nextnano
Software for semiconductor nanodevices

nextnano Documentation

Release August 2024

nextnano GmbH

Aug 26, 2024

GETTING STARTED

CHAPTER

ONE

ABOUT THE NEXTNANO SOFTWARE

The nextnano software allows modelling electronic and optoelectronic semiconductor nanodevices. It is characterized by:

- High performance
- Short runtimes
- Python API
- 1D, 2D & 3D simulations
- Seamless workflow integration
- Material database included

Figure 1.1: Applications of the nextnano software

The nextnano software typically is used for research purposes, in the most cases for structures containing quantum wells. However, it can be also successfully used for other type of structures, such as quantum wires, quantum dots, more complicated quantum structures, structures based on p-n junction, 2-dimensional gases, and much more.

Software license includes free access to hundreds of maintained example files and tutorials featuring state-of-the-art physics and publications in semiconductor and quantum science. Easily adapt example files to your device geometry to get started without high initial training effort.

The nextnano software consist of multiple tools aiming at providing you with the best experience while modeling your structures or devices. The tools *nextnano++*, *nextnano*^{3}, *nextnano.MSB*, and *nextnano.NEGF* are our engines of the nextnano software solving sets of specified equations and computing desired results. These tools require input files specifying the structures, models and outputs of interest. The graphical user interface *nextnanomat* is your workflow manager whose purpose is to let you go through the entire process of modeling seamlessly. For those who are looking for automated simulation processes, we provide and maintain an open source *nextnanopy* Python package.

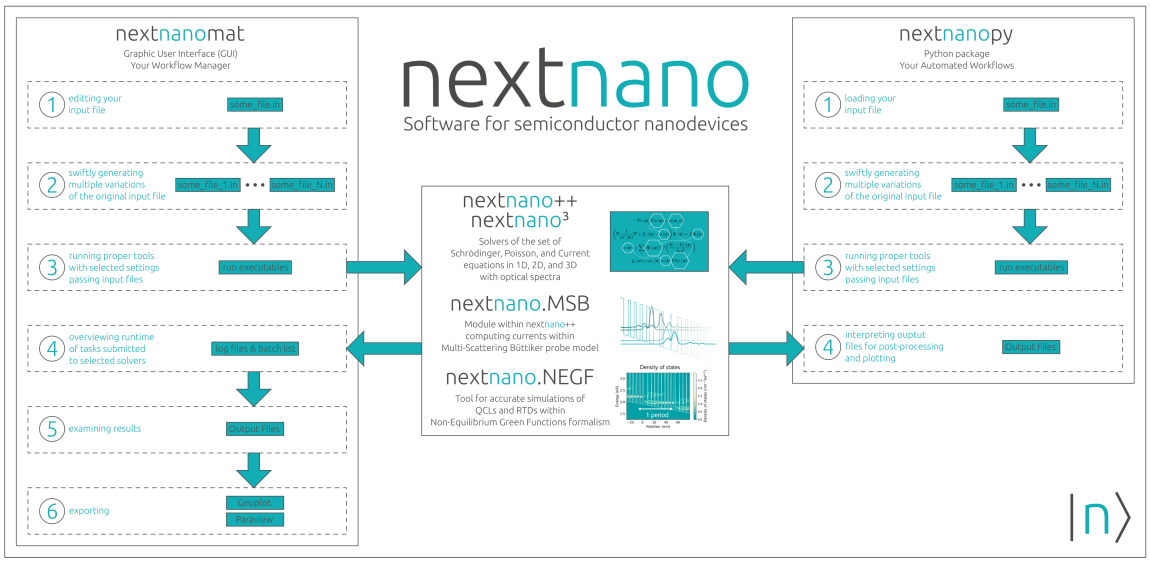


Figure 1.2: Structure and typical workflows of the nextnano software

CHAPTER

TWO

INSTALLATION

2.1 First Steps

Note: We are currently rewriting our documentation. Please note that this documentation is not complete yet but will be improved step by step.

nextnano GmbH - Software for **next** generation of **nano** devices

1. Register

   First, your have to register here.

   You will then receive an email with a link for the following password protected downloads, and instructions on how to activate your free 1-month evaluation license.

2. Download

   You can download the latest version of nextnano GmbH from our *Downloads* website.

   Our most recent zip files contain executables for Windows, Linux and macOS. Experienced users can use Wine to run *nextnanomat* on Linux. However, it does not work smoothly on Linux. Therefore, we strongly recommend Windows.

   Older versions are also listed on our Downloads website.

   a) We recommend downloading the zip file with the Installer.

   b) If you do not have Administrator rights on your computer, you can download and unzip the file without the installer. No installation is required. Just unzip the file and start `nextnanomat.exe`. The *nextnanomat* GUI is the graphical user interface while *nextnano++* and *nextnano*^{3} are the actual scientific software doing the calculations.

   You can typically install several versions of the nextnano software on the same computer. There is no need to uninstall previous versions if you want to keep them.

3. Start *nextnanomat*

   The *nextnanomat* (`nextnanomat.exe`) is the Front End & Workflow Manager that runs the *nextnano++* and *nextnano*^{3} tools.

4. License Activation

   The license activation procedure is documented here: *License Activation*

5. Number of licenses

   For university and government institutions, the license fee covers all group members of the research group purchasing the software.

6. Example input files (Tutorials)

The zip file contains many example input files.

We are happy to provide any further tutorial input files that you need. Input files for all tutorials are contained in the installation folder.

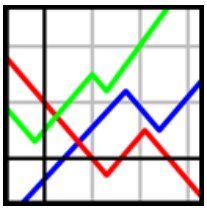
The complete lists of tutorials for each tool:

- *nextnano++*,
- *nextnano3*,
- *nextnano.NEGF*,
- *nextnano.MSB*.

If you can tell us what you want to simulate, we can point you to the most relevant examples.

7. Gnuplot

   We also recommend installing the free gnuplot software:

   Download Gnuplot

   Then you can generate plots from within our GUI *nextnanomat* conveniently.

8. First steps

   Start e.g. with this Laser Diode example. This input file solves the Schrödinger, Poisson and drift-diffusion current equation for a multi-quantum well laser diode.

   Detailed documentation: *InGaAs Multi-quantum well laser diode*

   The number of quantum wells can be adjusted in the input file using the variable `NUMBER_OF_WELLS`.

   This laser diode input file is included in the zip file of the latest update.

9. Facebook

   Follow us on Facebook to stay informed about the latest news of the nextnano software:

   facebook.com/nextnano

10. YouTube Videos

    We also have a YouTube Channel:

    There are two short videos that give you a first impression. If you are unsure whether to install the nextnano software, this video might be of help:

    - Tutorial - How to install the software
    - Tutorial - How to start a simulation

11. Python package

    We offer a Python package for post-processing the results. However, *nextnanopy* is currently suited for experienced Python and experienced nextnano GmbH users only.

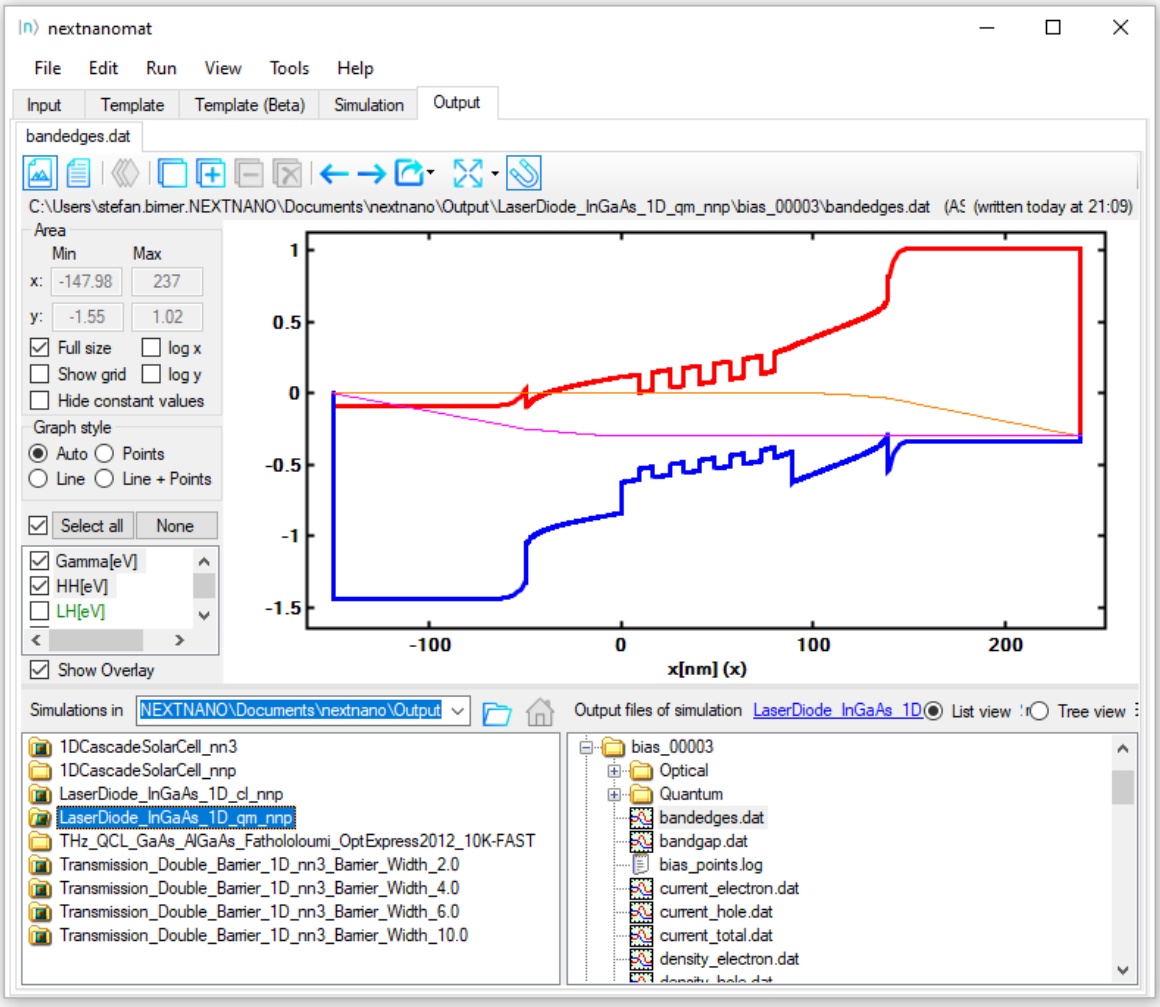


Figure 2.1.1: Conduction band edge, valence band edge, and quasi-Fermi levels for electrons and holes of a Laser Diode

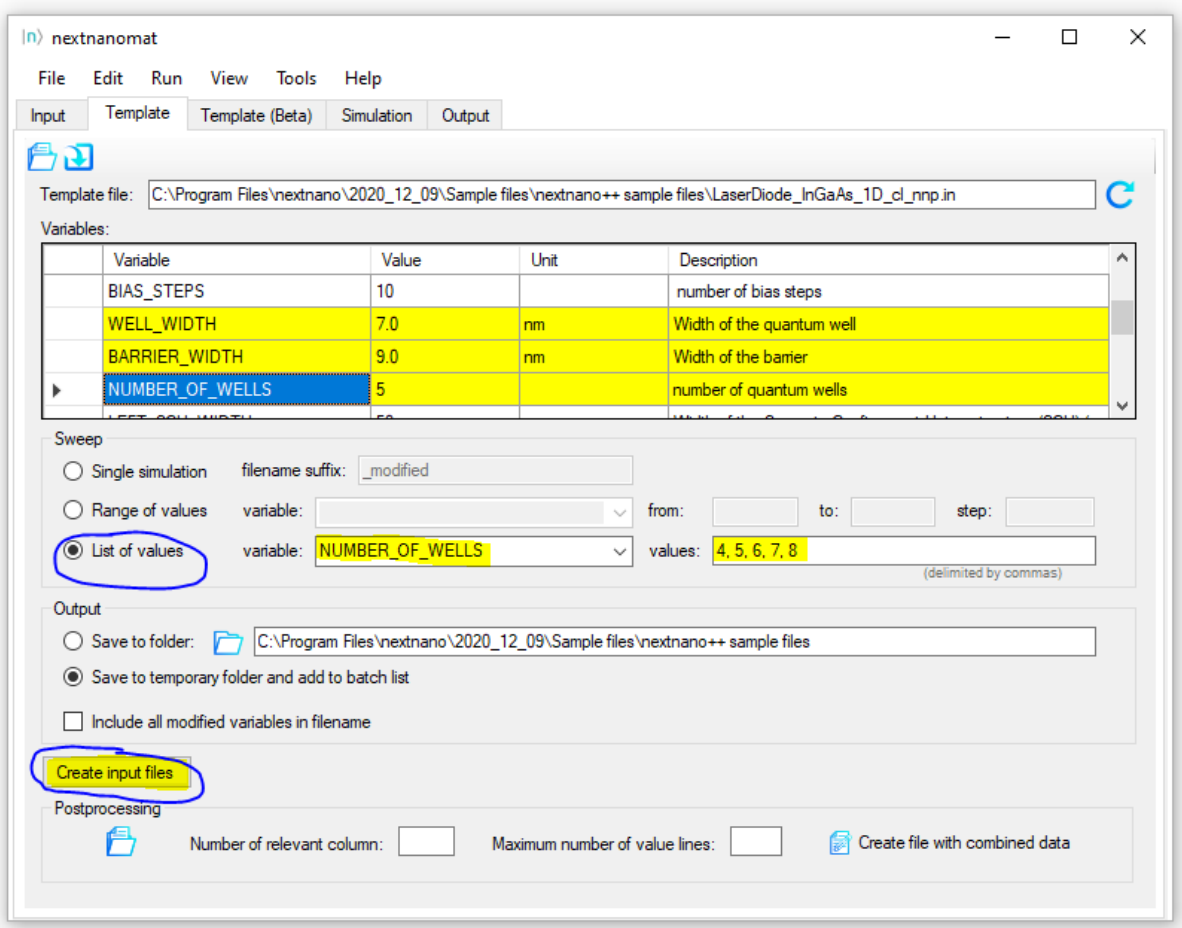


Figure 2.1.2: Template tab to define parameter sweeps. Here, the variable `NUMBER_OF_WELLS` is varied using the **List of values** 4, 5, 6, 7, 8. By clicking on **Create input files**, 5 input files are generated, each having a different number of quantum wells.

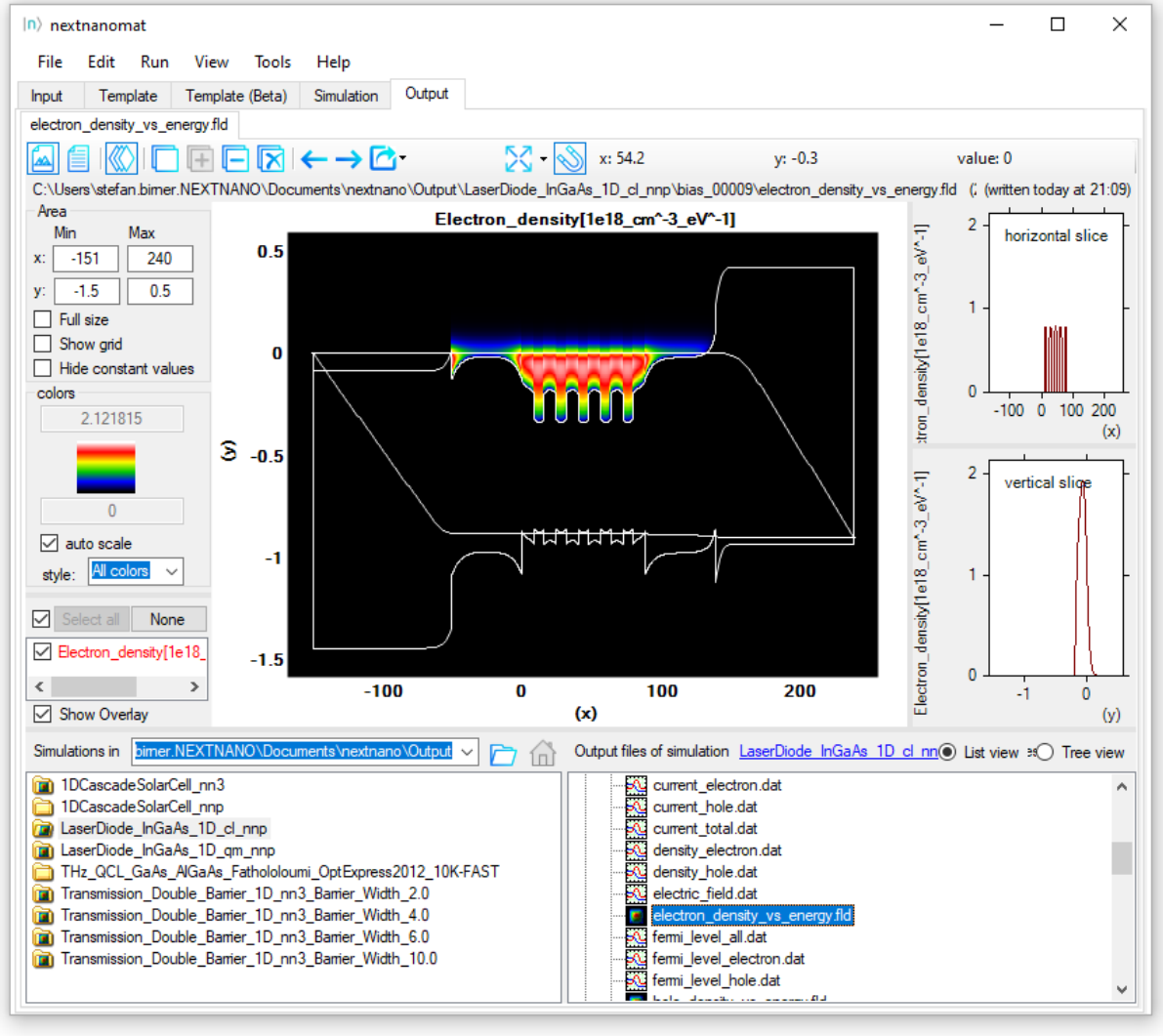


Figure 2.1.3: Energy and position resolved electron density $n(x, E)$ of the Laser Diode.

```
pip install nextnanopy
```

12. Feedback and Support

    Just let us know if you have any further questions, or if you have any feedback on our software. Please send support questions to *nextnano Help Center* or use our nextnano support Help Center. You will then receive a support ticket.

    The nextnano GmbH team

2.2 Operating system

2.2.1 Windows

The nextnano software has been developed for Windows.

2.2.2 Linux

There are three options:

A: Run simulations from Terminal (Linux native executable).

B: Run simulations with *nextnanopy* (Linux native executable).

C: Run simulations with *nextnanomat* using Wine or Mono (Windows executable).

We strongly recommend the option B.

Running nextnano++ & nextnano^3 on Linux from Terminal

The *nextnano++* and *nextnano^3* simulations can be executed from the **Linux terminal** by directly using the Linux executables. In order to make the programs executable, one needs to run the command **chmod a+x *.exe**, where (*) is replaced by the corresponding *nextnano++* or *nextnano^3* Linux executable.

For *nextnano^3* simulations, you need to set the environment variable nextnano GmbH by executing `$ export NEXTNANO="/<directory_name>/nextnano/<date>"`.

For *nextnano++* simulations using the 'Intel_Ubuntu' executable, the path to the OpenMP library has to be specified. We ship a `.so` file together with the executable. Go to the folder containing the executable and run:

```
currentFolder=$(pwd)
export LD_LIBRARY_PATH=$currentFolder
```

The terminal commands for two *nextnano++* and *nextnano^3* sample input files can then be given as follows, to be executed from the nextnano GmbH folder directory:

```
"nextnano3/Linux gcc 64bit/nextnano3_Linux_gcc_64bit.exe" -log -license "License/
↪License_nnp.lic" -outputdirectory "Output/<name_of_input_file>" -inputfile "Sample␣
↪files/nextnano3 sample files/<name_of_input_file>"
```

```
"nextnano++/bin 64bit/nextnano++_Linux_gcc_64bit.exe" --license "License/License_nnp.
↪lic" --outputdirectory "Output" -log "Sample files/nextnano++ sample files/Quantum␣
↪Mechanics examples/<name_of_input_file>"
```

Detail documentation on command line features can be found on the pages *Command Line* and *Command Line*. Further information and more examples can be found in the README_Linux file and the shell scripts included in the nextnano GmbH folder.

Running nextnano++ & nextnano^3 on Linux using nextnanopy

nextnanopy is our open-source Python package for running simulations, sweeping variables and post-processing results. Please follow the instructions from the link.

License activation

To run a Linux-native executable, two types of licenses are available:

1. license.txt (old licensing system, will be deprecated in the future)
2. License_nnp.lic (new licensing system, same as Windows executable)

Which license to use depends on the Linux executable you want to run.

To obtain 1, please contact us at *nextnano Help Center* to request issuing the license.

To obtain 2, see *License activation via command line*.

The nextnanomat GUI on Linux

We strongly recommend using |nextnanopy| on Linux since it runs Linux-native executables and provides methods to modify input files and interact with simulation output.

The *nextnanomat* GUI is programmed in C#, and can thus be executed on any operating system. It is, however, developed on and optimized for Windows. On Linux, you have to either install Wine or Mono and then execute *nextnanomat*.

Please note that we developed *nextnanomat* for Windows and not for Linux, therefore many things might not work as expected.

Status 2020-04-07: Recommended to use Wine as described below.

Status 2018-05-03: Recommended to use Wine as described below.

Option 1: Wine

The following instructions are for **Ubuntu version 20.10** (Groovy Gorilla). Commands for the Wine installation on various distributions are given in *Wine Installation*. A detailed description on how to install WineHQ on Ubuntu can be found here: Install WineHQ package on Ubuntu

The following commands are needed to install WineHQ on Ubuntu 20.10 (Groovy Gorilla). For older versions of Ubuntu, please see here: *Wine Installation*.

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo apt-add-repository 'deb https://dl.winehq.org/wine-builds/ubuntu/ groovy main'
sudo apt update
sudo apt install --install-recommends winehq-stable
```

Download the zip file of nextnano GmbH, i.e. the **portable version** from the *download site*.

- Unzip this file.
- Open a Terminal window and change to the directory where the `nextnanomat.exe` is located.
- Type: `wine nextnanomat.exe`
- There will be a pop-up message that a Mono package is missing which can be installed automatically. Click ''Install".
- There will be a pop-up message that a Gecko package is missing which can be installed automatically. Click ''Install".
- If *nextnanomat* does not work, close the terminal and open a new terminal and then try again.

The first thing you have to do when *nextnanomat* opens is to activate your license. Normally this operation fails when performed under Wine or Mono. Instead you should activate your licenses via command line *License activation via command line*. Afterwards, you need to update the paths within *Tools* ==> *Options* ==> *Licenses*. (You can cancel the activation dialog of nextnanomat)

Please note that we developed *nextnanomat* for Windows and not for Linux, therefore some things might not work as expected. Optionally, one can execute the *nextnano++* and *nextnano*[3] executables on Linux using Wine without nextnano GmbH. You have to type in: `...` (*Add an example here...*)

Known problems are:

- On very old CPUs, the 64-bit version of *nextnano++* does not work. In this case, please select the 32-bit version of *nextnano++*. Tools => Options => Simulation => *nextnano++* executable => `<path>\nextnano++\bin 32bit\nextnano++_Microsoft_32bit_serial.exe`
- Somehow the settings are not saved, i.e. you need to re-enter custom paths or settings at each program start.
- The program crashes.

Things that have to be fixed for future updates are:

- Use a monospaced font for the Input tab. (It seems that the default font for Windows is not found and then another default font (Tahoma) which is not monospaced is chosen.) One can choose a different font for the editor. Tools => Options => Editor => Editor font

---

The following instructions are for **Ubuntu version 18.04 LTS**.

Run commands in a terminal:

```
sudo apt update      # Update system
```

```
sudo apt install wine-stable winetricks # Install required packages (without "-stable
↪" in versions bevore Ubuntu 16.10)
```

```
winetricks dotnet45  # Install .net support, follow along on-screen instructions,␣
↪ignore warnings
```

```
winetricks corefonts # Install basic fonts of Windows
```

Finally run `winecfg` -> Select "Windows 8" in Applications -> Windows Version **and** reboot afterwards.

To start the application, use `wine /<your_directory_name>/nextnanomat.exe`

Example: ''wine '/home/ubuntu/nextnanoTestversion/nextnanomat.exe' ''

Option 2: Mono

(This documentation should be updated for the latest Ubuntu and Mono versions.) It is not clear if it still works!

```
mono nextnanomat.exe &
```

On some Linux distributions (e.g. Ubuntu), Mono is already preinstalled.

The current Mono version uses .NET Framework version 4.0 by default. The *nextnanomat* GUI, however, still requires the installation of the .NET Framework version 2.0. Therefore the following error occurs:

```
The assembly mscorlib.dll was not found or could not be loaded.
It should have been installed in the `/usr/lib/mono/2.0/mscorlib.dll' directory.
```

The following solution works:

```
sudo apt-get install libmono-corlib2.0-cil
sudo apt-get install libmono-winforms2.0-cil
sudo apt-get install mono-complete
```

All *nextnanomat* features seem to work on Linux Ubuntu. However, what does not work satisfactorily is the coloring of the 2D plots. They look too bright, so one can hardly recognize the results. A quick fix is to activate "Show grid". Then the figure looks better.

If you have any feedback on these instructions, please let us know, then we can keep our documentation up to date.

We can also provide a command line version of nextnano GmbH for the various Linux distributions on request.

You can contact us at *nextnano Help Center*.

Wine Installation

Below are the commands for installing Wine on various Ubuntu distributions.

Ubuntu 20.10 (Groovy Gorilla)

On Ubuntu 20.10, the following commands will install the stable branch of Wine:

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo add-apt-repository 'deb https://dl.winehq.org/wine-builds/ubuntu/ groovy main'
sudo apt update
sudo apt install --install-recommends winehq-stable
```

Ubuntu 20.04 (Focal Fossa)

On Ubuntu 20.04, the following commands will install the stable branch of Wine:

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo add-apt-repository 'deb https://dl.winehq.org/wine-builds/ubuntu/ focal main'
sudo apt update
sudo apt install --install-recommends winehq-stable
```

Ubuntu 18.04 (Bionic Beaver)

On Ubuntu 18.04, the following commands are instead used:

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo apt-add-repository 'deb https://download.opensuse.org/repositories/Emulators:/
↪Wine:/Debian/xUbuntu_18.04/ ./'
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys DFA175A75104960E
sudo apt update
sudo apt install --install-recommends winehq-stable
```

Debian 10

On Debian 10, the following commands will install Wine:

```
sudo dpkg --add-architecture i386
sudo apt update
sudo apt -y install gnupg2 software-properties-common
wget -qO - https://dl.winehq.org/wine-builds/winehq.key | sudo apt-key add -
sudo apt-add-repository https://dl.winehq.org/wine-builds/debian/
wget -O- -q https://download.opensuse.org/repositories/Emulators:/Wine:/Debian/Debian_
↪10/Release.key | sudo apt-key add -
echo "deb http://download.opensuse.org/repositories/Emulators:/Wine:/Debian/Debian_10␣
↪./" | sudo tee /etc/apt/sources.list.d/wine-obs.list
sudo apt update
sudo apt install --install-recommends winehq-stable
```

RHEL/CentOS 8

We have successfully installed Wine 5 on RHEL 8 using the following commands:

(These build Wine from source, hence the installation takes considerably longer.)

```
sudo -i
dnf clean all
dnf update
dnf groupinstall 'Development Tools'
dnf install libX11-devel freetype-devel zlib-devel libxcb-devel libxslt-devel␣
↪libgcrypt-devel libxml2-devel gnutls-devel libpng-devel libjpeg-turbo-devel libtiff-
↪devel gstreamer1-devel dbus-devel fontconfig-devel
cd /opt
wget https://dl.winehq.org/wine/source/5.0/wine-5.0.tar.xz
tar -Jxf wine-5.0.tar.xz
cd wine-5.0
##For 32-Bit Systems:
./configure
##For 64-Bit Systems:
./configure  --enable-win64
make
make install
```

Configuring Wine

After installing Wine, if you encounter errors about Mono, a useful trick is to delete the folder **.Wine** and force Wine to download and install Gecko and Mono automatically. Normally, Wine will suggest installing these the first time it is used. Furthermore, opening the `wine uninstaller` and removing interfering applications might solve some problems. In the end, we are able to use Wine without installing Winetricks or dotnet manually. If you are working on a remote machine, you might need to enable X11 forwarding to see the windows created during the Wine configuration. This is done by connecting via **ssh -X** and modifying the **ssh.config** file under **etc/ssh** accordingly.

2.2.3 macOS

- *Running nextnanomat*

There are three options:

A: Run simulations from Terminal (Mac native executable).

B: Run simulations with *nextnanopy* (Mac native executable).

C: Run simulations with *nextnanomat* using Wine or Mono (Windows executable).

We strongly recommend the option B.

Basic simulation is possible from (A) Terminal. (B) The *nextnanopy* Python package can not only run the simulations but also sweep variables and postprocess the results. For the options A and B, you need nextnano GmbH executables compiled for Mac, which we can provide you on request. Please feel free to contact us at *nextnano Help Center*. For the option C, Mono requires the Mac executables but Wine uses the ones compiled for Windows.

License activation

Attention: If you use ARM Mac for simulation, you have to "emulate Intel processor" before initiating the license activation. Use this code to switch:

```
$env/usr/bin/arch -x86_64 /bin/zsh ---login
```

When you enter `arch`, and the output shows `i386`, the computer is Intel.

To run a Mac-native executable, two types of licenses are available:

1. license.txt (old licensing system, will be deprecated in the future)
2. License_nnp.lic (new licensing system, same as Windows executable)

Which license to use depends on the Mac executable you want to run.

To obtain 1, please contact us via *nextnano Help Center* to request issuing the license.

To obtain 2, see *License activation via command line*.

Attention: If you use ARM Mac for simulation, as of 5th March 2024, you can run simulation with only **license.txt**. We are working on this issue.

Permission denied

You might be asked to allow your computer to execute some commands. To add execute permission to the executable, run:

```
chmod 744 nextnanoLicenseActivator_macOS
```

Option A: Run simulations from Terminal

Install Homebrew, gcc

You have to install Homebrew to install gcc. Go to \nextnano++\bin and run the command below:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
↪install.sh)"
```

Furthermore, run the following two codes as Homebrew requires:

```
(echo; echo 'eval "$(/usr/local/bin/brew shellenv)"') >> /Users/yuta/.zprofile
```

```
eval "$(/usr/local/bin/brew shellenv)"
```

Then finally, you can install gcc.

