAs }
        cuboid{
            x = [0E0, 20E0]
            y = [0E0, 20E0]
            z = [25E0, 45E0]
        }
        doping{
            gaussian1D{
                name = "n-Si-in-GaAs"
                conc = 1.0E18
                z = 25
                sigma_z = 6.0
            }
        }
    }

    output_impurities{}
}
impurities{
    donor{
        name = "n-Si-in-GaAs"
        ...
    }
}
global{
    simulate3D{}
    ...
}
```

Nested keywords

- *remove{ }*
- *constant{ }*
- *constant{ name }*
- *constant{ conc }*
- *constant{ add }*
- *linear{ }*
- *linear{ name }*
- *linear{ conc }*
- *linear{ x }*
- *linear{ y }*
- *linear{ z }*
- *linear{ add }*
- *gaussian1D{ }*
- *gaussian1D{ name }*
- *gaussian1D{ conc }*
- *gaussian1D{ dose }*
- *gaussian1D{ x }*
- *gaussian1D{ y }*
- *gaussian1D{ z }*
- *gaussian1D{ sigma_x }*
- *gaussian1D{ sigma_y }*
- *gaussian1D{ sigma_z }*
- *gaussian1D{ add }*
- *gaussian2D{ }*
- *gaussian2D{ name }*
- *gaussian2D{ conc }*
- *gaussian2D{ dose }*
- *gaussian2D{ x }*
- *gaussian2D{ y }*
- *gaussian2D{ z }*
- *gaussian2D{ sigma_x }*
- *gaussian2D{ sigma_y }*
- *gaussian2D{ sigma_z }*
- *gaussian2D{ add }*
- *gaussian3D{ }*
- *gaussian3D{ name }*

- *gaussian3D{ conc }*
- *gaussian3D{ dose }*
- *gaussian3D{ x }*
- *gaussian3D{ y }*
- *gaussian3D{ z }*
- *gaussian3D{ sigma_x }*
- *gaussian3D{ sigma_y }*
- *gaussian3D{ sigma_z }*
- *gaussian3D{ add }*
- *import{ }*
- *import{ name }*
- *import{ import_from }*

---

remove{ }

Calling sequence

`structure{ region{ doping{ remove{ } } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Removes all dopants from a specific region.

Example

```
structure{
    region{
        doping{
            remove{}
        }
        ...
    }
}
impurities{
    ...
}
```

---

constant{ }

Calling sequence

`structure{ region{ doping{ constant{ } } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines constant doping profile over the region.

Example

```
structure{
    region{
        doping{
            constant{
                name = "n-Si"
                conc = 1.0e18
                add  = no
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
```

---

constant{ name }

Calling sequence

`structure{ region{ doping{ constant{ name } } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            constant{
                name = "n-Si"
                ...
            }
        }
```

(continues on next page)

(continued from previous page)

```
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

constant{ conc }

Calling sequence

`structure{ region{ doping{ constant{ conc } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** cm^{-3}

Functionality

Defines value dopant concentration.

Example

```
structure{
    region{
        doping{
            constant{
                conc = 1.0e18
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

constant{ add }

Calling sequence

`structure{ region{ doping{ constant{ add } } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Choses the mode of assigning doping. If `add = yes` then the doping in the region is added to already defined ones. Otherwise, the previously defined doping is replaced.

Example

```
structure{
    region{
        doping{
            constant{
                add  = no
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

linear{ }

Calling sequence

`structure{ region{ doping{ linear{ } } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines linear doping profile along a defined line

Examples

```
structure{
    region{
        doping{
            linear{
                name = "n-Si"
                conc = [ 1.0e18, 2.0e18 ]
                x    = [ 50.0, 100.0 ]
                add  = no
            }
```

(continues on next page)

(continued from previous page)

```
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
global{
    simulate1D{ }
}
```

```
structure{
    region{
        doping{
            linear{
                name = "n-Si"
                conc = [ 1.0e18, 2.0e18 ]
                x    = [ 50.0, 100.0 ]
                y    = [ 50.0, 100.0 ]
                z    = [ 50.0, 100.0 ]
                add  = no
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
global{
    simulate3D{ }
}
```

---

linear{ name }

Calling sequence

`structure{ region{ doping{ linear{ name } } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            linear{
                name = "n-Si"
                ...
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

linear{ conc }

Calling sequence

`structure{ region{ doping{ linear{ conc } } } }`

Properties

- **usage:** required
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** `[0.0, ...)` for every dimension
- **unit:** cm^{-3}

Functionality

Defines values of linear dopant profile at the ends of the ends of defined line. The first value corresponds to the starting point of the line and the second value to the ending point of the line.

Example

```
structure{
    region{
        doping{
            linear{
                conc = [ 1.0e18, 2.0e18 ]
                ...
            }
        }
        ...
    }
}
```

(continues on next page)

(continued from previous page)

```
impurities{
    ...
}
```

---

linear{ x }

Calling sequence

`structure{ region{ doping{ linear{ x } } } }`

Properties

- **usage:** conditional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **unit:** nm

Functionality

Defines x-coordinates of the starting point and ending point of the line, along which the linear distribution of dopants is defined.

Example

```
structure{
    region{
        doping{
            linear{
                x = [ 50.0, 100.0 ]
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

linear{ y }

Calling sequence

`structure{ region{ doping{ linear{ y } } } }`

Properties

- **usage:** conditional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **unit:** nm

Functionality

Defines y-coordinates of the starting point and ending point of the line, along which the linear distribution of dopants is defined.

Example

```
structure{
    region{
        doping{
            linear{
                x = [ 50.0, 100.0 ]
                y = [ 50.0, 100.0 ]
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

linear{ z }

Calling sequence

`structure{ region{ doping{ linear{ z } } } }`

Properties

- **usage:** conditional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **unit:** nm

Functionality

Defines z-coordinates of the starting point and ending point of the line, along which the linear distribution of dopants is defined.

Example

```
structure{
    region{
        doping{
            linear{
                x = [ 50.0, 100.0 ]
                y = [ 50.0, 100.0 ]
                z = [ 50.0, 100.0 ]
                ...
            }
        }
        ...
    }
```

(continues on next page)

(continued from previous page)

```
}
impurities{
    ...
}
```

---

linear{ add }

Calling sequence

`structure{ region{ doping{ linear{ add } } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Choses the mode of assigning doping. If `add = yes` then the doping in the region is added to already defined ones. Otherwise, the previously defined doping is replaced.

Example

```
structure{
    region{
        doping{
            linear{
                add = no
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ }

Calling sequence

`structure{ region{ doping{ gaussian1D{ } } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines Gaussian distribution function in one direction, constant in remaining perpendicular directions.

> **Note**
>
> This profile corresponds to LSS theory (Lindhard, Scharff, Schiott theory) for doping - Gaussian distribution of ion implantation.

Examples

```
structure{
    region{
        doping{
            gaussian1D{
                name = "n-Si"
                conc = 1.0e18
                x = 50.0
                sigma_x = 5.0
                add = no
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
global{
    simulate1D{ }
}
```

```
structure{
    region{
        doping{
            gaussian1D{
                name = "n-Si"
                dose = 1e12
                y = 50.0
                sigma_y = 5.0
                add = no
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
```

(continues on next page)

(continued from previous page)

```
global{
    simulate2D{ }
}
```

---

gaussian1D{ name }

Calling sequence

`structure{ region{ doping{ gaussian1D{ name } } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                name = "n-Si"
                ...
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

gaussian1D{ conc }

Calling sequence

`structure{ region{ doping{ gaussian1D{ conc } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** cm^{-3}

Functionality

Defines maximum of dopant concentration.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                conc = 1.0e18
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ dose }

Calling sequence

`structure{ region{ doping{ gaussian1D{ dose } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** cm^{-2}

Functionality

Defines implantation dose. It is an integrated density of Gaussian function conc = dose / (SQRT(2*pi) * sigma_x).

Hint

Typical values range from 1e11 cm^{-2} to 1e16 cm^{-2}.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                dose = 1.0e12
                ...
            }
        }
        ...
```

(continues on next page)

(continued from previous page)

```
    }
}
impurities{
    ...
}
```

---

gaussian1D{ x }

Calling sequence

`structure{ region{ doping{ gaussian1D{ x } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the x-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                x = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ y }

Calling sequence

`structure{ region{ doping{ gaussian1D{ y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the y-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                y = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ z }

Calling sequence

`structure{ region{ doping{ gaussian1D{ z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the z-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                z = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ sigma_x }

Calling sequence

`structure{ region{ doping{ gaussian1D{ sigma_x } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the x-axis.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                x = 50.0
                sigma_x = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ sigma_y }

Calling sequence

`structure{ region{ doping{ gaussian1D{ sigma_y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the y-axis.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                y = 50.0
                sigma_y = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ sigma_z }

Calling sequence

`structure{ region{ doping{ gaussian1D{ sigma_z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the z-axis.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                z = 50.0
                sigma_z = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ add }

Calling sequence

`structure{ region{ doping{ gaussian1D{ add } } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Choses the mode of assigning doping. If `add = yes` then the doping in the region is added to already defined ones. Otherwise, the previously defined doping is replaced.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                add = no
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ }

Calling sequence

`structure{ region{ doping{ gaussian2D{ } } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines Gaussian distribution function in two directions, constant in remaining perpendicular direction.

Examples

```
structure{
    region{
        doping{
            gaussian2D{
```

(continues on next page)

(continued from previous page)

```
                name = "n-Si"
                conc = 1.0e18
                x = 50.0
                y = 50.0
                sigma_x = 5.0
                sigma_y = 5.0
                add = no
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
global{
    simulate2D{ }
}
```

```
structure{
    region{
        doping{
            gaussian2D{
                name = "n-Si"
                dose = 1e6
                x = 50.0
                z = 50.0
                sigma_x = 5.0
                sigma_z = 5.0
                add = no
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
global{
    simulate3D{ }
}
```

gaussian2D{ name }

Calling sequence

`structure{ region{ doping{ gaussian2D{ name } } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                name = "n-Si"
                ...
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

gaussian2D{ conc }

Calling sequence

`structure{ region{ doping{ gaussian2D{ conc } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** cm^{-3}

Functionality

Defines maximum of dopant concentration.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                conc = 1.0e18
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ dose }

Calling sequence

`structure{ region{ doping{ gaussian2D{ dose } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** cm^{-1}

Functionality

Defines implantation dose. It is an integrated density of Gaussian function.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                dose = 1.0e6
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ x }

Calling sequence

`structure{ region{ doping{ gaussian2D{ x } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the x-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                x = 50.0
                y = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ y }

Calling sequence

`structure{ region{ doping{ gaussian2D{ y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the y-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                y = 50.0
                z = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ z }

Calling sequence

`structure{ region{ doping{ gaussian2D{ z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the z-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                z = 50.0
                x = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ sigma_x }

Calling sequence

`structure{ region{ doping{ gaussian2D{ sigma_x } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the x-axis.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                x = 50.0
                y = 50.0
                sigma_x = 5.0
                sigma_y = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ sigma_y }

Calling sequence

`structure{ region{ doping{ gaussian2D{ sigma_y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the y-axis.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                y = 50.0
                z = 50.0
                sigma_y = 5.0
                sigma_z = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ sigma_z }

Calling sequence

`structure{ region{ doping{ gaussian2D{ sigma_z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the z-axis.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                z = 50.0
                x = 50.0
                sigma_z = 5.0
                sigma_x = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ add }

Calling sequence

`structure{ region{ doping{ gaussian2D{ add } } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Choses the mode of assigning doping. If `add = yes` then the doping in the region is added to already defined ones. Otherwise, the previously defined doping is replaced.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                add = no
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ }

Calling sequence

`structure{ region{ doping{ gaussian3D{ } } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Defines Gaussian distribution function in three directions.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                name = "n-Si"
                conc = 1.0e18
                x = 50.0
                y = 50.0
                z = 50.0
                sigma_x = 5.0
                sigma_y = 5.0
                sigma_z = 5.0
                add = no
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        degeneracy = 2
        energy = 0.0058
    }
}
global{
    simulate3D{ }
}
```

---

gaussian3D{ name }

Calling sequence

`structure{ region{ doping{ gaussian3D{ name } } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                name = "n-Si"
                ...
            }
        }
```

(continues on next page)

(continued from previous page)

```
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

gaussian3D{ conc }

Calling sequence

`structure{ region{ doping{ gaussian3D{ conc } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** cm^{-3}

Functionality

Defines maximum of dopant concentration.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                conc = 1.0e18
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ dose }

Calling sequence

`structure{ region{ doping{ gaussian3D{ dose } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** –

Functionality

Defines implantation dose. It is an integrated density of Gaussian function.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                dose = 1.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ x }

Calling sequence

`structure{ region{ doping{ gaussian3D{ x } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the x-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                x = 50.0
                y = 50.0
                z = 50.0
```

(continues on next page)

(continued from previous page)

```
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ y }

Calling sequence

`structure{ region{ doping{ gaussian3D{ y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the y-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                x = 50.0
                y = 50.0
                z = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ z }

Calling sequence

`structure{ region{ doping{ gaussian3D{ z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Defines the z-coordinate of the center of the Gauss distribution.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                x = 50.0
                y = 50.0
                z = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ sigma_x }

Calling sequence

`structure{ region{ doping{ gaussian3D{ sigma_x } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the x-axis.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                x = 50.0
```

(continues on next page)

(continued from previous page)

```
                y = 50.0
                z = 50.0
                sigma_x = 5.0
                sigma_y = 5.0
                sigma_z = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ sigma_y }

Calling sequence

`structure{ region{ doping{ gaussian3D{ sigma_y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the y-axis.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                x = 50.0
                y = 50.0
                z = 50.0
                sigma_x = 5.0
                sigma_y = 5.0
                sigma_z = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ sigma_z }

Calling sequence

`structure{ region{ doping{ gaussian3D{ sigma_z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** nm

Functionality

Defines standard deviation of the Gauss distribution along the z-axis.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                x = 50.0
                y = 50.0
                z = 50.0
                sigma_x = 5.0
                sigma_y = 5.0
                sigma_z = 5.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ add }

Calling sequence

`structure{ region{ doping{ gaussian3D{ add } } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Choses the mode of assigning doping. If `add = yes` then the doping in the region is added to already defined ones. Otherwise, the previously defined doping is replaced.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                add = no
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

import{ }

Calling sequence

`structure{ region{ doping{ import{ } } } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Imports generation profile from an external file and adds it to already defined (if defined) profiles.

Example

```
structure{
    region{
        doping{
            import{
                name = "n-Si"
                import_from = "importing_dopant_profile"
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

(continues on next page)

(continued from previous page)

```
import{
    file{
        name = "importing_dopant_profile"
        filename = "precious_dopant_profile.dat"
        ...
    }
}
```

---

import{ name }

Calling sequence

`structure{ region{ doping{ import{ name } } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            import{
                name = "n-Si"
                ...
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
import{
    ...
}
```

---

import{ import_from }

Calling sequence

`structure{ region{ doping{ import{ import_from } } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Reference to imported data in *import{ }*.

Example

```
structure{
    region{
        doping{
            import{
                import_from = "importing_dopant_profile"
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
import{
    file{
        name = "importing_dopant_profile"
        filename = "precious_dopant_profile.dat"
        ...
    }
}
```

7.8.2 output_region_index{ }

Calling sequence

`structure{ output_region_index{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs last index of the regions and material region assigned to each grid point. Each region has associated number ordered from top to bottom as written in the input file. In the case of overlapping regions, the number of the last defined region is taken into account. Material region is a region which specifies a material.

Examples

```
structure{
    output_region_index{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`structure{ output_region_index{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_region_index{
        boxes = yes
    }
}
```

7.8.3 output_material_index{ }

Calling sequence

`structure{ output_material_index{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs material index at each grid point. Each material has associated index assigned based on the order of materials defined in the database file used for the simulation. The material on top of the file has the index 1 assigned.

Examples

```
structure{
    output_material_index{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`structure{ output_material_index{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_material_index{
        boxes = yes
    }
}
```

7.8.4 output_user_index{ }

Calling sequence

`structure{ output_user_index{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs last user-defined index for each grid point.

Examples

```
structure{
    output_user_index{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`structure{ output_user_index{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_user_index{
        boxes = yes
    }
}
```

7.8.5 output_contact_index{ }

Calling sequence

`structure{ output_contact_index{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs contact index for each grid point.

Examples

```
structure{
    output_contact_index{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`structure{ output_contact_index{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_contact_index{
        boxes = yes
    }
}
```

7.8.6 output_alloy_composition{ }

Calling sequence

`structure{ output_alloy_composition{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs alloy composition for each grid point

Examples

```
structure{
    output_alloy_composition{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`structure{ output_alloy_composition{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_alloy_composition{
        boxes = yes
    }
}
```

7.8.7 output_impurities{ }

Calling sequence

`structure{ output_impurities{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs doping concentration for each grid point in units of $[10^{18}/cm^3]$

Examples

```
structure{
    output_impurities{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`structure{ output_impurities{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_impurities{
        boxes = yes
    }
}
```

7.8.8 output_generation{ }

Calling sequence

`structure{ output_generation{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs generation rate for each grid point in units of $[10^{18}/(cm^3s)]$.

Examples

```
structure{
    output_generation{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`structure{ output_generation{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_generation{
        boxes = yes
    }
}
```

7.8.9 output_injection{ }

Calling sequence

`structure{ output_injection{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs injection rate for each grid point in units of $[10^{18}/(cm^3 s)]$.

Examples

```
structure{
    output_injection{ }
}
```

Nested keywords

---

boxes

Calling sequence

`structure{ output_injection{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
structure{
    output_injection{
        boxes = yes
    }
}
```

7.8.10 structure{ region{} } - generation & electron injection

Specifications that define information on generation and injection rates.

injection{}

Injection refers here to explicit electron injection e.g. by electron beam (no holes for now). It used the same keywords as generation. Similarly to generation, this only has an effect when the current equations are solved.

Attention

The `injection{}` group can be used in exactly the same way as the `generation{}` group.

Specifications of generation rate profile

The generation rate profile is assiged to a certain region. The following syntaxes are put under `structure{ region{ generation{} } }`.

- `constant`
- `linear`
- `gaussian1D`
- `gaussian2D`
- `gaussian3D`
- `import` (import generation rate profile from external file)

constant

constant generation rate over the region

Example

```
constant{
    rate = 1.0e18               # generation rate [1/cm³s] (applies␣
↪to 1D, 2D and 3D)
    add  = yes                  # (optional) yes or no (default =␣
↪yes)
}
```

linear

linearly varying generation rate along the line from start to end point

Example

```
linear{
    rate = [1e18,2e18]          # start and end value of generation␣
↪rate [1/cm3s]
    x     = [50.0,100.0]        # x coordinates of start and end␣
↪point [nm]
    y     = [50.0,100.0]        # y coordinates of start and end␣
↪point [nm] (2D or 3D only)
    z     = [50.0,100.0]        # z coordinates of start and end␣
↪point [nm] (3D only)
                                # This defines a generation rate␣
↪profile, which varies linearly along the line from the point (50,
↪50,50) to the point (100,100,100)

↪# and stays constant in the perpendicular planes.
    add  = yes                  # (optional) yes or no (default =␣
↪yes)
}
```

gaussian1D

Gaussian distribution function in one direction, constant in perpendicular directions

Example

```
gaussian1D{                     # Gaussian distribution function in␣
↪one direction, constant in perpendicular directions
    rate     = 1.0e18           # maximum of generation rate [1/cm3s]
    dose     = 1e12             # dose of implant [cm-2] (integrated␣
↪density of Gaussian function), typical ranges are from 1e11 to␣
↪1e16.
```

(continues on next page)

(continued from previous page)

```
                                # Either rate or dose has to be␣
↪specified, but not both simultaneously.
                                # rate = dose / ( SQRT(2*pi) * sigma_
↪x )
    x       = 50.0              # x coordinate of Gauss center (ion
↪'s projected range Rp, i.e. the depth where most ions stop) [nm]
    sigma_x = 5.0               # standard deviation in x direction␣
↪(statistical fluctuation of Rp) [nm]
    y       = ...               # (2D or 3D only)
    sigma_y = ...               #
    z       = ...               # (3D only)
    sigma_z = ...               #
                                # Only one out of x, y, z and the␣
↪appropriate standard deviation (sigma) has to be specified.
    add   = yes                 # (optional) yes or no (default =␣
↪yes)
}
```

Note

This profile corresponds to LSS theory (Lindhard, Scharff, Schiott theory) for doping - Gaussian distribution of ion implantation.

gaussian2D

Gaussian distribution function in two directions, constant in perpendicular direction (2D or 3D only)

Example

```
gaussian2D{                     # Gaussian distribution function in␣
↪two directions, constant in perpendicular direction (2D or 3D only)
    rate    = 1.0e18            # maximum of generation rate [1/cm3s]
    dose    = 1.0               # dose of implant [cm-1] (integrated␣
↪density of 2D Gaussian function)
                                # Either rate or dose has to be␣
↪specified, but not both simultaneously.
    x       = 50.0              # x coordinate of Gauss center [nm]
    sigma_x = 5.0               # standard deviation in x direction␣
↪[nm]
    y       = 50.0              # y coordinate of Gauss center [nm]
    sigma_y = 5.0               # standard deviation in y direction␣
↪[nm]
    z       = ...               # (3D only)
    sigma_z = ...               #
                                # Exactly two out of x, y, z and the␣
↪appropriate standard deviations (sigma) have to be specified.
    add   = yes                 # (optional) yes or no (default =␣
↪yes)
}
```

gaussian3D

Gaussian distribution function in three directions (3D only)

Example

```
gaussian3D{                     # Gaussian distribution function in␣
```

(continues on next page)

(continued from previous page)

```
↪three directions (3D only)
    rate     = 1.0e18            # maximum of generation rate in [1/
↪cm3s]
    dose     = 1.0               # dose of implant [dimensionless]␣
↪(integrated density of 3D Gaussian function)
    x        = 50.0              # x coordinate of Gauss center [nm]
    sigma_x = 5.0                # standard deviation in x direction␣
↪[nm]
    y        = 50.0              # y coordinate of Gauss center [nm]
    sigma_y = 5.0                # standard deviation in y direction␣
↪[nm]
    z        = 50.0              # z coordinate of Gauss center [nm]
    sigma_z = 5.0                # standard deviation in z direction␣
↪[nm]
                                 # All three x, y, z and the␣
↪appropriate standard deviations (sigma) have to be specified.
    add   = yes                  # (optional) yes or no (default =␣
↪yes)
}
```

import

import generation profile from external file

```
import{                                             # import␣
↪generation profile from external file.
    import_from = "import_generation_profile"      # reference to␣
↪imported data in import{ }. The file being imported must have␣
↪exactly one data component.
}
```

Print out

These generation rate profile can be printed out by `output_generation{}` under `structure{ }`:

output_generation{}

```
structure{
    output_generation{                # output generation rate for each␣
↪grid point in units of [10^18/(cm3 s)]
        boxes = yes/no                # (optional)
    }
}
```

Remove

It is also possible to remove a generation rate from a specific region.

remove{}

```
structure{
    region{
        generation{ remove{} }              # remove generation rate␣
↪from this region, to keep certain regions free from generation rate.
    }     # region
}       # structure
```

Note

`doping{}` and `generation{}` is always additive per default (`add = yes`) (unless import is different), i.e. each profile adds to the already existing dopants/fixed charges/generation at a given point. At the same time, using `remove{}`, all species of the already existing doping or generation concentrations can be removed. However, there is also the problem that `remove{}` removes all species of dopants/fixed charges at a given point. Thus, removing e.g. only donors but not acceptors is difficult. This problem is solved by the new "`add = yes/no`" flag, which the user can specify for each profile (and thus for the species of that profile), whether the profile should add to (which is the default) or replace the already existing concentration of the profile species.

For `import{ }`, this flag has not been implemented yet.

Example

3D

Figure 7.8.10.1 shows a 3D generation profile that is defined inside a 20 nm x 20 nm x 50 nm cube where the 50 nm are the z direction. The generation rate profile is homogeneous with respect to the (x,y) plane, it only varies along the z direction.

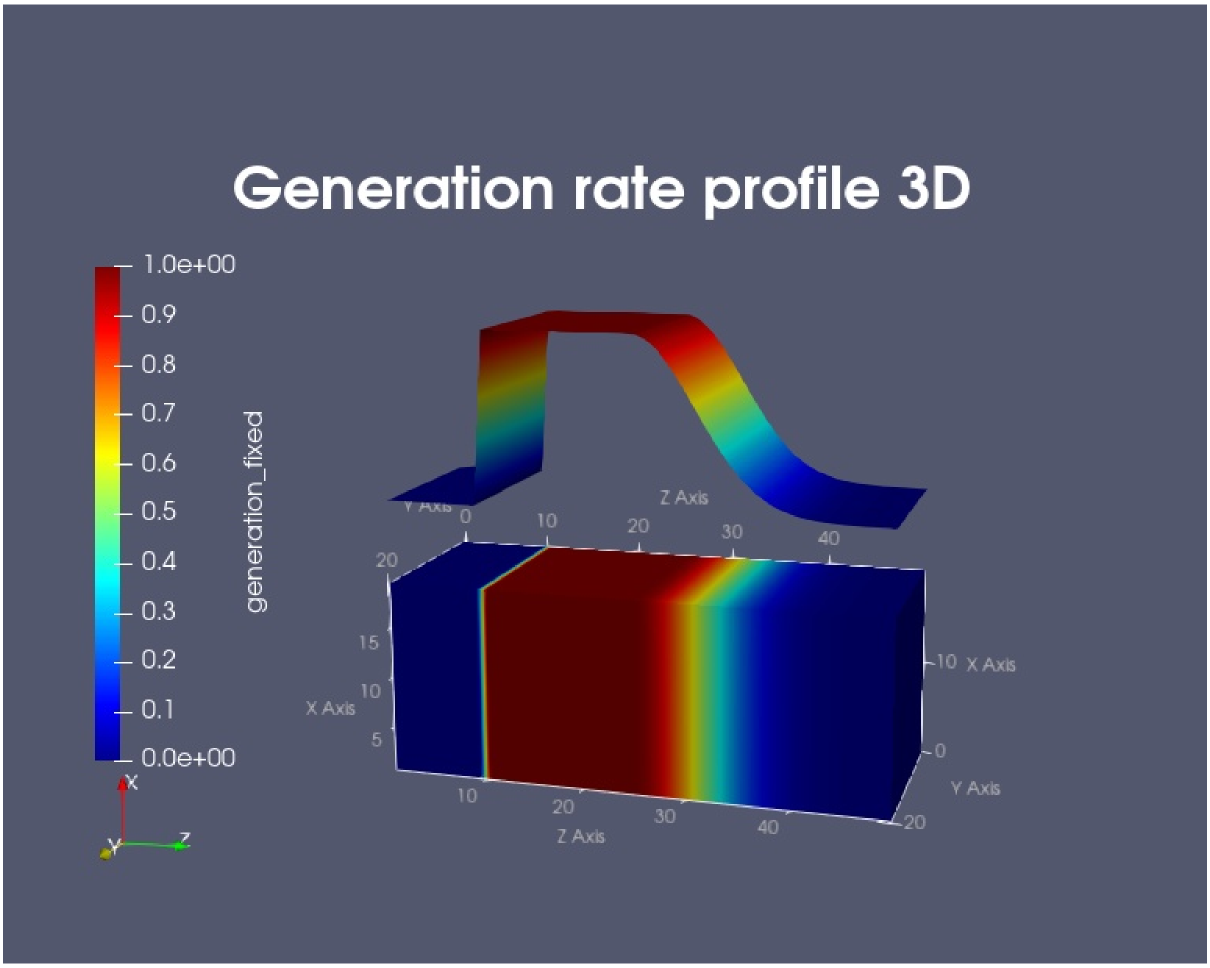


Figure 7.8.10.1: Three-dimensional generation rate profile. (Image generated by ParaView.)

The generation rate profile is constant between z = 10 nm and z = 25 nm with a rate of 1 x 1018 $[1/(cm^3 s)]$. It has Gaussian shape from z = 25 nm to z = 45 nm (gaussian1D). It is zero between z = 0 nm and z = 10 nm, as well as between z = 45 nm and z = 50 nm.

	z = 0 ~ 10 nm	z = 10 ~ 25 nm	z = 25 ~ 45 nm	z = 45 ~ 50 nm
generation rate $[1/(cm^3 s)]$	0.0	constant (1.0 × 1018)	Gaussian (center = 25 nm, σ_z = 6.0 nm)	0.0

Here is the `structure` part of the input file that generates the above generation profile.

```
structure{
    output_generation{}             # output generation rate for each grid point in␣
↪units of [10^18/(cm3 s)]

    region{                         # default material
        everywhere{}
        binary{ name = GaAs }
        contact{ name = contact }
    }
    region{
        binary{ name = GaAs }
        cuboid{
            x = [0E0, 20E0]
            y = [0E0, 20E0]
            z = [0E0, 10E0]
        }
    }
    region{
        binary{ name = GaAs }
        cuboid{
            x = [0E0, 20E0]
            y = [0E0, 20E0]
            z = [10E0, 25E0]
        }
        generation{
            constant{
                rate = 1.0E18       # generation rate [1/cm3s] (applies to 1D, 2D and␣
↪3D)
            }
        }
    }
    region{
        binary{ name = GaAs }
        cuboid{
            x = [0E0, 20E0]
            y = [0E0, 20E0]
            z = [25E0, 45E0]
        }
        generation{
            gaussian1D{
                rate = 1.0E18       # maximum of generation rate [1/cm3s]
                z = 25              # z coordinate of Gauss center (ion's projected␣
↪range Rp, i.e. the depth where most ions stop) [nm]
                sigma_z = 6.0       # root mean square deviation in z direction␣
↪(statistical fluctuation of Rp) [nm]
            }
        }
    }
}<>
```

7.8.11 structure{ region{ integrate{ } } }

integrate{}

spatial integration of profiles in the region.

Example

```
integrate{                                   # spatial integration of profiles in this␣
↪region.
        electron_density{}                   # integrate electron density.
        hole_density{}                       # integrate hole density.
        ionized_donor_density{}              # integrates density of ionized donors
        ionized_acceptor_density{}           # integrates density of ionized acceptors
        piezo_density{}                      # integrate piezo charge density.
        pyro_density{}                       # integrate pyro charge density.
        polarization_density{}               # integrate the polarization charges␣
↪density. ( = piezo + pyro)
        fixed_charge_density{}               # integrates density of fixed charges.
        label = "channel"                    # (optional) defines meaningful label for␣
↪columns in output files.
                                             # If not defined, the number of the␣
↪region is taken as a label.
}
```

Note

Due to the finite descretization of the space, it is advised to define the region for integration slightly larger than the region of actual interest, especially if there is a significantly high density at the boundaries of the integration region.

7.8.12 structure{ region{} } - assigning materials

Binary, ternary and quaternary materials are possible, with several choices of alloy functions. Depending on the dimension of the simulation domain, different options are available.

binary{}

binary material

Example

```
binary{
        name    = "GaAs"             # binary material name for this region
}
```

ternary_constant{}

ternary material with constant alloy profile

Example

```
ternary_constant{
        name    = "Al(x)Ga(1-x)As"   # ternary material name for this region with␣
↪constant alloy profile
        alloy_x = 0.2                # x content of the alloy (minimum value is 0.0,␣
↪maximum value is 1.0)
}
```

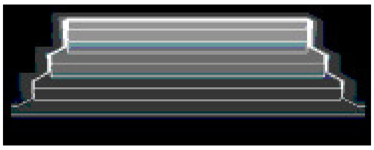
ternary_linear{}

ternary material name which varies linearly along the line from start to end point

Example

```
ternary_linear{
        name    = "In(x)Al(1-x)As"  # ternary material name for this region with␣
↪linear alloy profile
        alloy_x = [0.8, 0.2]        # start and end value of x content (minimum value␣
↪is 0.0, maximum value is 1.0)
        x       = [75.0, 125.0]     # x coordinates of start and end point [nm]
        y       = [10.0, 20.0]      # y coordinates of start and end point [nm] (2D␣
↪or 3D only)
        z       = [10.0, 20.0]      # z coordinates of start and end point [nm] (3D␣
↪only)
                                    # This defines an alloy profile, which varies␣
↪linearly along the line from the point (75,10,10) to the point (125,20,20)
                                    # and stays constant in the perpendicular planes.
}
```

(3D quantum dot)



ternary_pyramid{}

ternary material name with pyramidal alloy profile

Example

```
ternary_pyramid{                    # (e.g. for InGaAs quantum dots) starting point␣
↪and direction (3D only)
        name    = "In(x)Ga(1-x)As"  # ternary material name for this region with␣
↪pyramidal alloy profile
        alloy_x = [0.28, 0.80]      # c_{min} and c_{max} value of x content (minimum␣
↪value is 0.0, maximum value is 1.0)
                                    # vary alloy concentration from apex/axis x = 0.
↪80 (In0.80Ga0.20As)
                                    # to plane through apex perpendicular to axis x =␣
↪0.28 (In0.28Ga0.72As) (see figure below)
        x       = [20.0, 0]         # x coordinate of apex and x component of axis␣
↪direction [nm]
        y       = [20.0, 0]         # y coordinate of apex and y component of axis␣
↪direction [nm]
        z       = [11.0, 1]         # z coordinate of apex and z component of axis␣
↪direction [nm]
                                    # apex located at point (20.0,20.0,11.0) (top of␣
↪inverted pyramid)
                                    # direction of center axis (0,0,1), i.e. along z␣
↪axis
                                    # The profile is symmetric with respect to the␣
↪inverse of the direction of the center axis,
                                    # i.e. (0,0,1) will lead to the same pyramidal␣
↪profile as (0,0,-1).
}
```

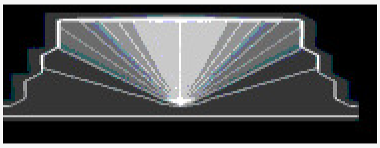
Note

The indium content is given by the following formula, which considers an additional lateral variation of the indium content:

$$c = c_{min} + (c_{max} - c_{min}) \cos^2 \phi$$

where ϕ is the angle to the center axis. The formula is based on the model proposed by Tersoff (N. Liu et al., PRL 84, 334 (2000)). For simplicity the alloy profile is still isotropic around the center axis of the quantum dot. The indium content depends solely on the angle to the center axis, with high indium content for small angles as indicated by the light regions in the figure shown below.

(3D quantum dot)



ternary_trumpet{}

ternary material with "trumpet" alloy profile

Example

```
ternary_trumpet{                    # (e.g. for InGaAs quantum dots) starting point␣
↪and direction (3D only)
        name    = "In(x)Ga(1-x)As"  # ternary material name for this region with
↪"trumpet" alloy profile
        alloy_x = [0.2, 0.5]        # :math:`c_{min}` and :math:`c_{max}` value of x␣
↪content (minimum value is 0.0, maximum value is 1.0)
        x       = [20.0, 0]         # x coordinate of apex and x component of axis␣
↪direction [nm]
        y       = [20.0, 0]         # y coordinate of apex and y component of axis␣
↪direction [nm]
        z       = [11.0, 1]         # z coordinate of apex and z component of axis␣
↪direction [nm]
                                    # apex located at point (20.0,20.0,11.0) (top of␣
↪inverted pyramid)
                                    # direction of center axis (0,0,1), i.e. along z␣
↪axis
                                    # The profile is symmetric with respect to the␣
↪inverse of the direction of the center axis,
                                    # i.e. (0,0,1) will lead to the same trumpet␣
↪profile as (0,0,-1).
        z0      = 1.25              # parameter to vary the shape of the alloy␣
↪profile (minimum value is 1e-10)
        rho0    = 0.6               # parameter to vary the shape of the alloy␣
↪profile (minimum value is 1e-10)
}
```

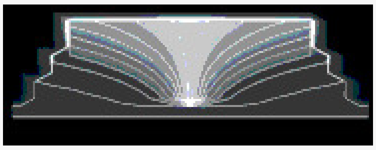
Note

The indium content is given by the formula:

$$c = c_{min} + (c_{max} - c_{min}) \exp\left[(-\sqrt{x^2 + y^2} \exp(-z_1/z_0))/\rho_0\right]$$

The formula is based on the more refined model proposed by Migliorato (M.A. Migliorato et al., PRB 65, 115316 (2002)). This profile resembles the horn of a trumpet and is thus called 'trumpet'. The maximum indium concentration is on the center axis of the quantum dot. The parameters z0 and rho0 can be used to vary the shape of the alloy profile while keeping the average indium content fixed.

(3D quantum dot)



ternary_import{ }

ternary material which uses **imported** alloy profile

Example

```
ternary_import{
        name        = "In(x)Al(1-x)As"          # ternary material name for this␣
↪region which uses imported alloy profile
        import_from = "import_alloy_profile1D"  # reference to imported data in␣
↪``import{ }``. The imported profile must have exactly one data component (x).
}
```

quaternary_import{ }

quaternary material which uses **imported** alloy profile

Example

```
quaternary_import{
        name        = "Al(x)Ga(y)In(1-x-y)As"   # quaternary material name for this␣
↪region which uses imported alloy profile
        import_from = "import_alloy_profile1D"  # reference to imported data in import
↪{ }. The imported profile must have exactly two data components (x,y).
}
```

quinternary_import{ }

quinternary material which uses **imported** alloy profile

Example

```
quinternary_import{
        ...                                    # analogous for quaternaries:
}
```

quaternary_constant{}

quaternary material with constant alloy profile

Example

```
quaternary_constant{
        name        = "Al(x)Ga(y)In(1-x-y)As" # quaternary material name for this␣
↪region with constant alloy profile
        alloy_x     = 0.2                      # x content of the alloy (minimum value␣
↪is 0.0, maximum value is 1.0)
        alloy_y     = 0.5                      # y content of the alloy (minimum value␣
```

(continues on next page)

(continued from previous page)

```
↪is 0.0, maximum value is 1.0)
}
```

Note

For quaternaries of type AxByC1-x-yH, $x + y \leq 1$ must hold.

The interpolation of AxByC1-x-yH is done according to eq. (E.10) in PhD thesis of T. Zibold apart from changes in sign of bowing parameters. The interpolation of AxB1-xCyD1-y is done according to eq. (E.15) in PhD thesis of T. Zibold apart from changes in sign of bowing parameters.

quaternary_linear{}

quaternary material with linear alloy profile

Example

```
quaternary_linear{
        name    = "Al(x)Ga(y)In(1-x-y)As" # quaternary material name for this region␣
↪with linear alloy profile
        alloy_x = [0.2, 0.5]            # start and end value of x content (minimum␣
↪value is 0.0, maximum value is 1.0)
        alloy_y = [0.1, 0.3]            # start and end value of y content (minimum␣
↪value is 0.0, maximum value is 1.0)
        x       = [20.0, 20.0]          # x coordinates of start and end point [nm]
        y       = [20.0, 20.0]          # y coordinates of start and end point [nm]␣
↪(2D or 3D only)
        z       = [11.0, 20.0]          # z coordinates of start and end point [nm]␣
↪(3D only)
}
```

quaternary_pyramid{}

quaternary material with pyramid alloy profile

Example

```
quaternary_pyramid{                     # (e.g. for InGaAs quantum dots) (3D only)
name    = "Al(x)Ga(y)In(1-x-y)As"       # quaternary material name for this region␣
↪with pyramidal alloy profile
alloy_x = [0.2, 0.5]                    # minimum and maximum value of x content
alloy_y = [0.1, 0.3]                    # minimum and maximum value of y content
x       = [20.0, 0]                     # x coordinate of apex and x component of␣
↪axis direction [nm]
y       = [20.0, 0]                     # y coordinate of apex and y component of␣
↪axis direction [nm]
z       = [11.0, 1]                     # z coordinate of apex and z component of␣
↪axis direction [nm]
                                        # apex located at point (20.0,20.0,11.0) (top␣
↪of inverted pyramid)
                                        # direction of center axis (0,0,1), i.e.␣
↪along z axis
                                        # The profile is symmetric with respect to␣
↪the inverse of the direction of the center axis,
                                        # i.e. (0,0,1) will lead to the same␣
↪pyramidal profile as (0,0,-1).
}
```

quaternary_trumpet{}

quaternary material with “trumpet” alloy profile

Example

```
quaternary_trumpet{                       # (e.g. for InGaAs quantum dots) (3D only)
        name    = "Al(x)Ga(y)In(1-x-y)As" # quaternary material name for this region␣
↪with "trumpet" alloy profile
        alloy_x = [0.2, 0.5]              # minimum and maximum value of x content
        alloy_y = [0.1, 0.3]              # minimum and maximum value of y content
        x       = [20.0, 0]               # x coordinate of apex and x component of␣
↪axis direction [nm]
        y       = [20.0, 0]               # y coordinate of apex and y component of␣
↪axis direction [nm]
        z       = [11.0, 1]               # z coordinate of apex and z component of␣
↪axis direction [nm]
                                            # apex located at (20.0,20.0,11.0) (top␣
↪of inverted pyramid)
                                            # direction of center axis (0,0,1), i.e.␣
↪along z axis
                                            # The profile is symmetric with respect␣
↪to the inverse of the direction of the center axis,
                                            # i.e. (0,0,1) will lead to the same␣
↪trumpet profile as (0,0,-1).
        z0      = 1.25                    # parameter to vary the shape of the alloy␣
↪profile (minimum value is 1e-10)
        rho0    = 0.6                     # parameter to vary the shape of the alloy␣
↪profile (minimum value is 1e-10)
}
```

analogous for quinternaries:

quinternary_constant{}

quinternary_linear{}

quinternary_pyramid{}

quinternary_trumpet{}

7.8.13 structure{ region{} } - shape objects

- *1D simulations*
    - *line{}*
- *2D simulations*
    - *rectangle{}*
    - *circle{}*
    - *trapezoid{}*
    - *semiellipse{}*
    - *triangle{}*
    - *polygon{}*
    - *regular_polygon{}*

  - *hexagon{}*
- *3D simulations*
  - *cuboid{}*
  - *sphere{}*
  - *cylinder{}*
  - *obelisk{}*
  - *hexagon_obelisk{}*
  - *semiellipsoid{}*
  - *cone{}*
  - *polygonal_prism{}*
  - *regular_prism{}*
  - *hexagonal_prism{}*
  - *polygonal_pyramid{}*
  - *regular_pyramid{}*
  - *hexagonal_pyramid{}*
  - *pyramid{}*

Every region needs to have a certain shape, which can be defined by several objects. It consists of a certain material and/or contact, and it can have a doping profile.

Any subsequently defined region overwrites previously defined ones in the overlapping area. For exclusive properties such as material and contact, this implies a substitution of the old value.

Concerning doping, the new profile is added to any previously defined one.

Geometric objects may also be defined such that they are partially, mostly, or completely outside of the simulation region. Only the parts of structures which are inside of the simulation region will be used, everything else is ignored.

The following structures are supported. These are put under `structure{ region{} }`.

1D simulations

line{}

1D object. a line from start to end point along the specified direction

Example

```
line{                                   # 1D object
        x = [10.0, 20.0]                # a line from 10 nm to 20 nm along␣
↪the x direction
}
```

2D simulations

rectangle{}

2D object, a rectangle defined by two lines along the x and y directions

Example

```
rectangle{                          # 2D object, a rectangle defined by␣
↪two lines along the x and y directions
        x = [10.0, 20.0]            # a line from 10 nm to 20 nm along␣
↪the x direction
        y = [ 0.0,  5.0]            # a line from  0 nm to  5 nm along␣
↪the y dire
```

circle{}

2D object, a circle is defined by its center and radius

Example

```
circle{                                         # 2D object, a circle␣
↪is defined by its center and radius
        center{ x = 10.5   y = 14.0 }           # same as for regular_
↪polygon
        radius = 10.0                           # radius
}
```

trapezoid{}

2D object e.g. a simple trapezoid along the x axis

Example

```
trapezoid{                          # 2D object e.g. a simple trapezoid␣
↪along the x axis
        base_x     = [ 5, 15]       # base line extends in x direction␣
↪from 5 to 15 nm
        base_y     = [25, 25]       # base line has a constant y␣
↪coordinate y = 25 nm
        top_x      = [ 8, 12]       # top line extends in x direction␣
↪from 8 to 12 nm
        top_y      = [30, 30]       # top line has a constant y␣
↪coordinate y = 30 nm
}
```

Note

Exactly one of the elements `base_x` and `base_y` has to be set by two equal numbers to define the base line. The same holds for `top_x` and `top_y` to define the top line.

semiellipse{}

2D object, e.g. a simple semiellipse along the x axis

Example

```
semiellipse{                        # 2D object, e.g. a simple␣
↪semiellipse along the x axis
        base_x     = [45, 55]       # extension of base plane in x␣
↪direction, i.e. from 45 to 55 nm.
        base_y     = [ 5,  5]       # base line at y = 5 nm
        top        = [50, 15]       # top coordinate of the semiellipse␣
↪(x,y) = (50,15) in units of [nm]
}
```

Note

Exactly one of the elements base_x, and base_y has to be set by two equal numbers to define the base line.

triangle{}

2D object, a triangle defined by its 3 vertices

Example

```
triangle{                               # 2D object, a triangle defined␣
↪by its 3 vertices.
        vertex{ x = 10.5   y = 14.0 }   # a vertex P is defined by its␣
↪x and y coordinates: P=(x,y).
        vertex{ x =  0.0   y =  0.0 }   #
        vertex{ x =  5.0   y = 10.0 }   #
}
```

polygon{}

2D object, a polygon defined by its vertices. If the first and the last defined vertex are not identical, then they are joined with a line.

Example

```
polygon{                                # 2D object, a polygon defined␣
↪by its vertices. If the first and the last defined vertex are not␣
↪identical, then they are joined with a line.
        vertex{ x = 10.5   y = 14.0 }   # a vertex P is defined by its␣
↪x and y coordinates: P=(x,y). Multiple vertices can and must be␣
↪defined for a polygon.
                                        # Vertices must be ordered␣
↪either clockwise or counterclockwise, otherwise the behavior during␣
↪structure generation will be undefined.
}
```

regular_polygon{}

2D object, a polygon with equal angles and equal side lengths. It is defined by its center, one vertex and the number of facets.

Example

```
regular_polygon{                        # 2D object, a polygon with␣
↪equal angles and equal side lengths. It is defined by its center, one␣
↪vertex and the number of facets.
        center{ x = 10.5   y = 14.0 }   # The center point M is defined␣
↪by its x and y coordinates: M=(x,y).
        corner{ x = 20.0   y = 30.0 }   # A corner vertex P is defined␣
↪by its x and y coordinates: P=(x,y). Only one corner must be␣
↪specified. By modifying the corner coordinates the whole polygon can␣
↪easily be rotated around its center.
        number_of_facets = 7            # number of facets (= number of␣
↪vertices), must be >= 3
}
```

hexagon{}

2D object, a polygon with equal angles and equal side lengths and 6 facets. It is defined by its center and one corner vertex.

Example

```
hexagon{                                # 2D object, a polygon with␣
↪equal angles and equal side lengths and 6 facets. It is defined by␣
↪its center and one corner vertex.
        center{ x = 10.5    y = 14.0 }  # same as for regular_polygon
        corner{ x = 20.0    y = 30.0 }  # same as for regular_polygon
}
```

3D simulations

cuboid{}

3D object, a cuboid defined by three lines along the x, y and z directions

Example

```
cuboid{                             # 3D object, a cuboid defined by␣
↪three lines along the x, y and z directions
        x = [10.0, 20.0]            # a line from 10 nm to 20 nm along␣
↪the x direction
        y = [ 0.0,  5.0]            # a line from  0 nm to  5 nm along␣
↪the y direction
        z = [ 0.0,  5.0]            # a line from  0 nm to  5 nm along␣
↪the z direction
}
```

sphere{}

3D object, a sphere is defined by its center and radius

Example

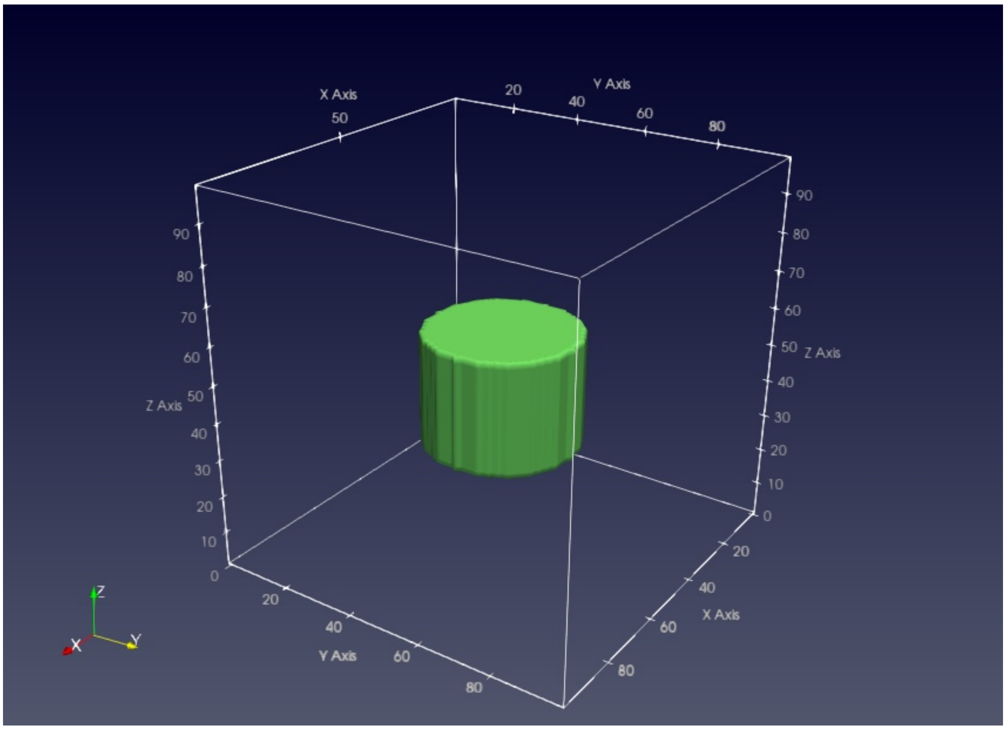
```
sphere{                                         # 3D object, a sphere␣
↪is defined by its center and radius
        center{ x = 10.5    y = 14.0    z = 1.0 } # similar as for circle
        radius = 10.0                           # radius
}
```

cylinder{}

3D object, e.g. a cylinder with a freely oriented axis

Example

```
cylinder{                                 # 3D object, e.g. a cylinder␣
↪with a freely oriented axis
        axis_start = [50.0, 50.0, 30.0]   # coordinates of starting␣
↪point of cylinder axis
        axis_end   = [50.0, 50.0, 60.0]   # coordinates of ending point␣
↪of cylinder axis
        radius     = 20.0                 # radius of cylinder
}
```



obelisk{}

3D object, e.g. an obelisk parallel to the (x,y) plane with top below bottom

Example

```
obelisk{                                 # 3D object, e.g. an obelisk␣
↪parallel to the (x,y) plane with top below bottom
        base_x     = [ 11, 19]           # extension of base plane in x␣
↪direction, i.e. from 11 to 19 nm.
        base_y     = [ 9,  21]           # extension of base plane in y␣
↪direction, i.e. from 9 to 21 nm.
        base_z     = [10, 10]            # base plane at z = 10 nm
        top_x      = [ 12, 18]           # extension of top plane in x␣
↪direction, i.e. from 12 to 18 nm.
        top_y      = [ 11, 19]           # extension of top plane in y␣
↪direction, i.e. from 11 to 19 nm.
        top_z      = [22, 22]            # top plane at z = 22 nm
}
```

> **Note**
>
> Exactly one of the elements `base_x`, `base_y` and `base_z` has to be set by two equal numbers to define the base plane. The same holds for `top_x`, `top_y` and `top_z` to define the top line.

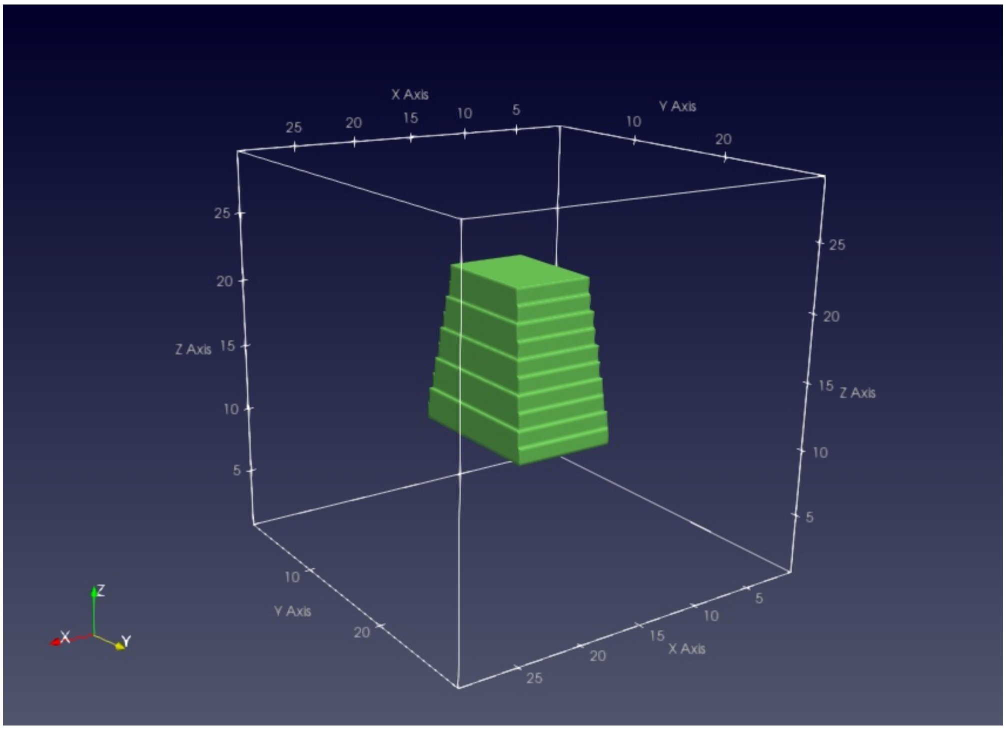
hexagon_obelisk{}

3D object, an obelisk with its base and top planes given by hexagons

Example

```
hexagon_obelisk{                     # 3D object, an obelisk with its␣
↪base and top planes given by hexagons
```

(continues on next page)



(continued from previous page)

```
        ... (same as obelisk to define position, orientation and␣
↪extension of object)
        permute     = yes/no          # (optional) switch between two␣
↪possible orientations of the hexagon within the rectangularly defined␣
↪planes
}
```

semiellipsoid{}

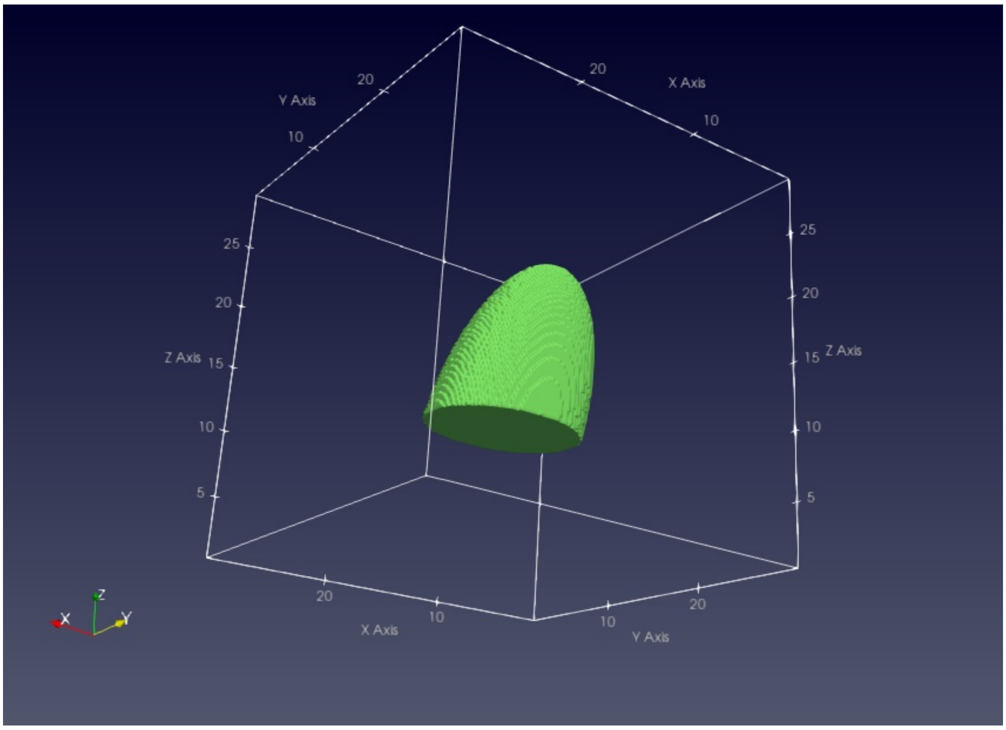
3D object, e.g. a semiellipsoid parallel to the (y,z) plane with top below bottom

Example

```
semiellipsoid{                                # 3D object, e.g. a␣
↪semiellipsoid parallel to the (x, y) plane with top below bottom
        base_x      = [9, 21]                 # extension of base plane in x␣
↪direction, i.e. from 9 to 21 nm.
        base_y      = [11,  20]               # extension of base plane in y␣
↪direction, i.e. from 11 to 20 nm.
        base_z      = [10, 10]                # base plane at z = 10 nm
        top         = [11, 15, 24]            # top coordinate of the␣
↪semiellipsoid (x,y,z) = (11,15,24) in units of [nm]
}
```

Note

Exactly one of the elements `base_x`, `base_y`, and `base_z` has to be set by two equal numbers to define the base plane.



cone{}

3D object, e.g. a cone parallel to the (x,z) plane

Example

```
cone{                                # 3D object, e.g. a cone parallel␣
↪to the (x,z) plane
        base_x     = [ 5, 20]        # extension of base plane in x␣
↪direction, i.e. from 5 to 20 nm.
        base_y     = [20, 20]        # base plane at y = 20 nm
        base_z     = [ 7, 19]        # extension of base plane in z␣
↪direction, i.e. from 7 to 19 nm.
        top        = [10, 30, 11]    # top coordinate of the cone (x,y,
↪z) = (10,30,11) in units of [nm]
        diminution = 0.0             # (optional) minimum value is 0.0␣
↪(i.e. cone), maximum value is 1.0 (i.e. cylinder)
                                     # diminution = 0.5 corresponds to
↪"half diameter of base diameter", default is 0.0 (i.e. cone)
}
```

> **Note**
>
> Exactly one of the elements `base_x`, `base_y`, and `base_z` has to be set by two equal numbers to define the base plane.

polygonal_prism{}

3D object (= 2D polygon with extension into the perpendicular direction; vertices define the circumference of the prism.)

Example

```
polygonal_prism{                        # 3D object (= 2D polygon with␣
↪extension into the perpendicular direction; vertices define the␣
↪circumference of the prism.)
        z = [0, 10]                     # define the extent in the␣
↪desired height direction. Here: Height is defined with respect to z␣
↪direction.
        vertex{ x = 10.5   y = 14.0 }   # a vertex P is defined by its␣
↪x and y coordinates: P=(x,y). Multiple vertices can and must be␣
↪defined for a polygon.
                                        # Vertices must be ordered␣
↪either clockwise or counterclockwise, otherwise the behavior during␣
↪structure generation will be undefined.
        axis = [0, 1, 1]                # (optional) inclination␣
↪(shear) of prism structure
                                        # (Obviously, cyclic␣
↪permutation of x, y, z are possible.)
}
```

regular_prism{}

3D object (= 2D regular_polygon with extension into the perpendicular direction; center and/or corner define the circumference of the prism.)

Example

```
regular_prism{                          # 3D object (= 2D regular_
↪polygon with extension into the perpendicular direction; center and/
↪or corner define the circumference of the prism.)
        z = [0, 10]                     # define the extent in the␣
↪desired height direction. Here: Height is defined with respect to z␣
↪direction.
        center{ x = 10.5   y = 14.0 }   # The center point M is defined␣
↪by its x and y coordinates: M=(x,y).
        corner{ x = 20.0   y = 30.0 }   # A corner vertex P is defined␣
↪by its x and y coordinates: P=(x,y). Only one corner must be␣
↪specified. By modifying the corner coordinates the whole polygon can␣
↪easily be rotated around its center.
        number_of_side_facets = 7       # number of side facets (=␣
↪number of vertices), must be >= 3
        axis = [0, 1, 1]                # (optional) inclination␣
↪(shear) of prism structure
                                        # (Obviously, cyclic␣
↪permutation of x, y, z are possible.)
}
```

hexagonal_prism{}

3D object (= 2D hexagon with extension into the perpendicular direction; center and/or corner define the circumference of the prism.)

Example

```
hexagonal_prism{                        # 3D object (= 2D hexagon with␣
↪extension into the perpendicular direction; center and/or corner␣
↪define the circumference of the prism.)
        z = [0, 10]                     # define the extent in the␣
↪desired height direction. Here: Height is defined with respect to z␣
```

(continues on next page)

(continued from previous page)

```
↪direction.
        center{ x = 10.5   y = 14.0 }   # same as for regular_polygon
        corner{ x = 20.0   y = 30.0 }   # same as for regular_polygon
        axis = [0, 1, 1]                # (optional) inclination␣
↪(shear) of prism structure
                                        # (Obviously, cyclic␣
↪permutation of x, y, z are possible.)
}
```

Note

Per default, all prisms (`polygonal_prism`, `regular_prism`, `hexagonal_prism`) are assumed to extend along the respective layer thickness direction (i.e. normal to the defining coordinate plane). But, using the `axis` vector, an arbitrary axis (inclination) direction for the prism can be defined in the simulation system. The axis vector does not need to be normalized, however, its orientation defines which side of the prism layer is the base to be used as reference for the inclination. For example,

```
regular_prism{
        z = [50, -70]                   # automatically reordered to [-70, 50]
        center{ x = 10   y = 10 }
        corner{ x = 30   y = 40 }
        number_of_side_facets = 8       # regular octagon wanted
        axis = [15 , 25 , 120]          # no normalization needed here
}
```

defines a regular octahedral prism extending primarily in the z direction (end surfaces are x-y planes at `z = -70` and `z = +50`). Since the axis points upwards in z direction (`z = 120`), the base surface to be taken as reference is the lower x-y plane at z = -70. There, the octagon center is at `{ x = 10 y = 10 }` with an octagon corner at `{ x = 30 y = 40 }` With the axis vector defined as above, we then find for the x-y plane at `z = +50`

- the octagon center at `{ x = 10+15 y = 10+25 }` and
- the octagon corner at `{ x = 30+15 y = 40+25 }`.

In analogy to polygon, we provide pyramidal structures.

polygonal_pyramid{}

Example

```
polygonal_pyramid{                      # 3D object
        z = [70, -70]                   # same as for polygonal_prism
        vertex{ x = 10.5   y = 14.0 }   # a vertex P is defined by its␣
↪x and y coordinates: P=(x,y). Multiple vertices can and must be␣
↪defined for a polygon.
                                        # Vertices must be ordered␣
↪either clockwise or counterclockwise, otherwise the behavior during␣
↪structure generation will be undefined.
        apex{   x = 10   y = 10   z = 120}
}
```

regular_pyramid{}

Example

```
regular_pyramid{                        # 3D object
        z = [70, -70]                   # same as for regular_prism
        center{ x = 10   y = 10 }       # same as for regular_prism
        corner{ x = 70   y = 70 }       # same as for regular_prism
        number_of_side_facets = 8       # same as for regular_prism
        apex{   x = 10   y = 10   z = 120}
}
```

hexagonal_pyramid{}

Example

```
hexagonal_pyramid{                        # 3D object
    z = [70, -70]                         # same as for hexagonal_prism
    center{ x = 10   y = 10 }             # same as for hexagonal_prism
    corner{ x = 70   y = 70 }             # same as for hexagonal_prism
    apex{   x = 10   y = 10   z = 120}
}
```

Note

Similar to the prismatic structures, use x, y, and z at the beginning of the respective primitive to define the extent in the desired height direction, use vertex, center, and/or corner to define the circumference of the base of the pyramid, and apex to define the position of the apex of the pyramid.

Note that, for polygonal_pyramid (as for polygon), the vertices must be ordered either clockwise or counterclockwise, otherwise the behavior during structure generation will be undefined.

Also note that if the apex is located outside of the interval defined by x, y, or z at the beginning in the height direction, the pyramid will be truncated. Also, the pyramid will point upwards if the apex is above the center of said interval (and the lower plane is used as base), and will point downwards if the apex is below the center (and the upper plane is used as base). And in case a symmetric regular pyramid is desired, please make sure to laterally align the apex with the center point.

For example

```
regular_pyramid{
        z = [70, -70]
        center{ x = 10   y = 10 }
        corner{ x = 70   y = 70 }
        number_of_side_facets = 8
        apex{   x = 10   y = 10   z = 120}
}
```

defines a regular octahedral pyramid with base at `z = -70`, centered there at `{ x = 10 y = 10 }` and a corner there at `{ x = 70 y = 70 }`. The apex of the pyramid would be at `{ x = 10 y = 10 z = 120}`, making the structure rotationally symmetric, except that the pyramid is truncated at `z = +70`. Thus, a rotationally symmetric truncated octahedral pyramid has been defined.

pyramid{}

3D object, e.g. a pyramid with 4 freely defined corner points

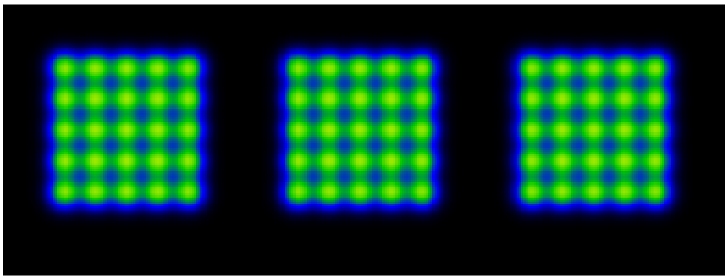
Example

```
pyramid{                              # 3D object, e.g. a pyramid with␣
↪4 freely defined corner points
    point1 = [50.0, 20.0, 30.0]       # coordinates of first point of␣
↪pyramid
    point2 = [50.0, 50.0, 80.0]       # coordinates of second point of␣
↪pyramid
    point3 = [80.0, 50.0, 50.0]       # coordinates of third point of␣
↪pyramid
    point4 = [50.0, 80.0, 30.0]       # coordinates of fourth point of␣
↪pyramid
}
```

Note

When `periodic{...}` is used, objects extending over an edge of the simulation region will not automatically be continued on the opposite side. If such objects are present in a periodic simulation, for each periodic coordinate direction (x, y or z), please either define a repetition (using the size of the simulation region as shift with `max = 1` and/or `min = 1` as needed), or extend an already present repetition to the edge of the simulation region (by increasing `min` and `max` as needed).

Additional Examples and Comments



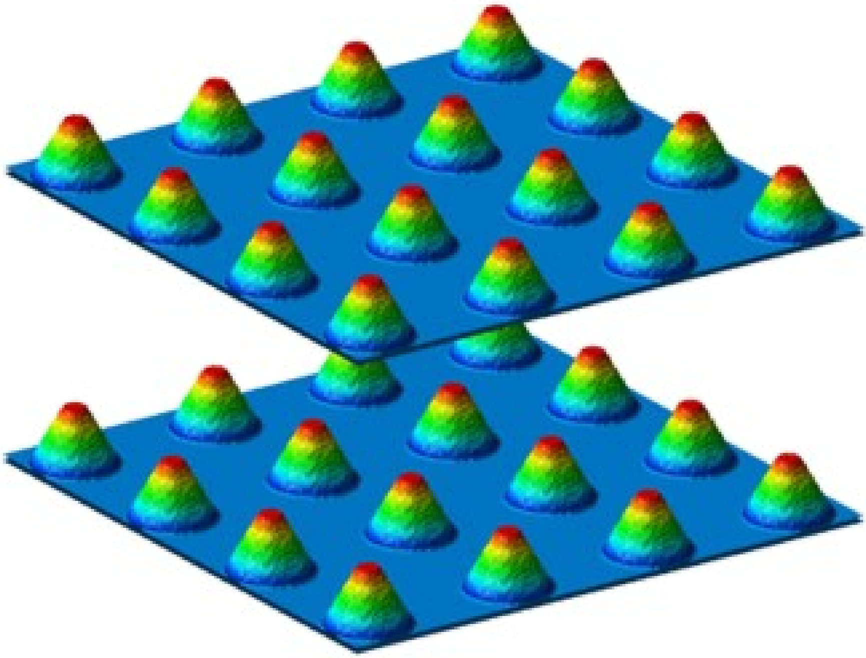
The pattern above can be produced by

```
structure{
    region{
        repeat_profiles = 'other doping'
        binary{ name = "InAs" }
        array_x{  shift=20  num=5 }
        array_y{  shift=20  num=5 }
        array2_x{ shift=150 num=3 }
        array2_y{ shift=150 num=3 }
        circle{
                center{ x = 100 y = 100 }
                radius = 30
        }
        doping{
            gaussian2D{
                name = B
                conc = 1e18
                x = 100
                y = 100
                sigma_x = 7
                sigma_y = 7
                add = yes
            }
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```



Two identical layers containing 16 quantum dots each, can be easily generated by specifying only one quantum dot geometry.

```
region{
    cone{                       # Here, the quantum dot has the shape of a cone.
        base_x      = [1.0,7.0]      # extension of base plane in x direction, i.e.␣
↪from 1.0 to 7.0 nm
        base_y      = [1.0,7.0]      # extension of base plane in y direction, i.e.␣
↪from 1.0 to 7.0 nm
        base_z      = [6.0,6.0]      # base plane at z = 6.0 nm
        top         = [4.0,4.0,10.0] # top coordinate of the cone (x,y,z) = (4.0,4.0,
↪10.0) in units of [nm]
        diminution = 0.25            # cone: diminution = 0.0, cylinder: diminution =␣
↪1.0
    }
        Note: Exactly one of the elements base_x, base_y, and base_z has to be set by␣
↪two equal numbers to define the base plane.

    ternary_linear{
        name     = "Al(x)Ga(1-x)As"  # AlxGa1-xAs
        alloy_x = [0.25, 1.0]        # vary alloy composition from x = 0.25 (Al0.25Ga0.
↪75As) to x = 1.0 (AlAs)
        z = [10, 6]            # vary alloy content from z = 10 nm to z = 6 nm
    }

    array_x{
        shift = 11.0
        max = 3
    }
```

(continues on next page)

(continued from previous page)

```
    array_y{
        shift = 11.0
        max = 3
    }
    array_z{
        shift = 20.0
        max = 1
    }
    repeat_profiles = "alloy"
}
```

Warning

Special care has to be taken when using `remove{}` or `add = no` for `doping{}`/`fixed charge`/`generation{}` in some repeated regions. Namely, repeated regions are created by sequentially creating multiple instances of a given region at the different positions defined by the `array_*` and `array2_*` statements. But the order in which these instances are created depends on undocumented implementation details and thus may change from release to release. For additive dopants/fixed charges/generation, or for repeated regions which do not self-overlap, the final structure and profiles do not depend on this undocumented creation order and thus no problems will occur. However, for repeated regions which self-overlap (e.g. due to small region shifts), using `remove{}` or `add = no` results in the final structure and profiles being dependent on that creation order and often being different from the user's intentions. Therefore, in case of doubt, please visually inspect your structure and profiles to avoid such issues.