```
brew install gcc
```

Run simulation

The terminal commands for two *nextnano++* and *nextnano*^{3} sample input files can then be given as follows, to be executed from the nextnano GmbH folder:

```
./nextnano3_gcc_macOS -l "License/License_nnp.lic" "Sample files/nextnano3/examples/
↪1D_simple_GaAs_QW.in"
```

```
./nextnano++_gcc_macOS -l "License/License_nnp.lic" "Sample files/nextnano++/examples/
↪Quantum Mechanics examples/QW_finite_1D_nnp.in"
```

Attention: If you use ARM Mac for simulation, you have to use `license.txt` rather than `License_nnp.lic` as of 5th March 2024. The command line in this case will be:

```
./nextnano++_gcc_macOS --old -l "License/license.txt" "Sample files/nextnano++/
↪examples/Quantum Mechanics examples/QW_finite_1D_nnp.in"
```

Detailed documentation on command line features can be found in *Command Line* (*nextnano++*) and *Command Line* (*nextnano*^{3}).

Option B: Run simulations with nextnanopy

nextnanopy is our open-source Python package for running simulations, sweeping variables and post-processing results.

Install Python

You can install Python package Anaconda to establish a Python environment including NumPy, Matplotlib and an IDE called "Spyder". With this, you can use *nextnanopy* from a graphical user interface.

Here, we explain an alternative way to install a Python package from Terminal via Homebrew. We have tested this with ARM64 Mac with macOS 11.4 (Big Sur).

1. In a macOS Terminal, type in:

```
# install Command Line Tools, if not installed on your machine
xcode-select --install

# install Homebrew, if not installed (cf. Homebrew website)
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪HEAD/install.sh)"

# update Homebrew
brew update

# search for available Python packages
brew search python

# install Python3
brew install python3
```

2. Unversioned commands 'python', 'pip' etc. pointing to 'python3', 'pip3' etc., respectively, are installed into, e.g., /opt/homebrew/opt/python@3.9/libexec/bin. It is useful to set this path to ~./zprofile.

```
# open ~/.zprofile with a text editor and write
``eval "$(/opt/homebrew/bin/brew shellenv)"```
``export PATH=/opt/homebrew/opt/python@3.9/libexec/bin:$PATH``
# apply the changes
source ~/.zprofile

# make sure that the version 3.9 or later has been installed
python --version

# upgrade pip (NOT update)
pip install --upgrade pip
```

3. Using `pip`, please install NumPy and Matplotlib which are required for *nextnanopy*.

```
pip install numpy
pip install matplotlib
```

Install nextnanopy

You can either manually or automatically install *nextnanopy*. For more details, please refer to: *How do I install it?*

```
## manual installation
# go to a folder where you want to store local repository of |nextnanopy| project
cd <folder name>

# clone source code from Github
git clone https://github.com/nextnanopy/nextnanopy.git
```

(continues on next page)

(continued from previous page)

```
# build nextnanopy
cd nextnanopy/
python setup.py install
```

For the automatic installation, you can use `pip`:

```
pip install nextnanopy
```

Configure nextnanopy

Open the file `config_nextnano.py` with an text editor to adjust the paths to your license, output and executable installation folders:

```
open config_nextnano.py
# (adjust the paths)
# (save the file)
# run the config file to apply changes
python config_nextnano.py
```

Running nextnanopy

Please see *Basic Tutorials* and sample Python scripts to learn how to run a simulation with *nextnanopy*. The repository of *nextnanopy* includes sample Python scripts under `/nextnanopy/templates`.

Option C: Run simulations with nextnanomat using Wine or Mono

This option is for those who wish to use GUI *nextnanomat* to run simulations.

The *nextnanomat* GUI is programmed in C#, and can thus be executed on any operating system. It is, however, developed on and optimized for Windows. On macOS, you have to install either Wine or Mono to run *nextnanomat*.

Wine was available from Mountain Lion 10.8 until Mojave 10.14. We confirmed that using **Wine** one could run nextnanomat.exe on Mojave 10.14.

Wine did not work on Catalina 10.15 or later. However, Wine version 6.0.1 released on 7 Jun. 2021 is said to support wine64 on Apple M1. We will test once the built package becomes available.

The following is for macOS 10.14 Mojave.

Installation procedure

1. Install XQuartz (version 2.7.7 or later).
   - Please open Xquartz and check if it starts without errors.
2. Install Wine Stable for macOS.
   - On the website, both .pkg files and .tar.gz files are provided. Installation from .pkg files is handy. In this case, however, only the 32-bit version of the nextnano software can be used, and currently one has to make sure that appropriate libiomp5.dll files are located in the same directory as "nextnano3.exe" and "nextnano++.exe" (even for the serial version of nextnano GmbH). If you need libiomp5.dll files, please contact us.
   - If you install Tarball for "Wine Stable" (32 + 64-bit), the 64-bit version is also available.
3. Install winetricks.
   - Launch **Wine stable** from Applications or Launchpad.

- Terminal window shows up with a short introduction of important commands.

```
################################################################################
#                            Wine Is Not an Emulator                           #
################################################################################

 Welcome to wine-4.0.2.

 In order to start a program:
   .exe: wine program.exe
   .msi: wine msiexec /i program.msi

 If you want to configure wine:
   winecfg

 To get information about app compatibility:
   appdb Program Name
```

- **Run**

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
brew install winetricks
```

4. Install .NET Framework using winetricks. .NET Framework of version 4.5.2 or later is needed.

```
winetricks dotnet452

- Follow the instructions and ignore Warnings.
- Restart the computer.
```

1. Fonts and configuration
   - Launch **Wine stable** again.
   - **Run:**

```
winetricks corefonts   # install basic fonts of Windows
winecfg  # Configure Wine
```

   - Setting window shows up. Select “Windows 10” for Windows version in the “Application” tab.
   - Press “OK” button.

Running nextnanomat

1. Launch **Wine Stable** from Application or Launchpad.
2. Run:

```
wine /<your directory>/nextnanomat.exe
```

3. Activate the license with your email address (only once) and have fun!

If you have any feedback on these instructions, please let us know by sending an email to *nextnano Help Center*. It helps us keeping our documentation up to date.

2.3 Downloads

2.3.1 Free nextnano tools

We offer a number of our tools which can be run **without registering** and **without license**.

The nextnano++ package — free edition

Windows

beta (2024-08-26)

The Free *nextnano++* package is a limited version of the standard product (see the table in the product overview).

Included Tools	Version
nextnanomat GUI	*4.3.2.15*
nextnano++ (free edition)	*1.20.8*
nextnano^3 (free edition)	*2.6.4*

The nextnanomat GUI

See the table in the product overview for description of the tool.

Windows

beta (2024-03-28)

Included Tools	Version
nextnanomat	*4.3.2.15*

- Download: nextnanomat_beta_installer — recommended
- Download: nextnanomat_beta_portable

The nextnanopy package

The *nextnanopy* Python package is developed for *nextnano++*, *nextnano*^{3}, and *nextnano.NEGF* tools to automate simulations and analysis of the results.

More details about the package and instruction of installation can be found *here*.

2.3.2 Standard & Evaluation

This page lists links to the standard and evaluation packages of the nextnano software. To run them you need to have an active **license**, which can be obtained through our registration page.

The nextnano++ package

See the table in the product overview for description of the tools.

Windows

stable (2023-08-07)

Included Tools	Version
nextnanomat	*4.3.2.8*
nextnano++	*1.17.20*
nextnano[3]	*2.5.4*

- Download: nextnano_standard_stable_installer
- Download: nextnano_standard_stable_portable (includes executables for **Linux** and **macOS**)

Note: The file permissions have to be manually added to the scripts and executable files for **Linux** and **macOS** executables in the portable package.

beta (2024-08-23)

Included Tools	Version
nextnanomat	*4.3.2.15*
nextnano++	*1.20.8*
nextnano[3]	*2.6.4*

- Download: nextnano_standard_beta_installer — recommended
- Download: nextnano_standard_beta_portable

alpha (2024-07-01)

Included Tools	Version
nextnanomat	*4.3.2.15*
nextnano++	*1.19.61*
nextnano^3	*2.6.2*

- Download: nextnano_standard_alpha_installer
- Download: nextnano_standard_alpha_portable

Ubuntu

beta (2024-08-23)

Included Tools	Version
nextnano++	*1.20.8*
nextnano^3	*2.6.4*

- Download: nextnano_standard_beta_portable_Ubuntu

alpha (2024-07-01)

Included Tools	Version
nextnano++	*1.19.61*
nextnano^3	*2.6.2*

- Download: nextnano_standard_alpha_portable_Ubuntu

Linux Red Hat Enterprise Linux 8.1

beta (2024-08-23)

Included Tools	Version
nextnano++	*1.20.8*
nextnano^3	*2.6.4*

- Download: nextnano_standard_beta_portable_RHEL81

Linux Red Hat Enterprise Linux 7.6

Note: It is likely to work on CentOS, SUSE, and AlmaLinux

beta (2024-08-23)

Included Tools	Version
nextnano++	*1.20.8*
nextnano^3	*2.6.4*

- Download: nextnano_standard_beta_portable_RHEL76

---

macOS

beta (2024-08-23)

Included Tools	Version
nextnano++	*1.20.8*
nextnano^3	*2.6.4*

- Download: nextnano_standard_beta_portable_macOS

alpha (2024-05-03)

Included Tools	Version
nextnano++	*1.19.17*
nextnano^3	*2.6.2*

- Download: nextnano_standard_alpha_portable_macOS

The nextnano.NEGF package

See the table in the product overview for description of the tools.

Windows

beta (2024-08-23)

Included Tools	Version
nextnanomat	*4.3.2.15*
nextnano.NEGF	*2024-07-25*
nextnano.NEGF_classic	*2022-06-13*

- Download: nextnano_NEGF_beta_installer — recommended

- Download: nextnano_NEGF_beta_portable

The nextnano++ & nextnano.NEGF bundle

See the table in the product overview for description of the tools.

Windows

beta (2024-08-23)

Included Tools	Version
nextnanomat	*4.3.2.15*
nextnano++	*1.20.8*
nextnano^3	*2.6.4*
nextnano.NEGF	*2024-07-25*
nextnano.NEGF_classic	*2022-06-13*

- Download: nextnano_bundle_beta_installer — recommended
- Download: nextnano_bundle_beta_portable

alpha (2024-06-19)

Included Tools	Version
nextnanomat	*4.3.2.15*
nextnano++	*1.19.49*
nextnano^3	*2.6.2*
nextnano.NEGF	*2024-06-21*
nextnano.NEGF_classic	*2022-06-13*

- Download: nextnano_bundle_alpha_installer
- Download: nextnano_bundle_alpha_portable

Ubuntu

beta (2024-08-23)

Included Tools	Version
nextnanomat	*4.3.2.15*
nextnano++	*1.20.8*
nextnano^3	*2.6.4*
nextnano.NEGF	*2024-07-25*

- Download: nextnano_bundle_beta_portable_Ubuntu

alpha (2024-05-24)

Included Tools	Version
nextnanomat	*4.3.2.15*
nextnano++	*1.19.49*
nextnano³	*2.6.2*
nextnano.NEGF	*2024-06-21*

- Download: nextnano_bundle_alpha_portable_Ubuntu

2.3.3 Advanced downloads

Special versions of some of our tools can be found here. The simulation tools can be run only with an active **license**, which can be obtained through our registration page.

---

nextnanomat

Windows (32-bit)

stable (2023-08-01)

Included Tools	Version
nextnanomat	4.3.2.6

- Download: nextnanomat_stable_portable_32bit

The nextnano License Activators

A stand-alone package for license activator executables (2023-12-20) for Windows, Linux and macOS.

Windows

- Download: nextnanoLicenseActivator

Ubuntu

- Download: nextnanoLicenseActivator_Ubuntu

macOS

- Download: nextnanoLicenseActivator_macOS

2.3.4 Types of Packages and Versioning

Installer vs Portable

On this page one can find download links to multiple versions of our tools. Currently, we are providing packages marked as *installer* or *portable*.

installer - These packages contain an installation file that manages generation of a folder structure, generating shortcuts, etc., and connecting all tools that we are providing within the packages. New installations are independent of the previous ones, which means that one does not lose the previous versions of the tool during the installation process.

portable - These packages are zip files containing folder structure with all the tools. One does not need to install this package; It is ready to use directly as downloaded and unpacked. However, no shortcuts, system variables, or paths for license files are set up in these packages. These require manual setup. It may be a good choice for those who do not have administrator rights on the computer.

Versions

We are currently providing three versions of our tools: alpha, beta, and stable.

alpha - It is a version of the tool that has been developed most recently, often containing new minor features requested by our customers. We are publishing these versions, so you can have your new features as soon as possible.

- Some major features are quantitatively tested.
- Some features may be unstable.
- Not all provided input files may be updated to the new syntax.
- Some of provided input may not run until finished.
- Included documentation (pdf) may not be fully consistent with the provided package.

beta - It is a version of the tool which typically is internally consistent, and contains major changes introduced in some tools.

- Some major features are quantitatively tested.
- All provided input files are updated to the new syntax and run until finished.
- Included documentation (pdf) is updated and consistent with the provided package.

stable - It is a version of the tool which fulfills all requirements for being called beta, and for which we have carefully tested values outputted from simulations to make sure that all the models are running as they should.

- All major features are quantitatively tested.
- No known bugs are present at the date of the release.

CHAPTER

THREE

LICENSE ACTIVATION

3.1 License activation via command line

3.1.1 Available license file on different operating systems

operating system	*nextnano++*	*nextnano³*	*nextnano.NEGF*
Ubuntu (Linux)	`.lic` file	`.txt` file	`.lic` file
Intel Mac	`.txt` file	`.txt` file	unavailable
Arm Mac	`.txt` file (as of 5th March 2024)	`.txt` file	On request

To obtain `.txt` license file, please contact us via *nextnano Help Center* to request issuing the license.

3.1.2 Online and offline activation

Important: nextnano GmbH licenses are hardware specific. You have to perform the license activation on the same computer where you want to run the software.

The executable necessary for license activation is called:

- *deprecated* Windows native: **ClientSideServerActivation** and **ClientSideActivator** (before May 2023)
- *deprecated* Linux and Mac native: **ClientSideActivator** (between May and October 2023)
- Windows, Linux, and Mac native: **nextnanoLicenseActivator.exe** for Windows or **nextnanoLicenseActivator_<OS NAME>** for other operating systems (since October 2023)

On Linux and Mac, you might have to add the execute permission to the executable by

::

    chmod 744 nextnanoLicenseActivator_<OS NAME>

For each activator version, the necessary and optional parameters will be displayed if you run the executable without giving any parameters.

Note: In the previously distributed and deprecated license-activation executables the argument "email" refers to the *License-Key*.

Note: When you activate the license, please enter exactly your License Key for the licensekey parameter. You will receive the key via email after purchasing a license, it has the format:

`<email of license owner>-<end of validity>-<5-15 digits hash>`

Attention: If you use ARM Mac for simulation, you have to "emulate Intel processor" before initiating the license activation. See *macOS* for details.

Command line syntax:

```
./nextnanoLicenseActivator --firstname <person first name>  --lastname <person last␣
↪name>  --email <person email>
--institution <institution> --licensekey <license key> --outpath <output folder path>
[--fingerprint <path to fingerprint file> in case of Fingerprint activation]
[--address <server IP address> in case default one shall be overwritten]
[--offline <> in case of offline activation]
```

Note that `nextnanoLicenseActivator` changes, depending on your computer. e.g. if you use macOS, it becomes `nextnanoLicenseActivator_macOS`.

If everything works, you will find the license file(s) `License_<nextnano tool>.lic` in your specified output directory. The number of the received files depends on your contract. Each of our products has its own distinctive license file, however `License_nnp.lic` can be used for both the *nextnano++* and *nextnano*^{3} tools.

If the communication with the licensing server is not successful, please specify the `--offline` parameter to obtain a fingerprint file. After we receive your fingerprint file, we will activate and send you your license(s).

Please contact us via *nextnano Help Center* if you encounter any issues.

Attention: If you use ARM Mac for simulation, as of 5th March 2024, you can run simulation with only **license.txt**. We are working on this issue. See also *macOS*.

3.1.3 Fingerprint activation

Fingerprint activation is a special form of online activation. Instead of sending the hardware fingerprint of the current computer, the selected hardware fingerprint file will be sent to the nextnano GmbH license server. As a result, the received license files will only work on the computer on which the fingerprint file was generated and not on the computer used for license activation.

3.2 License activation via nextnanomat

After purchasing a software license or starting a trial period, you will receive a license key per email. This license key is used to generate a license for your computer by *activating your license*.

Note: *Activation* won't influence the validity period of your license. The validity/trial interval starts according to your purchase process. *Activation* will activate the license **on your computer**.

Important: nextnano GmbH licenses are hardware specific. You have to perform the license activation on the same computer where you want to run the software.

If no licenses are found at program start of *nextnanomat*, the license activation dialog will prompt automatically. Else you can start the license activation dialog by clicking the main menu function *Tools* ==> *Activate License*.

When you activate the license, please enter exactly your License Key into the field *License key*. You will receive the key via email after purchasing a license, it has the format:

`<email of license owner>-<end of validity>-<5-15 digits hash>`

1. Online activation

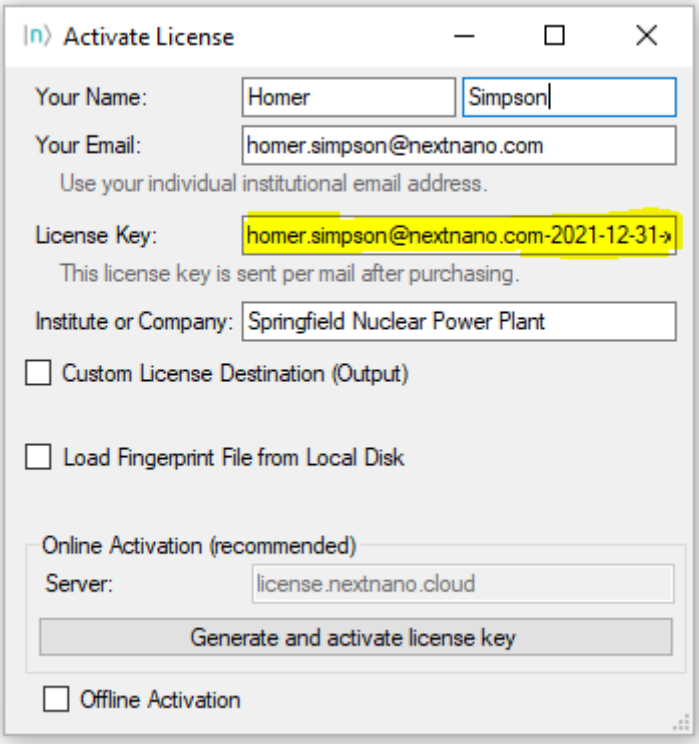


Figure 3.2.1: License activation dialog with example License Key.

By clicking *Generate and activate license key* your hardware fingerprint will be sent to our license server and you will instantly receive all license files associated with your license key.

2. Offline activation

   If your institution does not allow *Online Activation* or the connection fails, you can can use *Offline activation*. Check this option in the activation dialog, then the hardware fingerprint will be written to a text file. After saving the file on your computer you have two options to receive your license file(s):

   1. Fingerprint activation from another device (by your colleague or yourself)
   2. Fingerprint activation via nextnano GmbH support (can take up to two weekdays)

3. Fingerprint activation

   Fingerprint activation is a special form of online activation. Instead of sending the hardware fingerprint of the current computer, the selected hardware fingerprint file will be sent to the nextnano GmbH license server. As a result, the received license files will only work on the computer on which the fingerprint file was generated and not on the computer used for license activation. To use this option check *Load Fingerprint File from Local Disk*.

Hint: If you haven't done so already, you first have to allow expert features in *nextnanomat* ==> *Tools* ==> *Options* ==> *Expert settings*.

The number of the received files depends on your contract. Each of our products has its own distinctive license file, however `License_nnp.lic` can be used for both the *nextnano++* and *nextnano*^{3} tools. After a successful online activation you can choose to automatically update the paths to your license file(s). If you want to update or check the location of the license file(s) manually, go to:

nextnanomat ==> *Tools* ==> *Options* ==> *Licenses*.

If you encounter any issues regarding license activation or validity, please check our FAQ section: *FAQ - Licensing*

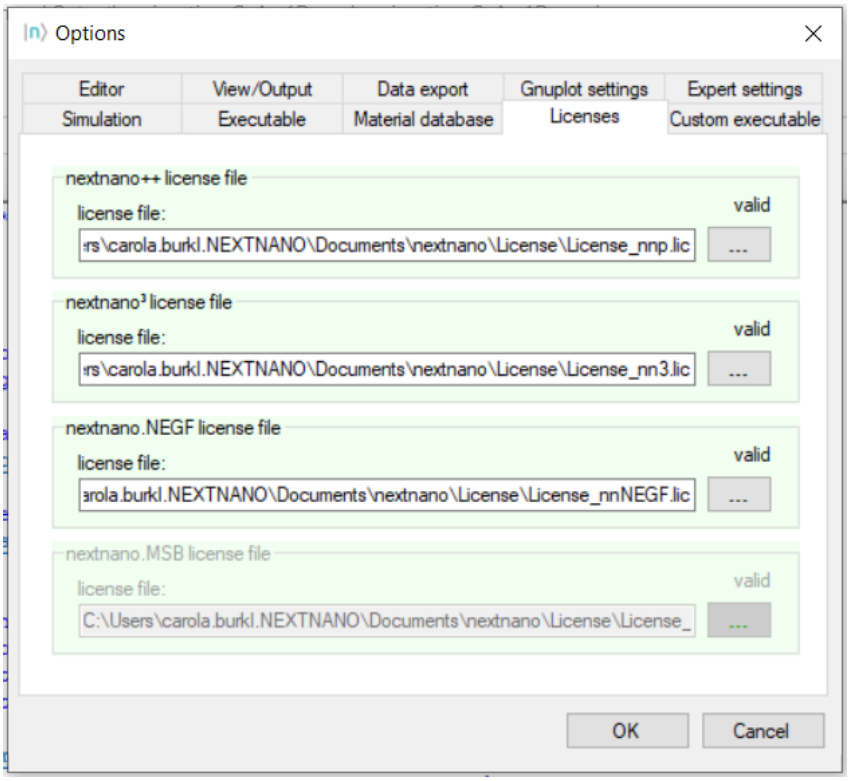


Figure 3.2.2: Path settings for the nextnano GmbH license files.

CHAPTER

FOUR

DIGITAL ACCESSIBILITY

The digital accessibility statement and the detailed report for the nextnano software can be found here.

4.1 Accessibility Statement for nextnano Software

The company nextnano GmbH is producing scientific software for a very specific audience. Due to its small company size, targeting a full accessibility especially retroactively is impossible. However, we aim to consider accessibility in any new software developments.

4.1.1 Conformance status

The Web Content Accessibility Guidelines (WCAG) define requirements for designers and developers to improve accessibility for people with disabilities. It defines three levels of conformance: Level A, Level AA, and Level AAA. The nextnano software is partially conformant with WCAG 2.1 Level A. Partially conformant means that some parts of the content do not fully conform to the accessibility standard. On some functionality conformance up to Level AA is achieved.

4.1.2 Feedback

We welcome your feedback on the accessibility of the nextnano software. Please let us know if you encounter accessibility barriers:

- Phone: +49-8121-7603205
- Support widget on this website
- support(at)nextnano.com

4.1.3 Compatibility with assistive technology

Tested and recommended on Windows 10. Partially compatible with screen reader software NVDA. Fully compatible with Windows built-in Screen Magnifier up to 400%, App Zoom up to 175% and color inversion. Not compatible with dark high contrast themes.

4.1.4 Limitations and alternatives

Cognitive and Learning Disabilities

Not suited due to scientific purpose of software and challenging cognitive performance for meaningful usage.

Hearing impairment

No limitations. Software does not include sound or audio.

Low vision

Graphical user interface (GUI) ***nextnanomat***

- Font style and size of Editor tab adjustable.
- GUI compatible with screen magnifier at least up to 400%.
- Colormap choices for most common types of color blindness - Protanopia, Deuteranopia and Tritanopia.
- Screen reader compatibility, only partially given.

Alternative 1: Without using GUI

- Possibility to execute the scientific program by command line only (screen reader compatibility)
- Common output format → Visualization by any preferred visualization software possible.

Alternative 2:
nextnano GmbH also accepts simulation requests (charged) → Employees can simulate requested structures for you.

4.1.5 Assessment approach

The company nextnano GmbH assessed the accessibility of the nextnano software by self-evaluation.

4.2 Accessibility Evaluation Report for nextnano Software

4.2.1 Executive Summary

This report describes the conformance of the nextnano software with W3C's Web Content Accessibility Guidelines (WCAG). Based on this evaluation, the nextnano software is close to meeting WCAG 2.1, Conformance Level A. Partially conforming to Level AA as well. Detailed review results are available below.

4.2.2 Scope of Review

Full software: nextnano GmbH Graphical User Interface (GUI) *nextnanomat* *Version 4.3.1.0, date 2021-Dec-14.*

4.2.3 Reviewers

Carola Burkl, Developer of *nextnanomat* (GUI)

4.2.4 Review Process

Conformance was tested for WCAG 2.1 Level A and AA. Manual review based on Praxent Accessibility Guidelines adapted for software. Compatibility with Windows built-in accessibility tools tested. Compatibility with screen reader NVDA tested.

4.2.5 Results

Interpretative summary of review results

Mostly conformant on WCAG 2.1 Level A. On some functionality conformance up to Level AA is achieved.

Detailed results

Audio

No audio

Video

No video

Content

- Meaningful Order (1.3.2) [A] → yes
- Sensory Capabilities (1.3.3) [A] → yes
- Under three flashes (2.3.1 & 2.2.2) [A] → yes
- Images of text (1.4.5) [AA] → yes
- Text Size & Spacing (1.4.12 & 1.4.4) [AA] → partially

  Compatible with screen magnifier at least up to 300% without loss of functionality or content.
- Content on Hover & Focus (1.4.13) [AA] → no
  - Content present on hover cannot be dismissed.
  - Hover content does not block region of mouse pointer.
- Alternative Text (1.1.1) [A] → partially
  - All images and non-text content of software have alternative text. → yes
  - Visual output of simulation results is not text interpreted. But result data can be viewed as text file as well.
- Page titles (2.4.2) [A] → yes
- Descriptive headings & labels (2.4.6) [AA] → partially
  - Buttons of main menu → yes (same functionality as buttons of tab pages)
  - Buttons of tab pages → no
- Language Changes (3.1.2) [AA] → no

Navigation & Architecture

- Clear text links (2.4.4) [A] → yes
- Multiple Paths (2.4.5) [AA] → yes

  Functionality of buttons can be accessed via buttons or navigation menu

Interface Elements

- Form Labels & Instructions (3.3.2) [A] → yes

  All forms and input fields are labeled
- Label in name (2.5.3) [A] → partially
- Consistent Identification (3.2.4) [AA] → no
- Name, Role, Value (4.1.2) [A] → yes, although name is not always descriptive
- Consistent Identification (3.2.4) [AA] → not relevant: no corresponding representation in non-html software

Interactions

- Focus order (2.4.3) [A] → no
- No focus change (3.2.1) [A] → yes
- No input change (3.2.2) [A] → yes
- Error identification (3.3.1) [A] → yes
- Keyboard focus visible (2.4.7) [AA] → yes
- Error suggestions (3.3.3) [AA] → yes
- Error preventions (3.3.4) [AA] → not relevant: no corresponding representation in software
- Pointer gestures (2.5.1) [A] → not relevant: no high precise pointer movements necessary
- Pointer Cancellations (2.5.2) [A] → no
- Motion actuation (2.5.4) [A] → not relevant: no motion functionality is used
- Keyboard only (2.1.1) [A] → no
  - Inside a tab → yes.
  - But to change tabs → no.
  - Logical tab order → partially
- No keyboard traps (2.1.2) [A] → no
- Character Key Shortcuts (2.1.4) [A] → yes (only active at focus)
- Adjustable Time (2.2.1) [A] → not relevant: no existing time limits

Visibility

- Use of color (1.4.1) [A] → no
  - Visualization of 2D or 3D data in the output tab uses heat maps to represent data values.
  - However, the color maps include choices for different types of color blindness as well as monochromous options.
- Orientation (1.3.4) [AA] → yes (GUI can be resized to custom preference)
- Text & Image contrast (1.4.3) [AA] → yes
- Additional element contrast (1.4.11) [AA] → no

Code

- Clean code (4.1.1) [A] → not relevant: no html in software
- Reflow (1.4.10) → yes

Testing

- Assistive technologies
  - Screen reader NVDA tested
  - Screen magnifier (Windows) tested, up to 400%
  - Color Inversion (Windows) tested
  - Display increased App size (Windows) tested, up to 175%
  - Make text bigger (Windows) - does not work well
  - High contrast (Windows)
    - Black background does not work
    - Other background works but is not recommended
- Tested and recommended on Windows 10

4.2.6 References

1. **Web Content Accessibility Guidelines (WCAG) Overview**
   https://www.w3.org/WAI/intro/wcag
2. **Web Content Accessibility Guidelines 2.1**
   https://www.w3.org/TR/WCAG21/
3. **Techniques for WCAG 2.1**
   https://www.w3.org/WAI/WCAG21/Techniques/
4. **Accessibility Evaluation Resources**
   http://www.w3.org/WAI/eval/
5. **Praxent Accessibility Guidelines Checklist**
   https://praxent.com/wp-content/uploads/2020/01/Accessibility_Guidelines_Linked.pdf

CHAPTER

FIVE

NEXTNANOMAT

The *nextnanomat* GUI is a convenient graphical user interface for *nextnano++*, *nextnano*^{3}, and *nextnano.NEGF*. It allows for 1D, 2D and 3D visualizations of computed results. This workflow manager is designed in a general way allowing to run any executables and overview generated output files.

5.1 GUI tabs

5.1.1 Input

The input file tab supports both ASCII and XML format.

General features:

- Syntax highlighting
- Autocomplete
- Context-sensitive online help
- Use of variables

The syntax and the supported features depend on the tool used for calculations. For detailed information see related documentation:

- *nextnano++*
- *nextnano*^{3}
- *nextnano.MSB*
- *nextnano.NEGF*

Autocomplete feature

By using the key combination 'ctrl' plus 'space' the autocomplete menu will pop up. It will show available keywords for the current section as well as the names of defined variables.

5.1.2 Template

This web page describes the template feature of *nextnanomat*.

In the Template tab you can **overview all variables** which are defined in the input file. You can **sweep** a chosen variable by a list of values or a range of values. Additionally, there is the option to compare the sweep results by the **post-processing feature**: It collects specified data from sweep files and stores them in a separate file, so that e.g. interband transition energy can be visualized as a function of well width.

An example how the template can be used to sweep over a variable, e.g. the quantum well width, is shown in this *Exciton Binding Energy in an Infinite Quantum Well*.

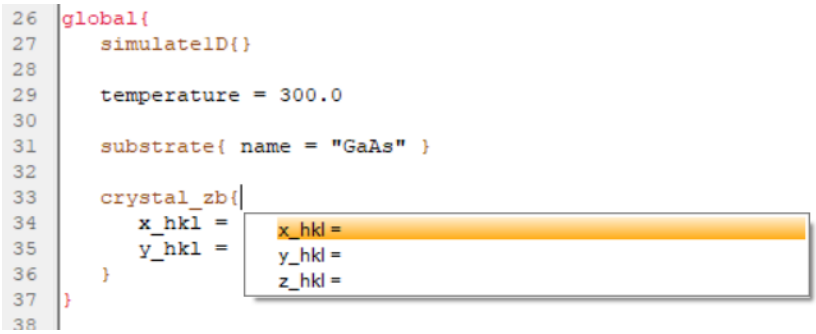


Figure 5.1.1.1: The *nextnanomat* autocomplete menu.

Variable definition in input file (required)

```
<Variable> = <Value> <Comment>
```

with the following syntax definition:

Variable (required)
A <Variable> starts with a $ or % sign (% should be preferred), can contain the characters A-Z, a-z, 0-9 and the underscore (_). It is case-sensitive.

- For *nextnano++*, the definition of variables and examples can be found in *Input Syntax*.
- For *nextnano*[3], the definition of variables and examples can be found in *Macro features*.

Value (required)
Any string after the = sign without # and ! characters is considered as <Value>. It is used as default value for the Template and *Template (Beta)* user interface.

Comment (optional)
Starts with a # or ! sign and is used as a description of the variable in the Template user interface. The comment may also include the following keywords (including brackets):

- `# (DoNotShowInUserInterface)`
  This variable is not shown to the user in the Template user interface.
- `# (DisplayUnit:<Unit>)`
  A string to show the unit for the variable (optional)
- `# (ListOfValues:<default>)`
  Default values for `List of values` input control
- `# (RangeOfValues:From=1,To=10,Step=1)`
  Default values for `Range of values` input control
- `# (HighlightInUserInterface)`
  Highlights variable in Template user interface in yellow

Note: Every line that starts with a $ (not nn3) or % (after stripping leading spaces) and includes a = is considered a variable definition.

Example

```
$well_width = 6 # Variable for quantum well width. (DisplayUnit:<nm>)␣
↪(RangeOfValues:From=6,To=18,Step=2) (HighlightInUserInterface)
```

Sweep over a variable + optional post-processing

If an input file includes at least one variable definition, it is possible to automatically sweep over it. The `Template Tab` of *nextnanomat* can be seen in Figure 5.1.2.1, already pre-filled for a variable sweep. To reproduce, please follow these steps:

1. The input file has to be chosen as the template file. Thus all correct variable definitions are loaded into the list.
2. Choose a sweeping variable and complete the required fields, e.g. `List of values` for `QW_Separation`.
3. The input files can be created by clicking the `Create input files` button.
4. They will be added to the batch list. To start the simulations you have to switch to the `Simulation`-tab, see Figure 5.1.2.2.

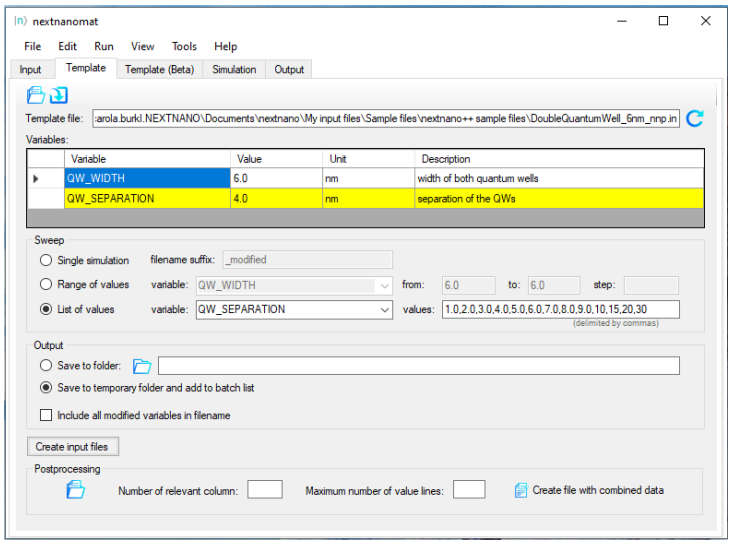


Figure 5.1.2.1: Template tab of *nextnanomat* with `double quantum well input file` loaded.

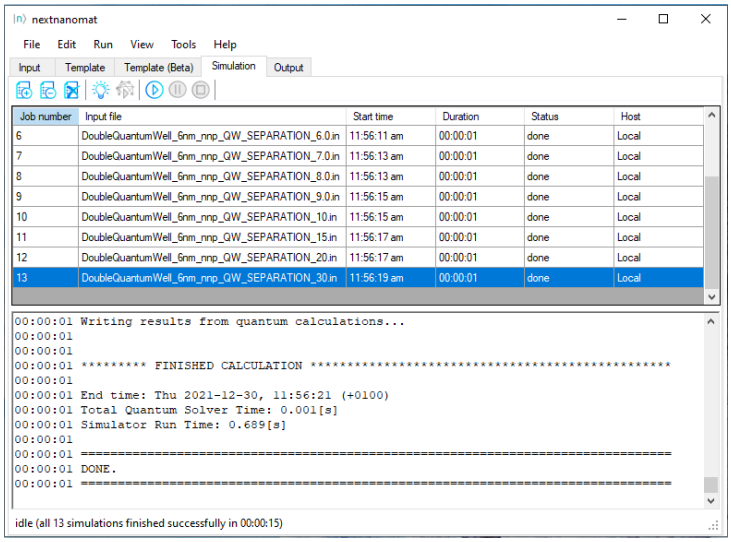


Figure 5.1.2.2: Batchlist of successfully simulated template sweep.

The sweep of a variable is now completed.

When the simulations are done, you have the possibility to use the post-processing feature. Its input fields are also located in the Template tab and can be seen in Figure 5.1.2.3. To use post-processing,

5. Choose the output file (*.dat) and the column number of the variable you want to compare, e.g. the file *energy_spectrum_quantum_region_Gamma.dat* displayed in Figure 5.1.2.4 and Figure 5.1.2.5
6. Additionally, state the maximum number of values (rows) which will be compared. (It is not allowed to exceed the existing number of rows.)
7. Now push the button `Create file with combined data` and
8. Visualize your results in the `Output`-tab, see Figure 5.1.2.6.

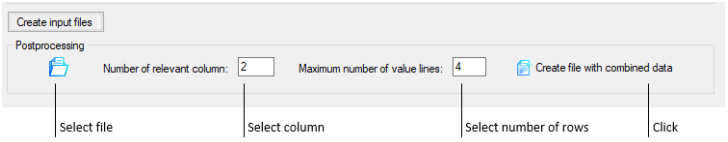


Figure 5.1.2.3: Steps to post-process a variable sweep.

For the steps 5 and 6 it is necessary to check the data structure of the simulation result you are interested in. Go to `Output` tab, select the data file and toggle `Text view`.

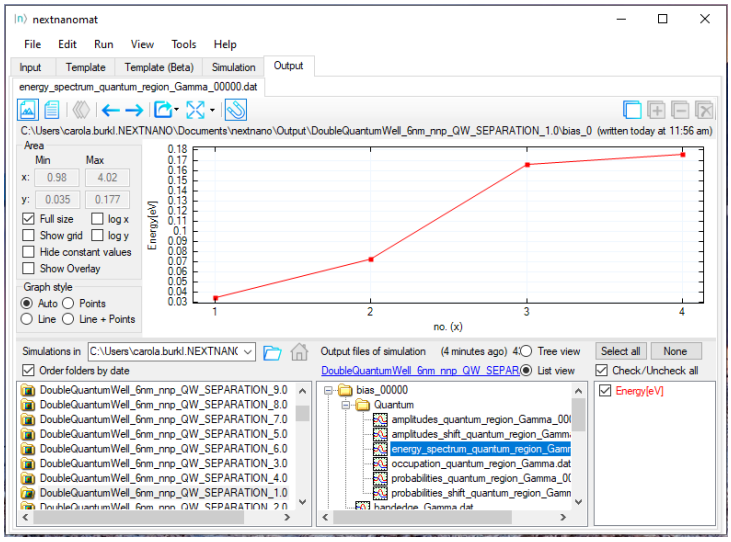


Figure 5.1.2.4: Energy eigenvalues of the first four electron states. Visualized as curves.

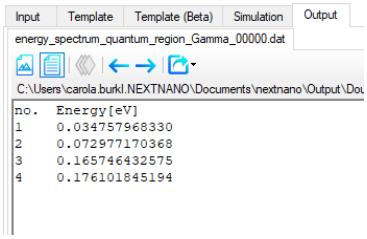


Figure 5.1.2.5: Text view of same output file.

The result plots the chosen data in relation of the sweeping variable, see Figure 5.1.2.6.

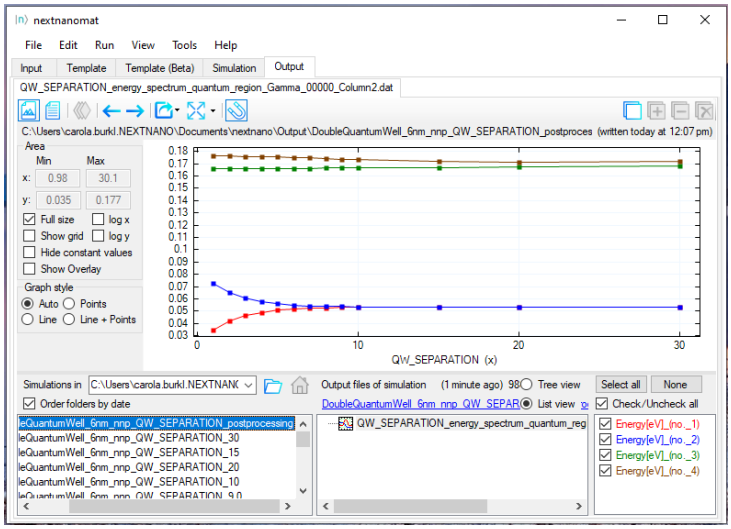


Figure 5.1.2.6: Result of post-processing. Electron energy eigenvalue in relation of quantum well separation.

Post-processing of existing sweep

If you have already done some sweeps in the past, you can also use the post-processing feature of *nextnanomat* retroactively. Note, additional to the post-processing steps described above, you need to load the template file and select and fill in the variable values which have been used for the sweep.

5.1.3 Template (Beta)

Introduction

The Template (Beta) tab has a similar functionality to the Template tab. It allows convenient sweeps of an input file through the use of variables. In comparison to the standard Template feature, the Beta version allows nested sweeps of multiple variables. Its use is intended for experienced users who need to investigate their structure in complex dependency of more than one variable.

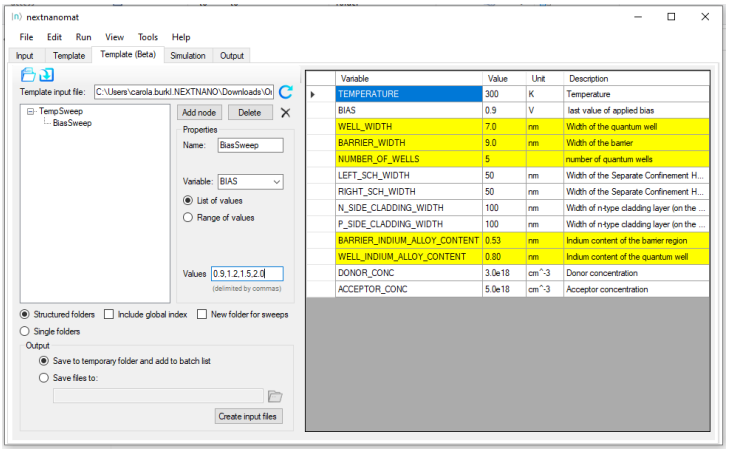


Figure 5.1.3.1: Full view of the Template (Beta) tab.

The input file

Any input file, which contains variables, can be used. At least two variables should be defined to use the full functionality of Template beta. Different options are provided to load an input file. It can be imported from the currently active input tab, searched within the file explorer or by inserting the path. Once opened, the defined variables should be shown on the right hand side. If changes to the file have been made by/with an external application, it can easily be updated through the reload button.

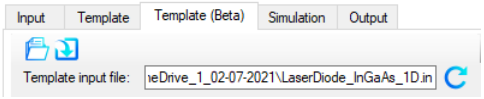


Figure 5.1.3.2: Menu to load and update the template file.

How to create a sweep

Follow the next steps:

1. (If the white panel is empty, click on the **Add node** button.)
2. Select the node **New Sweep**.
3. In the *property panel* you can define the parameters for this sweep:
    1. Choose a variable (e.g. `$TEMPERATURE`).
    2. Set a correlating name for the sweep (e.g. "TempSweep", Figure 5.1.3.3).
    3. Select **List of values** or **Range of values** and define the values for the sweep.
4. To add a second, third etc. sweep, repeat the steps above.
    1. To create a nested sweep, select a parent node before clicking on the **Add node** button. The new node will be nested below the selected one. (e.g. "BiasSweep", Figure 5.1.3.4)
    1. If no node is selected, the new sweep will be on the same level and they will not be correlated to each other.

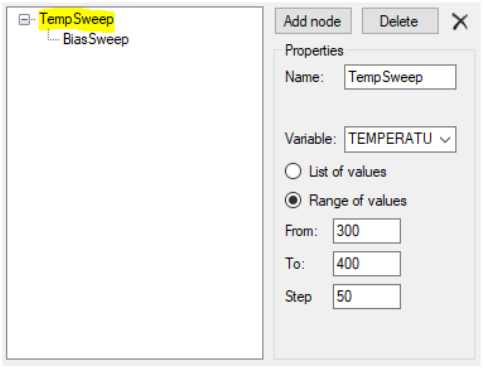


Figure 5.1.3.3: Parent sweep

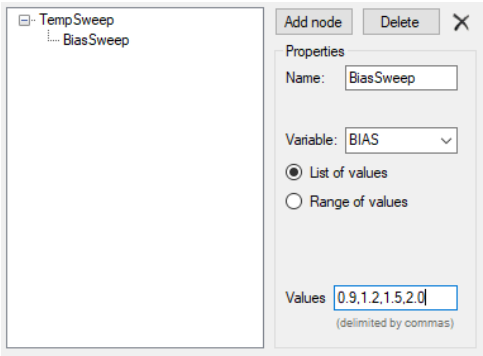


Figure 5.1.3.4: Nested sweep

Save input files and run a sweep

In this example we use three different temperature values and a nested bias sweep with four different bias values for each temperature (see Figure 5.1.3.3 and Figure 5.1.3.4). Thus 3x4 = 12 different input files are to be created. Each input file receives the name and the values of both sweep parameters for identification.

Before running the simulation, there are different options for the simulation output:

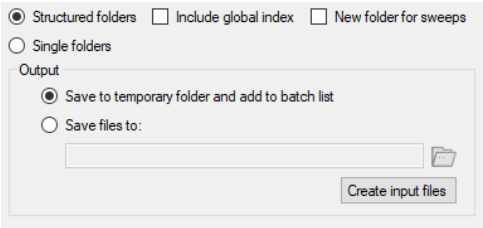


Figure 5.1.3.5: Output options

Either the option **Single folders** or **Structured folders** can be selected. The first option generates one unique simulation output folder for each input file (ergo, for this example separate 12 folders, see Figure 5.1.3.6), whereas the **structured** option generates nested output folders matching the sweep structure. For the structured option a global index can be included which enumerates each sweep (Figure 5.1.3.7). And/Or a parent folder - for each sweep - can be added (Figure 5.1.3.8).

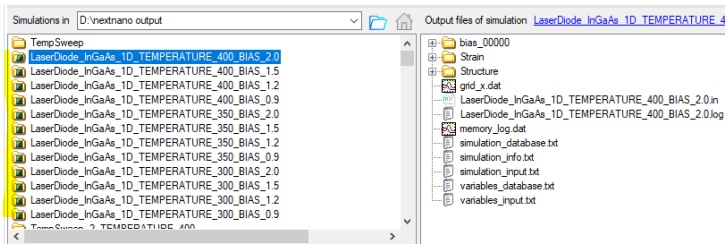
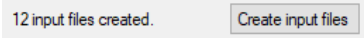


Figure 5.1.3.6: Output option a - Single folders.

Lastly you can choose whether you want to create the input files only temporary to directly execute them or if you want to store them on the disk for later use.

After setting all your output preferences, click on the **Create Input files** button. You should see a confirmation after your input files are created successfully:



Next, go to the **Simulation** tab to find the created input files and execute them.

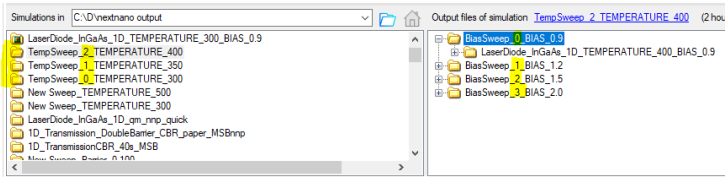


Figure 5.1.3.7: Output option b - Structured folders plus global index.

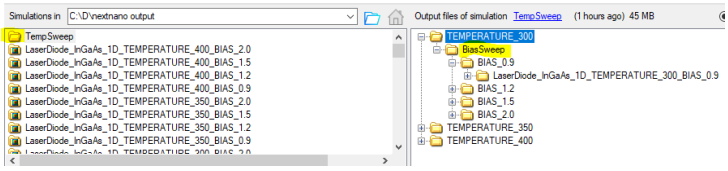


Figure 5.1.3.8: Output option c - Structured folders plus parent folder.

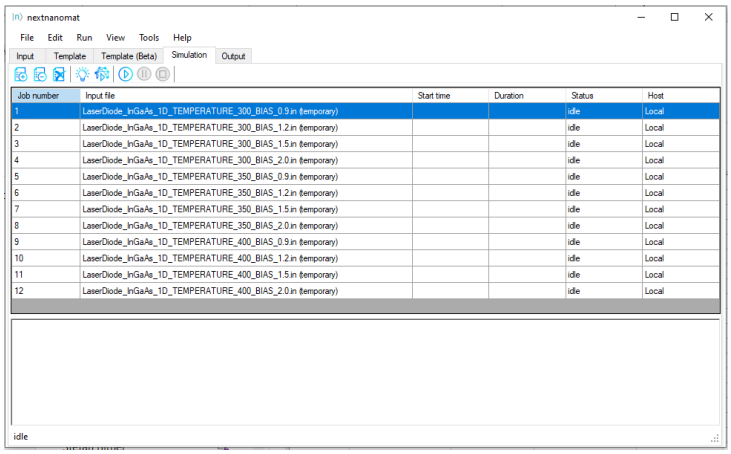


Figure 5.1.3.9: Created input files in simulation tab.

5.1.4 Simulation

The processing of simulations is done here:

- The simulations can be started, paused and stopped.
- There is a **batch list**, where input files can be added or removed. It is also possible to add complete directories plus sub-directories by the option `Add Directory Tree to Batch List` of the `Simulation`- menu tab.
- Information of the processed simulation is provided. Such as Start time, Duration and Status.
- The **.log-file** of the currently running simulation is written below. This file, with information on the calculation, is also stored in the output folder.

5.1.5 Output

Navigation

Menu

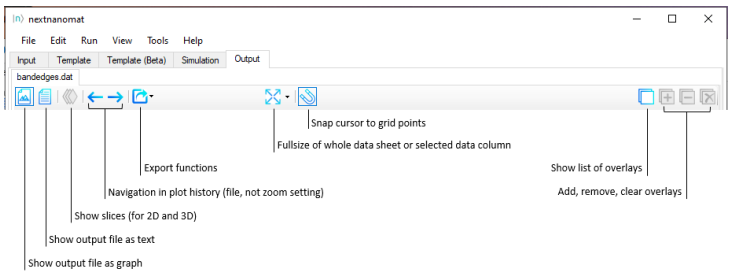


Figure 5.1.5.1: Menu, containing the most important functions to visualize the output data.

By right-clicking on the active output tab - which shows the output path or filename - you can create additional output tabs, for example to visualize different output directories. However these tasks are quite memory intensive, so use this feature with caution. We recommend to use a single output tab.

Panels for 1D, 2D and 3D

More specific visualization functions have their own panel. (For convenience some of them can be hidden.) Depending on the dimension of a data file, different functionality is available. For example, whether the graph should be represented by a dots or a line is only relevant for 1D data. Color maps and slices are only available for 2D and 3D data. The position and function of the different panels is shown in the following pictures.

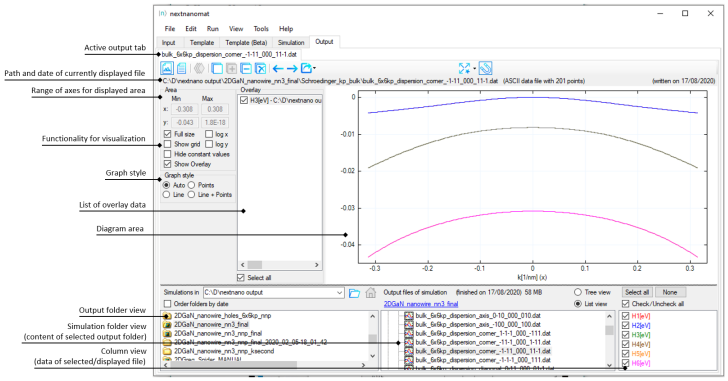


Figure 5.1.5.2: Panels for a 1D data file

Visualization

Supported Output Formats

The workflow manager *nextnanomat* can display the following output formats:

.txt
: The file is displayed as a **text**.

.dat
: The file is displayed as a **graph**, e.g. scalar field *f(x)* or vector field **F**(x)

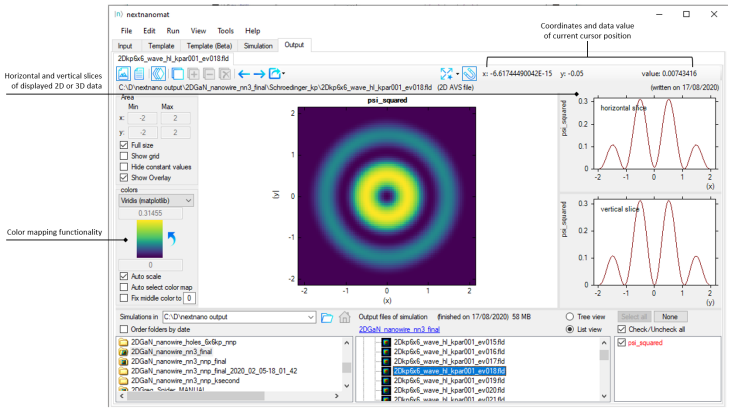


Figure 5.1.5.3: Panels for a 2D data file

x_1	$f(x_1)$
x_2	$f(x_2)$
...	
x_n	$f(x_n)$

or

x_1	$f_1(x_1)$	$f_2(x_1)$	...	$f_m(x_1)$
x_2	$f_1(x_2)$	$f_2(x_2)$	...	$f_m(x_2)$
...				
x_n	$f_1(x_n)$	$f_2(x_n)$	...	$f_m(x_n)$

.mtx or .mat

Matrix format for e.g. a matrix, a table, a *f(x,y)* graph. The x and y axes are labeled with integer numbers.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & ... & A_{1n} \\ A_{21} & A_{22} & A_{23} & ... & A_{2n} \\ ... & & & & \\ A_{m1} & A_{m2} & A_{m3} & ... & A_{mn} \end{bmatrix}$$

.vtr - 2D/3D

VTK format for rectilinear grid - scalar field *f(x,y,z)* or vector field **F**(x,y,z). They can be viewed using the ParaView software which is a full 3D visualization software while *nextnanomat* only displays 2D slices of 3D data files.

.fld - 1D/2D/3D

AVS format for rectilinear grid - scalar field *f(x,y,z)* or vector field **F**(x,y,z)

If a file extension is unknown it is treated as if it were a .txt file.

Text view

If `text view` is toggled, the content of the selected file is displayed. This view is read-only. For editing, the file can be exported to a custom texteditor. (We do not recommend to edit data files!)

1D View

Within graph view, one-dimensional data is displayed as curves. Whether data values are represented as points or lines is up to the user. Antialiasing can be switched on or off via *Output settings*. Additionally, the line thickness, background color or the usage of a grid can be customized.

If a data file contains multiple columns, each column is represented by its own curve. The list of available columns is shown within the `column view` panel. Only checked curves will be displayed. Selected curves are highlighted to allow fast recognition of related data.

Helpful features for one-dimensional data visualization are:

- *Overlay*
- *Hide constant values*
- *Show differences*
- *Snap to gridpoints*
- *Special fullsize*
- *Gnuplot export*

2D View

Two-dimensional data is displayed as a heat map, also known as pseudo-coloring. Each value of the two-dimensional grid is mapped to a distinct color. The color difference perceived allows the visual interpretation of value differences. Therefore the change of the color gradient needs to be perceived linearly by the human eye. Some color maps, like the rainbow map, introduce artefacts and have misleading visual perception, which increases the risk for misinterpretaion of scientific data. Find more information about *Color maps*.

Helpful features for two-dimensional data visualization are:

- *Auto select color map*
- *Fix middle color to specific value*
- *Gnuplot export* of surface plots

3D View

Real three-dimensional plots are not supported by *nextnanomat*. Instead 2D planes of the 3D data are visualized through heat maps, same as for 2D data. An additional panel allows the selection of the displayed plane (xy, xz or yz) and the position of the slice. This panel can be seen in Figure 5.1.5.13. If three-dimensional visualization is aimed for, we recommend exporting data to ParaView.

Helpful features for three-dimensional data visualization are:

- All of the 2D features
- *Exporting to ParaView*

Features

Overlay

For the best experience when visually analyzing the results of the simulation, it is sometimes necessary to look at different files at the same time. We call this the Overlay feature.

1. Select the plots you want to memorize in the `column view` panel.
2. Click the `Add to Overlay` button, see Figure 5.1.5.4. Alternatively, use the keys `a` (add) or + (Numpad only).
3. Select another output file.
4. (optional) add plots of multiple files to the overlay list.

⇒ Your memorized plots will be displayed in gray on top of the currently selected file.

You can check and edit the content of your overlay in the `overlay list` panel, this panel can be shown or hidden by clicking the `Show list of overlays` button. Remove selected curves with the `Remove from Overlay` button, shown in Figure 5.1.5.4, or use the `Del/Entf` or `d` (delete) key on your keyboard.

You can also export the overlay graphs to one combined image file, for further information refer to *Gnuplot export*.

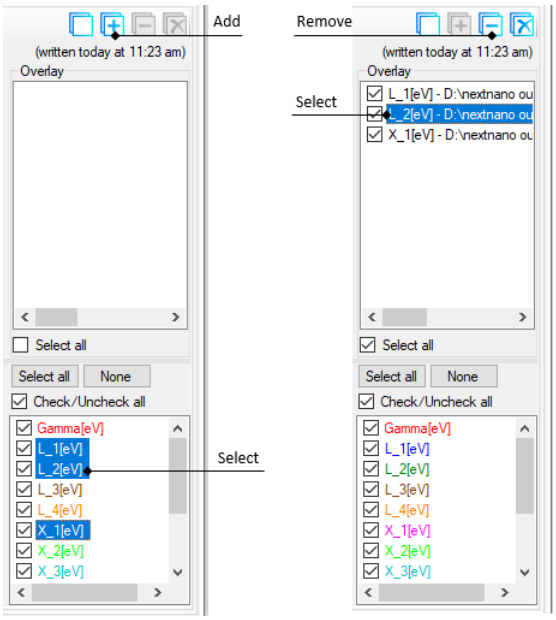


Figure 5.1.5.4: Workflow to add or remove files from Overlay.

Hide constant values

Note: Be careful when using this feature. We recommend to always disable it after usage.

This feature hides any parts of the curves where the value does not change within 5 grid points (two to the left and right). Its purpose is for visualizing wave functions and probability densities on top of band edges. The constant part which represents the energy level will be hidden. This allows to focus on the changing parts as well as for the underlying band edge to be seen. The influence of this feature can be seen in Figure 5.1.5.5 and Figure 5.1.5.6.

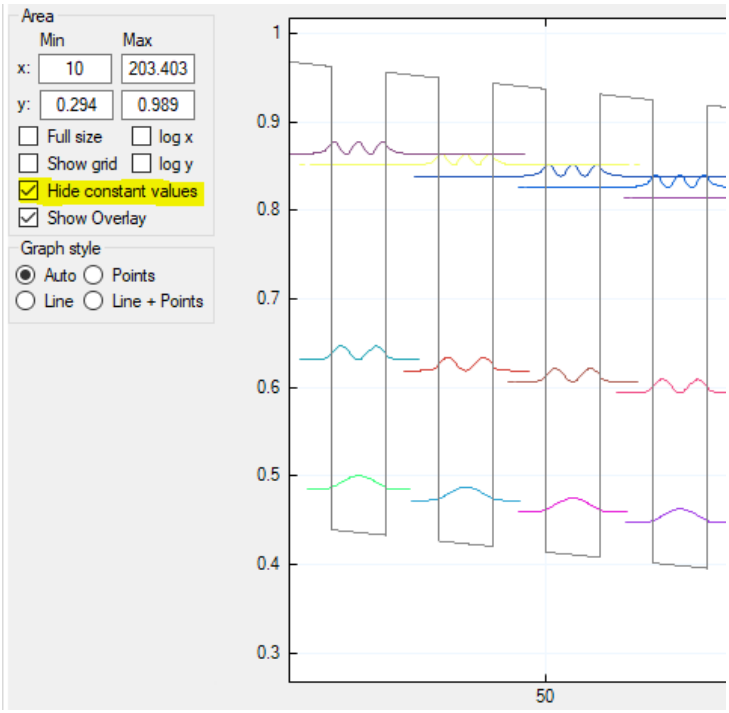


Figure 5.1.5.5: `Hide constant values` on.

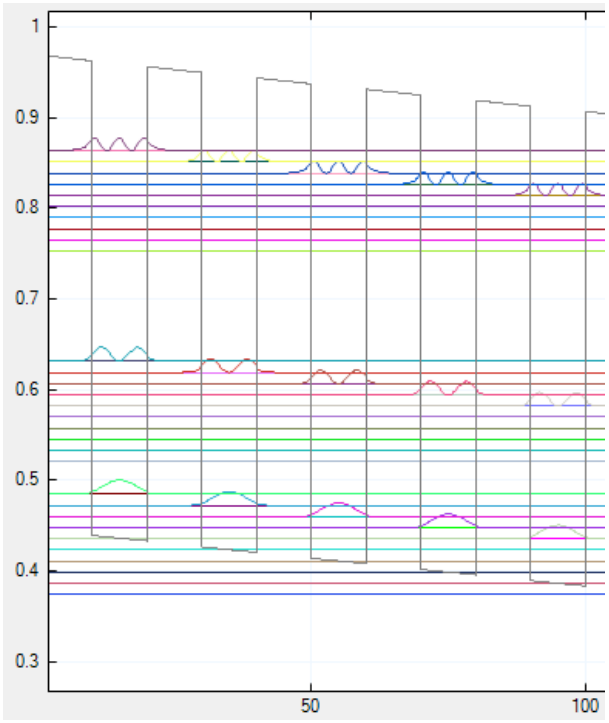


Figure 5.1.5.6: `Hide constant values` off.