7.9 grid{ }

Calling sequence

`grid{ }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

Specifications of the non-uniform rectangular grid lines.

Example

```
grid{
    xgrid{}
}

global{
    simulate1D{}
}
```

```
grid{
    xgrid{}
    ygrid{}
}

```

(continues on next page)

(continued from previous page)

```
global{
    simulate2D{}
}
```

```
grid{
    xgrid{}
    ygrid{}
    zgrid{}
}

global{
    simulate3D{}
}
```

Nested keywords

- *xgrid{ }*
- *xgrid{ min_pos }*
- *xgrid{ max_pos }*
- *xgrid{ allow_spacing_jumps }*
- *xgrid{ line{ } }*
- *xgrid{ line{ pos } }*
- *xgrid{ line{ spacing } }*
- *xgrid{ line{ array{ } } }*
- *xgrid{ line{ array{ shift } } }*
- *xgrid{ line{ array{ min } } }*
- *xgrid{ line{ array{ max } } }*
- *xgrid{ line{ array2{ } } }*
- *xgrid{ line{ array2{ shift } } }*
- *xgrid{ line{ array2{ min } } }*
- *xgrid{ line{ array2{ max } } }*
- *ygrid{ }*
- *zgrid{ }*
- *energy_grid{ }*
- *energy_grid{ min_energy }*
- *energy_grid{ max_energy }*
- *energy_grid{ energy_resolution }*

7.9.1 xgrid{ }

Calling sequence

`grid{ xgrid{ } }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

This group is used to define simulation space grid along the x-axis.

Example

```
grid{
    xgrid{}
}
```

7.9.2 xgrid{ min_pos }

Calling sequence

`grid{ xgrid{ min_pos } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Definition of the smallest, possible x-coordinate of the simulation domain. Grid lines specified with smaller x-coordinates are ignored.

Example

```
grid{
    xgrid{
        min_pos = -50
    }
}
```

7.9.3 xgrid{ max_pos }

Calling sequence

`grid{ xgrid{ max_pos } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Definition of the largest, possible x-coordinate of the simulation domain. Grid lines specified with larger x-coordinates are ignored.

Example

```
grid{
    xgrid{
        min_pos = 150
    }
}
```

7.9.4 xgrid{ allow_spacing_jumps }

Calling sequence

`grid{ xgrid{ allow_spacing_jumps } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes`, then it is possible to assign two different grid spacing values to the same grid line, which creates a jump in the grid spacing.

Example

```
grid{
    xgrid{
        allow_spacing_jumps = yes
    }
}
```

7.9.5 xgrid{ line{ } }

Calling sequence

`grid{ xgrid{ line{ } } }`

Properties

- **usage:** required
- **items:** minimum 2

Functionality

Group defining a grid lines. As the lines define the total size of the device, at least two of them have to be present for each simulation direction.

Example

```
grid{
    xgrid{
        line{ }
    }
}
```

7.9.6 xgrid{ line{ pos } }

Calling sequence

`grid{ xgrid{ line{ pos } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

Position of the line.

Hint

A good practice is to define lines on all interfaces in the device to provide the geometry definition possibly independent to the choice of the spacing.

Example

```
grid{
    xgrid{
        line{ pos = 5.0   spacing = 0.2 }
    }
}
```

7.9.7 xgrid{ line{ spacing } }

Calling sequence

`grid{ xgrid{ line{ spacing } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-3, ...)`
- **unit:** nm

Functionality

A grid spacing in the vicinity of the position of the line.

Example

```
grid{
    xgrid{
        line{ pos = 5.0   spacing = 0.2 }
    }
}
```

7.9.8 xgrid{ line{ array{ } } }

Calling sequence

`grid{ xgrid{ line{ array{ } } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- *xgrid{ line{ array{ } } }* is required if *xgrid{ line{ array2{ } } }* is specified.

Functionality

Repeating a single grid line multiple times at equidistant positions. The grid lines are placed according to the following equation:

$x_n = \texttt{pos} + \texttt{shift} \times n,$

where $n = \texttt{min}, \ldots, \texttt{max}$

Example

```
grid{
    xgrid{
        line{
            pos = 5.0   spacing = 0.2
            array{...}
        }
    }
}
```

7.9.9 xgrid{ line{ array{ shift } } }

Calling sequence

`grid{ xgrid{ line{ array{ shift } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

The distance between repeated grid lines.

Example

```
grid{
    xgrid{
        line{
        line{
            pos = 5.0   spacing = 0.2
            repeat{ shift = 1.8 }
        }
    }
}
```

7.9.10 xgrid{ line{ array{ min } } }

Calling sequence

`grid{ xgrid{ line{ array{ min } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** $-$
- **default:** $z = 0$

Functionality

Number of repeated grid lines in negative x-direction, without counting the original grid line.

Example

```
grid{
    xgrid{
        line{
            pos = 5.0   spacing = 0.2
            array{ shift = 1.8   min = 5 }
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

7.9.11 xgrid{ line{ array{ max } } }

Calling sequence

`grid{ xgrid{ line{ array{ max } } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of repeated grid lines in positive x-direction, without counting the original grid line.

Example

```
grid{
    xgrid{
        line{ pos = 5.0   spacing = 0.2
            array{ shift = 1.8   max = 5 }
        }
    }
}
```

7.9.12 xgrid{ line{ array2{ } } }

Calling sequence

`grid{ xgrid{ line{ array2{ } } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- *xgrid{ line{ array{ } } }* is required to use *xgrid{ line{ array2{ } } }*.

Functionality

This group is intended to be used in conjunction with the group *xgrid{ line{ array{ } } }*. It allows to repeat the pattern of grid lines generated by *xgrid{ line{ array{ } } }* multiple times at equidistant positions.

Example

```
grid{
    xgrid{
        line{ pos = 5.0   spacing = 0.2
            array{ shift = 1.8   max = 5 }
            array2{...}
        }
    }
}
```

7.9.13 xgrid{ line{ array2{ shift } } }

Calling sequence

`grid{ xgrid{ line{ array2{ shift } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm

Functionality

The distance between repeated grid lines.

Example

```
grid{
    xgrid{
        line{ pos = 5.0   spacing = 0.2
            array{ shift = 1.8   max = 5 }
            array2{ shift = 20.0 }
        }
    }
}
```

7.9.14 xgrid{ line{ array2{ min } } }

Calling sequence

`grid{ xgrid{ line{ array2{ min } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **unit:** $-$
- **default:** $z = 0$

Functionality

Number of repetitions in negative x-direction, without counting the original array of grid lines.

Example

```
grid{
    xgrid{
        line{ pos = 5.0   spacing = 0.2
            array{ shift = 1.8   max = 5 }
            array2{ shift = 20.0   min = 7 }
        }
    }
}
```

7.9.15 xgrid{ line{ array2{ max } } }

Calling sequence

`grid{ xgrid{ line{ array2{ max } } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **unit:** $-$

Functionality

Number of repetitions in positive x-direction, without counting the original array of grid lines.

Example

```
grid{
    xgrid{
        line{ pos = 5.0   spacing = 0.2
            array{ shift = 1.8   min = 2   max = 5 }
            array2{ shift = 20.0   min = 1   max = 3 }
        }
    }
}
```

7.9.16 ygrid{ }

Calling sequence

`grid{ ygrid{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is required if either *simulate2D{ }* or *simulate2D{ }* is specified in the *global{ }* group.
- It is not allowed if *simulate1D{ }* is specified in the *global{ }* group.

Functionality

This group is used to define simulation space grid along the y-axis. This group has the same properties and allowed keywords as *xgrid{ }*.

Example

```
grid{
    ygrid{
        line{ pos = 5.0   spacing = 0.2
            array{ shift = 1.8   min = 2   max = 5 }
            array2{ shift = 20.0   min = 1   max = 3 }
        }
    }
}
```

7.9.17 zgrid{ }

Calling sequence

`grid{ zgrid{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is required if *simulate3D{ }* is specified in the *global{ }* group.
- It is not allowed if either *simulate1D{ }* or *simulate2D{ }* is specified in the *global{ }* group.

Functionality

This group is used to define simulation space grid along the z-axis. This group has the same properties and allowed keywords as *xgrid{ }*.

Example

```
grid{
    zgrid{
        line{ pos = 5.0   spacing = 0.2
            array{ shift = 1.8   min = 2   max = 5 }
            array2{ shift = 20.0   min = 1   max = 3 }
        }
    }
}
```

7.9.18 energy_grid{ }

Calling sequence

`grid{ energy_grid{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Specifying the discretization of energy.

Example

```
grid{
    energy_grid{...}
}
```

7.9.19 energy_grid{ min_energy }

Calling sequence

`grid{ energy_grid{ min_energy } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

Low-energy boundary of the energy grid.

Example

```
grid{
    energy_grid{
        min_energy = - 2.1
        max_energy = 1.7
    }
}
```

7.9.20 energy_grid{ max_energy }

Calling sequence

`grid{ energy_grid{ max_energy } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

High-energy boundary of the energy grid.

Example

```
grid{
    energy_grid{
        min_energy = - 2.1
        max_energy = 1.7
    }
}
```

7.9.21 energy_grid{ energy_resolution }

Calling sequence

`grid{ energy_grid{ energy_resolution } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV
- **default:** $r = 1e - 2$

Functionality

Spacing between subsequent energy grid points.

Example

```
grid{
    energy_grid{
        min_energy = - 2.1
        max_energy = 1.7
        energy_resolution = 0.005
    }
}
```

7.10 classical{ }

Calling sequence

`classical{ }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

This group specifies bands entering simulation, allows computing bulk electronic band structures, selects carrier statistics, initializes some energy resolved calculations, controls outputs of bulk-like properties.

Examples

```
classical{
    Gamma{}
    X{}
    L{}
    HH{}
    LH{}
    SO{}
}

global{
    ...
    crystal_zb{...}
}
```

```
classical{
    Gamma{}
    HH{}
    LH{}
    SO{}
}

global{
    ...
    crystal_wz{...}
}
```

Nested keywords

7.10.1 Gamma{ }

Calling sequence

`classical{ Gamma{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of the following: *Gamma{ }*, *X{ }*, *Delta{ }*, and *L{ }* is required if *global{ crystal_zb{ } }* is present in the input file.
- The *Gamma{ }* is required if *global{ crystal_wz{ } }* is present in the input file.

Functionality

By calling this group, a **conduction** band with a minimum at Γ point becomes available in the model. This band is referred to as `Gamma` in output files.

Example

```
classical{
    Gamma{}
    HH{}
}
```

Nested keywords

- *output_bandedge{ }*
- *output_bandedge{ averaged }*

---

output_bandedge{ }

Calling sequence

`classical{ Gamma{ output_bandedge{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output minimum (band edge) of this band as energy profile in a single file [eV].

Example

```
classical{
    Gamma{
        output_bandedge{}
    }
    HH{}
}
```

---

output_bandedge{ averaged }

Calling sequence

`classical{ Gamma{ output_bandedge{ averaged } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then, for each grid point, the energy profile will be averaged between neighboring material grid points. If set to `no` then abrupt discontinuities at interfaces are visible in the output files (in 1D two points, in 2D four points, in 3D eight points for each grid point).

Example

```
classical{
    Gamma{
        output_bandedge{
            averaged = yes
        }
    }
    HH{}
}
```

7.10.2 HH{ }

Calling sequence

`classical{ HH{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *LH{ }*, *HH{ }*, and *SO{ }* is required.

Functionality

By calling this group, a **heavy-hole valence** band with maximum at Γ point becomes available in the model. This band is referred to as `HH` in output files.

Example

```
classical{
    Gamma{}
    HH{}
}
```

Nested keywords

- *output_bandedge{ }*
- *output_bandedge{ averaged }*

---

output_bandedge{ }

Calling sequence

`classical{ HH{ output_bandedge{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output minimum (band edge) of this band as energy profile in a single file [eV].

Example

```
classical{
    HH{
        output_bandedge{}
    }
    Gamma{}
}
```

---

output_bandedge{ averaged }

Calling sequence

`classical{ HH{ output_bandedge{ averaged } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then, for each grid point, the energy profile will be averaged between neighboring material grid points. If set to `no` then abrupt discontinuities at interfaces are visible in the output files (in 1D two points, in 2D four points, in 3D eight points for each grid point).

Example

```
classical{
    HH{
        output_bandedge{
            averaged = yes
        }
    }
    Gamma{}
}
```

7.10.3 LH{ }

Calling sequence

`classical{ LH{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *LH{ }*, *HH{ }*, and *SO{ }* is required.

Functionality

By calling this group, a **light-hole valence** band with maximum at Γ point becomes available in the model. This band is referred to as `LH` in output files.

Example

```
classical{
    Gamma{}
    LH{}
}
```

Nested keywords

- *output_bandedge{ }*
- *output_bandedge{ averaged }*

---

output_bandedge{ }

Calling sequence

`classical{ LH{ output_bandedge{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output minimum (band edge) of this band as energy profile in a single file [eV].

Example

```
classical{
    LH{
        output_bandedge{}
    }
    Gamma{}
}
```

output_bandedge{ averaged }

Calling sequence

`classical{ LH{ output_bandedge{ averaged } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then, for each grid point, the energy profile will be averaged between neighboring material grid points. If set to `no` then abrupt discontinuities at interfaces are visible in the output files (in 1D two points, in 2D four points, in 3D eight points for each grid point).

Example

```
classical{
    LH{
        output_bandedge{
            averaged = yes
        }
    }
    Gamma{}
}
```

7.10.4 SO{ }

Calling sequence

`classical{ SO{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *LH{ }*, *HH{ }*, and *SO{ }* is required.

Functionality

By calling this group, a **split-off valence** (or **crystal-field split-off** in wurtzite) band with maximum at Γ point becomes available in the model. This band is referred to as `SO` in output files.

Example

```
classical{
    Gamma{}
    SO{}
}
```

Nested keywords

- *output_bandedge{ }*
- *output_bandedge{ averaged }*

---

output_bandedge{ }

Calling sequence

`classical{ SO{ output_bandedge{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output minimum (band edge) of this band as energy profile in a single file [eV].

Example

```
classical{
    SO{
        output_bandedge{}
    }
    Gamma{}
}
```

---

output_bandedge{ averaged }

Calling sequence

`classical{ SO{ output_bandedge{ averaged } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then, for each grid point, the energy profile will be averaged between neighboring material grid points. If set to `no` then abrupt discontinuities at interfaces are visible in the output files (in 1D two points, in 2D four points, in 3D eight points for each grid point).

Example

```
classical{
    SO{
        output_bandedge{
            averaged = yes
        }
    }
    Gamma{}
}
```

7.10.5 X{ }

Calling sequence

`classical{ X{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The *X{ }* and *Delta{ }* cannot be defined simultaneously.
- The *X{ }* is not allowed if *global{ crystal_wz{ } }* is present in the input file.
- At least one of *Gamma{ }*, *X{ }*, *Delta{ }*, and *L{ }* is required if *global{ crystal_zb{ } }* is present in the input file.

Functionality

By calling this group, three **conduction** bands with minimums at X points become available in the model. The bands are referred to as `X_1`, `X_2`, and `X_3` for the X valleys located at `[1 0 0]`, `[0 1 0]`, and `[0 0 1]` directions, respectively, in output files.

Attention

This group does not apply to Si, Ge, GaP, and to materials with wurtzite symmetry

Example

```
classical{
    X{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

Nested keywords

- *output_bandedge{ }*

- *output_bandedge{ averaged }*

---

output_bandedge{ }

Calling sequence

`classical{ X{ output_bandedge{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output minimum (band edge) of this band as energy profile in a single file [eV].

Example

```
classical{
    X{
        output_bandedge{}
    }
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

---

output_bandedge{ averaged }

Calling sequence

`classical{ X{ output_bandedge{ averaged } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then, for each grid point, the energy profile will be averaged between neighboring material grid points. If set to `no` then abrupt discontinuities at interfaces are visible in the output files (in 1D two points, in 2D four points, in 3D eight points for each grid point).

Example

```
classical{
    X{
        output_bandedge{
            averaged = yes
        }
    }
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

7.10.6 Delta{ }

Calling sequence

`classical{ Delta{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The *X{ }* and *Delta{ }* cannot be defined simultaneously.
- The *Delta{ }* is not allowed if *global{ crystal_wz{ } }* is present in the input file.
- At least one of *Gamma{ }*, *X{ }*, *Delta{ }*, and *L{ }* is required if *global{ crystal_zb{ } }* is present in the input file.

Functionality

By calling this group, three **conduction** bands with minimums along the Δ lines become available in the model. The bands are referred to as `Delta_1`, `Delta_2`, and `Delta_3` for the Δ valleys located at `[1 0 0]`, `[0 1 0]`, and `[0 0 1]` directions, respectively, in output files.

Attention

This group applies to Si, Ge, GaP

Example

```
classical{
    Delta{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

Nested keywords

- *output_bandedge{ }*
- *output_bandedge{ averaged }*

---

output_bandedge{ }

Calling sequence

`classical{ Delta{ output_bandedge{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output minimum (band edge) of this band as energy profile in a single file [eV].

Example

```
classical{
    Delta{
        output_bandedge{}
    }
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

---

output_bandedge{ averaged }

Calling sequence

`classical{ Delta{ output_bandedge{ averaged } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then, for each grid point, the energy profile will be averaged between neighboring material grid points. If set to `no` then abrupt discontinuities at interfaces are visible in the output files (in 1D two points, in 2D four points, in 3D eight points for each grid point).

Example

```
classical{
    Delta{
        output_bandedge{
            averaged = yes
        }
    }
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

7.10.7 L{ }

Calling sequence

`classical{ L{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The *L{ }* is not allowed if *global{ crystal_wz{ } }* is present in the input file.
- If *global{ crystal_zb{ } }* is present in the input file, then at least one of the following: *Gamma{ }*, *X{ }*, *Delta{ }*, and *L{ }* must be defined.

Functionality

By calling this group, four **conduction** bands with minimums at L points become available in the model. The bands are referred to as `L_1`, `L_2`, `L_3`, and `L_4` for the L valleys located at `[1 1 1]`, `[1 -1 1]`, `[1 -1 -1]`, and `[1 1 -1]` directions, respectively, in output files.

> **Note**
>
> This group does not apply to materials with wurtzite symmetry.

Example

```
classical{
    L{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

Nested keywords

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

---

output_bandedge{ }

Calling sequence

`classical{ L{ output_bandedge{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output minimum (band edge) of this band as energy profile in a single file [eV].

Example

```
classical{
    L{
        output_bandedge{}
    }
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

---

output_bandedge{ averaged }

Calling sequence

`classical{ L{ output_bandedge{ averaged } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then, for each grid point, the energy profile will be averaged between neighboring material grid points. If set to `no` then abrupt discontinuities at interfaces are visible in the output files (in 1D two points, in 2D four points, in 3D eight points for each grid point).

Example

```
classical{
    L{
        output_bandedge{
            averaged = yes
        }
    }
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

7.10.8 carrier_statistics

Calling sequence

`classical{ carrier_statistics }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `maxwell_boltzmann` or `fermi_dirac`
- **default:** `fermi_dirac`

Functionality

Attribute to chose carrier statistics.

If set to `maxwell_boltzmann`, then Maxwell-Boltzmann statistics is used for the classical densities. If set to `fermi_dirac`, then Fermi-Dirac statistics is used for the classical densities. It is not recommended as this is only an approximation which is only applicable in certain cases.

In order to maintain consistency, also the (integrated) energy distribution (density_vs_energy) and the classical emission spectra and densities are computed using the same statistics. Use together with quantum regions is possible but not recommended, and convergence of the current-Poisson or quantum-current-Poisson equation may become worse (please readjust convergence parameters accordingly).

Note

- $n = N_c\, \mathcal{F}_{1/2}\left(\frac{E_F - E_c}{k_B T}\right)$ (electron density for `fermi_dirac`)
- $p = N_c\, \mathcal{F}_{1/2}\left(\frac{E_v - E_F}{k_B T}\right)$ (hole density for `fermi_dirac`)
- $n = N_c \exp\left(\frac{E_F - E_c}{k_B T}\right)$ (electron density for `maxwell_boltzmann`)
- $p = N_c \exp\left(\frac{E_v - E_F}{k_B T}\right)$ (hole density for `maxwell_boltzmann`)
- where $\mathcal{F}_n(E)$ is a Fermi-Dirac integral of the order n.

Example

```
classical{
    carrier_statistics = maxwell_boltzmann

    Gamma{}
    HH{}
}
```

7.10.9 limit_classical_density

Calling sequence

`classical{ limit_classical_density }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 2$
- **default:** $z = 0$

Functionality

This keyword can be used to improve convergence of classical Poisson and current equations by using different Fermi-Dirac Integral. It may help for solving systems with locally degenerate materials, like systems with 2DEGs, as semi-classical convergence will be reached more likely. Using it for degenerate systems may notably alter solutions and is not advised. When set to `0` then integral $F_{1/2}$ is used. 1 results in F_0, and 2 in F_{-1} being used. See *Charge densities* for more details.

Note

The choice of the value does not impact solutions when quasi-Fermi levels are located deep in the band gap.

Last update: 02/04/2025

7.10.10 energy_distribution{ }

Calling sequence

`classical{ energy_distribution{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Definition and output of integrated electron and hole density as a function of energy, $n(E)$, $p(E)$ in units of $[\mathrm{cm}^{-2}\mathrm{eV}^{-1}]$ in 1D, $[\mathrm{cm}^{-1}\mathrm{eV}^{-1}]$ in 2D, and $[\mathrm{eV}^{-1}]$ in 3D.

Attention

- *min_energy*, *max_energy* always refer to a zero point at the (local) conduction band edge, and not to the photon energy.
- max should be set high enough above 0 to contain all occupied electron states and min should be set far enough below the band gap to contain all occupied hole states.
- The respective values for *energy_resolution* should be set smaller than $k_B T$ if one wishes to fully resolve the structures of the integrated densities and/or of the emission spectra.
- However, while setting *energy_resolution* as low as 0.001 eV has little influence on program execution time, using similarly small values for *energy_resolution* in *energy_resolved_density{ }* will result in massive slowdowns (and in 3D also in massive memory use), since the computational effort for obtaining emission spectra grows quadratically with the number of energy bins.

Note

Currently available only for 1-band models.

Example

```
classical{
    energy_distribution{...}

    Gamma{}
    HH{}
}
```

Nested keywords

- *min_energy*
- *max_energy*
- *energy_resolution*
- *only_density_quantum_regions*

---

min_energy

Calling sequence

`classical{ energy_distribution{ min_energy } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

minimum energy

Example

```
classical{
    energy_distribution{
        min_energy = -0.5
        max_energy = 1.8
    }

    Gamma{}
    HH{}
}
```

---

max_energy

Calling sequence

`classical{ energy_distribution{ max_energy } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

maximum energy

Example

```
classical{
    energy_distribution{
        min_energy = -0.5
        max_energy = 1.8
    }

    Gamma{}
    HH{}
}
```

energy_resolution

Calling sequence

`classical{ energy_distribution{ energy_resolution } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV
- **default:** $r = 0.1$

Functionality

energy spacing

Example

```
classical{
    energy_distribution{
        min_energy = -0.5
        max_energy = 1.8
        energy_resolution = 0.01
    }

    Gamma{}
    HH{}
}
```

---

only_density_quantum_regions

Calling sequence

`classical{ energy_distribution{ only_density_quantum_regions } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- *quantum{ region{ } }* must be defined in the input file to use *only_density_quantum_regions*.

Functionality

This keyword can be used to suppress contributions from outside the quantum regions of interest. This works even if quantum mechanics is not enabled in *run{ }*.

> **Note**
>
> Note that *energy_distribution{ }*, which directly calculates the space-integrated energy-resolved density, is independent on the group *energy_resolved_density{ }*.

Example

```
classical{
    energy_distribution{
        only_density_quantum_regions = yes

        min_energy = -0.5
        max_energy = 1.8
    }

    Gamma{}
    HH{}
    quantum{
        region{...}
    }
}
```

7.10.11 energy_resolved_density{ }

Calling sequence

`classical{ energy_resolved_density{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The group *grid{ energy_grid{ } }* must be present in the input file.

Functionality

Generates and outputs electron and hole density as a function of energy and position, $n(x, E)$, $p(x, E)$ in units of $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ in 1D, $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ in 2D, and $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ in 3D.

Examples

```
classical{
    energy_resolved_density{}

    Gamma{}
    HH{}
}

grid{
    energy_grid{...}
}
```

Nested keywords

- *only_density_quantum_regions*
- *output_energy_resolved_densities{ }*
- *output_LDOS{ }*

---

only_density_quantum_regions

Calling sequence

`classical{ energy_resolved_density{ only_density_quantum_regions } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- *quantum{ region{ } }* must be defined in the input file to use *only_density_quantum_regions*.

Functionality

If set to `yes` then only quantum regions are considered for densities of states. It can be used to suppress contributions from outside the quantum regions of interest. The keyword works also if quantum mechanics is not enabled in *run{ }*.

Examples

```
classical{
    energy_resolved_density{
        only_density_quantum_regions = yes
    }

    Gamma{}
    HH{}
}

grid{
    energy_grid{...}
}
quantum{
    region{...}
}
```

---

output_energy_resolved_densities{ }

Calling sequence

`classical{ energy_resolved_density{ output_energy_resolved_densities{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The *output_energy_resolved_densities{ }* is not allowed if *global{ simulate3D{ } }* is already present in the input file.

Functionality

If defined then energy-resolved carrier densities $n(x, E)$, $p(x, E)$ in units of $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ in 1D and $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ in 2D.

> **Note**
>
> Currently available only for 1-band models.

Examples

```
classical{
    energy_resolved_density{
        output_energy_resolved_densities{}
    }

    Gamma{}
    HH{}
}

grid{
    energy_grid{...}
}
global{
    simulate1D{...}
}
```

---

output_LDOS{ }

Calling sequence

`classical{ energy_resolved_density{ output_LDOS{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The *output_LDOS{ }* is not allowed if *global{ simulate3D{ } }* is already present in the input file.

Functionality

If defined then energy-resolved densities of states in units of $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ in 1D and $[\mathrm{cm}^{-3}\mathrm{eV}^{-1}]$ in 2D.

Note

Currently available only for 1-band models.

Examples

```
classical{
    energy_resolved_density{s
        output_LDOS{}
    }

    Gamma{}
    HH{}
}

grid{
    energy_grid{...}
}
global{
    simulate2D{...}
}
```

7.10.12 bulk_dispersion{ }

Calling sequence

`classical{ bulk_dispersion{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- The *global{ magnetic_field{ } }* is must not be specified in the input file.

Functionality

This group allows calculating bulk band structures of the materials at specific positions in the simulation domain within 1-band approximations or $\mathbf{k}\cdot\mathbf{p}$ models. The computation is performed just after initialization of the structure. Related outputs are located in the root output directory of the simulation.

Example

```
classical{
    bulk_dispersion{
        Gamma{}
        KP8{}
        KP30{}

        path{
            name = "name_1"
```

(continues on next page)

(continued from previous page)

```
            ...
        }
        path{
            name = "name_2"
            ...
        }
        full{
            name = "name_3"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

Nested keywords

- *Gamma{ }*
- *X{ }*
- *Delta{ }*
- *L{ }*
- *HH{ }*
- *LH{ }*
- *SO{ }*
- *KP6{ }*
- *KP6{ use_Luttinger_parameters }*
- *KP6{ approximate_kappa }*
- *KP8{ }*
- *KP8{ use_Luttinger_parameters }*
- *KP8{ from_6band_parameters }*
- *KP8{ evaluate_S }*
- *KP8{ rescale_S_to }*
- *KP8{ approximate_kappa }*
- *KP8{ electron_far_band }*
- *KP8{ correct_electron_gfactor }*
- *KP8{ rescale_kp_everywhere }*
- *KP8{ avoid_spurious }*
- *KP14{ }*

- *KP14{ use_Luttinger_parameters }*
- *KP14{ from_6band_parameters }*
- *KP14{ evaluate_S }*
- *KP30{ }*
- *full{ }*
- *full{ name }*
- *full{ position{ } }*
- *full{ position{ x } }*
- *full{ position{ y } }*
- *full{ position{ z } }*
- *full{ shift_holes_to_zero }*
- *full{ kxgrid{ } }*
- *full{ kxgrid{ line{ } } }*
- *full{ kxgrid{ line{ pos } } }*
- *full{ kxgrid{ line{ spacing } } }*
- *full{ kygrid{ } }*
- *full{ kzgrid{ } }*
- *path{ }*
- *path{ name }*
- *path{ position{ } }*
- *path{ position{ x } }*
- *path{ position{ y } }*
- *path{ position{ z } }*
- *path{ shift_holes_to_zero }*
- *path{ point{ } }*
- *path{ point{ k } }*
- *path{ spacing }*
- *path{ num_points }*
- *lines{ }*
- *lines{ name }*
- *lines{ position{ } }*
- *lines{ position{ x } }*
- *lines{ position{ y } }*
- *lines{ position{ z } }*
- *lines{ shift_holes_to_zero }*
- *lines{ k_max }*
- *lines{ spacing }*
- *output_bulk_dispersions{ }*

- *output_masses{ }*
- *output_inverse_masses{ }*
- *output_k_vectors{ }*

Gamma{ }

Calling sequence

`classical{ bulk_dispersion{ Gamma{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, or *KP8{ }* is required if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, the bulk electronic band structure is computed within 1-band parabolic model using effective mass tensor for the conduction band at Γ.

Example

```
classical{
    bulk_dispersion{
        Gamma{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

X{ }

Calling sequence

`classical{ bulk_dispersion{ X{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

—

Functionality

When this group is defined, the bulk electronic band structure is computed within 1-band parabolic model using effective mass tensor for the conduction band at X.

Delta{ }

Calling sequence

`classical{ bulk_dispersion{ Delta{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

—

Functionality

When this group is defined, the bulk electronic band structure is computed within 1-band parabolic model using effective mass tensor for the conduction band at Δ.

L{ }

Calling sequence

`classical{ bulk_dispersion{ L{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

—

Functionality

When this group is defined, the bulk electronic band structure is computed within 1-band parabolic model using effective mass tensor for the conduction band at L.

HH{ }

Calling sequence

`classical{ bulk_dispersion{ HH{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, or *KP8{ }* is required if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, the bulk electronic band structure is computed within 1-band parabolic model using effective mass tensor for the heavy-hole valence band.

Example

```
classical{
    bulk_dispersion{
        HH{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

LH{ }

Calling sequence

`classical{ bulk_dispersion{ LH{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, or *KP8{ }* is required if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, the bulk electronic band structure is computed within 1-band parabolic model using effective mass tensor for the light-hole valence band.

Example

```
classical{
    bulk_dispersion{
        LH{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

SO{ }

Calling sequence

`classical{ bulk_dispersion{ SO{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, or *KP8{ }* is required if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, the bulk electronic band structure is computed within 1-band parabolic model using effective mass tensor for the split-off valence band.

Example

```
classical{
    bulk_dispersion{
        SO{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
```

(continues on next page)

(continued from previous page)

```
    Gamma{}
    HH{}
}
```

KP6{ }

Calling sequence

`classical{ bulk_dispersion{ KP6{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, or *KP8{ }* is required if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, 6-band $\mathbf{k} \cdot \mathbf{p}$ model is applied to compute the bulk electronic band structure.

Example

```
classical{
    bulk_dispersion{
        KP6{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

KP6{ use_Luttinger_parameters }

Calling sequence

`classical{ bulk_dispersion{ KP6{ use_Luttinger_parameters } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** yes or no
- **default:** no

Dependencies

- *KP6{ use_Luttinger_parameters }* and *KP6{ approximate_kappa }* are not allowed if *global{ crystal_wz{ } }* is already present.

Functionality

By default the solver uses the DKK (Dresselhaus-Kip-Kittel) parameters (L, M, N). If enabled then it uses Luttinger parameters (γ_1, γ_2, γ_3) instead.

Example

```
classical{
    bulk_dispersion{
        KP6{
            use_Luttinger_parameters = yes
        }

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP6{ approximate_kappa }

Calling sequence

`classical{ bulk_dispersion{ KP6{ approximate_kappa } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- *KP6{ use_Luttinger_parameters }* and *KP6{ approximate_kappa }* are not allowed if *global{ crystal_wz{ } }* is already present.

Functionality

By default the κ for zinc blende crystal structure is taken from the database or input file. If this is enabled then the solver is forced to approximate kappa through others 6-band $\mathbf{k} \cdot \mathbf{p}$ parameters, even though kappa is given in database or input file.

Example

```
classical{
    bulk_dispersion{
        KP6{
            approximate_kappa = yes
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP8{ }

Calling sequence

`classical{ bulk_dispersion{ KP8{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, or *KP8{ }* is required if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, 8-band $\mathbf{k} \cdot \mathbf{p}$ model is applied to compute the bulk electronic band structure.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
```

(continues on next page)

(continued from previous page)

```
    Gamma{}
    HH{}
}
```

KP8{ use_Luttinger_parameters }

Calling sequence

`classical{ bulk_dispersion{ KP8{ use_Luttinger_parameters } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- *KP6{ use_Luttinger_parameters }* and *KP6{ approximate_kappa }* is not allowed if *global{ crystal_wz{ } }* is already present.

Functionality

By default the solver uses the DKK (Dresselhaus-Kip-Kittel) parameters (L, M, N). If enabled then it uses Luttinger parameters (γ_1, γ_2, γ_3) instead.

Example

```
classical{
    bulk_dispersion{
        KP8{
            use_Luttinger_parameters = yes
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP8{ from_6band_parameters }

Calling sequence

`classical{ bulk_dispersion{ KP8{ from_6band_parameters } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

By default the 8-band $\mathbf{k}\cdot\mathbf{p}$ parameters are taken from database or input file. If enabled then it evaluates the 8-band $\mathbf{k}\cdot\mathbf{p}$ parameters from 6-band $\mathbf{k}\cdot\mathbf{p}$ parameters, Kane parameter E_P and temperature dependent band gap E_g.

Example

```
classical{
    bulk_dispersion{
        KP8{
            from_6band_parameters = yes
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP8{ evaluate_S }

Calling sequence

`classical{ bulk_dispersion{ KP8{ evaluate_S } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

By default S (S_1, S_2 for wurtzite) $\mathbf{k}\cdot\mathbf{p}$ parameter(s) is (are) taken from database or input file. If enabled it evaluates S (S_1, S_2 for wurtzite) $\mathbf{k}\cdot\mathbf{p}$ parameter(s) from effective mass m_e ($m_{e,par}$, $m_{e,perp}$ for wurtzite), Kane parameter(s), spin-orbit coupling(s) and temperature dependent band gap.

Example

```
classical{
    bulk_dispersion{
        KP8{
            evaluate_S = yes
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP8{ rescale_S_to }

Calling sequence

`classical{ bulk_dispersion{ KP8{ rescale_S_to } } }`

Properties for Zincblende: - **usage:** optional - **type:** real number - **values:** no constraints - **default:** $r = 0.0$ - **unit:** $-$

Properties for Wurtzite: - **usage:** optional - **type:** vector of 2 real numbers: (r_1, r_2) - **values:** no constraints - **default:** $r_1 = 0.0$, $r_2 = 0.0$ - **unit:** $-$

Functionality

Sets S for zinc blende crystal structure to specified value and rescale E_P, L', N^+ in order to preserve electron's effective mass.

Sets S_1, S_2 for wurtzite crystal structure to specified values respectively and rescale E_{P1}, E_{P2}, L'_1, L'_2, N_1^+, N_2^+ in order to preserve electron's effective masses.

Examples

```
classical{
    bulk_dispersion{
        KP8{
            rescale_S_to = 1.0
        }
        path{
            name = "name"
            ...
```

(continues on next page)

(continued from previous page)

```
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

```
classical{
    bulk_dispersion{
        KP8{
            rescale_S_to = [ 1.0, 1.0 ]
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_wz{...}
}
```

KP8{ approximate_kappa }

Calling sequence

`classical{ bulk_dispersion{ KP8{ approximate_kappa } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

- *KP6{ use_Luttinger_parameters }* and *KP6{ approximate_kappa }* is not allowed if *global{ crystal_wz{ } }* is already present.

Functionality

By default, the κ for zincblende crystal structure is taken from the database or input file. If this is enabled then the solver is forced to approximate kappa through others 8-band $\mathbf{k}\cdot\mathbf{p}$ parameters, even though kappa is given in database or input file.

Example

```
classical{
    bulk_dispersion{
        KP8{
            approximate_kappa = yes
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP8{ electron_far_band }

Calling sequence

`classical{ bulk_dispersion{ KP8{ electron_far_band = ... } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** $-$

Dependencies

—

Functionality

Far-band contribution to electrons $S = 1.0 + r$. The default results in rescaling such that $S = 1.0$.

Note

It can be useful to set this value to `r = -1.0` which then corresponds to setting $S = 0.0$.

KP8{ correct_electron_gfactor }

Calling sequence

`classical{ bulk_dispersion{ KP8{ correct_electron_gfactor = ... } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = -1.0$

Dependencies

—

Functionality

When $r < 0$ then the g-factor is set to 2.
When $r = 0$ then the g-factor is computed.
When $r > 0$ then the g-factor is computed assuming energy gap equal r.
See more details in *Zeeman Term*.

KP8{ rescale_kp_everywhere }

Calling sequence

`classical{ bulk_dispersion{ KP8{ rescale_kp_everywhere } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Dependencies

—

Functionality

If set to `yes` then N, M, and P parameters are rescaled. See more details in *Zeeman Term*.

KP8{ avoid_spurious }

Calling sequence

`classical{ bulk_dispersion{ KP8{ avoid_spurious } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

—

Functionality

If set to `yes` then algorithm avoiding spurious solutions is used.

KP14{ }

Calling sequence

`classical{ bulk_dispersion{ KP14{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- *KP14{ }* and *KP30{ }* is not allowed if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, 30-band $\mathbf{k} \cdot \mathbf{p}$ model is applied to compute the bulk electronic band structure.

Example

```
classical{
    bulk_dispersion{
        KP14{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP14{ use_Luttinger_parameters }

Calling sequence

`classical{ bulk_dispersion{ KP14{ use_Luttinger_parameters } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

By default the solver uses the DKK (Dresselhaus-Kip-Kittel) parameters (L, M, N). If enabled then it uses Luttinger parameters (γ_1, γ_2, γ_3) instead.

Example

```
classical{
    bulk_dispersion{
        KP14{
            use_Luttinger_parameters = yes
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP14{ from_6band_parameters }

Calling sequence

`classical{ bulk_dispersion{ KP14{ from_6band_parameters } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

By default the 14-band $\mathbf{k} \cdot \mathbf{p}$ parameters are taken from database or input file. If enabled then it evaluates the 14-band $\mathbf{k} \cdot \mathbf{p}$ parameters from 6-band $\mathbf{k} \cdot \mathbf{p}$ parameters, Kane parameter E_P and temperature dependent band gap E_g.

Example

```
classical{
    bulk_dispersion{
        KP14{
            from_6band_parameters = yes
        }
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP14{ evaluate_S }

Calling sequence

`classical{ bulk_dispersion{ KP14{ evaluate_S } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

By default S $\mathbf{k} \cdot \mathbf{p}$ parameter(s) is (are) taken from database or input file. If enabled it evaluates S $\mathbf{k} \cdot \mathbf{p}$ parameter(s) from effective mass m_e ($m_{e,par}$, $m_{e,perp}$ for wurtzite), Kane parameter(s), spin-orbit coupling(s) and temperature dependent band gap.

Example

```
classical{
    bulk_dispersion{
        KP14{
            evaluate_S = yes
        }
        path{
            name = "name"
```

(continues on next page)

(continued from previous page)

```
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

KP30{ }

Calling sequence

`classical{ bulk_dispersion{ KP30{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *Gamma{ }*, *HH{ }*, *LH{ }*, *SO{ }*, *KP6{ }*, *KP8{ }*, *KP14{ }*, or *KP30{ }* is required if *global{ crystal_zb{ } }* is already present.
- *KP14{ }* and *KP30{ }* is not allowed if *global{ crystal_wz{ } }* is already present.

Functionality

When this group is defined, 30-band $\mathbf{k}\cdot\mathbf{p}$ model *[RideauPRB2006]* is applied to compute the bulk electronic band structure.

Example

```
classical{
    bulk_dispersion{
        KP30{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    crystal_zb{...}
}
```

---

full{ }

Calling sequence

`classical{ bulk_dispersion{ full{ } } }`

Properties

- **usage:** conditional
- **items:** no constraints

Dependencies

- At least one of *full{ }*, *path{ }*, or *lines{ }* is required.

Functionality

Calculates bulk $\mathbf{k} \cdot \mathbf{p}$ dispersion in 3D k-space. Multiple instances are allowed.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            name = "name_1"
            ...
        }
        full{
            name = "name_2"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ name }

Calling sequence

`classical{ bulk_dispersion{ full{ name } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Name of the dispersion which also defines the names of the output files.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ position{ } }

Calling sequence

`classical{ bulk_dispersion{ full{ position{ } } } }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

Specifies the point (x,y,z) in the simulation domain, where the dispersion has to be calculated.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            name = "name"
            position{...}
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ position{ x } }

Calling sequence

`classical{ bulk_dispersion{ full{ position{ x } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

x-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            name = "name"
            position{
                x = 10.5
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    simulate1D{}
}
```

---

full{ position{ y } }

Calling sequence

`classical{ bulk_dispersion{ full{ position{ y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Dependencies

- *full{ position{ y } }* is required if any *global{ simulate1D{ } }* or *global{ simulate2D{ } }* is specified in the input file.
- *full{ position{ y } }* is not allowed if *global{ simulate1D{ } }* is specified in the input file.

Functionality

y-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            name = "name"
            position{
                x = 10.5
                y = 35.0
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    simulate2D{}
}
```

---

full{ position{ z } }

Calling sequence

`classical{ bulk_dispersion{ full{ position{ z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

usage: conditional - *full{ position{ z } }* is required if *global{ simulate1D{ } }* is specified in the input file. - *full{ position{ z } }* is not allowed if any *global{ simulate1D{ } }* or *global{ simulate2D{ } }* is specified in the input file.

Functionality

z-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            name = "name"
            position{
                x = 10.5
                y = 35.0
                z = 12.3
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    simulate3D{}
}
```

---

full{ shift_holes_to_zero }

Calling sequence

`classical{ bulk_dispersion{ full{ shift_holes_to_zero } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If enabled shifts the whole dispersion, so that the energy for the Gamma point for the highest hole band is equal to 0.0 `(eV)`.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            shift_holes_to_zero = yes
```

(continues on next page)

(continued from previous page)

```
            name = "name"
            position{...}
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ kxgrid{ } }

Calling sequence

`classical{ bulk_dispersion{ full{ kxgrid{ } } } }`

Properties

- **usage:** required
- **items:** no constraints

Functionality

Specifies a grid along k_x for a 1D/2D/3D plot of the energy dispersion $E(k_x, k_y, k_z)$.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            kxgrid{...}
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ kxgrid{ line{ } } }

Calling sequence

`classical{ bulk_dispersion{ full{ kxgrid{ line{ } } } } }`

Properties

- **usage:** required
- **items:** minimum 2

Functionality

Setting options defining the grid in k-space.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            kxgrid{
                line{...}
                line{...}
                line{...}
            }
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ kxgrid{ line{ pos } } }

Calling sequence

`classical{ bulk_dispersion{ full{ kxgrid{ line{ pos } } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm^{-1}

Functionality

Position of defined k-grid spacing along kx direction.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            kxgrid{
                line{ pos =-1.0   spacing = 0.2 }
                line{ pos = 0.0   spacing = 0.2 }
```

(continues on next page)

(continued from previous page)

```
                line{ pos = 1.0   spacing = 0.2 }
            }
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ kxgrid{ line{ spacing } } }

Calling sequence

`classical{ bulk_dispersion{ full{ kxgrid{ line{ spacing } } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Functionality

k-grid spacing at defined positions

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            kxgrid{
                line{ pos =-1.0   spacing = 0.2 }
                line{ pos = 0.0   spacing = 0.2 }
                line{ pos = 1.0   spacing = 0.2 }
            }
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ kygrid{ } }

Calling sequence

`classical{ bulk_dispersion{ full{ kygrid{ } } } }`

Properties

- **usage:** required
- **items:** no constraints

Functionality

Specifies a grid along k_y for a 1D/2D/3D plot of the energy dispersion $E(k_x, k_y, k_z)$. The keywords allowed within this group are analogous to *full{ kxgrid{ } }*.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            kxgrid{...}
            kygrid{
                line{ pos =-1.0   spacing = 0.2 }
                line{ pos = 0.0   spacing = 0.2 }
                line{ pos = 1.0   spacing = 0.2 }
            }
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

full{ kzgrid{ } }

Calling sequence

`classical{ bulk_dispersion{ full{ kzgrid{ } } } }`

Properties

- **usage:** required
- **items:** no constraints

Functionality

Specifies a grid along k_z for a 1D/2D/3D plot of the energy dispersion $E(k_x, k_y, k_z)$. The keywords allowed within this group are analogous to *full{ kxgrid{ } }*.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        full{
```

(continues on next page)

(continued from previous page)

```
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            kxgrid{...}
            kygrid{...}
            kzgrid{
                line{ pos =-1.0   spacing = 0.2 }
                line{ pos = 0.0   spacing = 0.2 }
                line{ pos = 1.0   spacing = 0.2 }
            }
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    simulate3D{}
}
```

---

path{ }

Calling sequence

`classical{ bulk_dispersion{ path{ } } }`

Properties

- **usage:** conditional
- **items:** no constraints

Dependencies

- At least one of *full{ }*, *path{ }*, or *lines{ }* is required.

Functionality

Calculate bulk $\mathbf{k} \cdot \mathbf{p}$ dispersion along custom path in k-space. Multiple instances are allowed.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name_1"
            spacing = 0.2
            ...
        }
        path{
            name = "name_2"
            num_points = 10
```

(continues on next page)

(continued from previous page)

```
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ name }

Calling sequence

`classical{ bulk_dispersion{ path{ name } } }`

Properties

- **usage:** required
- **type:** character string

Dependencies

name of the dispersions which also defines the names of the output files.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ position{ } }

Calling sequence

`classical{ bulk_dispersion{ path{ position{ } } } }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

Specifies the point (x,y,z) in the simulation domain, where the dispersion has to be calculated.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{ }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ position{ x } }

Calling sequence

`classical{ bulk_dispersion{ path{ position{ x } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

x-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{
                x = 10.5
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
```

(continues on next page)

(continued from previous page)

```
    ...
    simulate1D{}
}
```

---

path{ position{ y } }

Calling sequence

`classical{ bulk_dispersion{ path{ position{ y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Dependencies

- *path{ position{ y } }* is required if any *global{ simulate1D{ } }* or *global{ simulate2D{ } }* is specified in the input file.
- *path{ position{ y } }* is not allowed if *global{ simulate1D{ } }* is specified in the input file.

Functionality

y-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{
                x = 10.5
                y = 35.0
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    simulate2D{}
}
```

---

path{ position{ z } }

Calling sequence

`classical{ bulk_dispersion{ path{ position{ z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

usage: conditional - *path{ position{ z } }* is required if *global{ simulate1D{ } }* is specified in the input file. - *path{ position{ z } }* is not allowed if any *global{ simulate1D{ } }* or *global{ simulate2D{ } }* is specified in the input file.

Functionality

z-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{
                x = 10.5
                y = 35.0
                z = 12.3
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    simulate3D{}
}
```

---

path{ shift_holes_to_zero }

Calling sequence

`classical{ bulk_dispersion{ path{ shift_holes_to_zero } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If enabled shifts the whole dispersion, so that the energy for the Gamma point for the highest hole band is equal to 0.0 `(eV)`.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ point{ } }

Calling sequence

`classical{ bulk_dispersion{ path{ point{ } } } }`

Properties

- **usage:** required
- **items:** minimum 2

Functionality

Specifies points in the path through k-space. At least two k points have to be defined. Line between two such points is called segment.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{...}
```

(continues on next page)

(continued from previous page)

```
            point{...}
            point{...}
            point{...}
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ point{ k } }

Calling sequence

`classical{ bulk_dispersion{ path{ point{ k } } } }`

Properties

- **usage:** required
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$, $r_3 = 0.0$
- **unit:** nm^{-1}

Functionality

k-point represented by vector $[k_x, k_y, k_z]$.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{...}
            point{ k = [-0.1,-0.1,-0.1 ] }
            point{ k = [ 0.0, 0.0, 0.0 ] }
            point{ k = [ 0.1, 0.0, 0.0 ] }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ spacing }

Calling sequence

`classical{ bulk_dispersion{ path{ spacing } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Dependencies

- Exactly one of *path{ spacing }* or *path{ num_points }* required.

Functionality

It specifies approximate spacing for intermediate points in the path segments.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{...}
            point{ k = [-0.1,-0.1,-0.1 ] }
            point{ k = [ 0.0, 0.0, 0.0 ] }
            point{ k = [ 0.1, 0.0, 0.0 ] }
            spacing = 0.2
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ num_points }

Calling sequence

`classical{ bulk_dispersion{ path{ num_points } } }`

Properties

- **usage:** conditional
- **type:** integer
- **values:** $z \geq 2$

Dependencies

- Exactly one of *path{ spacing }* or *path{ num_points }* required.

Functionality

It specifies number of points (intermediate + two corner-points) for each single path segment.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        path{
            name = "name"
            position{...}
            point{ k = [-0.1,-0.1,-0.1 ] }
            point{ k = [ 0.0, 0.0, 0.0 ] }
            point{ k = [ 0.1, 0.0, 0.0 ] }
            num_points = 20
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ }

Calling sequence

`classical{ bulk_dispersion{ lines{ } } }`

Properties

- **usage:** conditional
- **items:** no constraints

Dependencies

- At least one of *full{ }*, *path{ }*, or *lines{ }* is required.

Functionality

Calculates dispersions along some predefined paths of high symmetry in k-space, e.g. [100], [110], [111] and their equivalents (13 in total).

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name_1"
            ...
```

(continues on next page)

(continued from previous page)

```
        }
        lines{
            name = "name_2"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ name }

Calling sequence

`classical{ bulk_dispersion{ lines{ name } } }`

Properties

- **usage:** required
- **type:** character string

Dependencies

Name of the dispersions which also defines the names of the output files.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ position{ } }

Calling sequence

`classical{ bulk_dispersion{ lines{ position{ } } } }`

Properties

- **usage:** required
- **items:** exactly 1

Functionality

Specifies the point (x,y,z) in the simulation domain, where the dispersion has to be calculated.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name"
            position{ }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ position{ x } }

Calling sequence

`classical{ bulk_dispersion{ lines{ position{ x } } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

x-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name"
            position{
                x = 10.5
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
```

(continues on next page)

(continued from previous page)

```
    HH{}
}

global{
    ...
    simulate1D{}
}
```

---

lines{ position{ y } }

Calling sequence

`classical{ bulk_dispersion{ lines{ position{ y } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Dependencies

- *lines{ position{ y } }* is required if any *global{ simulate1D{ } }* or *global{ simulate2D{ } }* is specified in the input file.
- *lines{ position{ y } }* is not allowed if *global{ simulate1D{ } }* is specified in the input file.

Functionality

y-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name"
            position{
                x = 10.5
                y = 35.0
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
```

(continues on next page)

(continued from previous page)

```
    ...
    simulate2D{}
}
```

---

lines{ position{ z } }

Calling sequence

`classical{ bulk_dispersion{ lines{ position{ z } } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

usage: conditional - *lines{ position{ z } }* is required if *global{ simulate1D{ } }* is specified in the input file. - *lines{ position{ z } }* is not allowed if any *global{ simulate1D{ } }* or *global{ simulate2D{ } }* is specified in the input file.

Functionality

z-coordinate of interest

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name"
            position{
                x = 10.5
                y = 35.0
                z = 12.3
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
    ...
    simulate3D{}
}
```

---

lines{ shift_holes_to_zero }

Calling sequence

`classical{ bulk_dispersion{ lines{ shift_holes_to_zero } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If enabled shifts the whole dispersion, so that the energy for the Gamma point for the highest hole band is equal to 0.0 `(eV)`.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            shift_holes_to_zero = yes
            name = "name"
            position{...}
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ k_max }

Calling sequence

`classical{ bulk_dispersion{ lines{ k_max } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Functionality

Specifies a maximum absolute value (radius) for the k-vector in nm^{-1}

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name"
            position{...}
            k_max = 1.0
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ spacing }

Calling sequence

`classical{ bulk_dispersion{ lines{ spacing } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Functionality

Specifies approximate spacing for intermediate points in the path segments in nm^{-1}.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
            name = "name"
            position{...}
            k_max = 1.0
            spacing = 0.2
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

output_bulk_dispersions{ }

Calling sequence

`classical{ bulk_dispersion{ output_bulk_dispersions{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *output_bulk_dispersions{ }*, *output_masses{ }*, or *output_inverse_masses{ }* is required.

Functionality

Outputs all defined bulk $\mathbf{k} \cdot \mathbf{p}$ dispersions.

Example

```
classical{
    bulk_dispersion{
        output_bulk_dispersions{}

        KP8{}
        path{...}
    }
    Gamma{}
    HH{}
}
```

---

output_masses{ }

Calling sequence

`classical{ bulk_dispersion{ output_masses{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *output_bulk_dispersions{ }*, *output_masses{ }*, or *output_inverse_masses{ }* is required.

Functionality

Outputs effective masses calculated from the dispersions.

Example

```
classical{
    bulk_dispersion{
        output_masses{}

        KP8{}
```

(continues on next page)

(continued from previous page)

```
        path{...}
    }
    Gamma{}
    HH{}
}
```

---

output_inverse_masses{ }

Calling sequence

`classical{ bulk_dispersion{ output_inverse_masses{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- At least one of *output_bulk_dispersions{ }*, *output_masses{ }*, or *output_inverse_masses{ }* is required.

Functionality

Outputs inverse of effective masses calculated from the dispersions.

Example

```
classical{
    bulk_dispersion{
        output_inverse_masses{}

        KP8{}
        path{...}
    }
    Gamma{}
    HH{}
}
```

---

output_k_vectors{ }

Calling sequence

`classical{ bulk_dispersion{ output_k_vectors{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs k-vectors for which the dispersions are computed.

Example

```
classical{
    bulk_dispersion{
        output_k_vectors{}

        KP8{}
        path{...}
    }
    Gamma{}
    HH{}
}
```

Last update: 27/05/2025

7.10.13 output_bandgap{ }

Calling sequence

`classical{ output_bandgap{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output band gaps for Gamma, L, X (or Delta) bands with reference to the highest valence band edge. Additionally, the difference between the lowest conduction band and the highest valence band edges is written out: MIN(Gamma, L, X (or Delta)) - MAX(hh, lh, so) [eV]

Example

```
classical{
    output_bandgap{}

    Gamma{}
    HH{}
}
```

Nested keywords

- *averaged*

---

averaged

Calling sequence

`classical{ output_bandgap{ averaged } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes`, then for each grid point the band gaps will be averaged between neighboring material grid points. If set to `no`, then abrupt discontinuities at interfaces introducing two points, four points, and eight points for each grid point for 1D, 2D, and 3D simulations, respectively.

Example

```
classical{
    output_bandgap{
        averaged = yes
    }

    Gamma{}
    HH{}
}
```

7.10.14 output_bandedges{ }

Calling sequence

`classical{ output_bandedges{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output selected band edges (extrema of selected bands of bulk materials) and Fermi levels in one file named *bandedges.dat*.

Example

```
classical{
    output_bandedges{}

    Gamma{}
    HH{}
}
```

Nested keywords

- *profiles*
- *averaged*

profiles

Calling sequence

`classical{ output_bandedges{ profiles } }`

Properties

- **usage:** optional
- **type:** enumerator
- **values:** `Gamma`; `HH`; `LH`; `SO`; `X`; `Delta`; `L`; `electron_fermi_level`; `hole_fermi_level`
- **default:** `Gamma ``; ``HH ``; ``LH ``; ``SO ``; ``X ``; ``Delta ``; ``L ``; ``electron_fermi_level ``; ``hole_fermi_level`

Functionality

Enumerate band edges and quasi-Fermi levels for output. If this keyword is not defined then all profiles are written to the ouptut.

Examples

```
classical{
    output_bandedges{
        profiles = "Gamma"
    }

    Gamma{}
    HH{}
}
```

```
classical{
    output_bandedges{
        profiles = "Gamma HH electron_fermi_level"
    }

    Gamma{}
    HH{}
}
```

---

averaged

Calling sequence

`classical{ output_bandedges{ averaged } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes`, then for each grid point the band gaps will be averaged between neighboring material grid points. If set to `no`, then abrupt discontinuities at interfaces introducing two points, four points, and eight points for each grid point for 1D, 2D, and 3D simulations, respectively.

Examples

```
classical{
    output_bandedges{
        averaged = yes
    }

    Gamma{}
    HH{}
}
```

7.10.15 output_carrier_densities{ }

Calling sequence

`classical{ output_carrier_densities{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output electron and hole densities into files *density_electron.dat* and *density_hole.dat* expressed in units dependent on dimensionality of the simulation.

- In 1D simulation the unit is $1/\mathrm{cm}^2$
- In 2D simulation the unit is $1/\mathrm{cm}$
- In 3D simulation the unit is 1

Example

```
classical{
    output_carrier_densities{}

    Gamma{}
    HH{}
}
```

7.10.16 output_band_densities{ }

Calling sequence

`classical{ output_band_densities{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

The densities (outside the quantum regions) for the individual bands are output if this group is defined.

Example

```
classical{
    output_band_densities{}

    Gamma{}
    HH{}
}
```

7.10.17 output_ionized_dopant_densities{ }

Calling sequence

`classical{ output_ionized_dopant_densities{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output densities of ionized acceptors and donors to *density_acceptor_ionized.dat* and *density_donor_ionized.dat*, respectively. The densities are expressed in $10^{18}/\mathrm{cm}^3$.

Example

```
classical{
    output_ionized_dopant_densities{}

    Gamma{}
    HH{}
}
```

7.10.18 output_carrier_densities_matgrid{ }

Calling sequence

`classical{ output_carrier_densities_matgrid{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output electron and hole densities defined on the material grid into files *density_electron.dat* and *density_hole.dat* expressed in units dependent on dimensionality of the simulation. These are values entering the drift-diffusion model.

Hint

They typically look better than *output_carrier_densities{ }* for data presentation.

- In 1D simulation the unit is $1/\mathrm{cm}^2$
- In 2D simulation the unit is $1/\mathrm{cm}$
- In 3D simulation the unit is 1

Example

```
classical{
    output_carrier_densities_matgrid{}

    Gamma{}
    HH{}
}
```

---

output_carrier_densities_matgrid{ boxes }

Calling sequence

`classical{ output_carrier_densities_matgrid{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
classical{
    output_carrier_densities_matgrid{
        boxes = yes
    }

    Gamma{}
    HH{}
}
```

7.10.19 output_band_densities_matgrid{ }

Calling sequence

`classical{ output_band_densities_matgrid{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

The densities (outside the quantum regions) for the individual bands defined on the material grid are output if this group is defined. These are values corresponding drift-diffusion model.

> **Hint**
>
> They typically look better than *output_band_densities{ }* for data presentation.

Example

```
classical{
    output_band_densities_matgrid{}

    Gamma{}
    HH{}
}
```

---

output_band_densities_matgrid{ boxes }

Calling sequence

`classical{ output_band_densities_matgrid{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
classical{
    output_band_densities_matgrid{
        boxes = yes
    }

    Gamma{}
    HH{}
}
```

7.10.20 output_ionized_dopant_densities_matgrid{ }

Calling sequence

`classical{ output_ionized_dopant_densities_matgrid{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output densities of ionized acceptors and donors defined on the material grid to *density_acceptor_ionized.dat* and *density_donor_ionized.dat*, respectively. The densities are expressed in $10^{18}/\mathrm{cm}^3$.

Example

```
classical{
    output_ionized_dopant_densities_matgrid{}

    Gamma{}
    HH{}
}
```

---

output_ionized_dopant_densities_matgrid{ boxes }

Calling sequence

`classical{ output_ionized_dopant_densities_matgrid{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
classical{
    output_ionized_dopant_densities_matgrid{
        boxes = yes
    }

    Gamma{}
    HH{}
}
```

7.10.21 output_intrinsic_density{ }

Calling sequence

`classical{ output_intrinsic_density{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Output intrinsic density expressed in $1/\mathrm{cm}^3$.

Example

```
classical{
    output_intrinsic_density{}

    Gamma{}
    HH{}
}
```

Nested keywords

- *output_intrinsic_density{ boxes }*

---

output_intrinsic_density{ boxes }

Calling sequence

`classical{ output_intrinsic_density{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
classical{
    output_intrinsic_density{
        boxes = yes
    }

    Gamma{}
    HH{}
}
```

Last update: 02/04/2025

```
classical{
    # conduction bands
    Gamma{
        output_bandedge{
            averaged = yes
        }
    }
    L{      output_bandedge{ ... } }
    X{      output_bandedge{ ... } }     # or Delta{ output_bandedge{ ... } }

    # valence bands
    HH{     output_bandedge{ ... } }
    LH{     output_bandedge{ ... } }
    SO{     output_bandedge{ ... } }

    #Further output definitions
    output_bandedges{
        profiles = "Gamma HH LH"
        averaged =  no
    }

    output_bandgap{
        averaged =  no
    }

    output_carrier_densities{}

    output_ionized_dopant_densities{}

    output_intrinsic_density{
        boxes = yes
    }

    energy_distribution{
        min = -5.0
        max =  5.0
        energy_resolution = 0.1
        only_density_quantum_regions = yes
    }

    energy_resolved_density{
        min = -5.0
        max =  5.0
        energy_resolution = 0.1
        only_density_quantum_regions = yes
        output_energy_resolved_densities{}
    }

    output_photon_density = yes
    output_power_density  = yes

    # Carrier statistics for classical densities
    carrier_statistics = fermi_dirac

    bulk_dispersion{
        KP6{}
```

(continues on next page)

(continued from previous page)

```
        path{
            name = "from_Gamma_to_L"
            position{
                x = 5.5
                y = 10.0
                z = -1.1
            }
            shift_holes_to_zero = yes
            point{
                k = [1.0, 0.0, 0.0]
            }
            ...

            spacing             = 0.5
            num_points          = 10
        }

        lines{
            name = "lines"
            position{
                x = 5.5
                y = 10.0
                z = -1.1
            }
            shift_holes_to_zero = yes
            spacing             = 0.5
            k_max               = 1.0
        }

        full{
            name = "3D"
            position{
                x = 5.5
                y = 10.0
                z = -1.1
            }
            shift_holes_to_zero = yes
            kxgrid{
                line{
                pos     = -1
                spacing = 0.02
                }
                line{
                pos     = 1
                spacing = 0.02
                }
                ...
            }
            kygrid{
                ...
            }
            kzgrid{
                ...
            }
        }
```

(continues on next page)

(continued from previous page)

```
        output_bulk_dispersions{}
        output_masses{}

    } # end: bulk_dispersion{}
}
```

7.11 strain{ }

Calling sequence

`strain{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

The group controls selection of strain models and related outputs.

> **Attention**
>
> Definition of this group does not result in running any strain models. To do so, use it together with *strain{ }* from within *run{ }*.

Examples

```
strain{
    pseudomorphic_strain{}
}
```

```
strain{
    minimized_strain{}
}
```

Nested keywords

7.11.1 debuglevel

Calling sequence

`strain{ debuglevel }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq -1$
- **unit:** –
- **default:** $z = 2$

Functionality

The higher this integer number, the more information on the numerical solver is printed to the screen output

Example

```
strain{
    debuglevel = 3
    pseudomorphic_strain{}
}
```

7.11.2 no_strain{ }

Calling sequence

`strain{ no_strain{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- Exactly one out of *no_strain{ }*, *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* must be defined.
- *linear_solver{ }* and *residual_strain* are not allowed if this group is defined.

Functionality

Strain is not taken into account.

Example

```
strain{
    no_strain{}
}
```

7.11.3 pseudomorphic_strain{ }

Calling sequence

`strain{ pseudomorphic_strain{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- *linear_solver{ }* is not allowed if this group is defined.
- Exactly one out of *no_strain{ }*, *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* must be defined.
- One of *pseudomorphic_strain{ }* and *minimized_strain{ }* must be present to allow using *output_distortion_tensor{ }*, *output_stress_tensor{ }*, *output_force_density{ }*, *output_elastic_energy_density{ }*, *output_lattice_constants{ }*, *output_elastic_constants{ }*.

- One of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* must be present to allow using *output_hydrostatic_strain{ }*, *output_strain_tensor{ }*, *output_piezo_constants{ }*, *output_second_order_piezo_constants{ }*.

Functionality

Homogeneous strain for 1D layer structures (analytical calculation). This feature also works in 2D or 3D but the user must be sure that the model makes sense from a physical point of view, i.e., the 2D/3D structure should consist of different layers along the growth direction whereas the layers must be homogenous along the two perpendicular directions.

Example

```
strain{
    pseudomorphic_strain{}
}
```

7.11.4 minimized_strain{ }

Calling sequence

`strain{ minimized_strain{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is required if *linear_solver{ }* is defined.
- Exactly one out of *no_strain{ }*, *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* must be defined.
- This keyword must be present to allow using *output_displacement{ }*.
- One of *pseudomorphic_strain{ }* and *minimized_strain{ }* must be present to allow using *output_distortion_tensor{ }*, *output_stress_tensor{ }*, *output_force_density{ }*, *output_elastic_energy_density{ }*, *output_lattice_constants{ }*, *output_elastic_constants{ }*.
- One of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* must be present to allow using *output_hydrostatic_strain{ }*, *output_strain_tensor{ }*, *output_piezo_constants{ }*, *output_second_order_piezo_constants{ }*.

Functionality

Minimization of the elastic energy for 2D and 3D geometries (numerical calculation). It can also be used for 1D simulations. In this case, the results will be equivalent to the analytical model pseudomorphic_strain{ }.

Example

```
strain{
    minimized_strain{}
}
```

7.11.5 growth_direction

Calling sequence

`strain{ growth_direction }`

Properties

- **usage:** conditional
- **type:** vector of 3 integers: (z_1, z_2, z_3)
- **values:** no constraints
- **default:** $r_1 = 1.0$, $r_2 = 0.0$, $r_3 = 0.0$
- **unit:** $-$

Dependencies

- This keyword is not allowed if *simulate1D{ }* is defined.

Functionality

Defines a normal vector to a substrate surface, corresponding to the growth direction, for a pseudomorphic strain model. It is defined in crystal coordinate system. It can be specified in a 2D and 3D simulations, but not in a 1D simulation as the crystal direction along the x-axis is always chosen in this case.

Example

```
strain{
    growth_direction = [1, 1, 0]
    pseudomorphic_strain{}
}

global{
    simulate2D{}
    ...
}
```

7.11.6 residual_strain

Calling sequence

`strain{ residual_strain }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** $-1.0 \leq r \leq 1.0$
- **unit:** $-$

Dependencies

- This group is not allowed if any of *no_strain{ }* and *import_strain{ }* is defined.

Functionality

Residuals strain in the substrate η scales lattice parameter of the substrate (only for the purpose of strain computation) according to the formula $a_{\eta,s} = (1+\eta) \cdot a_{0,s}$, where $a_{0,s}$ is the (unstrained) lattice parameter of the substrate and $a_{\eta,s}$ the modified (strained) lattice parameter of the substrate. The latter one represents the substrate during evaluation of the strain tensor.

Example

```
strain{
    residual_strain = 0.2
    pseudomorphic_strain{}
}
```

7.11.7 linear_solver{ }

Calling sequence

`strain{ linear_solver{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- *minimized_strain{ }* must be defined.
- This group is not allowed if any of *no_strain{ }*, *pseudomorphic_strain{ }*, and *import_strain{ }* is defined.

Functionality

—

Example

```
strain{
    minimized_strain{}
    linear_solver{}
}
```

Nested keywords

- *iterations*
- *abs_accuracy*
- *rel_accuracy*
- *use_cscg*
- *force_diagonal_preconditioner*

iterations

Calling sequence

`strain{ linear_solver{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 10000$
- **unit:** $-$

Functionality

Number of iterations for linear equation solver in strain algorithm

Example

```
strain{
    minimized_strain{}
    linear_solver{
        iterations = 50000
    }
}
```

---

abs_accuracy

Calling sequence

`strain{ linear_solver{ abs_accuracy } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 1e - 8$
- **unit:** GP for 1D; $\mathrm{GP}\,\mathrm{nm}$ for 2D; $\mathrm{GP}\,\mathrm{nm}^2$ for 3D

Functionality

—

Example

```
strain{
    minimized_strain{}
    linear_solver{
        abs_accuracy = 1e-9
    }
}
```

---

rel_accuracy

Calling sequence

`strain{ linear_solver{ rel_accuracy } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.0 \leq r \leq 10^{-6}$
- **default:** $r = 1e - 12$
- **unit:** $-$

Functionality

—

Example

```
strain{
    minimized_strain{}
    linear_solver{
        rel_accuracy = 1e-10
    }
}
```

---

use_cscg

Calling sequence

`strain{ linear_solver{ use_cscg } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Composite step conjugate gradient solver (try this one if standard solver fails to converge)

Example

```
strain{
    minimized_strain{}
    linear_solver{
        use_cscg = yes
    }
}
```

force_diagonal_preconditioner

Calling sequence

`strain{ linear_solver{ force_diagonal_preconditioner } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    minimized_strain{}
    linear_solver{
        force_diagonal_preconditioner = yes
    }
}
```

Last update: 02/04/2025

7.11.8 import_strain{ }

Calling sequence

`strain{ import_strain{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- *import{ }* must be defined.
- *linear_solver{ }* and *residual_strain* are not allowed if this group is defined.
- Exactly one out of *no_strain{ }*, *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* must be defined.
- One of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* must be present to allow using *output_hydrostatic_strain{ }*, *output_strain_tensor{ }*, *output_piezo_constants{ }*, *output_second_order_piezo_constants{ }*.