Show differences

Whenever exactly two columns are selected in the `column view` panel, the value difference between these two curves is shown while the mouse is hovered over the data , see Figure 5.1.5.7.

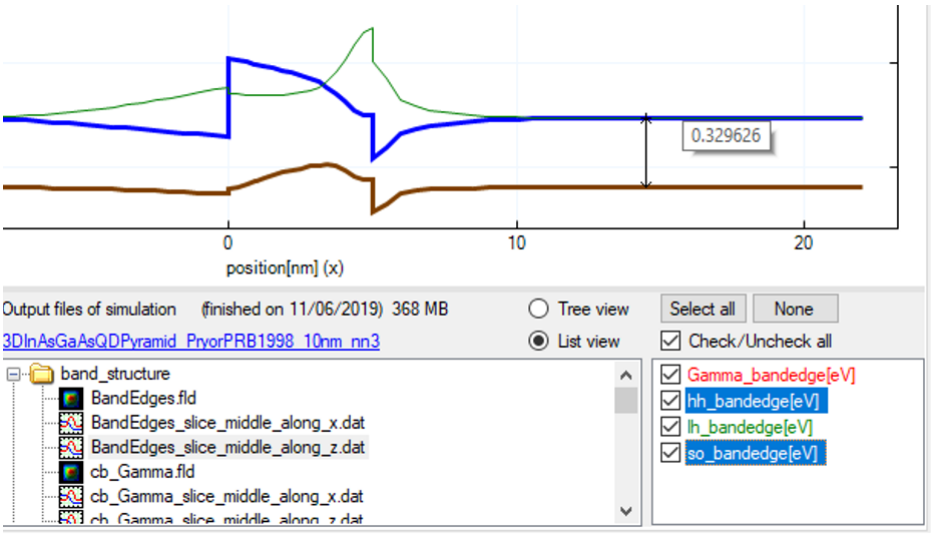


Figure 5.1.5.7: Arrow showing value difference of two data curves.

Snap to gridpoints

If `snap to grid points` button is toggled (default: on) a small cursor cross follows the nearest data curve while the mouse cursor is hovered over the data. The coordinates and value displayed in the upper right corner of *nextnanomat* match the data values of the curve, see Figure 5.1.5.8.

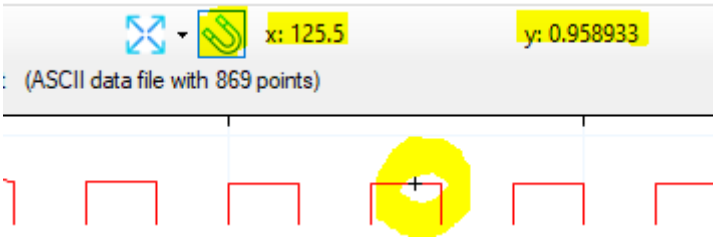


Figure 5.1.5.8: `Snap to grid points` feature enabled.

Special fullsize

By using this feature, the view of the output diagram will be matched perfectly to a single graph. Either the selected graph of the column view, or it is iterating through all available columns of the currently displayed file. This feature is useful for example if a file contains multiple columns with a highly different range of values. Displaying all graphs at the same time (normal fullsize mode) can lead to certain curves appearing to be zero. By selecting such curve and clicking the `special fullsize` button, the y-axis limits are set to its respective minimum and maximum and the curve is displayed correctly. An example for this use-case can be seen in Figure 5.1.5.9.

Furthermore this feature is also useful for any other file containing more than one column. Iterating through the columns and displaying each curve in its optimal frame, allows for fast and convenient data evaluation, see Figure 5.1.5.10. .. example one high number of curves

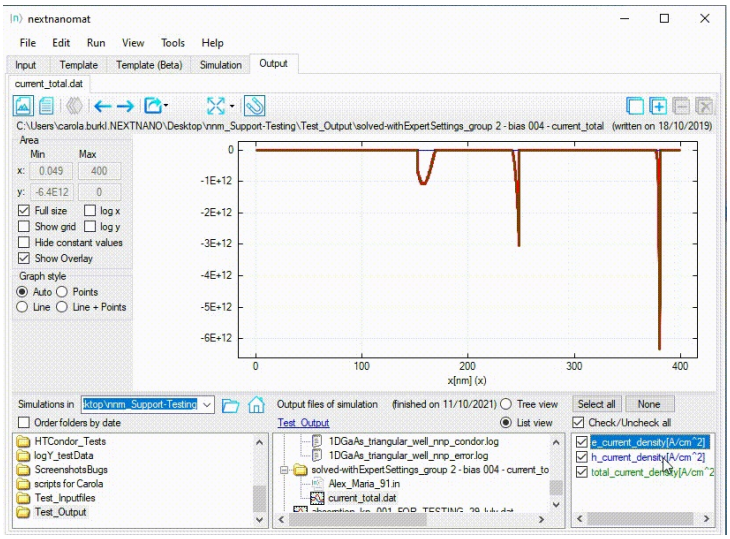


Figure 5.1.5.9: Special fullsize for a diverging range of values.

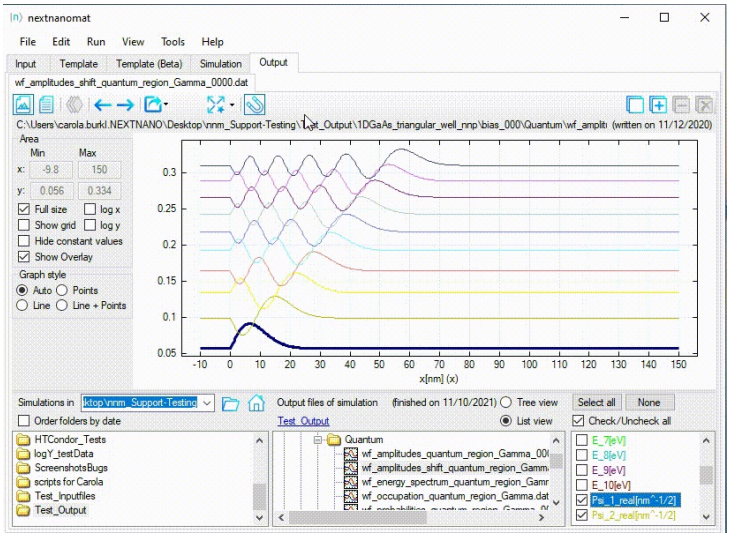


Figure 5.1.5.10: Special fullsize for a high number of curves.

Auto select color map

If auto selection of a color map is active, a diverging color map is selected for data files spanning from negative to positive values. Else a linear color map is chosen.

Fix middle color to specific value

This feature is aimed for the visualization of diverging data in combination with a diverging color map. It fixes the mapping of the neutral middle color to a specific value and thus ensures a symmetric color representation of diverging data.

Note: To be able to support custom color maps, this feature is not limited to predefined diverging color maps. To avoid unintentional usage we recommend to enable this feature on demand only.

The feature is enabled by checking the corresponding check box. If enabled, it is active whenever a diverging color map is assumed (error prone to enable flexibility) **and** the data file contains the specified value. Else it is inactive and grayed out. The visual feedback on the activity of the feature can be seen in Figure 5.1.5.11 and Figure 5.1.5.12

Example usage of the feature can be seen in Figure 5.1.5.13.

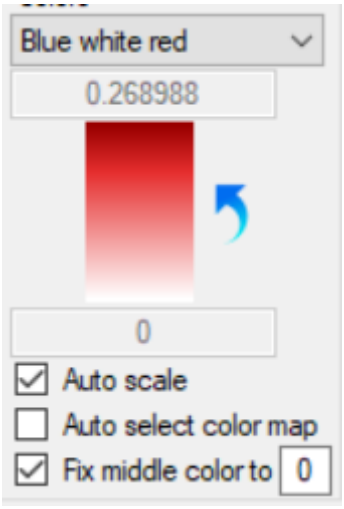


Figure 5.1.5.11: Enabled and active

Export functionality

Most of the export options are available from the context menu (right-click on the visualization). Some specific options can be accessed from the output menu button `Export and Open in specific Format`. For the latter option the custom defined paths to installed applications are used (with exception of the Gnuplot application). These paths need to be defined in the *Output settings*.

You can find a detailed description of the various export mechanism in the section *Export Functionalities*

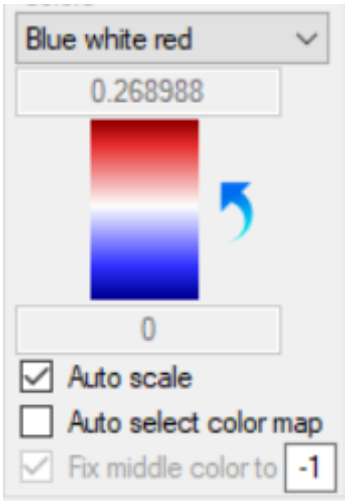


Figure 5.1.5.12: Enabled but inactive

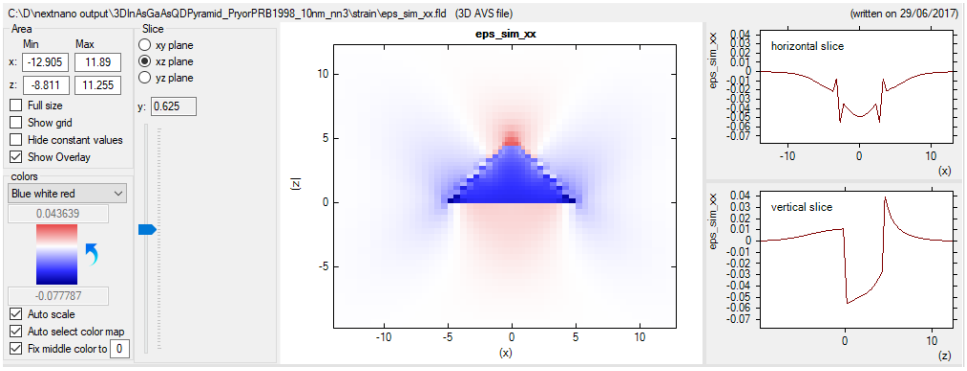


Figure 5.1.5.13: Visualization of diverging data with `fix middle color` feature enabled and active.

Further information

References, you might find helpful...

Output settings

Options: View/Output

Clean up output folder

Clean Up Simulation Output Folder

Color maps

Color maps

5.2 Settings

referred to and found in menu under **Tools** -> **Options**

5.2.1 Options: Simulation

5.2.2 Options: Material database

5.2.3 Options: Licenses

5.2.4 Options: Editor

5.2.5 Options: View/Output

5.2.6 Options: Expert settings

> **Warning:** Including memory usage in the log file notably extends time of simulation. Therefore, this option should be used only when necessary.

5.2.7 Options: Gnuplot settings

Gnuplot can be used to export and save nextnano GmbH simulation results, see documentation: *Gnuplot export*.

The default settings for the Gnuplot export can be adjusted in this options tab. For example, you can specify the line thickness, position of the legend (called `key` in Gnuplot) and many more.

If you want to use a **stylesheet** to incorporate advanced settings - like the specification of color sequence or line styles, you can either link an existing one or create one from the default nextnano GmbH Gnuplot settings. If you create a new one from the nextnano GmbH settings, the path to the stylesheet will be automatically updated. It is recommended to open that file with a texteditor to get familiar with the syntax. You can then easily adapt it to your needs.

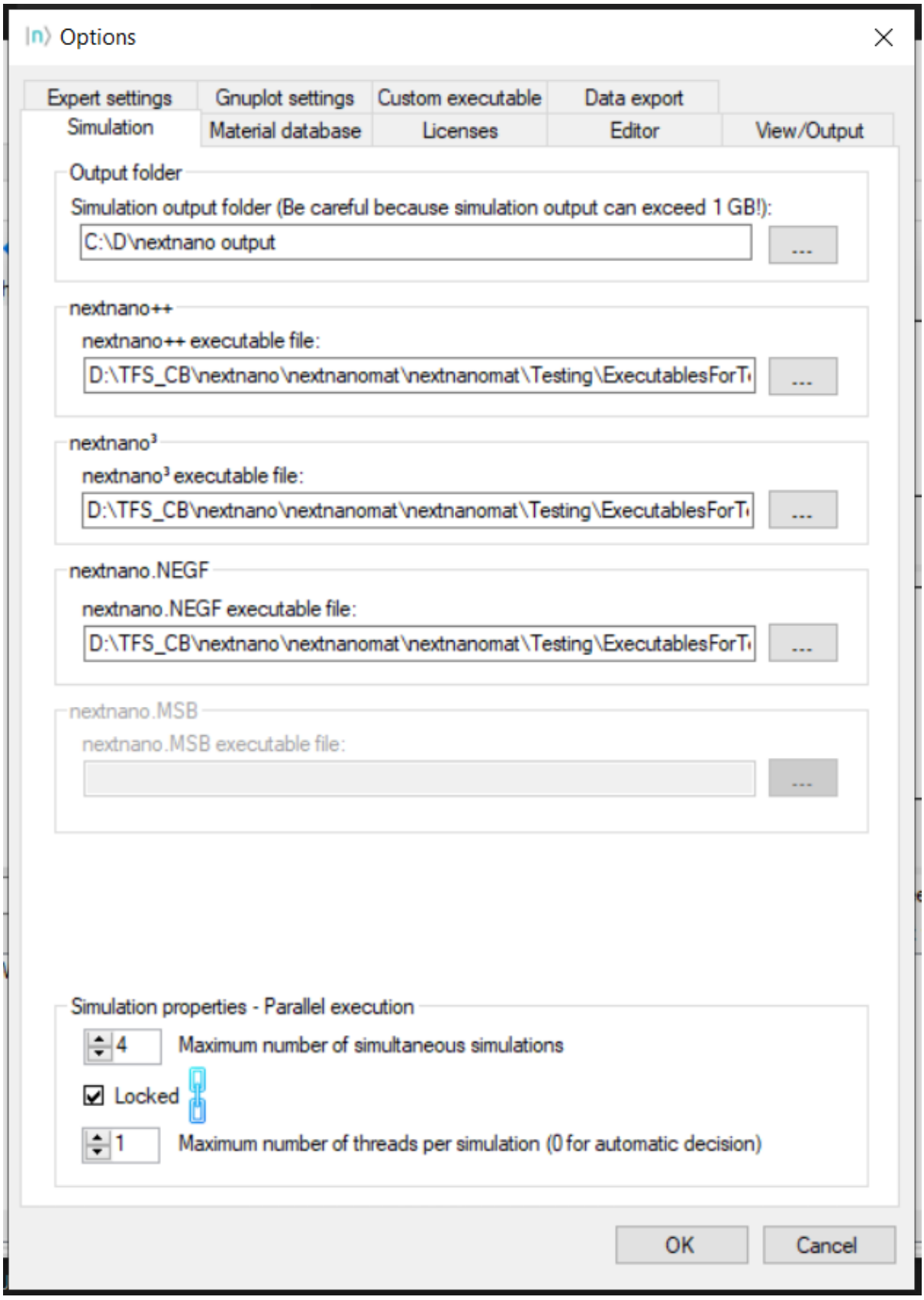


Figure 5.2.1.1: Settings for simulation and threading.

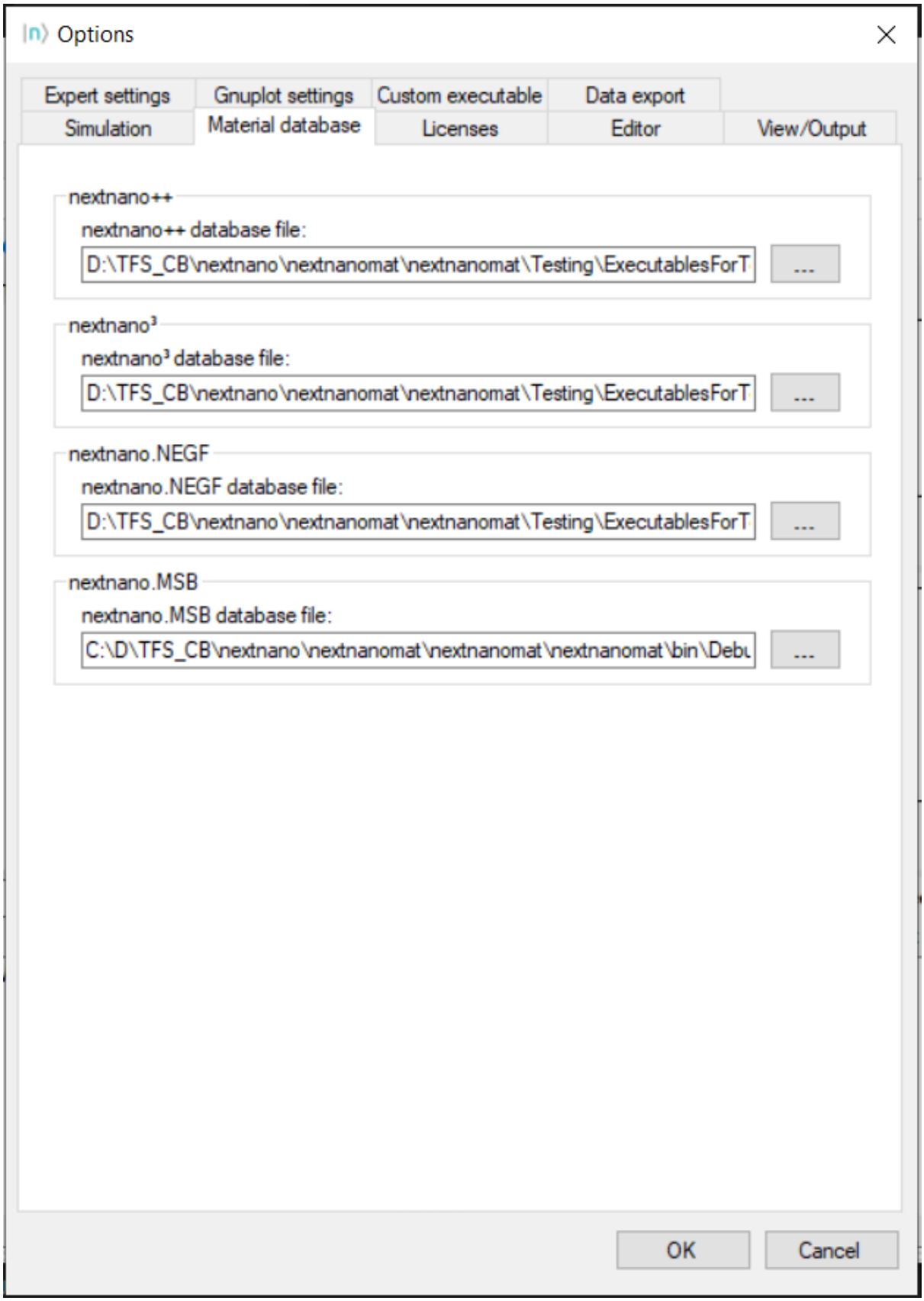


Figure 5.2.2.1: Settings for material databases.

Figure 5.2.3.1: Settings for nextnano GmbH Licenses.

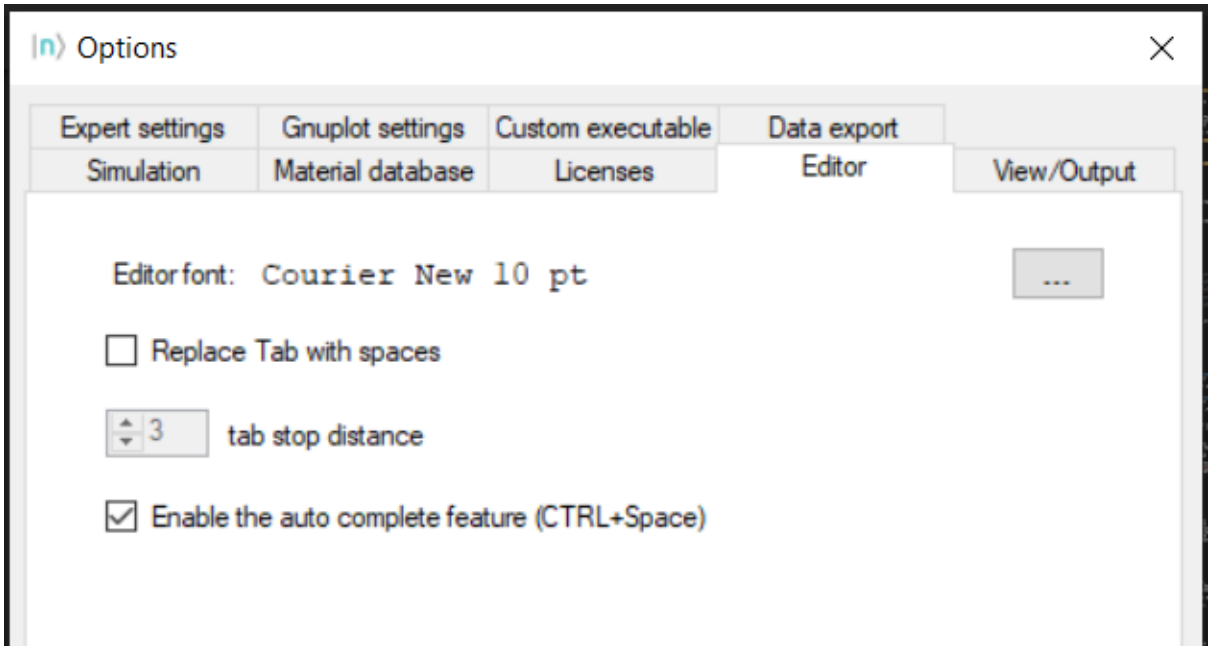


Figure 5.2.4.1: Settings for input tab.

Advantages of using a stylesheet

- Central, single file with your settings for plotting Gnuplot plots
- Use of links and variables (linking to the stylesheet) within the plot files
- Adjust settings at one place and automatically get the changes for all your plots
- Great for publications to achieve uniform style with possibility for last minute changes

To use a central stylesheet, select the checkbox *Insert link only*.

Note: You can create multiple stylesheets for different purposes. Set the path within Gnuplot settings to the one you want to use for new plots generated by *nextnanomat*.

5.2.8 Options: Custom executable

In the settigs shown in Figure 5.2.8.1 a custom executable can be defined, to be executed by the workflow manager *nextnanomat*.

Note: This is no default use-case for *nextnanomat* and should only be used by advanced users. Additionally, as it is a nice-to-have feature only, support regarding this feature is limited.

Most executables which can be started by command line, can be started by *nextnanomat*. This allows the usage of the batch list for polling and job execution control, as well as the visualization of output results (if the format is supported) or comfortable editing of input files. In many cases a combination with our cluster computing feature HTCondor is possible.

After defining the path to the executable as well as the working directory, a keyword identifier is necessary.

In case your executable needs input files:
: You need to find a common string within these files, thus *nextnanomat* can choose the correct executable for each job execution.

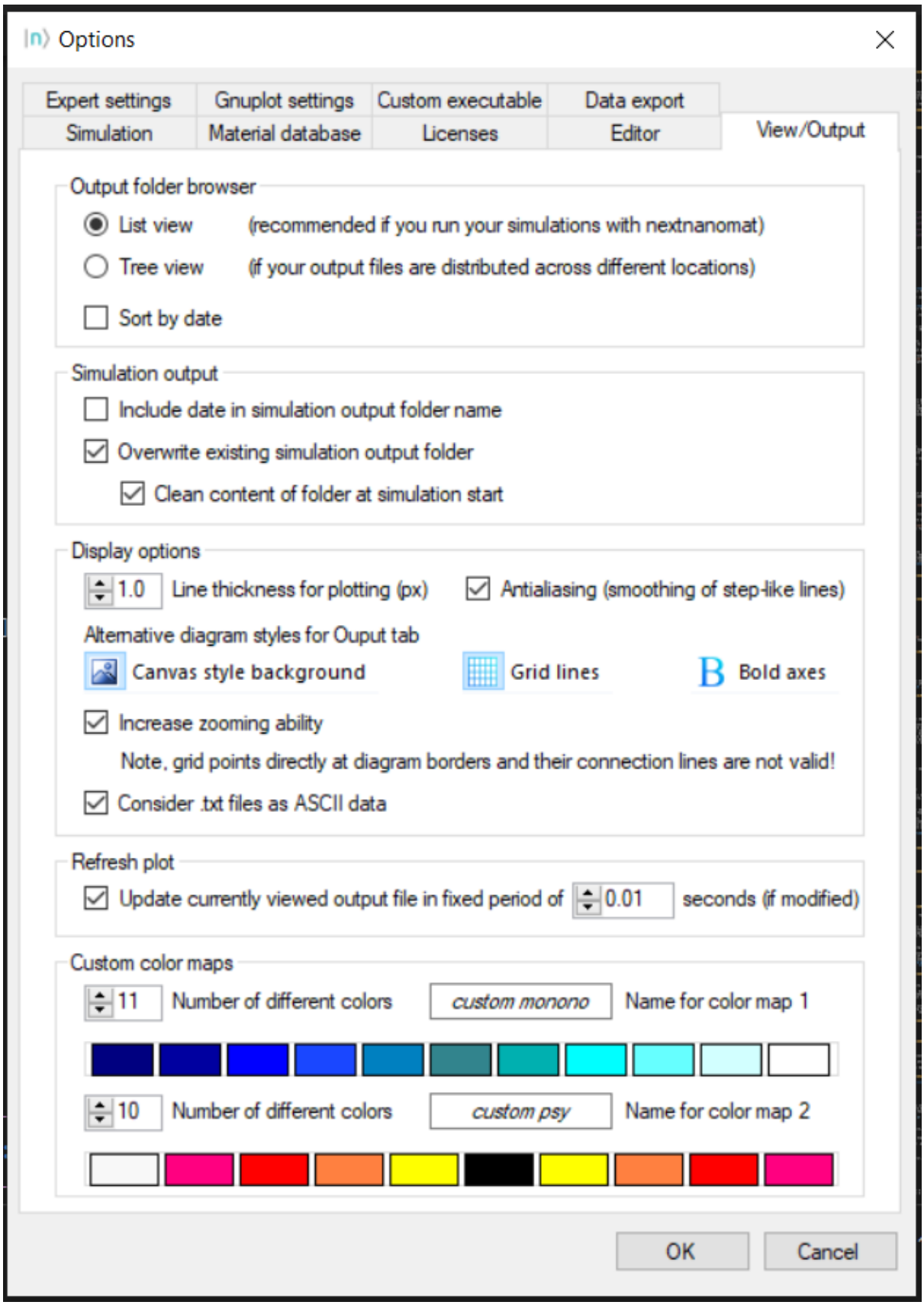


Figure 5.2.5.1: Settings for output tab and graph style.

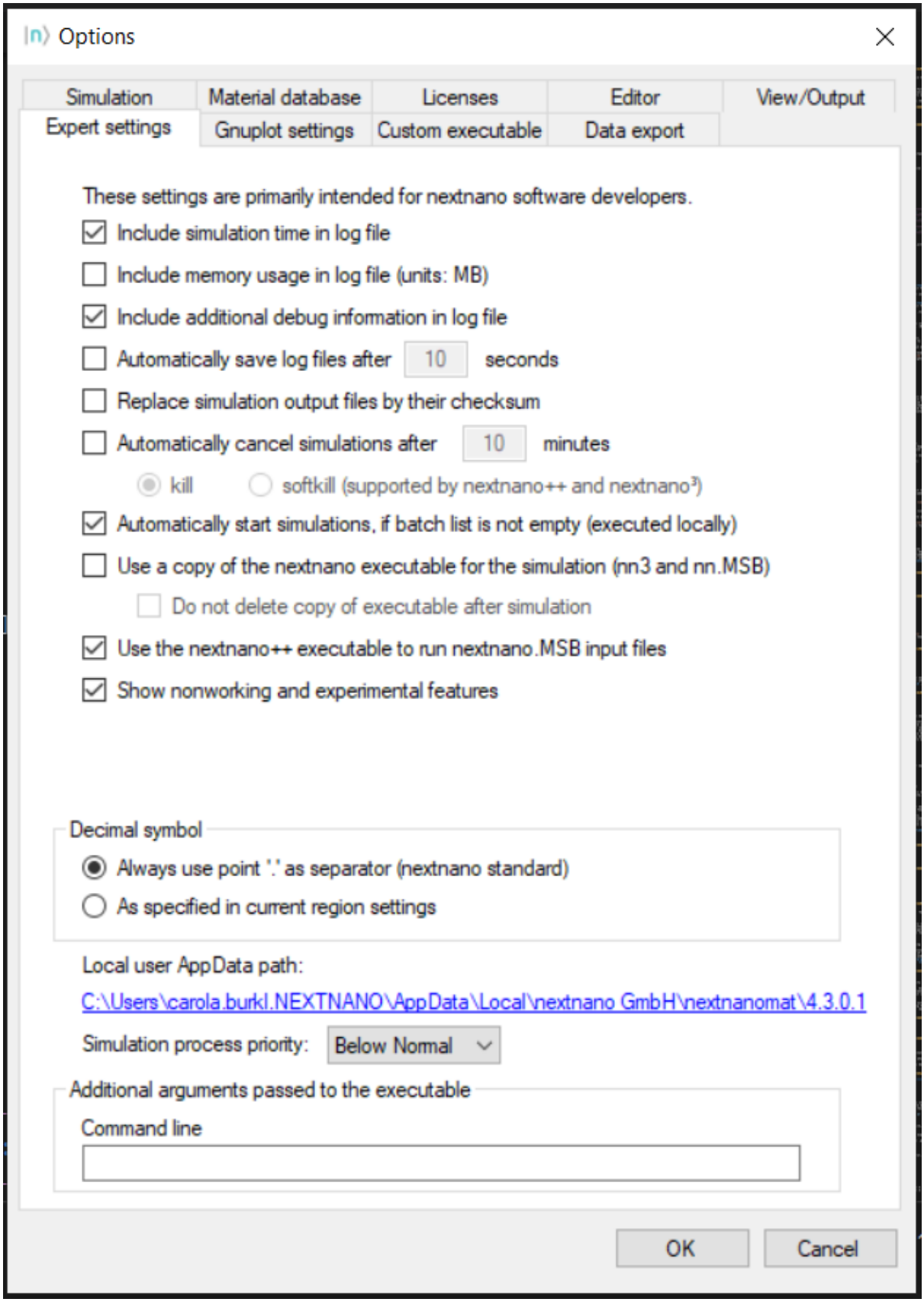


Figure 5.2.6.1: Settings for advanced user experience.