Functionality

Controls importing strain tensor elements to the simulation from an external file.

Example

```
strain{
    import_strain{...}
}

import{...}
```

Nested keywords

- *import_from*
- *coordinate_system*

---

import_from

Calling sequence

`strain{ import_strain{ import_from } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Reference to imported data in import{ }.

The data being imported must have exactly 6 components. The expected order of strain tensor components is: ε_{xx} ε_{yy} ε_{zz} ε_{xy} ε_{xz} ε_{yz}

Example

```
strain{
    import_strain{
        import_from = "strain_import"
    }
}

import{
    file{
        name = "strain_import"
        ...
    }
}
```

---

coordinate_system

Calling sequence

`strain{ import_strain{ coordinate_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `crystal` or `simulation`
- **default:** `simulation`

Functionality

The imported strain tensor is with respect to the simulation or crystal coordinate system (optional parameter).

Example

```
strain{
    import_strain{
        import_from = "strain_import"
        coordinate_system = "simulation"
    }
}

import{
    file{
        name = "strain_import"
        ...
    }
}
```

7.11.9 piezo_density

Calling sequence

`strain{ piezo_density }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Calculate piezoelectric charge density and take it into account while solving the Poisson equation.

If no strain is solved, this flag is ignored.

Example

```
strain{
    piezo_density = no
    pseudomorphic_strain{}
}
```

7.11.10 second_order_piezo

Calling sequence

`strain{ second_order_piezo }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Include 2nd order piezoelectric coefficients in the calculation.

> **Warning**
>
> Only "standard growth directions" are supported for wurtzite.

Example

```
strain{
    second_order_piezo = yes
    pseudomorphic_strain{}
}
```

7.11.11 pyro_density

Calling sequence

`strain{ pyro_density }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Dependencies

- This keyword is allowed only if *crystal_wz{ }* is defined.

Functionality

Calculate pyroelectric charge density and take it into account while solving the Poisson equation.

If material system is not wurtzite, this flag is ignored. The pyroelectric charge density due to spontaneous polarization applies to wurtzite only. In order to obtain pyroelectric charges, it is not necessary to calculate strain. Pyroelectric charges are only present in wurtzite materials but not in zincblende .

Example

```
strain{
    pyro_density = no
    pseudomorphic_strain{}
}

global{
    crystal_wz{...}
    ...
}
```

7.11.12 output_hydrostatic_strain{ }

Calling sequence

`strain{ output_hydrostatic_strain{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* is defined.

Functionality

prints out the hydrostatic strain, i.e. the trace of the strain tensor $\mathrm{Tr}[\varepsilon_{ij}] = \varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}$ [dimensionless]

Example

```
strain{
    output_hydrostatic_strain{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`strain{ output_hydrostatic_strain{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** yes or no
- **default:** no

Functionality

—

Example

```
strain{
    output_hydrostatic_strain{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.13 output_strain_tensor{ }

Calling sequence

`strain{ output_strain_tensor{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* is defined.

Functionality

output (symmetric) strain tensor : $\varepsilon_{ij} = (u_{ij} + u_{ji})/2$ [dimensionless]

Example

```
strain{
    output_strain_tensor{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *crystal_system*
- *simulation_system*
- *boxes*

---

crystal_system

Calling sequence

`strain{ output_strain_tensor{ crystal_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

output strain tensor in crystal coordinate system

Example

```
strain{
    output_strain_tensor{
        crystal_system = yes
    }
    pseudomorphic_strain{}
}
```

---

simulation_system

Calling sequence

`strain{ output_strain_tensor{ simulation_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

output strain tensor in simulation coordinate system (useful if simulation coordinate system differs from crystal coordinate system)

Note

The ordering of the strain tensor components is: ε_{xx}, ε_{yy}, ε_{zz}, ε_{xy}, ε_{xz}, ε_{yz}

Example

```
strain{
    output_strain_tensor{
        simulation_system = no
    }
    pseudomorphic_strain{}
}
```

---

boxes

Calling sequence

`strain{ output_strain_tensor{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
strain{
    output_strain_tensor{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.14 output_distortion_tensor{ }

Calling sequence

`strain{ output_distortion_tensor{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }* and *minimized_strain{ }* is defined.

Functionality

output distortion tensor u_{ij} (which can be nonsymmetric for certain growth directions) u_{xx} u_{yy} u_{zz} u_{xy} u_{yx} u_{xz} u_{zx} u_{yz} u_{zy} [dimensionless]

Example

```
strain{
    output_distortion_tensor{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *crystal_system*
- *simulation_system*
- *boxes*

---

crystal_system

Calling sequence

`strain{ output_distortion_tensor{ crystal_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

output distortion tensor in crystal coordinate system

Example

```
strain{
    output_distortion_tensor{
        crystal_system = yes
    }
    pseudomorphic_strain{}
}
```

---

simulation_system

Calling sequence

`strain{ output_distortion_tensor{ simulation_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

output distortion tensor in crystal coordinate system

Example

```
strain{
    output_distortion_tensor{
        simulation_system = no
    }
    pseudomorphic_strain{}
}
```

---

boxes

Calling sequence

`strain{ output_distortion_tensor{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_distortion_tensor{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.15 output_stress_tensor{ }

Calling sequence

`strain{ output_stress_tensor{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }* and *minimized_strain{ }* is defined.

Functionality

output (symmetric) stress tensor : $\sigma_{ij} = C_{ijkl}\,\varepsilon_{kl}$ [GPa]

Example

```
strain{
    output_stress_tensor{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *crystal_system*
- *simulation_system*
- *boxes*

---

crystal_system

Calling sequence

`strain{ output_stress_tensor{ crystal_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

output stress tensor in crystal coordinate system

Example

```
strain{
    output_stress_tensor{
        crystal_system = yes
    }
    pseudomorphic_strain{}
}
```

---

simulation_system

Calling sequence

`strain{ output_stress_tensor{ simulation_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

output stress tensor in simulation coordinate system (useful if simulation coordinate system differs from crystal coordinate system)

> **Note**
>
> The ordering of the stress tensor components is: σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{xz}, σ_{yz}

Example

```
strain{
    output_stress_tensor{
        simulation_system = no
    }
    pseudomorphic_strain{}
}
```

---

boxes

Calling sequence

`strain{ output_stress_tensor{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_stress_tensor{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.16 output_displacement{ }

Calling sequence

`strain{ output_displacement{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if *minimized_strain{ }* is defined.

Functionality

output displacement vector [nm]

Example

```
strain{
    output_displacement{}
    minimized_strain{}
}
```

Nested keywords

- *crystal_system*
- *simulation_system*

---

crystal_system

Calling sequence

`strain{ output_displacement{ crystal_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

output displacement vector in crystal coordinate system

Example

```
strain{
    output_displacement{
        crystal_system = yes
    }
    minimized_strain{}
}
```

---

simulation_system

Calling sequence

`strain{ output_displacement{ simulation_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

output displacement vector in simulation coordinate system

Example

```
strain{
    output_displacement{
        simulation_system = no
    }
    minimized_strain{}
}
```

7.11.17 output_force_density{ }

Calling sequence

`strain{ output_force_density{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }* and *minimized_strain{ }* is defined.

Functionality

output force density vector field f_i [nN/nm^3] (at moment output may be not fully correct; not tested sufficiently)

Example

```
strain{
    output_force_density{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *crystal_system*
- *simulation_system*

crystal_system

Calling sequence

`strain{ output_force_density{ crystal_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

output force density vector field in crystal coordinate system

Example

```
strain{
    output_force_density{
        crystal_system = yes
    }
    pseudomorphic_strain{}
}
```

---

simulation_system

Calling sequence

`strain{ output_force_density{ simulation_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

output force density vector field in simulation coordinate system

Example

```
strain{
    output_force_density{
        simulation_system = no
    }
    pseudomorphic_strain{}
}
```

7.11.18 output_elastic_energy_density{ }

Calling sequence

`strain{ output_elastic_energy_density{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }* and *minimized_strain{ }* is defined.

Functionality

output elastic energy density ($\frac{1}{2}\ C_{ijkl}\ \varepsilon_{ij}\ \varepsilon_{kl}$) [eV/nm^3] The integrated elastic energy is printed out in log file.

Example

```
strain{
    output_elastic_energy_density{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`strain{ output_elastic_energy_density{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** yes or no
- **default:** no

Functionality

—

Example

```
strain{
    output_elastic_energy_density{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.19 output_polarization_charge{ }

Calling sequence

`strain{ output_polarization_charge{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs sum of piezoelectric and pyroelectric polarization charge densities (ρ_{pz} and ρ_{py}) expressed in $10^{18}/\mathrm{cm}^3$to a file *Strain\polarization_charge_density_ptotal.dat*.

$$\rho_{\mathrm{pol}} = \rho_{\mathrm{pz}} + \rho_{\mathrm{py}}$$

Example

```
strain{
    output_polarization_charge{}
    pseudomorphic_strain{}
}
```

7.11.20 output_polarization_charge_components{ }

Calling sequence

`strain{ output_polarization_charge_components{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs piezoelectric and pyroelectric charge densities (ρ_{pz} and ρ_{py}) expressed in $10^{18}/\mathrm{cm}^3$to the files *Strain\polarization_charge_density_piezo.dat* and *Strain\polarization_charge_density_pyro.dat*, respectively.

Pyroelectric charges due to spontaneous polarization apply to wurtzite only. It applies to wurtzite only and is independent of strain. It is present due to spontaneous polarization.

Piezoelectric charges can be calculated for both zinc blende and wurtzite in case the strain was calculated. For diamond-like crystal structures, that have an inversion center such a Si or Ge, piezoelectric charges do not exist.

Example

```
strain{
    output_polarization_charge_components{}
    pseudomorphic_strain{}
}
```

7.11.21 output_polarization_vector{ }

Calling sequence

`strain{ output_polarization_vector{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* is defined.

Functionality

Outputs the sum of piezo and pyroelectric polarization vectors expressed in [C/cm^2] to files *Strain\polarization_vector_total_simulation.dat* and *Strain\polarization_vector_total_crystal.dat*, depending on selected otpions.

Example

```
strain{
    output_polarization_vector{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *crystal_system*
- *simulation_system*
- *boxes*

---

crystal_system

Calling sequence

`strain{ output_polarization_vector{ crystal_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Outputs polarization vector in crystal coordinate system to a file *Strain\polarization_vector_total_crystal.dat*.

Example

```
strain{
    output_polarization_vector{
        crystal_system = yes
    }
```

(continues on next page)

(continued from previous page)

```
    pseudomorphic_strain{}
}
```

---

simulation_system

Calling sequence

`strain{ output_polarization_vector{ simulation_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Outputs polarization vector in simulation coordinate system to a file *Strain\polarization_vector_total_simulation.dat*.

Example

```
strain{
    output_polarization_vector{
        simulation_system = no
    }
    pseudomorphic_strain{}
}
```

---

boxes

Calling sequence

`strain{ output_polarization_vector{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_polarization_vector{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.22 output_polarization_vector_components{ }

Calling sequence

`strain{ output_polarization_vector_components{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Functionality

Outputs piezo and pyroelectric polarization vectors expressed in [C/cm^2] separately to files *Strain\polarization_vector_piezo_simulation.dat* and *Strain\polarization_vector_pyro_simulation.dat* or *Strain\polarization_vector_piezo_crystal.dat* and *Strain\polarization_vector_pyro_crystal.dat*, depending on selected otpions.

Example

```
strain{
    output_polarization_vector_components{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *crystal_system*
- *simulation_system*
- *boxes*

---

crystal_system

Calling sequence

`strain{ output_polarization_vector_components{ crystal_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

output polarization vector in crystal coordinate system to files *Strain\polarization_vector_piezo_crystal.dat* and *Strain\polarization_vector_pyro_crystal.dat*.

Example

```
strain{
    output_polarization_vector_components{
        crystal_system = yes
    }
    pseudomorphic_strain{}
}
```

---

simulation_system

Calling sequence

`strain{ output_polarization_vector_components{ simulation_system } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

output polarization vector in simulation coordinate system to files *Strain\polarization_vector_piezo_simulation.dat* and *Strain\polarization_vector_pyro_simulation.dat*.

Example

```
strain{
    output_polarization_vector_components{
        simulation_system = no
    }
    pseudomorphic_strain{}
}
```

---

boxes

Calling sequence

`strain{ output_polarization_vector_components{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_polarization_vector_components{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.23 output_lattice_constants{ }

Calling sequence

`strain{ output_lattice_constants{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }* and *minimized_strain{ }* is defined.

Functionality

Output lattice constants to a file …*\Structure\lattice_constants.dat*

Example

```
strain{
    output_lattice_constants{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`strain{ output_lattice_constants{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_lattice_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.24 output_elastic_constants{ }

Calling sequence

`strain{ output_elastic_constants{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }* and *minimized_strain{ }* is defined.

Functionality

Output elastic constants.

Example

```
strain{
    output_elastic_constants{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`strain{ output_elastic_constants{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_elastic_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.25 output_piezo_constants{ }

Calling sequence

`strain{ output_piezo_constants{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* is defined.

Functionality

Output piezoelectric constants.

Example

```
strain{
    output_piezo_constants{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`strain{ output_piezo_constants{ boxes } }`

Properties

- **usage:** optional
- **type:** choice

- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_piezo_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.26 output_second_order_piezo_constants{ }

Calling sequence

`strain{ output_second_order_piezo_constants{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is allowed if one of *pseudomorphic_strain{ }*, *minimized_strain{ }*, and *import_strain{ }* is defined.

Functionality

Output 2nd order piezoelectric constants.

Example

```
strain{
    output_second_order_piezo_constants{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`strain{ output_second_order_piezo_constants{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_second_order_piezo_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.11.27 output_pyro_constants{ }

Calling sequence

`strain{ output_pyro_constants{ } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

- This keyword is not allowed if *crystal_zb{ }* is defined.

Functionality

Output pyroelectric constants, i.e. spontaneous polarization constants.

Example

```
strain{
    output_pyro_constants{}
    pseudomorphic_strain{}
}
```

- *boxes*

---

boxes

Calling sequence

`strain{ output_pyro_constants{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

Example

```
strain{
    output_pyro_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

7.12 poisson{ }

Calling sequence

`poisson{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- Exactly one of *import_potential{ }*, *electric_field{ }*, *between_fermi_levels{ }*, *charge_neutral{ }*, and *zero_charge{ }* must be specified.

Functionality

Presence of this group is triggering initialization of the Poisson equation. Calling it is required if Poisson equation is to be solved during a simulation. It gathers keywords controlling initialization of the electrostatic potential, numerical parameters of solvers, and related outputs.

Examples

```
poisson{
    between_fermi_levels{}
}
```

```
poisson{
    electric_field{...}
}
```

Nested keywords

- *debuglevel*
- *import_potential{ }*
- *import_potential{ import_from }*
- *import_potential{ component_number }*
- *electric_field{ }*
- *electric_field{ direction }*
- *electric_field{ strength }*
- *electric_field{ reference_potential }*
- *between_fermi_levels{ }*
- *charge_neutral{ }*
- *zero_charge{ }*
- *newton_solver{ }*
- *newton_solver{ iterations }*
- *newton_solver{ search_steps }*
- *newton_solver{ residual }*
- *newton_solver{ gradient_shift }*
- *linear_solver{ }*
- *linear_solver{ iterations }*
- *linear_solver{ abs_accuracy }*
- *linear_solver{ rel_accuracy }*
- *linear_solver{ dkr_value }*
- *linear_solver{ use_cscg }*
- *linear_solver{ force_diagonal_preconditioner }*
- *linear_solver{ force_iteration }*
- *bisection{ }*
- *bisection{ delta }*
- *bisection{ residual }*
- *bisection{ iterations }*
- *bisection{ robust }*
- *output_potential{ }*
- *output_electric_field{ }*
- *output_electric_displacement{ }*
- *output_electric_polarization{ }*
- *output_dielectric_tensor{ }*
- *output_dielectric_tensor{ boxes }*

7.12.1 debuglevel

Calling sequence

`poisson{ debuglevel }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq -1$
- **default:** $z = 1$
- **unit:** $-$

Functionality

The higher this integer number, the more information on the numerical solver is printed to the screen output. Increasing the respective debuglevel to 2 or more significantly increases the volume of the diagnostic output displayed in *nextnanomat* (or a shell window). As result of the additional I/O load, particularly 1D simulations will slow down correspondingly (especially for `current{ }` and `poisson{ }`)

Example

```
poisson{
    debuglevel = 2
    between_fermi_levels{}
}
```

7.12.2 import_potential{ }

Calling sequence

`poisson{ import_potential{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The global group *import{ }* must be present.

Functionality

Import electrostatic potential from file or analytic function and use it as initial guess for solving the Poisson equation. If no Poisson equation is solved, the imported data determines the electrostatic potential that is used throughout the simulation, i.e. in this case an electrostatic potential can be read in that is fixed during the rest of the simulation and is used as input to the Schrödinger equation and for the calculation of the densities. The solution obtained from a problem solved previously using a different meshing is accepted.

Example

```
poisson{
    import_potential{...}
}

import{...}
```

7.12.3 import_potential{ import_from }

Calling sequence

`poisson{ import_potential{ import_from } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Reference to imported data in *import{ }*. The data may have more than one component (e.g. vector field).

Example

```
poisson{
    import_potential{
        import_from = "qpc_landscape"
    }
}

import{
    file{
        name = "qpc_landscape"
        ...
    }
}
```

7.12.4 import_potential{ component_number }

Calling sequence

`poisson{ import_potential{ component_number } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 1$
- **unit:** $-$

Functionality

If imported data is a vector field, one may want to specify the component.

Example

```
poisson{
    import_potential{
        import_from = "qpc_landscape"
        component_number = 2
    }
}

import{
    file{
        name = "qpc_landscape"
        ...
    }
}
```

7.12.5 electric_field{ }

Calling sequence

`poisson{ electric_field{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- *electric_field{ direction }* must be defined if *simulate2D{ }* or *simulate3D{ }* is already present.

Functionality

If `electric_field{}` is defined, this value in units of [V] is being added to the electrostatic potential.

Examples

```
poisson{
    electric_field{...}
}

global{
    simulate1D{ }
}
```

```
poisson{
    electric_field{
        direction = ...
        ...
    }
}

```

(continues on next page)

(continued from previous page)

```
global{
    simulate2D{ }
}
```

7.12.6 electric_field{ direction }

Calling sequence

`poisson{ electric_field{ direction } }`

Properties

- **usage:** optional
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **default:** $r_1 = 1.0$, $r_2 = 0.0$, $r_3 = 0.0$
- **unit:** $-$

Functionality

Orientation of electric field vector with respect to (x, y, z) simulation coordinate system. For 1D simulations, the direction can be omitted and in this case the default will be used.

Examples

```
poisson{
    electric_field{
        direction = [ -1.0, 0.0, 0.0 ]
        ...
    }
}
```

```
poisson{
    electric_field{
        direction = [ 0.0, 0.5, 0.5 ]
        ...
    }
}

global{
    simulate3D{ }
}
```

7.12.7 electric_field{ strength }

Calling sequence

`poisson{ electric_field{ strength } }`

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** $\mathrm{V/m}$

Functionality

Defines a constant electric field in the structure. If electric_field is defined, and the absolute value is larger than zero, then it is being used for the electrostatic potential calculation.

Example

```
poisson{
    electric_field{
        direction = [ -1.0, 0.0, 0.0 ]
        strength = 0.42
    }
}
```

7.12.8 electric_field{ reference_potential }

Calling sequence

`poisson{ electric_field{ reference_potential } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** V

Functionality

Note

If `poisson{ }` group is not called at all, then electric potential $\phi = 0$ is assumed everywhere.

Example

```
poisson{
    electric_field{
        direction = [ -1.0, 0.0, 0.0 ]
        strength = 0.42
        reference_potential = -1.3
    }
}
```

7.12.9 between_fermi_levels{ }

Calling sequence

`poisson{ between_fermi_levels{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

When this group is used then the average value of quasi-Fermi levels is taken as the $\phi_{i=0}$ at every non-Dirichlet point of the simulation grid. Non-Dirichlet points are the grid points in the regions of the simulation, for which Dirichlet boundary conditions (in this case for potential) are not imposed. The group **between_fermi_levels{}** is used by default if the **poisson{ }** group is not specified in the input file at all.

Example

```
poisson{
    between_fermi_levels{}
}
```

7.12.10 charge_neutral{ }

Calling sequence

`poisson{ charge_neutral{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

The **recommended** keyword for specifying $\phi_{i=0}$ is **charge_neutral{}**. By using it, $\phi_{i=0}$ is evaluated by requirement of charge neutrality at every point of the simulation grid. The potential is determined by solving charge neutrality equation with the bisection algorithm.

Example

```
poisson{
    charge_neutral{}
}
```

7.12.11 zero_charge{ }

Calling sequence

`poisson{ zero_charge{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Linear Poisson equation is solved without charges as initialization step of electrostatic potential $\phi_{i=0}$.

7.12.12 newton_solver{ }

Calling sequence

`poisson{ newton_solver{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

The Newton solver is used for solving the nonlinear Poisson equation. It is solved with a Newton iteration using inexact line search.The Poisson equation is nonlinear because the charge carrier density ρ depends on the electrostatic potential ϕ, i.e. $\rho(\phi)$. For each Newton step a system of linear equations, $A \cdot x = b$, is solved with a linear solver, in order to obtain a gradient. This gradient is used for the inexact line search. Generally, low temperature simulations make the Poisson equation extremely nonlinear at the beginning of the iteration and thus require more line search steps than usual. Using `debuglevel = 2` displays information on the line searchs steps (search_steps): In the *.log* file of your simulation, you can find more information on the convergence of the Newton solver Parameters for solver of nonlinear poisson equation are as follows:

7.12.13 newton_solver{ iterations }

Calling sequence

`poisson{ newton_solver{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 30$
- **unit:** $-$

Functionality

Number of iterations for Newton solver

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{}
}
```

7.12.14 newton_solver{ search_steps }

Calling sequence

`poisson{ newton_solver{ search_steps } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 50$
- **default:** $z = 30$
- **unit:** $-$

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{
        search_steps = 40
    }
}
```

7.12.15 newton_solver{ residual }

Calling sequence

`poisson{ newton_solver{ residual } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 10^3$ for 1D; $r = 10^1$ for 2D; $r = 10^{-4}$ for 3D
- **unit:** cm^{-2} for 1D; cm^{-1} for 2D; none for 3D

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{
        residual = 1e2
    }
}
```

7.12.16 newton_solver{ gradient_shift }

Calling sequence

`poisson{ newton_solver{ gradient_shift } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $-10^{-6} \leq r \leq 10^{-6}$
- **default:** $r = 1e - 11$
- **unit:** $-$

Functionality

Slightly nudges the gradient in case it is effectively zero

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{
        residual = -1e-8
    }
}
```

7.12.17 linear_solver{ }

Calling sequence

`poisson{ linear_solver{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Parameters for linear equation solver in Newton algorithm.

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{}
}
```

7.12.18 linear_solver{ iterations }

Calling sequence

`poisson{ linear_solver{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 1000$
- **unit:** $-$

Functionality

number of iterations for linear equation solver

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        iterations = 2000
    }
}
```

7.12.19 linear_solver{ abs_accuracy }

Calling sequence

`poisson{ linear_solver{ abs_accuracy } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 10^1$ for 1D; $r = 10^{-3}$ for 2D; $r = 10^{-8}$ for 3D
- **unit:** cm^{-2} for 1D; cm^{-1} for 2D; none for 3D

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        abs_accuracy = 1e-2
    }
}
```

7.12.20 linear_solver{ rel_accuracy }

Calling sequence

`poisson{ linear_solver{ rel_accuracy } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.0 \leq r \leq 10^{-6}$
- **default:** $r = 1e - 13$
- **unit:** $-$

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        rel_accuracy = 1e-15
    }
}
```

7.12.21 linear_solver{ dkr_value }

Calling sequence

`poisson{ linear_solver{ dkr_value } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `(..., 0.5]`
- **default:** $r = 0.0$
- **unit:** $-$

Functionality

A parameter to speed up calculations, affects preconditioning. Negative values are ignored but will switch to a slightly slower but more stable preconditioner.

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        dkr_value = 0.1
    }
}
```

7.12.22 linear_solver{ use_cscg }

Calling sequence

`poisson{ linear_solver{ use_cscg } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Forces the slower but occasionally more robust CSCG (Composite Step Conjugate Gradient) linear solver to be used rather than the cg (Conjugate Gradient) linear solver. May occasionally prevent a diagonalization failure.

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        use_cscg = yes
    }
}
```

7.12.23 linear_solver{ force_diagonal_preconditioner }

Calling sequence

`poisson{ linear_solver{ force_diagonal_preconditioner } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Forces the use of a slower but more robust diagonal preconditioner. Should be used only for debugging purposes, enabling will make code much slower or prevent convergence. Try setting it to yes in case preconditioning fails or the linear solver diverges. If set to yes, iterations may have to be further increased.

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        force_diagonal_preconditioner = yes
    }
}
```

7.12.24 linear_solver{ force_iteration }

Calling sequence

`poisson{ linear_solver{ force_iteration } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Only for debugging purposes, enabling will make code much slower or prevent convergence

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        force_iteration = yes
    }
}
```

7.12.25 bisection{ }

Calling sequence

`poisson{ bisection{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Parameters for bisection search. Used for the initial solution of the Poisson equation when `charge_neutral = yes` is set. Bisection is performed in order to achieve local charge neutrality at each grid point:

$$\rho = p - n + sum(N_{D,ionized}) - sum(N_{A,ionized}) = 0$$

Thus, a true classical charge neutrality is computed for classical carrier and doping situations.

Additionally, bisection is also used to determine the electrostatic potential at which contacts become charge neutral, which is also needed for **ohmic contacts** and **charge-neutral contacts**. The bisection for these contacts is performed in any case, i.e. independently to the bisection used when `charge_neutral = yes` is set. The bisection method is a well known algorithm for finding the root of a function. The delta is the so-called convergence tolerance parameter. Specifically in *nextnano*++ we use this method to find the initial solution of the Poisson equation that generally converges very fast using the default parameters and no extra tuning is required.

Example

```
poisson{
    between_fermi_levels{}
    bisection{}
}
```

7.12.26 bisection{ delta }

Calling sequence

`poisson{ bisection{ delta } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 10.0$
- **unit:** eV

Functionality

Range of bisection search.

Example

```
poisson{
    between_fermi_levels{}
    bisection{
        delta = 7.0
    }
}
```

7.12.27 bisection{ residual }

Calling sequence

`poisson{ bisection{ residual } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 1e3$
- **unit:** cm^{-3}

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    bisection{
        residual = 1e1
    }
}
```

7.12.28 bisection{ iterations }

Calling sequence

`poisson{ bisection{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 100$
- **default:** $z = 40$
- **unit:** $-$

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    bisection{
        iterations = 60
    }
}
```

7.12.29 bisection{ robust }

Calling sequence

`poisson{ bisection{ robust } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

When `robust=yes` then a slower charge neutrality algorithm designed to be stable for large band gaps or low temperatures.

Note

The bisection algorithm is also used for initializing quasi-Fermi levels in Ohmic and charge-neutral contacts. In this case, the values specified in the input file may become internally modified. - `iterations` is always increased to be at least 40 - *residual* is reduced to be at most 1e3 cm^{-3} - *robust* is always equal *yes*

Therefore, the contact setup ignores bisection definitions which provide lower accuracy than these default settings.

The intrinsic density in GaN at T=300 K is of the order 1e-10 cm^-3, even smaller in AlN. Extremely low carrier densities may be also expected at low temperatures. In such cases the residual needs to be adjusted to obtain reasonable initialization of the contacts.

Attention

Reducing the default value of `residual` may result in significantly longer initialization times, especially in 3D simulations.

Example

```
poisson{
    between_fermi_levels{}
    bisection{
        robust = yes
    }
}
```

7.12.30 output_potential{ }

Calling sequence

`poisson{ output_potential{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Prints out the electrostatic potential in `(eV)`.

Example

```
poisson{
    between_fermi_levels{}
    output_potential{}
}
```

7.12.31 output_electric_field{ }

Calling sequence

`poisson{ output_electric_field{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Prints out the electric field in kv/cm.

Example

```
poisson{
    between_fermi_levels{}
    output_electric_field{}
}
```

7.12.32 output_electric_displacement{ }

Calling sequence

`poisson{ output_electric_displacement{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Prints out the output electric displacement

Example

```
poisson{
    between_fermi_levels{}
    output_electric_displacement{}
}
```

7.12.33 output_electric_polarization{ }

Calling sequence

`poisson{ output_electric_polarization{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Prints out the output electric polarization

Example

```
poisson{
    between_fermi_levels{}
    output_electric_polarization{}
}
```

7.12.34 output_dielectric_tensor{ }

Calling sequence

`poisson{ output_dielectric_tensor{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Prints out the output dielectric tensor in simulation coordinate system, as it is used while setting up the sparse matrix for the Poisson solver.

Example

```
poisson{
    between_fermi_levels{}
    output_dielectric_tensor{}
}
```

7.12.35 output_dielectric_tensor{ boxes }

Calling sequence

`poisson{ output_dielectric_tensor{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
poisson{
    between_fermi_levels{}
    output_dielectric_tensor{
        boxes = yes
    }
}
```

Last update: 02/04/2025

7.13 currents{ }

Calling sequence

`currents{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- *insulator_bandgap* is not allowed if any of *import_electron_fermi_level{ }* or *import_hole_fermi_level{ }* is already defined.

Functionality

Presence of this group is required to run solver of the current equation. Keywords contained allow selecting mobility and recombination models for the drift-diffusion model of currents, as well as to control some numerical aspects of the related solver and outputs.

Examples

```
currents{
    recombination_model{}
}
```

```
currents{
    recombination_model{}
    insulator_bandgap = 0.5
}
```

```
currents{
    recombination_model{}
    import_electron_fermi_level{}
    import_hole_fermi_level{}
}
```

```
currents{
    recombination_model{}
    minimum_density_electrons = 1e-5
    minimum_density_holes = 1e-7
}
```

Nested keywords

7.13.1 debuglevel

Calling sequence

`currents{ debuglevel }`

Properties

- **usage:** optional
- **type:** integer
- **unit:** $-$
- **values:** $z \geq -1$
- **default:** $z = 1$

Functionality

The higher this integer number, the more information on the numerical solver is printed to the screen output. Increasing the value to 2 or more significantly increases the volume of the diagnostic output displayed in *nextnanomat* (or a shell window). As result of the additional I/O load, particularly 1D simulations will slow down correspondingly (especially for *currents{ }* and *poisson{ }*).

Examples

```
currents{
    recombination_model{}
    debuglevel = 3
}
```

7.13.2 import_electron_fermi_level{ }

Calling sequence

`currents{ import_electron_fermi_level{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The global group *import{ }* and the nested group *import_hole_fermi_level{ }* must be defined.

Functionality

The group allows importing electron quasi-Fermi level to initialize related solver.

Please note that, in case importing an already converged result from solving a (classical or quantum) current-Poisson equation, one should import the electrostatic potential as well to obtain the best convergence.

Moreover, if the imported quasi-Fermi levels and electrostatic potential are resulting from a self-consistent simulation including the Schrödinger equation, and the current simulation is also aiming at solving all three equations self-consistently, then one should omit the running the classical mode of simulation, namely should not call any of the groups *poisson{ }* and *current_poisson{ }*. The simulation should begin already with solving the Schrödinger equation to get the best convergence, i.e., *quantum_current_poisson{ }*.

In case of changed contact bias, one should note that quasi-Fermi levels and electric potential are only imported for areas where they are not defined by boundary conditions (see *contacts{ }*), i.e., they cannot be used to replace these definitions.

> **Warning**
>
> Importing Fermi levels or potential from a simulation with different contact biases results in discontinuities of both quasi-Fermi levels and electric potential at the edge of the contacts, which may lead either to nonphysical results without subsequent iteration or to very poor convergence in subsequent iterations.

Example

```
currents{
    recombination_model{}
    import_electron_fermi_level{...}
    import_hole_fermi_level{...}
}

import{...}
```

Nested keywords

- *import_from*
- *component_number*

import_from

Calling sequence

`currents{ import_electron_fermi_level{ import_from } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A reference name to the path of the imported file defined in *import{ }*.

Example

```
currents{
    recombination_model{}
    import_electron_fermi_level{
        import_from = "reference_name_1"
    }
    import_hole_fermi_level{
        import_from = "reference_name_2"
    }
}

import{
    file{
        name = "reference_name_1"
        ...
    }
    analytic_function{
        name = "reference_name_2"
        ...
    }
}
```

---

component_number

Calling sequence

`currents{ import_electron_fermi_level{ component_number } }`

Properties

- **usage:** optional
- **type:** integer
- **unit:** $-$
- **values:** $z \geq 1$
- **default:** $z = 1$

Functionality

A number referring to the column of numbers in the imported file to be used as the electron quasi-Fermi level.

Example

```
currents{
    recombination_model{}
    import_electron_fermi_level{
        import_from = "reference_name"
        component_number = 2
    }
    import_hole_fermi_level{
        import_from = "reference_name"
        component_number = 3
    }
}

import{
    file{
        name = "reference_name"
        ...
    }
}
```

7.13.3 import_hole_fermi_level{ }

Calling sequence

`currents{ import_hole_fermi_level{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- The global group *import{ }* and the nested group *import_electron_fermi_level{ }* must be defined.

Functionality

The group allows importing hole quasi-Fermi level to initialize related solver.

Please note that, in case importing an already converged result from solving a (classical or quantum) current-Poisson equation, one should import the electrostatic potential as well to obtain the best convergence.

Moreover, if the imported quasi-Fermi levels and electrostatic potential are resulting from a self-consistent simulation including the Schrödinger equation, and the current simulation is also aiming at solving all three equations self-consistently, then one should omit the running the classical mode of simulation, namely should not call any of the groups *poisson{ }* and *current_poisson{ }*. The simulation should begin already with solving the Schrödinger equation to get the best convergence, i.e., *quantum_current_poisson{ }*.

In case of changed contact bias, one should note that quasi-Fermi levels and electric potential are only imported for areas where they are not defined by boundary conditions (see *contacts{ }*), i.e., they cannot be used to replace these definitions.

> **Warning**
>
> Importing Fermi levels or potential from a simulation with different contact biases results in discontinuities of both quasi-Fermi levels and electric potential at the edge of the contacts, which may lead either to nonphysical results without subsequent iteration or to very poor convergence in subsequent iterations.

Example

```
currents{
    recombination_model{}
    import_electron_fermi_level{...}
    import_hole_fermi_level{...}
}

import{...}
```

Nested keywords

- *Maintained Keywords*
    - *import_from*
    - *component_number*

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

---

import_from

Calling sequence

`currents{ import_hole_fermi_level{ import_from } }`

Properties

- **usage:** required
- **type:** character string

Functionality

A reference name to the path of the imported file defined in *import{ }*.

Example

```
currents{
    recombination_model{}
    import_electron_fermi_level{
        import_from = "reference_name_1"
    }
```

(continues on next page)

(continued from previous page)

```
    import_hole_fermi_level{
        import_from = "reference_name_2"
    }
}

import{
    file{
        name = "reference_name_1"
        ...
    }
    analytic_function{
        name = "reference_name_2"
        ...
    }
}
```

---

component_number

Calling sequence

`currents{ import_hole_fermi_level{ component_number } }`

Properties

- **usage:** optional
- **type:** integer
- **unit:** –
- **values:** $z \geq 1$
- **default:** $z = 1$

Functionality

A number referring to the column of numbers in the imported file to be used as the electron quasi-Fermi level.

Example

```
currents{
    recombination_model{}
    import_electron_fermi_level{
        import_from = "reference_name"
        component_number = 2
    }
    import_hole_fermi_level{
        import_from = "reference_name"
        component_number = 3
    }
}

import{
    file{
        name = "reference_name"
        ...
    }
}
```

7.13.4 insulator_bandgap

Calling sequence

`currents{ insulator_bandgap }`

Properties

- **usage:** optional
- **type:** real number
- **unit:** eV
- **values:** `[1e-6, ...)`
- **default:** $r = 1.0$

Functionality

This keyword, I_{gap}, initializes the quasi-Fermi levels following the formula:

$$\operatorname{div} \exp \left(E_{\mathrm{gap}} / I_{\mathrm{gap}} \right) \nabla E_{\mathrm{F}} = 0,$$

where the intrinsic density is assumed to exponentially depend on the band gap E_{gap} with I_{gap} as a parameter.

A large value (relative to band gap) of I_{gap} allows the Fermi level to drop slowly through antire simulation domain. A small value of I_{gap} results in the quasi-Fermi levels drop rapidly in barriers and makes it flat in small band gap regions.

Adjusting this keyword can improve convergence by changing the initial conditions for the algorithm.

Example

```
currents{
    recombination_model{}
    insulator_bandgap = 0.5
}
```

7.13.5 electron_mobility{ }

Calling sequence

`currents{ electron_mobility{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Selects mobility models for electrons. Both low-field and high-field mobility models are possible to be selected in this group.

Example

```
currents{
    recombination_model{}
    electron_mobility{
        low_field_model = constant
    }
}
```

Nested keywords

- *low_field_model*
- *high_field_model*

---

low_field_model

Calling sequence

`currents{ electron_mobility{ low_field_model } }`

Properties

- **usage:** required
- **type:** choice
- **values:** `constant` or `masetti` or `arora` or `minimos`

Functionality

Selects low-field model for electrons.

choice	model	database
`constant`	*Constant model*	*database{ …{ mobility_constant{} } }*
`masetti`	*Masetti model*	*database{ …{ mobility_masetti{} } }*
`arora`	*Arora model*	*database{ …{ mobility_arora{} } }*
`minimos`	*MINIMOS 6 model*	*database{ …{ mobility_minimos{} } }*

Example

```
currents{
    recombination_model{}
    electron_mobility{
        low_field_model = masetti
    }
}
```

---

high_field_model

Calling sequence

`currents{ electron_mobility{ high_field_model } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** none or `haensch` or `canali` or `transferred` or `eastman` or `eastman4`
- **default:** none

Functionality

Selects high-field mobility model for electrons.

choice	model	database
`none`	High-field models are not used	—
`haensch`	*Hänsch model*	*mobility_haensch{ }*
`canali`	*Extended Canali model*	*mobility_canali{ }*
`transferred`	*Hänsch model*	*mobility_transferred{ }*
`eastman`	*Eastman-Tiwari-Shur model with standard parametrization*	*mobility_eastman{ }*
`eastman4`	*Eastman-Tiwari-Shur model with observable parametrization*	*mobility_eastman4{ }*

Warning

Convergence may be poor or non-existent for some choices of parameters. One should pay attention to selecting high-field model which is suitable for the semiconductor system of choice

7.13.6 hole_mobility{ }

Calling sequence

`currents{ hole_mobility{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Selects mobility models for holes. Both low-field and high-field mobility models are possible to be selected in this group.

Example

```
currents{
    recombination_model{}
    hole_mobility{
        low_field_model = constant
    }
}
```

Nested keywords

- *low_field_model*
- *high_field_model*

low_field_model

Calling sequence

`currents{ hole_mobility{ low_field_model } }`

Properties

- **usage:** required
- **type:** choice
- **values:** `constant` or `masetti` or `arora` or `minimos`

Functionality

Selects low-field model for holes.

choice	model	database
`constant`	*Constant model*	*database{ …{ mobility_constant{} } }*
`masetti`	*Masetti model*	*database{ …{ mobility_masetti{} } }*
`arora`	*Arora model*	*database{ …{ mobility_arora{} } }*
`minimos`	*MINIMOS 6 model*	*database{ …{ mobility_minimos{} } }*

Example

```
currents{
    recombination_model{}
    hole_mobility{
        low_field_model = masetti
    }
}
```

---

high_field_model

Calling sequence

`currents{ hole_mobility{ high_field_model } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `none` or `haensch` or `canali` or `transferred` or `eastman` or `eastman4`
- **default:** `none`

Functionality

Selects high-field mobility model for holes.

choice	model	database
`none`	High-field models are not used	—
`haensch`	*Hänsch model*	*mobility_haensch{ }*
`canali`	*Extended Canali model*	*mobility_canali{ }*
`transferred`	*Hänsch model*	*mobility_transferred{ }*
`eastman`	*Eastman-Tiwari-Shur model with standard parametrization*	*mobility_eastman{ }*
`eastman4`	*Eastman-Tiwari-Shur model with observable parametrization*	*mobility_eastman4{ }*

Warning

Convergence may be poor or non-existent for some choices of parameters. One should pay attention to selecting high-field model which is suitable for the semiconductor system of choice

7.13.7 recombination_model{ }

Calling sequence

`currents{ recombination_model{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

This group controls which recombination processes are included in the drift-diffusion model, and if generation for these processes is taken into account as well.

Generation process can be is enabled and disabled using `enable_generation` for all recombination processes at once. Thus, enabling only generation without also enabling recombination is not possible (`enable_generation = yes` has no effect then).

If radiative recombination is calculated (`radiative = yes`), then the `photo_current` is included in the file `IV_characteristics.dat`. Additionally, the internal quantum efficiency is written to the file `internal_quantum_efficiency.dat`.

Example

```
currents{
    recombination_model{}
}
```

Nested keywords

- *SRH*
- *Auger*
- *radiative*
- *enable_generation*

SRH

Calling sequence

`currents{ recombination_model{ SRH } }`

Properties

- **usage:** optional
- **type:** choice

- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then bulk Shockley-Read-Hall recombination (*Shockley-Read-Hall (SRH) recombination*) is included in the model.

Example

```
currents{
    recombination_model{
        SRH = yes
    }
}
```

Auger

Calling sequence

`currents{ recombination_model{ Auger } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then bulk Auger recombination (*Auger recombination*) is included in the model.

Example

```
currents{
    recombination_model{
        Auger = yes
    }
}
```

---

radiative

Calling sequence

`currents{ recombination_model{ radiative } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then bulk radiative recombination (direct recombination) (*Radiative recombination*) is included in the model.

Example

```
currents{
    recombination_model{
        radiative = yes
    }
}
```

---

enable_generation

Calling sequence

`currents{ recombination_model{ enable_generation } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If set to `yes` then bulk generation processes for SRH and Auger recombination processes, if they are included in the model.

Example

```
currents{
    recombination_model{
        SRH = yes
        Auger = yes
        enable_generation = yes
    }
}
```

7.13.8 linear_solver{ }

Calling sequence

`currents{ linear_solver{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- *extended_accuracy* is not allowed if any of *global{ simulate2D{ } }* or *global{ simulate3D{ } }* is already defined.

Functionality

This group allows modifying parameters impacting algorithm of linear equation solver in current equation.

Examples

```
currents{
    recombination_model{}
    linear_solver{}
}
```

```
currents{
    recombination_model{}
    linear_solver{
        extended_accuracy = 1
    }
}

global{
    simulate1D{}
}
```

Nested keywords

- *iterations*
- *abs_accuracy*
- *rel_accuracy*
- *dkr_value*
- *use_cscg*
- *force_diagonal_preconditioner*
- *force_iteration*
- *extended_accuracy*

---

iterations

Calling sequence

`currents{ linear_solver{ iterations } }`

Properties

- **usage:** optional
- **type:** integer
- **unit:** $-$
- **values:** $z \geq 1$
- **default:** $z = 10000$

Functionality

Maximum number of iterations

Example

```
currents{
    recombination_model{}
    linear_solver{
        iterations = 50000
    }
}
```

---

abs_accuracy

Calling sequence

`currents{ linear_solver{ abs_accuracy } }`

Properties

- **usage:** optional
- **type:** real number
- **unit:** –
- **values:** `[0.0, ...)`
- **default:** $r = 1e-30$

Functionality

—

Example

```
currents{
    recombination_model{}
    linear_solver{
        abs_accuracy = 1e-32
    }
}
```

---

rel_accuracy

Calling sequence

`currents{ linear_solver{ rel_accuracy } }`

Properties

- **usage:** optional
- **type:** real number
- **unit:** –
- **values:** $0.0 \leq r \leq 10^{-6}$
- **default:** $r = 1e-13$

Functionality

—

Example

```
currents{
    recombination_model{}
    linear_solver{
        rel_accuracy = 1e-15
    }
}
```

---

dkr_value

Calling sequence

`currents{ linear_solver{ dkr_value } }`

Properties

- **usage:** optional
- **type:** real number
- **unit:** –
- **values:** $0.0 \leq r \leq 0.5$
- **default:** $r = -1.0$

Functionality

A parameter to speed up calculations, affects preconditioning

Note

Negative values are ignored but will switch to a slightly slower but more stable preconditioning.

Example

```
currents{
    recombination_model{}
    linear_solver{
        dkr_value = 0.1
    }
}
```

---

use_cscg

Calling sequence

`currents{ linear_solver{ use_cscg } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Forces the slower but occasionally more robust CSCG (Composite Step Conjugate Gradient) linear solver to be used rather than the cg (Conjugate Gradient) linear solver. May occasionally prevent a diagonalization failure.

Example

```
currents{
    recombination_model{}
    linear_solver{
        use_cscg = yes
    }
}
```

---

force_diagonal_preconditioner

Calling sequence

`currents{ linear_solver{ force_diagonal_preconditioner } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Only for debugging purposes, enabling will make code much slower or prevent convergence. Forces the use of a slower but more robust diagonal preconditioner.

This keyword should be used only for debugging purposes. Enabling the diagonal preconditioner makes algorithm much slower or prevent convergence. It can be enabled in case when then default preconditioning fails or the linear solver diverges. In such circumstances, also *iterations* may require further increasing.

Example

```
currents{
    recombination_model{}
    linear_solver{
        force_diagonal_preconditioner = yes
    }
}
```

---

force_iteration

Calling sequence

`currents{ linear_solver{ force_iteration } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

The keyword should be used only for debugging purposes. It will force iteration to reach maximum set by *iterations* regardless of whether the requested accuracy was reached or not.

Example

```
currents{
    recombination_model{}
    linear_solver{
        force_iteration = yes
    }
}
```

---

extended_accuracy

Calling sequence

`currents{ linear_solver{ extended_accuracy } }`

Properties

- **usage:** optional
- **type:** integer
- **unit:** –
- **values:** $z = 0$ or $z = 1$
- **default:** $z = 0$

Functionality

If set to 1, then current equation is solved using slower but more accurate solver. It is only implemented for not periodic 1D simulations.

⚠ **Warning**

This feature is at the prototyping stage and may not bring expected improvements.

Example

```
currents{
    recombination_model{}
    linear_solver{
        extended_accuracy = 1
    }
}
```

Last update: 02/04/2025

7.13.9 minimum_density_electrons

Calling sequence

`currents{ minimum_density_electrons }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{20}$
- **default:** $r = 1.0$
- **unit:** cm^{-3}

Functionality

A keyword allowing to improve the condition number of the matrix representing the current equation.

Minimum carrier density, ρ_{min}, is defined for electrons as the lower limit for the respective density distributions entering the drift-diffusion current equations. If a density distribution computed based on quasi-Fermi levels and densities of states for a given carrier type, $\rho_{\mathrm{sim}}\left(x\right)$, is smaller than ρ_{min} within some region, then its values in the region are replaced by the ρ_{min} for the equation. In other words, electron distribution entering the current equation, $\rho_{\mathrm{current}}\left(x\right)$, is given by

$$\rho_{\mathrm{current}}\left(x\right) = \max\left[\rho_{\mathrm{sim}}\left(x\right), \rho_{\mathrm{min}}\right].$$

This operation is not visible in the output files.

As the drift-diffusion current is proportional to the charge carrier density, this keyword also indirectly sets the lower limit of the electron current.

Aside from the rather practical issue that real-life minority carrier densities are not in thermal equilibrium and thus never become as small as predicted, it seems nonphysical that one carrier per kilometer can be relevant in semiconductors or insulators. Therefore, the minimum density parameter as specified for the current equation typically should is not be smaller than $10^{-10}\ \mathrm{cm}^{-3}$. This value corresponds to a conductivity 10 orders of magnitude lower than of the best insulators.

Note

The ρ_{min} affects only the current operators ($\nabla\ \mu\ \rho_{\mathrm{current}}\ \nabla$) and the corresponding current for each type of carriers. Thus it has no direct influence on computed densities, Poisson equation, etc.

Hint

- The ρ_{min} might have to be increased in order to obtain convergence for the drift-diffusion current equations.
- The ρ_{min} should be as low as possible, depending on the problem solved.
- The ρ_{min} can be chosen as large as possible but should be small enough to obtain convergence with meaningful results.
- Typically $\rho_{\mathrm{min}} = 10^{12}\ \mathrm{cm}^{-3}$ seems to be already too high.

Attention

Setting the minimum density far too low may result in convergence issues or even in the matrix solvers exiting with an error message. The smallest reasonable number depends on the simulated system. Currently the algorithm allows using the value as small as $10^{-100}\ \mathrm{cm}^{-3}$. Any smaller values are rounded up to this number.

When restricting effective densities in the current equations from below, one should consider impact on the physics of the modelled device, i.e., increasing minimum densities decreases resistivity of insulating regions.

Example

```
currents{
    recombination_model{}
    minimum_density_electrons = 1e10 # cm^-3
}
```

Unimportant currents in Insulators and Barriers The computed current of a given type of carriers often varies over 10 orders of magnitude between barriers (insulators) and conducting regions as a result of extremely small carrier densities in the barriers. If the density in the latter regions reaches values below approximately $10^{3}\ \mathrm{cm}^{-3}$, then the current flowing through them can be practically considered zero in comparison to the total current present in the structure. As a result the matrix representing the current equation, entering the linear solver, is not well conditioned and convergence of the drift-diffusion current equations may be strongly affected by round-off errors. If, the current running through the barriers is not important from the physical point of view, such that increasing it a number of orders of magnitude does not change the final result (e.g., I-V characteristic), then increasing the ρ_{min} to overestimate the current in these regions is a very good way to restore or improve the convergence while preserving meaningful results.

Currents within intrinsic materials If one requires to properly compute the currents within intrinsic regions, then the optimal ρ_{min} should be chosen such that $\rho_{\mathrm{min}} < \rho_{\mathrm{sim}}(x)$ in these regions. The maximum value of a properly chosen ρ_{min} strongly depends on the band gap of the considered material.

Undoped wide-band-gap and highly-doped semiconductors Minority carriers in highly-doped semiconductors or any carriers in undoped wide-band-gap semiconductors have extremely small equilibrium densities (much less than $1.0\ \mathrm{cm}^{-3}$). Computing all currents in these doped materials or for wide-band-gap semiconductor heterostructures, will typically require also considering currents over 15 orders of magnitude higher, which may lead to complete breakdown of the solvers for current equation due to underflow.

7.13.10 minimum_density_holes

Calling sequence

`currents{ minimum_density_holes }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{20}$

- **default:** $r = 1.0$
- **unit:** cm^{-3}

Functionality

A keyword allowing to improve the condition number of the matrix representing the current equation.

Minimum carrier density, ρ_{min}, is defined for holes as the lower limit for the respective density distributions entering the drift-diffusion current equations. If a density distribution computed based on quasi-Fermi levels and densities of states for a given carrier type, $\rho_{\mathrm{sim}}(x)$, is smaller than ρ_{min} within some region, then its values in the region are replaced by the ρ_{min} for the equation. In other words, hole distribution entering the current equation, $\rho_{\mathrm{current}}(x)$, is given by

$$\rho_{\mathrm{current}}(x) = \max\left[\rho_{\mathrm{sim}}(x), \rho_{\mathrm{min}}\right].$$

This operation is not visible in the output files.

As the drift-diffusion current is proportional to the charge carrier density, this keyword also indirectly sets the lower limit of the hole current.

Aside from the rather practical issue that real-life minority carrier densities are not in thermal equilibrium and thus never become as small as predicted, it seems nonphysical that one carrier per kilometer can be relevant in semiconductors or insulators. Therefore, the minimum density parameter as specified for the current equation typically should is not be smaller than $10^{-10}\ \mathrm{cm}^{-3}$. This value corresponds to a conductivity 10 orders of magnitude lower than of the best insulators.

Note

The ρ_{min} affects only the current operators ($\nabla\ \mu\ \rho_{\mathrm{current}}\ \nabla$) and the corresponding current for each type of carriers. Thus it has no direct influence on computed densities, Poisson equation, etc.

Hint

- The ρ_{min} might have to be increased in order to obtain convergence for the drift-diffusion current equations.
- The ρ_{min} should be as low as possible, depending on the problem solved.
- The ρ_{min} can be chosen as large as possible but should be small enough to obtain convergence with meaningful results.
- Typically $\rho_{\mathrm{min}} = 10^{12}\ \mathrm{cm}^{-3}$ seems to be already too high.

Attention

Setting the minimum density far too low may result in convergence issues or even in the matrix solvers exiting with an error message. The smallest reasonable number depends on the simulated system. Currently the algorithm allows using the value as small as $10^{-100}\ \mathrm{cm}^{-3}$. Any smaller values are rounded up to this number.

When restricting effective densities in the current equations from below, one should consider impact on the physics of the modelled device, i.e., increasing minimum densities decreases resistivity of insulating regions.

Example

```
currents{
    recombination_model{}
```

(continues on next page)

(continued from previous page)

```
    minimum_density_holes = 1e10 # cm^-3
}
```

Unimportant currents in Insulators and Barriers The computed current of a given type of carriers often varies over 10 orders of magnitude between barriers (insulators) and conducting regions as a result of extremely small carrier densities in the barriers. If the density in the latter regions reaches values below approximately $10^3\ \text{cm}^{-3}$, then the current flowing through them can be practically considered zero in comparison to the total current present in the structure. As a result the matrix representing the current equation, entering the linear solver, is not well conditioned and convergence of the drift-diffusion current equations may be strongly affected by round-off errors. If, the current running through the barriers is not important from the physical point of view, such that increasing it a number of orders of magnitude does not change the final result (e.g., I-V characteristic), then increasing the ρ_{min} to overestimate the current in these regions is a very good way to restore or improve the convergence while preserving meaningful results.

Currents within intrinsic materials If one requires to properly compute the currents within intrinsic regions, then the optimal ρ_{min} should be chosen such that $\rho_{\text{min}} < \rho_{\text{sim}}(x)$ in these regions. The maximum value of a properly chosen ρ_{min} strongly depends on the band gap of the considered material.

Undoped wide-band-gap and highly-doped semiconductors Minority carriers in highly-doped semiconductors or any carriers in undoped wide-band-gap semiconductors have extremely small equilibrium densities (much less than $1.0\ \text{cm}^{-3}$). Computing all currents in these doped materials or for wide-band-gap semiconductor heterostructures, will typically require also considering currents over 15 orders of magnitude higher, which may lead to complete breakdown of the solvers for current equation due to underflow.

7.13.11 maximum_density_electrons

Calling sequence

`currents{ maximum_density_electrons }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{30}$
- **default:** $r = 1e30$
- **unit:** cm^{-3}

Functionality

A keyword allowing to improve the condition number of the matrix representing the current equation.

Maximum carrier density, ρ_{max}, is defined for holes as the upper limit for the respective density distributions entering the drift-diffusion current equations. If a density distribution computed based on quasi-Fermi levels and densities of states for holes, $\rho_{\text{sim}}(x)$, is higher than ρ_{max} within some region, then its values in the region are replaced by the ρ_{max} for the equation. In other words, every carrier distribution entering the current equation, $\rho_{\text{current}}(x)$, is given by

$$\rho_{\text{current}}(x) = \min\left[\rho_{\text{sim}}(x), \rho_{\text{max}}\right].$$

This operation is not visible in the output files.

As the drift-diffusion current is proportional to the charge carrier density, this keyword also indirectly sets the upper limit of the current.

Note

The $\rho_{\max}$ affects only the current operators ($\nabla\ \mu\ \rho_{\text{current}}\ \nabla$) and the corresponding current for each type of carriers. Thus it has no direct influence on computed densities, Poisson equation, etc.

Hint

- The $\rho_{\max}$ might have to be reduced in order to stabilize convergence for the drift-diffusion current equations.
- The $\rho_{\max}$ should be as high enough to represent current of majority carriers.
- The $\rho_{\max}$ can be chosen as low as possible but should be large enough to not affect the results.

When restricting effective densities in the current equations from above, one should consider impact on the physics of the modelled device, i.e., decreasing maximum densities may decrease conductivity of conducting regions.

Example

```
currents{
    recombination_model{}
    maximum_density_electrons = 1e10  # cm^-3
}
```

7.13.12 maximum_density_holes

Calling sequence

`currents{ maximum_density_holes }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-100} \leq r \leq 10^{30}$
- **default:** $r = 1e30$
- **unit:** cm^{-3}

Functionality

A keyword allowing to improve the condition number of the matrix representing the current equation.

Maximum carrier density, $\rho_{\max}$, is defined for holes as the upper limit for the respective density distributions entering the drift-diffusion current equations. If a density distribution computed based on quasi-Fermi levels and densities of states for holes, $\rho_{\text{sim}}(x)$, is higher than $\rho_{\max}$ within some region, then its values in the region are replaced by the $\rho_{\max}$ for the equation. In other words, every carrier distribution entering the current equation, $\rho_{\text{current}}(x)$, is given by

$$\rho_{\text{current}}(x) = \min\left[\rho_{\text{sim}}(x), \rho_{\max}\right].$$

This operation is not visible in the output files.

As the drift-diffusion current is proportional to the charge carrier density, this keyword also indirectly sets the upper limit of the current.

Note

The $\rho_{\max}$ affects only the current operators ($\nabla\ \mu\ \rho_{\text{current}}\ \nabla$) and the corresponding current for each type of carriers. Thus it has no direct influence on computed densities, Poisson equation, etc.

Hint

- The $\rho_{\max}$ might have to be reduced in order to stabilize convergence for the drift-diffusion current equations.
- The $\rho_{\max}$ should be as high enough to represent current of majority carriers.
- The $\rho_{\max}$ can be chosen as low as possible but should be large enough to not affect the results.

When restricting effective densities in the current equations from above, one should consider impact on the physics of the modelled device, i.e., decreasing maximum densities may decrease conductivity of conducting regions.

Example

```
currents{
    recombination_model{}
    maximum_density_holes = 1e10  # cm^-3
}
```

7.13.13 electron_contact

Calling sequence

`currents{ electron_contact }`

Properties

- **usage:** optional
- **type:** character string

Functionality

Current equation for electrons around a contact having a name assigned to this keyword is solved with enhanced accuracy.

Example

```
currents{
    recombination_model{}
    electron_contact = "contact_name"
}

contacts{
    schottky{
        name = "contact_name"
        ...
    }
}
```

7.13.14 hole_contact

Calling sequence

`currents{ hole_contact }`

Properties

- **usage:** optional
- **type:** character string

Functionality

Current equation for holes around a contact having a name assigned to this keyword is solved with enhanced accuracy.

Example

```
currents{
    recombination_model{}
    hole_contact = "contact_name"
}

contacts{
    schottky{
        name = "contact_name"
        ...
    }
}
```

7.13.15 output_fermi_levels{ }

Calling sequence

`currents{ output_fermi_levels{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs quasi-Fermi levels for electrons and holes in eV.

Example

```
currents{
    recombination_model{}
    output_fermi_levels{}
}
```

7.13.16 output_fermi_level_difference{ }

Calling sequence

`currents{ output_fermi_level_difference{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs the difference of quasi-Fermi levels for electrons and holes $\Delta E_\mathrm{F} = E_\mathrm{F,n} - E_\mathrm{F,p}$ in eV. By overlaying the quasi-Fermi level difference over the band gaps, you may determine where and involving which bands lasing may occur.

Example

```
currents{
    recombination_model{}
    output_fermi_level_difference{}
}
```

7.13.17 output_velocities{ }

Calling sequence

`currents{ output_velocities{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs electron and hole drift velocities in $\mathrm{cm/s}$.

Example

```
currents{
    recombination_model{}
    output_velocities{}
}
```

7.13.18 output_forces{ }

Calling sequence

`currents{ output_forces{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs driving forces of electrons and holes $\mathrm{eV/nm}$

Example

```
currents{
    recombination_model{}
    output_forces{}
}
```

7.13.19 output_currents{ }

Calling sequence

`currents{ output_currents{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs the electron and hole current densities expressed in $\mathrm{A/cm^2}$.

The electron, hole, and total currents (integrated over the contacts surfaces) are always written into the files *IV_electrons.dat*, *IV_holes.dat*, and *IV_characteristics.dat* in `[A/cm^2]`, `[A/cm]`, and `[A]` for 1D, 2D, and 3D simulations, respectively. If radiative recombination is used, then the file *IV_characteristics.dat* also contains the photo current.

In all *IV_*.dat* files, the **first columns** indicate the voltages at each contact. Typically, the first column should be the one that is swept, as it is then easier to plot the results within *nextnanomat* as the first column is the x-axis in such a plot. You can switch the columns by reordering the contacts, see *contacts{ }*. The consumed power is written in *IV_Power.dat* in `[W/cm^2]`, `[W/cm]`, and `[W]` for 1D, 2D, and 3D simulations, respectively. The **emitted power** column is added if the energy resolved density integration is enabled.

Example

```
currents{
    recombination_model{}
    output_currents{}
}
```

7.13.20 output_power_density{ }

Calling sequence

`currents{ output_power_density{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs power density of Joule heating expressed in $\mathrm{W/cm^3}$.

Example

```
currents{
    recombination_model{}
    output_power_density{}
}
```

7.13.21 output_mobilities{ }

Calling sequence

`currents{ output_mobilities{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs the electron and hole mobilities expressed in $\mathrm{cm}^2/\mathrm{V\,s}$

Example

```
currents{
    recombination_model{}
    output_mobilities{}
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

`currents{ output_mobilities{ boxes } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

Example

```
currents{
    recombination_model{}
    output_mobilities{
        boxes = yes
    }
}
```

7.13.22 output_recombination{ }

Calling sequence

`currents{ output_recombination{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs all recombination and generation rates (if included in the model) expressed in $10^{18}/\mathrm{cm}^3\,\mathrm{s}$.

Example

```
currents{
    recombination_model{}
    output_recombination{}
}
```

7.13.23 output_injection{ }

Calling sequence

`currents{ output_injection{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs all injection rates (if included in the model) expressed in $10^{18}/\mathrm{cm}^3\,\mathrm{s}$.

Example

```
currents{
    recombination_model{}
    output_injection{}
}
```

7.14 quantum{ }

Calling sequence

`quantum{ }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

A group of keywords specifying quantum models, i.e. how the Schrödinger equation should be solved.

Nested keywords

7.14.1 debuglevel

Calling sequence

`quantum{ debuglevel = ... }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 1$

Functionality

The higher this integer number, the more information on the numerical solver is printed to the screen output. Increasing the respective debug level to 2 or more significantly increases the volume of the diagnostic output displayed in *nextnanomat* (or a shell window). As result of the additional I/O load, particularly 1D simulations will slow down correspondingly (especially for `current{}` and `poisson{ }`).

7.14.2 allow_overlapping_regions

Calling sequence

`quantum{ allow_overlapping_regions = ... }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Overlapping quantum regions computing the same band(s) are not allowed. Note that, in case such overlap is allowed, the quantum densities of the respective regions are added in the overlap region and a too high density will be computed. Thus, please only allow such overlap when the quantum densities are known to be extremely small in the overlap region.

Last update: 27/05/2025

7.14.3 region{ }

Calling sequence

`quantum{ region{ } }`

Properties

- **usage:** optional
- **items:** no constraints

Functionality

It is the most important nested group in `quantum{ }`. It allows to define a region in the simulation domain and assign a specific model to solve the Schrödinger equation inside the defined region.

Nested keywords

name

Calling sequence

`quantum{ region{ name = "..." } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Provides a name for the quantum region. This name can be further used to link other models to this specific region.

---

no_density

Calling sequence

`quantum{ region{ no_density = "..." } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

`yes` → solutions of the Schrödinger equation for this region **are not** used for computation of charge densities. `no` → solutions of the Schrödinger equation for this region **are** used for computation of charge densities.

---

x

Calling sequence

`quantum{ region{ x = [..., ...] } }`

Properties

- **usage:** required
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **units:** $[r_1] = \text{nm}$, $[r_2] = \text{nm}$

Functionality

Defines the range of quantum region along the x-axis of the simulation domain with r_1 and r_2 defining the beginning and the end of the range, respectively.

---

y

Calling sequence

`quantum{ region{ y = [..., ...] } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **units:** $[r_1] = \text{nm}$, $[r_2] = \text{nm}$

Dependencies

— Can be used for 2D or 3D calculations only.

Functionality

Defines the range of quantum region along the y-axis of the simulation domain with r_1 and r_2 defining the beginning and the end of the range, respectively.

---

z

Calling sequence

`quantum{ region{ z = [..., ...] } }`

Properties

- **usage:** optional
- **type:** vector of 2 real numbers: (r_1, r_2)
- **values:** no constraints
- **default:** $r_1 = 0.0$, $r_2 = 0.0$
- **units:** $[r_1] = \text{nm}$, $[r_2] = \text{nm}$

Dependencies

— Can be used for 3D calculation only.

Functionality

Defines the range of quantum region along the z-axis of the simulation domain with r_1 and r_2 defining the beginning and the end of the range, respectively.

---

spin_quantization_axis

Calling sequence

`quantum{ region{ spin_quantization_axis = [..., ..., ...] } }`

Properties

- **usage:** conditional
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- —
- **units:** $[r_1] = \mathrm{nm}$, $[r_2] = \mathrm{nm}$

Dependencies

—

Functionality

Defines spin quantization axis within the simulation coordinate system.

Last update: 27/05/2025

array_x{ }, array_y{ }, array_z{ }

Calling sequence

```
quantum{ region{ array_x{ } } }
quantum{ region{ array_y{ } } }
quantum{ region{ array_z{ } } }
```

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

—

Functionality

—

Nested keywords

- *array_x{ shift }, ...*
- *array_x{ min }, ...*
- *array_x{ max }, ...*

---

array_x{ shift }, ...

Calling sequence

```
quantum{ region{ array_x{ shift = ... } } }
quantum{ region{ array_y{ shift = ... } } }
quantum{ region{ array_z{ shift = ... } } }
```

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

—

---

array_x{ min }, ...

Calling sequence

```
quantum{ region{ array_x{ min = ... } } }
quantum{ region{ array_y{ min = ... } } }
quantum{ region{ array_z{ min = ... } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **default:** $z = 0$

Functionality

—

---

array_x{ max }, ...

Calling sequence

```
quantum{ region{ array_x{ max ... } } }
quantum{ region{ array_y{ max ... } } }
quantum{ region{ array_z{ max ... } } }
```

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **default:** $z = 0$

Functionality

—

Last update: 27/05/2025

array2_x{ }, array2_y{ }, array2_z{ }

Calling sequence

```
quantum{ region{ array2_x{ } } }
quantum{ region{ array2_y{ } } }
quantum{ region{ array2_z{ } } }
```

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

—

Functionality

—

Nested keywords

- *array2_x{ shift }, . . .*
- *array2_x{ min }, . . .*
- *array2_x{ max }, . . .*

---

array2_x{ shift }, . . .

Calling sequence

```
quantum{ region{ array2_x{ shift = ... } } }
quantum{ region{ array2_y{ shift = ... } } }
quantum{ region{ array2_z{ shift = ... } } }
```

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

—

---

array2_x{ min }, . . .

Calling sequence

```
quantum{ region{ array2_x{ min = ... } } }
quantum{ region{ array2_y{ min = ... } } }
quantum{ region{ array2_z{ min = ... } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \leq 0$
- **default:** $z = 0$

Functionality

—

---

array2_x{ max }, . . .

Calling sequence

```
quantum{ region{ array2_x{ max ... } } }
quantum{ region{ array2_y{ max ... } } }
quantum{ region{ array2_z{ max ... } } }
```

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 0$
- **default:** $z = 0$

Functionality

—

Last update: 27/05/2025

Gamma{}, L{}, X{}, Delta{}, HH{}, LH{}, SO{}

Calling sequence

```
quantum{ region{ Gamma{ } } }
quantum{ region{ L{ } } }
quantum{ region{ X{ } } }
quantum{ region{ Delta{ } } }
quantum{ region{ HH{ } } }
quantum{ region{ LH{ } } }
quantum{ region{ SO{ } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

`Gamma{}`, `L{}`, `X{}`, `Delta{}`, `HH{}`, `LH{}`, and `SO{}` trigger solving single-band effective mass Schrödinger equation for the **Gamma** conduction band, the **L** conduction band, the **X** conduction band, the **Delta** conduction band, the **heavy hole** valence band, the **light hole** valence band, and the **split-off hole** valence band, respectively.