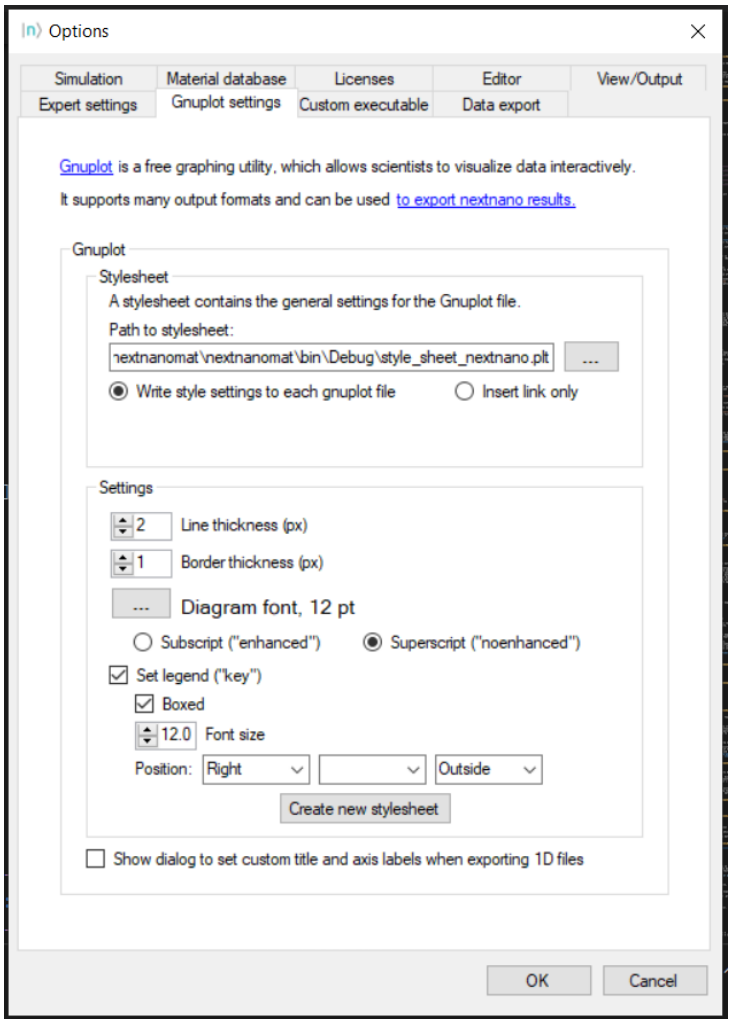


Figure 5.2.7.1: Settings for Gnuplot export and graph style.

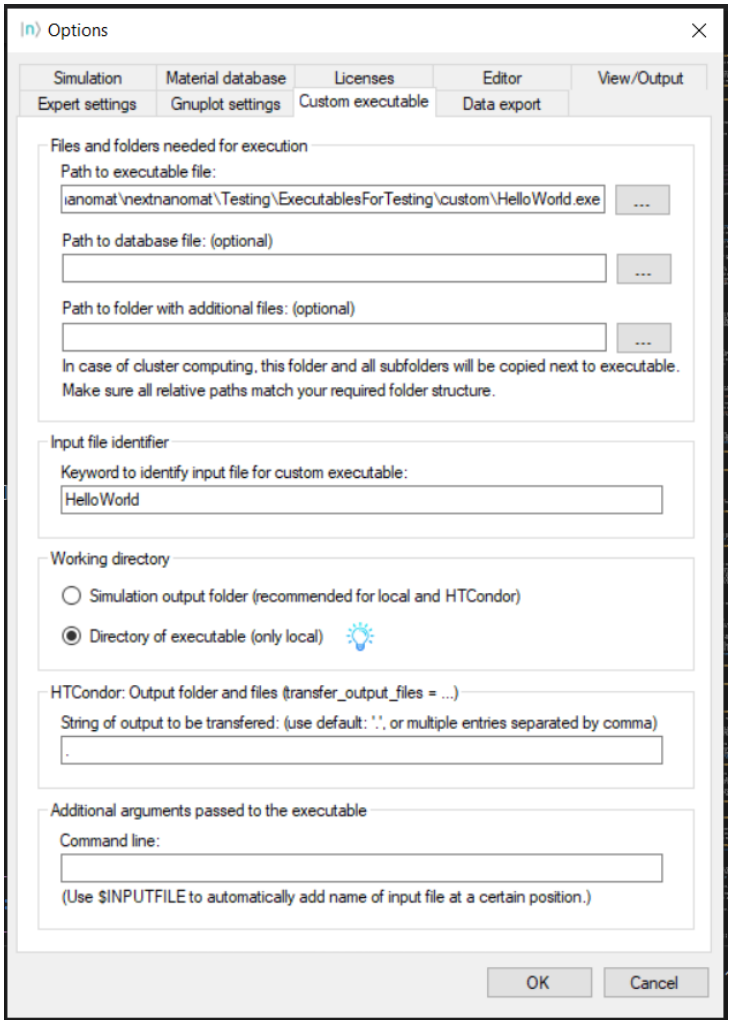


Figure 5.2.8.1: Settings for custom executable.

In case your executable does not need input files:
You still need to create some dummy files as placeholder or job submission files. The keyword can be arbitrary.

For example the *HelloWorld* executable (see Figure 5.2.8.1), which is executing a fixed script independent of input variables, can be started by submission of a text file containing `HelloWorld`. This enables the workflow manager *nextnanomat* to start and monitor the execution and to collect the results.

Optionally a path to a folder structure or the database can be defined, if the executable needs additional files during execution. Specific command line arguments can be added at the bottom of this settings tab. The variable `$INPUTFILE` will insert the filename of the actual input file at the specified position of the command line arguments.

If it does not work as wished for, we recommend to switch the flag `output additional debug info` on (Expert Seetings). The log then contains the command line arguments *nextnanomat* uses to start the application. Thus allows you to debug easily.

Example: How to run TiberCAD from nextnanomat

To run TiberCAD you need to finish its installation and put your TiberCAD license file into the respective folder. Move the example or your personal input files to a folder with write permission. Choose the following settings in the custom executable tab, Figure 5.2.8.2:

Note: A special folder has to be created, which contains all input *.tib*, geometry *.geo* and mesh *.msh* files on the same hierarchy level. The path to that folder needs to specified in the settings for the custom executable.

This enables the editing - including syntax highlighting - of the input files within *nextnanomat*. And furthermore the parallel execution of input files (see Figure 5.2.8.3) and visualization of the output.

5.2.9 Options: Data export

write the path to the ParaView executable in your computer,
e.g. *C:\Program Files\ParaView 5.6.0-Windows-msvc2015-64bit\bin\paraview.exe*

5.3 Main menu functions

Drop down functions of main menu, which do not clone a button functionality of any tab page.

5.3.1 Activate license

By clicking *Activate License* in the *Tools* menu, a license activation dialog will appear. If you have troubles finding your previously downloaded license files on your PC, you can just re-activate them. No additional licenses will be generated, you will just receive another copy of your previously generated files. The license activation procedure is documented here: *License activation via nextnanomat*

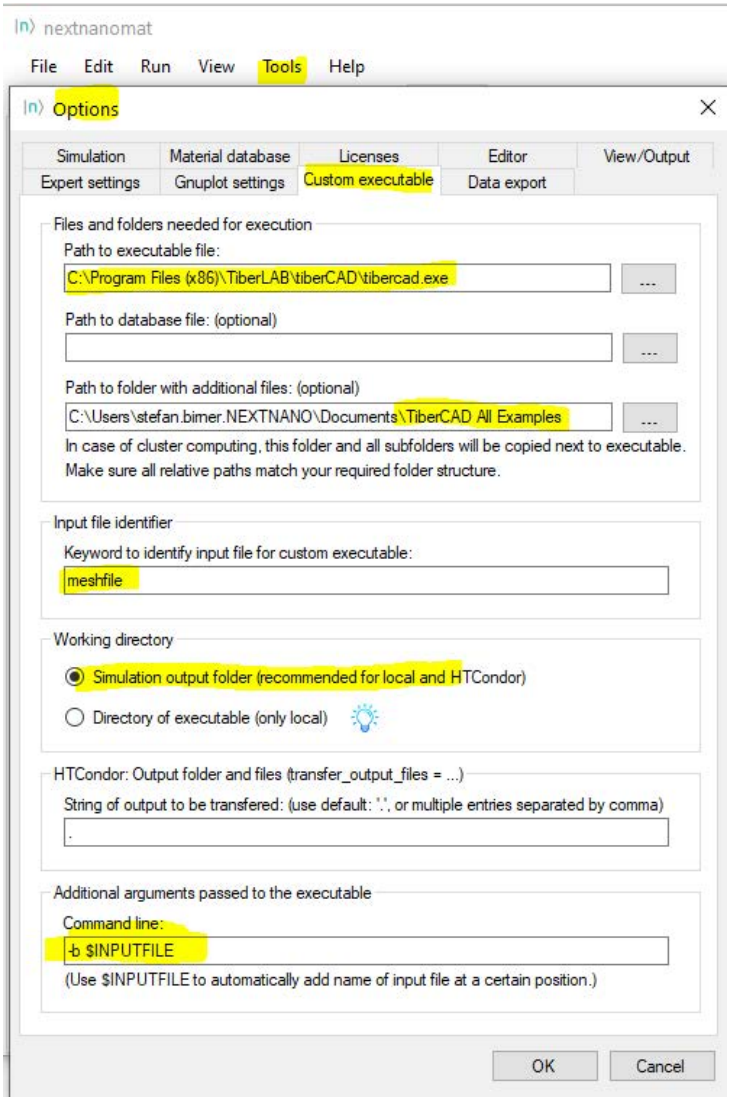


Figure 5.2.8.2: Settings to run TiberCAD as a custom executable.

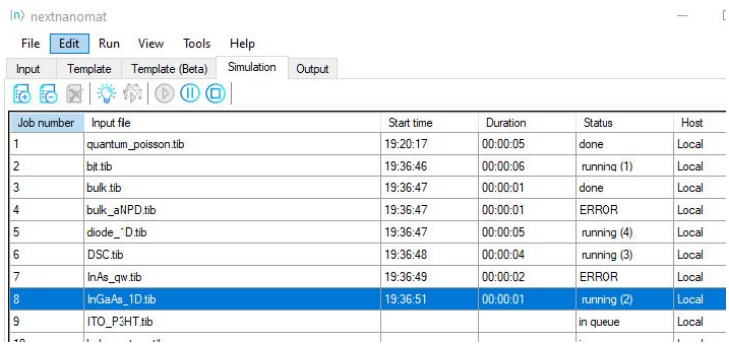


Figure 5.2.8.3: Parallel execution of TiberCAD files by the *nextnanomat* Job Manager.

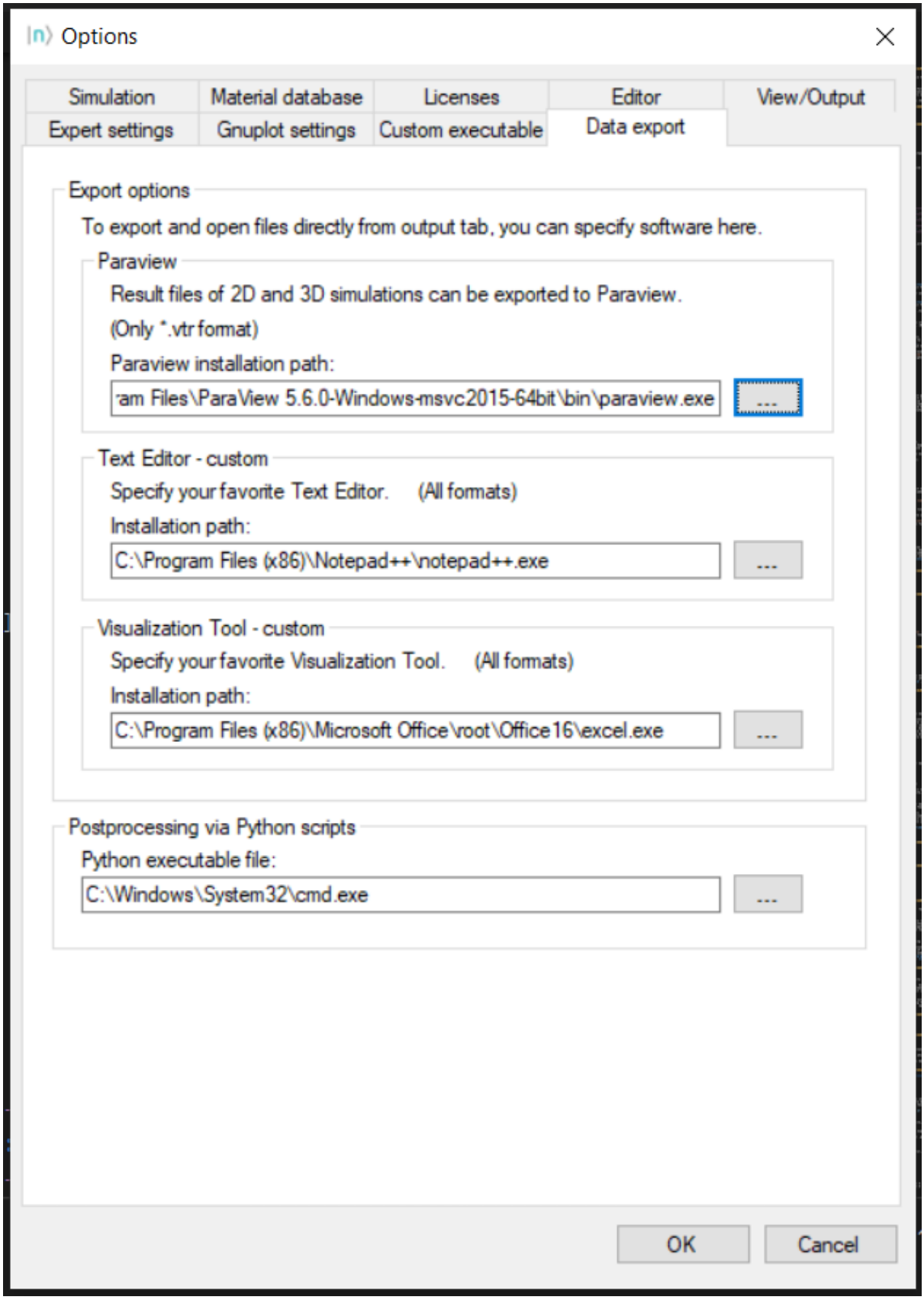


Figure 5.2.9.1: Settings for data export.

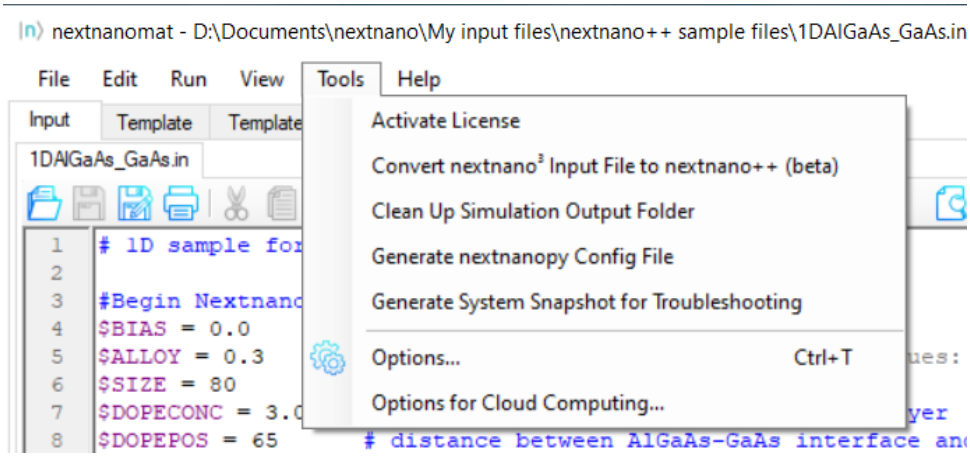


Figure 5.3.1: Tools functions

5.3.2 Convert nextnano³ input file to nextnano++

How to use this automatic conversion:

In the menu select `Tools ==> Convert nextnano³ input file to nextnano++`

When you use the automatic conversion of *nextnano³* input file into *nextnano++*, you will find that it probably does not work completely.

If you save and run the *nextnano++* input file that has been converted and that has the suffix `_nnp.in`, very likely some errors appear indicating which line(s) to change. Then some manual adjustments are needed, but the rough structure should help a lot for the conversion.

5.3.3 Clean Up Simulation Output Folder

After some months of simulating, the output folder can get quite huge. There is a feature to conveniently clean up the output folder with additional options, e.g. creating a backup of the input files. In the main menu, click on `Tools` and select `Clean Up Simulation Output Folder`. The dialog can be seen in Figure 5.3.3.1. To check multiple files at the same time, you need to select multiple files first and click on one checkbox afterwards.

5.3.4 Generate nextnanopy config file

If you want to execute a nextnano GmbH product using *nextnanopy*, a config file is needed. This file contains all relevant information and paths to the licenses, nextnano GmbH executables and material database.

To make it more comfortable for you to use both *nextnanomat* and *nextnanopy* or switch to *nextnanopy*, you can export the settings already stored within *nextnanomat* into the format needed by *nextnanopy*.

Taken from the *nextnanopy* documentation on https://github.com/nextnanopy/nextnanopy/blob/master/docs/examples/Example0_Set_up_the_configuration.rst:

> Per default the config file needs to be located at your home directory (usually `C:\Users\Your_User\.nextnanopy-config`). When you import *nextnanopy* for the first time, it will automatically generate the configuration file with few default parameters. If this file already exists, it will not modify it. You can set up this configuration file only once and you do not need to worry about it anymore, except when you renew your license or update the nextnano GmbH version.

Thus, if you want to have your custom paths as specified in *nextnanomat*, just export a config file using this function and save it into your home directory. Then *nextnanopy* will not overwrite that file. After a new installation of a nextnano GmbH update, you can easily update the paths for *nextnanopy* as well, by using this function again.

Example config file:

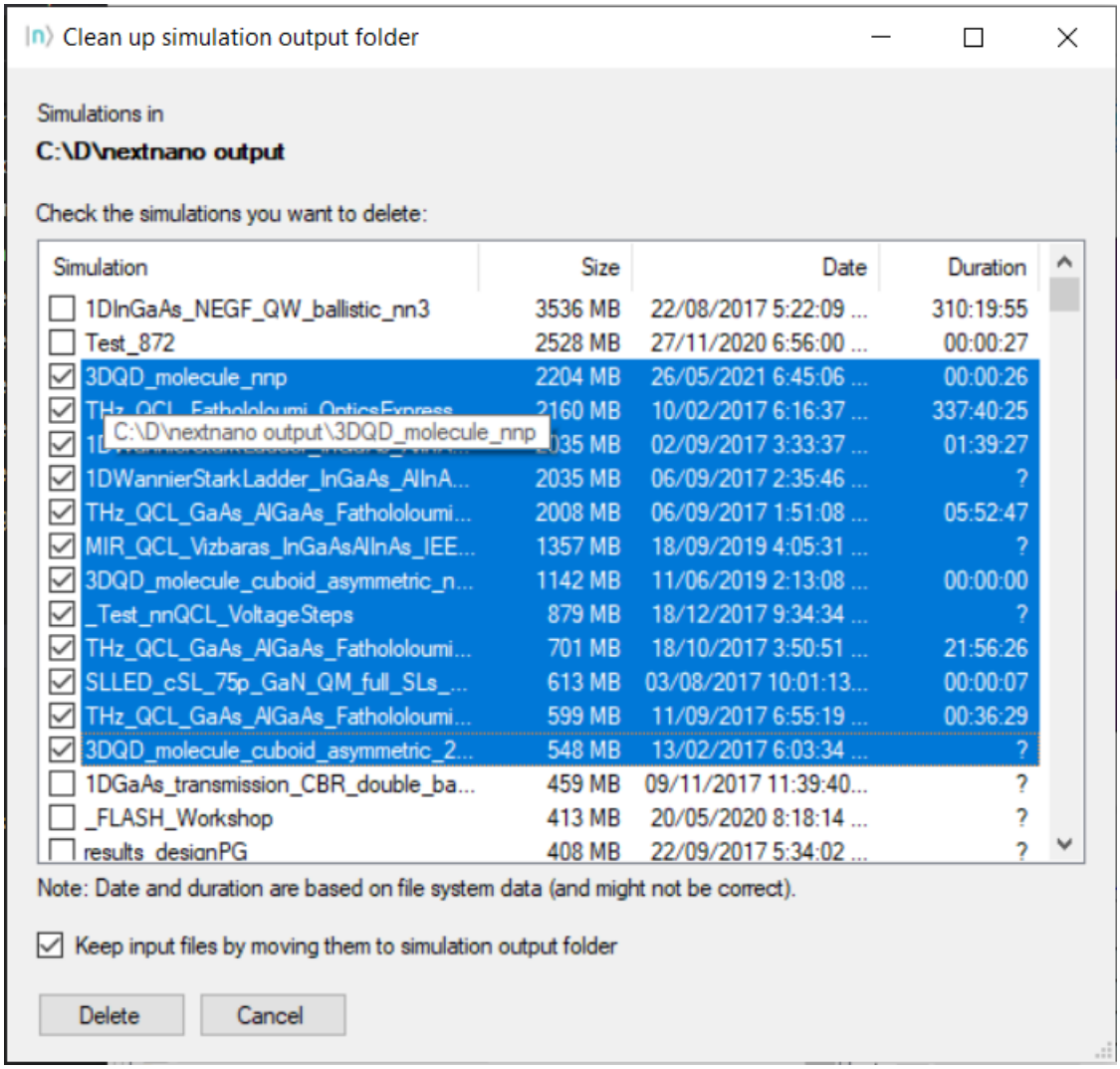


Figure 5.3.3.1: Dialog to help clean up the simulation output folder.

```
[nextnano++]
exe = C:\Program Files\nextnano\2021_12_24\nextnano++\bin 64bit\nextnano++_Intel_
↪64bit.exe
license = C:\Users\homer.simpson\Documents\nextnano\License\License_nnp.lic
database = C:\Program Files\nextnano\2021_12_24\nextnano++\Syntax\database_nnp.in
outputdirectory = D:\nextnano output
threads = 3

[nextnano3]
exe = C:\Program Files\nextnano\2021_12_24\nextnano3\Intel 64bit\nextnano3_Intel_
↪64bit.exe
license = C:\Users\homer.simpson\Documents\nextnano\License\License_nnp.lic
database = C:\Program Files\nextnano\2021_12_24\nextnano3\Syntax\database_nn3.in
outputdirectory = D:\nextnano output
threads = 0
debuglevel = -1
cancel = -1
softkill = -1

[nextnano.NEGF]
exe = C:\Program Files\nextnano\2021_12_24\nextnanoNEGF_2020_06_22\nextnano.NEGF\
↪nextnano.NEGF.exe
license = C:\Users\homer.simpson\Documents\nextnano\License\License_nnQCL.lic
database = C:\Program Files\nextnano\2021_12_24\nextnanoNEGF_2020_06_22\nextnano.NEGF\
↪Material_Database.xml
outputdirectory = D:\nextnano output
threads = 3

[nextnano.MSB]
exe = C:\Program Files\nextnano\2021_12_24\nextnano.MSB\nextnano.MSB\x64\nextnano.MSB.
↪exe
license = C:\Users\homer.simpson\Documents\nextnano\License\License_nnMSB.lic
database = C:\Program Files\nextnano\2021_12_24\nextnano.MSB\nextnano.MSB\Materials.
↪xml
outputdirectory = D:\nextnano output
debug = 0
```

5.3.5 Generate System Snapshot for Troubleshooting

In the menu, you can either find it under 'Tools' -> 'Generate System Snapshot for Troubleshooting' or under 'Help' -> 'Generate Troubleshooting Snapshot'.

By clicking this function, a text file - containing most relevant information for debugging - will be created. Save this file and send it to the nextnano GmbH Support Team to assist you.

5.4 Color maps

On this page you will find some background information about scientific visualization of data. Additionally, the color map choices and customization options of the nextnano software are explained. The references and sources used to generate these maps can be found in the last section.

5.4.1 Introduction

Color maps for scientific software need to intuitively represent data without visual distortion.

5.4.2 Implementation

5.4.3 References

5.5 Export Functionalities

5.5.1 Exporting to ParaView

ParaView is a free graphing utility specialized for 2D and 3D visualization. It can be downloaded from here: ParaView To be able to export results in *nextnanomat* directly to ParaView specify the Executable file in Tools - Options:

Setting Up nextnanomat

For the exercises we will require two different settings in *nextnanomat* (our user interface).

Direct Exporting

The first setting is the simplest one and it is used to open the selected .vtr file in *nextnanomat* into ParaView. It will require to follow the next steps:

1. In the menu bar of *nextnanomat*, click on `Tools :math:`\Rightarrow` Options :math:`\Rightarrow` Data Export`
2. Provide the path of the executable of ParaView in your computer in the field `ParaView installation path` (e.g. *C:Program FilesParaView 5.11.1binparaview.exe*).
3. Click `OK`

- In *Options: Data export*, write the path to the ParaView executable in your computer, e.g. `C:\\Program Files\\ParaView 5.6.0-Windows-msvc2015-64bit\\bin\\paraview.exe`

Exporting with Python Scripts

The second setting is used for launching ParaView through Python scripts. As we will see in the next examples, this is very useful for post-processing several input files, including setting up overlayed plots. For this setting it will be require to follow the same steps as before, but not including the path of the executable of a terminal window (*cmd.exe* for Windows, as example) in the field `Python executable file`.

Figure 5.5.1.1 presents the final result provided the required information.

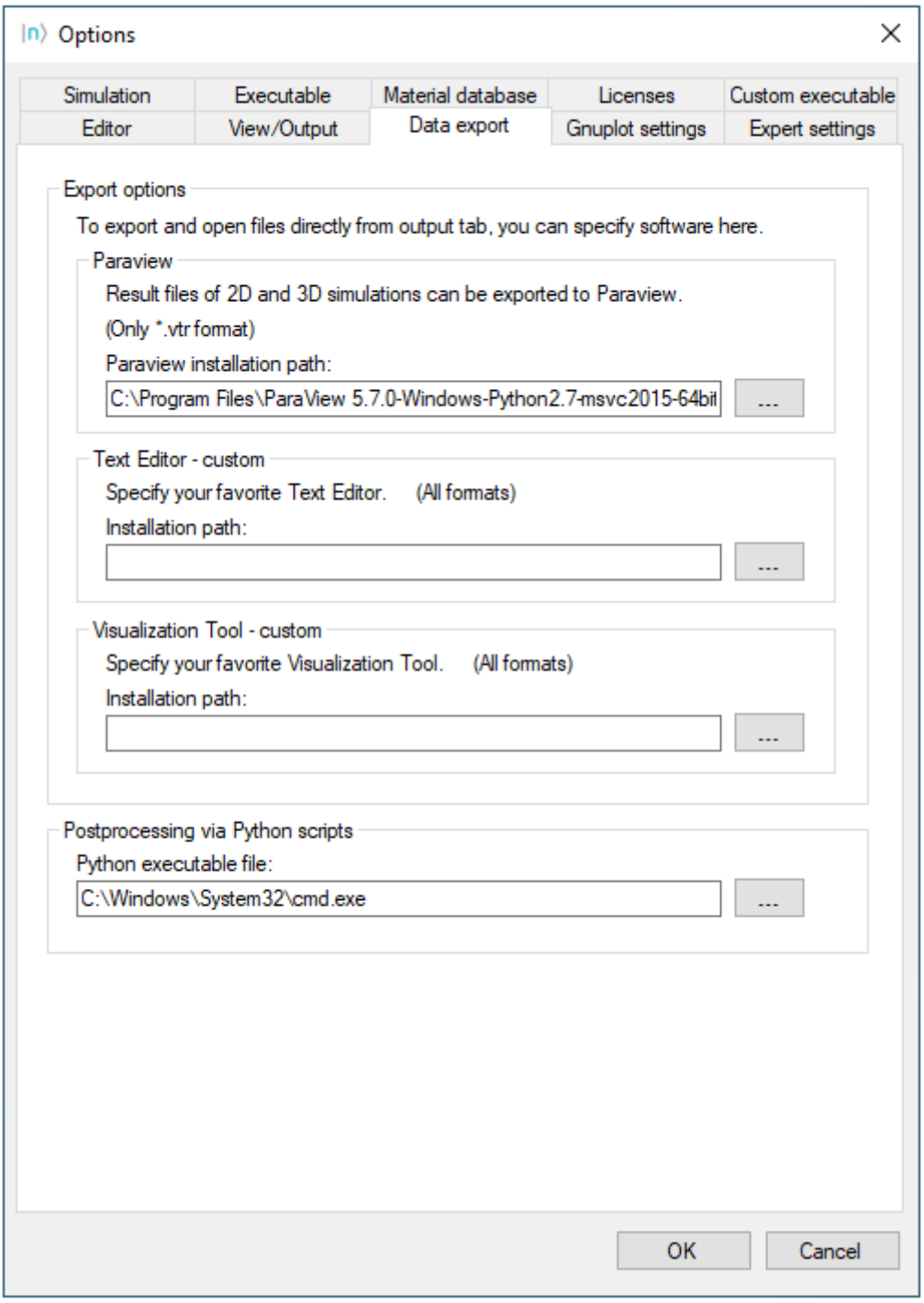


Figure 5.5.1.1: Required settings for exporting .vtr files from *nextnanomat* to ParaView in this tutorial. This menu is accessed clicking in `Tools :math:`\Rightarrow` Options :math:`\Rightarrow` Data Export`, within *nextnanomat*.

Installing the Python scripts in your Computer

Copy the three python scripts shown above in a folder and create an empty file called *__init__.py*. This last file is required in order Gui5.py be identified as a package by Python.

A more powerful resource in the integration of nextnano GmbH and ParaView is the possibility to incorporate Python scripts in the post-processing of the simulation data. For performing the next exercises, we will use the second setting described above, where ParaView will be called from a command line in a terminal application (for example, *C:Program Filescmd.exe* in Windows).

Actually, we will make use of three scripts:

- *Paraview_postprocessing.py*, that contains all post-processing tasks to be performed on each file exported
- *dialog_window.py*, that will capture information of *nextnanomat* and to launch a new window for selecting files to be exported, and predefine some global properties of the plots. It is the actually the one that will launch ParaView and the post-processing script.
- *Gui_5.py*, is a simple script for generating the window for selecting files, described before. In order to be recognized as a package to be imported by the previous script, it is required to create an empty file called *__init__.py* in the same folder where this script will be stored.

Copy them in on your computer. All scripts shall be stored in the same directory.

Direct Export

1. Specify path to ParaView in Tools - Options if you have not done so before
2. **Select a 3D .vtr file of the simulation output folder and click on the `Export and open in specific format` button.**
   Choose `Open File with ParaView` (ParaView will open automatically.)
3. Within ParaView:
   a. (The selected file should be highlighted automatically.) Click on `Apply`.
   b. Other settings:
      I. Representation: Surface
      II. Cell/Point Array Status: Choose the array to be displayed. For example, one can display the file `bandedges.vtr` and choose the array `Gamma` that corresponds to the Gamma conduction band. Click on `Apply`.
      III. Coloring: Choose the array to be displayed. (`Gamma`, for example)

Now you can play with the tool. Rotating, changing opacity and adding filters. ParaView is a very rich tool, hence it requires some time to explore all its capabilities. We recommend investing some time to learn about its filters:

- First, the file should be highlighted and then click on `Filter` ⇒ `Alphabetical`.

  Some interesting ones are:

  – Edit cells by region
  – Calculator
  – Contour

Using `File` ⇒ `Save State`, the values of all variables can be stored on your PC. These states can be reloaded, which is useful for the generation of scripts.

For further information, we recommend a good tutorial from TACC.

When to use: This single-click-procedure is very convenient when we want to load a single file in the Pipeline Browser of ParaView. Having already experience with this tool it will require some basic steps to display the data on the screen of this platform.

Exporting 2D Outputs with Python Scripts

Input Files:

- *2DQuantumCorral_nnp.in* from the tutorial *Electron wave functions in a cylindrical well (2D Quantum Corral)*

Scope of the tutorial:

- Export 2D vtr files from *nextnanomat* to ParaView
- Choosing predefined filters in ParaView
- Preset of some parameters for the plots
- Defining overlay files
- Dealing with files with mixed units

Introduced Keywords:

- `output{ format2D }`
- `output{ format3D }`

Relevant output Files:

- *\bias_00000\bandedges.vtr*
- *\bias_00000\Quantum\probabilities_quantum_region_Gamma_.vtr*
- *\bias_00000\Quantum\probabilities_shift_quantum_region_Gamma.vtr*

Scripts:

- *nn_ParaView_Integration.py*
- *nn_ParaView_GUI.py*
- *nn_ParaView_Plotter.py*

ParaView supports the use of Python scripts for loading and displaying objects in its `Pipeline Browser`. In the next tutorials we will present how to perform this integration using nextnano GmbH.

Our scripts come to assist not only to load the results of the simulations in this platform, but also allow to preset the application of some filters and some global settings even before launching the program.

Setting up nextnanomat for exporting .vtr files using Python scripts

In order to load files in ParaView through Python scripts from *nextnanomat* it will require the following settings:

1. In the menu bar of *nextnanomat*, click on `Tools >> Options >> Data Export`
2. Provide the path of the executable of ParaView in your computer in the field `Paraview installation path` (e.g. *C:\Program Files\ParaView 5.11.1\bin\paraview.exe*).
3. Provide the path of the executable of a terminal window in the field `Python executable file` (e.g. *C:Windows\System32\cmd.exe*).
4. Click on `OK`

Figure 5.5.1.2 presents the final result provided the required information.

These scripts are very useful for post-processing several data files, specially for setting up overlayed plots as we will see below.

Scripts and files of this tutorial

In the nextnano GmbH package you will find all necessary files we will use in this tutorial. The scripts for the integration are in the folder *Scripts\ParaView*.

The first (*nn_ParaView_Integration.py*) captures information of *nextnanomat* and launches the graphical user interface shown in Figure 5.5.1.3 using the script *nn_ParaView_GUI.py*. After pressing the `Launch ParaView` button, the script *nn_ParaView_Plotter* is copied to the simulation folder and modified with all information captured in the user interface. Automatically *nextnanomat* will launch ParaView.

ParaView supports the use of Python scripts for loading and displaying objects in its `Pipeline Browser`. In the next topics, we will illustrate some examples how this can be done.

The input file *2DQuantumCorral_nnp.in* used in this tutorial can be found in the folder *\Sample files\nextnano++ sample Files* within the nextnano GmbH package. This corresponds to a two-dimensional simulation and, for now, we are interested in the results of the quantum computations.

In the section `output{ }` of the input file we can specify the format of these files whose default is `AvsBinary_one_file`, for the case of the 2D and 3D plots, because being less memory demanding that the other available options. Nevertheless, ParaView does not support this format. In this way, for exporting files from nextnano GmbH into ParaView it is required that the results be coded as VTK file. For this reason it is important to specify in the input file

```
``format2D = VTKAscii (for 2D plots)``
```

or

```
``format3D = VTKAscii (for 3D plots)``
```

as we can observe in the example.

For making our discussion more interesting we will modify the input file *2DQuantumCorral_nnp.in* reducing some dimensions, and adding more information about the band edges, as illustrated below.

```
8     $RADIUS_CORE = 4.0                                                    # Core␣
↪radius
9     $RADIUS_SHELL = 6.0                                                   #␣
↪Shell radius
```

```
15    $GRID_SPACING1 = 0.5                                                  #␣
↪(DoNotShowInUserInterface)
16    $GRID_SPACING2 = 0.5                                                  #␣
↪(DoNotShowInUserInterface)
```

(continues on next page)

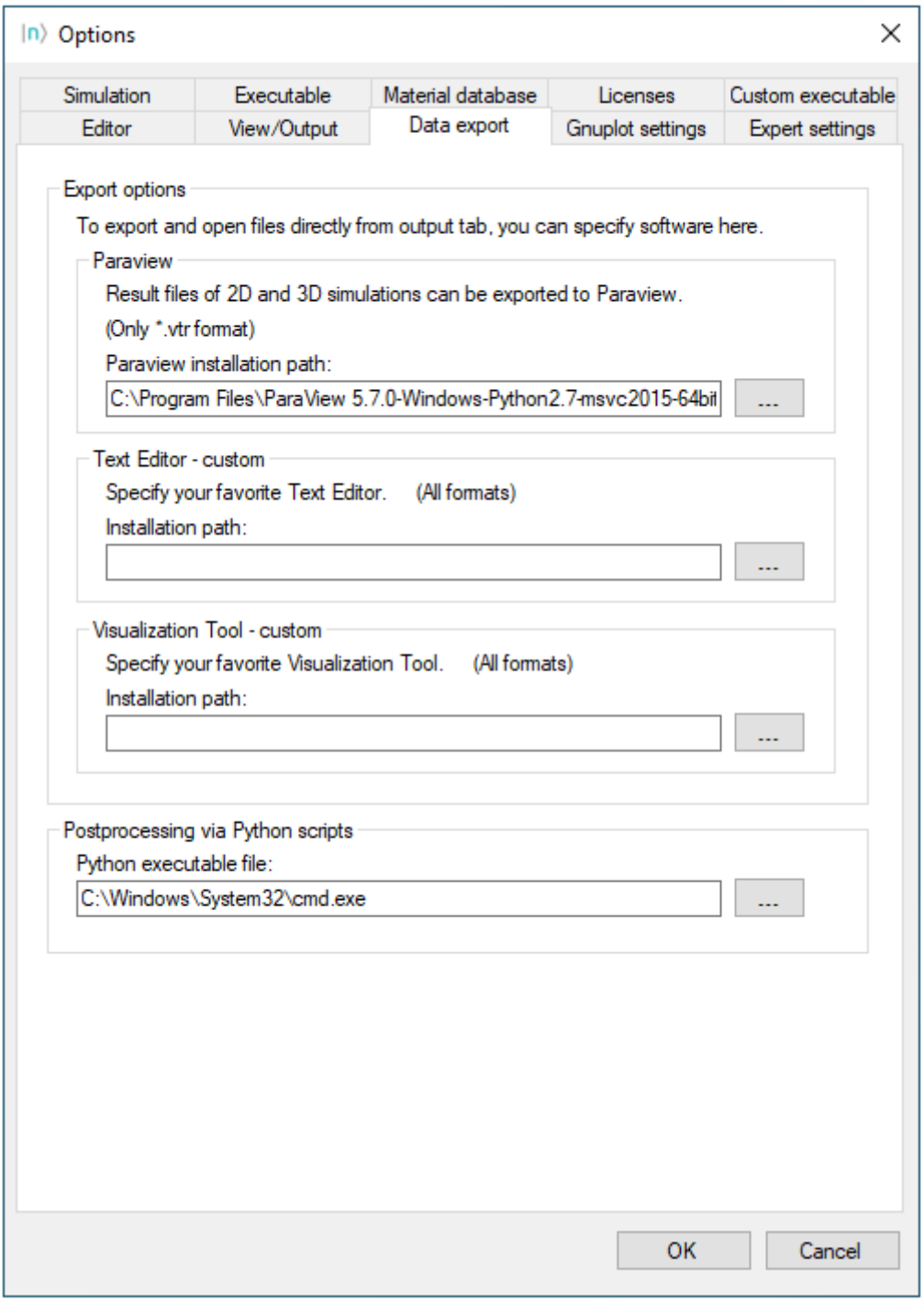


Figure 5.5.1.2: Settings in *nextnanomat* for launching ParaView using Python scripts

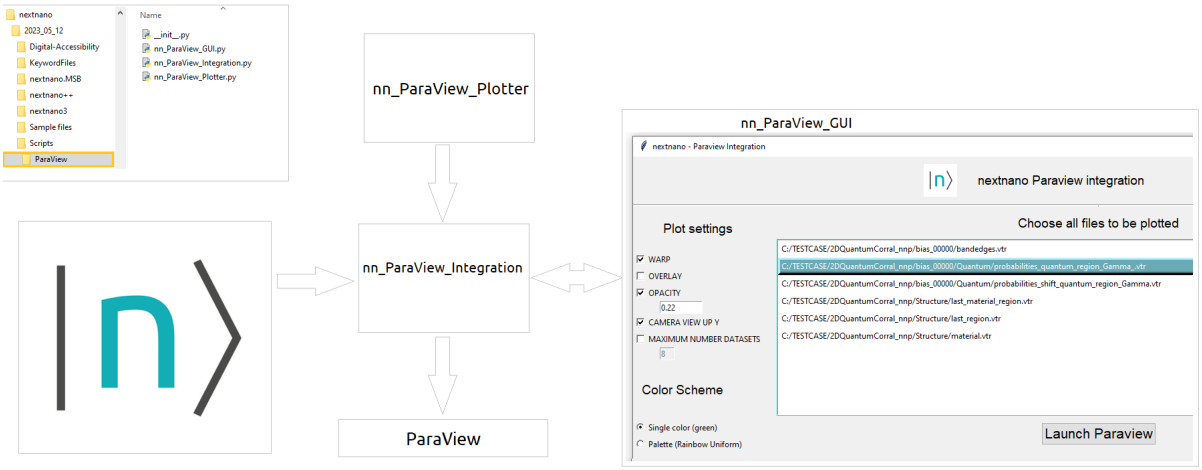


Figure 5.5.1.3: Scripts for integration of *nextnanomat* and ParaView and the interations between the components.

(continued from previous page)

```
    $GRID_SPACING3 = 0.5 / 3                                                    #␣
↪(DoNotShowInUserInterface)
    #-------------------------------------------------------------------------#

    #-------------------------------------------------------------------------#
    # derived parameters                                                      #
    #-------------------------------------------------------------------------#
    $START_x = $CENTER_x - $RADIUS_SHELL - 1.0                                  #␣
↪(DoNotShowInUserInterface)
    $START_y = $CENTER_y - $RADIUS_SHELL - 1.0                                  #␣
↪(DoNotShowInUserInterface)
    $END_x   = $CENTER_x + $RADIUS_SHELL + 1.0                                  #␣
↪(DoNotShowInUserInterface)
    $END_y   = $CENTER_y + $RADIUS_SHELL + 1.0                                  #␣
↪(DoNotShowInUserInterface)
```

```
    classical{
        Gamma{}
        HH{}
        LH{}
        SO{}
        output_bandedges{}
    }
```

Let us simulate the modified input file using *nextnanomat* and observe the results in the output folder. Suppose that we are interested to plot the file *probabilities_quantum_region_Gamma_.vtr* ` present in the *Quantum* folder. It represents the probability density of several states of our device. Selecting the plots in the menu at the bottom right corner of *nextnanomat* we can display the beautiful set of patterns for this geometry. Let us choose one of them – the plot `Psi^2_14`, as example.

After that, the whole file (not only the selected plot) will be loaded in the `Pipeline Browser` of ParaView, but no other task will be performed. This is what we called before as the direct method. Let us see how we can improve this integration.

Now select the file to be exported (a .vtr file), click in the icon `Export and Open in specific Format` and select `Open File with Paraview`, as shown in Figure 5.5.1.4.

After that the whole file (not only the selected plot) will be loaded in the `Pipeline Browser` of Paraview, but no other task will be performed. This is what we called before as the direct method. Let us see how we can improve

this integration.

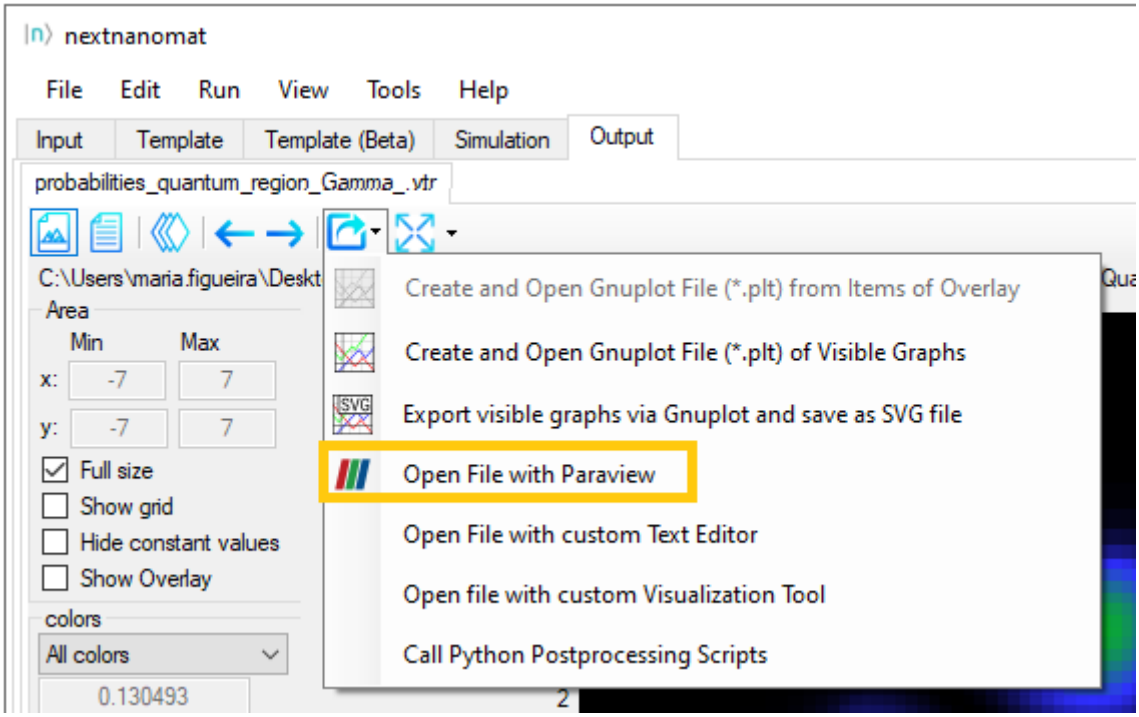


Figure 5.5.1.4: Button for exporting datafiles to ParaView using the direct method.

Single plot

Now let us check how the integration is performed using these scripts.

In nextnanomat

Let us return to *nextnanomat* and select some .vtr file as we did it before (*\Quantum\probabilities_quantum_region_Gamma.vtr*). Click on the icon `Export and Open in specific Format` and select `Call Python Postprocessing scripts`, as shown in Figure 5.5.1.5. Within the *\Scripts\ParaView* folder in the nextnano GmbH package select the script *nn_ParaView_Integration.py*.

In the Integration GUI

A new panel will be opened with a list of all .vtr files of the simulation folder. By default, the selected item in the list of files to be plotted corresponds to the file that was initially selected in *nextnanomat* when launching the Python script. For deselecting it, click once on its file name. At least one file must be chosen in order to open ParaView. For selecting more than one file, just click on it (not need to press the `CTRL` key).

At the left of this window you can find some plot settings that can be chosen, and they correspond to general settings. In another words: all selected files to be plotted will be affected by these general settings.

Instead of describing all these settings one by one in this section of the tutorial, it is more convenient to introduce them with practical examples. Let us start with a simple transformation (in ParaView usually called `Filter`) on the original 2D plots. We will select `WARP` in the menu for creating a 3D plot, where the third axis corresponds to probability density at each (x,y) position of the plane. Additionally, we will select `OPACITY` and `Single color` as color scheme. Click on the button `Launch ParaView`. The final result of applying these settings is shown in Figure 5.5.1.6.

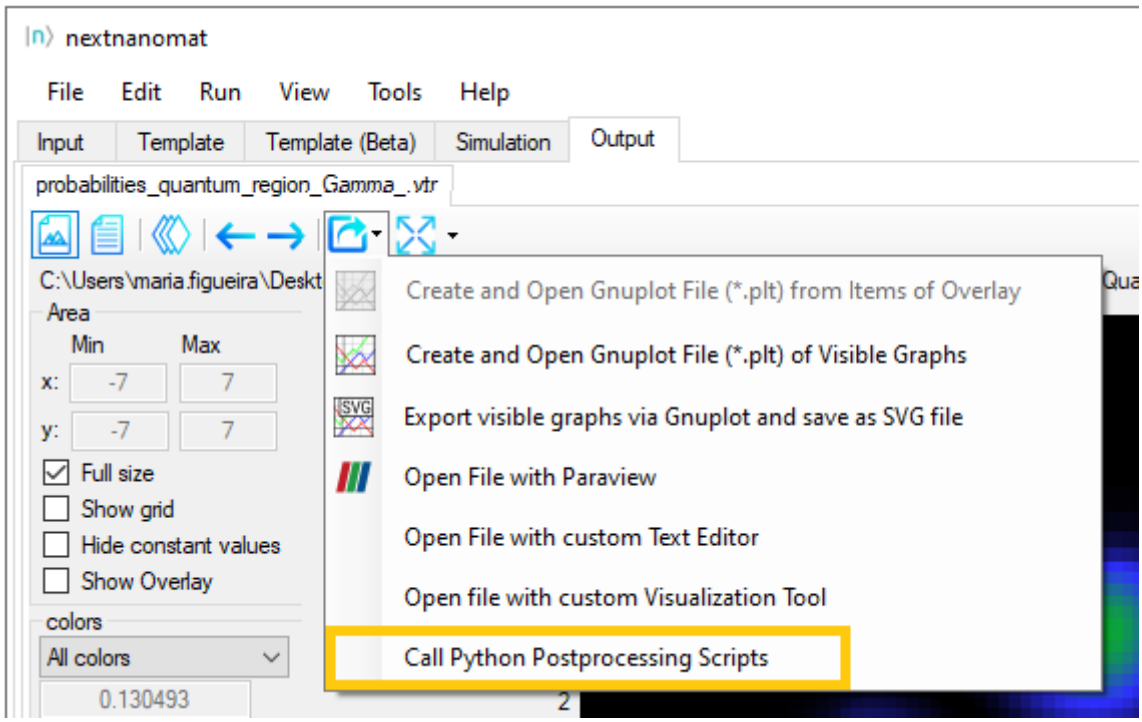


Figure 5.5.1.5: Button in *nextnanomat* for exporting datafiles to ParaView using Python Scripts.

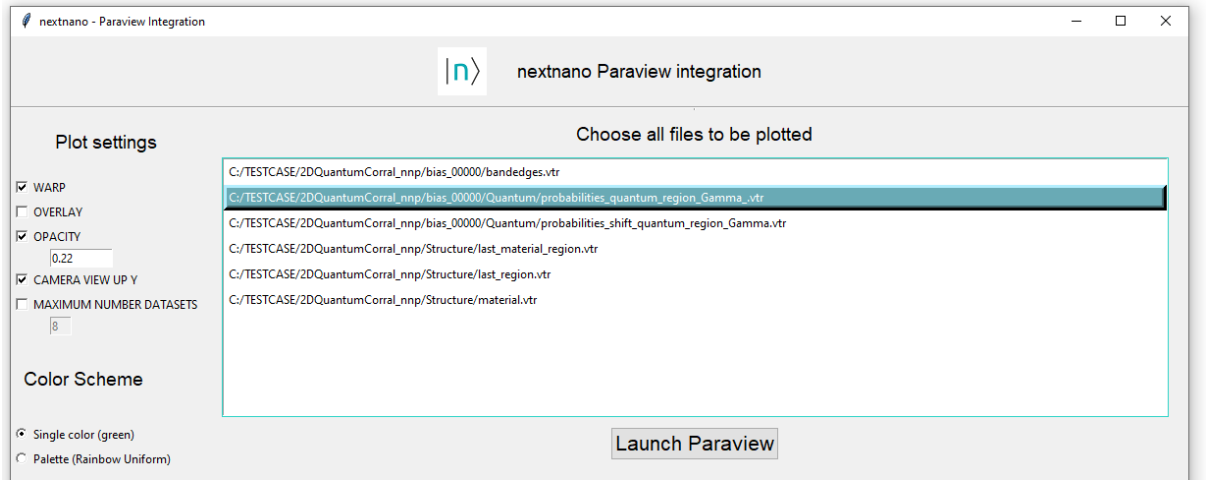


Figure 5.5.1.6: Settings in Graphical user interface for integration of *nextnanomat* and ParaView for the case of single plot.

In ParaView

Now we can observe a larger number of plots is presented in the `Pipeline Browser`. The first object has the same name of the absolute path of the file plotted. Below it, each column (dataset) of the file is plotted with corresponding sequence of post-processing tasks and settings defined previously in the interface.

As we can see in the next images, they were plotted in a single color and semi-transparent.

The Figure 5.5.1.7 we can observe that for this example, where the option `WARP` was selected, the original 2D representation for this probability density function (`Psi^2_1(nm^-2)`, `Psi^2_2(nm^-2)`,...) will be presented in colors, while its "warped" version (`Psi^2_1_warped`, `Psi^2_2_warped`,...) will be displayed as semi-transparent surfaces.

The warped objects are the result of applying the `WarpByScalar` filter of ParaView to each component, that converts 2D functions in a 3D representation, where the third axis, in this case, corresponds to the value of probability density at each (x,y) in the plane.

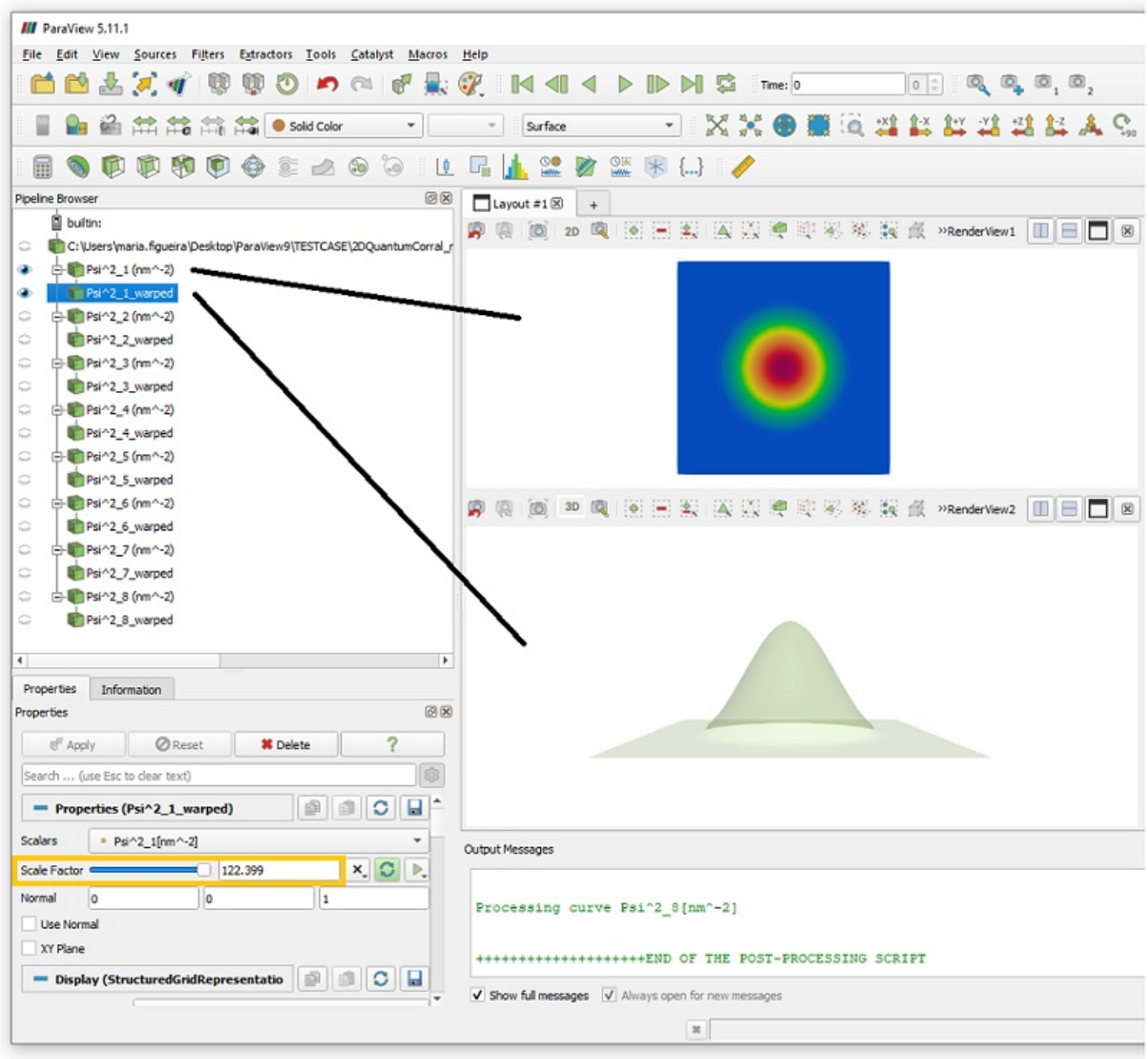


Figure 5.5.1.7: Organization of the files in the Pipeline Browser of ParaView. The imported datafile is registered with the absolute path of the file. For each component of the datafile, the 2D map and the warped version are presented. Highlighted in orange is the value estimated for the `Scale Factor` for the `WarpByScalar` filter.

The `Scale Factor` used for the warped plot is estimated by the script, and it can be dynamically changed within ParaView, moving the corresponding slider in the `Properties` tab or filling a numeric value in front of it. Let

us try Choosing in ParaView some of the warped files, write the number `300` after the slider, and press the button `Apply`. Looking at the other warped plots of this file, you will observe that this change was applied also to them. This is an important feature implemented by our script and only applies to the `Scale Factor`. Figure 5.5.1.8 shows one example how the interconnection of the components react when the `Scale Factor` of any component is changed.

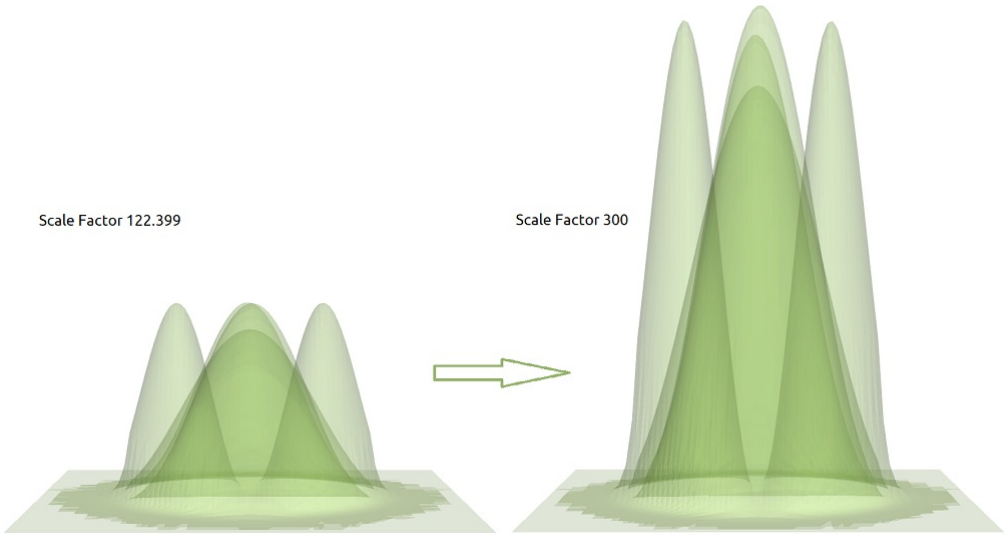


Figure 5.5.1.8: The first three lower probability density functions of electrons in the Gamma band using two different scale factors in the `WapByScalar` filter in ParaView. The scale factors of the objects of the same datafile are connected: changing the value in one component will automatically update the `Scale Factor` of the other warped plots.

Looking at the `Information` tab under the `Pipeline Browser` of ParaView (Figure 5.5.1.9) see a larger number of datasets was actually exported, but only 8 are displayed. This is because we limited to a maximum of datasets to be displayed through the variable `MAXIMUM_NUMBER_DATASETS` as default in the *nn_ParaView_Plotter* script. This limitation is used to avoid data explosion in ParaView. Nevertheless, this number can be changed by modifying the value of `MAXIMUM_NUMBER_DATASETS` in the user interface for nextnano GmbH-ParaView integration, as we will do in the next section.

Hint: The presets captured in the user interface for integration (Figure 5.5.1.6) aims to automate the configuration of each object to be displayed in the `Pipeline Browser`. Nevertheless they can be dynamically changed in this platform. Figure 5.5.1.9 shows two examples of plot controls that can be used to active and desactivate plots (the eyeball) and to adjust the camera for a plot with perfect alignment of the plot to the z-axis.

Clicking on the eyeballs in ParaView, each component can be activated or hidden individually. In Figure 5.5.1.11 the eight lowest probability functions for the electrons in the Gamma band are displayed.

Hint: Use semi-transparent surfaces for displaying superposition of two or more states, as presented in Figure 5.5.1.12

When to use the single plot method: This method is ideal for the case you want to export a single plot, using the standard parameters and/or simple filters of the script. In this example we need only one click for requesting the `WarpByScalar` filter and another to launch the tool. When not changing any plot setting, no other clicks are necessary.