Nested keywords

- *force_complex_solver*
- *force_pauli_solver*
- *accuracy*
- *iterations*
- *num_ev*
- *lapack{ }*
- *arpack{ }*
- *preconditioner*
- *order_polynomial*
- *order_chebyshev*
- *order_legendre*
- *cutoff*
- *abs_cutoff*
- *k_integration{ }*
- *k_integration{ relative_size }*
- *k_integration{ max_symmetry }*
- *k_integration{ num_points }*
- *k_integration{ num_subpoints }*
- *k_integration{ force_k0_subspace }*
- *dispersion{ }*
- *dispersion{ path{ } }*
- *dispersion{ path{ name } }*
- *dispersion{ path{ point{ } } }*
- *dispersion{ path{ point{ k } } }*
- *dispersion{ path{ spacing } }*
- *dispersion{ path{ num_points } }*
- *dispersion{ lines{ } }*
- *dispersion{ lines{ name } }*
- *dispersion{ lines{ k_max } }*
- *dispersion{ lines{ spacing } }*
- *dispersion{ full{ } }*
- *dispersion{ full{ name } }*
- *dispersion{ full{ kxgrid{ }, … } }*
- *dispersion{ full{ kxgrid{ line{ } }, … } }*
- *dispersion{ full{ kxgrid{ line{ pos } }, … } }*
- *dispersion{ full{ kxgrid{ line{ spacing } }, … } }*
- *dispersion{ superlattice{ } }*
- *dispersion{ superlattice{ name } }*
- *dispersion{ superlattice{ num_points } }*
- *dispersion{ superlattice{ num_points_x, … } }*
- *dispersion{ output_dispersions{ } }*
- *dispersion{ output_dispersions{ max_num } }*
- *dispersion{ output_masses{ } }*
- *dispersion{ output_masses{ max_num } }*

---

force_complex_solver

Calling sequence

```
quantum{ region{ Gamma{ force_complex_solver = ... } } }
quantum{ region{ L{ force_complex_solver = ... } } }
quantum{ region{ X{ force_complex_solver = ... } } }
quantum{ region{ Delta{ force_complex_solver = ... } } }
quantum{ region{ HH{ force_complex_solver = ... } } }
quantum{ region{ LH{ force_complex_solver = ... } } }
quantum{ region{ SO{ force_complex_solver = ... } } }
```

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes`, then resulting wave functions are expressed as complex functions, even though imaginary part is equal to zero.

> **Note**
>
> Complex envelopes are needed for *optics{ }* group.

---

force_pauli_solver

Calling sequence

```
quantum{ region{ Gamma{ force_pauli_solver = ... } } }
quantum{ region{ L{ force_pauli_solver = ... } } }
quantum{ region{ X{ force_pauli_solver = ... } } }
quantum{ region{ Delta{ force_pauli_solver = ... } } }
quantum{ region{ HH{ force_pauli_solver = ... } } }
quantum{ region{ LH{ force_pauli_solver = ... } } }
quantum{ region{ SO{ force_pauli_solver = ... } } }
```

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

When se to `yes`, the a Pauli equation is solved even in the absence of magnetic field.

---

accuracy

Calling sequence

```
quantum{ region{ Gamma{ accuracy = ... } } }
quantum{ region{ L{ accuracy = ... } } }
quantum{ region{ X{ accuracy = ... } } }
quantum{ region{ Delta{ accuracy = ... } } }
quantum{ region{ HH{ accuracy = ... } } }
quantum{ region{ LH{ accuracy = ... } } }
quantum{ region{ SO{ accuracy = ... } } }
```

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.0 \leq r \leq 10^{-6}$
- **default:** $r = 10^{-10}$ for ARPACK; $r = 10^{-7}$ for ARPACK_INV and DAVIDSON
- **unit:** $-$

Functionality

Sets accuracy of finding eigenvalues by APRACK, ARPACK_INV, and DAVIDSON routines.

---

iterations

Calling sequence

```
quantum{ region{ Gamma{ iterations = ... } } }
quantum{ region{ L{ iterations = ... } } }
quantum{ region{ X{ iterations = ... } } }
quantum{ region{ Delta{ iterations = ... } } }
quantum{ region{ HH{ iterations = ... } } }
quantum{ region{ LH{ iterations = ... } } }
quantum{ region{ SO{ iterations = ... } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 100000$

Functionality

Number of iterations for eigenvalue solvers.

---

num_ev

Calling sequence

```
quantum{ region{ Gamma{ num_ev = ... } } }
quantum{ region{ L{ num_ev = ... } } }
quantum{ region{ X{ num_ev = ... } } }
quantum{ region{ Delta{ num_ev = ... } } }
quantum{ region{ HH{ num_ev = ... } } }
quantum{ region{ LH{ num_ev = ... } } }
quantum{ region{ SO{ num_ev = ... } } }
```

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 1$

Functionality

Sets the number of eigenvalues to be calculated.

---

lapack{ }

Calling sequence

```
quantum{ region{ Gamma{ lapack{ } } } }
quantum{ region{ L{ lapack{ } } } }
quantum{ region{ X{ lapack{ } } } }
quantum{ region{ Delta{ lapack{ } } } }
quantum{ region{ HH{ lapack{ } } } }
quantum{ region{ LH{ lapack{ } } } }
quantum{ region{ SO{ lapack{ } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Triggers use of LAPACK eigensolver to solve dense matrix problem. It should be used for 1D and small 2D systems. For 1D simulations without periodic boundary conditions a tridiagonal LAPACK solver is used for the single-band Hamiltonian as default.

arpack{ }

Calling sequence

```
quantum{ region{ Gamma{ arpack{ } } } }
quantum{ region{ L{ arpack{ } } } }
quantum{ region{ X{ arpack{ } } } }
quantum{ region{ Delta{ arpack{ } } } }
quantum{ region{ HH{ arpack{ } } } }
quantum{ region{ LH{ arpack{ } } } }
quantum{ region{ SO{ arpack{ } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

ARPACK eigensolver is used to solve eigenvalue problem using sparse matrix routines. It ARPACK should be faster for large matrices (N > 1000) where only a few eigenvalues are sought (~5-30). Memory usage of arpack (and also arpack_inv) only depends on the number of eigenvectors requested, and is not influenced by the type of preconditioner used.Essentially, for each requested eigenvector (i.e. wave function), additional temporary space corresponding to 2.5 eigenvectors is needed during runtime. Among the preconditioners, Chebyshev preconditioning and Legendre preconditioning are comparably fast, but require both the specification of a cutoff energy under (above) which all eigenvalues of interest are assumed to be located. If this assumption is violated, only spurious parts of the energy spectrum will be computed. On the other hand, setting the cutoff energy too generous will slow down convergence. Since the energy spectrum often shifts during the Quantum-Poisson iteration, a more generous initial cutoff energy is also needed for the first Quantum-Poisson iteration step. If this initial cutoff energy is not provided, much slower but more predictable polynomial preconditioning will be used for the first Quantum-Poisson iteration step instead of the specified Chebyshev / legendre preconditioner. Alternatively, this slower polynomial preconditioning can also be used for the entire Quantum-Poisson iteration. In this case, no cutoff energies need to be specified at all. Generally, it is advisable to use polynomial preconditioning when simulating a new structure until the distribution of the eigenvalues, the location of the Fermi level(s), and the required numbers of eigenvalues are better known. Performance of all preconditioners can be further tuned by changing the order of the respective polynomial used, with optimal values typically lying between 10 and 30. ARPACK will terminate once the desired accuracy has been reached or the specified number of iterations has been exceeded. In the latter case, not all requested eigenvectors may have been calculated, or convergence may be incomplete.

> ⚠ **Warning**
>
> Too low cutoff energy, not enough number of states selected to compute, and residuals set too low for large systems are common reasons of failure of ARPACK eigensolver. The method may occur unstable for 8-band model in general.

> ℹ **Note**
>
> The default behavior of ARPACK eigensolver is the following: When the Schrödinger equation is solved for the first time, the polynomial preconditioner is used, because there is no suitable cutoff energy known. In all later Quantum-Poisson iterations the Chebyshev preconditioner will be used (up to two times faster) with a cutoff energy slightly above the highest eigenvalue, which was calculated in the last iteration.

preconditioner

Calling sequence

```
quantum{ region{ Gamma{ preconditioner = ... } } }
quantum{ region{ L{ preconditioner = ... } } }
quantum{ region{ X{ preconditioner = ... } } }
quantum{ region{ Delta{ preconditioner = ... } } }
quantum{ region{ HH{ preconditioner = ... } } }
quantum{ region{ LH{ preconditioner = ... } } }
quantum{ region{ SO{ preconditioner = ... } } }
```

Properties

- **usage:** optional
- **type:** choice
- **values:** `polynomial` or `chebyshev` or `legendre`
- **default:** `chebyshev`

Functionality

The Polynomial preconditioner is the slowest but does not require to specify cutoff energy whereas Chebyshev or Legendre preconditioners requires you to specify cutoff energy.

---

order_polynomial

Calling sequence

```
quantum{ region{ Gamma{ order_polynomial = ... } } }
quantum{ region{ L{ order_polynomial = ... } } }
quantum{ region{ X{ order_polynomial = ... } } }
quantum{ region{ Delta{ order_polynomial = ... } } }
quantum{ region{ HH{ order_polynomial = ... } } }
quantum{ region{ LH{ order_polynomial = ... } } }
quantum{ region{ SO{ order_polynomial = ... } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 0$
- **default:** $z = 20$

Functionality

Order of the polynomial used for polynomial preconditioning.

---

order_chebyshev

Calling sequence

```
quantum{ region{ Gamma{ order_chebyshev = ... } } }
quantum{ region{ L{ order_chebyshev = ... } } }
quantum{ region{ X{ order_chebyshev = ... } } }
quantum{ region{ Delta{ order_chebyshev = ... } } }
quantum{ region{ HH{ order_chebyshev = ... } } }
quantum{ region{ LH{ order_chebyshev = ... } } }
quantum{ region{ SO{ order_chebyshev = ... } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 20$

Functionality

Order of the polynomial used for Chebyshev preconditioning.

order_legendre

Calling sequence

```
quantum{ region{ Gamma{ order_legendre = ... } } }
quantum{ region{ L{ order_legendre = ... } } }
quantum{ region{ X{ order_legendre = ... } } }
quantum{ region{ Delta{ order_legendre = ... } } }
quantum{ region{ HH{ order_legendre = ... } } }
quantum{ region{ LH{ order_legendre = ... } } }
quantum{ region{ SO{ order_legendre = ... } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 20$

Functionality

Order of the polynomial used for Legendre preconditioning.

cutoff

Calling sequence

```
quantum{ region{ Gamma{ cutoff = ... } } }
quantum{ region{ L{ cutoff = ... } } }
quantum{ region{ X{ cutoff = ... } } }
quantum{ region{ Delta{ cutoff = ... } } }
quantum{ region{ HH{ cutoff = ... } } }
quantum{ region{ LH{ cutoff = ... } } }
quantum{ region{ SO{ cutoff = ... } } }
```

Properties

- **usage:** optional
- **type:** real number
- **values:** `[1e-3, ...)`
- **default:** $r = 0.3$
- **unit:** eV

Functionality

—

---

abs_cutoff

Calling sequence

```
quantum{ region{ Gamma{ abs_cutoff = ... } } }
quantum{ region{ L{ abs_cutoff = ... } } }
quantum{ region{ X{ abs_cutoff = ... } } }
quantum{ region{ Delta{ abs_cutoff = ... } } }
quantum{ region{ HH{ abs_cutoff = ... } } }
quantum{ region{ LH{ abs_cutoff = ... } } }
quantum{ region{ SO{ abs_cutoff = ... } } }
```

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** eV

Functionality

—

---

k_integration{ }

Calling sequence

`quantum{ region{ Gamma{ k_integration{ } } } }` `quantum{ region{ L{ k_integration{ } } } }` `quantum{ region{ X{ k_integration{ } } } }` `quantum{ region{ Delta{ k_integration{ } } } }` `quantum{ region{ HH{ k_integration{ } } } }` `quantum{ region{ LH{ k_integration{ } } } }` `quantum{ region{ SO{ k_integration{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Provides options for integration over $\mathbf{k}_{||}$ space for density calculations (for 1D and 2D only).

---

k_integration{ relative_size }

Calling sequence

`quantum{ region{ Gamma{ k_integration{ relative_size = ... } } } }` `quantum{ region{ L{ k_integration{ relative_size = ... } } } }` `quantum{ region{ X{ k_integration{ relative_size = ... } } } }` `quantum{ region{ Delta{ k_integration{ relative_size = ... } } } }` `quantum{ region{ HH{ k_integration{ relative_size = ... } } } }` `quantum{ region{ LH{ k_integration{ relative_size = ... } } } }` `quantum{ region{ SO{ k_integration{ relative_size = ... } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-3} \leq r \leq 10.0$
- **default:** $z = 1$
- **unit:** eV

Functionality

—

---

k_integration{ max_symmetry }

Calling sequence

`quantum{ region{ Gamma{ k_integration{ max_symmetry = ... } } } }` `quantum{ region{ L{ k_integration{ max_symmetry = ... } } } }` `quantum{ region{ X{ k_integration{ max_symmetry = ... } } } }` `quantum{ region{ Delta{ k_integration{ max_symmetry = ... } } } }` `quantum{ region{ HH{ k_integration{ max_symmetry = ... } } } }` `quantum{ region{ LH{ k_integration{ max_symmetry = ... } } } }` `quantum{ region{ SO{ k_integration{ max_symmetry = ... } } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `no` or `C2` or `full`
- **default:** `full`

Functionality

—

---

k_integration{ num_points }

Calling sequence

`quantum{ region{ Gamma{ k_integration{ num_points = ... } } } }` `quantum{ region{ L{ k_integration{ num_points = ... } } } }` `quantum{ region{ X{ k_integration{ num_points = ... } } } }` `quantum{ region{ Delta{ k_integration{ num_points = ... } } } }` `quantum{ region{ HH{ k_integration{ num_points = ... } } } }` `quantum{ region{ LH{ k_integration{ num_points = ... } } } }` `quantum{ region{ SO{ k_integration{ num_points = ... } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $2 \leq z \leq 100$
- **default:** $z = 10$

Functionality

—

---

k_integration{ num_subpoints }

Calling sequence

`quantum{ region{ Gamma{ k_integration{ num_subpoints = ... } } } }` `quantum{ region{ L{ k_integration{ num_subpoints = ... } } } }` `quantum{ region{ X{ k_integration{ num_subpoints = ... } } } }` `quantum{ region{ Delta{ k_integration{ num_subpoints = ... } } } }` `quantum{ region{ HH{ k_integration{ num_subpoints = ... } } } }` `quantum{ region{ LH{ k_integration{ num_subpoints = ... } } } }` `quantum{ region{ SO{ k_integration{ num_subpoints = ... } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 1000$
- **default:** $z = 4$

Functionality

—

---

k_integration{ force_k0_subspace }

Calling sequence

`quantum{ region{ Gamma{ k_integration{ force_k0_subspace = ... } } } }` `quantum{ region{ L{ k_integration{ force_k0_subspace = ... } } } }` `quantum{ region{ X{ k_integration{ force_k0_subspace = ... } } } }` `quantum{ region{ Delta{ k_integration{ force_k0_subspace = ... } } } }` `quantum{ region{ HH{ k_integration{ force_k0_subspace = ... } } } }` `quantum{ region{ LH{ k_integration{ force_k0_subspace = ... } } } }` `quantum{ region{ SO{ k_integration{ force_k0_subspace = ... } } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

When ste to `yes` then wave functions computed at the Γ point are used for computation of carrier densities for every other wave vector within each band.

---

dispersion{ }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ } } } }
quantum{ region{ L{ dispersion{ } } } }
quantum{ region{ X{ dispersion{ } } } }
quantum{ region{ Delta{ dispersion{ } } } }
quantum{ region{ HH{ dispersion{ } } } }
quantum{ region{ LH{ dispersion{ } } } }
quantum{ region{ SO{ dispersion{ } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

These groups provide keywords to define a path for computation of $\mathbf{k}_{||}$ and $\mathbf{k}_{\mathbf{superlattice}}$ (if applicable) dispersions. The energy dispersion E($\mathbf{k}$) along the specified paths and for the specified $\mathbf{k}$ space resolutions are completely independent from the $\mathbf{k}$ space resolution that was used within the self-consistent cycle where the k.p density has been calculated. The latter is specified in `k_integration{ }`.

---

dispersion{ path{ } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ path{ } } } } }
quantum{ region{ L{ dispersion{ path{ } } } } }
quantum{ region{ X{ dispersion{ path{ } } } } }
quantum{ region{ Delta{ dispersion{ path{ } } } } }
quantum{ region{ HH{ dispersion{ path{ } } } } }
quantum{ region{ LH{ dispersion{ path{ } } } } }
quantum{ region{ SO{ dispersion{ path{ } } } } }
```

Properties

- **usage:** optional
- **items:** no constraints

Functionality

Calculates dispersion along custom path in k-space. Multiple instances are allowed.

---

dispersion{ path{ name } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ path{ name = ... } } } } }
quantum{ region{ L{ dispersion{ path{ name = ... } } } } }
quantum{ region{ X{ dispersion{ path{ name = ... } } } } }
quantum{ region{ Delta{ dispersion{ path{ name = ... } } } } }
quantum{ region{ HH{ dispersion{ path{ name = ... } } } } }
quantum{ region{ LH{ dispersion{ path{ name = ... } } } } }
quantum{ region{ SO{ dispersion{ path{ name = ... } } } } }
```

Properties

- **usage:** required
- **type:** character string

Functionality

Is a name of the dispersions which also defines the names of the output files.

---

dispersion{ path{ point{ } } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ path{ point{ } } } } } }
quantum{ region{ L{ dispersion{ path{ point{ } } } } } }
quantum{ region{ X{ dispersion{ path{ point{ } } } } } }
quantum{ region{ Delta{ dispersion{ path{ point{ } } } } } }
quantum{ region{ HH{ dispersion{ path{ point{ } } } } } }
quantum{ region{ LH{ dispersion{ path{ point{ } } } } } }
quantum{ region{ SO{ dispersion{ path{ point{ } } } } } }
```

Properties

- **usage:** required
- **items:** minimum 2

Functionality

Specifies points in the path through k-space. At least two k points have to be defined. Line between two such points is called segment.

---

dispersion{ path{ point{ k } } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ path{ point{ k = [ ..., ..., ... ] } } } } } }
quantum{ region{ L{ dispersion{ path{ point{ k = [ ..., ..., ... ] } } } } } }
quantum{ region{ X{ dispersion{ path{ point{ k = [ ..., ..., ... ] } } } } } }
quantum{ region{ Delta{ dispersion{ path{ point{ k = [ ..., ..., ... ] } } } } } }
quantum{ region{ HH{ dispersion{ path{ point{ k = [ ..., ..., ... ] } } } } } }
quantum{ region{ LH{ dispersion{ path{ point{ k = [ ..., ..., ... ] } } } } } }
quantum{ region{ SO{ dispersion{ path{ point{ k = [ ..., ..., ... ] } } } } } }
```

Properties

- **usage:** required
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **unit:** nm^{-1}

Functionality

Is a k-point represented by vector $[k_x, k_y, k_z]$.

For 1D simulation the $\mathbf{k}_{||}$ space is a $k_y - k_z$ plane so k_y, k_z can be freely choosed. k_x can only be different from zero, if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain.

for 2D simulation the $\mathbf{k}_{||}$ space is a k_z axis so k_z can be freely choosed. kx can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain. k_y can only be different from zero if a periodic boundary condition along the y-direction is defined and the quantum region extends over the whole y-domain.

for 3D simulation the $\mathbf{k}_{||}$ space is empty. k_x can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain. k_y can only be different from zero if a periodic boundary condition along the y-direction is defined and the quantum region extends over the whole y-domain. k_z can only be different from zero if a periodic boundary condition along the z-direction is defined and the quantum region extends over the whole z-domain.

---

dispersion{ path{ spacing } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ path{ spacing = ... } } } } } }
quantum{ region{ L{ dispersion{ path{ spacing = ... } } } } } }
```

```
quantum{ region{ X{ dispersion{ path{ spacing = ... } } } } }
quantum{ region{ Delta{ dispersion{ path{ spacing = ... } } } } }
quantum{ region{ HH{ dispersion{ path{ spacing = ... } } } } }
quantum{ region{ LH{ dispersion{ path{ spacing = ... } } } } }
quantum{ region{ SO{ dispersion{ path{ spacing = ... } } } } }
```

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Functionality

Specifies approximate spacing for intermediate points in the path segments in nm^{-1}. Excludes `num_points`.

dispersion{ path{ num_points } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ path{ num_points = ... } } } } }
quantum{ region{ L{ dispersion{ path{ num_points = ... } } } } }
quantum{ region{ X{ dispersion{ path{ num_points = ... } } } } }
quantum{ region{ Delta{ dispersion{ path{ num_points = ... } } } } }
quantum{ region{ HH{ dispersion{ path{ num_points = ... } } } } }
quantum{ region{ LH{ dispersion{ path{ num_points = ... } } } } }
quantum{ region{ SO{ dispersion{ path{ num_points = ... } } } } }
```

Properties

- **usage:** conditional
- **type:** integer
- **values:** $z \geq 2$

Functionality

Specifies number of points (intermediate + two corner points) for each single path segment. Excludes `spacing`.

dispersion{ lines{ } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ lines{ } } } } }
quantum{ region{ L{ dispersion{ lines{ } } } } }
quantum{ region{ X{ dispersion{ lines{ } } } } }
quantum{ region{ Delta{ dispersion{ lines{ } } } } }
quantum{ region{ HH{ dispersion{ lines{ } } } } }
quantum{ region{ LH{ dispersion{ lines{ } } } } }
quantum{ region{ SO{ dispersion{ lines{ } } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates dispersions along some predefined paths of high symmetry in k-space, e.g. [100], [110], [111] and their equivalents (in total maximally 13).

---

dispersion{ lines{ name } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ lines{ name = ... } } } } }
quantum{ region{ L{ dispersion{ lines{ name = ... } } } } }
quantum{ region{ X{ dispersion{ lines{ name = ... } } } } }
quantum{ region{ Delta{ dispersion{ lines{ name = ... } } } } }
quantum{ region{ HH{ dispersion{ lines{ name = ... } } } } }
quantum{ region{ LH{ dispersion{ lines{ name = ... } } } } }
quantum{ region{ SO{ dispersion{ lines{ name = ... } } } } }
```

Properties

- **usage:** required
- **type:** character string

Functionality

Is a name of the dispersions which also defines the names of the output files.

---

dispersion{ lines{ k_max } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ lines{ k_max = ... } } } } }
quantum{ region{ L{ dispersion{ lines{ k_max = ... } } } } }
quantum{ region{ X{ dispersion{ lines{ k_max = ... } } } } }
quantum{ region{ Delta{ dispersion{ lines{ k_max = ... } } } } }
quantum{ region{ HH{ dispersion{ lines{ k_max = ... } } } } }
quantum{ region{ LH{ dispersion{ lines{ k_max = ... } } } } }
quantum{ region{ SO{ dispersion{ lines{ k_max = ... } } } } }
```

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Functionality

Specifies a maximum absolute value (radius) for the k-vector in nm^{-1}.

---

dispersion{ lines{ spacing } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ lines{ spacing = ... } } } } }
quantum{ region{ L{ dispersion{ lines{ spacing = ... } } } } }
quantum{ region{ X{ dispersion{ lines{ spacing = ... } } } } }
quantum{ region{ Delta{ dispersion{ lines{ spacing = ... } } } } }
quantum{ region{ HH{ dispersion{ lines{ spacing = ... } } } } }
quantum{ region{ LH{ dispersion{ lines{ spacing = ... } } } } }
quantum{ region{ SO{ dispersion{ lines{ spacing = ... } } } } }
```

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Functionality

Specifies approximate spacing for intermediate points in the path segments in nm^{-1}.

---

dispersion{ full{ } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ full{ } } } } }
quantum{ region{ L{ dispersion{ full{ } } } } }
quantum{ region{ X{ dispersion{ full{ } } } } }
quantum{ region{ Delta{ dispersion{ full{ } } } } }
quantum{ region{ HH{ dispersion{ full{ } } } } }
quantum{ region{ LH{ dispersion{ full{ } } } } }
quantum{ region{ SO{ dispersion{ full{ } } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates dispersion in 1D/2D/3D k-space depending on simulation dimensionality and pereodic boundary conditions.

dispersion{ full{ name } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ full{ name = ... } } } } }
quantum{ region{ L{ dispersion{ full{ name = ... } } } } }
quantum{ region{ X{ dispersion{ full{ name = ... } } } } }
quantum{ region{ Delta{ dispersion{ full{ name = ... } } } } }
quantum{ region{ HH{ dispersion{ full{ name = ... } } } } }
quantum{ region{ LH{ dispersion{ full{ name = ... } } } } }
quantum{ region{ SO{ dispersion{ full{ name = ... } } } } }
```

Properties

- **usage:** required
- **type:** character string

Functionality

Is a name of the dispersion which also defines the name of the output file.

---

dispersion{ full{ kxgrid{ }, … } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kzgrid{ } } } } } }
quantum{ region{ L{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ L{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ L{ dispersion{ full{ kzgrid{ } } } } } }
quantum{ region{ X{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ X{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ X{ dispersion{ full{ kzgrid{ } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kzgrid{ } } } } } }
quantum{ region{ HH{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ HH{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ HH{ dispersion{ full{ kzgrid{ } } } } } }
quantum{ region{ LH{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ LH{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ LH{ dispersion{ full{ kzgrid{ } } } } } }
quantum{ region{ SO{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ SO{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ SO{ dispersion{ full{ kzgrid{ } } } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Specifies a `grid{...}` in k-space for a 1D/2D/3D plot of the energy dispersion E(kx, ky, kz). Allowed only, if simulation is periodic along respective direction and current quantum region extends over the entire domain.

---

dispersion{ full{ kxgrid{ line{ } }, ... } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kzgrid{ line{ } } } } } } }
quantum{ region{ L{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ L{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ L{ dispersion{ full{ kzgrid{ line{ } } } } } } }
quantum{ region{ X{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ X{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ X{ dispersion{ full{ kzgrid{ line{ } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kzgrid{ line{ } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kzgrid{ line{ } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kzgrid{ line{ } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kzgrid{ line{ } } } } } } }
```

Properties

- **usage:** required
- **items:** minimum 2

Functionality

—

---

dispersion{ full{ kxgrid{ line{ pos } }, ... } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ full{ kxgrid{ line{ pos = ... } } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kygrid{ line{ pos = ... } } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kzgrid{ line{ pos = ... } } } } } } }
quantum{ region{ L{ dispersion{ full{ kxgrid{ line{ pos = ... } } } } } } }
quantum{ region{ L{ dispersion{ full{ kygrid{ line{ pos = ... } } } } } } }
quantum{ region{ L{ dispersion{ full{ kzgrid{ line{ pos = ... } } } } } } }
quantum{ region{ X{ dispersion{ full{ kxgrid{ line{ pos = ... } } } } } } }
```

```
quantum{ region{ X{ dispersion{ full{ kygrid{ line{ pos = ... } } } } } } }
quantum{ region{ X{ dispersion{ full{ kzgrid{ line{ pos = ... } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kxgrid{ line{ pos = ... } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kygrid{ line{ pos = ... } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kzgrid{ line{ pos = ... } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kxgrid{ line{ pos = ... } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kygrid{ line{ pos = ... } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kzgrid{ line{ pos = ... } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kxgrid{ line{ pos = ... } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kygrid{ line{ pos = ... } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kzgrid{ line{ pos = ... } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kxgrid{ line{ pos = ... } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kygrid{ line{ pos = ... } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kzgrid{ line{ pos = ... } } } } } } }
```

Properties

- **usage:** required
- **type:** real number
- **values:** no constraints
- **unit:** nm^{-1}

Functionality

—

---

dispersion{ full{ kxgrid{ line{ spacing } }, … } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ full{ kxgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kygrid{ line{ spacing = ... } } } } } } }
quantum{ region{ Gamma{ dispersion{ full{ kzgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ L{ dispersion{ full{ kxgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ L{ dispersion{ full{ kygrid{ line{ spacing = ... } } } } } } }
quantum{ region{ L{ dispersion{ full{ kzgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ X{ dispersion{ full{ kxgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ X{ dispersion{ full{ kygrid{ line{ spacing = ... } } } } } } }
quantum{ region{ X{ dispersion{ full{ kzgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kxgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kygrid{ line{ spacing = ... } } } } } } }
quantum{ region{ Delta{ dispersion{ full{ kzgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kxgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kygrid{ line{ spacing = ... } } } } } } }
quantum{ region{ HH{ dispersion{ full{ kzgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kxgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kygrid{ line{ spacing = ... } } } } } } }
quantum{ region{ LH{ dispersion{ full{ kzgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kxgrid{ line{ spacing = ... } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kygrid{ line{ spacing = ... } } } } } } }
quantum{ region{ SO{ dispersion{ full{ kzgrid{ line{ spacing = ... } } } } } } }
```

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** nm^{-1}

Functionality

—

---

dispersion{ superlattice{ } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ superlattice{ } } } } }
quantum{ region{ L{ dispersion{ superlattice{ } } } } }
quantum{ region{ X{ dispersion{ superlattice{ } } } } }
quantum{ region{ Delta{ dispersion{ superlattice{ } } } } }
quantum{ region{ HH{ dispersion{ superlattice{ } } } } }
quantum{ region{ LH{ dispersion{ superlattice{ } } } } }
quantum{ region{ SO{ dispersion{ superlattice{ } } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Is a convenience group to calculate superlattice dispersion $E(k_{SL})$ along periodic directions. The intervals are set automatically to $[-\pi/L_i, \pi/L_i]$, where L_i is the simulation domain range along periodic directions with $i = x, y, z$.

---

dispersion{ superlattice{ name } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ superlattice{ name = ... } } } } }
quantum{ region{ L{ dispersion{ superlattice{ name = ... } } } } }
quantum{ region{ X{ dispersion{ superlattice{ name = ... } } } } }
quantum{ region{ Delta{ dispersion{ superlattice{ name = ... } } } } }
quantum{ region{ HH{ dispersion{ superlattice{ name = ... } } } } }
quantum{ region{ LH{ dispersion{ superlattice{ name = ... } } } } }
quantum{ region{ SO{ dispersion{ superlattice{ name = ... } } } } }
```

Properties

- **usage:** required
- **type:** character string

Functionality

Is a name of the dispersion which also defines the name of the output file.

dispersion{ superlattice{ num_points } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ superlattice{ num_points = ... } } } } }
quantum{ region{ L{ dispersion{ superlattice{ num_points = ... } } } } }
quantum{ region{ X{ dispersion{ superlattice{ num_points = ... } } } } }
quantum{ region{ Delta{ dispersion{ superlattice{ num_points = ... } } } } }
quantum{ region{ HH{ dispersion{ superlattice{ num_points = ... } } } } }
quantum{ region{ LH{ dispersion{ superlattice{ num_points = ... } } } } }
quantum{ region{ SO{ dispersion{ superlattice{ num_points = ... } } } } }
```

Properties

- **usage:** conditional
- **type:** integer
- **values:** $z \geq 2$

Functionality

Is a convenience keyword to specifies number of points along all appropriate directions in k space.

dispersion{ superlattice{ num_points_x, … } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ superlattice{ num_points_x = ... } } } } }
quantum{ region{ Gamma{ dispersion{ superlattice{ num_points_y = ... } } } } }
quantum{ region{ Gamma{ dispersion{ superlattice{ num_points_z = ... } } } } }
quantum{ region{ L{ dispersion{ superlattice{ num_points_x = ... } } } } }
quantum{ region{ L{ dispersion{ superlattice{ num_points_y = ... } } } } }
quantum{ region{ L{ dispersion{ superlattice{ num_points_z = ... } } } } }
quantum{ region{ X{ dispersion{ superlattice{ num_points_x = ... } } } } }
quantum{ region{ X{ dispersion{ superlattice{ num_points_y = ... } } } } }
quantum{ region{ X{ dispersion{ superlattice{ num_points_z = ... } } } } }
quantum{ region{ Delta{ dispersion{ superlattice{ num_points_x = ... } } } } }
quantum{ region{ Delta{ dispersion{ superlattice{ num_points_y = ... } } } } }
quantum{ region{ Delta{ dispersion{ superlattice{ num_points_z = ... } } } } }
quantum{ region{ HH{ dispersion{ superlattice{ num_points_x = ... } } } } }
quantum{ region{ HH{ dispersion{ superlattice{ num_points_y = ... } } } } }
quantum{ region{ HH{ dispersion{ superlattice{ num_points_z = ... } } } } }
quantum{ region{ LH{ dispersion{ superlattice{ num_points_x = ... } } } } }
quantum{ region{ LH{ dispersion{ superlattice{ num_points_y = ... } } } } }
quantum{ region{ LH{ dispersion{ superlattice{ num_points_z = ... } } } } }
quantum{ region{ SO{ dispersion{ superlattice{ num_points_x = ... } } } } }
quantum{ region{ SO{ dispersion{ superlattice{ num_points_y = ... } } } } }
quantum{ region{ SO{ dispersion{ superlattice{ num_points_z = ... } } } } }
```

Properties

- **usage:** conditional
- **type:** integer
- **values:** $z \geq 2$

Functionality

Specifies number of points along x direction in **k** space where dispersion is calculated. The simulation must be periodic along the x, y, or z directions in the position space.

dispersion{ output_dispersions{ } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ output_dispersions{ } } } } }
quantum{ region{ L{ dispersion{ output_dispersions{ } } } } }
quantum{ region{ X{ dispersion{ output_dispersions{ } } } } }
quantum{ region{ Delta{ dispersion{ output_dispersions{ } } } } }
quantum{ region{ HH{ dispersion{ output_dispersions{ } } } } }
quantum{ region{ LH{ dispersion{ output_dispersions{ } } } } }
quantum{ region{ SO{ dispersion{ output_dispersions{ } } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs all defined dispersions.

dispersion{ output_dispersions{ max_num } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ output_dispersions{ max_num = ... } } } } }
quantum{ region{ L{ dispersion{ output_dispersions{ max_num = ... } } } } }
quantum{ region{ X{ dispersion{ output_dispersions{ max_num = ... } } } } }
quantum{ region{ Delta{ dispersion{ output_dispersions{ max_num = ... } } } } }
quantum{ region{ HH{ dispersion{ output_dispersions{ max_num = ... } } } } }
quantum{ region{ LH{ dispersion{ output_dispersions{ max_num = ... } } } } }
quantum{ region{ SO{ dispersion{ output_dispersions{ max_num = ... } } } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 9999$
- **default:** not defined

Functionality

It is a maximum number of bands to print out.

---

dispersion{ output_masses{ } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ output_masses{ } } } } }
quantum{ region{ L{ dispersion{ output_masses{ } } } } }
quantum{ region{ X{ dispersion{ output_masses{ } } } } }
quantum{ region{ Delta{ dispersion{ output_masses{ } } } } }
quantum{ region{ HH{ dispersion{ output_masses{ } } } } }
quantum{ region{ LH{ dispersion{ output_masses{ } } } } }
quantum{ region{ SO{ dispersion{ output_masses{ } } } } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs effective masses m^* calculated from the dispersions, expressed in masses of a free electron m_0, following the formula:

$$\frac{1}{m^*} = \frac{m_0}{\hbar^2} \cdot \frac{\partial^2}{\partial k^2} E\left(k\right),$$

where k is a “distance” along the path onto which the related band structure is computed.

---

dispersion{ output_masses{ max_num } }

Calling sequence

```
quantum{ region{ Gamma{ dispersion{ output_masses{ max_num = ... } } } } }
quantum{ region{ L{ dispersion{ output_masses{ max_num = ... } } } } }
quantum{ region{ X{ dispersion{ output_masses{ max_num = ... } } } } }
quantum{ region{ Delta{ dispersion{ output_masses{ max_num = ... } } } } }
quantum{ region{ HH{ dispersion{ output_masses{ max_num = ... } } } } }
quantum{ region{ LH{ dispersion{ output_masses{ max_num = ... } } } } }
quantum{ region{ SO{ dispersion{ output_masses{ max_num = ... } } } } }
```

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 9999$
- **default:** not defined

Functionality

It is a maximum number of bands to print out.

Last update: 27/05/2025

kp_6band{ }

Calling sequence

`quantum{ region{ kp_6band{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Solves 6-band $\mathbf{k} \cdot \mathbf{p}$ Schrödinger equation for the ** heavy, light and split-off hole** valence band. The options are the same as *Gamma{ }* with some additional options, which are

Nested keywords

- *accuracy*
- *iterations*
- *num_ev*
- *lapack{ }*
- *arpack{ }*
- *preconditioner*
- *order_polynomial*
- *order_chebyshev*
- *order_legendre*
- *cutoff*
- *abs_cutoff*
- *forward_differences*
- *kp_parameters{ }*
- *kp_parameters{ use_Luttinger_parameters }*
- *kp_parameters{ approximate_kappa } }*
- *k_integration{ }*
- *k_integration{ relative_size }*
- *k_integration{ max_symmetry }*
- *k_integration{ num_points }*
- *k_integration{ num_subpoints }*
- *k_integration{ force_k0_subspace }*
- *dispersion{ }*
- *dispersion{ lines{ } }*
- *dispersion{ lines{ name } }*
- *dispersion{ lines{ k_max } }*
- *dispersion{ lines{ spacing } }*
- *dispersion{ path{ } }*
- *dispersion{ path{ name } }*
- *dispersion{ path{ point{ } } }*
- *dispersion{ path{ point{ k } } }*
- *dispersion{ path{ spacing } }*
- *dispersion{ path{ num_points } }*
- *dispersion{ full{ } }*
- *dispersion{ full{ name } }*
- *dispersion{ full{ kxgrid{ }, … } }*
- *dispersion{ full{ kxgrid{ line{ } }, … } }*
- *dispersion{ full{ kxgrid{ line{ pos } }, … } }*
- *dispersion{ full{ kxgrid{ line{ spacing } }, … } }*
- *dispersion{ superlattice{ } }*
- *dispersion{ superlattice{ name } }*
- *dispersion{ superlattice{ num_points } }*
- *dispersion{ superlattice{ num_points_x, … } }*
- *dispersion{ output_dispersions{ } }*
- *dispersion{ output_dispersions{ max_num } }*
- *dispersion{ output_masses{ } }*
- *dispersion{ output_masses{ max_num } }*

---

accuracy

Calling sequence

`quantum{ region{ kp_6band{ accuracy = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $0.0 \leq r \leq 10^{-6}$
- **default:** $r = 10^{-10}$ for ARPACK; $r = 10^{-7}$ for ARPACK_INV and DAVIDSON
- **unit:** –

Functionality

Sets accuracy of finding eigenvalues by APRACK, ARPACK_INV, and DAVIDSON routines.

---

iterations

Calling sequence

`quantum{ region{ kp_6band{ iterations = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 100000$

Functionality

Number of iterations for eigenvalue solvers.

---

num_ev

Calling sequence

`quantum{ region{ kp_6band{ num_ev = ... } } }`

Properties

- **usage:** required
- **type:** integer
- **values:** $z \geq 1$

Functionality

Sets the number of eigenvalues to be calculated.

---

lapack{ }

Calling sequence

`quantum{ region{ kp_6band{ lapack{ } } } }`

Properties

—

Functionality

LAPACK eigensolver: solves dense matrix problem (for 1D and small 2D systems only)

---

arpack{ }

Calling sequence

`quantum{ region{ kp_6band{ arpack{ } } } }`

Properties

—

Functionality

ARPACK eigensolver (default) ARPACK should be faster for large matrices (N > 1000) where only a few eigenvalues are sought (~5-30).

---

preconditioner

Calling sequence

`quantum{ region{ kp_6band{ preconditioner = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `polynomial` or `chebyshev` or `legendre`
- **default:** `chebyshev`

Functionality

The Polynomial preconditioner is the slowest but does not require to specify cutoff energy whereas Chebyshev or Legendre preconditioners requires you to specify cutoff energy.

---

order_polynomial

Calling sequence

`quantum{ region{ kp_6band{ order_polynomial = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 0$
- **default:** $z = 20$

Functionality

Order of the polynomial used for polynomial preconditioning.

---

order_chebyshev

Calling sequence

`quantum{ region{ kp_6band{ order_chebyshev = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 20$

Functionality

Order of the polynomial used for Chebyshev preconditioning.

---

order_legendre

Calling sequence

`quantum{ region{ kp_6band{ order_legendre = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **default:** $z = 20$

Functionality

Order of the polynomial used for Legendre preconditioning.

---

cutoff

Calling sequence

`quantum{ region{ kp_6band{ cutoff = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[1e-3, ...)`
- **default:** $r = 0.3$
- **unit:** eV

Functionality

—

---

abs_cutoff

Calling sequence

`quantum{ region{ kp_6band{ abs_cutoff = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** eV

Functionality

—

---

forward_differences

Calling sequence

`quantum{ region{ kp_6band{ forward_differences = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then forward and backward differences are used for the first derivative discretization of the Kane parameter P in the the 8-band k.p Hamiltonian. By default, set to `no`, centered differences are used. This parameter might affect spurious solutions of the wave functions. See eq. (1.50) and eq. (1.51) of *PhD thesis T. Andlauer* for more details.

kp_parameters{ }

Calling sequence

`quantum{ region{ kp_6band{ kp_parameters{ } } } }`

Properties

—

Functionality

advanced manipulation of $\mathbf{k} \cdot \mathbf{p}$ parameters from the database.

Attention

The groups `use_Luttinger_parameters` and `approximate_kappa` are available only for simulations with zincblende crystal symmetry.

kp_parameters{ use_Luttinger_parameters }

Calling sequence

`quantum{ region{ kp_6band{ kp_parameters{ use_Luttinger_parameters } } } }`

Properties

—

Functionality

By default the solver uses the DKK (Dresselhaus-Kip-Kittel) parameters (L, M, N). If enabled then it uses Luttinger parameters (γ_1, γ_2, γ_3) instead.

value
`yes` or `no`

default
`no`

kp_parameters{ approximate_kappa } }

Calling sequence

`quantum{ region{ kp_6band{ kp_parameters{ approximate_kappa } } } }`

Properties

—

Functionality

By default the κ for zincblende crystal structure is taken from the database or input file. If this is enabled then the solver is forced to approximate kappa through others 6-band $\mathbf{k} \cdot \mathbf{p}$ parameters, even though kappa is given in database or input file.

value
`yes` or `no`

default
no

---

k_integration{ }

Calling sequence

`quantum{ region{ kp_6band{ k_integration{ } } } }`

Properties

—

Functionality

Provides options for integration over $\mathbf{k}_{||}$ space for $\mathbf{k}\cdot\mathbf{p}$ density calculations (for 1D and 2D only). By default the quantum mechanical charge density is calculated (`no_density = no`). Therefore, `k_integration{}` is required. If you do not need a quantum mechanical density, e.g. because you are not interested in a self-consistent simulation, the calculation is much faster if you use (`no_density = yes`). Then you can omit `k_integration{}` and only the eigenstates for $\mathbf{k}_{||} = (k_y, k_z) = (0, 0) = 0$ are calculated.

---

k_integration{ relative_size }

Calling sequence

`quantum{ region{ kp_6band{ k_integration{ relative_size } } } }`

Properties

—

Functionality

Range of $\mathbf{k}_{||}$ integration relative to size of Brillouin zone. Often a value between 0.1-0.2 is sufficient.

value
float between 0.0 and 1.0

default
1.0

---

k_integration{ max_symmetry }

Calling sequence

`quantum{ region{ kp_6band{ k_integration{ max_symmetry } } } }`

Properties

—

Functionality

If `max_symmetry = no` then the solver does not use symmetry of Brillouin zone to reduce number of $\mathbf{k}_{||}$ points.

If `max_symmetry = C2` then the solver uses up to C_2 symmetry of Brillouin zone to reduce number of $\mathbf{k}_{||}$ points.

If `max_symmetry = full` then the solver uses full symmetry of Brillouin zone to reduce number of $\mathbf{k}_{||}$ points. For example for a cubic k space the 1/8th of the zone.

value
1 or no 2 or C2 3 or full

default
full

---

k_integration{ num_points }

Calling sequence

`quantum{ region{ kp_6band{ k_integration{ num_points } } } }`

Properties

—

Functionality

number of $\mathbf{k}_{||}$ points, where Schrödinger equation has to be solved (in one direction). In 1D, the number of Schrödinger equations that have to be solved depends quadratically on `num_points`. In 2D, the number of Schrödinger equations that have to be solved depends linearly on `num_points`.

value
integer > 1

default
10

---

k_integration{ num_subpoints }

Calling sequence

`quantum{ region{ kp_6band{ k_integration{ num_subpoints } } } }`

Properties

—

Functionality

number of points between two $\mathbf{k}_{||}$ points, where wave functions and eigenvalues will be interpolated.

value
integer >= 1

default
5

k_integration{ force_k0_subspace }

Calling sequence

`quantum{ region{ kp_6band{ k_integration{ force_k0_subspace } } } }`

Properties

—

Functionality

If set to `yes`, $k_{||}$ integration in quantum{ } is modified in that only states for point $k = 0$ are computed exactly, whereas all other k points are computed in the subspace of the $k = 0$ wave functions. As a result of this approximation, computational speed is much improved (you may even be able to also enlarge the number of eigenvalues). In case you are planning to use this approximation for final results, please make sure to check whether the resulting loss of accuracy in density is acceptable.

value
`yes` or `no`

default
`no`

---

dispersion{ }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ } } } }`

Properties

—

Functionality

These groups provide keywords to define a path for computation of $\mathbf{k}_{||}$ and $\mathbf{k}_{\mathbf{superlattice}}$ (if applicable) dispersions. The energy dispersion E(**k**) along the specified paths and for the specified **k** space resolutions are completely independent from the **k** space resolution that was used within the self-consistent cycle where the k.p density has been calculated. The latter is specified in `k_integration{ }`.

---

dispersion{ lines{ } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ lines{ } } } } }`

Properties

—

Functionality

Calculates dispersions along some predefined paths of high symmetry in k-space, e.g. [100], [110], [111] and their equivalents (in total maximally 13).

dispersion{ lines{ name } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ lines{ name } } } } }`

Properties

—

Functionality

value
string

Is a name of the dispersions which also defines the names of the output files.

---

dispersion{ lines{ k_max } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ lines{ k_max } } } } }`

Properties

—

Functionality

value
float

Specifies a maximum absolute value (radius) for the k-vector in nm^{-1}.

---

dispersion{ lines{ spacing } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ lines{ spacing } } } } }`

Properties

—

Functionality

value
float

Specifies approximate spacing for intermediate points in the path segments in nm^{-1}.

---

dispersion{ path{ } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ path{ } } } } }`

Properties

—

Functionality

Calculates dispersion along custom path in k-space. Multiple instances are allowed.

---

dispersion{ path{ name } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ path{ name } } } } }`

Properties

—

Functionality

Is a name of the dispersions which also defines the names of the output files.

value
string

---

dispersion{ path{ point{ } } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ path{ point{ } } } } } }`

Properties

—

Functionality

Specifies points in the path through k-space. At least two k points have to be defined. Line between two such points is called segment.

---

dispersion{ path{ point{ k } } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ path{ point{ k } } } } } }`

Properties

—

Functionality

value
3D float vector

Is a k-point represented by vector $[k_x, k_y, k_z]$. The units are nm^{-1}.

For 1D simulation the $\mathbf{k}_{||}$ space is a $k_y - k_z$ plane so k_y, k_z can be freely choosed. k_x can only be different from zero, if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain.

for 2D simulation the $\mathbf{k}_{||}$ space is a k_z axis so k_z can be freely choosed. kx can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain. k_y can only be different from zero if a periodic boundary condition along the y-direction is defined and the quantum region extends over the whole y-domain.

for 3D simulation the $\mathbf{k}_{||}$ space is empty. k_x can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain. k_y can only be different from zero if a periodic boundary condition along the y-direction is defined and the quantum region extends over the whole y-domain. k_z can only be different from zero if a periodic boundary condition along the z-direction is defined and the quantum region extends over the whole z-domain.

---

dispersion{ path{ spacing } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ path{ spacing } } } } }`

Properties

—

Functionality

value
float

Specifies approximate spacing for intermediate points in the path segments in nm^{-1}. Excludes `num_points`.

---

dispersion{ path{ num_points } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ path{ num_points } } } } }`

Properties

—

Functionality

value
integer > 1

Specifies number of points (intermediate + two corner points) for each single path segment. Excludes `spacing`.

---

dispersion{ full{ } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ full{ } } } } }`

Properties

—

Functionality

Calculates dispersion in 1D/2D/3D k-space depending on simulation dimensionality and pereodic boundary conditions.

---

dispersion{ full{ name } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ full{ name } } } } }`

Properties

—

Functionality

value
string

Is a name of the dispersion which also defines the name of the output file.

---

dispersion{ full{ kxgrid{ }, … } }

Calling sequence

```
quantum{ region{ kp_6band{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kzgrid{ } } } } } }
```

Properties

—

Functionality

Specifies a `grid{...}` in k-space for a 1D/2D/3D plot of the energy dispersion E(kx, ky, kz). Allowed only, if simulation is periodic along x-direction and current quantum region extends over the whole x-domain. The options are same as *grid{ }*

---

dispersion{ full{ kxgrid{ line{ } }, … } }

Calling sequence

```
quantum{ region{ kp_6band{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kzgrid{ line{ } } } } } } }
```

Properties

—

Functionality

—

---

dispersion{ full{ kxgrid{ line{ pos } }, … } }

Calling sequence

```
quantum{ region{ kp_6band{ dispersion{ full{ kxgrid{ line{ pos } } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kygrid{ line{ pos } } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kzgrid{ line{ pos } } } } } } }
```

Properties

—

Functionality

—

---

dispersion{ full{ kxgrid{ line{ spacing } }, … } }

Calling sequence

```
quantum{ region{ kp_6band{ dispersion{ full{ kxgrid{ line{ spacing } } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kygrid{ line{ spacing } } } } } } }
quantum{ region{ kp_6band{ dispersion{ full{ kzgrid{ line{ spacing } } } } } } }
```

Properties

—

Functionality

—

---

dispersion{ superlattice{ } }

Calling sequence

```
quantum{ region{ kp_6band{ dispersion{ superlattice{ } } } } }
```

Properties

—

Functionality

Is a convenience group to calculate superlattice dispersion $E(k_{SL})$ along periodic directions. The intervals are set automatically to $[-\pi/L_i, \pi/L_i]$, where L_i is the simulation domain range along periodic directions with $i = x, y, z$.

dispersion{ superlattice{ name } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ superlattice{ name } } } } }`

Properties

—

Functionality

value
string

Is a name of the dispersion which also defines the name of the output file.

---

dispersion{ superlattice{ num_points } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ superlattice{ num_points } } } } }`

Properties

—

Functionality

Is a convenience keyword to specifies number of points along all appropriate directions in k space.

value
any integer > 1

---

dispersion{ superlattice{ num_points_x, … } }

Calling sequence

```
quantum{ region{ kp_6band{ dispersion{ superlattice{ num_points_x } } } } }
quantum{ region{ kp_6band{ dispersion{ superlattice{ num_points_y } } } } }
quantum{ region{ kp_6band{ dispersion{ superlattice{ num_points_z } } } } }
```

Properties

—

Functionality

value
any integer > 1

Specifies number of points along x direction in **k** space where dispersion is calculated. The simulation must be periodic along the x direction in direct space. Specifies number of points along y direction in **k** space where dispersion is calculated. The simulation must be periodic along the y direction in direct space. Specifies number of points along z direction in **k** space where dispersion is calculated. The simulation must be periodic along the z direction in direct space.

---

dispersion{ output_dispersions{ } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ output_dispersions{ } } } } }`

Properties

—

Functionality

Outputs all defined dispersions.

---

dispersion{ output_dispersions{ max_num } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ output_dispersions{ max_num } } } } }`

Properties

—

Functionality

Is a number of bands to print out

value
any integer between 1 and 9999

---

dispersion{ output_masses{ } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ output_masses{ } } } } }`

Properties

—

Functionality

Outputs effective masses m^* calculated from the dispersions, expressed in masses of a free electron m_0, following the formula:

$$\frac{1}{m^*} = \frac{m_0}{\hbar^2} \cdot \frac{\partial^2}{\partial k^2} E\left(k\right),$$

where k is a “distance” along the path onto which the related band structure is computed.

---

dispersion{ output_masses{ max_num } }

Calling sequence

`quantum{ region{ kp_6band{ dispersion{ output_masses{ max_num } } } } }`

Properties

—

Functionality

Outputs effective masses calculated from the dispersions.

value
any integer between 1 and 9999

Last update: 27/05/2025

kp_8band{ }

Calling sequence

`quantum{ region{ kp_8band{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Triggers solver of 8-band $\mathbf{k} \cdot \mathbf{p}$ Schrödinger equation for the Gamma conduction band and the heavy, light and split-off hole valence bands.

Nested keywords

- *accuracy*
- *iterations*
- *num_electrons*
- *num_holes*
- *shift*
- *abs_shift*
- *shift_window*
- *lapack{ }*
- *arpack_inv{ }*
- *davidson{ }*
- *forward_differences*
- *electron_far_band*
- *correct_electron_gfactor*
- *rescale_kp_everywhere*
- *avoid_spurious*
- *kp_parameters{ }*
- *kp_parameters{ use_Luttinger_parameters }*
- *kp_parameters{ from_6band_parameters }*
- *kp_parameters{ approximate_kappa }*
- *kp_parameters{ evaluate_S }*
- *kp_parameters{ rescale_S_to }*
- *k_integration{ }*
- *k_integration{ relative_size }*
- *k_integration{ max_symmetry }*
- *k_integration{ num_points }*
- *k_integration{ num_subpoints }*
- *k_integration{ force_k0_subspace }*
- *interface{ }*
- *interface{ position }*
- *interface{ array_x{ } }*
- *interface{ array_x{ shift } }*
- *interface{ array_x{ min } }*
- *interface{ array_x{ max } }*
- *interface{ kp_parameters{ } }*
- *interface{ kp_parameters{ D_s, D_x, D_z } }*
- *interface{ kp_parameters{ alpha, beta } }*
- *interface{ kp_parameters{ reverse } }*
- *dispersion{ }*
- *dispersion{ full{ } }*
- *dispersion{ full{ name } }*
- *dispersion{ full{ kxgrid{ }, … } }*
- *dispersion{ full{ kxgrid{ line{ } }, … } }*
- *dispersion{ full{ kxgrid{ line{ pos } }, … } }*
- *dispersion{ full{ kxgrid{ line{ spacing } }, … } }*
- *dispersion{ path{ } }*
- *dispersion{ path{ name } }*
- *dispersion{ path{ point{ } } }*
- *dispersion{ path{ point{ k } } }*
- *dispersion{ path{ spacing } }*
- *dispersion{ path{ num_points } }*
- *dispersion{ lines{ } }*
- *dispersion{ lines{ name } }*
- *dispersion{ lines{ k_max } }*
- *dispersion{ lines{ spacing } }*
- *dispersion{ superlattice{ } }*
- *dispersion{ superlattice{ name } }*
- *dispersion{ superlattice{ num_points } }*
- *dispersion{ superlattice{ num_points_x, … } }*
- *dispersion{*

output_dispersions{ } }
- *dispersion{ output_dispersions{ max_num } }*
- *dispersion{ output_masses{ } }*
- *dispersion{ output_masses{ max_num } }*
- *classify_none{ }*
- *classify_by_energy{ }*
- *classify_by_energy{ method }*
- *classify_by_energy{ shift_electrons }*
- *classify_by_energy{ shift_holes }*
- *classify_by_energy{ cutoff }*
- *classify_by_all_energies{ }*
- *classify_by_all_energies{ method }*
- *classify_by_all_energies{ shift_electrons }*
- *classify_by_all_energies{ shift_holes }*
- *classify_by_all_energies{ permissive }*
- *classify_by_all_energies{ cutoff }*
- *classify_by_spinor{ }*
- *classify_by_spinor{ threshold_electron }*
- *classify_by_spinor{ threshold_hole }*
- *classify_by_spinor{ cutoff }*
- *classify_by_all_spinors{ }*
- *classify_by_all_spinors{ threshold_electron }*
- *classify_by_all_spinors{ threshold_hole }*
- *classify_by_all_spinors{ permissive }*
- *classify_by_all_spinors{ cutoff }*
- *linear_solver{ }*
- *linear_solver{ iterations }*
- *linear_solver{ abs_accuracy }*
- *linear_solver{ rel_accuracy }*
- *linear_solver{ use_cscg }*
- *linear_solver{ force_diagonal_preconditioner }*

---

accuracy

Calling sequence

`quantum{ region{ kp_8band{ accuracy } } }`

Properties

—

Functionality

value
any float > 0

default
1e-7

accuracy of eigenvalue

---

iterations

Calling sequence

`quantum{ region{ kp_8band{ iterations } } }`

Properties

—

Functionality

value
any integer > 1

default
500

number of iterations for eigenvalue solver

---

num_electrons

Calling sequence

`quantum{ region{ kp_8band{ num_electrons } } }`

Properties

—

Functionality

value
integer >= 0

default
0

number of electron eigenvalues

---

num_holes

Calling sequence

`quantum{ region{ kp_8band{ num_holes } } }`

Properties

—

Functionality

value
integer >= 0

default
0

number of hole eigenvalues

---

shift

Calling sequence

`quantum{ region{ kp_8band{ shift } } }`

Properties

—

Functionality

value
float >=0

default
`0.1 # (eV)`

energy shift relative to band edges in `arpack_inv`.

---

abs_shift

Calling sequence

`quantum{ region{ kp_8band{ abs_shift } } }`

Properties

—

Functionality

value
float >=0

default
`0.0` # `(eV)`

energy shift on an absolute energy scale in `arpack_inv`.

shift_window

Calling sequence

`quantum{ region{ kp_8band{ shift_window } } }`

Properties

—

Functionality

value
integer

default
0

When LAPACK is used, shifts the window of computed states by the specified number of states up (for positive integers) or down (for negative integers). Adjust when the computed states are not centered around the band gap.

lapack{ }

Calling sequence

`quantum{ region{ kp_8band{ lapack{ } } } }`

Properties

—

Functionality

Triggers use of LAPACK solver which is for dense matrix problems (for 1D and small 2D systems only)

arpack_inv{ }

Calling sequence

`quantum{ region{ kp_8band{ arpack_inv{ } } } }`

Properties

—

Functionality

Triggers use of ARPACK shift invert eigensolver. It should be faster for large matrices (N > 1000) where only a few eigenvalues are sought (~5-30).

---

davidson{ }

Calling sequence

`quantum{ region{ kp_8band{ davidson{ } } } }`

Properties

—

Functionality

Triggers Davidson solver for 8-band $\mathbf{k} \cdot \mathbf{p}$. It offers both better speed as well as increased stability compared to ARPACK inverse in 2D and 3D.

⚠ **Warning**

The implementation of teh Davidson solver is still under development, therefore, should be considered as an experimental feature.

For example, it has the tendency to fail in the presence of degenerate eigenvalues (e.g. Pauli or k.p quantum mechanics without magnetic field). In this case, breaking the degeneracies by slightly changing the geometry of the system or adding a weak magnetic field can be tried. Alternatively, switching back to ARPAPCK inverse or, in 1D or smaller 2D systems, to LAPACK may be considered.

---

forward_differences

Calling sequence

`quantum{ region{ kp_8band{ forward_differences = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then forward and backward differences are used for the first derivative discretization of the Kane parameter P in the the 8-band k.p Hamiltonian. By default, set to `no`, centered differences are used. This parameter might affect spurious solutions of the wave functions. See eq. (1.50) and eq. (1.51) of *PhD thesis T. Andlauer* for more details.

---

electron_far_band

Calling sequence

`quantum{ region{ kp_8band{ electron_far_band = ... } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** $-$

Dependencies

—

Functionality

Far-band contribution to electrons $S = 1.0 + r$. The default results in rescaling such that $S = 1.0$.

Note

It can be useful to set this value to `r = -1.0` which then corresponds to setting $S = 0.0$.

---

correct_electron_gfactor

Calling sequence

`quantum{ region{ kp_8band{ correct_electron_gfactor = ... } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = -1.0$

Dependencies

—

Functionality

When $r < 0$ then the g-factor is set to 2.
When $r = 0$ then the g-factor is computed.
When $r > 0$ then the g-factor is computed assuming energy gap equal r.
See more details in *Zeeman Term*.

---

rescale_kp_everywhere

Calling sequence

`quantum{ region{ kp_8band{ rescale_kp_everywhere } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Dependencies

—

Functionality

If set to `yes` then N, M, and P parameters are rescaled. See more details in *Zeeman Term*.

---

avoid_spurious

Calling sequence

`quantum{ region{ kp_8band{ avoid_spurious } } }`

Properties

- **usage:** conditional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Dependencies

—

Functionality

If set to `yes` then algorithm avoiding spurious solutions is used.

kp_parameters{ }

Calling sequence

`quantum{ region{ kp_8band{ kp_parameters{ } } } }`

Properties

—

Functionality

Provides options for advanced manipulation of k.p parameters from database.

Attention

The groups `use_Luttinger_parameters` and `approximate_kappa` are available only for simulations with zincblende crystal symmetry.

---

kp_parameters{ use_Luttinger_parameters }

Calling sequence

`quantum{ region{ kp_8band{ kp_parameters{ use_Luttinger_parameters } } } }`

Properties

—

Functionality

By default the solver uses the DKK (Dresselhaus-Kip-Kittel) parameters (L, M, N). If enabled then it uses Luttinger parameters (γ_1, γ_2, γ_3) instead.

value
`yes` or `no`

default
`no`

---

kp_parameters{ from_6band_parameters }

Calling sequence

`quantum{ region{ kp_8band{ kp_parameters{ from_6band_parameters } } } }`

Properties

—

Functionality

By default the 8-band $\mathbf{k} \cdot \mathbf{p}$ parameters are taken from database or input file. If enabled then it evaluates the 8-band $\mathbf{k} \cdot \mathbf{p}$ parameters from 6-band $\mathbf{k} \cdot \mathbf{p}$ parameters, Kane parameter

E_P and temperature dependent band gap E_g. :value: `yes` or `no` :default: `no`

---

kp_parameters{ approximate_kappa }

Calling sequence

`quantum{ region{ kp_8band{ kp_parameters{ approximate_kappa } } } }`

Properties

—

Functionality

By default the κ for zinc blende crystal structure is taken from the database or input file. If this is enabled then the solver is forced to approximate kappa through others 8-band $\mathbf{k} \cdot \mathbf{p}$ parameters, even though kappa is given in database or input file.

value
`yes` or `no`

default
`no`

---

kp_parameters{ evaluate_S }

Calling sequence

`quantum{ region{ kp_8band{ kp_parameters{ evaluate_S } } } }`

Properties

—

Functionality

By default S (S_1, S_2 for wurtzite) $\mathbf{k} \cdot \mathbf{p}$ parameter(s) is (are) taken from database or input file. If enabled it evaluates S (S_1, S_2 for wurtzite) $\mathbf{k} \cdot \mathbf{p}$ parameter(s) from effective mass m_e ($m_{e,par}$, $m_{e,perp}$ for wurtzite), Kane parameter(s), spin-orbit coupling(s) and temperature dependent band gap.

value
`yes` or `no`

default
`no`

---

kp_parameters{ rescale_S_to }

Calling sequence

`quantum{ region{ kp_8band{ kp_parameters{ rescale_S_to } } } }`

Properties

—

Functionality

set S for zinc blende crystal structure to specified value and rescale E_P, L', N^+ in order to preserve electron's effective mass.

set S_1, S_2 for wurtzite crystal structure to specified values respectively and rescale E_{P1}, E_{P2}, L'_1, L'_2, N_1^+, N_2^+ in order to preserve electron's effective masses.

value
float for zinc blende crystal structure

2D float vector for wurtzite crystal structure

---

k_integration{ }

Calling sequence

`quantum{ region{ kp_8band{ k_integration{ } } } }`

Properties

—

Functionality

Provides options for integration over $\mathbf{k}_{||}$ space for $\mathbf{k} \cdot \mathbf{p}$ density calculations (for 1D and 2D only).

---

k_integration{ relative_size }

Calling sequence

`quantum{ region{ kp_8band{ k_integration{ relative_size } } } }`

Properties

—

Functionality

—

---

k_integration{ max_symmetry }

Calling sequence

`quantum{ region{ kp_8band{ k_integration{ max_symmetry } } } }`

Properties

—

Functionality

—

---

k_integration{ num_points }

Calling sequence

`quantum{ region{ kp_8band{ k_integration{ num_points } } } }`

Properties

—

Functionality

—

---

k_integration{ num_subpoints }

Calling sequence

`quantum{ region{ kp_8band{ k_integration{ num_subpoints } } } }`

Properties

—

Functionality

—

---

k_integration{ force_k0_subspace }

Calling sequence

`quantum{ region{ kp_8band{ k_integration{ force_k0_subspace } } } }`

Properties

—

Functionality

—

---

interface{ }

Calling sequence

`quantum{ region{ kp_8band{ interface{ } } } }`

Properties

—

Functionality

Note

Better description will be available soon.

Optional group to add interface effects to the Hamiltonian [*LivnehPRB2012*], [*LivnehPRB2014*]. It can be used multiple times.

interface{ position }

Calling sequence

`quantum{ region{ kp_8band{ interface{ position } } } }`

Properties

—

Functionality

A real number defining position of the interface.

---

interface{ array_x{ } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ array_x{ } } } } }`

Properties

—

Functionality

The group that copies the interface object along the simulation axis.

---

interface{ array_x{ shift } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ array_x{ shift } } } } }`

Properties

—

Functionality

value
a real number

---

interface{ array_x{ min } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ array_x{ min } } } } }`

Properties

—

Functionality

value
`{..., -3, -2, -1 , 0}`

default
0

---

interface{ array_x{ max } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ array_x{ max } } } } }`

Properties

—

Functionality

value
`{0, 1, 2, 3, ...}`

---

interface{ kp_parameters{ } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ kp_parameters{ } } } } }`

Properties

—

Functionality

The group storing all parameters for the interface Hamiltonian.

---

interface{ kp_parameters{ D_s, D_x, D_z } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ kp_parameters{ D_s } } } } }`
`quantum{ region{ kp_8band{ interface{ kp_parameters{ D_x } } } } }`
`quantum{ region{ kp_8band{ interface{ kp_parameters{ D_z } } } } }`

Properties

—

Functionality

a real number

interface{ kp_parameters{ alpha, beta } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ kp_parameters{ alpha } } } } }`
`quantum{ region{ kp_8band{ interface{ kp_parameters{ beta } } } } }`

Properties

—

Functionality

a real number

---

interface{ kp_parameters{ reverse } }

Calling sequence

`quantum{ region{ kp_8band{ interface{ kp_parameters{ reverse } } } } }`

Properties

—

Functionality

— choice (yes/no)

---

dispersion{ }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ } } } }`

Properties

—

Functionality

These groups provide keywords to define a path for computation of $\mathbf{k}_{||}$ and $\mathbf{k}_{\mathbf{superlattice}}$ (if applicable) dispersions. The energy dispersion E(**k**) along the specified paths and for the specified **k** space resolutions are completely independent from the **k** space resolution that was used within the self-consistent cycle where the k.p density has been calculated. The latter is specified in `k_integration{ }`.

---

dispersion{ full{ } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ full{ } } } } }`

Properties

—

Functionality

Calculates dispersion in 1D/2D/3D k-space depending on simulation dimensionality and pereodic boundary conditions.

---

dispersion{ full{ name } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ full{ name } } } } }`

Properties

—

Functionality

value
string

Is a name of the dispersion which also defines the name of the output file.

---

dispersion{ full{ kxgrid{ }, … } }

Calling sequence

```
quantum{ region{ kp_8band{ dispersion{ full{ kxgrid{ } } } } } }
quantum{ region{ kp_8band{ dispersion{ full{ kygrid{ } } } } } }
quantum{ region{ kp_8band{ dispersion{ full{ kzgrid{ } } } } } }
```

Properties

—

Functionality

Specifies a `grid{...}` in k-space for a 1D/2D/3D plot of the energy dispersion E(kx, ky, kz). Allowed only, if simulation is periodic along x-direction and current quantum region extends over the whole x-domain. The options are same as *grid{ }*

---

dispersion{ full{ kxgrid{ line{ } }, … } }

Calling sequence

```
quantum{ region{ kp_8band{ dispersion{ full{ kxgrid{ line{ } } } } } } }
quantum{ region{ kp_8band{ dispersion{ full{ kygrid{ line{ } } } } } } }
quantum{ region{ kp_8band{ dispersion{ full{ kzgrid{ line{ } } } } } } }
```

Properties

—

Functionality

—

---

dispersion{ full{ kxgrid{ line{ pos } }, . . . } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ full{ kxgrid{ line{ pos } } } } } } }`
`quantum{ region{ kp_8band{ dispersion{ full{ kygrid{ line{ pos } } } } } } }`
`quantum{ region{ kp_8band{ dispersion{ full{ kzgrid{ line{ pos } } } } } } }`

Properties

—

Functionality

—

---

dispersion{ full{ kxgrid{ line{ spacing } }, . . . } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ full{ kxgrid{ line{ spacing } } } } } } }`
`quantum{ region{ kp_8band{ dispersion{ full{ kygrid{ line{ spacing } } } } } } }`
`quantum{ region{ kp_8band{ dispersion{ full{ kzgrid{ line{ spacing } } } } } } }`

Properties

—

Functionality

—

---

dispersion{ path{ } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ path{ } } } } }`

Properties

—

Functionality

Calculates dispersion along custom path in k-space. Multiple instances are allowed.

dispersion{ path{ name } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ path{ name } } } } }`

Properties

—

Functionality

Is a name of the dispersions which also defines the names of the output files.

value
string

---

dispersion{ path{ point{ } } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ path{ point{ } } } } } }`

Properties

—

Functionality

Specifies points in the path through k-space. At least two k points have to be defined. Line between two such points is called segment.

---

dispersion{ path{ point{ k } } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ path{ point{ k } } } } } }`

Properties

—

Functionality

value
3D float vector

Is a k-point represented by vector $[k_x, k_y, k_z]$. The units are nm^{-1}.

For 1D simulation the $\mathbf{k}_{||}$ space is a $k_y - k_z$ plane so k_y, k_z can be freely choosed. k_x can only be different from zero, if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain.

for 2D simulation the $\mathbf{k}_{||}$ space is a k_z axis so k_z can be freely choosed. kx can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain. k_y can only be different from zero if a periodic boundary condition along the y-direction is defined and the quantum region extends over the whole y-domain.

for 3D simulation the $\mathbf{k}_{||}$ space is empty. k_x can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain. k_y can only be different from zero if a periodic boundary condition along the y-direction is defined and the quantum region extends over the whole y-domain. k_z can only be different from zero if a periodic boundary condition along the z-direction is defined and the quantum region extends over the whole z-domain.

---

dispersion{ path{ spacing } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ path{ spacing } } } } }`

Properties

—

Functionality

value
float

Specifies approximate spacing for intermediate points in the path segments in nm^{-1}. Excludes `num_points`.

---

dispersion{ path{ num_points } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ path{ num_points } } } } }`

Properties

—

Functionality

value
integer > 1

Specifies number of points (intermediate + two corner points) for each single path segment. Excludes `spacing`.