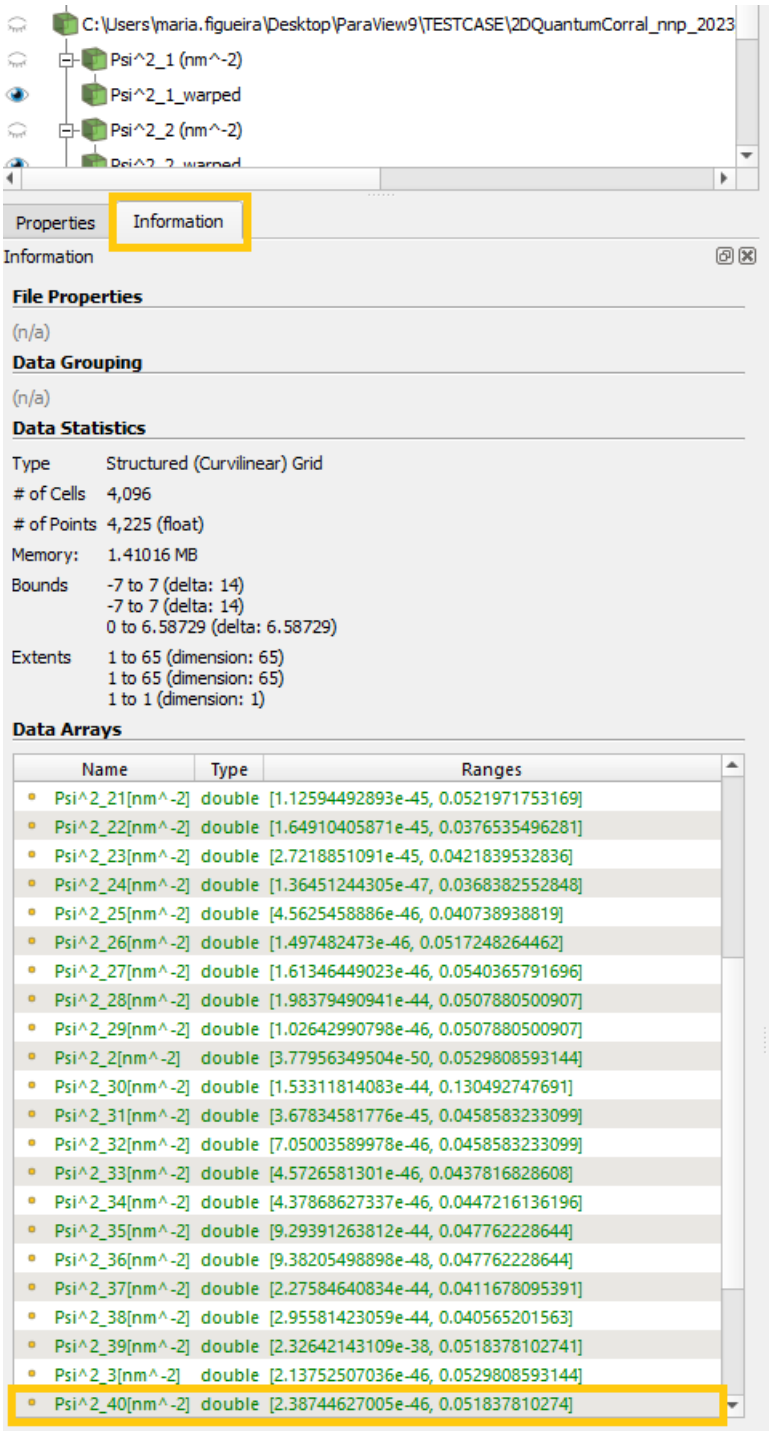


Figure 5.5.1.9: `Information` tab in ParaView showing the all sets within the datafile. For avoiding data explosion, only a reduced number were actually processed setting the variable `MAXIMUM_NUMBER_DATASETS` to 8 in the Python scripts.

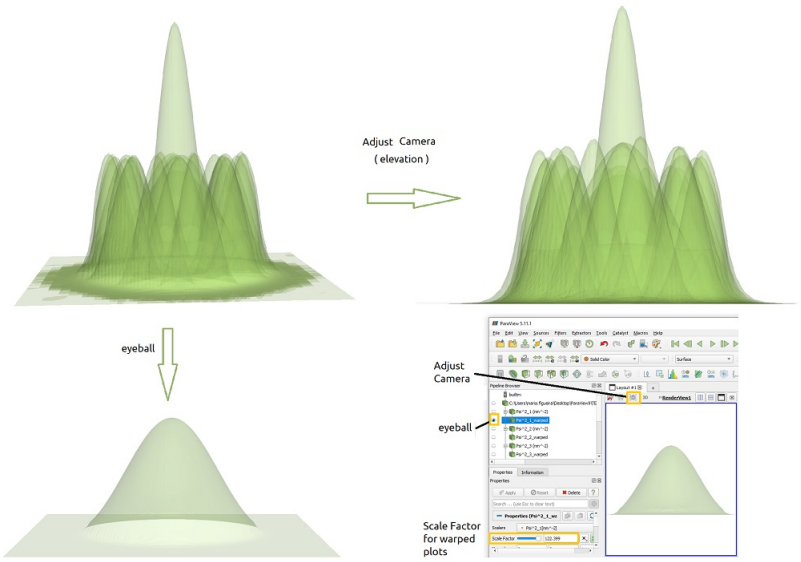


Figure 5.5.1.10: Plot controls in ParaView for selection of the component to be displayed (eyeball) and adjust of the camera. Moving the slider of `Scale Factor` the height of the peaks can be modified.

Plot with overlay

Displaying overlayed images represents a powerful feature in *nextnanomat*. Our script for integration is capable to setup all ressources to reproduce overlay of warped-3D plots also in ParaView. We will demonstrate how to do it plotting the file *\Quantum\probabilities_shift_quantum_region_Gamma.vtr* overlayed with the *bandedges.vtr*.

In nextnanomat

We will repeat the same procedure as before following the next steps:

- select a .vtr file of the simulation within *nextnanomat*
- click on the icon `Export and Open in specific Format` and select `Call Python Postprocessing scripts`
- select the script *nn_ParaView_integration.py*
- select the file: *\Quantum\probabilities_shift_quantum_region_Gamma.vtr*

In Integration GUI

This time we will make the following selections:

- `MAXIMUM_NUMBER_DATASETS` and the number `5` (The box shall be checked in order to capture this number)
- `WARP`
- `OVERLAY`
- Color scheme: `Palette`

Once the option `OVERLAY` is checked, a list of .vtr files is displayed. Choose, for this example, the file *\bias_00000\bandedges.vtr*. In this current implementation only one file shall be selected as overlay. Press the button `Launch ParaView`. Figure 5.5.1.13 shows the final configuration.

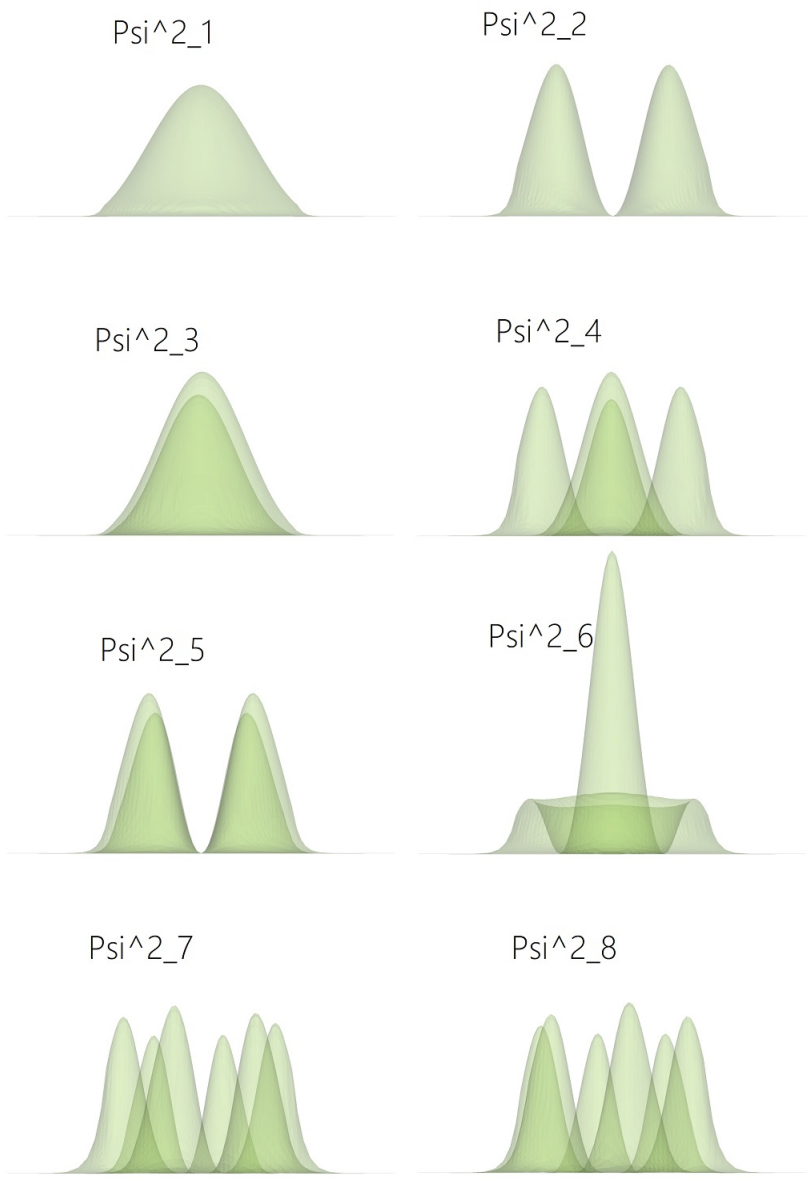


Figure 5.5.1.11: Probability density function of the first lower states of the electrons in the Gamma band.

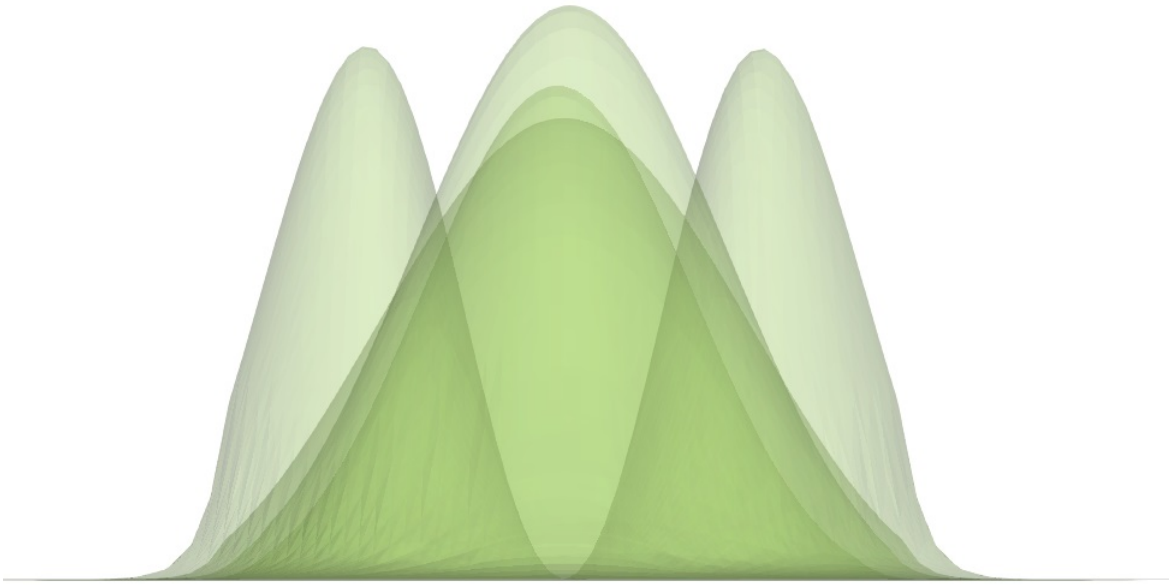


Figure 5.5.1.12: Use of semi-transparent surfaces is advantageous for comparing two functions, as in this example.

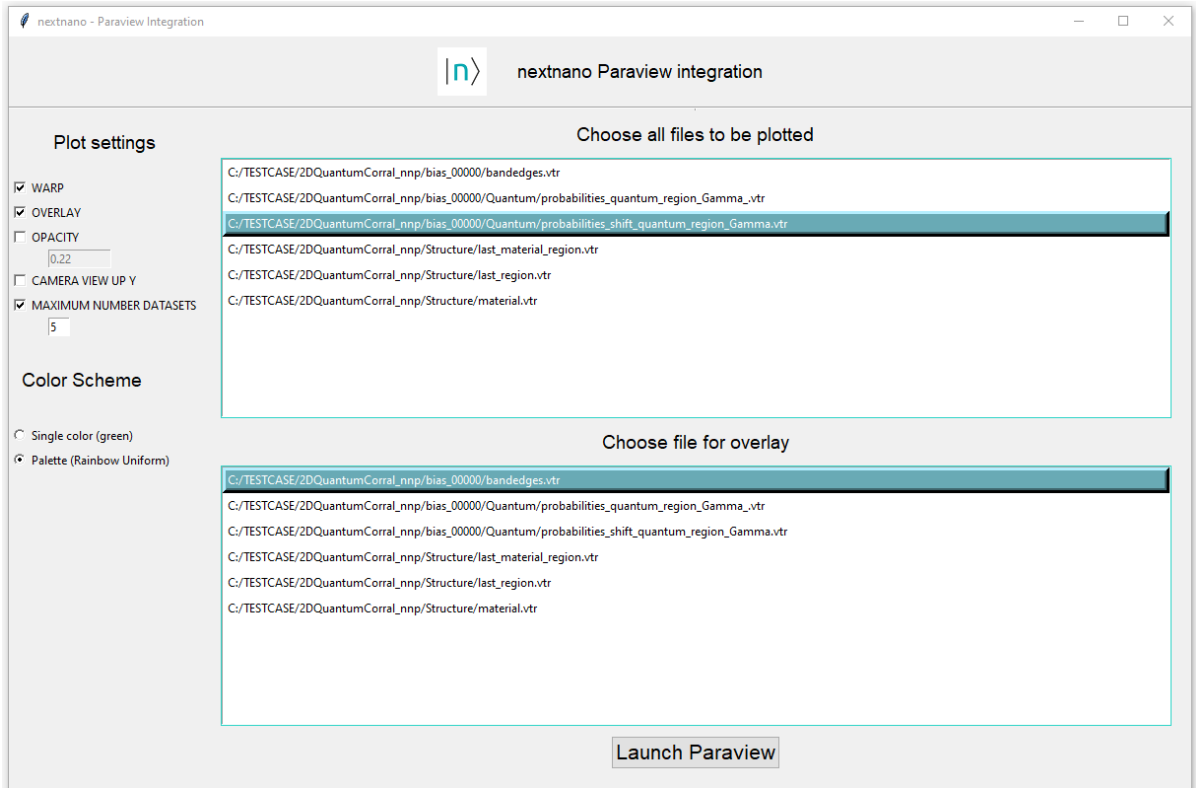


Figure 5.5.1.13: Settings in Graphical user interface for integration of *nextnanomat* and ParaView for the case of overlayed plots.

In ParaView

After the message `END OF THE POST-PROCESSING SCRIPT` is displayed in ParaView, we will observe that two group of files are presented: the one corresponding to the overlay (the band edges) and the other the main plot (shifted probabilities). Now only 5 components are shown for each file.

Once again the 2D plots were warped (option `WARP`). Now they are presented using the `Rainbow Uniform` palette and they are not longer with the Z-axis aligned vertically. With a simple click in ParaView, they can be realigned vertically as we can see in Figure 5.5.1.14.

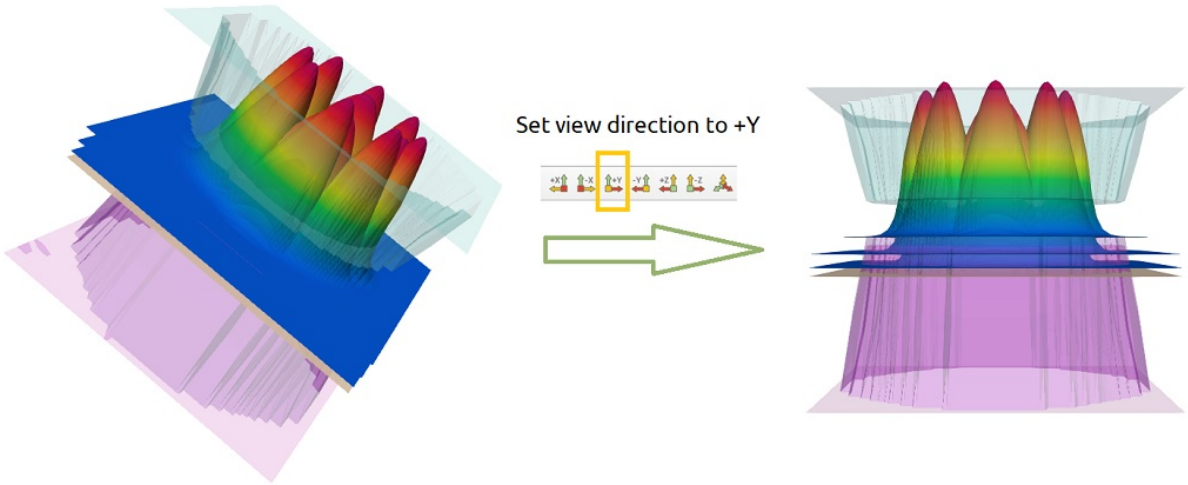


Figure 5.5.1.14: Plot of the first five lowest probability density functions. Clicking on the `+Y` button in ParaView they can be realigned vertically

The warped plots of the overlay files are always shown at the top of the pipeline as a semi-transparent surface using a different palette. The only exception opaque surface for the overlay corresponds to the electron and hole Fermi levels (see Figure 5.5.1.15).

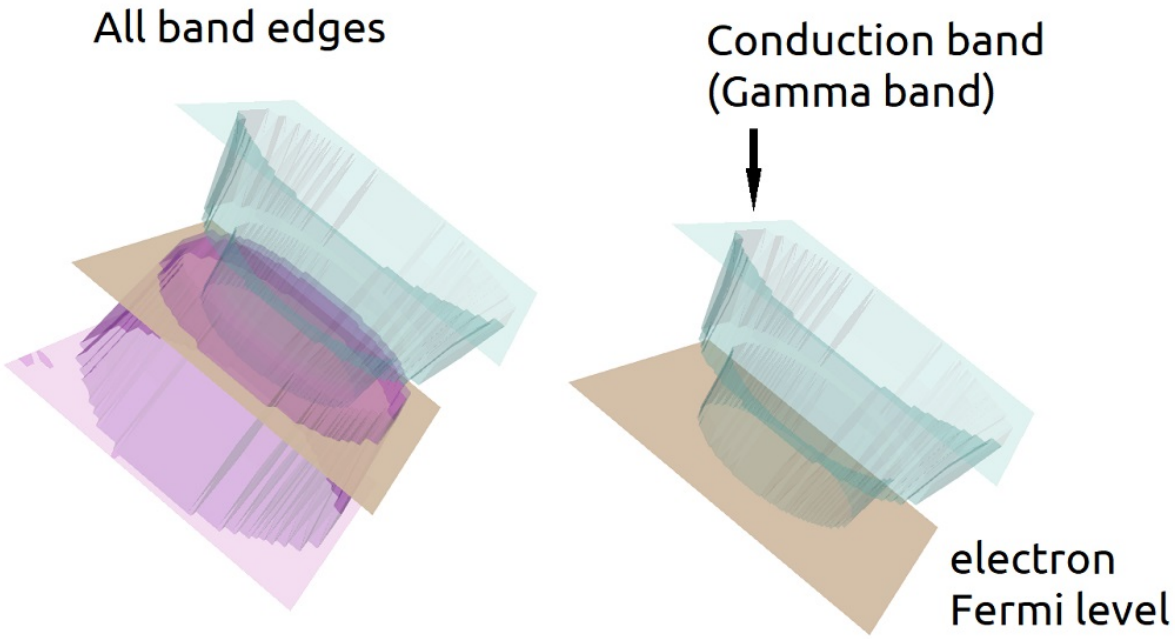


Figure 5.5.1.15: Overlay files are always plotted using a different palette and using semi-transparent surfaces. The Fermi levels are plotted as opaque surface.

Now let us discuss in more detail the second group of plots concerning the “shifted” wave functions (actually

probability densities). Looking at the tab `Information` under the `Pipeline Browser` we will observe that actually we have in total 40 columns about the eigenvalues and 40 columns about the shifted wave functions. Nevertheless, in the pipeline we observe that only the 5 shifted wave functions are plotted. In another words, the eigenvalues are not considered as dataset for this specific kind of file.

Changing Scale Factor of warped files of shifted probability density

The file related with the shifted probability densities requires an special care, because each component correspond to the combination of two different informations: the probability density (in unit of nm^-2 in the case of 2D simulations) and the shift of this function by the corresponding eigenvalue (in unit of eV), as shown in Figure 5.5.1.16. When overlayed with any file whose unit is energy, these components shall be plotted in the same scale as the overlay plot, independent of the scalar factor in the warp transformation of part corresponding to the probability density.

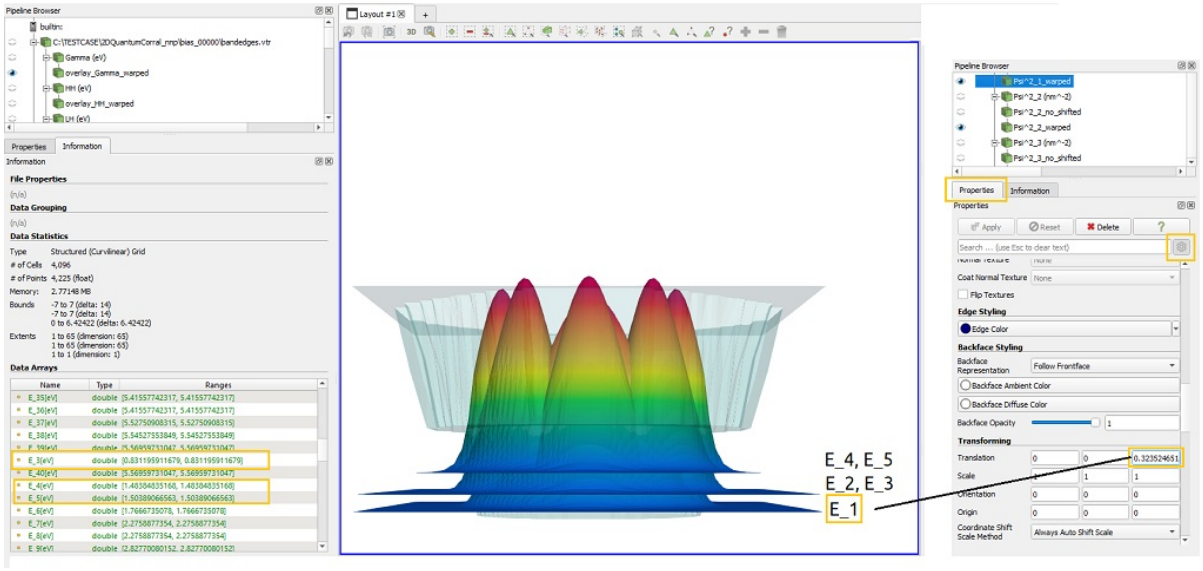


Figure 5.5.1.16: Probability density functions shifted corresponds to the sum of the probability function and its corresponding eigenvalue. For representing correctly when overlayed with the band edges, the warped version shall translate the basis of this plot by the corresponding eigenvalue.

Similar to the case of a single plot, all components of this datafile present the `Scale Factor` for the warped file interconnected. Then, changing the Scale Factor of one component of the datafile (*probabilities_shift*), will change the height of each function, but will not shift the eigenvalue energy (the base of the plot), as shown in the Figure 5.5.1.17

As demonstration, click on the component `Psi^2_1_warped`, and change the `Scale Factor`. As we mentioned, you will observe that the bases does not change.

Changing Scale Factor of warped files of the band edges overlayed with shifted probability density

Neverthless, changing the `Scale Factor` of the overlay file (the band edges in this case), whose by default is `1.0`, does not affect the plot of the shifted probability densities. This is expected because components of different datafiles are not interconnected. In this case, it is necessary to change the energy scale used in the plot of the shifted probabilities. This is done multiplying all the values of `translation` in the warped file of the shifted probabilites by the new factor of the overlay `Scale Factor`.

As example, let us change the `Scale Factor` of the overlay_bandedge_gamma_warped by 0.1. Then for the correct overlay with the shifted probabilities we will require to multiply the translation value by 0.1, for *Psi^2_1_warped*, *Psi^2_2_warped*, and so on. This is illustrated in Figure 5.5.1.18.

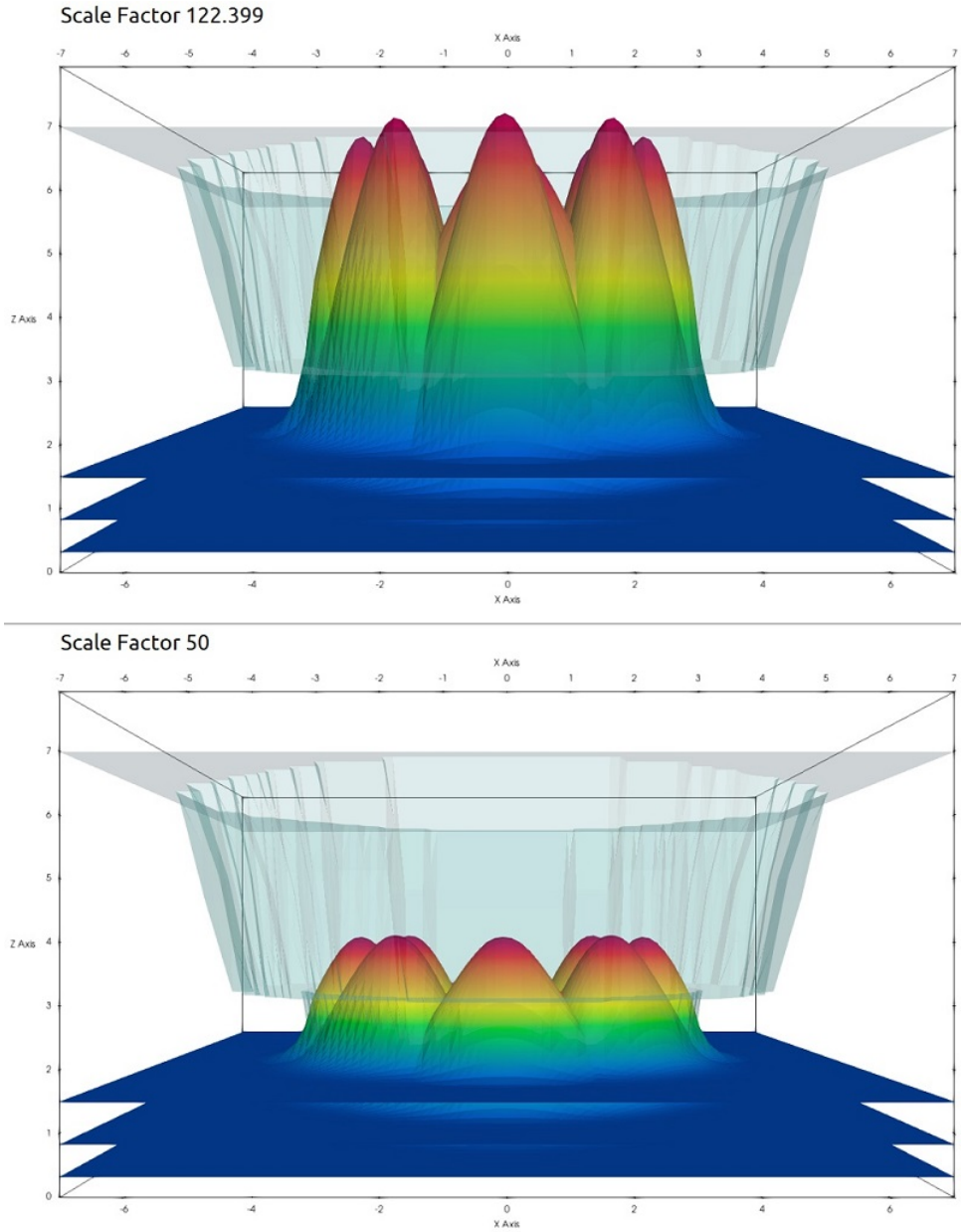


Figure 5.5.1.17: Probability density functions shifted by their corresponding eigenvalues. Changing the `Scale Function` of the warped plots of one component will shift the height of the plots, but not the base of them.

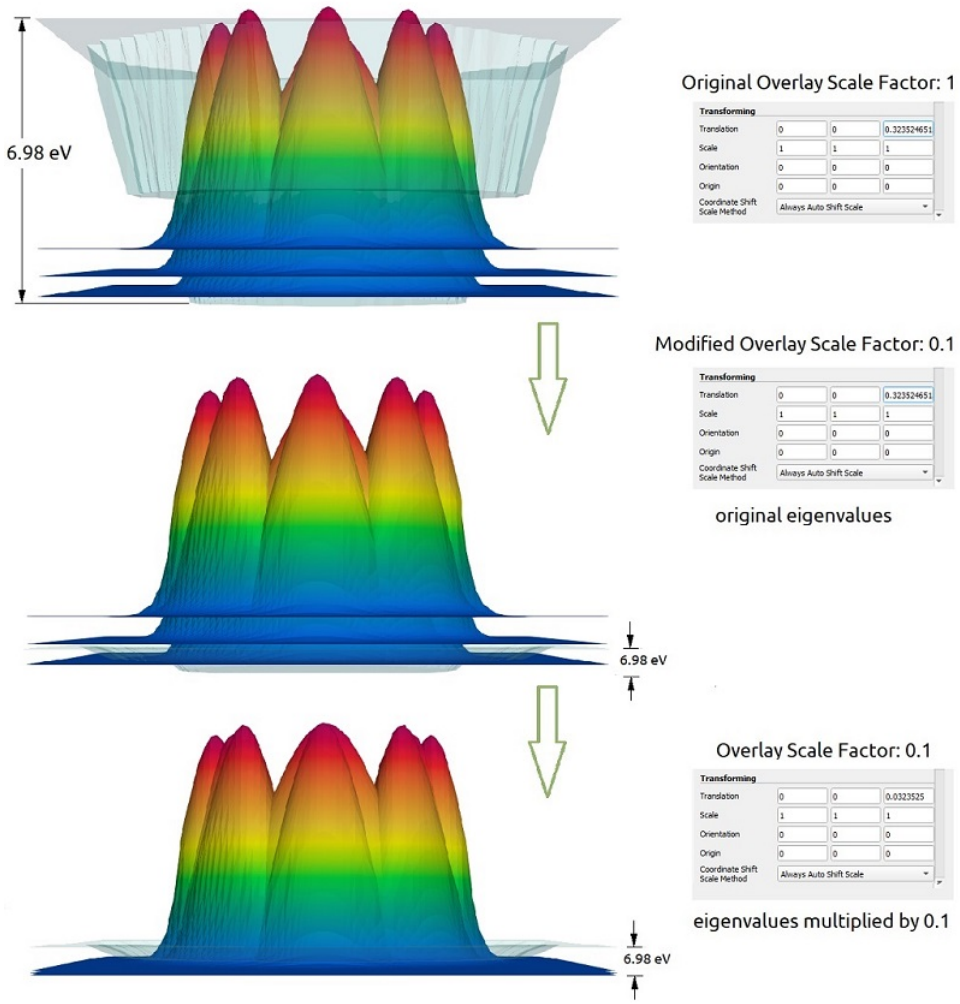


Figure 5.5.1.18: Probability density functions shifted by their corresponding eigenvalues. Changing the `Scale Function` of the warped plots of one component will shift the height of the plots of the other components of the same datafile, but not the base of them.