---

dispersion{ lines{ } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ lines{ } } } } }`

Properties

—

Functionality

Calculates dispersions along some predefined paths of high symmetry in k-space, e.g. [100], [110], [111] and their equivalents (in total maximally 13).

---

dispersion{ lines{ name } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ lines{ name } } } } }`

Properties

—

Functionality

value
string

Is a name of the dispersions which also defines the names of the output files.

---

dispersion{ lines{ k_max } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ lines{ k_max } } } } }`

Properties

—

Functionality

value
float

Specifies a maximum absolute value (radius) for the k-vector in nm^{-1}.

---

dispersion{ lines{ spacing } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ lines{ spacing } } } } }`

Properties

—

Functionality

value
float

Specifies approximate spacing for intermediate points in the path segments in nm^{-1}.

---

dispersion{ superlattice{ } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ superlattice{ } } } } }`

Properties

—

Functionality

Is a convenience group to calculate superlattice dispersion $E(k_{SL})$ along periodic directions. The intervals are set automatically to $[-\pi/L_i, \pi/L_i]$, where L_i is the simulation domain range along periodic directions with $i = x, y, z$.

---

dispersion{ superlattice{ name } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ superlattice{ name } } } } }`

Properties

—

Functionality

value
  string

Is a name of the dispersion which also defines the name of the output file.

---

dispersion{ superlattice{ num_points } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ superlattice{ num_points } } } } }`

Properties

—

Functionality

Is a convenience keyword to specifies number of points along all appropriate directions in k space.

value
  any integer > 1

---

dispersion{ superlattice{ num_points_x, … } }

Calling sequence

```
quantum{ region{ kp_8band{ dispersion{ superlattice{ num_points_x } } } } }
quantum{ region{ kp_8band{ dispersion{ superlattice{ num_points_y } } } } }
quantum{ region{ kp_8band{ dispersion{ superlattice{ num_points_z } } } } }
```

Properties

—

Functionality

value
any integer > 1

Specifies number of points along x direction in **k** space where dispersion is calculated. The simulation must be periodic along the x direction in direct space. Specifies number of points along y direction in **k** space where dispersion is calculated. The simulation must be periodic along the y direction in direct space. Specifies number of points along z direction in **k** space where dispersion is calculated. The simulation must be periodic along the z direction in direct space.

---

dispersion{ output_dispersions{ } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ output_dispersions{ } } } } }`

Properties

—

Functionality

Outputs all defined dispersions.

---

dispersion{ output_dispersions{ max_num } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ output_dispersions{ max_num } } } } }`

Properties

—

Functionality

Is a number of bands to print out

value
any integer between 1 and 9999

---

dispersion{ output_masses{ } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ output_masses{ } } } } }`

Properties

—

Functionality

Outputs effective masses m^* calculated from the dispersions, expressed in masses of a free electron m_0, following the formula:

$$\frac{1}{m^*} = \frac{m_0}{\hbar^2} \cdot \frac{\partial^2}{\partial k^2} E\left(k\right),$$

where k is a "distance" along the path onto which the related band structure is computed.

---

dispersion{ output_masses{ max_num } }

Calling sequence

`quantum{ region{ kp_8band{ dispersion{ output_masses{ max_num } } } } }`

Properties

—

Functionality

Outputs effective masses calculated from the dispersions.

value
any integer between 1 and 9999

---

classify_none{ }

Calling sequence

`quantum{ region{ kp_8band{ classify_none{ } } } }`

Properties

—

Functionality

—

---

classify_by_energy{ }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_energy{ } } } }`

Properties

—

Functionality

—

---

classify_by_energy{ method }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_energy{ method } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 2$

Functionality

—

---

classify_by_energy{ shift_electrons }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_energy{ shift_electrons } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

—

---

classify_by_energy{ shift_holes }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_energy{ shift_holes } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

—

classify_by_energy{ cutoff }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_energy{ cutoff } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 4$

Functionality

—

---

classify_by_all_energies{ }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_energies{ } } } }`

Properties

—

Functionality

—

---

classify_by_all_energies{ method }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_energies{ method } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 2$

Functionality

—

---

classify_by_all_energies{ shift_electrons }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_energies{ shift_electrons } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

—

---

classify_by_all_energies{ shift_holes }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_energies{ shift_holes } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

—

---

classify_by_all_energies{ permissive }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_energies{ permissive } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 2$

Functionality

—

---

classify_by_all_energies{ cutoff }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_energies{ cutoff } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 4$

Functionality

—

---

classify_by_spinor{ }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_spinor{ } } } }`

Properties

—

Functionality

—

---

classify_by_spinor{ threshold_electron }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_spinor{ threshold_electron } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-2} \leq r \leq 0.99$
- **unit:** –

Functionality

—

---

classify_by_spinor{ threshold_hole }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_spinor{ threshold_hole } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-2} \leq r \leq 0.99$
- **unit:** –

Functionality

—

---

classify_by_spinor{ cutoff }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_spinor{ cutoff } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 4$

Functionality

—

---

classify_by_all_spinors{ }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_spinors{ } } } }`

Properties

—

Functionality

—

---

classify_by_all_spinors{ threshold_electron }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_spinors{ threshold_electron } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-2} \leq r \leq 0.99$
- **unit:** $-$

Functionality

—

---

classify_by_all_spinors{ threshold_hole }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_spinors{ threshold_hole } } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-2} \leq r \leq 0.99$
- **unit:** $-$

Functionality

—

---

classify_by_all_spinors{ permissive }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_spinors{ permissive } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 2$

Functionality

—

---

classify_by_all_spinors{ cutoff }

Calling sequence

`quantum{ region{ kp_8band{ classify_by_all_spinors{ cutoff } } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 4$

Functionality

—

---

linear_solver{ }

Calling sequence

`quantum{ region{ kp_8band{ linear_solver{ } } } }`

Properties

—

Functionality

Provides parameters for linear equation solver in `arpack_inv` shift invert preconditioner

---

linear_solver{ iterations }

Calling sequence

`quantum{ region{ kp_8band{ linear_solver{ iterations } } } }`

Properties

—

Functionality

value
integer > 1

default
10000

number of iterations in `arpack_inv`. Occasionally, using even larger values than 10000 may be necessary to avoid diagonalization failure.

---

linear_solver{ abs_accuracy }

Calling sequence

`quantum{ region{ kp_8band{ linear_solver{ abs_accuracy } } } }`

Properties

—

Functionality

value
float between 0.0 and 0.01

default
1e-8

absolute accuracy in `arpack_inv`.

---

linear_solver{ rel_accuracy }

Calling sequence

`quantum{ region{ kp_8band{ linear_solver{ rel_accuracy } } } }`

Properties

—

Functionality

value
float between 0.0 and 0.01

default
1e-8

relative accuracy in `arpack_inv`.

---

linear_solver{ use_cscg }

Calling sequence

`quantum{ region{ kp_8band{ linear_solver{ use_cscg } } } }`

Properties

—

Functionality

value
`yes` or `no`

default
`no`

When `arpack_inv` is used, forces the slower but occasionally more robust CSCG (Composite Step Conjugate Gradient) linear solver to be used rather than the cg (Conjugate Gradient) linear solver. May occasionally prevent a diagonalization failure.

---

linear_solver{ force_diagonal_preconditioner }

Calling sequence

`quantum{ region{ kp_8band{ linear_solver{ force_diagonal_preconditioner } } } }`

Properties

—

Functionality

value
`yes` or `no`

default
`no`

When `arpack_inv` is used, forces the use of a slower but more robust diagonal preconditioner. As result, total runtime and stability of the arpack_inv solver may actually become much better and diagonalization failures may be avoided.

Last update: 27/05/2025

boundary{ }

Calling sequence

`quantum{ region{ boundary{ } } }`

Properties

- **usage:** conditional
- **items:** maximum 1

Dependencies

—

Functionality

Specifies the boundary condition for Schrödinger equation along various axis dimensions. In general, **Dirichlet** boundary conditions correspond to $f = \mathrm{constant}$ and **Neumann** boundary conditions correspond to $df/dx = \mathrm{constant}$. Quantum densities may exhibit pathological density values on the boundary (e.g. 0 in the case of Dirichlet boundary conditions). Using `classical_boundary_x`, `classical_boundary_y`, `classical_boundary_z`, the computation of a classical density can be enforced on the respective boundary points for the respective band(s). The calculation within the quantum model itself and respective results such as wave functions are not affected by this setting. Using `num_classical_x`, `num_classical_y`, `num_classical_z` you can explicitly specify the number of points to be cut at each side.

Nested keywords

- *x, y, z*
- *classical_boundary_x, …*
- *num_classical_x, …*

---

x, y, z

Calling sequence

```
quantum{ region{ boundary{ x = ... } } }
quantum{ region{ boundary{ y = ... } } }
quantum{ region{ boundary{ z = ... } } }
```

Properties

- **usage:** optional
- **type:** choice
- **values:** `dirichlet` or `neumann` or `shifted_neumann`
- **default:** `neumann`

Functionality

Specifies boundary conditions at the borders of respective `quantum{ region{} }` in the **x** direction of the simulation. The `dirichlet` results in Dirichlet boundary conditions. The `neumann` results in Neumann boundary conditions. The `shifted_neumann` results in Neumann boundary conditions where the flux disappears half a grid spacing outside the boundary.

Note

Periodic boundary conditions along the appropriate direction(s) are taken automatically if `global { ... periodic{ x/y/z = yes} }` is specified **and** if the quantum region extends over the whole simulation region along the appropriate direction. In this case, the `dirichlet` or `neumann` specifications under `quantum{ ... {region{ ... boundary{...} } }` are ignored along the appropriate direction(s).

---

classical_boundary_x, …

Calling sequence

```
quantum{ region{ boundary{ classical_boundary_x = ... } } }
quantum{ region{ boundary{ classical_boundary_y = ... } } }
quantum{ region{ boundary{ classical_boundary_z = ... } } }
```

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

—

---

num_classical_x, …

Calling sequence

```
quantum{ region{ boundary{ num_classical_x = [ ..., ... ] } } }
quantum{ region{ boundary{ num_classical_y = [ ..., ... ] } } }
quantum{ region{ boundary{ num_classical_z = [ ..., ... ] } } }
```

Properties

- **usage:** conditional
- **type:** vector of 2 integers: (z_1, z_2)
- **values:** $z_i \geq 0$
- **default:** $(1, 1)$

Functionality

—

Last update: 27/05/2025

overlap_integrals{ }

Calling sequence

`quantum{ region{ overlap_integrals{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Provides the option to calculate overlap integrals between wave functions of two different bands.

Nested keywords

- *KP6_Gamma{ }*
- *HH_Gamma{ }*
- *LH_Gamma{ }*
- *SO_Gamma{ }*
- *HH_Delta{ }*
- *LH_Delta{ }*
- *SO_Delta{ }*
- *HH_X{ }*
- *LH_X{ }*
- *SO_X{ }*
- *HH_L{ }*
- *LH_L{ }*
- *SO_L{ }*
- *output_matrix_elements*
- *output_transition_energies*

KP6_Gamma{ }

Calling sequence

`quantum{ region{ overlap_integrals{ KP6_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

$\sum_k \langle kp6_{k,i} | \Gamma_j \rangle$, with k = 1 .. 6 indexing the component of the six-component $\mathbf{k} \cdot \mathbf{p}$ wave function and i, j indexing the wave function numbers. `kp_6band{ }` and `Gamma{ }` calculation must be present.

HH_Gamma{ }

Calling sequence

`quantum{ region{ overlap_integrals{ HH_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the heavy hole valence band and the gamma conduction band $\langle HH_i | \Gamma_j \rangle$

LH_Gamma{ }

Calling sequence

`quantum{ region{ overlap_integrals{ LH_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the light hole valence band and the gamma conduction band $\langle LH_i | \Gamma_j \rangle$

SO_Gamma{ }

Calling sequence

`quantum{ region{ overlap_integrals{ SO_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the split-off hole valence band and the gamma conduction band $\langle SO_i | \Gamma_j \rangle$

HH_Delta{ }

Calling sequence

`quantum{ region{ overlap_integrals{ HH_Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the heavy hole valence band and the Delta conduction band $\langle LH_i | \Delta_j \rangle$

LH_Delta{ }

Calling sequence

`quantum{ region{ overlap_integrals{ LH_Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the light hole valence band and the Delta conduction band $\langle LH_i|\Delta_j\rangle$

SO_Delta{ }

Calling sequence

`quantum{ region{ overlap_integrals{ SO_Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the split-off hole valence band and the Delta conduction band $\langle SO_i|\Delta_j\rangle$

HH_X{ }

Calling sequence

`quantum{ region{ overlap_integrals{ HH_X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the heavy hole valence band and the X conduction band $\langle HH_i|X_j\rangle$

LH_X{ }

Calling sequence

`quantum{ region{ overlap_integrals{ LH_X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the light hole valence band and the X conduction band $\langle LH_i|X_j\rangle$

SO_X{ }

Calling sequence

`quantum{ region{ overlap_integrals{ SO_X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the split-off valence band and the X conduction band $\langle SO_i | X_j \rangle$

HH_L{ }

Calling sequence

`quantum{ region{ overlap_integrals{ HH_L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the heavy hole valence band and the L conduction band $\langle HH_i | L_j \rangle$

LH_L{ }

Calling sequence

`quantum{ region{ overlap_integrals{ LH_L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the light hole valence band and the L conduction band $\langle LH_i | L_j \rangle$

SO_L{ }

Calling sequence

`quantum{ region{ overlap_integrals{ SO_L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Matrix element of the transition between the split-off valence band and the L conduction band $\langle SO_i | L_j \rangle$

output_matrix_elements

Calling sequence

`quantum{ region{ overlap_integrals{ output_matrix_elements = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If `output_matrix_elements = yes` then matrix elements are saved in output file.

---

output_transition_energies

Calling sequence

`quantum{ region{ overlap_integrals{ output_transition_energies = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `output_transition_energies = yes` then transition energies are saved in output file.

Last update: 27/05/2025

momentum_matrix_elements{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculate polarization dependent momentum matrix elements $\epsilon \cdot \langle\psi_i|\hat{\mathbf{p}}|\psi_j\rangle$ for wave functions within one band. The light polarization direction ϵ is automatically normalized in the program.

For further reading: J. H. Davies, *The Physics of Low-Dimensional Semiconductors. An Introduction*, 2006, Chapters 10 and 8.

Nested keywords

- *name*
- *direction*
- *Gamma{ }*
- *X{ }*
- *Delta{ }*
- *L{ }*
- *HH{ }*
- *LH{ }*
- *SO{ }*
- *KP6{ }*
- *KP8{ }*
- *output_matrix_elements*
- *output_transition_energies*
- *output_oscillator_strengths*

---

name

Calling sequence

`quantum{ region{ momentum_matrix_elements{ name = ... } } }`

Properties

- **usage:** optional
- **type:** character string

Functionality

defines suffix for related output files

---

direction

Calling sequence

`quantum{ region{ momentum_matrix_elements{ direction = [ ..., ..., ... ] } } }`

Properties

- **usage:** optional
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **default:** $r_1 = 1.0$, $r_2 = 0.0$, $r_3 = 0.0$
- **unit:** $-$

Functionality

It defines the polarization direction ϵ. From it a vector of unit length is calculated, which enters the calculation. In 1D simulation it can be omitted and [1,0,0] is then assumed.

---

output_matrix_elements

Calling sequence

`quantum{ region{ momentum_matrix_elements{ output_matrix_elements = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If `output_matrix_elements = yes` then matrix elements are saved in output file.

---

output_transition_energies

Calling sequence

`quantum{ region{ momentum_matrix_elements{ output_transition_energies = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `output_transition_energies = yes` then transition energies are saved in output file.

---

output_oscillator_strengths

Calling sequence

`quantum{ region{ momentum_matrix_elements{ output_oscillator_strengths = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `output_oscillator_strengths = yes` then oscillator strengths are saved in output file. Currently, only a simple formula is used, i.e. the free electron mass is used and not the *real* effective mass one.

Gamma{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle \Gamma_i | \epsilon \cdot \hat{\mathbf{p}} | \Gamma_j \rangle$.

X{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle X_i | \epsilon \cdot \hat{\mathbf{p}} | X_j \rangle$.

Delta{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle \Delta_i | \epsilon \cdot \hat{\mathbf{p}} | \Delta_j \rangle$.

L{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle L_i | \epsilon \cdot \hat{\mathbf{p}} | L_j \rangle$.

HH{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ HH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle HH_i|\epsilon \cdot \hat{\mathbf{p}}|HH_j\rangle$.

LH{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ LH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle LH_i|\epsilon \cdot \hat{\mathbf{p}}|LH_j\rangle$.

SO{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ SO{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle SO_i|\epsilon \cdot \hat{\mathbf{p}}|SO_j\rangle$.

KP6{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ KP6{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\sum_k \langle kp6_{k,i}|\epsilon \cdot \hat{\mathbf{p}}|kp6_{k,j}\rangle$, k = 1,. . .,6.

KP8{ }

Calling sequence

`quantum{ region{ momentum_matrix_elements{ KP8{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\sum_k \langle kp8_{k,i}|\epsilon \cdot \hat{\mathbf{p}}|kp8_{k,j}\rangle$, k = 1,. . .,8.

Last update: 27/05/2025

dipole_moment_matrix_elements{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Triggers calculation of polarization-dependent dipole moment matrix elements $\epsilon \cdot \langle\psi_i|\hat{\mathbf{d}}|\psi_j\rangle = \epsilon \cdot \langle\psi_i|e\hat{\mathbf{r}}|\psi_j\rangle$ for wave functions within selected band models. The light polarization direction ϵ is automatically normalized in the program. For further reading: J. H. Davies, *The Physics of Low-Dimensional Semiconductors. An Introduction*, 2006, Chapters 10 and 8.

Nested keywords

- *name*
- *direction*
- *Gamma{ }*
- *X{ }*
- *Delta{ }*
- *L{ }*
- *HH{ }*
- *LH{ }*
- *SO{ }*
- *KP6{ }*
- *KP8{ }*
- *output_matrix_elements*
- *output_transition_energies*
- *output_oscillator_strengths*

---

name

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ name = ... } } }`

Properties

- **usage:** optional
- **type:** character string

Functionality

defines suffix for related output files

---

direction

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ direction = [ ..., ..., ... ] } } }`

Properties

- **usage:** optional
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **default:** $r_1 = 1.0$, $r_2 = 0.0$, $r_3 = 0.0$
- **unit:** $-$

Functionality

It defines the polarization direction ϵ. From it a vector of unit length is calculated, which enters the calculation. In 1D simulation it can be omitted and [1,0,0] is then assumed.

---

output_matrix_elements

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ output_matrix_elements = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

If `output_matrix_elements = yes` then matrix elements are saved in output file.

---

output_transition_energies

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ output_transition_energies = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `output_transition_energies = yes` then transition energies are saved in output file.

---

output_oscillator_strengths

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ output_oscillator_strengths = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If `output_oscillator_strengths = yes` then oscillator strengths are saved in output file.

Currently, only a simple formula is used, i.e. the free electron mass is used and not the *real* effective mass one.

Gamma{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle\Gamma_i|\epsilon\cdot\hat{\mathbf{d}}|\Gamma_j\rangle$.

X{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle X_i|\epsilon \cdot \hat{\mathbf{d}}|X_j\rangle$.

Delta{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle \Delta_i|\epsilon \cdot \hat{\mathbf{d}}|\Delta_j\rangle$.

L{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle L_i|\epsilon \cdot \hat{\mathbf{d}}|L_j\rangle$.

HH{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ HH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle HH_i|\epsilon \cdot \hat{\mathbf{d}}|HH_j\rangle$.

LH{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ LH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle LH_i|\epsilon \cdot \hat{\mathbf{d}}|LH_j\rangle$.

SO{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ SO{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\langle SO_i|\epsilon \cdot \hat{\mathbf{d}}|SO_j\rangle$.

KP6{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ KP6{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\sum_k \langle kp6_{k,i}|\epsilon \cdot \hat{\mathbf{d}}|kp6_{k,j}\rangle$, k = 1,…,6.

KP8{ }

Calling sequence

`quantum{ region{ dipole_moment_matrix_elements{ KP8{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculates the matrix element $\sum_k \langle kp8_{k,i}|\epsilon \cdot \hat{\mathbf{d}}|kp8_{k,j}\rangle$, k = 1,. . . ,8.

Last update: 27/05/2025

transition_energies{ }

Calling sequence

`quantum{ region{ transition_energies{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculate transition energies (energy difference) between two states in certain bands. Use this if you want to calculate transition energies but but do not want to calculate the matrix elements. Note that the matrix elements defined above also include specifiers for transition energies: `output_transition_energies = yes`.

Nested keywords

- *KP8{ }*
- *KP6{ }*
- *HH{ }*
- *LH{ }*
- *SO{ }*
- *Gamma{ }*
- *L{ }*
- *X{ }*
- *Delta{ }*
- *KP6_Gamma{ }*
- *HH_Gamma{ }*
- *HH_L{ }*
- *HH_X{ }*
- *HH_Delta{ }*
- *LH_Gamma{ }*
- *LH_L{ }*
- *LH_X{ }*
- *LH_Delta{ }*
- *SO_Gamma{ }*
- *SO_L{ }*
- *SO_X{ }*
- *SO_Delta{ }*

KP6_Gamma{ }

Calling sequence

`quantum{ region{ transition_energies{ KP6_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

HH_Gamma{ }

Calling sequence

`quantum{ region{ transition_energies{ HH_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

LH_Gamma{ }

Calling sequence

`quantum{ region{ transition_energies{ LH_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

SO_Gamma{ }

Calling sequence

`quantum{ region{ transition_energies{ SO_Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

Delta{ }

Calling sequence

`quantum{ region{ transition_energies{ Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

HH_Delta{ }

Calling sequence

`quantum{ region{ transition_energies{ HH_Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

LH_Delta{ }

Calling sequence

`quantum{ region{ transition_energies{ LH_Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

SO_Delta{ }

Calling sequence

`quantum{ region{ transition_energies{ SO_Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

Gamma{ }

Calling sequence

`quantum{ region{ transition_energies{ Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

X{ }

Calling sequence

`quantum{ region{ transition_energies{ X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

HH_X{ }

Calling sequence

`quantum{ region{ transition_energies{ HH_X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

LH_X{ }

Calling sequence

`quantum{ region{ transition_energies{ LH_X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

SO_X{ }

Calling sequence

`quantum{ region{ transition_energies{ SO_X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

L{ }

Calling sequence

`quantum{ region{ transition_energies{ L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

HH_L{ }

Calling sequence

`quantum{ region{ transition_energies{ HH_L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

LH_L{ }

Calling sequence

`quantum{ region{ transition_energies{ LH_L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

SO_L{ }

Calling sequence

`quantum{ region{ transition_energies{ SO_L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

HH{ }

Calling sequence

`quantum{ region{ transition_energies{ HH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

LH{ }

Calling sequence

`quantum{ region{ transition_energies{ LH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

SO{ }

Calling sequence

`quantum{ region{ transition_energies{ SO{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

KP6{ }

Calling sequence

`quantum{ region{ transition_energies{ KP6{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

KP8{ }

Calling sequence

`quantum{ region{ transition_energies{ KP8{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

Last update: 27/05/2025

quantum{ region{ quantize_x{ }, … } }

Calling sequence

```
quantum{ region{ quantize_x{ } }
quantum{ region{ quantize_y{ } }
quantum{ region{ quantize_z{ } }
```

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

In 2D or 3D simulation, the Schrödinger equation is solved within the 1D slices parallel to the x direction when `quantize_x{}` is called. This results in the reduction of the calculation time. For example, if a 2D simulating has 100 grid points in x-direction and 50 grid points in y-direction, the normal calculation solves the eigenvalue problem of a (100x50) x (100x50) matrix. When `quantize_x{}` is specified, on the other hand, *nextnano*++ solves the 1D Schrödinger equation along the x-direction at each grid point in y-direction. Therefore, 50 eigenvalue problems of 100x100 matrices are solved. Thus, the runtime of the eigenvalue solver can be roughly estimated (number of y-grids):math:^*{-1}*, but we should note that the runtime also depends on the number of eigenvalues to be calculated.

Only one quantization direction (x, y, z) can be specified at a time when quantum decomposition is used. Typically, the quantization direction is the growth direction.

Note that a similar number of states should be requested as for a corresponding 1D simulation (i.e. much less than normally needed in 2D or 3D), and that lateral (i.e. orthogonal to the quantization direction) grid spacing can be much larger than for "normal" quantum simulation, as the density from quantum decomposition is NOT affected by wide lateral grid spacing.

Currently, only one-band model (Gamma, X, Delta, LH, HH, etc.) without k-integration and without magnetic field is supported. Outputs based on wave functions (e.g., all outputs generated by *run{ quantum_optics{ } }*, any type of matrix elements, lifetimes, excitons) are not evaluated, since proper wave functions are not computed within this approximate method.

`quantize_y{ }` and `quantize_z{}` are analogous to `quantize_x{}`, triggering solving 1D Schrödinger equations along y- and z- directions, respectively.

Last update: 27/05/2025

lifetimes{ }

Calling sequence

`quantum{ region{ lifetimes{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Calculate the lifetimes of the state due to LO phonon scattering. For more information check R. Ferreira, G. Bastard, PRB 40, 1074 (1989) and Section 2.1.3 of the PhD thesis of G. Scarpa, Technische Universität München.

Nested keywords

- *phonon_energy*
- *Gamma{ }*
- *X{ }*
- *Delta{ }*
- *L{ }*
- *HH{ }*
- *LH{ }*
- *SO{ }*

---

phonon_energy

Calling sequence

`quantum{ region{ lifetimes{ phonon_energy = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[1e-9, ...)`
- **default:** $r = 1e - 2$
- **unit:** eV

Functionality

LO phonon energy

Gamma{ }

Calling sequence

`quantum{ region{ lifetimes{ Gamma{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

One-band model for computing the lifetimes.

X{ }

Calling sequence

`quantum{ region{ lifetimes{ X{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

One-band model for computing the lifetimes.

Delta{ }

Calling sequence

`quantum{ region{ lifetimes{ Delta{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

One-band model for computing the lifetimes.

L{ }

Calling sequence

`quantum{ region{ lifetimes{ L{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

One-band model for computing the lifetimes.

HH{ }

Calling sequence

`quantum{ region{ lifetimes{ HH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

One-band model for computing the lifetimes.

LH{ }

Calling sequence

`quantum{ region{ lifetimes{ LH{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

One-band model for computing the lifetimes.

SO{ }

Calling sequence

`quantum{ region{ lifetimes{ SO{ } } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

One-band model for computing the lifetimes.

Last update: 27/05/2025

excitons{ }

Calling sequence

`quantum{ region{ excitons{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

An **optional** group triggering computation of binding energies of excitons.

Note

This feature is under development.

Attention

This model can be used only for 1D simulations.

Nested keywords

- *electron_mass*
- *hole_mass*
- *density_averaged_masses*
- *dielectric_const*
- *energy_cutoff*
- *accuracy*

---

electron_mass

Calling sequence

`quantum{ region{ excitons{ electron_mass = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-3} \leq r \leq 10.0$
- **default:** volume average of values from the material database
- **unit:** –

Functionality

Effective mass of electron involved in the exciton.

---

hole_mass

Calling sequence

`quantum{ region{ excitons{ hole_mass = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-3} \leq r \leq 10.0$
- **default:** volume average of values from the material database
- **unit:** –

Functionality

Effective mass of hole involved in the exciton.

---

density_averaged_masses

Calling sequence

`quantum{ region{ excitons{ density_averaged_masses = ... } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Effective masses of hole and electron are averaged with weights taken from probability densities of related states

---

dielectric_const

Calling sequence

`quantum{ region{ excitons{ dielectric_const = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $1.0 \leq r \leq 10^3$
- **default:** volume average of values from the material database
- **unit:** $-$

Functionality

Effective dielectric constant assumed for electron-hole Coulomb interaction; If no explicit value of the dielectric constant is set, then the material values of the static dielectric constant (as given by the database and used in Poisson equation) are volume-averaged over the quantum region

---

energy_cutoff

Calling sequence

`quantum{ region{ excitons{ energy_cutoff = ... } } }`

Properties

- **usage:** required
- **type:** real number
- **values:** `[1e-3, ...)`
- **unit:** eV

Functionality

Maximum energy difference of electron and hole states involved in forming exciton

---

accuracy

Calling sequence

`quantum{ region{ excitons{ accuracy = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-10} \leq r \leq 0.1$
- **default:** $r = 1e-4$
- **unit:** $-$

Functionality

Accuracy used in minimisation procedure to compute the exciton binding energy

---

Last update: 27/05/2025

output_wavefunctions{ }

Calling sequence

`quantum{ region{ output_wavefunctions{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Provides options for output of wave function data

Nested keywords

- *max_num*
- *all_k_points*
- *amplitudes*
- *probabilities*
- *in_one_file*
- *scale*
- *structured*
- *energy_shift*
- *include_energies_in_shifted_files*

---

max_num

Calling sequence

`quantum{ region{ output_wavefunctions{ max_num = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 9999$
- **default:** not defined

Functionality

—

---

all_k_points

Calling sequence

`quantum{ region{ output_wavefunctions{ all_k_points = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Prints out the wave functions for all $k_{||}$ points (1D: $k_{||} = (k_y, k_z)$, 2D: $k_{||} = k_z$) that are used in the `k_integration{}` or `dispersion{}`. Enabling this option can produce a large number of output files.

---

amplitudes

Calling sequence

`quantum{ region{ output_wavefunctions{ amplitudes = "..." } } }`

Properties

- **usage:** optional
- **type:** enumerator
- **values:** `yes`; `no`; `S_X_Y_Z`; `CB_HH_LH_SO`
- **default:** `no`

Functionality

Prints out the wave functions ψ in units of 1D: $\mathrm{nm}^{-1/2}$, 2D: nm^{-1}, 3D: $\mathrm{nm}^{-3/2}$.

options

“ yes “ : for *k.p* it is equivalent to S_X_Y_Z

“ no “ : no output is done for amplitudes.

“ S_X_Y_Z “ : prints out the wave functions (psi) with respect to the basis (*k.p* only) $|S+\rangle|S-\rangle|X+\rangle|Y+\rangle|Z+\rangle|X-\rangle|Y-\rangle|Z-\rangle$. $|X+\rangle|Y+\rangle|Z+\rangle$ correspond to the x, y, z of the simulation coordinate system (and not crystal coordinate system) and + and - correspond to the spin projection along the z axis of the crystal system.

“ CB_HH_LH_SO “ : prints out the wave functions (psi) with respect to the basis (*k.p* only) $|cb+\rangle|cb-\rangle|hh+\rangle|lh+\rangle|lh-\rangle|hh-\rangle|so+\rangle|so-\rangle$. This basis is the same as used in L. C. Lew Yan Voon, M. Willatzen, *The k.p method`(2009) (Table 3.4); G. Bastard, `Wave Mechanics Applied to Semiconductor Heterostructures* (1988) and B. A. Foreman, PRB 48, 4964 (1993).

If multiple choices are required type them together inside a string like

```
amplitudes = "S_X_Y_Z CB_HH_LH_SO"
```

---

probabilities

Calling sequence

`quantum{ region{ output_wavefunctions{ probabilities = "..." } } }`

Properties

- **usage:** optional
- **type:** enumerator
- **values:** `yes`; `no`; `S_X_Y_Z`; `CB_HH_LH_SO`
- **default:** `yes`

Functionality

Prints out the wave functions $|\psi|^2$ in units of 1D: nm^{-1}, 2D: nm^{-2}, 3D: nm^{-3}.

yes : for *k.p* it is the sum of the squares of all components of a spinor
no : no output
S_X_Y_Z : same as for the amplitudes (*k.p* only)
CB_HH_LH_SO : same as for the amplitudes (*k.p* only)

Multiple choices are possible.

```
probabilities = "yes CB_HH_LH_SO"
```

---

in_one_file

Calling sequence

`quantum{ region{ output_wavefunctions{ in_one_file = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Prints out the amplitudes into one file and the probabilities into one file. If no is chosen, for each eigenvalue a separate file is written out.

---

scale

Calling sequence

`quantum{ region{ output_wavefunctions{ scale = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 1.0$
- **unit:** $-$

Functionality

Scale factor for output of amplitudes and probabilities

---

structured

Calling sequence

`quantum{ region{ output_wavefunctions{ structured = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

The whole output for `quantum{ }` is written in subdirectory *Quantum/*. If enabled, additional subdirectories are created in subdirectory *Quantum/* to organize the structure of the output files in a meaningful way. It is recommended to set this parameter to yes if a lot of output files are created, e.g. in case `all_k_points = yes`, and both `amplitudes` and `probabilities` are printed out.

---

energy_shift

Calling sequence

`quantum{ region{ output_wavefunctions{ energy_shift = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `shifted` or `not_shifted` or `both`
- **default:** `both`

Functionality

shifted : prints out the amplitudes and the probabilities shifted by the energy.
not_shifted : prints out the amplitudes and the probabilities as they are (an integral over volume is equal to 1).
both : prints out the amplitudes and the probabilities with and without energy shift.

---

include_energies_in_shifted_files

Calling sequence

`quantum{ region{ output_wavefunctions{ include_energies_in_shifted_files = "..." } } }`

Properties

—

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Selects if the energy levels are added in output of shifted amplitudes and probabilities or not. If no is selected a separate file with energy levels is written out.

> **Note**
>
> The energy spectrum (i.e. the eigenvalues) are always written into the files *energy_spectrum_*.dat*. The projections of the eigenfunctions on the basis states of the bulk Hamiltonian are written into the files *spinor_composition_*.dat*.

Last update: 27/05/2025

output_subband_densities{ }

Calling sequence

`quantum{ region{ output_subband_densities{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Provides options for output of subband densities.

Nested keywords

- *max_num*
- *in_one_file*

---

max_num

Calling sequence

`quantum{ region{ output_subband_densities{ max_num = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 9999$
- **default:** not defined

Functionality

number of subband densities to be printed out. If `max_num` is not present, the subband density is written out for each eigenvalue.

---

in_one_file

Calling sequence

`quantum{ region{ output_subband_densities{ in_one_file = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Prints out the subband densities into one file. If `no` is chosen, for each subband density a separate file is written out. This feature only makes sense for 1D simulations.

Last update: 27/05/2025

quantum{ region{ output_quantum_densities{ } } }

Calling sequence

`quantum{ region{ output_quantum_densities{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

If set, the quantum density belonging to each quantum solver (i.e. for each valley) will be output. Essentially, it contained the values of `output_subband_densities{}` summed over the subbands.

Last update: 27/05/2025

quantum{ region{ output_occupations_on_grid{ } } }

Calling sequence

`quantum{ region{ output_occupations_on_grid{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

—

Nested keywords

- *max_num*
- *in_one_file*

---

max_num

Calling sequence

`quantum{ region{ output_occupations_on_grid{ max_num = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 9999$
- **default:** not defined

Functionality

—

---

in_one_file

Calling sequence

`quantum{ region{ output_occupations_on_grid{ in_one_file = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

—

Last update: 27/05/2025

quantum{ region{ output_energies_on_grid{ } } }

Calling sequence

`quantum{ region{ output_energies_on_grid{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Generates and outputs subband occupations extended over the grid.

Nested keywords

- *max_num*
- *all_k_points*
- *structured*
- *in_one_file*

---

max_num

Calling sequence

`quantum{ region{ output_energies_on_grid{ max_num = ... } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 9999$
- **default:** not defined

Functionality

—

---

all_k_points

Calling sequence

`quantum{ region{ output_energies_on_grid{ all_k_points = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

Prints out the wave functions for all $k_{||}$ points (1D: $k_{||} = (k_y, k_z)$, 2D: $k_{||} = k_z$) that are used in the `k_integration{}` or `dispersion{}`. Enabling this option can produce a large number of output files.

---

structured

Calling sequence

`quantum{ region{ output_energies_on_grid{ structured = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

The whole output for `quantum{ }` is written in subdirectory *Quantum/*. If enabled, additional subdirectories are created in subdirectory *Quantum/* to organize the structure of the output files in a meaningful way.

---

in_one_file

Calling sequence

`quantum{ region{ output_energies_on_grid{ in_one_file = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

—

Last update: 27/05/2025

output_rotated_inverse_mass_tensor{ }

Calling sequence

`quantum{ region{ output_rotated_inverse_mass_tensor{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs components of tensor of the inverse mass in simulation coordinate system

Nested keywords

- *boxes*
- *structured*

---

boxes

Calling sequence

`quantum{ region{ output_rotated_inverse_mass_tensor{ boxes = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at material interfaces (in 2D four points, in 3D eight points)

---

structured

Calling sequence

`quantum{ region{ output_rotated_inverse_mass_tensor{ structured = "..." } } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

By default, whole output is written in subdirectory *Quantum/*. If `yes` is chosen then additional subdirectories are created in subdirectory *Quantum/* to organize the structure of the output files in a meaningful way.

Last update: 27/05/2025

Last update: 27/05/2025

7.14.4 exchange_correlation{ }

Calling sequence

`quantum{ exchange_correlation{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Exchange-correlation potential is added to the Hamiltonian within selected approximation.

Note

It is advised to use this keyword together with any of self-consistent run modes *quantum_density{ }*, *quantum_poisson{ }*, or *quantum_current_poisson{ }*. Using it with *quantum{ }* only will result in lack of self-consistency between the exchange-correlation potential and the final carrier densities.

Attention

The feature is currently available only for 1-band models. It is ignored for multi-band $\mathbf{k}\cdot\mathbf{p}$ models.

Nested keywords

- *type*
- *initial_spin_pol*
- *output_spin_polarization{ }*
- *output_exchange_correlation{ }*

---

type

Calling sequence

`quantum{ exchange_correlation{ type = ... } }`

Properties

- **usage:** required
- **type:** choice
- **values:** `lda` or `lsda`

Functionality

`lda` → Includes exchange-correlation effects in the LDA approximation (Local Density Approximation)
`lsda` → Includes exchange-correlation effects in the LSDA approximation (Local Spin Density Approximation)

---

initial_spin_pol

Calling sequence

`quantum{ exchange_correlation{ initial_spin_pol = ... } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $-1.0 \leq r \leq 1.0$
- **default:** $r = 0.0$
- **unit:** $-$

Functionality

Breaks spin symmetry if magnetic field is not present.

---

output_spin_polarization{ }

Calling sequence

`quantum{ exchange_correlation{ output_spin_polarization{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs spin polarization (dimensionless).

---

output_exchange_correlation{ }

Calling sequence

`quantum{ exchange_correlation{ output_exchange_correlation{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Outputs exchange correlation potentials expressed in `(eV)`.

Last update: 27/05/2025

7.14.5 cbr{ }

Calling sequence

`quantum{ cbr{ } }`

Properties

- **usage:** optional
- **items:** maximum 1

Dependencies

- if `global{ simulate1D{} }` is called then `quantum{ cbr{ lead } }` cannot be used
- `quantum{ cbr{ min_energy } }` and `quantum{ cbr{ rel_min_energy} }` cannot be used simultaneously
- `quantum{ cbr{ max_energy } }` and `quantum{ cbr{ rel_max_energy } }` cannot be used simultaneously

Functionality

Specifications that define CBR (**C**ontact **B**lock **R**eduction method) calculation, i.e. ballistic current calculations. This method is based on the following publications: *[BirnerCBR2009]*, *[MamaluyCBR2003]*

CBR current calculation at a glance:

- full 1D, 2D and 3D calculation of quantum mechanical ballistic transmission probabilities for open systems with scattering boundary conditions
- **Contact Block Reduction method:**
    - only incomplete set of quantum states needed (~ 100)
    - reduction of matrix sizes from $O(N^3)$ to $O(N^2)$
- ballistic current according to Landauer–Büttiker formalism

The CBR method is an efficient method that uses a limited set of eigenstates of the decoupled device and a few propagating lead modes to calculate the retarded Green's function of the device coupled to external contacts. From this Green's function, the density and the current is obtained in the ballistic limit using Landauer's formula with fixed Fermi levels for the leads. It is important to note that the efficiency of the calculation and also the convergence of the results are strongly dependent on the cutoff energies for the eigenstates and modes. Thus it is important to check during the calculation if the specified number of states and modes is sufficient for the applied voltages. To summarize, the code may do its job very efficiently but is far away from being a black box tool.

```
cbr{
    name = "qr" # quantum region to which cbr method will be
    lead{
        name = "lead_1"             # name of the lead
        x = 12.0                    # position of the lead in  1D simulation
        kinetic_coupling = 1.5
        rel_kinetic_coupling = 0.2
    }

    min_energy =  2.5        # lower boundary (absolute)
    max_energy =  2.6        # upper boundary (absolute)

    rel_min_energy = -0.01       # lower boundary (relative)
    rel_max_energy =  0.3        # upper boundary (relative)

    energy_resolution   =  1e-6       # energy grid resolution
    transmission_threshold  = 0.01

    ildos          =  yes        # outputs integrated LDOS
    ldos           =  yes        # outputs LDOS

    output_ldos_single_file = yes
}
```

Figure 7.14.5.1 shows the calculated transmission from lead 1 to lead 3 as a function of energy $T_{13}(E)$. Full line: All eigenfunctions of the decoupled device are taken into account. Dashed line: Only the lowest 7% of the eigenfunctions are included. Here, Neumann boundary conditions are used for the propagation direction. The vertical line indicates the cutoff energy, i.e. the highest eigenvalue that is taken into account.

Special boundary conditions are applied for the Schrödinger equation while using the CBR method:

- *Neumann* boundary conditions along the propagation direction.
- *Dirichlet* boundary conditions perpendicular to the propagation direction.

Note

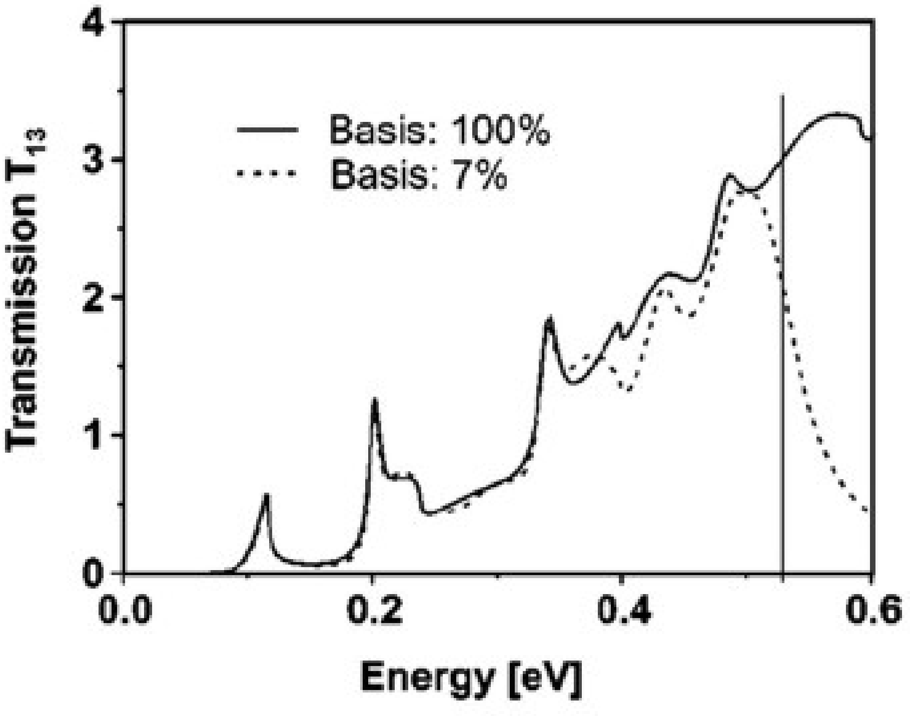


Figure 7.14.5.1: The transmission calculated with the CBR method using all eigenstates and only 7% of the eigenstates. In the latter case, the transmission is still very accurate for the lower energies.

The quantum region must be a surface in a 3D simulation, a line in a 2D simulation, and a point in a 1D simulation.

Nested keywords

- *name*
- *lead{ }*
- *lead{ name }*
- *lead{ x }*
- *lead{ kinetic_coupling }*
- *lead{ rel_kinetic_coupling }*
- *min_energy*
- *max_energy*
- *rel_min_energy*
- *rel_max_energy*
- *energy_resolution*
- *transmission_threshold*
- *ildos*
- *ldos*
- *output_ldos_single_file*
- *two_particle_options*

---

name

Calling sequence

`quantum{ cbr{ name = ... } }`

Properties

- **usage:** required
- **type:** character string

Functionality

refers to quantum region to which CBR method will be applied (d-dimensional)

lead{ }

Calling sequence

`quantum{ cbr{ lead{ } } }`

Properties

- **usage:** required
- **items:** minimum 2

Functionality

Defining a lead. The lead region has dimension $d - 1$.

lead{ name }

Calling sequence

`quantum{ cbr{ lead{ name = ... } } }`

Properties

- **usage:** required
- **type:** character string

Functionality

Provides the name of the quantum region of the lead. It must be corresponding to a defined `quantum{ region{} }` unless the global simulation is held in 1D.

lead{ x }

Calling sequence

`quantum{ cbr{ lead{ x = ... } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 0.0$
- **unit:** nm

Functionality

—

> **Note**
>
> Only needed for 1D.

lead{ kinetic_coupling }

Calling sequence

`quantum{ cbr{ lead{ kinetic_coupling = ... } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** eV

Dependencies

`rel_kinetic_coupling` is not defined

Functionality

—

---

lead{ rel_kinetic_coupling }

Calling sequence

`quantum{ cbr{ lead{ rel_kinetic_coupling = ... } } }`

Properties

- **usage:** conditional
- **type:** real number
- **values:** `(0.0, ...)`
- **default:** $r = 1.0$
- **unit:** $-$

Dependencies

`kinetic_coupling` is not defined

Functionality

—

---

min_energy

Calling sequence

`quantum{ cbr{ min_energy = ... } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = -1e100$
- **unit:** eV

Dependencies

`rel_min_energy` is not defined

Functionality

Lower boundary for transmission energy interval on an absolute energy scale

---

max_energy

Calling sequence

`quantum{ cbr{ max_energy = ... } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 1e100$
- **unit:** eV

Dependencies

`rel_max_energy` is not defined

Functionality

Upper boundary for transmission energy interval on an absolute energy scale

---

rel_min_energy

Calling sequence

`quantum{ cbr{ rel_min_energy = ... } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = -1e100$
- **unit:** $-$

Dependencies

`min_energy` is not defined

Functionality

Lower boundary for transmission energy interval relative to the lowest eigenvalue

---

rel_max_energy

Calling sequence

`quantum{ cbr{ rel_max_energy = ... } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **default:** $r = 1e100$
- **unit:** $-$

Dependencies

`max_energy` is not defined

Functionality

Upper boundary for transmission energy interval relative to the highest eigenvalue

---

energy_resolution

Calling sequence

`quantum{ cbr{ energy_resolution = ... } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `(0.0, ...)`
- **default:** $r = 1e-4$
- **unit:** eV

Functionality

This value determines the resolution of the transmission curve $T(E)$.

---

transmission_threshold

Calling sequence

`quantum{ cbr{ transmission_threshold = ... } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 0.0$
- **unit:** $-$

Functionality

This value determines the resolution of the transmission curve $T(E)$.

---

ildos

Calling sequence

`quantum{ cbr{ ildos = ... } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then outputs integrated local density of states.

---

ldos

Calling sequence

`quantum{ cbr{ ldos = ... } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Functionality

If set to `yes` then outputs local density of states.

---

output_ldos_single_file

Calling sequence

`quantum{ cbr{ output_ldos_single_file = ... } }`

Properties

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Functionality

Outputs all LDOS data into a single large file.

> ⚠ **Warning**
>
> Enabling ILDOS or LDOS can massively increase runtime and RAM usage in 2D and 3D simulations. Moreover, enabling LDOS also will rewrite huge amounts of data to disk in 2D and 3D simulations.
>
> If your system environment cannot handle a huge number of files (e.g. you are using a slow hard disk instead of a SSD), outputting all LDOS data into a single large file (as set per default) is strongly recommended.
>
> Please note that writing all LDOS data in one file is not possible in 3D simulations or when `output{ only_sections = yes }` is set (the respective flag is ignored then). See *output{ }* for reference.

---

two_particle_options

Calling sequence

`quantum{ cbr{ two_particle_options = [ ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ... ] } }`

Properties

- **usage:** optional
- **type:** vector of 11 real numbers: $(r_1, r_2, \ldots, r_{11})$
- —

Functionality

Contains 11 values for two-particle model `[number of states, relative permittivity, x1, y1, z1, x2, y2, z2, splitting, tunneling]` with units [–, –, nm, nm, nm, nm, nm, nm, eV, eV]. Constraint: number of states = 2

Last update: 27/05/2025

Example

```
quantum{
    debuglevel      = 1
    allow_overlapping_regions = no

    #----------------
    # Quantum regions
    #----------------
    region{
        name = "qr1"

        quantize_x{}
        quantize_y{}
        quantize_z{}

        no_density = yes
```

(continues on next page)

(continued from previous page)

```
        x = [10.0, 20.0]
        y = [10.0, 20.0]
        z = [10.0, 20.0]

        # Boundary conditions
        #---------------------
        boundary{
            x = dirichlet
            y = dirichlet
            z = neumann
            classical_boundary_x = no
            classical_boundary_y = no
            classical_boundary_z = no
            num_classical_x  = [1,1]
            num_classical_y  = [1,1]
            num_classical_z  = [1,1]
        }

        # Output definitions
        #--------------------
        output_wavefunctions{
            max_num        = 10
            all_k_points   = yes/no
            structured     = no
            amplitudes = "S_X_Y_Z CB_HH_LH_SO"
            probabilities = "yes CB_HH_LH_SO"
            scale          = 0.7
            in_one_file    = yes
            energy_shift   = both
            include_energies_in_shifted_files = yes
        }
        output_subband_densities{
            max_num     = 10
            in_one_file = yes
        }
        output_sparse_matrix{
            type = all
            structured = no
        }
        output_rotated_inverse_mass_tensor{
            boxes      = yes
            structured = no
        }

        # Quantum models and solver definitions
        #--------------------------------------
        Gamma{
            num_ev            = 10
            # Eigensolvers (choose one)
            lapack{}
            arpack{}
            accuracy          = 1e-6
            iterations        = 200
            preconditioner    = chebyshev
            cutoff            = 0.3
            abs_cutoff        = 2.5
```

(continues on next page)

(continued from previous page)

```
            order_chebyshev  = 20

            # Dispersion
            #-----------
            dispersion{
                path{
                    name = "100"
                    point{
                        k = [1.0, 0.0, 0.0]
                        k = [1.0, 1.0, 0.0]
                    }
                    spacing    = 0.5
                    num_points = 10
                }
                lines{
                    name = "lines"
                    spacing  = 0.5
                    k_max    = 1.0
                }
                full{
                    name = "3D"
                    kxgrid{
                        line{
                            pos = -1
                            spacing = 0.02
                        }
                    }
                    kygrid{
                        line{
                            pos = -1
                            spacing = 0.02
                        }
                    }
                    kzgrid{
                        line{
                            pos = -1
                            spacing = 0.02
                        }
                    }
                }
                superlattice{
                    name = "superlattice"
                    num_points_x    = 10
                    num_points_y    = 15
                    num_points_z    = 20
                    num_points      = 20
                }
            }
        }

        L{
            ... (same as Gamma)
        }

        X{
            ... (same as Gamma)
```

(continues on next page)

(continued from previous page)

```
        }

        Delta{
            ... (same as Gamma)
        }

        HH{
            ... (same as Gamma)
        }

        LH{
            ... (same as Gamma)
        }

        SO{
            ... (same as Gamma)
        }

        kp_6band{
            ... (same as Gamma)

            kp_parameters{
                use_Luttinger_parameters = no
                approximate_kappa        = no
            }

            lapack{}
            #arpack{}

            k_integration{
                relative_size = 0.2
                num_points    = 5
                num_subpoints = 2
                max_symmetry  = no
                force_k0_subspace = yes
            }
        }

        kp_8band{
            num_electrons     = 6
            num_holes         = 12
            accuracy          = 1e-8
            iterations        = 200

            kp_parameters{
                use_Luttinger_parameters = no
                from_6band_parameters    = no
                approximate_kappa        = no
                evaluate_S               = no
                rescale_S_to             = 1.0
            }

            k_integration{
                ... (same as kp_6band)
            }
```

(continues on next page)

(continued from previous page)

```
            lapack{}
            #arpack_inv{}
            shift_window      = 0
            shift             = 0.2
            abs_shift         = 2.5

            linear_solver{
                iterations      = 500
                abs_accuracy    = 1e-9
                rel_accuracy    = 1e-9
                use_cscg        = no
                force_diagonal_preconditioner = no
            }

            #advanced settings for 8-band k.p quantum density
            shift_min_CB      = 0.0
            shift_max_VB      = 0.0
            tunneling         = yes

            classify_kspace   = 0
            threshold_classification = 0.5
        }

        #Matrix elements definitions
        #---------------------------
        overlap_integrals{
            KP6_Gamma{}
            HH_Gamma{}                 # < HH_i | Gamma_j >
            LH_Gamma{}                 # < LH_i | Gamma_j >
            SO_Gamma{}                 # < SO_i | Gamma_j >
            HH_Delta{}                 # < HH_i | Delta_j >
            LH_Delta{}                 # < LH_i | Delta_j >
            SO_Delta{}                 # < SO_i | Delta_j >
            HH_X{}                     # < HH_i | X_j >
            LH_X{}                     # < LH_i | X_j >
            SO_X{}                     # < SO_i | X_j >
            HH_L{}                     # < HH_i | L_j >
            LH_L{}                     # < LH_i | L_j >
            SO_L{}                     # < SO_i | L_j >

            output_matrix_elements      = yes
            output_transition_energies  = yes/no #
        }

        momentum_matrix_elements{
            direction = [1,1,0]
            Gamma{}
            Delta{}
            X{}
            L{}
            HH{}
            LH{}
            SO{}
            KP6{}
            KP8{}
```

(continues on next page)

(continued from previous page)

```
            output_matrix_elements      = yes/no                output_transition_
↪energies  = yes/no
            output_oscillator_strengths = yes/no
        }

        dipole_moment_matrix_elements{
            direction = [1,1,0]
            Gamma{}
            Delta{}
            X{}
            L{}
            HH{}
            LH{}
            SO{}
            KP6{}
            KP8{}

            output_matrix_elements      = yes
            output_transition_energies  = yes
            output_oscillator_strengths = yes
        }

        transition_energies{
            Gamma{}
            KP6_Gamma{}
            HH_Gamma{}
            LH_Gamma{}
            SO_Gamma{}
            Delta{}
            HH_Delta{}
            LH_Delta{}
            SO_Delta{}
            X{}
            HH_X{}
            LH_X{}
            SO_X{}
            L{}
            HH_L{}
            LH_L{}
            SO_L{}
            HH{}
            LH{}
            SO{}
            KP6{}
            KP8{}
        }


        lifetimes{
            phonon_energy = 0.036
            Gamma{}
            HH{}
            LH{}
        }

    } # end: region{}
```

(continues on next page)

(continued from previous page)

```
    #Many body effects
    #------------------
    exchange_correlation{
        type               = lda
        initial_spin_pol = 1.0
        output_spin_polarization{}
        output_exchange_correlation{}
    }

}
```

Last update: 2025/05/27

7.15 optics{ }

- **usage:** optional
- **items:** maximum 1

This group defines models to calculate optical spectra.

The following keywords are available within this group.

7.15.1 optics{ debuglevel }

- **usage:** optional
- **type:** integer
- **values:** $-1 \leq z \leq 4$
- **unit:** $-$
- **default:** $z = 0$

Parameter controlling diagnostic output in the *.log file. The larger the value is, the more details are included.

7.15.2 optics{ global_illumination{ } }

- **usage:** optional
- **items:** maximum 1

This group is defining a spectrum of radiation illuminating modelled device.

> **Note**
>
> Lorentzian, Gaussian and Planck illumination spectra are fully additive, i.e. several of each can be added as needed in order to synthesize more complex illumination spectra.

> **Hint**
>
> Spectral data can be defined in the database (see also *Optical groups in database{ }* for list of predefined illumination spectra), in the database section of the input file, or imported from external files.

Important

The following general conditions must be satisfied when defining *optics{ global_illumination{ } }*

- Maximum one of the following can be defined: *database_spectrum{ }*, *import_spectrum{ }*, *constant_spectrum{ }* within this group.
- Exactly one of the following must be defined: *direction_x*, *direction_y*, *direction_z* within this group.
- If *global{ simulate1D{ } }* is specified in the input file, then *direction_y* and *direction_z* are not allowed.
- If *global{ simulate2D{ } }* is specified in the input file, then *direction_z* is not allowed.

- *Maintained Keywords*
  - *direction_x*
  - *direction_y*
  - *direction_z*
  - *database_spectrum{ }*
  - *database_spectrum{ name }*
  - *database_spectrum{ concentration }*
  - *import_spectrum{ }*
  - *import_spectrum{ import_from }*
  - *import_spectrum{ cutoff }*
  - *import_spectrum{ energy_spectrum }*
  - *import_spectrum{ absolute_intensities }*
  - *import_spectrum{ concentration }*
  - *constant_spectrum{ }*
  - *constant_spectrum{ irradiance }*
  - *planck_spectrum{ }*
  - *planck_spectrum{ irradiance }*
  - *planck_spectrum{ temperature }*
  - *lorentzian_spectrum{ }*
  - *lorentzian_spectrum{ irradiance }*
  - *lorentzian_spectrum{ wavelength }*
  - *lorentzian_spectrum{ energy }*
  - *lorentzian_spectrum{ width }*
  - *lorentzian_spectrum{ gamma }*
  - *gaussian_spectrum{ }*
  - *gaussian_spectrum{ irradiance }*
  - *gaussian_spectrum{ wavelength }*
  - *gaussian_spectrum{ energy }*

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

---

direction_x

- **usage:** optional
- **type:** integer
- **values:** $z = -1$ or $z = +1$
- **unit:** $-$

Sets ascending $+1$ or descending -1 direction of illuminating radiation along the x-axis of simulation.

---

direction_y

- **usage:** optional
- **type:** integer
- **values:** $z = -1$ or $z = +1$
- **unit:** $-$

Sets ascending $+1$ or descending -1 direction of illuminating radiation along the y-axis of simulation.

---

direction_z

- **usage:** optional
- **type:** integer
- **values:** $z = -1$ or $z = +1$
- **unit:** $-$

Sets ascending $+1$ or descending -1 direction of illuminating radiation along the z-axis of simulation.

---

database_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing one of several spectra (solar spectra, CIE illuminants, coefficient, reflectivity, . . .), which can be found in the database file *Optical groups in database{ }*. Relative intensities (e.g. CIE illuminants) are normalized to 1.0 $\mathrm{W/m^2}$

---

database_spectrum{ name }

- **usage:** required
- **type:** character string

Name of the illumination spectrum contained in the database to be used.

database_spectrum{ concentration }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **default:** $r = 1.0$
- **unit:** $-$

Scaling factor multiplying the values of the spectrum.

import_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing spectrum from a file

Important

The following general conditions must be satisfied when defining *import_spectrum{ }*

- The *import{ }* must be specified in the input file.

import_spectrum{ import_from }

- **usage:** required
- **type:** character string

Reference name used in the `import{ }` group to label the imported spectrum.

import_spectrum{ cutoff }

- **usage:** required
- **type:** choice
- **values:** `yes` or `no`

If set to `yes`, then the values of the spectrum which are outside the definition interval are set to zero. Otherwise, the spectrum is extrapolated as a constant with the value on the boundary of the imported data.

import_spectrum{ energy_spectrum }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the imported spectrum is assumed to be given as a function of energy. Otherwise, the spectrum is assumed to be given as a function of wavelength.

---

import_spectrum{ absolute_intensities }

- **usage:** required
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the values are directly imported without normalization. Otherwise, the values of the imported spectrum are normalized to the total intensity of the spectrum.

---

import_spectrum{ concentration }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** –
- **default:** $r = 1.0$

Scaling factor multiplying the values of the spectrum.

---

constant_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Define illumination source with a constant radiation spectrum of the form

$$I(E) = \frac{I_0}{E_{\text{max}} - E_{\text{min}}}$$

---

constant_spectrum{ irradiance }

- **usage:** required
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $\mathrm{W/m^2}$

Total intensity :math:` I_0 = int I(E)dE` of the spectrum, integrated from E_{min} to E_{max}.

planck_spectrum{ }

- **usage:** optional
- **items:** no constraints

Define illumination source with a black-body radiation spectrum

$$I(E,T) = \frac{I_0}{\sigma T^4}\frac{2\pi E^3}{c^2h^3}\frac{1}{\exp\left\{\left(\frac{E}{k_B T}\right)\right\} - 1},$$

where σ is the Stefan–Boltzmann constant.

planck_spectrum{ irradiance }

- **usage:** required
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $\mathrm{W/m^2}$

Total intensity :math:` I_0 = int I(E)dE` of the spectrum

planck_spectrum{ temperature }

- **usage:** required
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** K

Temperature T entering the spectrum model

lorentzian_spectrum{ }

- **usage:** optional
- **items:** no constraints

Define illumination source with a Lorentzian radiation spectrum

$$I(E) = \frac{I_0}{\pi}\frac{\Gamma/2}{(E-E_0) + (\Gamma/2)^2}$$

Important

The following general conditions must be satisfied when defining *lorentzian_spectrum{ }*

- Exactly one, *lorentzian_spectrum{ wavelength }* or *lorentzian_spectrum{ energy }* is specified within this group.
- Exactly one, *lorentzian_spectrum{ width }* or *lorentzian_spectrum{ gamma }* is specified within this group.

lorentzian_spectrum{ irradiance }

- **usage:** required
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $\mathrm{W/m^2}$

Total intensity :math:` I_0 = int I(E)dE` of the spectrum

lorentzian_spectrum{ wavelength }

- **usage:** optional
- **type:** real number
- **values:** `[10.0, ...)`
- **unit:** nm

Central wavelength λ_0 of the spectrum

lorentzian_spectrum{ energy }

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

Central energy E_0 of the spectrum

lorentzian_spectrum{ width }

- **usage:** optional
- **type:** real number
- **values:** `[1e-3, ...)`
- **unit:** nm

Define the width of the spectrum in nm

lorentzian_spectrum{ gamma }

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

Define the width of the spectrum in eV

---

gaussian_spectrum{ }

Define illumination source with a Gaussian spectrum

- **usage:** optional
- **items:** no constraints

$$I(E) = \frac{I_0}{\sqrt{2\pi}\sigma}\exp\left\{\left[-\frac{(E-E_0)^2}{2\sigma^2}\right]\right\}$$

Important

The following general conditions must be satisfied when defining *gaussian_spectrum{ }*

- Exactly one, *gaussian_spectrum{ wavelength }* or *gaussian_spectrum{ energy }* is specified within this group.
- Exactly one, *gaussian_spectrum{ width }* or *gaussian_spectrum{ gamma }* is specified within this group.

---

gaussian_spectrum{ irradiance }

- **usage:** required
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $\mathrm{W/m^2}$

Total intensity :math:` I_0 = int I(E)dE` of the spectrum

---

gaussian_spectrum{ wavelength }

- **usage:** optional
- **type:** real number
- **values:** `[10.0, ...)`
- **unit:** nm

Central wavelength λ_0 of the spectrum

gaussian_spectrum{ energy }

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

Central energy E_0 of the spectrum

---

gaussian_spectrum{ width }

- **usage:** optional
- **type:** real number
- **values:** `[1e-3, ...)`
- **unit:** nm

Define the width of the spectrum in nm

---

gaussian_spectrum{ gamma }

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

Define the width of the spectrum in eV

Examples

```
constant_spectrum{
        irradiance = 10000.0  # in [W/m^2], integrated as min_energy...max_energy
}
```

```
planck_spectrum{
    irradiance = 10000.0  # in [W/m^2], for complete(!) Planck spectrum; real value >
↪= 0.0
    temperature = 5000.0  # real value >= 1e-6
}
```

```
global_illumination{
    direction_x = 1

    database_spectrum{
        name = "Solar-ASTM-G173-global"
    # name = "CIE-D75"
        concentration = 300 # e.g. 300 suns
    }
}
```

```
global_illumination{
    direction_x = 1

    import_spectrum{
        import_from = "filename"
        cutoff = yes  # yes/no: If yes, set values outside definition interval to␣
↪zero.
                    # (default=?)
        absolute_intensities = yes  # yes/no (default: yes)
                                    # If no, spectrum does not contain absolute␣
↪values,
                                    # normalize intensity to 1 [W/cm^2 nm^-1] before␣
↪concentration
        concentration = 300 # e.g. 300 suns
    }
}
```

```
lorentzian_spectrum{
        irradiance = 10000.0  # in [W/m^2], for complete(!) Lorentzian spectrum; real␣
↪value >= 0.0

        # Specify either wavelength and width, or ...
        wavelength = 500.0   # real value >= 10.0 in |unit:nm|
        width = 100.0        # real value >= 1e-3 in |unit:nm|

        # ... specify energy and gamma.
        energy = 2.5         # real value >= 1e-6 in |unit:eV|
        gamma = 1.0          # real value >= 1e-6 in |unit:eV|
}
```

```
gaussian_spectrum{
    irradiance = 1000.0  # in [W/m^2], for complete(!) Gaussian spectrum; real value >
↪= 0.0

    # Specify either wavelength and width, or ...
    wavelength = 500.0   # real value >= 10.0 in |unit:nm|
    width = 100.0        # real value >= 1e-3 in |unit:nm|

    # ... specify energy and gamma.
    energy = 2.5         # real value >= 1e-6 in |unit:eV|
    gamma = 1.0          # real value >= 1e-6 in |unit:eV|
}
```

7.15.3 optics{ global_reflectivity{ } }

- **usage:** optional
- **items:** maximum 1

This group defines the reflectance spectrum $R(\lambda)$ of the modelled device for the light entering the system.

Important

The following general conditions must be satisfied when defining *optics{ global_reflectivity{ } }*

- Exactly one of the following must be defined: *database_spectrum{ }*, *import_spectrum{ }*, *constant_spectrum{ }* within this group.

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

---

database_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing the spectrum from the database or external files.

---

database_spectrum{ name }

- **usage:** required
- **type:** character string

Name of the spectrum contained in the database.

---

import_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing spectrum from a file

Important

The following general conditions must be satisfied when defining *import_spectrum{ }*

- The global group *import{ }* is specified in the input file.

---

import_spectrum{ import_from }

- **usage:** required
- **type:** character string

Path to a spectrum for importing

---

import_spectrum{ cutoff }

- **usage:** required
- **type:** choice
- **values:** `yes` or `no`

If set to `yes`, then the values of the spectrum which are outside the definition interval are set to zero. Otherwise, the spectrum is extrapolated as a constant with the value on the boundary of the imported data.

---

import_spectrum{ energy_spectrum }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the imported spectrum is assumed to be given as a function of energy. Otherwise, the spectrum is assumed to be given as a function of wavelength.

---

constant_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Specify a constant reflectance spectrum $R(\lambda) = \text{const}$

---

constant_spectrum{ reflectivity }

- **usage:** required
- **type:** real number
- **values:** `(0.0, 1.0]`
- **unit:** $-$
- **default:** $r = 1.0$

The constant value of the reflectivity

Examples

```
global_reflectivity{
    database_spectrum{
        name = "Al0.80Ga0.20As"
    }
```

```
import_spectrum{
    import_from = "filename"
    cutoff = yes  # yes/no: If yes, set values outside definition interval to zero.
                # (default=?)
}
```

```
constant_spectrum{
    reflectivity = 0.5  # real value >= 0.0 and <= 1.0 (dimensionless)
}
```

7.15.4 optics{ global_absorption_coeff{ } }

- **usage:** optional
- **items:** maximum 1

This group is used to specify the global absorption spectrum for the entire device.

Important

The following general conditions must be satisfied when defining *optics{ global_absorption_coeff{ } }*

- Exactly one of the following must be defined: *database_spectrum{ }*, *import_spectrum{ }*, *constant_spectrum{ }* within this group.

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

---

database_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing absorption spectra from the database or external files.

---

database_spectrum{ name }

- **usage:** required
- **type:** character string

Name of the spectrum contained in the database.

---

import_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing spectrum from a file

Important

The following general conditions must be satisfied when defining *import_spectrum{ }*

- *import{ }* is specified in the input file.

---

import_spectrum{ import_from }

- **usage:** required
- **type:** character string

Path to a spectrum for importing

---

import_spectrum{ cutoff }

- **usage:** required
- **type:** choice
- **values:** `yes` or `no`

If set to `yes`, then the values of the spectrum which are outside the definition interval are set to zero. Otherwise, the spectrum is extrapolated as a constant with the value on the boundary of the imported data.

---

import_spectrum{ energy_spectrum }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the imported spectrum is assumed to be given as a function of energy. Otherwise, the spectrum is assumed to be given as a function of wavelength.

---

import_spectrum{ decadic_absorption_unit }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical absorption coefficient is assumed to be expressed in $\mathrm{dB}/\mu\mathrm{m}$.

---

constant_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Specify a constant absorption spectrum

Important

The following general conditions must be satisfied when defining *constant_spectrum{ }*

- Exactly one of the following must be defined: `absorption_coeff`, `decadic_absorption_coeff` within this group.

---

constant_spectrum{ absorption_coeff }

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** cm^{-1}

The constant value of the absorption coefficient expressed in $1/cm$

---

constant_spectrum{ decadic_absorption_coeff }

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** $\mathrm{dB}/\mu\mathrm{m}$

The constant value of the absorption coefficient expressed in $\mathrm{dB}/\mu\mathrm{m}$

Examples

```
global_absorption_coeff{
    database_spectrum{
        name = "GaAs"
    }
}
```

```
global_absorption_coeff{
    import_spectrum{
        import_from = "filename"
        cutoff = yes  # yes/no: If yes, set values outside definition interval to␣
↪zero.
                     # (default=?)
        decadic_absorption_unit = no # yes or no, default: no
    }
}
```

```
global_absorption_coeff{
    constant_spectrum{
        absorption = 0.5  # real value >= 0.0 [1/cm]
        # or
        decadic_absorption = 0.0 # real value >= 0.0
    }
}
```

7.15.5 optics{ global_refractive_index{ } }

- **usage:** optional
- **items:** maximum 1

This group is used to specify the effective refractive index $n_{\mathrm{eff}}(\lambda)$ of the modelled device.

Important

The following general conditions must be satisfied when defining *optics{ global_refractive_index{ } }*

- Exactly one of the following must be defined: *database_spectrum{ }*, *import_spectrum{ }*, *constant_spectrum{ }* within this group.

- *Maintained Keywords*

  - *database_spectrum{ }*
  - *database_spectrum{ name }*
  - *import_spectrum{ }*
  - *import_spectrum{ import_n_from }*
  - *import_spectrum{ import_k_from }*
  - *import_spectrum{ cutoff }*
  - *import_spectrum{ energy_spectrum }*
  - *constant_spectrum{ }*
  - *constant_spectrum{ n }*
  - *constant_spectrum{ k }*
  - *compute_absorption_coeff{ }*

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

database_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing the spectrum from the database or external files.

database_spectrum{ name }

- **usage:** required
- **type:** character string

Name of the spectrum contained in the database.

import_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Importing spectrum from a file

Important

The following general conditions must be satisfied when defining *import_spectrum{ }*

- The global group *import{ }* is specified in the input file.

import_spectrum{ import_n_from }

- **usage:** required
- **type:** character string

Path to a spectrum of the real part of the refractive index for importing

---

import_spectrum{ import_k_from }

- **usage:** optional
- **type:** character string

Path to a spectrum of the imaginary part of the refractive index for importing

---

import_spectrum{ cutoff }

- **usage:** required
- **type:** choice
- **values:** `yes` or `no`

If set to `yes`, then the values of the spectrum which are outside the definition interval are set to zero. Otherwise, the spectrum is extrapolated as a constant with the value on the boundary of the imported data.

---

import_spectrum{ energy_spectrum }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the imported spectrum is assumed to be given as a function of energy. Otherwise, the spectrum is assumed to be given as a function of wavelength.

---

constant_spectrum{ }

- **usage:** optional
- **items:** maximum 1

Specify a constant refractive index spectrum

---

constant_spectrum{ n }

- **usage:** required
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** –
- **default:** $r = 1.0$

Constant value of the real part of the refractive index.

constant_spectrum{ k }

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** –
- **default:** $r = 0.0$

Constant value of the imaginary part of the refractive index.

compute_absorption_coeff{ }

- **usage:** required
- **items:** maximum 1

When defined, then calculates absorption coefficient from imaginary part of the reflective index.

7.15.6 optics{ light_propagation{ } }

- **usage:** optional
- **items:** maximum 1

specifying options related to the light field propagating through the device.

Dependencies

- *optics{ global_illumination{ } }* is specified in the input file.
- Exactly one of the following must be defined: *min_wavelength*, *min_energy* within this group.
- Exactly one of the following must be defined: *max_wavelength*, *max_energy* within this group.
- Maximum one of *use_local_spectra{ }* and *use_computed_spectra{ }* can be defined within this group.

- *Maintained Keywords*
  - *min_wavelength*
  - *max_wavelength*
  - *min_energy*

  - *max_energy*
  - *use_global_spectra{ }*
  - *use_global_spectra{ energy_resolution }*
  - *use_local_spectra{ }*
  - *use_local_spectra{ energy_resolution }*
  - *use_computed_spectra{ }*
  - *output_global_spectra{ }*
  - *output_global_spectra{ reflectivity }*
  - *output_global_spectra{ absorption_coeff }*
  - *output_global_spectra{ decadic_absorption_coeff }*
  - *output_global_spectra{ refractive_index }*
  - *output_global_spectra{ spectra_over_energy }*
  - *output_global_spectra{ spectra_over_frequency }*
  - *output_global_spectra{ spectra_over_wavenumber }*
  - *output_global_spectra{ spectra_over_wavelength }*
  - *output_local_spectra{ }*
  - *output_local_spectra{ absorption_coeff }*
  - *output_local_spectra{ decadic_absorption_coeff }*
  - *output_local_spectra{ spectra_over_energy }*
  - *output_local_spectra{ spectra_over_frequency }*
  - *output_local_spectra{ spectra_over_wavenumber }*
  - *output_local_spectra{ spectra_over_wavelength }*
  - *output_light{ }*
  - *output_light{ illumination }*
  - *output_light{ total_absorption }*
  - *output_light{ total_transmission }*
  - *output_light{ lightflux }*
  - *output_light{ spectra_over_energy }*
  - *output_light{ spectra_over_frequency }*
  - *output_light{ spectra_over_wavenumber }*
  - *output_light{ spectra_over_wavelength }*
  - *output_light{ photon_spectra }*
  - *output_light{ power_spectra }*

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

---

min_wavelength

- **usage:** optional
- **type:** real number
- **values:** $10.0 \leq r \leq 10^6$
- **unit:** nm

—

---

max_wavelength

- **usage:** optional
- **type:** real number
- **values:** $10.0 \leq r \leq 10^6$
- **unit:** nm

—

---

min_energy

- **usage:** optional
- **type:** real number
- **values:** $10^{-6} \leq r \leq 10^2$
- **unit:** eV

Low-energy boundary of the energy grid for propagating photons.

---

max_energy

- **usage:** optional
- **type:** real number
- **values:** $10^{-6} \leq r \leq 10^2$
- **unit:** eV

High-energy boundary of the energy grid for propagating photons.

---

use_global_spectra{ }

- **usage:** optional
- **items:** maximum 1

Light propagation model uses single imported global absorption spectrum for all regions. It cannot be computed during the runtime. The absorption spectrum is also assigned to every region with any boundary conditions (contact regions).

---

use_global_spectra{ energy_resolution }

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV
- **default:** $r = 1e - 2$

Spacing between subsequent energy grid points.

---

use_local_spectra{ }

- **usage:** optional
- **items:** maximum 1

Light propagation model uses single imported global absorption spectrum within local absorption framework. Regions with boundary conditions imposed on the Poisson equation (electric potential) are treated as perfectly transparent, zero absorption coefficient is assigned.

Note

In the future, this feature is planned to use imported position-dependent optical absorption spectra.

Hint

See *contacts{ }* for further reference on boundary conditions.

---

use_local_spectra{ energy_resolution }

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV
- **default:** $r = 1e - 2$

Spacing between subsequent energy grid points.

use_computed_spectra{ }

- **usage:** optional
- **items:** maximum 1

Light propagation model uses absorption spectrum within local absorption framework which is computed within the runtime. Regions with boundary conditions imposed on the Poisson equation (electric potential) are treated as perfectly transparent, zero absorption coefficient is assigned.

Note

Broadening is not included.

Hint

See *contacts{ }* for further reference on boundary conditions.

Dependencies

- *energy_grid{ }* must be defined.
- *optics{ semiclassical_spectra{ } }* must be defined.
- *local_absorption* must be defined.
- *optics{ global_absorption_coeff{ } }* is not specified in the input file.
- *optics{ global_refractive_index{ } }* is not specified in the input file.

output_global_spectra{ }

- **usage:** optional
- **items:** maximum 1

This group is used to output optical spectra which entered the calculation of the light propagation through the device.

output_global_spectra{ reflectivity }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the reflectivity spectrum is outputted.

output_global_spectra{ absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the absorption spectrum is outputted.

---

output_global_spectra{ decadic_absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the absorption spectrum in decadic units is outputted.

---

output_global_spectra{ refractive_index }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the refractive index spectrum is outputted.

---

output_global_spectra{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes` then the selected spectra are outputted over photon energy.

---

output_global_spectra{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon frequency.

---

output_global_spectra{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon wavenumber.

---

output_global_spectra{ spectra_over_wavelength }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon wavelength.

---

output_local_spectra{ }

- **usage:** optional
- **items:** exactly 1

This group is used to output optical spectra which entered the calculation of the light propagation through the device within the framework of locally defined spectra.

---

output_local_spectra{ absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the absorption spectrum is outputted.

---

output_local_spectra{ decadic_absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the absorption spectrum in decadic units is outputted.

---

output_local_spectra{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes` then the selected spectra are outputted over photon energy.

output_local_spectra{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon frequency.

output_local_spectra{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon wavenumber.

output_local_spectra{ spectra_over_wavelength }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon wavelength.

output_light{ }

- **usage:** required
- **items:** exactly 1

—

output_light{ illumination }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the illumination spectrum is outputted.

---

output_light{ total_absorption }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the total_absorption is outputted, i.e. the fraction of absorbed photons in the device relative to the number of incident photons for each wavelength.

---

output_light{ total_transmission }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the total_transmission is outputted, i.e. the fraction of absorbed photons in the device relative to the number of incident photons for each wavelength, i.e. the fraction of transmitted photons through the device relative to the number of incident photons for each wavelength.

---

output_light{ lightflux }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the light flux $I(x, E)$ of the light propagating through the device

---

output_light{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes` then the selected spectra are outputted over photon energy.

output_light{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon frequency.

---

output_light{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon wavenumber.

---

output_light{ spectra_over_wavelength }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` then the selected spectra are outputted over photon wavelength.

---

output_light{ photon_spectra }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then spectrum of photon number is outputted with one of the following units $1/cm^2/s/eV$, $1/cm^2/s/nm$, $1/cm^2/s/THz$, or $1/cm^2/s/cm^{-1}$.

---

output_light{ power_spectra }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then photon power spectrum is outputted with units W/cm^2.

7.15.7 optics{ photogeneration{ } }

- **usage:** optional
- **items:** maximum 1

Triggers position-dependent generation rates, which are included in the current solver. Output generated carriers $G(x)$ and $G(x, E)$ due to photon absorption.

Important

The following general conditions must be satisfied when defining *optics{ photogeneration{ } }*

- *optics{ light_propagation{ } }* is specified in the input file.

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

---

output

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the generation rate as function of position $G(x)$ is outputted.

---

output_integrated

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the generation rate as function of energy $G(E)$ is outputted.

---

output_energy_resolved

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the generation rate as function of position and energy $G(x, E)$ is outputted.

Examples

```
photo_generation{
    output = yes                         # yes/no (default: yes)
    output_integrated = yes                # yes/no (default: no)
}
```

7.15.8 optics{ semiclassical_spectra{ } }

- **usage:** optional
- **items:** maximum 1

Compute and output emission spectra calculated from energy-resolved densities $n(x, E)$ and $p(x, E)$ computed by energy_resolved_density{}. Radiative recombination rate reads $R_{\text{radiative}}(x, E) = C(x) \int dE_h \int dE_e \; n(x, E_e) p(x, E_h) \delta(E_e - E_h - E)$, where $C(x)$ [cm^3/s] is the (material-dependent) radiative recombination parameter. “spectra” and “density” in the following refer to the integrals of $R_{\text{radiative}}$ over position and energy, respectively.

Dependencies

- All must be defined: *energy_grid{ }* / *energy_resolved_density{ }* / *Gamma{ }*
- At least on of *output_spectra{ }* and *output_local_spectra{ }* must be defined.

- *Maintained Keywords*
  - *refractive_index*
  - *energy_broadening_gaussian*
  - *energy_broadening_lorentzian*
  - *absorption*
  - *emission*
  - *local_absorption*
  - *local_emission*
  - *output_spectra{ }*
  - *output_spectra{ im_epsilon }*
  - *output_spectra{ absorption_coeff }*
  - *output_spectra{ decadic_absorption_coeff }*
  - *output_spectra{ gain }*

  - *output_spectra{ decadic_gain }*
  - *output_spectra{ emission_photons }*
  - *output_spectra{ emission_power }*
  - *output_spectra{ spectra_over_energy }*
  - *output_spectra{ spectra_over_frequency }*
  - *output_spectra{ spectra_over_wavenumber }*
  - *output_spectra{ spectra_over_wavelength }*
  - *output_local_spectra{ }*
  - *output_local_spectra{ im_epsilon }*
  - *output_local_spectra{ absorption_coeff }*
  - *output_local_spectra{ decadic_absorption_coeff }*
  - *output_local_spectra{ gain }*
  - *output_local_spectra{ decadic_gain }*
  - *output_local_spectra{ emission_photons }*
  - *output_local_spectra{ emission_power }*
  - *output_local_spectra{ spectra_over_energy }*
  - *output_local_spectra{ spectra_over_frequency }*
  - *output_local_spectra{ spectra_over_wavenumber }*
  - *output_local_spectra{ spectra_over_wavelegth }*
  - *output_photon_density*
  - *output_power_density*

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

refractive_index

- **usage:** optional
- **type:** real number
- **values:** `[1.0, ...)`
- **unit:** –
- **default:** `substrate`

Average refractive index n_r. Refractive index used for calculating gain and absorption spectra. The absorption/gain spectra is multiplied by the factor $1/n_r^2$. The values for the optical dielectric constant from the database are not used yet at this point.

energy_broadening_gaussian

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

—

energy_broadening_lorentzian

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

—

absorption

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

—

emission

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

—

local_absorption

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

—

local_emission

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

—

---

output_spectra{ }

- **usage:** optional
- **items:** maximum 1

When this group is defined then optical spectra computed within semi-classical models (based on carrier densities) are saved to the output folder. The spectra are averaged over the entire simulation domain.

---

output_spectra{ im_epsilon }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

The upper 30% of the spectra are cut off.

---

output_spectra{ absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Absorption spectra are outputted, both positive and negative parts. The upper 30% of the spectra are cut off.

---

output_spectra{ decadic_absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Decadic absorption spectra are outputted, both positive and negative parts. The upper 30% of the spectra are cut off.

---

output_spectra{ gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Gain spectra are outputted, only the positive part. The upper 30% of the spectra are cut off.

---

output_spectra{ decadic_gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Decadic gain spectra are outputted, only the positive part. The upper 30% of the spectra are cut off.

---

output_spectra{ emission_photons }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Photon emission spectra are outputted, only the positive part is shown. Stimulated emission assumes that all photon modes are occupied by one photon. Thus, not the actual stimulated emission in the device is calculated, but rather a spectral response similar to the gain.

Note

The model is not suitable for systems with occupation inversion, above the threshold. It can be successfully used for modeling, e.g., LEDs.

---

output_spectra{ emission_power }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Power emission spectra are outputted, only the positive part is shown. Stimulated emission assumes that all photon modes are occupied by one photon. Thus, not the actual stimulated emission in the device is calculated, but rather a spectral response similar to the gain.

Note

The model is not suitable for systems with occupation inversion, above the threshold. It can be successfully used for modeling, e.g., LEDs.

---

output_spectra{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

selected spectra are outputted over energy

---

output_spectra{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

selected spectra are outputted over frequency

---

output_spectra{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

selected spectra are outputted over wavenumber

---

output_spectra{ spectra_over_wavelength }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

selected spectra are outputted over wavelength

---

output_local_spectra{ }

- **usage:** optional
- **items:** maximum 1

When this group is defined then optical spectra computed within semi-classical models (based on carrier densities) are saved to the output folder. The spectra are position-dependent within the simulation domain.

---

output_local_spectra{ im_epsilon }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

The upper 30% of the spectra are cut off.

---

output_local_spectra{ absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Absorption spectra are outputted, both positive and negative parts. The upper 30% of the spectra are cut off.

---

output_local_spectra{ decadic_absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Decadic absorption spectra are outputted, both positive and negative parts. The upper 30% of the spectra are cut off.

---

output_local_spectra{ gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Gain spectra are outputted, only the positive part. The upper 30% of the spectra are cut off.

---

output_local_spectra{ decadic_gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Decadic gain spectra are outputted, only the positive part. The upper 30% of the spectra are cut off.

---

output_local_spectra{ emission_photons }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Photon emission spectra are outputted, only the positive part is shown. Stimulated emission assumes that all photon modes are occupied by one photon. Thus, not the actual stimulated emission in the device is calculated, but rather a spectral response similar to the gain.

Note

The model is not suitable for systems with occupation inversion, above the threshold. It can be successfully used for modeling, e.g., LEDs.

---

output_local_spectra{ emission_power }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Power emission spectra are outputted, only the positive part is shown. Stimulated emission assumes that all photon modes are occupied by one photon. Thus, not the actual stimulated emission in the device is calculated, but rather a spectral response similar to the gain.

Note

The model is not suitable for systems with occupation inversion, above the threshold. It can be successfully used for modeling, e.g., LEDs.

---

output_local_spectra{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`

- **default:** `yes`

selected spectra are outputted over energy

---

output_local_spectra{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

selected spectra are outputted over frequency

---

output_local_spectra{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

selected spectra are outputted over wavenumber

---

output_local_spectra{ spectra_over_wavelegth }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

selected spectra are outputted over wavelegth

---

output_photon_density

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output emitted photon density in $\mathrm{cm}^{-3}\mathrm{s}^{-1}$ to *emitted_photon_density.dat*

---

output_power_density

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output emitted power density in $\mathrm{W/cm^3}$ to *emitted_power_density.dat*

Last update: 10/12/2024

7.15.9 optics{ quantum_spectra{ } }

- **usage:** optional
- **items:** no constraints

This group specifies numerical properties of the quantum model used for computations of optical spectra base on the Fermi's Golden Rule.

> **Note**
>
> Our algorithms and models controlled by keywords in this group are intensively developed. For this reason, related syntax may substantially change with each next release. Users of this group are highly encouraged to update the tool regularly with the new releases and to use our support system to give us feedback on any related issues.

> **Note**
>
> In the current versions, this group should not be used for modeling optical spectra for transitions between two separate 1-band models (e.g., triggered by *Gamma{ }* and *HH{ }*) or between a 1-band model and 6-band model (e.g., *Gamma{ }* and *kp_6band{ }*). Computations within single models (e.g., only within *kp_8band{ }*, only within *Gamma{ }*, etc.) are supported.

Dependencies

- The global group *quantum{ }* must be defined.
- Up to one of *interband_approximation* and *intraband_approximation* can be defined.
- Up to one of *occupation_interpolate_invfermi* and *occupation_zero_fermilevel* can be defined.
- At least one of *energy_broadening_gaussian* and *energy_broadening_lorentzian* must be defined.
- The *k_integration{ }* must be defined if any of *simulate1D{ }* or *simulate2D{ }* is defined.
- The *excitons{ }* is not allowed to be defined if any of *simulate2D{ }* or *simulate3D{ }* is defined.
- The *k_integration{ }* is not allowed to be defined if *simulate3D{ }* is defined.
- None of *occupation_zero_fermilevel* and *occupation_interpolate_invfermi* are allowed to be defined if *simulate3D{ }* is defined.
- The *spin_align* is not allowed to be defined if *global{ magnetic_field{ } }* is defined.
- *output_energies*, *output_occupations*, *output_transitions*, and *output_spinor_components* are not allowed if *simulate3D{ }* is already specified in the *global{ }* group.

- The groups *output_energies*, *output_occupations*, *output_transitions*, and *output_spinor_components* are not allowed if the group *simulate3D{ }* is defined.

- *Maintained Keywords*
    - *name*
    - *spin_align*
    - *interband*
    - *intraband*
    - *interband_approximation*
    - *intraband_approximation*
    - *enable_hole_hole*
    - *enable_electron_hole*
    - *enable_electron_electron*
    - *use_kp8_EP*
    - *k_integration{ }*
    - *k_integration{ relative_size }*
    - *k_integration{ num_points }*
    - *k_integration{ num_integrationpoints }*
    - *k_integration{ force_k0_subspace }*
    - *energy_threshold*
    - *transition_threshold*
    - *occupation_threshold*
    - *occupation_ignore*
    - *occupation_zero_fermilevel*
    - *occupation_interpolate_invfermi*
    - *classify_none{ }*
    - *classify_by_energy{ }*
    - *classify_by_energy{ method }*
    - *classify_by_energy{ shift_electrons }*
    - *classify_by_energy{ shift_holes }*
    - *classify_by_spinor{ }*
    - *classify_by_spinor{ threshold_electron }*
    - *classify_by_spinor{ threshold_hole }*
    - *classify_states*
    - *classification_threshold*
    - *excitons{ }*
    - *excitons{ num_exciton_levels }*
    - *excitons{ coulomb_enhancement }*

  - *absorption*
  - *spontaneous_emission*
  - *local_absorption*
  - *local_spontaneous_emission*
  - *polarization{ }*
  - *polarization{ name }*
  - *polarization{ re }*
  - *polarization{ im }*
  - *refractive_index*
  - *normalization_volume*
  - *min_energy*
  - *max_energy*
  - *energy_resolution*
  - *energy_broadening_gaussian*
  - *energy_broadening_lorentzian*
  - *kramers_kronig{ }*
  - *kramers_kronig{ im_epsilon_extension }*
  - *kramers_kronig{ im_epsilon_rescale }*
  - *kramers_kronig{ delta_static_epsilon }*
  - *kramers_kronig{ delta_position }*
  - *kramers_kronig{ delta2_static_epsilon }*
  - *kramers_kronig{ delta2_position }*
  - *kramers_kronig{ delta3_static_epsilon }*
  - *kramers_kronig{ delta3_position }*
  - *kramers_kronig{ use_for_absorption }*
  - *kramers_kronig{ use_for_emission }*
  - *output_energies*
  - *output_occupations*
  - *output_transitions*
  - *output_spinor_components*
  - *output_spectra{ }*
  - *output_spectra{ im_epsilon }*
  - *output_spectra{ absorption_coeff }*
  - *output_spectra{ decadic_absorption_coeff }*
  - *output_spectra{ gain }*
  - *output_spectra{ decadic_gain }*
  - *output_spectra{ re_epsilon }*
  - *output_spectra{ refractive_index }*

Maintained Keywords

The keywords below are available in at least one of currently published releases and are planned to be included also in the next release.

name

- **usage:** required
- **type:** character string

The name of already defined region in *region{ }* for which optical generation should be calculated. Multiple numerical parameters are inherited after the definitions in the *region{ }* referred to.

spin_align

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes` for Pauli equation solved with 6-band or 8-band $\mathbf{k}\cdot\mathbf{p}$ method, a spin-basis transformation is performed for each pair of quantum states (i, i+1), with i being an odd number, such that matrix representation of the Pauli operator $\hat{\sigma}$ multiplied by a selected versor (along the z direction in 3D, and the x direction in 1D and 2D) becomes diagonal in the subspace defined by these two states. With other words, spinor compositions of degenerate (due to lack of magnetic field) pairs of quantum states are chosen as if magnetic field was parallel to the z direction (3D) or x direction (1D, 2D). This procedure is triggered before running an algorithm computing optical spectra.

interband

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Compute optical transitions dominating in interband transitions, typically conduction band to valence band transitions.

intraband

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Compute optical transitions dominating in intraband transitions, typically conduction band to conduction band transitions.

interband_approximation

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Only terms of the type $< c|p|v >$ and $< v|p|c >$ are taken into account ($c = s$ and $v = x, y, z$)

---

intraband_approximation

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Only terms of the type $< c|p|c >$ and $< v|p|v >$ are taken into account ($c = s$ and $v = x, y, z$)

---

enable_hole_hole

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If `yes` then transitions within valence bands are included according to applied classification.

---

enable_electron_hole

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If `yes` then transitions between conduction and valence bands are included according to applied classification.

---

enable_electron_electron

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If `yes` then transitions within conduction bands are included according to applied classification.

---

use_kp8_EP

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If `yes` then uses the P parameter from 8-band $\mathbf{k}\cdot\mathbf{p}$ material data is used to compute the strength of optical transitions when computing the spectra between 2 states computed within 1-band model, and when computing the spectra with conduction band expressed within 1-band model and valence bands within 6-band $\mathbf{k}\cdot\mathbf{p}$ model.

---

k_integration{ }

- **usage:** optional
- **items:** maximum 1

Group defining numerical parameters of integration over the states in the space of the wave vector $k_{||}$ space.

---

k_integration{ relative_size }

- **usage:** optional
- **type:** real number
- **values:** $10^{-3} \leq r \leq 1.0$
- **unit:** $-$
- **default:** $r = 1e-1$

Size of the integrated volume of the $k_{||}$ space expressed as relative value to the size of the First Brillouin Zone

---

k_integration{ num_points }

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 100$
- **unit:** $-$
- **default:** $z = 5$

Number of points counted from $k = 0$ to the border of considered $k_{||}$ space along $k_{||} = k_y$ or k_z excluding the point at $k = 0$. The Schrödinger equation is solved for optical spectra at the grid with the "radius" as described above. The transition intensities are computed at these points and later used in the integration procedure.

---

k_integration{ num_integrationpoints }

- **usage:** optional
- **type:** integer
- **values:** $z \geq 1$
- **unit:** $-$
- **default:** $z = 180$

Number of integration points in the $k_{||}$ defining an independent grid analogously as the attribute *k_integration{ num_points }*.

Spline interpolation at the grid defined with *k_integration{ num_integrationpoints }* of all quantities necessary for computation of the optical spectra is performed in the $k_{||}$ space based on solution obtained at the grid defined with the attribute *k_integration{ num_points }*. The transition intensities and energies resulting from this interpolation are integrated and included in the optical spectra.

> ⚠ **Warning**
>
> Assigning too small value to *k_integration{ num_integrationpoints }* may result in artificial oscillatory results in the spectra.

---

k_integration{ force_k0_subspace }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, $k_{||}$ integration is modified in a way that only states for point $k = 0$ are computed exactly, whereas for all other k points the wave functions are computed in the subspace of the solutions for the $k = 0$. Computational speed is notably improved as a result of this approximation. Therefore enlarging the number of eigenvalues included in the computation becomes more feasible.

> ⚠ **Attention**
>
> This approximation should be used carefully as it reduces accuracy of computed optical spectra.

---

energy_threshold

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** eV
- **default:** $r = 1e-6$

Only transitions between states with at least this energy difference are regarded when computing optical spectra.

transition_threshold

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** eV
- **default:** $r = 1e - 6$

Only transitions between states with at least this optical intensity are regarded when computing optical spectra. Increasing the value can reduce computational time but may neglect weak optical transitions.

occupation_threshold

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $-$
- **default:** $r = 0.0$

Only transitions between states with at least this occupation are regarded when computing optical spectra. Increasing the value can reduce computational time but may neglect weakly occupied states.

occupation_ignore

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Ignore the occupation of states when computing optical spectra: Valence bands and conduction bands are considered to be fully occupied and fully empty, respectively.

Warning

This feature is under development.

Attention

Occupation and classification of states are currently performed independently for carrier densities and for optical spectra.

occupation_zero_fermilevel

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

This keyword is active when *occupation_ignore* is set to `no`. In semi-classical current calculations, the quasi-Fermi level may depend on position. Optical spectra, on the other, hand are computed using a quantum mechanical model with where single states involved in the transitions exhibit non-locality (wave functions) resulting in their existence in areas with different quasi-Fermi levels assigned. As the model for the spectra assumes a specific quasi-Fermi level for each state, the inconsistency arises. Using this keyword set to `yes` resolves this inconsistency by taking both quasi-Fermi levels equal zero. Taking it `no`, position dependent occupation number is computed.

⚠ **Warning**

This feature is under development.

---

occupation_interpolate_invfermi

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

This keyword is active when *occupation_ignore* and *occupation_zero_fermilevel* are set to `no`. If `yes` then Fermi levels are interpolated between k-points before applying to the integrating algorithm which may increase accuracy of numerical $k_{||}$ space integration.

⚠ **Warning**

This feature is under development.

---

classify_none{ }

- **usage:** optional
- **items:** maximum 1

---

classify_by_energy{ }

- **usage:** optional
- **items:** maximum 1

---

classify_by_energy{ method }

Calling sequence

`optics{ quantum_spectra{ classify_by_energy{ method } } }`

Properties

- **usage:** optional
- **type:** integer
- **values:** $0 \leq z \leq 2$
- **default:** $z = 2???$

Functionality

—

---

classify_by_energy{ shift_electrons }

Calling sequence

`optics{ quantum_spectra{ classify_by_energy{ shift_electrons } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

—

---

classify_by_energy{ shift_holes }

Calling sequence

`optics{ quantum_spectra{ classify_by_energy{ shift_holes } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV

Functionality

—

---

classify_by_spinor{ }

- **usage:** optional
- **items:** maximum 1

---

classify_by_spinor{ threshold_electron }

Calling sequence

`optics{ quantum_spectra{ classify_by_spinor{ threshold_electron } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-2} \leq r \leq 0.99$
- **unit:** $-$

Functionality

—

---

classify_by_spinor{ threshold_hole }

Calling sequence

`optics{ quantum_spectra{ classify_by_spinor{ threshold_hole } } }`

Properties

- **usage:** optional
- **type:** real number
- **values:** $10^{-2} \leq r \leq 0.99$
- **unit:** $-$

Functionality

—

---

classify_states

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Classifies states as electrons if energy is higher than average value of minimum of the conduction band and maximum of the valence, $(EC_{min} + EV_{max})/2$, plus *classification_threshold*.

classification_threshold

- **usage:** optional
- **type:** real number
- **values:** no constraints
- **unit:** eV
- **default:** $r = 0.0$

A parameter shifting the reference energy for the classification of the states.

---

excitons{ }

- **usage:** optional
- **items:** maximum 1

Include excitonic effects.

Attention

Excitons are implemented only for 1D simulations.

---

excitons{ num_exciton_levels }

- **usage:** optional
- **type:** integer
- **values:** $1 \leq z \leq 10$
- **unit:** $-$
- **default:** $z = 1$

Number of exciton levels included in the model.

---

excitons{ coulomb_enhancement }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the Coulomb enhancement factor, also known as the Sommerfeld factor, is taken into account.

absorption

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

—

---

spontaneous_emission

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Calculate spontaneous emission rate using the momentum matrix element obtained by 8-band kp model. (This feature is not yet implemented in 3D simulation.)

---

local_absorption

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Absorption spectrum within local framework is computed and can be outputted using *output_local_spectra{ }*. Regions with boundary conditions imposed on the Poisson equation (electric potential) are treated as perfectly transparent, zero absorption coefficient is assigned.

Hint

See *contacts{ }* for further reference on boundary conditions.

Warning

The feature is experimental and may produce unphysical results.

---

local_spontaneous_emission

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Spontaneous emission spectrum within local framework is computed and can be outputted using *output_local_spectra{ }*. Regions with boundary conditions imposed on the Poisson equation (electric potential) are treated as perfectly transparent, zero absorption coefficient is assigned.

Hint

See *contacts{ }* for further reference on boundary conditions.

Warning

The feature is experimental and may produce unphysical results.

---

polarization{ }

- **usage:** required
- **items:** no constraints

Define polarization of incoming light for which optical absorption spectrum should be calculated.

Important

- At least one of the following must be specified within this group, *polarization{ re }*, *polarization{ im }*.

---

polarization{ name }

- **usage:** required
- **type:** character string

name attached to output files with computed spectra for the defined polarization

---

polarization{ re }

- **usage:** optional
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **unit:** $-$
- **default:** $r_1 = 0.0$, $r_2 = 0.0$, $r_3 = 0.0$

real part of the polarization vector

polarization{ im }

- **usage:** optional
- **type:** vector of 3 real numbers: (r_1, r_2, r_3)
- **values:** no constraints
- **unit:** $-$
- **default:** $r_1 = 0.0$, $r_2 = 0.0$, $r_3 = 0.0$

imaginary part of the polarization vector

---

refractive_index

- **usage:** optional
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** $-$
- **default:** `substrate`

Specify constant refractive index for the simulation of the optical spectra.

---

normalization_volume

- **usage:** optional
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** $\mathrm{nm}^{\mathrm{dimension}}$
- **default:** `related quantum region`

Specifies normalization volume for the optical spectra.

---

min_energy

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** eV
- **default:** $r = 0.0$

lower energy bound for optical spectra

max_energy

- **usage:** optional
- **type:** real number
- **values:** `[1e-3, ...)`
- **unit:** eV
- **default:** $r = 2.0$

upper energy bound for optical spectra

---

energy_resolution

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV
- **default:** $r = 1e - 3$

Spacing between subsequent energy grid points.

---

energy_broadening_gaussian

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

Set the broadening to value greater than 0.0 to make the Gaussian broadening

$$\mathcal{L}(E - E_0) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ \left(- \frac{(E - E_0)^2}{2\sigma^2} \right) \right\}$$

included to the calculation of the optical spectrums. The specifed value is read as the FWHM $\Gamma = 2\sqrt{\ln 2} \cdot \sigma$.

(In 1D and 2D, both Lorentzian and Gaussian can be used simultaneously. In 3D, either of these broadenings must be included.)

---

energy_broadening_lorentzian

- **usage:** optional
- **type:** real number
- **values:** `[1e-6, ...)`
- **unit:** eV

Set the broadening to value greater than 0.0 to make the Lorentzian broadening

$$\mathcal{L}(E - E_0) = \frac{1}{\pi} \frac{\Gamma/2}{(E - E_0) + (\Gamma/2)^2}$$

included to the calculation of the optical spectrums. The specifed value is read as the FWHM Γ.

---

kramers_kronig{ }

- **usage:** optional
- **items:** maximum 1

If specified, then Kramers-Kronig relations are used to evaluate real part of dielectric function and dispersion of complex refractive index based on previously computed imaginary part of dielectric function.

Attention

Available Hamiltonians, defined within 1-band, 6-band, or 8-band $\mathbf{k}\cdot\mathbf{p}$ models, will contribute to the **imaginary part of dielectric function** ε_i only with transitions close to the Γ point, therefore, underestimating the spectrum at higher energies. As Kramers-Kronig relations are non-local, the transformation of such ε_i is reproducing **real part of dielectric function** ε_r accurately only up to slow-varying background. The missing background accounts for not-computed high-energy ε_i. Therefore only local features of real part of dielectric function are accessible within the transformation.

To handle this problem, the missing background can be approximated analytically assuming additional contributions from ε_i at high energies with parameters: *kramers_kronig{ im_epsilon_extension }*, *kramers_kronig{ im_epsilon_rescale }*, *kramers_kronig{ delta_static_epsilon }*, and *kramers_kronig{ delta_position }*. These contributions are not shown in the ε_i output, but their effect is present in ε_r output.

Note

Specific values of parameters: *kramers_kronig{ im_epsilon_extension }*, *kramers_kronig{ im_epsilon_rescale }*, *kramers_kronig{ delta_static_epsilon }*, and *kramers_kronig{ delta_position }* have to be fitted individually for every device. No tables for materials nor devices are available.

---

kramers_kronig{ im_epsilon_extension }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** eV
- **default:** $r = 0.0$

If *kramers_kronig{ im_epsilon_extension }* is set to non-zero value then ε_i computed at *max_energy* multiplied by *kramers_kronig{ im_epsilon_rescale }* is assumed for ε_i in an energy range from *max_energy* to *max_energy* + *kramers_kronig{ im_epsilon_extension }*. Effectively a rectangle is attached to the end of the spectra with width of *kramers_kronig{ im_epsilon_extension }* and height of the ε_i at *max_energy* multiplied by *kramers_kronig{ im_epsilon_rescale }*, to be used in Kramers-Kronig transformation.

---

kramers_kronig{ im_epsilon_rescale }

- **usage:** optional
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** –
- **default:** $r = 1.0$

This parameter is rescaling value used to approximate constant ε_i at high energies, from *max_energy* to *max_energy* + *kramers_kronig{ im_epsilon_extension }*. When *kramers_kronig{ im_epsilon_rescale }* $= 1$ then exactly ε_i at *max_energy* is used.

---

kramers_kronig{ delta_static_epsilon }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $-$
- **default:** $r = 0.0$

If this attribute is set to non-zero value then Dirac delta-function is added to ε_i at energy *kramers_kronig{ delta_position }* to be used in Kramers-Kronig transformation. The Dirac delta-function is scaled such that it results in $\varepsilon_r\,(0)$ equal to this attribute.

---

kramers_kronig{ delta_position }

- **usage:** optional
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** eV

This parameter is defining energy at which the Dirac delta function is added to ε_i.

---

kramers_kronig{ delta2_static_epsilon }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $-$
- **default:** $r = 0.0$

If this attribute is set to non-zero value then second Dirac delta-function is added to ε_i at energy *kramers_kronig{ delta_position }* to be used in Kramers-Kronig transformation. The Dirac delta-function is scaled such that it results in $\varepsilon_r\,(0)$ equal to this attribute.

---

kramers_kronig{ delta2_position }

- **usage:** optional
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** eV

This parameter is defining energy at which the second Dirac delta function is added to ε_i.

---

kramers_kronig{ delta3_static_epsilon }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** –
- **default:** $r = 0.0$

If this attribute is set to non-zero value then the third Dirac delta-function is added to ε_i at energy *kramers_kronig{ delta_position }* to be used in Kramers-Kronig transformation. The Dirac delta-function is scaled such that it results in $\varepsilon_r\,(0)$ equal to this attribute.

---

kramers_kronig{ delta3_position }

- **usage:** optional
- **type:** real number
- **values:** `(0.0, ...)`
- **unit:** eV

This parameter is defining energy at which the third Dirac delta function is added to ε_i.

---

kramers_kronig{ use_for_absorption }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then computed refractive index is used to calculate absorption. Otherwise, constant value is used.

---

kramers_kronig{ use_for_emission }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the computed refractive index is used to calculate emission. Otherwise, constant value is used.

---

output_energies

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output energy dispersion for every transition.

---

output_occupations

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output occupation dispersion for every transition.

---

output_transitions

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output transition strength for every transition.

---

output_spinor_components

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output the spinor components for each state at each $k_{||}$ point (only relevant in multi-band $\mathbf{k} \cdot \mathbf{p}$ calculations).

Note

In 1-dimensional systems the axis of quantization for the angular momentum is x, in 3D z.

---

output_spectra{ }

- **usage:** required
- **items:** exactly 1

Control of optical spectra output

---

output_spectra{ im_epsilon }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Imaginary part of dielectric function is outputted.

---

output_spectra{ absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the optical absorption coefficient expressed in cm^{-1} is outputted.

---

output_spectra{ decadic_absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical absorption coefficient is expressed in $\mathrm{dB}/\mu\mathrm{m}$ is outputted.

---

output_spectra{ gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical gain coefficient expressed in cm^{-1} is outputted.

---

output_spectra{ decadic_gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical gain coefficient expressed in $\mathrm{dB}/\mu\mathrm{m}$ is outputted.

---

output_spectra{ re_epsilon }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the real part of dielectric function (relative dielectric permittivity) is outputted.

---

output_spectra{ refractive_index }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then dispersion of refractive index is outputted.

---

output_spectra{ emission_photons }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then spectrum of photon number is outputted with one of the following units $1/cm^2/s/eV$, $1/cm^2/s/nm$, $1/cm^2/s/THz$, or $1/cm^2/s/cm^{-1}$.

---

output_spectra{ emission_power }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then photon power spectrum is outputted with units W/cm^2.

---

output_spectra{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Output spectra with respect to the energy.

output_spectra{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the frequency.

---

output_spectra{ spectra_over_wavelength }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the wavelength.

---

output_spectra{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the wave number.

---

output_component_spectra{ }

- **usage:** required
- **items:** exactly 1

Control of output of components of spectra

If this group is defined then state-to-state spectral components are outputted.

---

output_component_spectra{ threshold_im_epsilon }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $-$
- **default:** $r = 1e - 2$

Only components of dielectric funtion for which transition strength is greater than this attribute are outputted.

---

output_component_spectra{ threshold_emission_photons }

- **usage:** optional
- **type:** real number
- **values:** `[0.0, ...)`
- **unit:** $\mathrm{cm^{-2} s^{-1} eV^{-1}}$ for 1D; $\mathrm{cm^{-1} s^{-1} eV^{-1}}$ for 2D; $\mathrm{s^{-1} eV^{-1}}$ for 3D
- **default:** $r = 10^{18}$ for 1D; $r = 10^{12}$ for 2D; $r = 01^{6}$ for 3D

Only components of emission spectra for which transition strength is greater than this attribute are outputted.

---

output_component_spectra{ im_epsilon }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Imaginary part of dielectric function is outputted.

---

output_component_spectra{ absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the optical absorption coefficient expressed in $\mathrm{cm^{-1}}$ is outputted.

---

output_component_spectra{ decadic_absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical absorption coefficient is expressed in $\mathrm{dB}/\mu\mathrm{m}$ is outputted.

---

output_component_spectra{ gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical gain coefficient expressed in $\mathrm{cm^{-1}}$ is outputted.

---

output_component_spectra{ decadic_gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical gain coefficient expressed in $\mathrm{dB}/\mu\mathrm{m}$ is outputted.

---

output_component_spectra{ emission_photons }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then spectrum of photon number is outputted with one of the following units $1/cm^2/s/eV$, $1/cm^2/s/nm$, $1/cm^2/s/THz$, or $1/cm^2/s/cm^{-1}$.

---

output_component_spectra{ emission_power }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then photon power spectrum is outputted with units W/cm^2.

---

output_component_spectra{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Output spectra with respect to the energy.

---

output_component_spectra{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the frequency.

---

output_component_spectra{ spectra_over_wavelength }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the wavelength.

---

output_component_spectra{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the wave number.

---

output_local_spectra{ }

- **usage:** required
- **items:** exactly 1

Control of output of local optical spectra

---

output_local_spectra{ im_epsilon }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Imaginary part of dielectric function is outputted.

---

output_local_spectra{ absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then the optical absorption coefficient expressed in cm^{-1} is outputted.

output_local_spectra{ decadic_absorption_coeff }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical absorption coefficient is expressed in $\mathrm{dB}/\mu\mathrm{m}$ is outputted.

---

output_local_spectra{ gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical gain coefficient expressed in cm^{-1} is outputted.

---

output_local_spectra{ decadic_gain }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then the optical gain coefficient expressed in $\mathrm{dB}/\mu\mathrm{m}$ is outputted.

---

output_local_spectra{ emission_photons }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

If set to `yes`, then spectrum of photon number is outputted with one of the following units $1/cm^2/s/eV$, $1/cm^2/s/nm$, $1/cm^2/s/THz$, or $1/cm^2/s/cm^{-1}$.

---

output_local_spectra{ emission_power }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

If set to `yes`, then photon power spectrum is outputted with units W/cm^2.

---

output_local_spectra{ spectra_over_energy }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `yes`

Output spectra with respect to the energy.

output_local_spectra{ spectra_over_frequency }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the frequency.

output_local_spectra{ spectra_over_wavelength }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the wavelength.

output_local_spectra{ spectra_over_wavenumber }

- **usage:** optional
- **type:** choice
- **values:** `yes` or `no`
- **default:** `no`

Output spectra with respect to the wave number.

Examples

We can generally write the electric field of a traveling wave propagating to $\mathbf{k}$ direction as follows:

$$\begin{aligned}\mathbf{E}(\mathbf{r};t) &= [E_x\hat{\mathbf{x}} + E_y\hat{\mathbf{y}} + E_z\hat{\mathbf{z}}]e^{\imath[\mathbf{kr}-\omega t]} \\ &= \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} e^{\imath[\mathbf{kr}-\omega t]} = \left(\begin{bmatrix} \mathrm{Re}(E_x) \\ \mathrm{Re}(E_y) \\ \mathrm{Re}(E_z) \end{bmatrix} + \imath \begin{bmatrix} \mathrm{Im}(E_x) \\ \mathrm{Im}(E_y) \\ \mathrm{Im}(E_z) \end{bmatrix} \right) e^{\imath[\mathbf{kr}-\omega t]}\end{aligned}$$

where $E_{x/y/z}$ are complex numbers.

`re=[ , , ]` and `im = [ , , ]` correspond to the first and second column in the last line.

```
# linearly polarized light in x direction.
# name is used for the file names of the output.
polarization{ name = "x"        re = [1,0,0]                }

# linearly polarized light in y direction
polarization{ name = "y"        re = [0,1,0]                }

# linearly polarized light in z direction
polarization{ name = "z"        re = [0,0,1]                }


# TM mode.
# This naming might be useful when analyzing heterostructure
# grown in x direction.
polarization{ name = "TM"        re = [1,0,0]               }

# TE mode
polarization{ name = "TEy"       re = [0,1,0]               }

# TE mode
polarization{ name = "TEz"       re = [0,0,1]               }


# (sigma+) circularly polarized light around the x axis
polarization{ name = "y+iz"     re = [0,1,0]   im = [0,0, 1] }

# (sigma-) circularly polarized light around the x axis
polarization{ name = "y-iz"     re = [0,1,0]   im = [0,0,-1] }


# an example for an arbitrary polarization direction
polarization{ name = "x1y1z2"   re = [1,1,2]                }
```

Last update: 02/04/2025

7.16 database{ }

Using the group **database{ }** allows to modify any parameters of materials defined in the default database. Use of this group might be necessary to obtain results corresponding to real devices or to reproduce other simulations as variety of the parameters are available in the literature established with various accuracies and under various conditions that may be relevant for specific simulation cases.

7.16.1 Top level keywords in database{ }

Top-level attributes in database{ }

> **Note**
>
> This section is under construction

There are two top-level attributes in `database{ }`, namely `default` and `mandatory`. These attributes allow the user to specify the location of the default database containing material parameters.

Note

By default, the program will read in the database which is specified under the installpath (installpath/Syntax/database_nnp.in).

Example: `..  \nextnano\2022_08_05\nextnano++\Syntax\database_nnp.in`

default (optional)

change default path to database

type

string

example

`" .. /Syntax/database_nnp.in"`

Warning

If the location of the database file is specified as a **command line argument**, this has higher priority than the location specified in the input file (attribute `default`).

Example: `nextnano++.exe --database D:\nextnano\2018_10_31\nextnano++\Syntax\database_nnp.in`

If you run *nextnano*++ via *nextnanomat*, the location of the default database is specified in `nextnanomat` ⇒ `Tools` ⇒ `Options` ⇒ `Material database` ⇒ `nextnano++ database file` as a **command line argument**. If you want to use the database location as specified in the input file (attribute `default`), the database location of *nextnanomat* must be empty.

mandatory (optional)

path to database

type

string

example

`"../Syntax/database_nnp.in"`

Note

If a mandatory database is defined, the **command line argument** for the database (`--database ...`) is ignored. This feature can e.g. be used to override the default setting in *nextnanomat* and to specify different databases in various input files or templates, e.g. in conjunction with the feature concatenated string variables (*Input Syntax*), one can dynamically switch between different databases in templates.

Zincblende-related ...zb{} groups in database{ }

Note

This section is under construction

Almost all of the groups related to materials with zincblende symmetry contain 23 groups of identical structure, which are listed in the sections:

- *Bands groups in database{ …_zb{} } and database{ …_wz{} }*
- *Strain groups in database{ …_zb{} } and database{ …_wz{} }*
- *Low-field mobility groups in database{ …_zb{} } and database{ …_wz{} }*
- *High-field mobility groups in database{ …_zb{ } } and database{ …_wz{ } }*
- *Recombination groups in database{ …_zb{} } and database{ …_wz{} }*
- *Phonons in database{ …_zb{} } and database{ …_wz{} }*
- *Other groups in database{ …_zb{} } and database{ …_wz{} }*

The exceptions are `optical_reflectivity{}`, `optical_absorption{}` and `optical_emission{}`, which contain completly differnet groups (see *Optical groups in database{ }* for more information). All other keywords can be found below.

database{ binary_zb{} }

name
specify material name

type
string

example
`GaAs`, `Si`, `GaAs`, `InP`, …

database{ ternary_zb{} }

name

type
string

binary_x
specify name of binary constituent

type
string

binary_1_x
specify name of binary constituent

type
string

database{ ternary2_zb{} }

> **Warning**
>
> Does not contain the groups `lattice_consts{}`, `mass_density{}`, `dielectric_consts{}`, `elastic_consts{}`, `piezoelectric_consts{}`, `acoustic_phonons{}`, `optical_phonons{}`, `conduction_bands{}`, `valence_bands{}`, `kp_6_bands{}`, `kp_8_bands{}`, `mobility_constant{}`, `mobility_masetti{}`, `mobility_arora{}`, `mobility_minimos{}`, `recombination{}`.

name

type
string

binary_x

type
string

binary_1_x

type
string

bowing_x

type
string

bowing_1_x

type
string

database{ bowing_zb{} }

name

type
string

database{ quaternary_zb{} }

name

type
string

binary1

type
string

binary2

type
string

binary3

type
string

ternary12

type
string

ternary13

type
string

ternary23

type
string

database{ quaternary4_zb{} }

name

type
string

binary1

type
string

binary2

type
string

binary3

type
string

binary4

type
string

ternary12

type
string

ternary23

type
string

ternary34

type
string

ternary14

type
string

database{ quinternary_zb{} }

name

type
string

binary_a

type
string

binary_b

type
string

binary_c

type
string

binary_d

type
string

ternary_ab

type
string

ternary_ac

type
string

ternary_ad

type
string

ternary_bc

type
string

ternary_bd

type
string

ternary_cd

type
string

quaternary_abc

type
string

quaternary_abd

type
string

quaternary_acd

type
string

quaternary_bcd

type
string

database{ quinternary6_zb{} }

name

type
string

binary_ad

type
string

binary_bd

type
string

binary_cd

type
string

binary_ae

type
string

binary_be

type
string

binary_ce

type
string

ternary_abd

type
string

ternary_acd

type
string

ternary_bcd

type
string

ternary_abe

type
string

ternary_ace

type
string

ternary_bce

type
string

ternary_a_de

type
string

ternary_b_de

type
string

ternary_c_de

type
string

quaternary_abc_d

type
string

quaternary_abc_e

type
string

quaternary_ab_de

type
string

quaternary_ac_de

type
string

quaternary_bc_de

type
string

Wurtzite-related …wz{} groups in database{ }

Note

This section is under construction

Almost all of the groups related to materials with wurtzite symmetry contain 23 groups of identical structure, which are listed in the sections:

- *Bands groups in database{ …_zb{} } and database{ …_wz{} }*
- *Strain groups in database{ …_zb{} } and database{ …_wz{} }*
- *Low-field mobility groups in database{ …_zb{} } and database{ …_wz{} }*
- *High-field mobility groups in database{ …_zb{ } } and database{ …_wz{ } }*
- *Recombination groups in database{ …_zb{} } and database{ …_wz{} }*
- *Phonons in database{ …_zb{} } and database{ …_wz{} }*
- *Other groups in database{ …_zb{} } and database{ …_wz{} }*

The exceptions are `optical_reflectivity{}`, `optical_absorption{}` and `optical_emission{}`, which contain completly differnet groups (see *Optical groups in database{ }* for more information). All other keywords can be found below.

database{ binary_wz{} }

name
material name

type
string

example
`GaN`, `AlN`, `InN`, …

database{ ternary_wz{} }

name

type
string

binary_x
specify name of binary constituent

type
string

binary_1_x
specify name of binary constituent

type
string

database{ ternary2_wz{} }

> **Warning**
>
> Does not contain the groups `lattice_consts{}`, `mass_density{}`, `dielectric_consts{}`, `elastic_consts{}`, `piezoelectric_consts{}`, `acoustic_phonons{}`, `optical_phonons{}`, `conduction_bands{}`, `valence_bands{}`, `kp_6_bands{}`, `kp_8_bands{}`, `mobility_constant{}`, `mobility_masetti{}`, `mobility_arora{}`, `mobility_minimos{}`, `recombination{}`.

name

  type
  string

binary_x

  type
  string

binary_1_x

  type
  string

bowing_x

  type
  string

bowing_1_x

  type
  string

database{ bowing_wz{} }

name

  type
  string

database{ quaternary_wz{} }

name

  type
  string

binary1

  type
  string

binary2

  type
  string

binary3

  type
  string

ternary12

type
string

ternary13

type
string

ternary23

type
string

database{ quaternary4_wz{} }

name

type
string

binary1

type
string

binary2

type
string

binary3

type
string

binary4

type
string

ternary12

type
string

ternary23

type
string

ternary34

type
string

ternary14

type
string

database{ quinternary_wz{} }

name

type
string

binary_a

type
string

binary_b

type
string

binary_c

type
string

binary_d

type
string

ternary_ab

type
string

ternary_ac

type
string

ternary_ad

type
string

ternary_bc

type
string

ternary_bd

type
string

ternary_cd

type
string

quaternary_abc

type
string

quaternary_abd

type
string

quaternary_acd

type
string

quaternary_bcd

type
string

database{ quinternary6_wz{} }

name

type
string

binary_ad

type
string

binary_bd

type
string

binary_cd

type
string

binary_ae

type
string

binary_be

type
string

binary_ce

type
string

ternary_abd

type
string

ternary_acd

type
string

ternary_bcd

type
string

ternary_abe

type
string

ternary_ace

type
string

ternary_bce

type
string

ternary_a_de

type
string

ternary_b_de

type
string

ternary_c_de

type
string

quaternary_abc_d

type
string

quaternary_abc_e

type
string

quaternary_ab_de

type
string

quaternary_ac_de

type
string

quaternary_bc_de

type
string

Optical groups in database{ }

> **Note**
>
> This section is under construction

In this section, we describe all the groups:

Maintained Keywords

database{ optical_reflectivity{} }

(as function of wavelength in `(nm)`)

name

type
string

cutoff

value
`yes` or `no`

at{}

energy

type
double

unit
eV

wavelength

type
double

unit
nm

reflectivity

type
double

database{ optical_absorption_coeff{} }

in units of (1/cm), (as function of wavelength in `(nm)` or energy in `(eV)`)

name

type
string

cutoff

value
`yes` or `no`

default
???

at{}

energy

type
double

unit
eV

wavelength

type
double

unit
nm

absorption_coeff

type
double

decadic_absorption_coeff

type
double

database{ optical_refractive_index{} }

name

type
string

cutoff

value
`yes` or `no`

default
???

at{}

energy

type
double

unit
eV

wavelength

type
double

unit
nm

n

type
double

k

type
double

database{ illumination{} }

in units of (1/cm), (as function of wavelength in `(nm)` or energy in `(eV)`)

name

type
string

cutoff

value
`yes` or `no`

default
???

absolute_intensities

value
`yes` or `no`

default
???

at{}

energy

type
double

unit
eV

wavelength

type
double

unit
nm

intensities

type
double

Examples

```
optical_reflectivity {
    name = "Si-polished-wafer"
    cutoff = no

    at{  wavelength = 250   reflectivity = 0.672612594  }
    at{  wavelength = 260   reflectivity = 0.705174     }
    ...
    at{  wavelength = 1000  reflectivity = 0.316252445  }
}


optical_reflectivity {
    name = "Al0.80Ga0.20As"
    ...
}
```

```
optical_absorption{
    name = "Si";
    cutoff = no

    at{  wavelength = 250    absorption_coeff = 1.84E+06  }
    at{  wavelength = 260    absorption_coeff = 1.97E+06  }
    ...
    at{  wavelength = 1450   absorption_coeff = 3.20E-08  }
} : {
    name = "Silicon";
}
```

(continues on next page)

(continued from previous page)

```


optical_absorption{
    name = "Ge";
    ...
} : {
    name = "Germanium";
}


optical_absorption{
    name = "GaAs";
    ...
}


optical_absorption{
    name = "InP";
    ...
}


optical_absorption{
    name = "GaN";
    ...
}


optical_absorption{
    name = "InN";
    ...
}


optical_absorption{
    name = "In0.20Ga0.80N";

    ...
}
```

Spectra

The database file contains optional optical data such as

- standard solar spectra
- large collection of CIE illuminants and light sources
- reflectivity spectra
- absorption spectra.

If you wish to use this data, just insert the data of interest to your database file or into a *database{ }* section of your input file.

Solar spectra

The following solar spectra are already predefined and do not need to be included into database or input files.

```
# extraterrestrial solar spectrum ASTM E-490 (1366.1 W/m^2 integrated)
# added cutoff at 119.5 nm and 1000000 nm to keep integrated irradiance finite
#
#    name = "Solar-ASTME490"


# ASTM G-173-03 solar spectrum - extra terrestrial reference (airmass 0.0)
# added cutoff at 280 nm and 4000 nm to keep integrated irradiance finite
#
#    name = "Solar-ASTM-G173-ETR"


# ASTM G-173-03 solar spectrum - air mass 1.5 global tilt (1000.4 W/m^2 integrated)
# added cutoff at 280 nm and 4000 nm to keep integrated irradiance finite
#
#    name = "Solar-ASTM-G173-global"


# ASTM G-173-03 solar spectrum - air mass 1.5 direct + circumsolar (900.1 W/m^2␣
↪integrated)
# added cutoff at 280 nm and 4000 nm to keep integrated irradiance finite
#
#    name = "Solar-ASTM-G173-direct"
```

CIE luminants and light sources

The following CIE luminants and light sources are already predefined and do not need to be included into database or input files.

```
# CIE illuminant A (tungsten - 2856 K) with additional cutoff at 300 nm and 780 nm ␣
↪(irradiance NOT normalized)
#
#    name = "CIE-A"



# CIE illuminant D50 (horizon daylight - 5003 K) with additional cutoff at 300 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-D50"


# CIE illuminant D55 (mind-morning/mid-afternoon daylight - 5503 K) with additional␣
↪cutoff at 300 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-D55"


# CIE illuminant D65 (noon daylight - 6504 K) with additional cutoff at 300 nm and␣
↪830 nm  (irradiance NOT normalized)
#
#    name = "CIE-D65"

```

(continues on next page)

(continued from previous page)

```
# CIE illuminant D75 (North sky daylight - 7504 K) with additional cutoff at 300 nm␣
↪and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-D75"



# CIE flourescent FL1 (normal, daylight - 6430 K) with additional cutoff at 380 nm␣
↪and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL1"


# CIE flourescent FL2 (normal, cool white - 4230 K - most representative) with␣
↪additional cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL2"


# CIE flourescent FL3 (normal, white - 3450 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3"


# CIE flourescent FL4 (normal, warm white - 2940 K) with additional cutoff at 380 nm␣
↪and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL4"


# CIE flourescent FL5 (normal, daylight - 6350 K) with additional cutoff at 380 nm␣
↪and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL5"


# CIE flourescent FL6 (normal, light white - 5150 K) with additional cutoff at 380 nm␣
↪and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL6"


# CIE flourescent FL7 (broad band, D65 simulator - 6500 K - most representative) with␣
↪additional cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL7"


# CIE flourescent FL8 (broad band, D50 simulator, Sylvania F40 Design 50 - 5000 K)␣
↪with additional cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL8"


# CIE flourescent FL9 (broad band, cool white - 5150 K) with additional cutoff at 380␣
```

(continues on next page)

(continued from previous page)

```
↪nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL9"


# CIE flourescent FL10 (three narrow bands, Philips TL85, Ultralume 50 - 5000 K) with␣
↪additional cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL10"


# CIE flourescent FL11 (three narrow bands, Philips TL84, Ultralume 40 - 4000 K -␣
↪most representative) with additional cutoff at 380 nm and 780 nm  (irradiance NOT␣
↪normalized)
#
#    name = "CIE-FL11"


# CIE flourescent FL12 (three narrow bands, Philips TL83, Ultralume 30 - 3000 K) with␣
↪additional cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL12"


# CIE flourescent FL3.1 (standard halophosphate - 2932 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.1"


# CIE flourescent FL3.2 (standard halophosphate - 3965 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.2"


# CIE flourescent FL3.3 (standard halophosphate - 6280 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.3"


# CIE flourescent FL3.4 (DeLuxe - 2904 K) with additional cutoff at 380 nm and 780 nm␣
↪ (irradiance NOT normalized)
#
#    name = "CIE-FL3.4"


# CIE flourescent FL3.5 (DeLuxe - 4086 K) with additional cutoff at 380 nm and 780 nm␣
↪ (irradiance NOT normalized)
#
#    name = "CIE-FL3.5"


# CIE flourescent FL3.6 (DeLuxe - 4894 K) with additional cutoff at 380 nm and 780 nm␣
↪ (irradiance NOT normalized)
```

(continues on next page)

(continued from previous page)

```
#
#    name = "CIE-FL3.6"


# CIE flourescent FL3.7 (three bands - 2979 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.7"


# CIE flourescent FL3.8 (three bands - 4006 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.8"


# CIE flourescent FL3.9 (three bands - 4853 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.9"


# CIE flourescent FL3.10 (three bands - 5000 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.10"


# CIE flourescent FL3.11 (three bands - 5854 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.11"


# CIE flourescent FL3.12 (multi-band - 2984 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.12"


# CIE flourescent FL3.13 (multi-band - 3896 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.13"


# CIE flourescent FL3.14 (multi-band - 5045 K) with additional cutoff at 380 nm and␣
↪780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.14"


# CIE flourescent FL3.15 (D65 simulator JIS Z 8716:1991 - 6509 K) with additional␣
↪cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-FL3.15"
```

(continues on next page)

(continued from previous page)

```
# CIE illuminant LED-B1 (phosphor-converted blue - 2733 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-B1"


# CIE illuminant LED-B2 (phosphor-converted blue - 2998 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-B2"


# CIE illuminant LED-B3 (phosphor-converted blue - 4103 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-B3"


# CIE illuminant LED-B4 (phosphor-converted blue - 5109 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-B4"


# CIE illuminant LED-B5 (phosphor-converted blue - 6598 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-B5"


# CIE illuminant LED-BH1 (red and phosphor-converted blue mixed - 2851 K) with␣
↪additional cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-BH1"


# CIE illuminant LED-RGB1 (red, green, and blue mixed - 2840 K) with additional␣
↪cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-RGB1"


# CIE illuminant LED-V1 (phosphor-converted violet - 2724 K) with additional cutoff␣
↪at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-V1"


# CIE illuminant LED-V2 (phosphor-converted violet - 4070 K) with additional cutoff␣
↪at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-LED-V2"
```

(continues on next page)

(continued from previous page)

```


# CIE recommended indoor illuminant ID50 (5096 K) with additional cutoff at 300 nm␣
↪and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-ID50"


# CIE recommended indoor illuminant ID65 (6596 K) with additional cutoff at 300 nm␣
↪and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-ID65"



# CIE high pressure discharge lamp HP1 (sodium - 1959 K) with additional cutoff at␣
↪380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-HP1"


# CIE high pressure discharge lamp HP2 (color-enhanced sodium - 2506 K) with␣
↪additional cutoff at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-HP2"


# CIE high pressure discharge lamp HP3 (metal halide - 3144 K) with additional cutoff␣
↪at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-HP3"


# CIE high pressure discharge lamp HP4 (metal halide - 4002 K) with additional cutoff␣
↪at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-HP4"


# CIE high pressure discharge lamp HP5 (metal halide - 4039 K) with additional cutoff␣
↪at 380 nm and 780 nm  (irradiance NOT normalized)
#
#    name = "CIE-HP5"
```

Light sources

The following light sources are predefined (from: R. W. G. Hunt et al., Measuring Color, Wiley 2011), and do not need to be included into database or input files.

```
# low pressure sodium lamp (MB - 1726 K) with additional cutoff at 380 nm and 780 nm ␣
↪(irradiance NOT normalized)
#
#    name = "Lamp-SOX"


# high pressure mercury lamp (MB - 5592 K) with additional cutoff at 380 nm and 780␣
```

(continues on next page)

(continued from previous page)

```
↪nm  (irradiance NOT normalized)
#
#    name = "Lamp-MB"


# high pressure mercury lamp (MBF - 3538 K) with additional cutoff at 380 nm and 780␣
↪nm  (irradiance NOT normalized)
#
#    name = "Lamp-MBF"


# high pressure mercury lamp (MBTF - 3652 K) with additional cutoff at 380 nm and 780␣
↪nm  (irradiance NOT normalized)
#
#    name = "Lamp-MBTF"


# high pressure mercury lamp (HMI - 5988 K) with additional cutoff at 380 nm and 780␣
↪nm  (irradiance NOT normalized)
#
#    name = "Lamp-HMI"


# Xenon lamp (6044 K) with additional cutoff at 380 nm and 780 nm  (irradiance NOT␣
↪normalized)
#
#    name = "Lamp-Xenon"
```

7.16.2 Nested groups in database{ …_zb{} } and database{ …_wz{} }

Bands groups in database{ …_zb{} } and database{ …_wz{} }

- *Bands for zincblende in database{ }*
  - *database{ …{ conduction_bands{} } } for zincblende*
  - *database{ …{ valence_bands{} } } for zincblende*
- *database{ …{ kp_6_bands{} } } for zincblende*
- *database{ …{ kp_8_bands{} } } for zincblende*
- *Bands for Wurtzite in database{ }*
  - *database{ …{ conduction_bands{} } } for wurtzite*
  - *database{ …{ valence_bands{} } } for wurtzite*
- *database{ …{ kp_6_bands{} } } for wurtzite*
- *database{ …{ kp_8_bands{} } } for wurtzite*

There are about 23 identical groups available directly under all zincblende- and wurtzite-related groups. In this section we describe four of them, specifically all groups related to band paramters:

- `conduction_bands{}`
- `valence_bands{}`
- `kp_6_bands{}`

- `kp_8_bands{}`

Bands for zincblende in database{ }

database{ …{ conduction_bands{} } } for zincblende

Gamma{}

material parameters for the conduction band valley at the Gamma point of the Brillouin zone:

mass

electron effective mass (isotropic, parabolic)

value

double

unit

m_0

This mass is used for the single-band Schrödinger equation and for the calculation of the densities.

bandgap

band gap energy at 0 K

value

double

unit

eV

bandgap_alpha

Varshni parameter α for temperature dependent band gap

value

double

unit

eV/K

bandgap_beta

Varshni parameter β for temperature dependent band gap

value

double

unit

K

defpot_absolute

absolute deformation potential of the Gamma conduction band: $a_{c,\Gamma} = a_v + a_\Gamma$

value

double

unit

eV

g

g-factor (for Zeeman splitting in magnetic fields)

value

double

L{}

Material parameters for the conduction band valley at the L point of the Brillouin zone

mass_l

longitudinal electron effective mass (parabolic)

value
double

unit
m_0

mass_t
transversal electron effective mass (parabolic)

value
double

unit
m_0

These masses are used for the single-band Schrödinger equation and for the calculation of the densities.

bandgap
band gap energy at 0 K

value
double

unit
eV

bandgab_alpha
Varshni parameter α for temperature dependent band gap

value
double

unit
eV/K

bandgab_beta
Varshni parameter β for temperature dependent band gap

value
double

unit
K

defpot_absolute
absolute deformation potential of the L conduction band: $a_{c, L} = a_v + a_{gap, L}$

value
double

unit
eV

defpot_uniaxial
uniaxial deformation potential of the L conduction band

value
double

unit
eV

g_l
longitudinal g factor (for Zeeman splitting in magnetic fields)

value
double

g_t
transversal g factor (for Zeeman splitting in magnetic fields)

value
double

X{}
material parameters for the conduction band valley at the X point of the Brillouin zone. The options are the same as for `L{}`

Note

In Si, Ge and GaP we have a Delta valley instead of the X conduction band valley.

Delta{}
material parameters for the conduction band valley at the X point of the Brillouin zone. The options are the same as `L{}`, however `Delta{}` has an extra paramter `position`:

position

value
double

Note

At present, the value for `position` does not enter into any of the equations.

database{ …{ valence_bands{} } } for zincblende

material parameters for the valence band valley at the Gamma point of the Brillouin zone

bandoffset
average valence band energy $E_{v,av} = (E_{hh} + E_{lh} + E_{so})/3$

value
double

unit
eV

HH{}

mass
heavy hole effective mass (isotropic, parabolic!)

value
double

unit
m_0

g
g factor (for Zeeman splitting in magnetic fields)

value
double

LH{}

mass
light hole effective mass (isotropic, parabolic!)

value
double

unit
m$_0$

g
g factor (for Zeeman splitting in magnetic fields)

value
double

SO{}

mass
split-off hole effective mass (isotropic, parabolic!)

value
double

unit
m$_0$

g
g factor (for Zeeman splitting in magnetic fields)

value
double

defpot_absolute
absolute deformation potential of the valence bands (average of the three valence bands: a_v)

value
double

unit
eV

defpot_uniaxial_b
uniaxial shear deformation potential b of the valence bands

value
double

unit
eV

defpot_uniaxial_d
uniaxial shear deformation potential d of the valence bands

value
double

unit
eV

delta_SO
spin-orbit split-off energy Δ_{so}

value
double

unit
eV

database{ …{ kp_6_bands{} } } for zincblende

gamma1
Luttinger parameter γ_1

value
double

gamma2
Luttinger parameter γ_2

value
double

gamma3
Luttinger parameter γ_3

value
double

Note

The user can either specify the Luttinger parameters (γ_1, γ_2, γ_3) or the Dresselhaus parameters (L, M, N) parameters

L
Dresselhaus parameter L

value
double

unit
$\hbar^2/(2m_0)$

M
Dresselhaus parameter M

value
double

unit
$\hbar^2/(2m_0)$

N
Dresselhaus parameter N

value
double

unit
$\hbar^2/(2m_0)$

Warning

There are different definitions of the L and M parameters available in the literature. Definition used in *nextnano++*:

$$\mathrm{L} = (-\gamma_1 - 4\gamma_2 - 1) \cdot \left[\frac{\hbar^2}{2\mathrm{m}_0}\right]$$

$$\mathrm{M} = (2\gamma_2 - \gamma_1 - 1) \cdot \left[\frac{\hbar^2}{2\mathrm{m}_0}\right]$$

database{ …{ kp_8_bands{} } } for zincblende

S
electron effective mass parameter S for 8-band k.p. The S parameter (S = 1 + 2F) is also defined in the literature as F, where F = (S - 1)/2, e.g. I. Vurgaftman et al., JAP **89**, 5815 (2001).

value
double

Note

The S parameter (S = 1 + 2F) is also defined in the literature as F where F = (S - 1)/2, e.g. I. Vurgaftman et al., JAP **89**, 5815 (2001).

E_p

Kane's momentum matrix element. The momentum matrix element parameter P is related to E_p: $P^2 = \hbar^2/(2m_0) \cdot E_p$

value
double

unit
eV

B

bulk inversion symmetry parameter (B=0 for diamond-type materials)

value
double

unit
$\hbar^2/(2m_0)$

gamma1

Luttinger parameter γ_1'

value
double

gamma2

Luttinger parameter γ_2'

value
double

gamma3

Luttinger parameter γ_3'

value
double

Note

The user can either specify the modified Luttinger parameters (γ_1', γ_2', γ_3') or the L', M' = M, N' parameters.

L

Dresselhaus parameter L'

value
double

unit
$\hbar^2/(2m_0)$

M

Dresselhaus parameter M'

value
double

unit
$\hbar^2/(2m_0)$

N
Dresselhaus parameter N’

value
double

unit
$\hbar^2/(2m_0)$

Bands for Wurtzite in database{ }

database{ …{ conduction_bands{} } } for wurtzite

Gamma{}
material parameters for the conduction band valley at the Gamma point of the Brillouin zone:

mass_t
electron effective mass perpendicular to hexagonal c axis (parabolic)

value
double

unit
m_0

mass_l
electron effective mass along hexagonal c axis (parabolic)

value
double

unit
m_0

This mass is used for the single-band Schrödinger equation and for the calculation of the densities.

bandgap
band gap energy at 0 K

value
double

unit
eV

bandgap_alpha
Varshni parameter α for temperature dependent band gap

value
double

unit
eV/K

bandgap_beta
Varshni parameter β for temperature dependent band gap

value
double

unit
K

defpot_absolute_t
absolute deformation potential of the Gamma conduction band perpendicular to hexagonal c axis $a_{c,a} = a_2$

value
double

unit
eV

defpot_absolute_l
absolute deformation potential of the Gamma conduction band perpendicular along hexagonal c axis $a_{c,c}$ = a_1

value
double

unit
eV

> **Note**
>
> Note that I. Vurgaftman et al., JAP **94**, 3675 (2003) lists a_1 and a_2 parameters. They refer to the interband deformation potentials, i.e. to the deformation of the band gaps. Thus, we have to add the deformation potentials of the valence bands to get the deformation potentials for the conduction band edge.
>
> $$\mathrm{a_{c,a} = a_2 = a_{2,Vurgaftman} + D2}$$
>
> $$\mathrm{a_{c,c} = a_1 = a_{1,Vurgaftman} + D1}$$

g_t (optional)
g factor perpendicular to hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

g_l (optical)
g factor along hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

database{ …{ valence_bands{} } } for wurtzite

material parameters for the valence band valley at the Gamma point of the Brillouin zone

bandoffset

value
double

unit
eV

average energy of the three valence band edges (S.L. Chuang, C.S. Chang, "**k** · **p** method for strained wurtzite semiconductors", Phys. Rev. B 54 (**4**), 2491 (1996)):

$$\mathrm{E_{v,av} = (E_{hh} + E_{lh} + E_{ch})/3 - 2/3 \cdot Delta_{cr}}$$

The valence band energies for heavy hole (HH), light hole (LH) and crystal-field split-hole (CH) are calculated by defining an "average" valence band energy E_v (=$E_{v,av}$) for all three bands and adding the spin-orbit-splitting and crystal-field splitting energies afterwards. The "average" valence band energy E_v (=$E_{v,av}$) is defined on an absolute energy scale and must take into accout the valence band offsets which are "averaged" over the three holes.

Note

This energy determines the valence band offset (VBO) between two materials:

$$\mathrm{VBO}_{\mathrm{v,av}} = \mathrm{bandoffset}_{\mathrm{material1}} - \mathrm{bandoffset}_{\mathrm{material2}}$$

HH{}

mass_t
heavy hole effective mass perpendicular to hexagonal c axis (parabolic !)

value
double

unit
m_0

mass_l
heavy hole effective mass along hexagonal c axis (parabolic !)

value
double

unit
m_0

g_t (optional)
g factor perpendicular to hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

g_l (optional)
g factor along hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

LH{}

mass_t
light hole effective mass perpendicular to hexagonal c axis (parabolic !)

value
double

unit
m_0

mass_l
light hole effective mass along hexagonal c axis (parabolic !)

value
double

unit
m_0

g_t (optional)
g factor perpendicular to hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

g_l (optional)
g factor along hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

SO{}

mass_t
crystal-field split-off hole effective mass perpendicular to hexagonal c axis (parabolic !)

value
double

unit
m_0

This mass is used for the single-band Schrödinger equation and for the calculation of the densities.

mass_l
crystal-field split-off hole effective mass along hexagonal c axis (parabolic !)

value
double

unit
m_0

This mass is used for the single-band Schrödinger equation and for the calculation of the densities.

g_t (optional)
g factor perpendicular to hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

g_l (optional)
g factor along hexagonal c axis (for Zeeman splitting in magnetic fields)

value
double

defpotentials
deformation potential of the valence bands: [D1, D2, D3, D4, D5, D6]

value
vector of 6 real numbers

units
eV

example
`[-3.7, 4.5, 8.2, -4.1, -4.0, -5.5]` (for GaN)

delta
crystal-field splitting energy $\mathrm{Delta}_{\mathrm{cr}}$ = Delta_1, spin-orbit splitting energy parameter Delta_2, spin-orbit splitting energy parameter Delta_3: [Delta_1, Delta_2, Delta_3]

value
vector of 3 real numbers

units
eV

example
`[0.010, 0.00567, 0.00567]` (for GaN)

Very often one assumes Delta_2 = Delta_3 = 1/3 $\mathrm{Delta}_{\mathrm{so}}$.

database{ …{ kp_6_bands{} } } for wurtzite

A1

6-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A1 (Rashba-Sheka-Pikus parameter)

value

double

A2

6-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A2 (Rashba-Sheka-Pikus parameter)

value

double

A3

6-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A3 (Rashba-Sheka-Pikus parameter)

value

double

A4

6-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A4 (Rashba-Sheka-Pikus parameter)

value

double

A5

6-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A5 (Rashba-Sheka-Pikus parameter)

value

double

A6

6-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A6 (Rashba-Sheka-Pikus parameter)

value

double

database{ …{ kp_8_bands{} } } for wurtzite

S1

electron effective mass parameter $\mathrm{S}_1 = \mathrm{S}_{\mathrm{parallel}}$ for 8-band $\mathbf{k} \cdot \mathbf{p}$

value

double

S2

electron effective mass parameter $\mathrm{S}_2 = \mathrm{S}_{\mathrm{perpendicular}}$ for 8-band $\mathbf{k} \cdot \mathbf{p}$

value

double

E_P1

Kane's momentum matrix elements $\mathrm{E}_{\mathrm{p1}} = \mathrm{E}_{\mathrm{p,\ parallel}}$

value

double

E_P2

Kane's momentum matrix elements $\mathrm{E}_{\mathrm{p2}} = \mathrm{E}_{\mathrm{p,perpendicular}}$

value

double

Note

The momentum matrix element parameter P is related to E_p : $P^2 = \frac{\hbar^2}{2m_0} E_p$

B1

bulk inversion symmetry parameter B1

value

double

B2

bulk inversion symmetry parameters B2

value

double

B3

bulk inversion symmetry parameters B3

value

double

A1

8-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A1' (Rashba-Sheka-Pikus parameter)

value

double

A2

8-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A2' (Rashba-Sheka-Pikus parameter)

value

double

A3

8-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A3' (Rashba-Sheka-Pikus parameter)

value

double

A4

8-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A4' (Rashba-Sheka-Pikus parameter)

value

double

A5

8-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A5' (Rashba-Sheka-Pikus parameter)

value

double

A6

8-band $\mathbf{k} \cdot \mathbf{p}$ hole effective mass parameter A6' (Rashba-Sheka-Pikus parameter)

value

double

Strain groups in database{ …_zb{} } and database{ …_wz{} }

Note

This section is under construction

There are about 23 identical groups available directly under all zincblende- and wurtzite-related groups. In this section we describe four of them, specifically all groups related to strain parameters:

- `lattice_consts{}`
- `elastic_consts{}`
- `piezoelectric_consts{}`
- `pyroelectric_consts{}` (only wurtzite)

Strain for zincblende

database{ …{ lattice_consts{} } } for zincblende

a

type
double

unit
Angstrom

Specify lattice constant at 300K. In a cubic crystal system (like diamond and zincblende), the lattice constants in all three crystal axes are equal.

a_expansion

type
double

unit
Angstom/K

The lattice constants are temperature dependent. The lattice constant `a` in the database should be given for 300 K. For all other temperatures, the lattice constant is calculated by the following formula:

$$\mathrm{a(T) = a_{300K} + a\_expansion \cdot (T - 300K)}$$

where T is the temperatue in units of K.

database{ …{ elastic_consts{} } } for zincblade

Specify elastic constants:

c11

type
double

unit
GPa

c12

type
double

unit
GPa

c44

type
double

unit
GPa

database{ …{ piezoelectric_consts{} } } for zincblade

Specify piezoelectric constants (If strain is present, then generally piezoelectric charges and thus piezoelectric fields arise):

e14

type
double

unit
C/m^2

B114 (optional)
2$^{\text{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B124 (optional)
2$^{\text{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B156 (optional)
2$^{\text{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

Note

For silicon and germanium there is no piezoelectric effect at all, thus the constants are zero in this case.

Strain for wurtzite

database{ …{ lattice_consts{} } } for wurtzite

a
Lattice constant at 300 K (perpendicular to hexagonal c axis). In a hexagonal crystal system, the two lattice constants perpendicular to the hexagonal c axis are equal.

type
double

unit
Angstrom

c
Lattice constant at 300 K (along hexagonal c axis)

type
double

unit
Angstrom

a_expansion

type
double

unit
Angstrom/K

c_expansion

type
double

unit
Angstrom/K

The formaula for the temperature dependency of the lattice constants `a` and `c` in wurtzite is the same as for `a` in zincblende.

database{ …{ elastic_consts{} } } for wurtzite

Specify elastic constants:

c11

type
double

unit
GPa

c12

type
double

unit
GPa

c13

type
double

unit
GPa

c33

type
double

unit
GPa

c44

type
double

unit
GPa

database{ …{ piezoelectric_consts{} } } for wurtzite

Specify piezoelectric constants (If strain is present, then generally piezoelectric charges and thus piezoelectric fields arise):

e31

type
double

unit
C/m^2

e33

type
double

unit
C/m^2

e15

type
double

unit
C/m^2

B311 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B312 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B313 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B333 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B115 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B125 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B135 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

B344 (optional)
2$^{\mathrm{nd}}$ order piezoelectric constant

type
double

unit
C/m^2

database{ …{ pyroelectric_consts{} } } for wurtzite

Specify pyroelectric constants (for spontaneous polarization).

p1

type
double

unit
C/m^2

The pyroelectric field is directed along the hexagonal c axis ([0 0 0 1] direction).

Low-field mobility groups in database{ …_zb{} } and database{ …_wz{} }

There are about 23 identical groups available directly under all zincblende- and wurtzite-related groups. In this section we describe four of them, specifically all groups related to low-field mobility models:

- *database{ …{ mobility_constant{} } }*
- *database{ …{ mobility_masetti{} } }*
- *database{ …{ mobility_arora{} } }*
- *database{ …{ mobility_minimos{} } }*

database{ …{ mobility_constant{} } }

The constant mobility model is due to lattice scattering (phonon scattering) and leads to a constant mobility that depends only on the temperature. See *Low-field mobility models* for details on models.

electrons{} ***(optional)***

mumax ***(optional)***
bulk phonon mobility for electrons (μ^n_{max})

type
double

unit
cm^2 V^{-1} s^{-1}

exponent ***(optional)***
temperature dependence exponent for electrons

type
double

unit
None

holes{} ***(optional)***

mumax ***(optional)***
bulk phonon mobility for holes (μ^p_{max})

type
double

unit
cm^2 V^{-1} s^{-1}

exponent ***(optional)***
temperature dependence exponent for holes

type
double

unit
None

database{ . . . { mobility_masetti{} } }

See *Low-field mobility models* for details on this model.

electrons{} ***(optional)***

mumax ***(optional)***
bulk phonon mobility (μ^n_{max})

type
double

unit
cm^2 V^{-1} s^{-1}

exponent ***(optional)***
temperature dependence exponent

type
double

unit
None

mumin1 ***(optional)***
reference mobility parameter (μ^n_{min1})

type
double

unit
cm^2 V^{-1} s^{-1}

mumin2 ***(optional)***
reference mobility parameter (μ^n_{min2})

type
double

unit
cm^2 V^{-1} s^{-1}

mu1 ***(optional)***
reference mobility parameter (μ_1^n)

type
double

unit
cm^2 V^{-1} s^{-1}

pc ***(optional)***
reference doping concentration parameter (P_c^n)

type
double

unit
None

cr ***(optional)***
reference doping concentration parameter (C_r^n)

type
double

unit
None

cs ***(optional)***
reference doping concentration parameter (C_s^n)

type
double

unit
None

alpha ***(optional)***
reference doping concentration parameter (α^n)

type
double

unit
None

beta ***(optional)***
reference doping concentration parameter (β^n)

type
double

unit
None

holes{} ***(optional)***

mumax ***(optional)***
bulk phonon mobility (μ_{max}^p)

type
double

unit
cm^2 V^{-1} s^{-1}

exponent ***(optional)***
temperature dependence exponent

type
double

unit
None

mumin1 ***(optional)***
reference mobility parameter (μ^p_{min1})

type
double

unit
$cm^2 V^{-1} s^{-1}$

mumin2 ***(optional)***
reference mobility parameter (μ^p_{min2})

type
double

unit
$cm^2 V^{-1} s^{-1}$

mu1 ***(optional)***
reference mobility parameter (μ^p_1)

type
double

unit
$cm^2 V^{-1} s^{-1}$

pc ***(optional)***
reference doping concentration parameter (P^p_c)

type
double

unit
None

cr ***(optional)***
reference doping concentration parameter (C^p_r)

type
double

unit
None

cs ***(optional)***
reference doping concentration parameter (C^p_s)

type
double

unit
None

alpha ***(optional)***
reference doping concentration parameter (α^p)

type
double

unit
None

beta ***(optional)***
reference doping concentration parameter (β^p)

type
double

unit
None

database{ …{ mobility_arora{} } }

See *Low-field mobility models* for details on this model.

electrons{} ***(optional)***

mumin ***(optional)***
reference mobility parameter (μ^n_{min})

type
double

unit
cm^2 V^{-1} s^{-1}

alm ***(Optional)***
reference mobility exponent (α^n_m)

type
double

unit
None

mud ***(Optional)***
reference mobility parameter (μ^n_d)

type
double

unit
cm^2 V^{-1} s^{-1}

ald ***(Optional)***
reference mobility exponent (α^n_d)

type
double

unit
None

n0 ***(Optional)***
reference impurity parameter (N^n_0)

type
double

unit
cm^{-3}

aln ***(Optional)***
reference impurity exponent (α^n_n)

type
double

unit
None

a ***(Optional)***
reference exponent (A_a^n)

type
double

unit
None

ala ***(Optional)***
reference exponent (α_a^n)

type
double

unit
None

holes{} ***(Optional)***

mumin
reference mobility parameter (μ_{min}^p)

type
double

unit
cm^2 V^{-1} s^{-1}

alm ***(Optional)***
reference mobility exponent (α_m^p)

type
double

unit
None

mud ***(Optional)***
reference mobility parameter (μ_d^p)

type
double

unit
cm^2 V^{-1} s^{-1}

ald ***(Optional)***
reference mobility exponent (α_d^p)

type
double

unit
None

n0 ***(Optional)***
reference impurity parameter (N_0^p)

type
double

unit
cm^{-3}

aln ***(Optional)***
reference impurity exponent (α_n^p)

type
double

unit
None

a ***(Optional)***
reference exponent (A_a^p)

type
double

unit
None

ala ***(Optional)***
reference exponent (α_a^p)

type
double

unit
None

database{ …{ mobility_minimos{} } }

See *Low-field mobility models* for details on this model.

electrons{} ***(Optional)***

muL300 ***(Optional)***
bulk phonon mobility for electrons (same as *database{ …{ mobility_constant{} } }*)

type
double

unit
cm^2 V^{-1} s^{-1}

muLexpT ***(Optional)***
temperature dependence exponent (same as *database{ …{ mobility_constant{} } }* apart from the sign)

type
double

unit
None

muLImin300 ***(Optional)***
reference mobility parameter

type
double

unit
cm^2 V^{-1} s^{-1}

muLIexpTabove ***(Optional)***
reference mobility exponent

type
double

unit
None

muLIexpTbelow ***(Optional)***
reference mobility exponent

type
double

unit
None

TSwitch ***(Optional)***
Switch between equations (2.10.1.5) and (2.10.1.6) at this temperature

type
double

unit
K

default
200

Cref300 ***(Optional)***
reference impurity parameter

type
double

unit
cm^{-3}

CrefexpT ***(Optional)***
reference impurity exponent

type
double

unit
None

alpha300 ***(Optional)***
reference exponent parameter

type
double

unit
None

alphaexpT ***(Optional)***
reference exponent

type
double

unit
None

holes{} ***(optional)***

muL300 ***(Optional)***
bulk phonon mobility for electrons (same as *database{ …{ mobility_constant{} } }*)

type
double

unit
cm^2 V^{-1} s^{-1}

muLexpT ***(Optional)***
temperature dependence exponent (same as *database{ …{ mobility_constant{} } }* apart from the sign)

type
double

unit
None

muLImin300 ***(Optional)***
reference mobility parameter

type
double

unit
cm^2 V^{-1} s^{-1}

muLIexpTabove ***(Optional)***
reference mobility exponent

type
double

unit
None

muLIexpTbelow ***(Optional)***
reference mobility exponent

type
double

unit
None

TSwitch ***(Optional)***
switch between equations (2.10.1.5) and (2.10.1.6) at this temperature

type
double

unit
K

Cref300 ***(Optional)***
reference impurity parameter

type
double

unit
cm^{-3}

CrefexpT ***(Optional)***
reference impurity exponent

type
double

unit
None

alpha300 ***(Optional)***
reference exponent parameter

type
double

unit
None

alphaexpT *(Optional)*
reference exponent exponent

type
double

unit
None

High-field mobility groups in database{ . . . _zb{ } } and database{ . . . _wz{ } }

There are about 23 identical groups available directly under all zincblende- and wurtzite-related groups. In this section we describe four of them, specifically all groups related to high-field mobility models:

Nested keywords

- *mobility_haensch{ }*
- *mobility_haensch{ electrons{ } }*
- *mobility_haensch{ electrons{ vsat } }*
- *mobility_haensch{ holes{ } }*
- *mobility_haensch{ holes{ vsat } }*
- *mobility_canali{ }*
- *mobility_canali{ electrons{ } }*
- *mobility_canali{ electrons{ vsat } }*
- *mobility_canali{ electrons{ alpha } }*
- *mobility_canali{ electrons{ beta } }*
- *mobility_canali{ holes{ } }*
- *mobility_canali{ holes{ vsat } }*
- *mobility_canali{ holes{ alpha } }*
- *mobility_canali{ holes{ beta } }*
- *mobility_transferred{ }*
- *mobility_transferred{ electrons{ } }*
- *mobility_transferred{ electrons{ vsat } }*
- *mobility_transferred{ electrons{ alpha } }*
- *mobility_transferred{ electrons{ beta } }*
- *mobility_transferred{ electrons{ gamma } }*
- *mobility_transferred{ electrons{ E0 } }*
- *mobility_transferred{ holes{ } }*
- *mobility_transferred{ holes{ vsat } }*
- *mobility_transferred{ holes{ alpha } }*
- *mobility_transferred{ holes{ beta } }*
- *mobility_transferred{ holes{ gamma } }*
- *mobility_transferred{ holes{ E0 } }*

- *mobility_eastman{ }*
- *mobility_eastman{ electrons{ } }*
- *mobility_eastman{ electrons{ vsat } }*
- *mobility_eastman{ electrons{ alpha } }*
- *mobility_eastman{ electrons{ beta } }*
- *mobility_eastman{ holes{ } }*
- *mobility_eastman{ holes{ vsat } }*
- *mobility_eastman{ holes{ alpha } }*
- *mobility_eastman{ holes{ beta } }*
- *mobility_eastman4{ }*
- *mobility_eastman4{ electrons{ } }*
- *mobility_eastman4{ electrons{ vsat } }*
- *mobility_eastman4{ electrons{ v_mid } }*
- *mobility_eastman4{ electrons{ v_peak } }*
- *mobility_eastman4{ electrons{ E_mid } }*
- *mobility_eastman4{ electrons{ E_peak } }*
- *mobility_eastman4{ holes{ } }*
- *mobility_eastman4{ holes{ vsat } }*
- *mobility_eastman4{ holes{ v_mid } }*
- *mobility_eastman4{ holes{ v_peak } }*
- *mobility_eastman4{ holes{ E_mid } }*
- *mobility_eastman4{ holes{ E_peak } }*

---

mobility_haensch{ }

Calling sequence

`database{ *_zb{ mobility_haensch{ } } }` `database{ *_wz{ mobility_haensch{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Stores parameters for the *Hänsh model*.

---

mobility_haensch{ electrons{ } }

Calling sequence

`database{ *_zb{ mobility_haensch{ electrons{ } } }` `database{ *_wz{ mobility_haensch{ electrons{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *Hänsh model* for electrons.

---

mobility_haensch{ electrons{ vsat } }

Calling sequence

`database{ *_zb{ mobility_haensch{ electrons{ vsat } } }` `database{ *_wz{ mobility_haensch{ electrons{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *Hänsh model* for electrons.

---

mobility_haensch{ holes{ } }

Calling sequence

`database{ *_zb{ mobility_haensch{ holes{ } } }` `database{ *_wz{ mobility_haensch{ holes{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *Hänsh model* for holes.

---

mobility_haensch{ holes{ vsat } }

Calling sequence

`database{ *_zb{ mobility_haensch{ holes{ vsat } } }` `database{ *_wz{ mobility_haensch{ holes{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *Hänsh model* for holes.

---

mobility_canali{ }

Calling sequence

`database{ *_zb{ mobility_canali{ } } }` `database{ *_wz{ mobility_canali{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Stores parameters for the *extended Canali model*.

---

mobility_canali{ electrons{ } }

Calling sequence

`database{ *_zb{ mobility_canali{ electrons{ } } }` `database{ *_wz{ mobility_canali{ electrons{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *extended Canali model* for electrons.

---

mobility_canali{ electrons{ vsat } }

Calling sequence

`database{ *_zb{ mobility_canali{ electrons{ vsat } } }` `database{ *_wz{ mobility_canali{ electrons{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *extended Canali model* for electrons.

---

mobility_canali{ electrons{ alpha } }

Calling sequence

`database{ *_zb{ mobility_canali{ electrons{ alpha } } }` `database{ *_wz{ mobility_canali{ electrons{ alpha } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $-$
- **values:** `[0.0, ...)`

Functionality

Parameter α for the *extended Canali model* for electrons.

---

mobility_canali{ electrons{ beta } }

Calling sequence

`database{ *_zb{ mobility_canali{ electrons{ beta } } }` `database{ *_wz{ mobility_canali{ electrons{ beta } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $-$
- **values:** `[1e-3, ...)`

Functionality

Parameter β for the *extended Canali model* for electrons.

Note

One should set `alpha = 0` if aiming at using the extended Canali model as in references. When `alpha = 1` and `beta = 2` then Hänsch model is obtained as a special case of implemented formula.

---

mobility_canali{ holes{ } }

Calling sequence

`database{ *_zb{ mobility_canali{ holes{ } } }` `database{ *_wz{ mobility_canali{ holes{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *extended Canali model* for holes.

---

mobility_canali{ holes{ vsat } }

Calling sequence

`database{ *_zb{ mobility_canali{ holes{ vsat } } }` `database{ *_wz{ mobility_canali{ holes{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *extended Canali model* for holes.

---

mobility_canali{ holes{ alpha } }

Calling sequence

`database{ *_zb{ mobility_canali{ holes{ alpha } } }` `database{ *_wz{ mobility_canali{ holes{ alpha } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $-$
- **values:** `[0.0, ...)`

Functionality

Parameter α for the *extended Canali model* for holes.

---

mobility_canali{ holes{ beta } }

Calling sequence

`database{ *_zb{ mobility_canali{ holes{ beta } } }` `database{ *_wz{ mobility_canali{ holes{ beta } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[1e-3, ...)`

Functionality

Parameter β for the *extended Canali model* for holes.

> **Note**
>
> One should set `alpha = 0` if aiming at using the extended Canali model as in references. When `alpha = 1` and `beta = 2` then Hänsch model is obtained as a special case of implemented formula.

mobility_transferred{ }

Calling sequence

`database{ *_zb{ mobility_transferred{ } } }` `database{ *_wz{ mobility_transferred{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Stores parameters for the *transferred electron model*.

mobility_transferred{ electrons{ } }

Calling sequence

`database{ *_zb{ mobility_transferred{ electrons{ } } }` `database{ *_wz{ mobility_transferred{ electrons{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ vsat } }

Calling sequence

`database{ *_zb{ mobility_transferred{ electrons{ vsat } } }` `database{ *_wz{ mobility_transferred{ electrons{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ alpha } }

Calling sequence

`database{ *_zb{ mobility_transferred{ electrons{ alpha } } }` `database{ *_wz{ mobility_transferred{ electrons{ alpha } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $-$
- **values:** `[1e-3, ...)`
- **default:** $r = 1.0$

Functionality

Parameter α for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ beta } }

Calling sequence

`database{ *_zb{ mobility_transferred{ electrons{ beta } } }` `database{ *_wz{ mobility_transferred{ electrons{ beta } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[1.001, ...)`

Functionality

Parameter β for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ gamma } }

Calling sequence

`database{ *_zb{ mobility_transferred{ electrons{ gamma } } }` `database{ *_wz{ mobility_transferred{ electrons{ gamma } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[0.0, ...)`
- **default:** $r = 0.0$

Functionality

Parameter γ for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ E0 } }

Calling sequence

`database{ *_zb{ mobility_transferred{ electrons{ E0 } } }` `database{ *_wz{ mobility_transferred{ electrons{ E0 } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[0.0, ...)`
- **default:** $r = 0.0$

Functionality

Parameter E_0 for the *transferred electron model* for electrons.

---

mobility_transferred{ holes{ } }

Calling sequence

`database{ *_zb{ mobility_transferred{ holes{ } } }` `database{ *_wz{ mobility_transferred{ holes{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ vsat } }

Calling sequence

`database{ *_zb{ mobility_transferred{ holes{ vsat } } }` `database{ *_wz{ mobility_transferred{ holes{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ alpha } }

Calling sequence

`database{ *_zb{ mobility_transferred{ holes{ alpha } } }` `database{ *_wz{ mobility_transferred{ holes{ alpha } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $-$
- **values:** `[1e-3, ...)`
- **default:** $r = 1.0$

Functionality

Parameter α for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ beta } }

Calling sequence

`database{ *_zb{ mobility_transferred{ holes{ beta } } }` `database{ *_wz{ mobility_transferred{ holes{ beta } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[1.001, ...)`

Functionality

Parameter β for the *transferred electron model* for holes.

mobility_transferred{ holes{ gamma } }

Calling sequence

`database{ *_zb{ mobility_transferred{ holes{ gamma } } }` `database{ *_wz{ mobility_transferred{ holes{ gamma } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[0.0, ...)`
- **default:** $r = 0.0$

Functionality

Parameter γ for the *transferred electron model* for holes.

mobility_transferred{ holes{ E0 } }

Calling sequence

`database{ *_zb{ mobility_transferred{ holes{ E0 } } }` `database{ *_wz{ mobility_transferred{ holes{ E0 } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[0.0, ...)`
- **default:** $r = 0.0$

Functionality

Parameter E_0 for the *transferred electron model* for holes.

---

mobility_eastman{ }

Calling sequence

`database{ *_zb{ mobility_eastman{ } } }` `database{ *_wz{ mobility_eastman{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Stores parameters for the *Eastman-Tiwari-Shur*.

---

mobility_eastman{ electrons{ } }

Calling sequence

`database{ *_zb{ mobility_eastman{ electrons{ } } }` `database{ *_wz{ mobility_eastman{ electrons{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ electrons{ vsat } }

Calling sequence

`database{ *_zb{ mobility_eastman{ electrons{ vsat } } }` `database{ *_wz{ mobility_eastman{ electrons{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ electrons{ alpha } }

Calling sequence

`database{ *_zb{ mobility_eastman{ electrons{ alpha } } }` `database{ *_wz{ mobility_eastman{ electrons{ alpha } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[0.0, ...)`

Functionality

Parameter α for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ electrons{ beta } }

Calling sequence

`database{ *_zb{ mobility_eastman{ electrons{ beta } } }` `database{ *_wz{ mobility_eastman{ electrons{ beta } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** –
- **values:** `[1.0, ...)`

Functionality

Parameter β for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ holes{ } }

Calling sequence

`database{ *_zb{ mobility_eastman{ holes{ } } }` `database{ *_wz{ mobility_eastman{ holes{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores parameters for the *Eastman-Tiwari-Shur* for holes.

mobility_eastman{ holes{ vsat } }

Calling sequence

`database{ *_zb{ mobility_eastman{ holes{ vsat } } }` `database{ *_wz{ mobility_eastman{ holes{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman{ holes{ alpha } }

Calling sequence

`database{ *_zb{ mobility_eastman{ holes{ alpha } } }` `database{ *_wz{ mobility_eastman{ holes{ alpha } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $-$
- **values:** `[0.0, ...)`

Functionality

Parameter α for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman{ holes{ beta } }

Calling sequence

`database{ *_zb{ mobility_eastman{ holes{ beta } } }` `database{ *_wz{ mobility_eastman{ holes{ beta } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $-$
- **values:** `[1.0, ...)`

Functionality

Parameter β for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman4{ }

Calling sequence

`database{ *_zb{ mobility_eastman4{ } } }` `database{ *_wz{ mobility_eastman4{ } } }`

Properties

- **usage:** optional
- **items:** maximum 1

Functionality

Stores alternative, observable, parameters for the *Eastman-Tiwari-Shur*.

---

mobility_eastman4{ electrons{ } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ electrons{ } } }` `database{ *_wz{ mobility_eastman4{ electrons{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores alternative, observable, parameters for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman4{ electrons{ vsat } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ electrons{ vsat } } }` `database{ *_wz{ mobility_eastman4{ electrons{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ v_mid } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ electrons{ v_mid } } }` `database{ *_wz{ mobility_eastman4{ electrons{ v_mid } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Velocity v_{mid} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ v_peak } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ electrons{ v_peak } } }` `database{ *_wz{ mobility_eastman4{ electrons{ v_peak } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Velocity v_{peak} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ E_mid } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ electrons{ E_mid } } }` `database{ *_wz{ mobility_eastman4{ electrons{ E_mid } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{V/cm}$
- **values:** `[1.0, ...)`

Functionality

Driving force E_mid for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ E_peak } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ electrons{ E_peak } } }` `database{ *_wz{ mobility_eastman4{ electrons{ E_peak } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{V/cm}$
- **values:** `[1.0, ...)`

Functionality

Driving force E_peak for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ holes{ } } }` `database{ *_wz{ mobility_eastman4{ holes{ } } }`

Properties

- **usage:** required
- **items:** maximum 1

Functionality

Stores alternative, observable, parameters for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman4{ holes{ vsat } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ holes{ vsat } } }` `database{ *_wz{ mobility_eastman4{ holes{ vsat } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ v_mid } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ holes{ v_mid } } }` `database{ *_wz{ mobility_eastman4{ holes{ v_mid } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Velocity v_{mid} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ v_peak } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ holes{ v_peak } } }` `database{ *_wz{ mobility_eastman4{ holes{ v_peak } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{cm/s}$
- **values:** `[1.0, ...)`

Functionality

Velocity v_{peak} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ E_mid } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ holes{ E_mid } } }` `database{ *_wz{ mobility_eastman4{ holes{ E_mid } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{V/cm}$
- **values:** `[1.0, ...)`

Functionality

Driving force E_{mid} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ E_peak } }

Calling sequence

`database{ *_zb{ mobility_eastman4{ holes{ E_peak } } }` `database{ *_wz{ mobility_eastman4{ holes{ E_peak } } }`

Properties

- **usage:** required
- **type:** real number
- **unit:** $\mathrm{V/cm}$
- **values:** `[1.0, ...)`

Functionality

Driving force E_{peak} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

Recombination groups in database{ ..._zb{} } and database{ ..._wz{} }

There are about 23 identical groups available directly under all zincblende- and wurtzite-related groups. In this section we describe one of them, specifically the group related to recombination models **recombination{}**.

database{ ...{ recombination{} } }

This section specifies the coefficients related to recombination processes. These are used when the current equation is solved. In *nextnano++*, the following recombination processes are included:

- *Shockley-Read-Hall (SRH) recombination*
- *Auger recombination*
- *Radiative recombination*

Example

```
binary_zb {
    name    = Si                        # material name, e.g. Si, GaAs, InP, ...

        ...

    recombination{
                                        # Shockley-Read-Hall recombination
        SRH{        tau_n  = 1.0e-9     # [s]    zero doping scattering time for␣
↪electrons
                    nref_n = 1.0e19     # [cm^-3] reference doping concentration for␣
↪electrons
                    tau_p  = 1.0e-9     # [s]    zero doping scattering time for holes
                    nref_p = 1.0e18     # [cm^-3] reference doping concentration for␣
↪holes
        }
```

(continues on next page)

(continued from previous page)

```
                                        # Auger recombination
        Auger{      c_n     = 2.8e-31   # [cm^6/s]
                    c_p     = 9.9e-31   # [cm^6/s]
        }

                                        # direct recombination
        radiative{  c = = 2.0e-10   }   # [cm^3/s]
                                        # 2.0e-10 for GaAs, 0 for Si (indirect␣
↪semiconductor)

    }

}
```

Shockley-Read-Hall (SRH) recombination

SRH model models the generation/recombination process that is assisted by impurities. The recombination/generation rates depend on the deviation of the carrier concentration from the equilibrium value and the scattering rates depend on the doping concentration. The rate is calculated using the following formulas:

$$
\begin{aligned}
R_{SRH} &= \frac{p \cdot n - n_i^2}{\tau_p(n + n_i) + \tau_n(p + p_i)}, \\
\tau_{p/n} &= \frac{\tau_{p0/n0}}{1 + \frac{N_D + N_A}{N_n/p,ref}},
\end{aligned}
$$

where τ_{n0} is zero doping scattering time for electrons, $N_{n,ref}$ is reference doping concentration for electrons, τ_{p0} is zero doping scattering time for holes, and $N_{p,ref}$ is reference doping concentration for holes.

tau_n

zero doping scattering time for electrons τ_{n0}

type

double

unit

s

nref_n

reference doping concentration for electrons $N_{n,ref}$

type

double

unit

cm^{-3}

tau_p

zero doping scattering time for holes τ_{p0}

type

double

unit

s

nref_p

reference doping concentration for holes and $N_{p,ref}$

type
double

unit
cm^{-3}

Auger recombination

More imformation on physics: Auger recombination processes in semiconductor heterostructures.

Auger process is a dominant recombination channel for devices with an extremely high carrier concentrations. It is a three-particle process, therefore, scaling with the third power of the carrier density.

The phonon-assisted Auger recombination rate, which plays an important role especially at high carrier injection, is modeled by the following equation:

$$R_{Auger} = (C_n n + C_p p) \cdot (np - n_i^2),$$

where C_n and C_p are coefficients.

c_n
coefficient C_n

type
double

unit
cm^6 s^{-1}

c_p
coefficient C_p

type
double

unit
cm^6 s^{-1}

More imformation on physics: Auger recombination processes in semiconductor heterostructures.

Radiative recombination

The simplest, and the most important for light emitting devices, process for the generation and recombination of electron-hole pairs is the direct emission or absorption spectra of a photon (radiative recombination) modelled within the formula

$$R_{radiative} = C(np - n_i^2),$$

where C is a coefficient.

c
a coefficient C

type
double

unit
cm^3 s^{-1}

example
2.0e-10 (for GaAs), 0.0 (for Si, indirect semiconductor)

c_absorption
If `c_absorption` > `c`, then `c_absorption` will be used instead of `c` as C to compute absorption coefficients in semiclassical optics. This can be used to enable and control absorption for indirect bandgap materials where `c` practically vanishes. Ideally, for these materials,

`c_absorption` should be set in the database to values which reproduce the experimentally observed absorption coefficients.

type
double

unit
cm^3 s^{-1}

default
1e-11

Phonons in database{ …_zb{} } and database{ …_wz{} }

There are about 23 identical groups available directly under all zincblende- and wurtzite-related groups. In this section we describe two of them, specifically all groups related to phonons:

- **acoustic_phonons{}**
- **optical_phonons{}**

Phonons in zincblende materials

database{ …{ acoustic_phonons{} } } for zincblade

LA_energy
longitudinal acoustic phonon energy

type
double

unit
eV

TA_energy
transverse acoustic phonon energy

type
double

unit
eV

database{ …{ optical_phonons{} } } for zincblade

LO_energy
energy of longitudinal optical phonon

type
double

unit
eV

LO_width
width of longitudinal optical phonon

type
double

unit
nm

TO_energy
energy of transverse optical phonon

type
double

unit
eV

Phonons in wurtzite materials

database{ …{ acoustic_phonons{} } } for wurtzite

LA_energy
energy of longitudinal acoustic phonon

type
double

unit
eV

TA_energy:
energy of transverse acoustic phonon

type
double

unit
eV

database{ …{ optical_phonons{} } } for wurtzite

LO_energy_l
energy of longitudinal optical phonon (along hexagonal c axis)

type
double

unit
eV

LO_energy_t
energy of longitudinal optical phonon (perpendicular to hexagonal c axis)

type
double

unit
eV

LO_width
width of longitudinal optical phonon

type
double

unit
nm

TO_energy_l
energy of transverse optical phonon (along hexagonal c axis)

type
double

unit
eV

TO_energy_t
energy of transverse optical phonon (perpendicular to hexagonal c axis)

type
double

unit
eV

Other groups in database{ …_zb{} } and database{ …_wz{} }

Note

This section is under construction

There are about 18 identical groups available directly under all zincblende- and wurtzite-related groups. In this section we describe three of them:

- `valence{}`
- `mass_density{}`
- `dielectric_consts{}`

Other groups and attributes for zincblende

database{ …{ valence } } for zincblende

A label allowing to group materials to prevent formation of unrealistic alloys. Only materials with corresponding labels can form an alloy.

value

- `IV_IV` for group IV materials (like Si, Ge, SiC, …)
- `III_V` for III-V materials (like GaAs, AlP, …)
- `II_VI` for II-VI materials (like ZnO, HgTe, …)
- `I_VII` for I-VII materials (like CuCl, …)

database{ …{ mass_density{} } } for zincblende

no information available

database{ …{ dielectric_consts{} } } for zincblende

static_a
static or low frequency ($\epsilon(\omega = 0)$) dielectric constant

type
double

optical_a
optical dielectric constant

type
double

Note

The optical dielectric constant is currently not in use but maybe it is necessary in the future for laser calculations.

The static dielectric constant enters the Poisson equation. It is also needed to calculate the optical absorption spectra and enters the equation for the exciton correction. In a cubic crystal system (like diamond and zincblende), the dielectric constants in all three crystal axes are equal.

Other groups and attributes for wurtzite

database{ . . . { valence } } for wurtzite

A label allowing to group materials to prevent formation of unrealistic alloys. Only materials with corresponding labels can form an alloy.

value

- `IV_IV` for group IV materials (like Si, Ge, SiC, . . .)
- `III_V` for III-V materials (like GaAs, AlP, . . .)
- `II_VI` for II-VI materials (like ZnO, HgTe, . . .)
- `I_VII` for I-VII materials (like CuCl, . . .)

database{ . . . { mass_density{} } } for wurtzite

no information available

database{ . . . { dielectric_consts{} } } for wurtzite

static_a

static or low frequency ($\epsilon(\omega = 0)$) dielectric constant (perpendicular to hexagonal c axis). In a hexagonal crystal system the two dielectric constants perpendicular to the hexagonal c axis are equal.

type

double

static_c

static or low frequency ($\epsilon(\omega = 0)$) dielectric constant (along hexagonal c axis)

type

double

optical_a

optical dielectric constant (perpendicular to hexagonal c axis)

type

double

optical_c

optical dielectric constant (along to hexagonal c axis)

type

double

Note

The optical dielectric constants (`optical_a`, `optical_c`) are currently not in use, but maybe they are necessary in the future for laser calculations.

The static dielectric constants enter the Poisson equation. They are also needed to calculate the optical absorption spectra and enter the equation for the exciton correction.

database{ …_zb{ kp_30_bands{} } } *(optional)*

Note

This is preliminary documentation of the implemented 30-band $\mathbf{k} \cdot \mathbf{p}$ model.

List of real parameters for 30-band $\mathbf{k} \cdot \mathbf{p}$ model:

- E1_q
- E5_d
- E3_t
- E1_u
- E5_c
- E1_c
- E1_w
- P_0
- P_1
- P_2
- P_3
- P_4
- P_5
- P_prime_0
- P_prime_1
- Q_0
- Q_1
- R_0
- R_1
- delta_5v
- delta_5c
- delta_5d
- delta_5v5c
- delta_5v5d

Documentation for the database in nn3 is available here (old documentation layout).

CHAPTER EIGHT

INPUT SYNTAX

Note

this site is under reconstruction

The syntax of input files for *nextnano++*, *nextnano.MSB* (included in *nextnano++*), and *nextnano.NEGF* has been unified. The syntax features described below are, therefore, valid for all abovementioned tools. For tool-specific elements of syntax, such as the meaning, use, and allowed combinations of various keywords, please see the respective documentations.

8.1 General

8.1.1 Case Sensitivity

Input files are always **CASE-SENSITIVE**, which means that uppercase and lowercase are distinguished in the input files.

Example

In the script

```
text
Text
TeXt
teXt
TEXT
```

there are 5 different entries.

8.1.2 White-Spaces

The input files are almost **white-space independent**.

Example 1

The two scripts

```
x=5 $y=6 z=[1,2]
```

and

```
x = 5
$y = 6
z = [1,
    2]
```

have the same effect.

Example 2

Elements of syntax

```
band{
```

and

```
band   {
```

are considered the same.

However, there are exceptions, when breaking line is not allowed.

`Example 3`

Adding a line breaks like

```
band
{
x
= 5
```

is not allowed.

8.1.3 Semicolons

For better readability, optional **semicolons** may be used to separate or terminate assignments.

Example 1

```
x=5; $y=6 ; z=[1,2];
```

However, placing semicolons at inappropriate places will result in a syntax error.

`Example 2`

Using a semicolon like

```
x = ; 5
```

is not allowed.

8.2 Variables

One can define variables and use them either to set some parameters or to evaluate other quantities for further use. Variable name always starts with a dollar sign ($) and is followed by a letter or underscore (_), and then by an arbitrary number of characters, numbers, or underscores.

Example 1

Script below contains 3 variable names

```
$a_43
$_BT
$_5c
```

which are `a_43`, `_BT`, and `_5c`.

8.2.1 Numbers and arrays

Variables can be defined to contain a number of an array of numbers. The numbers are of a double-precision floating-point format by default. If no rounding is needed then they get automatically converted to integers.

Example 1

In the script below

```
$x = 123
$y = 123.3
$z = 123.0
$zzz_ks = [12.3, 4]
```

`$x`, `$z`, and the last element of `$zzz_ks` are converted to integers. `$y` and the first element of `$zzz_ks` remain as doubles.

Variables always have a global scope. Therefore, they can be used everywhere after definition. The variables can be used for mathematical operations.

Example 2

Using variables for mathematical operation can look like

```
$y = sqrt($y)*$x
a = $zzz_ks
```

Attention

Element-wise mathematical operations between vectors or between scalars and vectors are not supported.

8.2.2 Strings

It is possible to define string variables, either by assigning a quoted string constant or unquoted string constant.

Example 1

Two string variables are defined in the script below.

```
$name = "some text"
$id = hello
```

While `$name` is defined with a quoted string constant, `$id` is defined with an unquoted string constant.

Attention

Similarly to variable names, unquoted string constants have to begin with a letter or an underscore. Also, they cannot contain white spaces. Quoted string constants does not have such limitations.

Note

While carriage returns are not allowed inside of string constants, they (and also comments) are allowed between quoted string constants to be concatenated.

Leading and trailing blanks are trimmed. Multiple string constants are automatically concatenated with blanks inserted in between them.

Example 2

Two quoted string constants

```
"aa b" "c"
```

and three unquoted string constants

```
aa b c
```

are automatically concatenated as

```
"aa b c"
```

To concatenate strings without inserted blanks, one can use + operator. All: string constants, string variables, double constants, and double variables can be concatenated with some string variable into a string.

Example 3

Concatenating multiple types of data into one string variable.

```
$id = hello
$id2 = "world"
$num = 3
$concat = $id + "_" + $id2 + $num + 5
```

As a result `$concat` contains `"hello_world35"`.

Attention

Limitation: Quoted string constants can only be added using + from the right. Therefore, in an expression like for `$concat`, the leftmost term in a concatenation (here `$id`) have to be a variable.

Double values are rounded into the nearest integer first, before being concatenated to a string variable.

Hint

Use conversion function `string()`, if no such rounding is wanted.

8.3 Comments

8.3.1 One-line comment

One-line comments can be started with `#`. They always run until the end of the line.

Example 1

Line comments can begin anywhere in the line.

```
# This is a comment line.
x = 3.0   # This is a comment, too.
```

8.3.2 Multi-line comment

Mult-line comments can be defined using text blocks `!TEXT` `!ENDTEXT`.

Example 2

```
!TEXT
almost arbitrary content can come here
!ENDTEXT
```

Attention

Nesting text blocks is not allowed.

8.4 Conditional Statements

8.4.1 Conditional lines

Conditional lines allow enabling and disabling individual lines.

Example 1

If `$x=0` then all three lines are ignored.

```
!WHEN $x    schottky{
!WHEN $x        name = air
!WHEN $x    }
```

The `$x` must be defined as a number, otherwise an error message will occur.

Note

In this example, the text is always commented out, unless `$x` is defined with value `$x != 0`.

Attention

No rounding or truncation is being performed here on `$x` so it has to be equivalent to `0.0` if defined as a double.

Warning

Conditional `#IF` and `#if` have been deprecated and are in the process of being removed. They should not be used for conditional lines.

8.4.2 Conditional blocks

Conditional blocks can be defined using `!IF`, `!ELIF`, `ELSE`, and `ENDIF`. They allow enabling and disabling entire blocks.

Note

Use of `!ELSE` and `!ELIF` is optional

Example 2

```
   !IF($x)
       name = air
       note = "Some text"
       !WHEN $y  note2 = "This is a nested conditional line."
   !ELIF($y)
       name = GaAs
   !ELIF($z)
       name = InAs
   !ELSE
       name = InGaAs
   !ENDIF

Here, variables also needs to be defined with non-zero values to be␣
↪considered ``TRUE``.
```

Attention

Nesting conditional blocks is not allowed.

8.5 Data section

A data section can be defined using `!DATA` statement. As everything below the `!DATA` statement will be ignored by the parser, it is available only at the end of the input file.

The data section can be, however, used by some simulators (currently *nextnano++*) to define and/or run post-processing scripts of generated data.

Example 1

One can write anything in the data section like it is a comment.

```
!DATA

An arbitrary text starting from here
until the end of the file.
```

However, it is not advised to use it for making comments in the input files.

8.6 Operators and functions

8.6.1 Tables for number variables

The following functions and operators (sorted with decreasing precedence) are available for the use with number variables.

functions	description
`sqrt()`	square root $\sqrt{}$
`cbrt()`	cubic root $\sqrt[3]{}$
`exp()`	exponential function $\exp(\,)$
`log()`	natural logarithm $\log$
`ln()`	natural logarithm $\ln$
`log2()`	decadic logarithm (base 2) $\log_2$
`log10()`	decadic logarithm (base 10) $\log_{10}$
`sin()`	sine $\sin(\,)$
`cos()`	cosine $\cos(\,)$
`tan()`	tangent $\tan(\,)$
`asin()`	acrsine $\sin^{-1}(\,)$
`acos()`	arccosine $\cos^{-1}(\,)$
`atan()`	arctangent $\tan^{-1}(\,)$
`sinh()`	hyperbolic sine $\sinh(\,)$
`cosh()`	hyperbolic cosine $\cosh(\,)$
`tanh()`	hyperbolic tangent $\tanh(\,)$
`asinh()`	inverse hyperbolic sine $\sinh^{-1}(\,)$
`acosh()`	inverse hyperbolic cosine $\cosh^{-1}(\,)$
`atanh()`	inverse hyperbolic tangent $\tanh^{-1}(\,)$
`erf()`	error function erf( )
`erfc()`	complementary error function erfc( )
`gamma()`	Gamma function $\Gamma(\,)$
`fdm3half()`	complete Fermi–Dirac integral $F_{-3/2}()$ of order -3/2 (includes the $1/\Gamma(-1/2)$ prefactor)
`fdmhalf()`	complete Fermi–Dirac integral $F_{-1/2}()$ of order -1/2 (includes the $1/\Gamma(1/2)$ prefactor)
`fdzero()`	complete Fermi–Dirac integral $F_0()$ of order 0 (includes the $1/\Gamma(1) = 1$ prefactor)
`fdphalf()`	complete Fermi–Dirac integral $F_{1/2}()$ of order 1/2 (includes the $1/\Gamma(3/2)$ prefactor)
`fdp3half()`	complete Fermi–Dirac integral $F_{3/2}()$ of order 3/2 (includes the $1/\Gamma(5/2)$ prefactor)
`abs()`	absolute value $\|\,\|$
`floor()`	floor function floor(x): largest integer $\leq x$
`ceil()`	ceiling function ceil(x): smallest integer $\geq x$
`round()`	rounds the number to the nearest integer
`sign()`	sign function
`heaviside()`	Heaviside step function (corresponds to `isnotnegative()`)
`ispositive()`	returns 1 if value is positive and `0` otherwise
`isnegative()`	returns 1 if value is negative and `0` otherwise
`iszero()`	returns 1 if value is zero and `0` otherwise
`isnotpositive()`	returns 1 if value is not positive and `0` otherwise
`isnotnegative()`	returns 1 if value is not negative and `0` otherwise (corresponds to `heaviside()`)
`isnotzero()`	returns 1 if value is not zero and `0` otherwise
`string()`	converts the argument into a string

operators	symbol	comment
round arithmetic brackets	( )	
power (exponentiation)	^	right associative
unary minus and unary plus	- +	right associative
arithmetic multiplication, division, remainder	* / %	remainder is modulo
arithmetic plus and minus	+ -	
arithmetic comparisons	< <= >= >	less than, less than or equal, . . .
arithmetic comparisons	== !=	equal, not equal
logical NOT	~	right associative
logical AND	&&	
logical OR	||	

8.6.2 Arithmetic comparisons and logical operators

You have to define separate variable beforehand if you want to use any for conditional statements. The logical operators, conditional blocks, and conditional comments consider any **nonzero** number as **true**, and **zero** as **false**.

Example 1

```
$a = 3
$b = 1
$c = $a > $b
!WHEN $c ...
```

The conditional line will be executed as `$c` equals `1`.

Example 2

```
$a = 3
$c = $a > 5
!WHEN $c ...
```

The conditional line will not be executed as `$c` equals `0`.

Example 3

```
$a = 1
$c = $a && 0
!WHEN $c ...
```

The conditional line will not be executed as `$c` equals `0`.

Attention

While the results of all comparison operators and logical operators are `1` and `0` as well, this may change in the future releases.

Attention

One should be careful when comparing the results of floating point computations, e.g., (1/3)*3 has the value 0.99999999. . . not 1.0, and use `round()` if necessary.

8.6.3 Dealing with floating-point numbers

Use `round()` if necessary when calling operators of arithmetic comparison on floating-point numbers to avoid errors.

Example 1

```
$a = (1/3)*3
$c = $a && 1
!WHEN $c ...
```

The conditional line will not be executed as `$a` has the value `0.99999999...` not 1.0 ``, therefore, ``$c equals `0`.

The function `string()` converts the argument into a string, which can be used to obtain a string representation of a floating point variable. This string representation may differ for different computer architectures, operating systems, and software releases.

8.6.4 Functions for array variables

Array variables can be subscripted using round brackets `()`. If the array subscript is out of range, a run-time error will occur.

Example 1

```
$vector = [1, 3, 5, 7]
$element = $vector(2)
```

`$element` equals 3

Example 2

```
$vector = [1, 3, 5, 7]
$element = $vector(5)
```

Run-time error occurs.

In addition, for the use with array variables, the following function is available:

Dimension of an array variable can be obtained using function `dim()`.

Example 3

```
$vector = [1, 3, 5, 7]
$size = dim($vector)
```

`$size` equals 4

8.7 Debug statements

Next, there are also a couple of debug statements available, that can be used at any (reasonable) point inside an input file or validation file:

```
!VARS    # prints all variables with their values into the standard output
!TABLE   # prints the entire symbol table into the standard output
```

Example:

```
--- Variables at line 14 -------------
$QW_WIDTH = 6
$QW_SEPERATION = 4
$QW_min = 20
```

(continues on next page)

(continued from previous page)

```
$QW_max = 26
--------------------------------------
```

But note that result of these debug statements obviously depends on their location in the file. Additionally, all variables and their values that are used in a simulation are written to the output folder into a file called

- `variables_input.txt` (for variables used in the input file)
- `variables_database.txt` (for variables used in the database file)

8.8 Groups and attributes

Next, we define *groups* and *attributes*. Their name follows (except for the leading dollar symbol `$`) the same convention as variable names. Validation files may also contain *groupnames* starting with a question mark `?`. We have here the following syntax:

```
groupname{
   attribute1 = value1
   group2{
      attribute1 = value2  # Each group has its own scope !!
      ....
   }
   group2{                 # groups with the same name and content may␣
↪repeat
      ....
   }
   attribute2 = value2     # but attributes are unique.
}                          # the group groupname ends here
```

Note that the order of groups is relevant in some cases, but the order of the attributes in a group is always ignored. Also note that groups may be empty as:

```
emptygroup{}
```

The curly brackets `{}` belonging to each group are checked for correctness.

There exist different types of attributes. Allowed are

- real numbers

  ```
  x = 12.121
  ```

- integers

  ```
  i = 12
  ```

- vectors of real numbers

  ```
  xV = [12.3e-4, 2, 3, sqrt(54.12)+2.1]
  ```

- vectors of integers

  ```
  iV = [1, 2]
  ```

- strings

  ```
  c="ohohi-oh ./opij "  # But many exotic characters are not allowed!
  ```

- choices and

```
color = red            # Pick one from a set of tokens
```

- enumerations

```
food = "juice bread dessert"  # Pick subset from a set of tokens
```

Attributes may also (like variables) be initialized with values of variables or the results of computations. But note that unlike variables, attributes may neither be redefined nor be used in mathematical expressions.

8.9 XML Tags

In addition, it is possible to add tags to explicitly check the current scope. For example,

```
groupname{
   ...
   <groupname>
   ...
}
```

or

```
groupname{
   ...
<groupname>}
```

will have no effect, while

```
groupname{
   ...
<differentgroupname> }
```

will cause an error message, since the assumed scope and the actual scope do not match.

Input files may also be decorated at the root level (i.e. outside of any group) with XML tags such as

```
<id>
</id>
```

or also:

```
<id/>
```

Here, `id` follows (except for the leading dollar symbol) the same convention as variable names. For backwards compatibility, in addition, also the empty (non-XML) tag <> is still available to e.g. check root level group closure. Please note that, whereas the simulator completely ignores the content of XML tags, they may have special meaning for calling programs such as *nextnanomat* and thus should not be altered without understanding their use. Practically, this means that, outside of groups, you may decorate input files/templates or also databases with XML tags in any way you wish. Just make sure to comment out stuff to be ignored by *nextnano*++ with double comments ## (to avoid possible collisions with conditional ifs) in order to add things such as:

```
<description>
##
## any stuff you want, e.g. rich text, nextnano++ will happily ignore it
##
</description>
<variables> $mass = 0.067 <unit># m_0</unit> </variables>
```

At the root level, one can use the empty tag

```
<>
```

to check for the root scope. This is optional and not required. Tags with these brackets `<...>` are ignored by the parser and can be used to provide additional meta data. That is, everything right (or left) of such symbols is executed normally, as if there was just a `;` (optional separator) instead of each tag.

Example:

```
<tag/>
```

It might look like an XML tag but it is much simpler. Nesting and matching tags are not checked. No blanks or special characters except underscores _ may be used within tags `<...>`.

8.10 Additional Examples and Remarks

E.g. you can define:

```
$pi = 4 * atan(1)
```

This will give `3.1415926535897932384626433832795029`. You can also specify:

```
$pi = 3.1415926535897932384626433832795029
```

Variable evaluation occurs already during parsing of the input/database file and thus before the beginning of the actual simulation. The input file after variable evaluation and the database file after the variable evaluation and possible modification by *database{ }* in the input file (which are the *real* inputs of the simulation) are written into files

- `simulation_input.txt` and
- `simulation_database.txt`.

In case of problems, or when many variables are used, it is highly recommended to review the file `simulation_input.txt` for possible mistakes. Similarly, `simulation_database.txt` will tell you (and our customer support) which values of material parameters were actually used for the simulation.

Further remarks

Except within comments, input files are strictly 7-bit ASCII. That is, no umlauts, diacritics, etc. in strings, names, etc. This is an inherent limitation of the parser. Command line parameters, file paths, and file names may contain all characters except \ / ? * ^ & ' ` < > : " and *control* characters (e.g. *newlines*). Unfortunately, e.g. on (US localized) windows, file names or file paths containing characters outside of code page 1252 (https://en.wikipedia.org/wiki/Windows-1252) may not be found or properly processed. Similar issues also may arise for other Windows localizations or for other operating systems. In order to avoid such problems, please make to sure to avoid characters outside of code page 1252 for all file names and file paths.

CHAPTER

NINE

SIMULATION OUTPUT

Here, we will add soon more information on the content of the output file names.

9.1 Basic information

For each simulation run, a new output folder is created in the simulation output folder. The created folder has the name of the input file. In addition date-time is added to the folder name if the option is selected in Options->Expert settings of *nextnanomat* (this option is recommended in order to avoid overwritten existing output data). The created output folder contains:

- the **input file** (.in).
- a folder '**...**' which gives material parameters used in the calculation.
- a folder **...** (only if the strain option is activated).
- Several files related to the sweep made. For a voltage sweep, it contains ....
- a **log file** is created at the end of the simulation, containing all the information displayed during the simulation.

9.2 Diagnostic information and error handling with log file

> Every simulation is generating a file with an extension .log. Let say one is running an input file *my_simulation.nnp*, then there is always a file *my_simulation.log* generated in the simulation output directory. This file contains diagnostic data on the the simulation process. The amount of this information can be controlled by the **debug** keywords available in most of the global groups. This file can contain warnings, if some solvers failed to converge to requested residuals, and error messages. For possible error messages please refer to the site: *Error and Warning Messages*.

9.3 Visualization - VTK and AVS

Specification of options for the visualization of the data with certain programs like

- Origin (1D/2D)
- VTK VTK format (2D/3D)
- AVS/Express (2D/3D)

9.3.1 VTK format for rectilinear grid

==> VTK - The Visualization Toolkit

The `.vtr` format can be read by the following software:

- VisIt visualization tool (free)
- ParaView (open source)
- ImageVis3D (open source)

9.3.2 AVS format for rectilinear grid

The `.fld` format can be read by the following software:

- AVS/Express visualization tool (commercial)

The main file of AVS format has `.fld` extention. Here is an example:

```
# AVS/Express field file       # necessary header
#
ndim   = 3                     # number of dimensions

dim1   = 6                     # number of nodes along 1st dimension
dim2   = 6                     # number of nodes along 2nd dimension
dim3   = 6                     # number of nodes along 3rd dimension
                               # number of dim* entry must be consistent to number of␣
↪dimensions "ndim"
nspace = 3                     # must be equal to "ndim"
veclen = 1                     # number of components of vector field, "1" = scalar␣
↪field
data   = double                # data type. Currently only "double" and "integer" are␣
↪supported.
field  = rectilinear           # type of mapping. Only rectilinear field is supported.
label  = bandedge_Gamma_1      # label for each vector field component
unit   = eV                    # unit of each vector field component (internally in␣
↪tool not used at the moment)

variable 1  file=3D_import.dat    filetype=ascii  skip=0   offset=0  stride=1  #␣
↪defines where 1st component of vector field is saved. Numbering must be ascending,␣
↪starting with "1"
                                                                               # and␣
↪number of "variable" "i" lines must be equal to "veclen". Supported file types are
↪"ascii" and "binary".
                                                                               # "skip
↪"   defines how many lines in file have to be skipped before data item appears.
                                                                               #
↪"offset" defines how many columns in line have to be skipped before searched data␣
↪items appear.
                                                                               #
↪"stride" defines how many steps have to be made before next data item appears.
coord 1     file=3D_import.coord  filetype=ascii  skip=0   offset=0  stride=1  #␣
↪contains information about where and how nodes of 1st coordinate are stored
coord 2     file=3D_import.coord  filetype=ascii  skip=6   offset=0  stride=1  #␣
↪contains information about where and how nodes of 2nd coordinate are stored
coord 3     file=3D_import.coord  filetype=ascii  skip=12  offset=0  stride=1  #␣
↪contains information about where and how nodes of 3rd coordinate are stored
                                                                               #␣
↪numbering must be ascending, starting with "1" and number of "coord" "i" lines must␣
↪be equal to "ndim"
# "skip=6" (=7) and "skip=12" (=14) take into account one empty line each that we use␣
↪to separate the coordinates.
```

The following shows an example of a file that can be imported using *import{ }*.

This example shows how to import

i, j, k, f_n(i,j,k), f_m(i,j,k)

ordered data via AVS format `3D_origin-format.fld` file into *nextnano*++:

```
# AVS/Express field file
#
ndim   = 3
dim1   = 3
dim2   = 3
dim3   = 3
nspace = 3
veclen = 2
data   = double
field  = rectilinear
label  = data_1
label  = data_2

variable 1  file=3D_origin-format.dat  filetype=ascii  skip=0   offset=3  stride=5
variable 2  file=3D_origin-format.dat  filetype=ascii  skip=0   offset=4  stride=5

coord 1     file=3D_origin-format.dat  filetype=ascii  skip=24  offset=0  stride=5
coord 2     file=3D_origin-format.dat  filetype=ascii  skip=18  offset=1  stride=15
coord 3     file=3D_origin-format.dat  filetype=ascii  skip=8   offset=2  stride=45
```

The corresponding data is contained in the `3D_origin-format.dat` file:

```
 0  0  0    1 -1    # The columns correspond to coordinates x,y,z and data values f_
↪1(x,y,z) and f_2(x,y,z).
 5  0  0    2 -2
10  0  0    3 -3
 0  5  0    4 -4
 5  5  0    5 -5
10  5  0    6 -6
 0 10  0    7 -7
 5 10  0    8 -8
10 10  0    9 -9
 0  0  5   10 -10
 5  0  5   11 -11
10  0  5   12 -12
 0  5  5   13 -13
 5  5  5   14 -14
10  5  5   15 -15
 0 10  5   16 -16
 5 10  5   17 -17
10 10  5   18 -18
 0  0 10   19 -19
 5  0 10   20 -20
10  0 10   21 -21
 0  5 10   22 -22
 5  5 10   23 -23
10  5 10   24 -24
 0 10 10   25 -25
 5 10 10   26 -26
10 10 10   27 -27
```

Note that the order of the values matters.

CHAPTER TEN

COMMAND LINE

Command line usage:

The general form is `nextnano++_Intel_64bit.exe [runmode] [options] filename1 [filename2 ... ]`, where `filename1` is the input file you want to simulate.

An example for Windows:

```
$ nextnano++_Intel_64bit.exe --license "C:\My Documents\nextnano\
↪License\License_nnp.lic" --database "C:\Program Files\nextnano\2024_
↪12_22\nextnano++\Syntax\database_nnp.in"  --threads 4 --
↪outputdirectory "C:\My Documents\nextnano\Output" "C:\Program Files\
↪nextnano\2024_12_22\Sample files\nextnano++ sample files\HEMT_1D_nnp.
↪in"
```

Available optional **runmodes** are:

-v, --version	Show version number only.
-h, --help	Show command line usage only.
-p, --parse	Parse input file(s) and quit.
-s, --structure	Parse input file(s), generate structure(s), and quit.
--postprocess	Enable postprocessing from input file using *postprocessor{ }*

Available **options** are:

-d database_file, --database database_file	Use database file <`database_file`>.
-l license_file, --license license_file	Use license file <`license_file`>. Example: `--license "C:\My Documents\nextnano\License\License_nnp.lic"`
-i input_directory, --inputdirectory input_directory	Specify input directory <`input_directory`>.
-o output_directory, --outputdirectory output_directory	Specify output directory <`output_directory`>.
-n, --noautooutdir	Do not create output directory(ies) with same name(s) as input file(s). (= **no auto**matic **out**put **dir**ectory)
-q, --quick	Enable quick updates of convergence log files.

Multi-threading

-t i, --threads i	Set number of parallel threads. Here, `i` threads are specified, any integer value between `0` and `1023` is allowed. *Not displayed and effective in serial executables. Currently we do not provide serial executables any more.*

Using `--threads 0` is equivalent to not specifying `--threads` at all, i.e. the code does not attempt to change the number of threads used.

Maximum value for `--threads` is the number of CPU cores, or possibly twice that number if Hyper-threading is enabled.

For default value of `0`, OpenMP system supplied maximal value is used.

If set (e.g. using *nextnanomat* Expert Settings), the number of parallel OpenMP threads is set to the supplied value. If the desired value is too large for the CPU, the maximum value available for the CPU is set. If not set or set to `0`, the default value as specified by the environment is used (usually `1` or all available). The actually used value is output near the beginning of the log file.

For example, on an i7-8700 CPU (6 cores and 12 threads with Hyper-threading on), the optimal number for best performance is 4. Using the extra threads from Hyper-threading rather hurts performance, and issues like memory speed seem to require a further reduction to less than 6 threads. With 4 threads, CPU load is about 45-50% on the tested CPU. This feature may also be useful for HTCondor to reduce background load, or to limit individual load for multiple parallel nextnano processes

-b i, --blas_threads i Set number of parallel threads in BLAS, LAPACK, etc. Here, `i` threads are specified, any integer value between `0` and `1023` is allowed.

Allows to separately set the number of BLAS (MKL) threads (MKL = Intel Math Kernel Library).

Maximum value for `--blas_threads` is typically the number of CPU cores.

Default value is `0` (Then uses the same number as the global number of threads which can be set by `-t` or `--threads`.)

For default value of `0`, and if `--threads` is not specified or `0`, the MKL library supplied maximal value is used.

Note

Additional notes on multi-threading

When only running one job at a time, setting `--threads` and `--blas_threads` to the number of CPU cores typically gives best performance. To force serial execution of each job, set both `--threads` and `--blas_threads` to 1.

Note that (the number of threads times the number of parallel jobs) and also (the number of BLAS threads times the number of parallel jobs) should not exceed the number of cores in order to avoid performance penalties from oversubscribing the CPU. Limited memory bandwidth may even impose lower limits on notebooks and lower grade desktop PCs.

Values for `--threads` and `--blas_threads` larger than the system supplied maximal values are automatically adjusted downwards. If unexpected values are automatically set (see log file for output), please also check your environment variables such as `OMP_NUM_THREADS` or `MKL_NUM_THREADS`.

-g, --generate **Generate additional debug information.**

Also outputs syntax definition files `input_syntax.txt` and `database_syntax.txt`.

Additionally, the files `keywords_nnp.xml` and `database_nnp.xml` are created, which are used by *nextnanomat* for its auto completion feature.

Example: `nextnano --license License_nnp.lic --outputdirectory "H:\nextnano\Output\" QuantumDot.in`

Soft kill

If the user places or creates a file called `SOFT_KILL` (without file extension) into the root output folder of the currently running simulation, a softkill will be performed, i.e. the program exits the iteration cycle and writes the output.

The concrete effects are the following:

1. As soon as the `SOFT_KILL` file is detected (may take a while), any running classical or quantum iteration will be terminated early, but all (incomplete) results will be written into files. Note that the detection is only performed at the beginning of each iteration step.
2. If the `SOFT_KILL` file is detected in the classical current-poisson equation, no quantum or optical calculations will be performed afterwards, i.e. only classical (incomplete) results will be written into files.
3. After any detection, subsequent sweeps will still be executed but their data will be incomplete in the same way. (We also could prevent further sweeps if this is the preferred approach.)
4. The `SOFT_KILL` file is not being removed at the end of the simulation. However, old `SOFT_KILL` files are automatically removed at the beginning of the simulation and thus will not cause any trouble.
5. If there are multiple simulations running in parallel (or being scheduled sequentially), separate `SOFT_KILL` files need to be placed in the respective root output folders.

Further remarks

Priorities in descending order

1. Full (*absolute*) paths with file names have the highest priority, e.g. `H:\nextnano\...`
2. Input and output directories (both *relative* and *absolute*), defined in command line, have priority over *absolute* directory paths (not file paths) defined in input file.

Rules

Default input directory is the directory, where the input file is located (not the current working directory). It can be redefined in command line (`--inputdirectory`) or in the input file (*import{ }*). By default the output of the simulation is written into an automatically generated directory with the same name as the input file. This default behavior can be suppressed using the command line flag `--noautooutdir`. If no output directory is defined in the command line or input file, the output of the simulation is written into the current working directory (including the automatically generated directory unless it is not suppressed). Relative input and output directory paths defined in the command line are relative to the current working directory. Relative paths to directories, defined in the command line and in the input file are always concatenated. Command line definitions have priority over definitions in the input file. If in the command line a *relative* or *absolute* path (`--inputdirectory` / `--outputdirectory`) is defined, the corresponding *absolute* directory path in the input file is ignored.

Examples

- `--inputdirectory` in command line is *not defined*

```
import{ # if no directory is specified,
        # the directory where the input file is located
        # is taken as the input directory
   directory   = "D:\import_files\"     # absolute path
               =   "\import_files\"     # root path
               =    "import_files\"     # relative path with respect
                                        # to current working directory

   file{
      ...
      filename = "D:\any_filename.fld"  # absolute path. The above specified␣
↪directory is ignored.
               =   "\any_filename.fld"  # root path. The above specified␣
↪directory is ignored.
```

(continues on next page)

(continued from previous page)

```
                = "any_directory\any_filename.fld" # relative path␣
↪concatenated with path specified by directory.
                =    "any_filename.fld"  # file is searched in directory
   }
}
```

- `--inputdirectory` in command line is *defined*, e.g.

  `--inputdirectory D:\inputdir` # absolute path

  `--inputdirectory \inputdir` # root path

  `--inputdirectory inputdir` # path relative to current directory

```
import{ # if no directory is specified,
        # the directory specified in the command line
        # is taken as the input directory
   directory   = "D:\import_files\"    # absolute path is ignored because of␣
↪definition in command line
               =   "\import_files\"    # root path is ignored because of␣
↪definition in command line
               =    "import_files\"    # relative path concatenated with␣
↪path specified in command line

   file{
      ...
      filename = "D:\any_filename.fld"           # absolute path. The above␣
↪specified directory and the path specified in the command line are ignored.
               =   "\any_filename.fld"           # root path. The above␣
↪specified directory and the path from the command line are ignored.
               = "any_directory\any_filename.fld" # relative path␣
↪concatenated with path specified by command line and/or path specified by␣
↪directory.
               =    "any_filename.fld"           # file is searched in␣
↪directory defined by the command line and directory
   }
}
```

The whole output of a simulation is written out in a directory named as the input file. This can be suppressed by command line flag `--noautooutdir`.

- `--outputdirectory` in command line is *not defined*

```
output{  # if no directory specified,
         # the current directory is taken as output directory
   directory = "D:\simulation_output\"   # absolute path
             =   "\simulation_output\"   # root path
             =    "simulation_output\"   # relative (to current directory)␣
↪path
}
```

- `--outputdirectory` in command line is *defined*, e.g.

  `--outputdirectory D:\outputdir` # absolute path

  `--outputdirectory \outputdir` # root path

  `--outputdirectory outputdir` # relative (to current directory) path

```
output{  # if no directory specified,
         # the directory specified in command line
         # is taken as output directory
   directory = "D:\simulation_output\"   # absolute path is ignored due to␣
↪definition in command line
             =   "\simulation_output\"   # root path is ignored due to␣
↪definition in command line
             =    "simulation_output\"   # relative path concatenated with␣
↪path specified in command line
}
```

Last update: 08/01/2025

CHAPTER

ELEVEN

MAXIMIZING PERFORMANCE

The *nextnano++* releases published after 2021/12/24 use significantly more parallelization than previous versions.

The following settings are recommended unless a notebook or an ancient PC is used. This is illustrated for the example of a CPU 8 physical having 16 logical cores:

1. Single simulation with multiple thread (especially useful for larger simulations)
    - Set maximum number of simulations to 1.
    - Set number of threads to the number of physical cores (8 in the present case).
    - Use "normal" process priority if running on a dedicate machine, or if you are not bothered by the CPU load.

> **Note**
>
> This settings provide high simulation performance with the lowest possible memory usage.

2. Settings for large number of simulations in parallel (especially useful for smaller simulations)
    - Set maximum number of simulations to the number of number of physical cores (8 in the present case).
    - Set number of threads per simulation to 1.
    - Use "normal" process priority if running on a dedicate machine, or if you are not bothered by the CPU load.

> **Attention**
>
> This settings require much more RAM than when a single simulation is run, as computer will allocate RAM for all simulations at the same time. Also, when writing output, all the simulation may try accessing output directory at the same time, resulting in high disc load. Make sure that your output folder is located on a fast SSD.

The max. number is specified from "Tools->Options->Simulation" in *nextnanomat*. The process priority is specified from "Tools->Options->Expert settings".

On Linux, corresponding optimal settings apply.

Also note that especially 3D simulations may write huge amounts of data (GBytes) to disk, i.e. using SSDs is highly recommended.

CHAPTER

TWELVE

RELEASE NOTES

12.1 1.25.13.b (2025-04-25)

Bugfixes

- Important bugfix for state classification in optics
- Bugfix for importing generation rates for periodic structures
- Output of matrix elements is restored
- Minor for interband matrix elements and number of states

General Changes

- Further improvements of convergence when currents are solved, different residuals may be needed to obtain meaningful solutions.
- Solver for optical spectra with Fermi's golden rule
- Improved convergence for current-poisson
- Output units of all densities has been set to cm^{-3}

classical{}

- New keyword `classical{ limit_classical_density }` introduced to allow improving convergence of classical Poisson in certain cases
- New 1-band band structures of bulk materials for valleys X, Δ, and L can be now output with new keywords: `classical{ bulk_dispersion{ X{ } } }`, `classical{ bulk_dispersion{ Delta{ } } }`, and `classical{ bulk_dispersion{ L{ } } }`
- Better control of the 8-band $\mathbf{k} \cdot \mathbf{p}$ model is now available for bulk electronic band structures via keywords `classical{ bulk_dispersion{ KP8{ electron_far_band } } }`, `classical{ bulk_dispersion{ KP8{ correct_electron_gfactor } } }`, `classical{ bulk_dispersion{ KP8{ rescale_kp_everywhere } } }`, and `classical{ bulk_dispersion{ KP8{ avoid_spurious } } }`

contacts{}

- New keyword for defining bias ramping along selected bias paths is introduced `contact{ bias_steps }`, `contact{ reuse_previous }`, and `contact{ bias_output_level }` are introduced.
- Previously used keywords `contacts{ ...{ steps{} } }` remain available with old behavior when used for sweeping bias for only one contact, but becomes obsolete and are going to be removed in the future.

currents{}

- Allowed range of values for `currents{ minimum_density_electrons }`, `currents{ minimum_density_holes }`, `currents{ maximum_density_electrons }`, `currents{ maximum_density_holes }` has been readjusted.

- Definition, defaults, and range of `currents{ linear_solver{ rel_accuracy } }` has been changed.

poisson{}

- New initialization method for Poisson equation has been introduced with a keyword `poisson{ zero_charge{ } }`. It provides better convergence conditions for certain simulations.
- Definition, defaults, and range of `poisson{ linear_solver{ rel_accuracy } }` has been changed.

run{}

- Maximum and minimum densities entering teh current equations can be controlled separately for algorithms with and without the Schrödinger equations using keywords `run{ current_poisson{ minimum_density_electrons } }`, `run{ current_poisson{ minimum_density_holes } }`, `run{ current_poisson{ maximum_density_electrons } }`, `run{ current_poisson{ maximum_density_holes } }`, `run{ quantum_current_poisson{ minimum_density_electrons } }`, `run{ quantum_current_poisson{ minimum_density_holes } }`, `run{ quantum_current_poisson{ maximum_density_electrons } }`, and `run{ quantum_current_poisson{ maximum_density_holes } }`
- The current equation can be additionally solved at the end with new keywords `run{ current_poisson{ smooth_currents } }` and `run{ quantum_current_poisson{ smooth_currents } }`

strain{}

- Definition, defaults, and range of `strain{ linear_solver{ rel_accuracy } }` has been changed.

quantum{}

- `quantum{ region{ quantize_x } }`, `quantum{ region{ quantize_y } }`, `quantum{ region{ quantize_z } }` are now allowed for wurtzites.
- More options allowed for state classification with groups `quantum{ region{ kp_8band{ classify_none{} } } }`, `quantum{ region{ kp_8band{ classify_by_energy{} } } }`, `quantum{ region{ kp_8band{ classify_by_all_energies{} } } }`, `quantum{ region{ kp_8band{ classify_by_spinor{} } } }`, and `quantum{ region{ kp_8band{ classify_by_all_spinors{} } } }`.
- spin quantization direction can be selected using `quantum{ region{ spin_quantization_axis } }`
- `quantum{ region{ interband_matrix_elements{ } } }` and `quantum{ region{ intraband_matrix_elements{ } } }`, with related outputs, have been finally properly named as `quantum{ region{ overlap_integrals{ } } }` and `quantum{ region{ momentum_matrix_elements{ } } }`, respectively.

12.2 1.22.18.b (2024-12-18)

- More Dirac deltas are possible to add to imaginary part of dielectric function for Kramers-Kronig relations; See new keywords in `optics{ quantum_spectra{ kramers_kronig{ } } }`.
- Experimental implementation of position resolved optical spectra, see `optics{ quantum_spectra{ local_absorption } }`, `optics{ quantum_spectra{ local_spontaneous_emission } }`, and `optics{ quantum_spectra{ output_local_spectra{ } } }`.
- Major changes in `optics{ }` group related to calling models and controlling output.
- Bugfixes for `optics{ light_propagation{ } }`
- New incomplete ionization model, see `impurities{ acceptor{ N_ref } }` and `impurities{ acceptor{ c } }`.

- Allowed range of values for `currents{ minimum_density_electrons }` and `currents{ minimum_density_holes }` is increased - lower values are available now.
- Improvement of diagnostic output for k.p models.
- Multiple fixes related to missing units and incorrect labels for 2D and 3D simulations.
- Improved output of integrated densities, see `bias_*\total_charges.txt`.
- Hardened security for script postprocessing, see *postprocessor{ }* and *Command Line*.
- Bugfix for B parameter of band k.p models.
- Important improvements of implementation of the interface Hamiltonian.
- Multiple bugfixes in drift-diffusion solver resulting in improvement of convergence for many simulations with currents, especially for 2D and 3D simulations.
- Integrals of total current around contact improved with visible impact on I-V characteristics for 2D and 3D simulations.
- Added output of summary log file `summary.log`.

12.3 1.20.8.b (2024-08-22)

- Important bugfixes and multiple improvements of the code for optical spectra
- Important bugfix for poisson equation for wurtzite simulations in 1D
- Output keywords related to piezo- and pyroelectric charges, and polarization vectors has been changed. Related output files are named differently; Still, they can be found in the folder `Strain`.
- `total_charges.txt` now includes also integrals of piezo- and pyroelectric charges.
- `optics{ semiclassical_spectra{ output_spectra{ emission } } }`, `optics{ semiclassical_spectra{ output_spectra{ photon_spectra } } }`, and `optics{ semiclassical_spectra{ output_spectra{ power_spectra } } }` has been placed by `optics{ semiclassical_spectra{ output_spectra{ emission_photons } } }` and `optics{ semiclassical_spectra{ output_spectra{ emission_power } } }`
- `optics{ semiclassical_spectra{ output_local_spectra{ emission } } }`, `optics{ semiclassical_spectra{ output_local_spectra{ photon_spectra } } }`, and `optics{ semiclassical_spectra{ output_local_spectra{ power_spectra } } }` has been placed by `optics{ semiclassical_spectra{ output_local_spectra{ emission_photons } } }` and `optics{ semiclassical_spectra{ output_local_spectra{ emission_power } } }`
- `optics{ quantum_spectra{ k_integration{ symmetry } } }` is removed as was not bringin expected improvement of computantional performance.
- `structure{ integrate{ ionized_donor_density{ } } }`, `structure{ integrate{ ionized_acceptor_density{ } } }`, and `structure{ integrate{ fixed_charge_density{ } } }` are introduced.
- `currents{ electron_mobility{ } }` has been deprecated and fully replaced by functionality of `currents{ electron_mobility{ } }` and `currents{ hole_mobility{ } }`
- `currents{ electron_mobility{ high_field_model{ } } }` has been deprecated and replaced by `currents{ electron_mobility{ high_field_model } }`

12.4 1.19.61.a (2024-06-28)

- improvements and bugfixes for `optics{ quantum_optics{ } }`
- excitons added to spectrum components output

- `grid{ xgrid{ repeat{ } } }`, `grid{ xgrid{ repeat2{ } } }`, `grid{ ygrid{ repeat{ } } }`, `grid{ ygrid{ repeat2{ } } }`, `grid{ zgrid{ repeat{ } } }`, `grid{ xgrid{ repeat2{ } } }` becomes deprecated
- `optics{ quantum_spectra{ make_spin_degenerate } }` becomes deprecated

12.5 1.19.49.a (2024-06-17)

- `region{ repeat_x }`, `region{ repeat_y }`, `region{ repeat_z }`, `region{ repeat2_x }`, `region{ repeat2_y }`, `region{ repeat2_z }` becomes deprecated
- Initial implementation of interface Hamiltonian for 8-band zincblende $\mathbf{k} \cdot \mathbf{p}$, `quantum{ region{ kp_8band{ interface{...} } } }`
- Multiple improvements and bugfixes for `optics{ }`
- Minor bugfix for exchange correlation

12.6 1.19.22.a (2024-05-14)

- missing terms added to the 14- and 30-band $\mathbf{k} \cdot \mathbf{p}$ models
- minor bugfix for strain in the 14- and 30-band $\mathbf{k} \cdot \mathbf{p}$ models
- other minor bugfixes

12.7 1.19.17.a (2024-04-28)

currents{ }

- `import_electron_fermi_level{ }` and `import_hole_fermi_level{ }` are introduced.

optics{ }

- `light_propagation{ use_local_absorption{ } }` got renamed to `light_propagation{ use_computed_absorption{ } }`
- `light_propagation{ use_local_absorption{ } }` reintroduced with different functionality
- multiple output settings added to `light_propagation{ }`, `photogeneration{ }`, and `semiclassical_spectra{ }`

12.8 1.18.63.b (2024-03-24)

quantum{ }

- `davidson{}` group introduced for 8-band $\mathbf{k} \cdot \mathbf{p}$ model
- `force_pauli_solver{}` group introduced for all one-band models

optics{ }

- bugfix for `irradiation{ illumination{ direction_* } } }`, now negative values are properly processed

- improvement of an existing feature `optics{ quantum_region{output_spectra{ output_components } } }` has different type and allows to output components of all spectra.
- syntax change from `irradiation{ photo_generation{ output_spectrum{} } }` to `irradiation{ photo_generation{ output_integrated{} } }`
- syntax change from `irradiation{ output_light_field }` to `irradiation{ photo_generation{ output_light_intensity }`
- `optics{ emission_spectrum{ output_spectra{ stimulated_emission } } }` removed
- `optics{ emission_spectrum{ output_local_spectra{} } }` introduced
- in multiple places `absorption` and `decadic_absorption` renamed to `absorption_coeff` and `decadic_absorption_coeff`
- `photogeneration{ }` updated and allowing to use computed generation rates within running simulation
- energy grid definitions are notably changed and partially moved to the group `grid{}`
- major groups `emission_spectrum{}` and `quantum_region{}` are renamed to `semiclassical_spectra{}` and `quantum_spectra{}`, respectively.
- `light_propagation{}` is introduced

classical{ }

- `output_energy_resolved_densities{}` moved inside `energy_resolved_density{}`
- `output_LDOS{}` group introduced
- `bulk_dispersion{ KP30{ } }` introduced following *[RideauPRB2006]*
- energy grid definitions from `grid{}` are used for densities

grid{}

- energy grid definitions introduced

run{ }

- `solve_strain{ }`, `solve_poisson{ }`, `solve_current_poisson{ }`, `solve_quantum{ }`, `outer_iteration{}` become deprecated and not supported anymore

database{ }

- complex refractive index is supported by `optical_refractive_index{}`
- extensive changes in the database relating to `optics{ }` group

command line

- `-r`, `--resume` option has been removed

12.9 1.17.20 (2023-08-07)

general input syntax

- `!DATA` statement got introduced for post-processor
- `!TEXT` and `!ENDTEXT` statements introduced for multi-line comments

classical{ }

- `output_band_densities{}` is introduced
- `bulk_dispersion{}` is moved from `quantum{ }` with a slight syntax change

- Bulk dispersions within 1-band models can be now also included in the output (offset might be still incorrect)

optics{ }

- `spin_align` is back after reviewing its functionality. Default value is changed to `no`
- `make_spin_degenerate`

currents{ }

- `robust` atribute is introduced to enhance accuracy of bisection algorithm.
- `eastman4{}` group is introduced to allow alternative parametrization of the Eastman-Tiwari-Shur mobility model
- `electron_contact` and `hole_contact` introduced to increase accuracy of quasi-Fermi levels

contacts{ }

- bisection algorithm initializing ohmic and charge-neutral contacts is enhanced

run{ }

- the group becomes required
- an attribute `output_local_residuals` is introduced for multiple groups

quantum{ }

- computing matrix elements for multiple polarization in one simulation is again possible within groups `intraband_matrix_elements{}` and `dipole_moment_matrix_elements{}`
- `bulk_dispersion{}` is moved to `classical{ }`

postprocessor{}

- entirely new group introduced to generate and run batch scripts after simulations

---

12.10 1.14.33 (2023-05-12)

optics{ }

- syntax change in k_integration{}: `num_integrationpoint` is introduced, `num_subpoints` is removed
- `spin_align` is removed
- `occupation_const_fermilevel` is renamed to `occupation_zero_fermilevel`
- `classify_states` and `classification_threshold` becomes available
- multiple improvements of the model
- optics for transitions between two 1-band models and between 1-band and 6-band remains under heavy development

quantum{ }

- definition of leads become mandatory for modeling transport with CBR method related syntax becomes improved and contained within a group `lead{}`

12.11 1.13.0 (2023-02-19)

optics{ }

- bug fix of sweeping bias related while computing optical spectra
- improvements of `excitons{}`

currents{ }

- new output group `output_forces{}` introduced
- `electron_contact` and `hole_contact` are introduced to allow enhanced accuracy for current equation around selected contacts

quantum{ }

- `cbr{}` group has been moved into the quantum group
- `ldos` choice attribute has been added to the `quantum{ cbr {} } group`

12.12 1.12.35 (2022-12-17)

In this release we introduced further syntax changes aiming at improving clarity of calling models. Selected bowing parameters of band edges has been updated in the default database. Multiple new sample input files are added to the installer. All input files with containing updated syntax.

output{ }

- `output{ section1D{} }` requires specifying at least two attributes **x**, **y**, and **z** for 3D simulations.

currents{ }

- New high-field mobility models (Hänch, Transferred-electron, Eastman-Tiwari-Shur) are implemented
- Improvement of algorithm convergence in case of using high-field velocities

quantum{ }

- Bug fix and syntax change relating computation of lifetimes, see `quantum{ region{ lifetimes} } }`
- Model of excitons within effective mass approximation is implemented, see `quantum{ regon{ excitons {} } }`

optics{ }

- **Major syntax change has been implemented. From now on:**
  - this group contains all keywords related to optical spectra - groups `emission_spectrum{}` and `irradiation{}` are included in this group, see *here*
  - group `region{}` has been renamed to `quantum_region{}`
- Excitonic effects can be included in spectra computations by calling `excitons{}` group.
- Bug fix related to symmetry attribute

classical{ }

- Major syntax change - groups `emission_spectrum{}` and `irradiation{}` has been removed from this group.
- From now on, this group relates only to choice of band edges and density outputs for semi-classical computations

strain{ }

- strain relaxation is initially implemented for entire by a scaling factor of all tensor elements, see `strain{ relaxation {} }`

run{ }

- calling quantum-optical simulation has been changed: `optics{ }` is renamed to `quantum_optics{ }`

12.13 1.10.19 (2022-08-09)

In this release we introduced some syntax changes and number of new keywords. Some algorithms got notably improved. We fixed number of bugs.

classical{ }

Default behavior of an attribute `refractive_index` has been changed. New Attributes are:

- `energy_broadening_gaussian`
- `energy_broadening_lorentzian`

optics{ }

Some algorithms have been improved so the group is faster. Number of new keywords has been introduced:

- `enable_hole_hole`
- `enable_electron_hole`
- `enable_electron_electron`
- `photon_spectra`
- `power_spectra`
- `use_for_emission`

poisson{ }

Self-consistent algorithm has been improved and convrges quicker. Behavior and way of initialising Poisson-equation solver has been improved. Related groups and attributes are:

- `import_potential{}`
- `electric_field{}`
- `between_fermi_levels{}`
- `charge_neutral{}` - it was an atribute before
- `reference_potential`

strain{ }

Rules of calling inside the group have changed. Related groups are:

- `no_strain{ }` - a new group
- `pseudomorphic_strain{ }`
- `minimized_strain{ }`
- `import_strain{ }`

12.14 1.9.92 (2022-06-08)

In this release we added support for decadic attenuation units (dB/μm) and new output options inside of `optics{ }` mirroring the corresponding functionality in `classical{ }`. Gain in `classical{ }` is now defined as the positive part of (minus absorption).

classical{ }
Introduced attributes are:

- `decadic_absorption`
- `decadic_gain`
- `decadic_absorption_unit`

optics{ }
Introduced attributes are:

- `decadic_absorption`
- `decadic_gain`

database{ }
Introduced attribute are:

- `decadic_absorption`

12.15 EARLIER

- Added periodic repetition of quantum regions
- Added electron injection (e.g. by electron beam) into structure definition
- Integration of *nextnano.MSB* into *nextnano*++, incl. *nextnano.MSB* sample files into installer
- Output reflection components of CBR transmissions
- New 2D CBR input files (QPC)
- Synonyms in material database (e.g. Al(x)In(x)As and In(x)Al(1-x)As)
- Calculation of reflection and extinction coefficient
- Gaussian and Lorentzian broadening for optical absorption
- Improvements for optical absorption (k.p)
- Improvements for k.p (speed: k=0 subspace expansion)
- Added more tutorial input files to samples folder
- New UVC LED AlGaN/GaN input files
- Improvements for intersubband absorption (k.p)
- Solar cell features, irradiation
- Added quaternaries and quinternaries to database; AlScN, AlYN, …
- Added XML support to input files
- New region objects: circle/sphere
- New region objects: triangle, polygonal_prism, regular_prism, hexagonal_prism, polygonal_pyramid, regular_pyramid, hexagonal_pyramid
- Array of different biases is allowed in addition to bias sweep using steps
- Output of emission spectrum for LEDs based on classical or quantum density

- Output of energy resolved density n(E) and n(x,E)
- Improved convergence and speed for current calculations
- More intuitive setting in run{ }
- MOSFET tutorial

ADDITIONAL NOTES

Note

The group **contact{ ohmic{} }** behaves like **contact{ charge_neutral{} }** by default since 2019-01-23, and it additionally contains a **shift** attribute.

Note

Currently, the group **contacts{ zero_field{} }** behaves like **contact{ ohmic{} }** before, until 2019-01-23.

Last update: 02/04/2025

CHAPTER

THIRTEEN

FAQS

13.1 Features and Functionalities

13.1.1 Where to find simulation LOG file

The simulation log file is a file with the same name as the input file and the extension `*.log`. It is located in the output folder of the simulation. It is necessary for the support team to debug issues with the simulation, thus it should always be attached to a support request. Please also include additional debug information in case the log file shall be used for support, refer to FAQ of *nextnanomat*.

13.1.2 How can I track how much memory is used during the simulations?

See FAQ of *nextnanomat*.

13.1.3 Can I pass additional command line arguments to the executable?

See FAQ of *nextnanomat*.

13.1.4 How can I speed up my calculations with respect to CPU time?

The most obvious way is to reduce the **number of grid points** you are using. For instance, for the following p-n junction simulation, a grid spacing of **1 nm** was used (gray lines in Figure 13.1.4.1). If one is using a coarse grid of only **10 nm**, the calculated values (squares in Figure 13.1.4.1) agree very well with the calculated values of the thin lines.

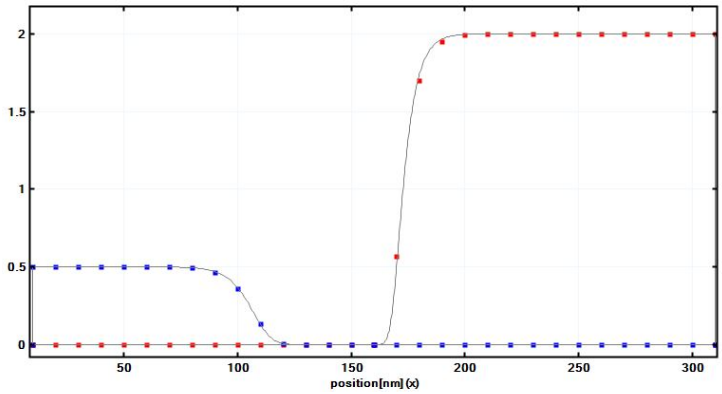


Figure 13.1.4.1: Hole (blue) and electron (red) densities of the p-n junction in units of $10^{18} cm^{-3}$. The gray lines are from simulations using a 1 nm grid spacing. The squares are from a simulation that uses only a 10 nm grid resolution. Note that the center coordinate of this plot is x=160 nm. The depletion width for the holes is around wp:math:*approx`50 nm, for the electrons it is wn:math:*approx`10 nm which is of the order of the grid spacing. Even in this case, the calculated electron density is reasonably accurate.

The difference in CPU time comes from the fact that for the 10 nm resolution the dimension of the matrix that is used for discretizing the Poisson equation is 30, while in the case for the 1 nm grid spacing it has the dimension 300. The proper choice of an optimal grid spacing is very relevant for 2D and 3D simulations, as can be seen in the following.

1D simulation (length of sample: x = 300 nm)

- 1 nm grid spacing: dimension of Poisson matrix: $N = 300$
- 10 nm grid spacing: dimension of Poisson matrix: $N = 30$

2D simulation (length of sample: x = 300 nm, y = 300 nm)

- 1 nm grid spacing: dimension of Poisson matrix: $N = 300 \cdot 300 = 90,000$
- 10 nm grid spacing: dimension of Poisson matrix: $N = 30 \cdot 30 = 900$

3D simulation (length of sample: x = 300 nm, y = 300 nm, z = 300 nm)

- 1 nm grid spacing: dimension of Poisson matrix: $N = 300 \cdot 300 \cdot 300 = 27,000,000$
- 10 nm grid spacing: dimension of Poisson matrix: $N = 30 \cdot 30 \cdot 30 = 27,000$

If a quantum mechanical simulation is performed, the numerical effort of eigenvalue solvers increases with the number of grid points N with order $O\left(N^2\right)$.

13.1.5 Can I take advantage of parallelization of the nextnano software on multi-core CPUs?

The short answer is:

Some numerical routines are parallelized which is done automatically. These are the numerical routines, e.g. for calculating the eigenvalues with a LAPACK solver (which itself uses BLAS).

The long answer is:

The nextnano software includes the Intel® Math Kernel Library (MKL). MKL includes the BLAS and LAPACK library routines for numerical operations. The MKL dynamically changes the number of threads.

- *nextnano*++ uses MKL (parallel version). The executables that are compiled with the Intel and Microsoft compilers use MKL (parallel version). The executable that is compiled with the GNU compiler (gcc/gfortran) uses the nonparallelized version of the BLAS and LAPACK source codes available from netlib webpage.
  - CBR (parallelization with respect to energy grid)

  - NEGF (parallelization with respect to energy grid and further loops) `number-of-MKL-threads = 8`
  - Calculation of eigenstates for each $k_{||}$ (1D and 2D simulations)
  - Matrix-vector products of numerical routines Note: Not all operations are thread-safe, e.g. one cannot combine $k_{||}$ parallelization with the ARPACK eigenvalue solver. Only for this executable, the flag `number-of-parallel-threads = 4` has an effect. The NEGF keyword also supports `number-of-MKL-threads = 4` (`0` means *dynamic* with is recommended) and `MKL-set-dynamic = yes` / `no`.
- *nextnano.NEGF* - uses MKL (parallel version)
- *nextnano.MSB* - uses MKL (parallel version)

The NEGF algorithms (*nextnano.NEGF*, *nextnano.MSB*, CBR) include matrix-matrix operations which are well parallelized within the BLAS routines.

If e.g. 4 nextnano simulations are running in parallel on a quad-core CPU, i.e. 4 nextnano executables are running simultaneously and each of them is using calls to the parallelized MKL library simultaneously, the total performance might be slower compared to running these simulations one after the other. In this case using a nextnano executable compiled with the serial version of the Intel MKL could be faster.

In fact, it strongly depends on your nextnano application (e.g. 1D vs. 3D simulation, LAPACK vs. ARPACK eigenvalue solver, …) if you benefit from parallelization or not. In general, the best parallelization can be obtained if you run several nextnano simulations in parallel. For instance, you could do parameter sweeps (e.g. sweep over quantum well width) using *nextnanomat*'s *Template* feature, i.e. if you run 4 simulations simultaneously on a quad-core CPU, e.g. for 4 different quantum well widths.

13.1.6 What boundary conditions are available?

There are three different boundary conditions that we use:

- *periodic*: $\psi(x=0)=\psi(x=L)$
- *Dirichlet*: $\psi(x=0)=\psi(x=L)=0$, and
- *Neumann*: $\frac{\mathrm{d}\psi}{\mathrm{d}x}=\mathrm{const}$ at the left ($x=0$) and right ($x=L$) boundary. Typically, const $=0$.

By specifying both Dirichlet and Neumann boundary conditions, the system would be over-determined.

13.1.7 What are quasi-Fermi levels

So-called *quasi-Fermi levels* which are different for electrons $E_{\mathrm{F,n}}$ and holes $E_{\mathrm{F,p}}$ are used to describe non-equilibrium carrier concentrations. In equilibrium the quasi-Fermi levels are constant and have the same value for both electrons and holes, $E_{\mathrm{F,n}}=E_{\mathrm{F,n}}=0$ eV. The electron current is proportional to the electron mobility $\mu_{\mathrm{n}}(x)$, carrier density $n(x)$ and the gradient of the quasi-Fermi level of the carriers, $\nabla E_{\mathrm{F,n}}(x)$, and analogously for the holes.

13.1.8 I do not understand the $\mathbf{k}\cdot\mathbf{p}$ parameters

In the literature, there are two different notations used:

- Dresselhaus–Kip–Kittel (DKK): L, M, N^+, N^- (zinc blende); $L_1, L_2, M_1, M_2, M_3, N_1^+, N_1^-, N_2^+, N_2^-$ (wurtzite)
- Luttinger parameters: $\gamma_1, \gamma_2, \gamma_3, \kappa$ (zinc blende); Rashba–Sheka–Pikus (RSP) parameters $A_1, A_2, A_3, A_4, A_5, A_6, A_7$ (wurtzite)

They are equivalent and can be converted into each other.

Some authors only use 3 parameters L, M, N (or $\gamma_1, \gamma_2, \gamma_3$) which is fine for bulk semiconductors without magnetic field but not for heterostructures because the latter require 4 parameters, i.e. N^+, N^- (instead of N only) or κ. If these parameters are not known, they can be approximated.

There are different $\mathbf{k}\cdot\mathbf{p}$ parameters for

- 6-band $\mathbf{k} \cdot \mathbf{p}$ and
- 8-band $\mathbf{k} \cdot \mathbf{p}$.

The 8-band $\mathbf{k} \cdot \mathbf{p}$ parameters can be calculated from the 6-band parameters taking into account the temperature dependent band gap E_{gap} and the Kane parameter E_{P} (zinc blende). For wurtzite the parameters are E_{gap} and the Kane parameters E_{P1}, E_{P2}.

The 8-band Hamiltonian also needs the conduction band mass parameter S (zinc blende) or S_1, S_2 (wurtzite). They can be calculated from the conduction band effective mass m_{c}, the band gap E_{gap}, the spin-orbit split-off energy Δ_{so} and the Kane parameter E_{P} (zinc blende). For wurtzite the parameters are $m_{\mathrm{c},\parallel}$, $m_{\mathrm{c},\perp}$, E_{gap}, Δ_{so}, the crystal-field split-off energy Δ_{cr} and the Kane parameters E_{P1}, E_{P2}.

Finally there is the inversion asymmetry parameter B for zinc blende. For wurtzite there are B_1, B_2, B_3.

For more details on these equations, please refer to Section *3.1 The multi-band* $\mathbf{k} \cdot \mathbf{p}$ *Schrödinger equation* in the PhD thesis of S. Birner.

Spurious solutions

Some people rescale the 8-band $\mathbf{k} \cdot \mathbf{p}$ in order to avoid *spurious solutions*. The 8-band $\mathbf{k} \cdot \mathbf{p}$ parameters can be calculated from the 6-band parameters taking into account the band gap E_{gap}, the spin-orbit split-off energy Δ_{so} and the Kane parameter E_{P} (zinc blende). For wurtzite the parameters are E_{gap}, the spin-orbit split-off energy Δ_{so}, the crystal-field split-off energy Δ_{cr} and the Kane parameters E_{P1}, E_{P2}.

For more details, please refer to Section *3.2 Spurious solutions* in the PhD thesis of S. Birner.

See section `quantum{ region{ kp_8band{} } }` in *kp_8band{ }*.

13.1.9 Can I add new materials to the database?

Yes.

Option 1 (quick)

Directly in the input file, you can overwrite certain material parameters of any material already defined in the database. For instance if you need ''HfO2", you can use the material ''SiO2" and just change the static dielectric constant and conduction and valence band edges or any other relevant parameters that you need. So basically, you are using the material ''SiO2" with modified static dielectric constant and band edges.

Please note that every material has assigned crystal symmetry of either zinc blende (including diamond type) or wurtzite.

Option 2 (general)

The material parameters are contained in ASCII text files. You can find them in the installation folder, e.g., `C:\Program Files\nextnano\<date>\nextnano++\database\database_nnp.in`.

These files can be edited with any text editor, such as Notepad++.

It is best if you search for a material such as ''GaSb" and then simply use ''Copy & Paste" to reproduce all relevant entries. Then you can rename ''GaSb" to anything else, like ''GaSb_test". Finally, you adjust the necessary material parameters that you need. In most cases you do not have to replace all material parameters. It is only necessary to replace the ones that you need in the simulation.

It is a good idea to save the new database to a new location, such as `C:\Users\<user name>\Documents\nextnano\My Database\database_nnp_GaSb_modified.in` You can then read in the new *nextnano++* database specifying the location within the Tools Options of *nextnanomat*.

`Tools` => `Options...` => `Workspace` => `nextnano++ database file:`

Hint

More information on how to add materials can be found in *Material Database*. Useful tutorial to practice definition is *— EDU — Interpolation of 2-component alloys*.

13.1.10 Should I use averaged outputs and boxes?

The `averaged = yes` is similar to `boxes = no`. Note that `boxes` is related to output of material grid points while `averaged` is related to output of simulation grid points.

2D and 3D simulations can produce a lot of output data (order of GB). It is strongly recommended to use `averaged = yes` for 2D and 3D simulations to avoid excessive consumption of your hard disk.

13.1.11 Current in my simulation seems unphysical. How to deal with it?

Most likely you are observing some spikes in the output of the total current. The reason is that the self-consistent algorithm did not converge to a reasonable solution.

To fix the problem you can begin with trying standard solutions listed in *Quantum-Current-Poisson fails to converge*. If they to not help you, then please follow our guides in *Convergence*. Meaning of residuals, which you should understand, is explained in *Residuals*.

If you are solving semi-classical system, then you should get familiar with the keyword group *current_poisson{ }*. In case of solving the Schrödinger equation as well, you need also to learn about functionalities of *quantum_current_poisson{ }*.

In the systems with extremely large or small carrier densities, you may need to stabilize the solver on the level of the drift-diffusion equations themselves, constraining carrier densities entering the equation by: *maximum_density_electrons*, *maximum_density_holes*, *minimum_density_electrons*, and *minimum_density_holes*.

Last update: 22/01/2025

13.2 Error and Warning Messages

13.2.1 Quantum-Current-Poisson fails to converge

Most often Quantum-Current-Poisson calculation fails to converge because:

a) The maximum number of iterations has been chosen too small.

   SOLUTION: Increase the number of iteration using *iterations*.

b) The number of electron or hole eigenstates has been chosen too small.

   SOLUTION: Check the occupations in the files `bias_*\Quantum\occupation_*.dat`. If the occupation does not drop from the ground state to the highest excited state by several orders of magnitude, you need to calculate more states. If you are using one-band model or 6-band $\mathbf{k}\cdot\mathbf{p}$ model then you need to adjust a keyword attributes `num_ev` in *Gamma{}, L{}, X{}, Delta{}, HH{}, LH{}, SO{}* or in *kp_6band{ }*. In teh case of using 8-band $\mathbf{k}\cdot\mathbf{p}$ model, the number of states can be adjusted by `num_electrons` and `num_holes` in *kp_8band{ }*.

c) The under-relaxation parameter has been chosen too large.

   SOLUTION: Check whether the solution oscillates, i.e., residuals do not drop continuously but increase in some iterations. Try to decrease the under-relaxation parameter controlled by *alpha_fermi* in order to damp the oscillations.

d) The minimum charge density in the current equation has been chosen too small (*currents{ }* ==> `minimum_density_*`).

   SOLUTION: Try to increase the minimum charge densities to improve conditioning of the current equation. Relevant keywords are *minimum_density_electrons* and *minimum_density_holes*.

13.2.2 WARNING: Linear solver residual of ARPACK-INVERSE set too large in multiband quantum solver

Linear solver residual of ARPACK-INVERSE set too large in multiband quantum solver. ARPACK-INV solver uses internally the linear solver on each iteration, therefore the accuracy of ARPACK-INV is limited by accuracy of the linear solver. It is recommended to set the residuals of linear solver to be smaller than residuals of the ARPACK-INV, otherwise this warning is thrown. The residual of the ARPACK-INV can be set by a keyword *accuracy*. The linear solver residuals can be set by keywords *linear_solver{ abs_accuracy }* or *linear_solver{ rel_accuracy }*.

13.2.3 Error: (nodes number of coordinate 1) != (lines number in file)

Simulation *.log file contains the following error

```
error:(nodes number of coordinate 1) != (lines number in file)
```

It means that you have defined values at some grid points twice inside the imported file. Check if some points are duplicated in the file that you are trying to import. For example, you should avoid situation as the following:

```
...
1.0    0.5
1.1    0.5
1.2    0.5
1.2    0.6
1.3    0.6
...
```

In this case the point `1.2` is defined twice, which is the source of the problem.

13.2.4 nextnano++ exit code: -1073741795

Simulation *.log file contains the following exit code -1073741795 or other big negative number

```
(nextnano++ exit code: -1073741795)
```

Most likely you need to install the Microsoft Visual C++ Redistributable . Choose the corresponding version matching your operation system architecture (most likely X64) from the section *Latest Microsoft Visual C++ Redistributable Version*. It's a typical error when running *nextnano++* on Windows Server OS.

It is also possible that your CPU is not suited for *nextnano++* executable, i.e., you are using processor from Pentium family, typical on old laptops. You can use **nextnano++_Microsoft_32bit_serial.exe**, see FAQ of *nextnanomat*.

Other possibility is that your computer enters sleep mode during the runtime of your simulation. Turn off disable sleep mode on your computer to fix the problem.

13.2.5 nextnano++ exit code: -1

Simulation *.log file contains the following exit code -1

```
(nextnano++ exit code: -1)
```

It means that the simulation has been aborted by *nextnanomat* or by other means.

13.2.6 nextnano++ exit code: 1

Simulation *.log file contains the following exit code 1

```
(nextnano++ exit code: 1)
```

It means that there is error in the command line calling the solver.

13.2.7 nextnano++ exit code: 3

Simulation *.log file contains the following exit code 3

```
(nextnano++ exit code: 3)
```

There is a problem with validating your license.

Last update: 22/01/2025

Last update: 25/10/2024

CHAPTER FOURTEEN

BOOKS

- [Adachi2009]
  Proerties of Semiconductor Alloys: Group-IV, III-V and II-VI Semiconductors
  S. Adachi
  John Wiley & Sons, Inc., New York (2009)
- [Beynon1996]
  Buffer solutions: The basics
  R. J. Beynon, and J. S. Easterby
  Oxford University Press (1996)
- [ChuangOpto1995]
  Physics of Optoelectronic Devices
  S. L. Chuang
  John Wiley & Sons, Inc., New York (1995)
- [DattaCambridge2005]
  Quantum Transport: Atom to Transistor
  S. Datta
  Cambridge University Press (2005)
- [DattaETMS1995]
  Electronic Transport in Mesoscopic Systems
  S. Datta
  Cambridge Studies in Semiconductor Physics and Microelectronic Engineering, Cambridge University Press, Cambridge (1995)
  DOI 10.1017/CBO9780511805776
- [Davies1998]
  J. H. Davies
  The Physics of Low-Dimensional Semiconductors - An Introduction
  Cambridge University Press (1998)
- [FaistQCL2013]
  Quantum Cascade Lasers
  J. Faist
  Oxford University Press (2013)
- [GrahnITSP1999]
  Introduction To Semiconductor Physics
  H. T. Grahn
  World Scientific Publishing (1999)
- [HarrisonQWWD2005]
  Quantum Wells, Wires and Dots
  P. Harrison

Theoretical and Computational Physics of Semiconductor Nanostructures, 2^{nd} ed., John Wiley & Sons, Ltd (2005)
DOI 10.1017/CBO9780511805776

- [HaugKinetics2008]
  Quantum Kinetics in Transport and Optics of Semiconductors
  H. Haug, A-P. Jauho
  Springer (2008)
- [NelsonPSC2003]
  The Physics of Solar Cells
  J. Nelson
  World Scientific Publishing (2003)
- [Landau Lifshitz]
  Quantum Mechanics Non-Relativistic Theory
  L. D. Landau, E. M. Lifshitz
  (Pergamon Press, Vol. 3)
- [LeviAQM2006]
  Applied Quantum Mechanics
  A. F. J. Levi
  Cambridge University Press (2006)
- [Shi-Dong-Liang]
  R.-S. Liang,
  Quantum Tunneling and Field Electron Emission Theories
  (World Scientific, Singapore, 2014)
  p. 61
- [Sze Kwok]
  S. M. Sze, Kwok K. Ng
  Physics of Semiconductor Devices
  John Wiley & Sons, Inc., New York (2007)
  DOI 10.1002/0470068329
- [MitinKochelapStroscio1999]
  Quantum Heterostructures (Microelectronics and Optoelectronics)
  V. Mitin, V. Kochelap, M. Stroscio
  Cambridge University Press (1999)
- [PiprekE2003]
  Semiconductor Optoelectronic Devices: Introduction to Physics and Simulation
  J. Piprek
  Elsevier (2003)

CHAPTER

FIFTEEN

JOURNAL PAPERS

- [AltermattI2006]
  A simulation model for the density of states and for incomplete ionization in crystalline silicon. I. Establishing the model in Si:P
  P. P. Altermatt, A. Schenk, G. Heiser
  J. Appl. Phys. **100**, 1113714 (2006)
- [AltermattII2006]
  A simulation model for the density of states and for incomplete ionization in crystalline silicon. II. Investigation of Si:As and Si:B and usage in device simulation
  P. P. Altermatt, A. Schenk, B. Schmitthüsen, G. Heiser
  J. Appl. Phys. **100**, 1113715 (2006)
- [Andrews2008]
  Doping dependence of LO-phonon depletion scheme THz quantum-cascade lasers
  A. M. Andrews, A. Benz, C. Deutsch, G. Fasching, K. Unterrainer, P. Klang, W. Schrenk, G. Strasser
  Materials Science and Engineering B **147**, 152 (2008)
- [Andreev2000]
  Theory of the electronic structure of GaN/AlN hexagonal quantum dots
  A. D. Andreev, E. P. O'Reilly
  Phys. Rev. B **62**, 15851 (2000)
- [Arora1982]
  Electron and hole mobilities in silicon as a function of concentration and temperature
  N.D. Arora, J.R. Hauser, D.J. Roulston
  IEEE Trans. Electron Devices **29**, 292 (1982)
- [BahderPRB1990]
  Eight-band kp model of strained zinc blende crystals
  Thomas B. Bahder
  Phys. Rev. B **41**, 11992 (1990)
- [Bai2010]
  Quantum cascade lasers that emit more light than heat
  Y. Bai, S. Slivken, S. Kuboya, S. R. Darvish, M. Razeghi
  Nature Photonics **4**, 99-102 (2010)
- [BastardPRB1982]
  Exciton binding energy in quantum wells
  G. Bastard, E. E. Mendez, L. L. Chang, and L. Esaki
  Physical Review B **26**, 1974 (1982)
- [Beynon1988]
  A Macintosh Hypercard stack for calculation of thermodynamically-corrected buffer recipes
  R. J. Beynon

Comput. Appl. Biosci. **4** (4), 487 (1988)

- [BirnerPhotonikInt2008]
  Simulation of quantum cascade lasers - optimizing laser performance
  S. Birner, T. Kubis, and P. Vogl
  Photonik international **2**, 60 (2008)
- [BirnerPhotonik2008]
  Simulation zur Optimierung von Quantenkaskadenlasern
  S. Birner, T. Kubis, and P. Vogl
  Photonik **1**, 44 (2008)
- [BirnerAPhys2006]
  Modeling of Semiconductor Nanostructures with nextnano3
  S. Birner, S. Hackenbuchner, M. Sabathil, G. Zandler, J.A. Majewski, T. Andlauer, T. Zibold, R. Morschl, A. Trellakis, P. Vogl
  Acta Physica Polonica A **110** (2), 111 (2006)
- [BirnerCBR2009]
  Ballistic Quantum Transport using the Contact Block Reduction (CBR) Method - An introduction
  S. Birner, C. Schindler, P. Greck, M. Sabathil, and P. Vogl
  Journal of Computational Electronics **8**, 267–286 (2009)
- [Biscani2020]
  A parallel global multiobjective framework for optimization: pagmo
  F. Biscani, D. Izzo
  Journal of Open Source Software **5** (53), 2338 (2020)
- [BelyakovBurdov2008]
  Anomalous splitting of the hole states in silicon quantum dots with shallow acceptors
  V. A. Belyakov, V. A. Burdov
  J. Phys. Condens. Matter **20**, 025213 (2008)
- [Burdov2002]
  Electron and hole spectra of silicon quantum dots
  V. A. Burdov
  JETP **94**, 411 (2002)
- [Canali1975]
  Electron and Hole Drift Velocity Measurements in Silicon and Their Empirical Relation to Electric Field and Temperature
  C. Canali, G. Majni, R. Minder, and G. Otaviani
  IEEE Trans. Electron Dev. **22** (11), 1045 (1975)
- [CapassoIEEE1986]
  Resonant Tunneling Through Double Barriers, Perpendicular Quantum Transport Phenomena in Superlattices, and Their Device Applications
  F. Capasso, K. Mohammed, and A. Y. Cho
  IEEE Journal of Quantum Electronics **QE-22**, 1853 (1986)
- [CardonaPR1966]
  Energy-band structure of Germanium and Silicon: The k.p method
  M. Cardona, F. H. Pollak
  Physical Review **142** (2), 530 (1996)
- [CarloSST2003]
  Microscopic theory of nanostructured semiconductor devices: beyond the envelope=function approximation
  A. D. Carlo

Semiconductor Science and Technology **18** (1), R1-31

- [CaugheyThomas1967]
  Carrier mobilities in silicon empirically related to doping and field
  D. Caughey, R. Thomas
  Proc. IEEE **55**, 2192 (1967)
- [Chatzikyriakou_PhysRevResearch_2022]
  Unveiling the charge distribution of a GaAs-based nanoelectronic device: A large experimental dataset approach
  E. Chatzikyriakou, J. Wang, L. Mazzella, A. Lacerda-Santos, M. C. da S. Figueira, A. Trellakis, S. Birner, T. Grange, C. Bäuerle, and X. Waintal
  Phys. Rev. Research **4**, 043163, December (2022)
- [Chilleri2021]
  An improved empirical model for a semiconductor's velocity-field characteristic applied to gallium arsenide
  J. Chilleri, Y. Wang, M. S. Shur, and S. K. O'Leary
  Solid State Communications **330**, 114240, (2021)
- [Chuang1996]
  k.p method for strained wurtizte semiconductors
  S. L. Chuang and C. S. Chang
  Physical Review B **54** (4), 2491, January (1996)
- [ChuangIEEE1996]
  Optical gain of strained wurtzite GaN quantum-well lasers
  S. L. Chuang
  IEEE Journal of Quantum Electronics **32**, no. 10, pp. 1791-1800, October (1996)
- [DarwishIEEE1997]
  An Improved Electron and Hole Mobility Model for General Purpose Device Simulation
  M. N. Darwish, J. L. Lentz, M. R. Pinto, P. M. Zeitzoff, T. J. Krutsick, and H. H. Vuong
  IEEE Transactions on Electron Devices **44**, 1529 (1997)
- [DattaSuperlatticesMicrostructures2000]
  Nanoscale device modeling: the Green's function method
  S. Datta
  Superlattices and Microstructures **28**, 253 (2000)
- [Dehlinger2000]
  Intersubband Electroluminescence from Silicon-Based Quantum Cascade Structures
  G. Dehlinger, L. Diehl, U. Gennser, H. Sigg, J. Faist, K. Ensslin, D. Grützmacher, E. Müller
  Science **290**, 2277 (2000)
- [Duboz2019]
  Theoretical estimation of tunnel currents in hetero-junctions: The special case of nitride tunnel junctions
  J-Y. Duboz and B. Vinter
  J. Appl. Phys. **126**, 174501 (2019)
- [DumitrasPRB2002]
  Surface photovoltage studies of InGaAs and InGaAsN quantum well structures
  Gh. Dumitras and H. Riechert
  Physical Review B **66**, 205324 (2002)
- [Eastman1980]
  DESIGN CRITERIA FOR GaAs MESFETs RELATED TO STATIONARY HIGH FIELD DOMAINS
  L. F. Eastman, S. Tiwiari, and M. S. Shur
  Solid-State Electronics **23** (4), 383 (1980)

- [Edlbauer2022]
  Semiconductor-based electron flying qubits: review on recent progress accelerated by numerical modelling
  Edlbauer, H., Wang, J., Crozes, T. et al.
  EPJ Quantum Technol. **9**, 21 (2022)
- [Farahmand2001]
  Monte Carlo Simulation of Electron Transport in the III-Nitride Wurtzite Phase Materials System: Binaries and Ternaries
  M. Farahmand, C. Garetto, E. Bellotti, K. F. Brennan, M. Goano, E. Ghillino, G. Ghione, J. D. Albrecht, and P. P. Ruden
  IEEE Trans. Electron Dev. **48** (3), 535 (2001)
- [FathololoumiOE2012]
  Terahertz quantum cascade lasers operating up to ~200 K with optimized oscillator strength and improved injection tunneling
  S. Fathololoumi, E. Dupont, C.W.I. Chan, Z.R. Wasilewski, S.R. Laframboise, D. Ban, A. Mátyás, C. Jirauschek, Q. Hu, and H. C. Liu
  Optics Express **20**, 3866 (2012)
- [FerreiraBastard1989]
  Evaluation of some scattering times for electrons in unbiased and biased single- and multiple-quantum-well structures
  R. Ferreira, G. Bastard
  Physical Review B **40** (2), 1074 (1989)
- [Ferreira2006]
  Optical Properties of Ellipsoidal CdSe Quantum Dots
  W.S. Ferreira, J.S. de Sousa, J.A.K. Freire, G.A. Farias, V.N. Freire
  Brazilian Journal of Physics **36**, 438 (2006)
- [Fischer2006]
  Tunnel-coupled one-dimensional electron systems with large subband separations
  S. F. Fischer, G. Apetrii, U. Kunze, D. Schuh, and G. Abstreiter
  Phys. Rev. B **74**, 115324
- [FischettiJAP2003]
  Six-band k”p calculation of the hole mobility in silicon inversion layers: Dependence on surface orientation, strain, and silicon thickness
  M. Fischetti, M. Solomon, M. Yang, and K. Rim
  J. Appl. Phys. **94**, 1079 (2003)
- [Fischetti1998]
  Theory of electron transport in small semiconductor devices using the Pauli master equation
  M. V. Fischetti
  J. Appl. Phys. **83**, 270-291 (1998)
- [FowlerNordheim1928]
  Electron emission in intense electric fields
  R. H. Fowler and L. Nordheim
  Royal Society **119** (781), 173 (1928)
- [FranceschiJancuBeltram1999]
  Boundary conditions in multiband k.p models: A tight-binding test
  S. De Franceschi, J.-M. Jancu, F. Beltram
  Physical Review B **59** (15), 9691 (1999)
- [Friedrich2005]
  Quantum-cascade lasers without injector regions operating above room temperature

A. Friedrich, G. Böhm, M.C. Amann, G. Scarpa
Applied Physics Letters **86**, 161114 (2005)

- [GovernalePRB1998]
  Gauge-invariant grid discretization of the Schrödinger equation
  M. Governale, C. Ungarelli
  Phys. Rev. B **58** (12), 7816 (1998)
- [GreckOE2015]
  Efficient Method for the Calculation of Dissipative Quantum Transport in Quantum Cascade Lasers
  P. Greck, S. Birner, B. Huber, and P. Vogl
  Optics Express **23**, 6587–6600 (2015)
- [GreckIWCE2010]
  The nonequilibrium Green's functions method and descendants: ways to avoid and to go
  P. Greck, C. Schindler, T. Kubis, and P. Vogl
  Abstracts of 14th International Workshop on Computational Electronics (IWCE), Pisa, Italy, 145 (2010)
- [GreinAIP1995]
  Long wavelength InAs/InGaSb infrared detectors: Optimization of carrier lifetimes
  C. H. Grein, P. M. Young
  J. Appl. Phys. **78**, 7143 (1995)
- [GunapalaJAP1991]
  Bound to continuum superlattice miniband long wavelength GaAs/AlxGa1-xAs photoconductors
  S. D. Gunapala, B. F. Levine, and Naresh Chand
  J. Appl. Phys. **70**, 305-308 (1991)
- [HalvorsenPR2000]
  Optical transitions in broken gap heterostructures
  E. Halvorsen, Y. Galperin, K. A. Chao
  Physical Review B **61**, 16743 (2000)
- [Hänsch1986]
  The hotelectron problem in small semiconductor devices
  W. Hänch and M. Miura-Mattausch
  J. Appl. Phys **60**, 650 (1986)
- [HirayamaJAP2005]
  Quaternary InAlGaN-based high-efficiency ultraviolet light-emitting diodes
  H. Hirayama
  J. Appl. Phys **97**, 091101 (2005)
- [Hu2005]
  Resonant-phonon-assisted THz quantum-cascade lasers with metal-metal waveguides
  Q. Hu, B.S. Williams, S. Kumar, H. Callebaut, S. Kohen, J.L. Reno
  Semiconductor Science and Technology **20**, S228 (2005)
- [Holleitner2007]
  Dimensionally constrained D'yakonov-Perel' spin relaxation in n-InGaAs channels: transition from 2D to 1D
  A.W. Holleitner, V. Sih, R.C. Myers, A.C. Gossard, D.D. Awschalom
  New Journal of Physics **9**, 342 (2007)
- [JancuPRB1998]
  Empirical spds* tight-binding calculation for cubic semiconductors: General method and material parameters
  J.-M. Jancu, R. Scholz, F. Beltram, F. Bassani

Physical Review B **57**, 6493 (1998)

- [Jiang1988]

  Temperature dependence of photoluminescence from GaAs single and multiple quantumwell heterostructures grown by molecularbeam epitaxy

  D. S. Jiang, H. Jung, K. Ploog

  J. Appl. Phys. **64**, 1371 (1988)

- [Jirauschek2014]

  Modeling techniques for quantum cascade lasers

  C. Jirauschek, T. Kubis

  Appl. Phys. Rev. **1**, 011307 (2014)

- [Jogai2003]

  Influence of surface states on the two-dimensional electron gas in AlGaN/GaN heterojunction field-effect transistors

  Jogai2003

  J. Appl. Phys. **93**, 1631 (2003)

- [Jeon1996]

  Valence band parameters of wurtzite materials

  J.-B. Jeon, Yu.M. Sirenko, K.W. Kim, M.A. Littlejohn, M.A. Stroscio

  Solid State Communications **99**, 423 (1996)

- [Kilpstein2010]

  Operator ordering and interface-band mixing in the Kane-like Hamiltonian of lattice-matched semiconductor superlattices with abrupt interfaces

  P. C. Kilpstein

  Phys. Rev. B 81, 235314 (2010)

- [Klaassen1992]

  A unified mobility model for device simulation - I. Model equations and concentration dependence

  D. B. M. Klaassen

  Solid-State Electronics **35**, 953 (1992)

- [KlimeckSM2000]

  sp3s* Tight-binding parameters for transport simulations in compound semiconductors

  G. Klimeck, R. C. Bowen, T. B. Boykin, T. A. Cwik

  Superlattices and Microstructures **27** (5), 519-524 (2000)

- [KnoetigLaserPhotRev2022]

  Mitigating valence intersubband absorption in interband cascade lasers

  H. Knoetig, J. Nauschuetz, N. Opacak, S. Hoefling, J. Koeth, R. Weih, B. Schwarz

  Laser & Photonics Reviews **16**, 2200156 (2022)

- [Kriekouki2022]

  Interpretation of 28 nm FD-SOI quantum dot transport data taken at 1.4 K using 3D quantum TCAD simulations

  I. Kriekouki, F. Beaudoin, P. Philippopoulos, C. Zhou, J. Camirand-Lemyre, S. Rochette, S. Mir, M.J. Barragan, M- Pioro-Ladriere, P. Galy

  Solid-State Electronics **194** (1), 108355 (2022)

- [KouwenhovenRPP2001]

  Few-electron quantum dots

  L.P. Kouwenhoven, D.G. Austing, and S. Tarucha

  Rep. Prog. Phys. **64**, 701 (2001)

- [KozodoyAPL1999]

  Enhanced Mg doping efficiency in Al0.2Ga0.8N/GaN

P. Kozodoy, M. Hansen, S. P. DenBaars, U. K. Mishra
Appl. Phys. Lett. **74**, 3681 (1999)

- [KubisNEGF2005]
  Self-consistent quantum transport theory of carrier capture in heterostructures
  T. Kubis, A. Trellakis, and P. Vogl
  Proceedings of the 14th International Conference on Nonequilibrium Carrier Dynamics in Semiconductors, M. Saraniti and U. Ravaioli, eds., Chicago, USA, July 25-19, 2005, Springer Proceedings in Physics, Vol. **110**, 369–372 (2005)
- [KumagaiChuangAndoPRB1998]
  Analytical solutions of the block-diagonalized Hamiltonian for strained wurtzite semiconductors
  M. Kumagai, S. L. Chuang, H. Ando
  Phys. Rev. B **57** 15303 (1998)
- [KumarHuPRB2009]
  Coherence of resonant-tunneling transport in terahertz quantum-cascade lasers
  S. Kumar, Q. Hu
  Phys. Rev. B **80**, 245316 (2009)
- [KuoNature2005]
  Strong quantum-confined Stark effect in germanium quantum-well structures on silicon
  Y.-H. Kuo, Y. K. Lee, Y. Ge, S. Ren, J. E. Roth, T. I. Kamins, D. A. B. Miller and J. S. Harris1
  Nature **437**, 7063 (2005)
- [Lazarenkova2001]
  Miniband formation in a quantum dot crystal
  O. L. Lazarenkova and A. A. Balandin
  J. Appl. Phys. **89**, 5509 (2001)
- [LenzlingerSnow1969]
  Fowler-Nordheim Tunneling into Thermally Grown SiO_2
  M. Lenzlinger and E. H. Snow
  J. Appl. Phys. **40**, 278 (1969)
- [LeverJLT2010]
  Design of Ge–SiGe Quantum-Confined Stark Effect Electroabsorption Heterostructures for CMOS Compatible Photonics
  L. Lever, Z. Ikonic, A. Valavanis, J. D. Cooper, and R. W. Kelsall
  Journal of Lightwave Technology **28**, 3273 (2010)
- [_LivnehPRB2014]
  Erratum: k.p model for the energy dispersions and absorption spectra of InAs/GaSb type-II superlattices [Phys. Rev. B 86, 235311 (2012)]
  Y. Livneh, P. C. Klipstein, O. Klin, N. Snapi, S. Grossman, A. Glozman, and E. Weiss
  Physical Review B **86**, 235311 (2012)
- [_LivnehPRB2012]
  k.p model for the energy dispersions and absorption spectra of InAs/GaSb type-II superlattices
  Y. Livneh, P. C. Klipstein, O. Klin, N. Snapi, S. Grossman, A. Glozman, and E. Weiss
  Physical Review B **86**, 235311 (2012)
- [LombardiIEEE1988]
  A physically based mobility model for numerical simulation of nonplanar devices
  C. Lombardi, S. Manzini, A. Saporito, and M. Vanzi
  IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **7**, 1164 (1988)
- [MamaluyCBR2003]
  Efficient method for the calculation of ballistic quantum transport

D. Mamaluy, M. Sabathil, and P. Vogl
J. Appl. Phys. **93**, 4628 (2003)

- [MasettiIEEE1983]
  Modeling of Carrier Mobility Against Carrier Concentration in Arsenic-, Phosphorus- and Boron-Doped Silicon
  G. Masetti, M. Severi, S. Solmi
  IEEE Trans. Electron Devices, Vol. ED-30 (7), 764 (1983)
- [MeyerMDPIPhot2020]
  The Interband Cascade Laser
  J. R. Meyer, W. W. Bewley, C. L. Canedy, C. S. Kim, M. Kim, C. D. Merritt, I. Vurgaftman
  Phot. 7(3), 75 (2020)
- [Mourokh2007]
  Vertically coupled quantum wires in a longitudinal magnetic field
  L. G. Mourokh, A. Y. Smirnov, S. F. Fischer
  Appl. Phys. Lett. **90**, 132108 (2007)
- [Page2001]
  300 K operation of a GaAs-based quantum-cascade laser at lambda=9 µm
  H. Page, C. Becker, A. Robertson, G. Glastre, V. Ortiz, C. Sirtori
  Appl. Phys. Lett. **78** (22), 3529 (2001)
- [ParkChuangPRB1999]
  Crystal-orientation effects on the piezoelectric field and electronic properties of strained wurtzite semiconductors
  S.-H. Park, S.-L. Chuang
  Phys. Rev. B **59**, 4725 (1999)
- [ParkChuang2000]
  Comparison of zinc-blende and wurtzite GaN semiconductors with spontaneous polarization and piezoelectric field effects
  S.-H. Park, S.L. Chuang
  J. Appl. Phys. **87**, 353 (2000)
- [Park2000]
  Crystal Orientation Effects on Electronic Properties of Wurtzite GaN/AlGaN Quantum Wells with Spontaneous and Piezoelectric Polarization
  S.-H. Park
  J. Appl. Phys. **39**, 3478 (2000)
- [Povolotskyi2005]
  Strain effects in freestanding three-dimensional nitride nanostructures
  M. Povolotskyi, M. Auf der Maur, A. Di Carlo
  Phys. Stat. Sol. (c) **2**, 3891 (2005)
- [Pryor1998]
  Eight-band calculations of strained InAs/GaAs quantum dots compared with one-, four-, and six-band approximations
  C. Pryor
  Physical Review B **57** (12), 7190 (1998)
- [ReichPR2002]
  Tight-binding description of graphene
  S. Reich, J. Maultzsch, C. Thomsen, P. Ordejon
  Physical Review B **66**, 035412 (2002)
- [RideauPRB2006]

Strained Si, Ge, and Si(1-x)Ge(x) alloys modelled with a first-principles-optimized full-zone k.p method
D. Rideau, M. Feraille, L. Ciampolini, M. Minondo, C. Tavernier, H. Jaouen, A. Ghetti
Physical Review B **74** (19), 195208 (2006)

- [Rochat2005]
  Low-threshold terahertz quantum-cascade lasers
  M. Rochat, L. Ajili, H. Willenberg, J. Faist, H. Beere, G. Davies, E. Linfield, D. Ritchie
  Applied Physics Letters **81** (8), 1381 (2002)
- [Sabathil2002]
  Towards fully quantum mechanical 3D device simulation
  M. Sabathil, S. Hackenbuchner, J. A. Majewski, G. Zandler, P. Vogl
  Journal of Computational Electronics **1**, 81 (2002)
- [SaitoS2001]
  Optical Properties and Raman Spectroscopy of Carbon Nanotubes
  M.S. Dresselhaus, G. Dresselhaus, P. Avouris (Eds.)
  Topics in Applied Physics **80**, Springer (2001)
- [Sarma2002]
  Realistic tight-binding model for the electronic structure of II-VI semiconductors
  S. Sapra, N. Shanthi, D. D. Sarma
  Physical Review B **66**, 205202 (2002)
- [SawamuraOME2018]
  Nearest-neighbor sp3d5s* tight-binding parameters based on the hybrid quasi-particle self-consistent GW method verified by modeling of type-II superlattices
  A. Sawamura, J. Otsuka, T. Kato, T. Kotani, S. Souma
  Optical Materials Express **8**, 1569 (2018)
- [Scholze2000]
  Single-Electron Device Simulation
  A. Scholze, A. Schenk, W. Fichtner
  IEEE Transactions on Electron Devices **47**, 1811 (2000)
- [Schwierz2010]
  Reversed Anionic Hofmeister Series: The Interplay of Surface Charge and Surface Polarity
  N. Schwierz, D. Horinek, R. R. Netz
  Langmuir **26**, 7370 (2010)
- [SirtoriPRB1994]
  Nonparabolicity and a sum rule associated with bound-to-bound and bound-to-continuum intersubband transitions in quantum wells
  C. Sirtori, F. Capasso, J. Faist, and S. Scandolo
  Physical Review B **50**, 8663 (1994)
- [Sirtori1998]
  GaAs/AlGaAs quantum cascade lasers
  C. Sirtori, P. Kruck, S. Barbieri, P. Collot, J. Nagle, M. Beck, J. Faist, U. Oesterle
  Applied Physics Letters **73** (24), 3486 (1998)
- [SchäfflerSST1997]
  High-Mobility Si and Ge structures
  F. Schäffler
  Semiconductor Science and Technology **12**, 1515 (1997)
- [SchubertAPL1996]
  Enhancement of deep acceptor activation in semiconductors by superlattice doping

E. F. Schubert, W. Grieshaber, I. D. Goepfert
Appl. Phys. Lett. **69**, 3737 (1996)

- [TsuEsaki1973]
  Tunneling in a finite superlattice
  R. Tsu and L. Esaki
  Appl. Phys. Lett. **22**, 562 (1973)
- [VurgaftmanJAP2001]
  Band parameters for III-V compound semiconductors and their alloys
  I. Vurgaftman and J. R. Meyer
  J. Appl. Phys. **89**, 5815 (2001)
- [VurgaftmanJAP2003]
  Band parameters for nitrogen-containing semiconductors
  I. Vurgaftman and J. R. Meyer
  J. Appl. Phys. **94**, 3675 (2003)
- [VurgaftmanNatCommun2011]
  Rebalancing of internally generated carriers for mid-infrared interband cascade lasers with very low power consumption
  I. Vurgaftman, W.W. Bewley, C.L. Canedy, C.S. Kim, M. Kim, C.D. Merritt, J. Abell, J.R. Lindle, J.R. Meyer
  Nature Communications **2**, 585 (2011)
- [VoglJPCS1983]
  A Semi-Empricial Tight-Binding Theory of the Electronic Structure of Semiconductors
  P. Vogl, H. P. Hjalmarson, and J. D. Dow
  Journal of Physics and Chemistry of Solids **44** (5), 365 (1983)
- [WackerAPL2005]
  Self-consistent theory of the gain linewidth for quantum-cascade lasers
  F. Banit, S.-C. Lee, A. Knorr, A. Wacker
  Appl. Phys. Lett. **86**, 041108 (2005)
- [WackerPRB2002]
  Gain in quantum cascade lasers and superlattices: A quantum transport theory
  A. Wacker
  Phys. Rev. B **66**, 085326 (2002)
- [Wang2004]
  Quantum mechanical calculation of hole mobility in silicon inversion layers under arbitrary stress
  E. Wang, P. Matagne, L. Shifren, B. Obradovic, R. Kotlyar, S. Cea, J. He, Z. Ma, R. Nagisetty, S. Tyagi, M. Stettler, M.D. Giles
  IEDM Technical Digest. IEEE International Electron Devices Meeting (2004)
- [ZakharovaPR2001]
  Hybridization of electron, light-hole, and heavy-hole states in InAs/GaSb quantum wells
  A. Zakharova, S. T. Yen, K. A. Chao
  Physical Review B **64**, 235332 (2001)
- [ZhangXia2006]
  Optical properties of GaN wurtzite quantum wires
  X. W. Zhang, J. B. Xia
  J. Phys.: Condens. Matter **18**, 3107 (2006)
- [Zhao2021]
  High Performance p-i-n Photodetectors on Ge-on-Insulator Platform

X. Zhao, G. Wang , H. Lin, Y. Du, X. Luo, Z. Kong, J. Su, J. Li , W. Xiong. Y. Miao, H. Li, G. Guo, H. H. Radamson

Nanomaterials **11**, 1125 (2021)

CHAPTER

SIXTEEN

THESES

- [AndlauerPhD2009]
  Optoelectronic and spin-related properties of semiconductor nanostructures in magnetic fields
  T. Andlauer
  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann, and P. Vogl, eds.), Vol. **105**, Verein zur Förderung des Walter Schottky Instituts der Technischen Universität München e.V., München, 157 pp. (2009)
- [BirnerPhD2011]
  Modeling of semiconductor nanostructures and semiconductor-electrolyte interfaces
  S. Birner
  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann, and P. Vogl, eds.), Vol. **135**, Verein zur Förderung des Walter Schottky Instituts der Technischen Universität München e.V., München, 239 pp. (2011)
  ISBN 978-3-941650-35-0
- [Eissfeller2008]
  Linear Optical Response of Semiconductor Nanodevices
  T. Eißfeller
  Diploma Thesis, Technische Universität München, Germany (2008)
- [GreckPhD2012]
  Efficient calculation of dissipative quantum transport properties in semiconductor nanostructures
  P. Greck
  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann, and P. Vogl, eds.), Vol. **153**,
  Verein zur Förderung des Walter Schottky Instituts der Technischen Universität München e.V., München, 149 pp. (2012)
  ISBN 978-3-941650-53-4
- [HackenbuchnerPhD2002]
  Elektronische Struktur von Halbleiter-Nanobauelementen im thermodynamischen Nichtgleichgewicht
  S. Hackenbuchner
  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann, and P. Vogl, eds.), Vol. **48**, Verein zur Förderung des Walter Schottky Instituts der Technischen Universität München e.V., München, 213 pp. (2002)
- [KubisPhD2009]
  Quantum transport in semiconductor nanostructures
  T. C. Kubis
  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann, and P. Vogl, eds.), Vol. **114**, Verein zur Förderung des Walter Schottky Instituts der Technischen Universität München e.V., München, 253 pp. (2009)
- [SabathilPhD2005]
  Opto-electronic and quantum transport properties of semiconductor nanostructures

M. Sabathil

Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann, and P. Vogl, eds.), Vol. **67**, Verein zur Förderung des Walter Schottky Instituts der Technischen Universität München e.V., München, 156 pp. (2005)

- [SchusterPhD2005]

  Hochortsaufgelöste optische Spektroskopie an niedrigdimensionalen Halbleiterstrukturen

  R. Schuster

  PhD Thesis, University of Regensburg, Germany, 2005

- [ZiboldPhD2007]

  Semiconductor based quantum information devices: Theory and simulations

  T. Zibold

  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann, and P. Vogl, eds.), Vol. **87**, Verein zur Förderung des Walter Schottky Instituts der Technischen Universität München e.V., München, 151 pp. (2007)

CHAPTER
SEVENTEEN

PDF DOCUMENTATION

- version 1.25.13 from 2025-04-25

Last update: 2025/06/26