There are a couple of another situations where a similar procedure is necessary and not all of the possible combinations are implemented in our script. ParaView provide several other ressources to make the interconnection among the objects of the `Pipeline Browser`. Use the current script as an example how to implement this kind of associations, and feel free to explore another possibilities when creating your own script.

Hint: The warped plots uses a general setting of the camera that it is not universal, and may not be ideal for all plots. This can be easily fixed clicking on the `Reset Camera` button within ParaView or interacting with the plot in this platform.

Warning: As discussed above, files with mixed units will require special attention when overlayed with another files. Be aware to adjust the components properly.

When to use the overlayed plot method: This method is ideal for the case you want to superpose information of two files with different information. In the case of plotting *probabilities_shift* files overlayed with *bandedges* the script align the two set of data in the same reference frame.

5.5.2 Gnuplot export

Gnuplot is a free graphing utility, which allows scientists to visualize data interactively. If installed it can be used to export 1D, 2D & 3D nextnano GmbH results.

How to export 1D plot files with Gnuplot

- *Moving plot-files to another device*

Either the currently selected file can be exported (by usage of the context menu) or the contents of the `overlay list` can be combined (1D) and exported (by usage of the output menu button). An example of such a combined file can be seen in Figure 5.5.2.1.

Moving plot-files to another device

1D-plots are linked to the original .dat file(s). So if you want to move your plot to another device, you can either save your plot as .pdf/.svg/.png file directly in gnuplot (recommended), or if you want to move the original .plt-file you also have to move all necessary .dat files (the paths can be adjusted when opening the .plt file with a text editor).

Warning: If you move the 1D plot file without data files or without adjusting the paths, it will be broken. (Won't open when file is double-clicked.)

How to export 2D or 3D plot files with Gnuplot

In 2D you have some additional options for your gnuplot file compared to the 1D export, displayed in Figure 5.5.2.2. The plot can be displayed as a color map, analog to the implementation of nextnano GmbH, or as a surface plot, which is a pseudo three-dimensional plot (Figure 5.5.2.3). Title and labels are optional and if they aren't specified they will be taken directly from the file (if provided). If `remember settings` is checked, the next time the panel will be pre-filled with these settings. If you always use the same settings and don't want this dialog to be displayed each time, you can chose the `Create Gnuplot file - last used (*.plt)` option in the context menu.

Gnuplots are interactive, see Figure 5.5.2.4, which makes them suitable to create animations e.g. for presentations. Furthermore they can be saved as vector graphics. To understand the dependency between plot files and their raw data files, please refer to the next section.

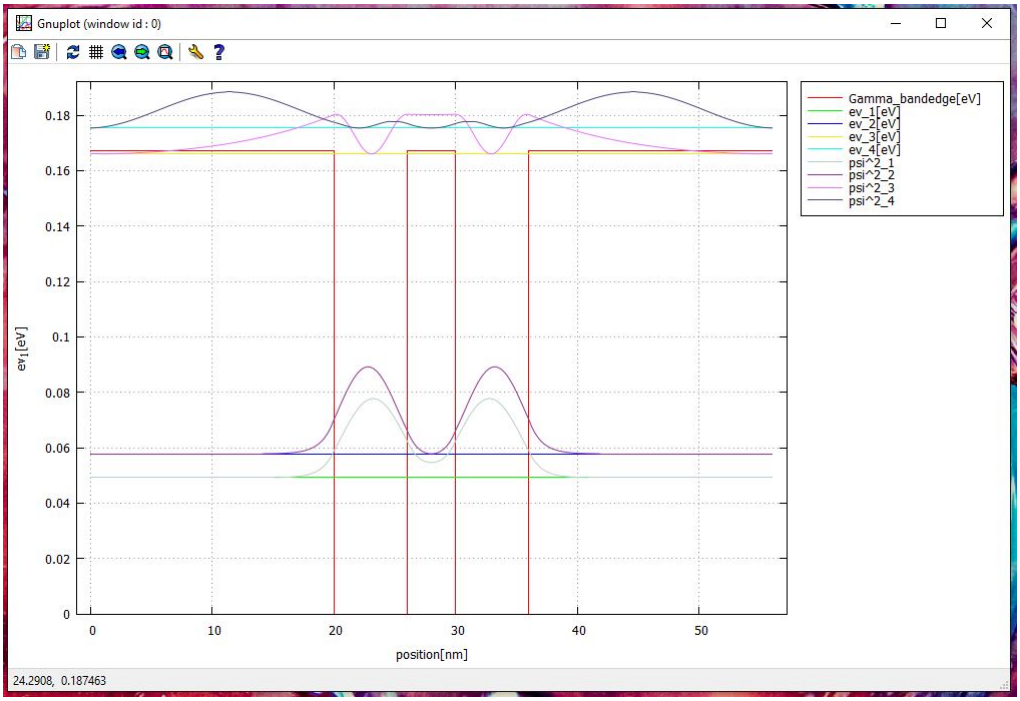


Figure 5.5.2.1: One-dimensional Gnuplot export of band edge plus probability densities.

How to optimze the looks of a Gnuplot graph

- *Optimize your Gnuplot Graph manually*
- *Generate high quality graphs*

The plot style for all Gnuplot exports can be customized in *Options: Gnuplot settings*. This is convenient to achieve uniformous style of the exported graphs e.g. for a presentation. A style-sheet can be used which can be shared within the whole work-group.

Optimize your Gnuplot Graph manually

Alternatively or additionally to the style-sheet you can customize each Gnuplot file on its own. Just open the file with a text editor of your choice and change, add or remove commands.

Collection of some useful gnuplot commands.

Semi-log plot

```
set logscale x
set logscale y
```

Change the line thickness (`lw 4`)

```
plot 'D:\bandedges.dat' linetype rgb "#FF0000" pt 5 ...
plot 'D:\bandedges.dat' linetype rgb "#FF0000" pt 5 lw 4 ...
```

Change font size of the x axis (`20`)

```
set xlabel "position (nm)" font "sans - serif"
set xlabel "position (nm)" font "sans - serif,20"
```

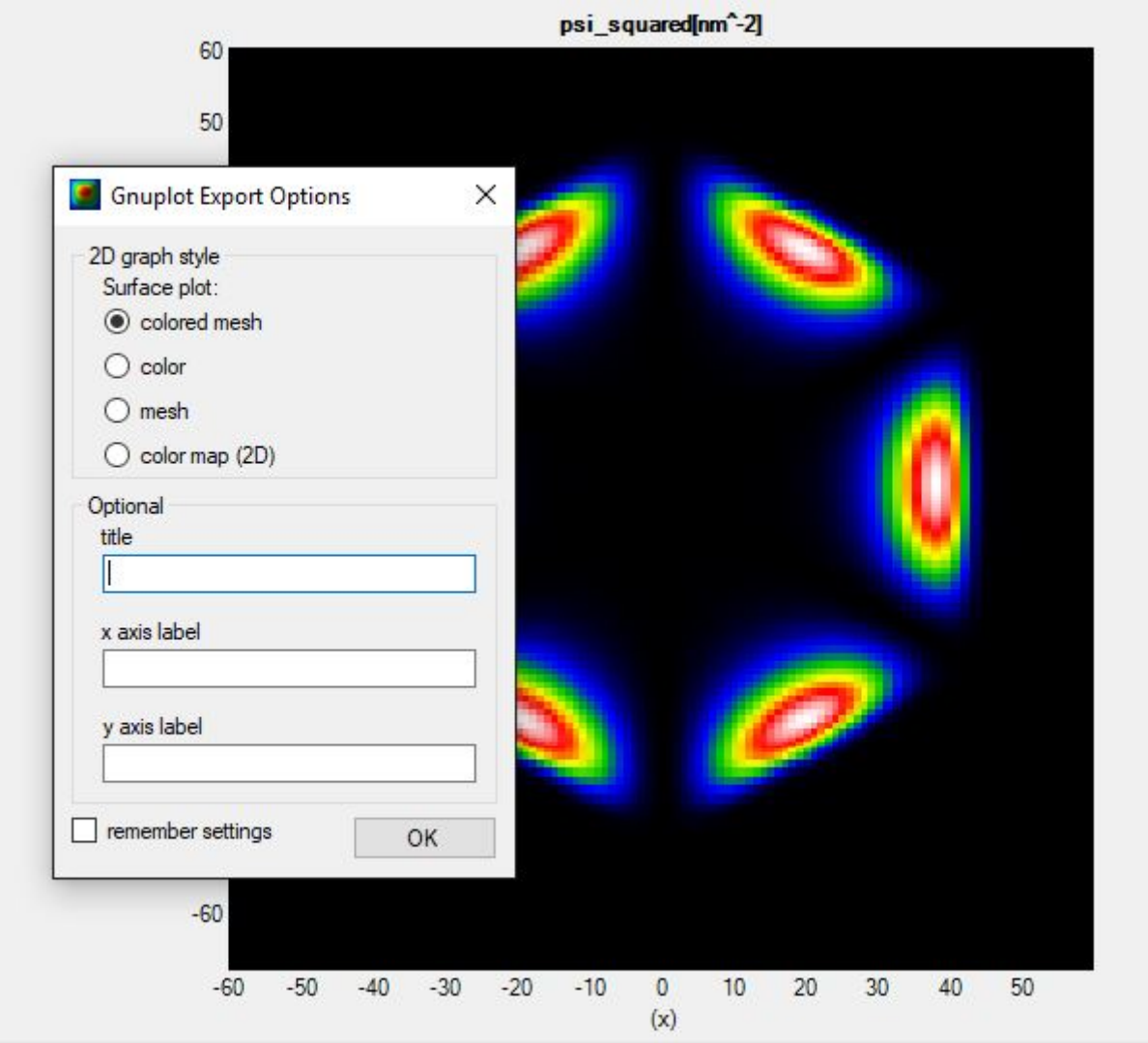


Figure 5.5.2.2: Additional options for more-dimensional Gnuplot export.

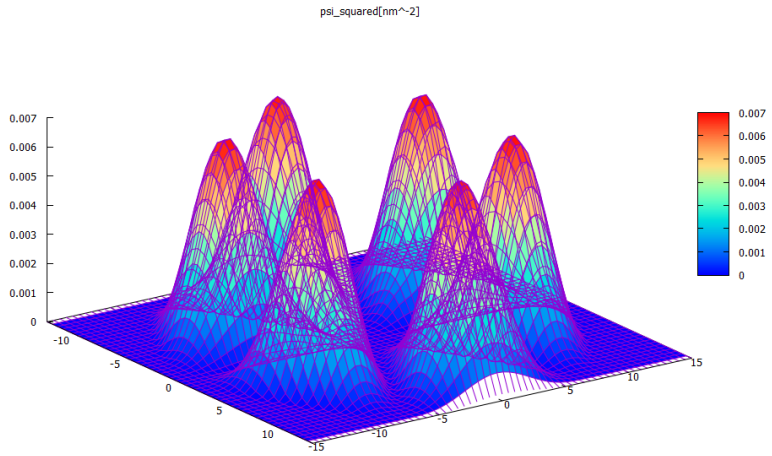


Figure 5.5.2.3: Gnuplot surface representation of Figure 5.5.2.2. (Probability density of the 10th wave function in a hexagonal structure.)

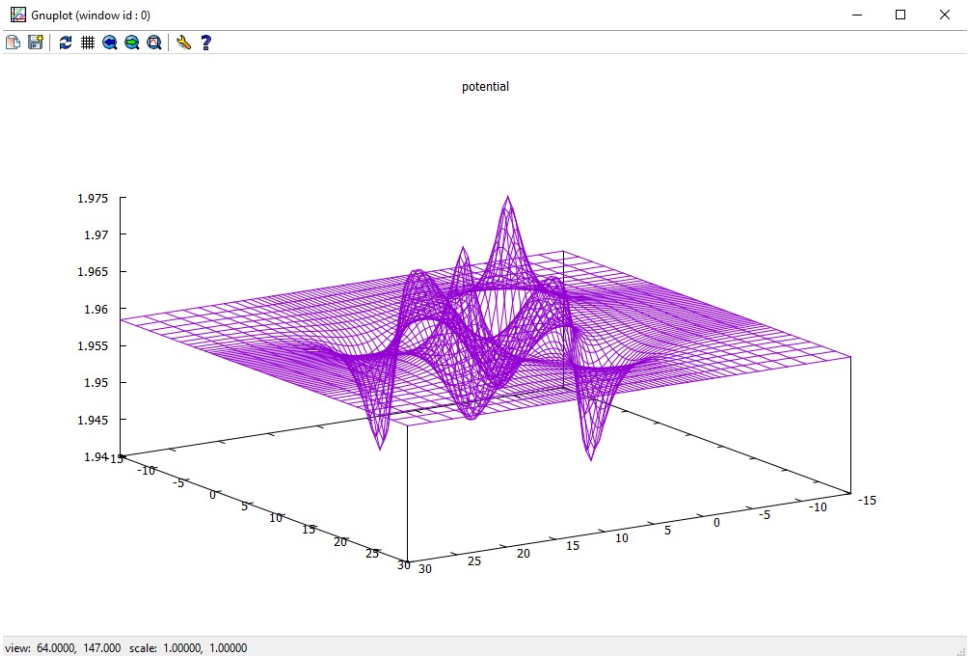


Figure 5.5.2.4: Interactive surface plot.

Use subscript and superscript (enhanced vs. noenhanced)

```
plot 'D:\density_hole.dat' using 1:2 title "p (10^{18} cm^{-3})
↪"enhanced ...
plot 'D:\density_hole.dat' using 1:2 title "p (10^18 cm^-3)" noenhanced␣
↪...
```

Change range [x_{min} , x_{max}] and [y_{min} , y_{max}] of the graph

```
set xrange [-0.3:100.3]
set xrange [0:100]
set yrange [-1.5:2.5]
set yrange [-1.0:0.5]
```

Set/Remove grid

```
set grid
unset grid
```

Change thickness of the border (lw)

```
set border lw 2
set border lw 3
```

Set/Remove legend

```
set key on
set key off
```

Set/Remove box around legend (box)

```
set key on ... box
set key on ... nobox
```

Increase font size in legend

```
set key on ... font "sans - serif,14"
set key on ... font "sans - serif,18"
```

Specify the location of legend

```
set key left top inside ...
set key right bottom outside ...
```

Remove line from legend (`notitle`)

```
plot 'D:\bandedges.dat' using 1:2 title "E_c " ...
plot 'D:\bandedges.dat' using 1:2 notitle "E_c " ...
```

Add Greek letter to line in legend (`"{/Symbol G}" enhanced`)

```
using 1:2 title "{/Symbol G} [eV]" enhanced
```

produces Γ.

e, l, m, q produce ϵ, λ, μ, and θ, respectively.

Add a label to the point (x, y) in the plot

```
set label "label" at 0.5,1.5
```

Add an arrow

```
set arrow from 1.5,0.3 to 4,2
```

Graph Title

```
set title "title" font "sans - serif,18"
```

Generate high quality graphs

1. In gnuplot window: Click on `Export plot to file`
2. Save as SVG files (.svg)
3. Open the saved .svg file with Inkscape
4. File ⇒ Export .PNG Image… ⇒ Select Drawing ⇒ Export

5.5.3 Export via python scripts

How to call custom python scripts directly from nextnanomat

Custom written python scripts can be called directly from the Output tab. The parent folder of the python script, Path to the currently displayed output file as well as the path to the simulation folder are transferred as system arguments, see Figure 5.5.3.1. (If you need other specific information to be exported, just mail your request to the nextnano GmbH support team or use the widget on this website.) By using those system arguments in your python script, individual post-processing of specific output files is reduced to a one-click effort.

Related: *Exporting 2D Outputs with Python Scripts*

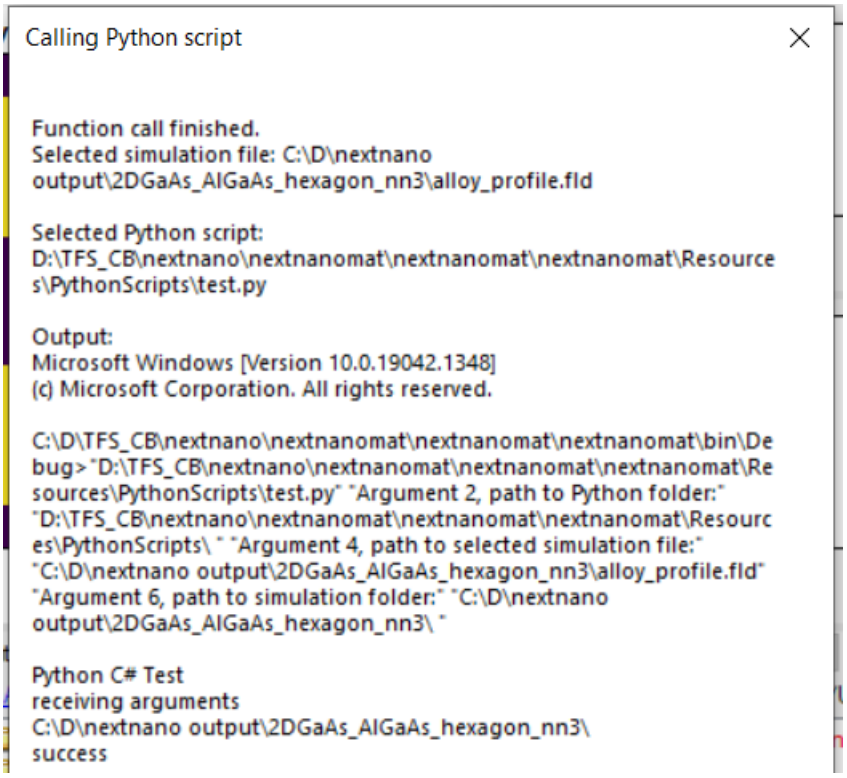


Figure 5.5.3.1: Example of transferred system arguments when calling a python script.

5.5.4 Other

In *Options: Data export* custom paths to installed applications (e.g. ParaView, notepad++) can be set, to allow convenient export of output files for the purpose of visualization or post-processing.

5.6 Release Notes

5.6.1 4.3.2.15 (2024-03-28)

New features

- Added option to kill whole process tree instead of process when stopping a simulation
- Added support of new nextnanoLicenseActivator executable (former ClientSideActivtor and ClientSideServerActivation are no longer needed)
- Editor tab: support of "!Data" syntax for *nextnano++*
- Editor tab: Feature 'Create new input file by template' is implemented for *nextnano++* input files
- Output tab: option to display Log(abs(y))

Updates & Improvements

- Improved layout for license activation message feed
- Input file conversion for *nextnano*^{3} -> *nextnano++* is now a standalone executable, can be called by command line
- Output tab: small viusalization changes, e.g. fonts
- Change of default performance settings
- Input tab, more accurate naming for context menu functions

- Improved folder structure for installation and portable packages
- Improve selection of matching database for different nextnano packages

Bugfixes

- Compatibility fix for new license check
- Cluster computing feature for *nextnano.NEGF* simulations
- HTCondor, change default OS target for cluster computing
- Update existing jobs in Batch List after license changes

Remaining known Bugs

- In installed versions running a simulation reverts Windows App Scaling back to 100%

5.6.2 4.3.2.8 (2023-08-07)

New features

- Added specific licenses for nextnano^3 and evaluation
- Added support for free licenses
- Distinguish license types, display type during simulations and in settings
- Added nextnano documentation (pdf) to installed package
- Editor tab: added html keywordtree for nextnano++ to help with input file syntax
- Editor tab: convert xml input files to negf input files (for nextnano.MSB and nextnano.NEGF)
- Editor tab: added html keyword trees for all products to help with input file syntax
- Settings: verify licenses directly within settings
- Settings: added preview possibility for Editor and Output settings
- Settings: added support for two nextnano.NEGF tool versions (C# & C++)
- Output tab: added “local” and “global” full-size differentiation

Updates & Improvements

- Editor tab: updating links to online documentation for context help menu
- Editor tab: converted input file (nextnano^3 to nextnano++) is opened in new tab
- Simulation tab: update queued simulations after change in settings
- Settings: functionality to disable syntax highlighting
- Settings: restructured Options Form
- Settings: added customizable tolerance for comparison of constant values (expert features)
- Output tab: autosize for pointer labels
- Output tab: changed numeric format for pointer labels
- Output tab: button position of full-size button
- Output tab: performance improvement in 1D output visualization during curve(s) selection
- Output tab: separate, re-order and re-name check/select buttons
- Output tab: improved spacing between data visualization and diagram axis
- Output tab: advanced python and ParaView export routines

Bugfixes

- Event handling, custom export function
- License activation prompt, appearing at every program start
- Editor tab: syntax highlighting for conditional statements
- Editor tab: input file conversion: added error handling for files located in program files
- Editor tab: Warning message for deprecated syntax "#IF to !WHEN"
- Simulation tab: running a cloud nextnano.NEGF simulation does no longer change the input file extension
- Simulation tab: command line arguments for starting nextnano.NEGF simulations
- Simulation tab: fix performance issues for simulating a batch of jobs
- Settings: display of Pool names for HTCondor
- Settings: opening cloud options form
- Output tab: color scale initialization
- Output tab: adjust reference of message box to new menu location of export feature
- Output tab: visualization of txt licenses
- Support free licenses to be located in program files

Remaining known Bugs

- In installed versions running a simulation reverts Windows App Scaling back to 100%

5.6.3 4.3.1.8 (2022-08-05)

New features

- standard selection of color maps for 2D and 3D
- feature to customize color maps
- couple of features related to color maps (e.g. invert scale, fix middle color etc. -> refer to online documentation)
- syntax support for new input file format *.negf
- customizable syntax highlighting
- support for new NEGF licenses
- feature to run batch list automatically when new files are added
- added accessibility evaluation and statement for the nextnano software
- allow integration for tiberCAD input files and output
- output, show value difference for two selected curves

Updates & Improvements

- 1D Export to Gnuplot of overlay items includes currently displayed file
- add or remove items to overlay list by keyboard only
- improved syntax highlighting
- options to specify whether and how output should be overwritten
- set online help references to the new nextnano GmbH documentation

Bugfixes

- prevent crash at application start due to color-map initialization
- prevent application crash due to IPv6 IP addresses
- modified file star
- prevent duplicate opening of the same input file
- increase functionality and stability of Template(Beta) tab

Remaining known Bugs

- input file shows in simulation tab during CPU heavy simulations

Input files

- quite a few new ones (check them out in the sample files folder)

5.6.4 EARLIER

- New colormaps
- Arrow showing difference value between two selected graphs
- HTCondor support for mixed Linux/Windows pool

CHAPTER

SIX

NEXTNANO++

The *nextnano++* tool is a Schrödinger-Poisson-current solver and simulates quantum wells, quantum wires, quantum dots, … The *nextnano++* tool (written in C++) is the successor of the *nextnano*[3] code (written in Fortran).

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:

6.1 Overview

6.1.1 Running

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

6.1.2 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.

6.1.3 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.

6.1.4 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.

6.1.5 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.

If you have further questions, see the *Frequently Asked Questions (FAQ)* or contact *nextnano Help Center*.

6.2 Models

This set of tutorials focus on introducing models implemented in *nextnano++* tool.

6.2.1 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.

6.2.2 Hamiltonian: 8-band model for zincblende

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

$$\hat{H} = \frac{\hat{\mathbf{p}}^2}{2m} + V_0(\mathbf{r}) + \frac{\hbar}{4m^2c^2}\left[\hat{\sigma} \times \nabla V_0(\mathbf{r})\right] \circ \hat{\mathbf{p}} \tag{6.2.2.1}$$

The description below contains also definitions and relations that can be found in *[BirnerPhD2011]* and *[Bahder-PRB1990]*.

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:

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

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

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

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

Diagonal elements for the conduction band are defined as

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

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

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

Blocks introducing interaction between conduction and valence bands are given by

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

and

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

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

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

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

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

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

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

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

$$G = \frac{\hbar^2}{2m_0^2} \sum_{nj}^{\mathcal{B}} \frac{|\langle x_1| \hat{p}_1 |n\Gamma_3 j\rangle|^2}{E_\mathrm{v} - E_{n,\Gamma_3}}$$

$$F' = \frac{\hbar^2}{2m_0^2} \sum_{nj}^{\mathcal{B}} \frac{|\langle x_1| \hat{p}_1 |n\Gamma_1 j\rangle|^2}{E_\mathrm{v} - E_{n,\Gamma_1}}$$

$$H_1 = \frac{\hbar^2}{2m_0^2} \sum_{nj}^{\mathcal{B}} \frac{|\langle x_1| \hat{p}_1 |n\Gamma_5 j\rangle|^2}{E_\mathrm{v} - E_{n,\Gamma_5}}$$

$$H_2 = \frac{\hbar^2}{2m_0^2} \sum_{nj}^{\mathcal{B}} \frac{|\langle x_1| \hat{p}_1 |n\Gamma_4 j\rangle|^2}{E_\mathrm{v} - E_{n,\Gamma_4}}$$

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

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

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

Offsets

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

Where the following mapping to our database is applied.

Table 6.2.2.1: Mapping of offsets to the database

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

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

Deformation potentials

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

Where the following mapping to our database is applied.

Table 6.2.2.2: Mapping of deformation potentials to the database

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

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 6.2.2.3: 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 6.2.2.4: 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.

Last update: nn/nn/nnnn

6.2.3 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

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:

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

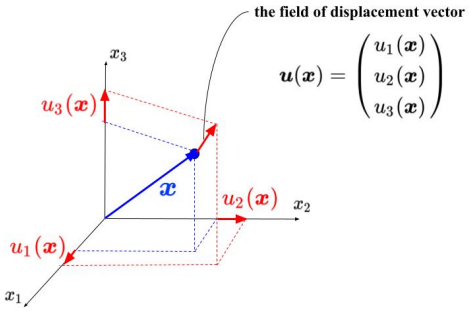


Figure 6.2.3.1: The field of displacement vector $\mathbf{u}$ at $\mathbf{x}$. This is the vector along which the point that was at the position $\mathbf{x}$ moved through the displacement.

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 6.2.3.2.

The off-diagonal elements $\varepsilon_{ij(i\neq j)}$ arise due to shear deformations of the crystal. Figure 6.2.3.3 shows the deformation of an infinitesimal rectangle in x_1x_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.

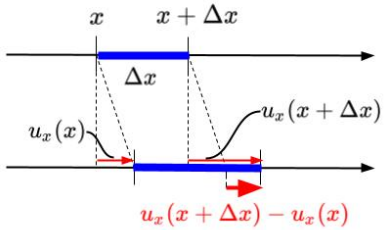


Figure 6.2.3.2: Deformation of a dilatable string in an unstrained (top) and strained state (bottom). We can see the diagonal element $\varepsilon_{ii} = \frac{\partial u_i}{\partial x_i}$ represents the length changes per unit length in x_i-direction.

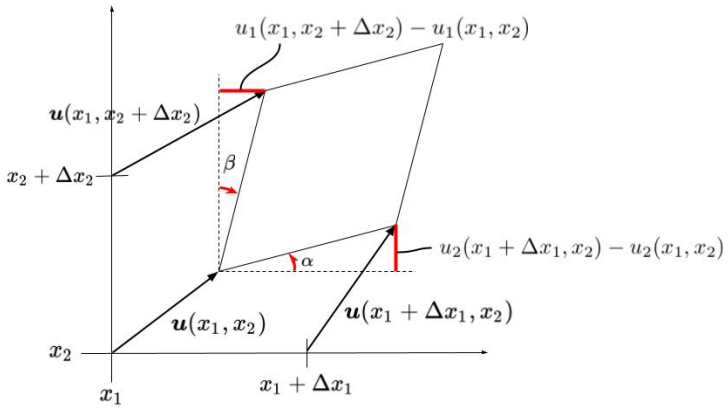


Figure 6.2.3.3: Deformation of an infinitesimal rectangle in a strained state.

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:

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

and

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

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 ($[\mathrm{Pa}] = [\mathrm{N/m^2}]$).

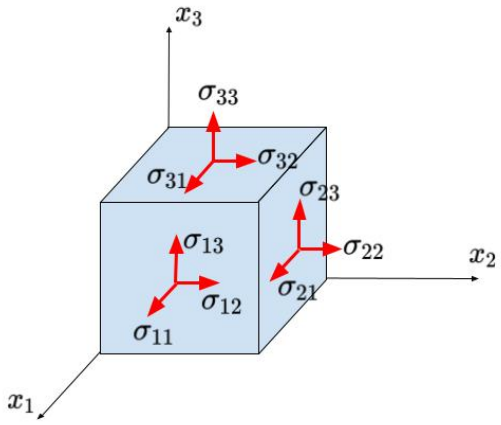


Figure 6.2.3.4: The components of stress tensor σ.

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

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: nn/nn/nnnn

6.2.4 Piezoelectricity in wurtzite

The *nextnano++* and *nextnano*^{3} tools 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*

Specify crystal orientation

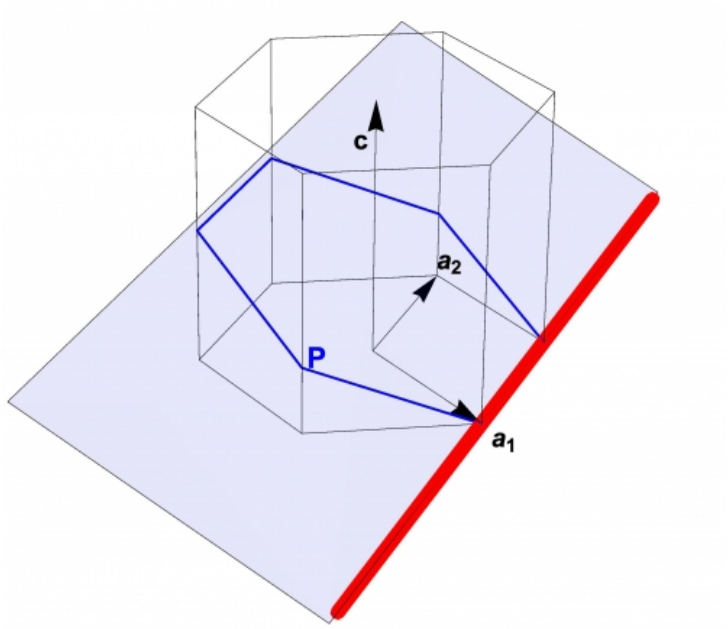


Figure 6.2.4.1: Rotation of a wurtzite structure. The blue plane is parallel to the interface.

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 6.2.4.1. 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 6.2.4.2 shows the cross-section of a wurtzite lattice that is perpendicular to the rotation axis in Figure 6.2.4.1.

- 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

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

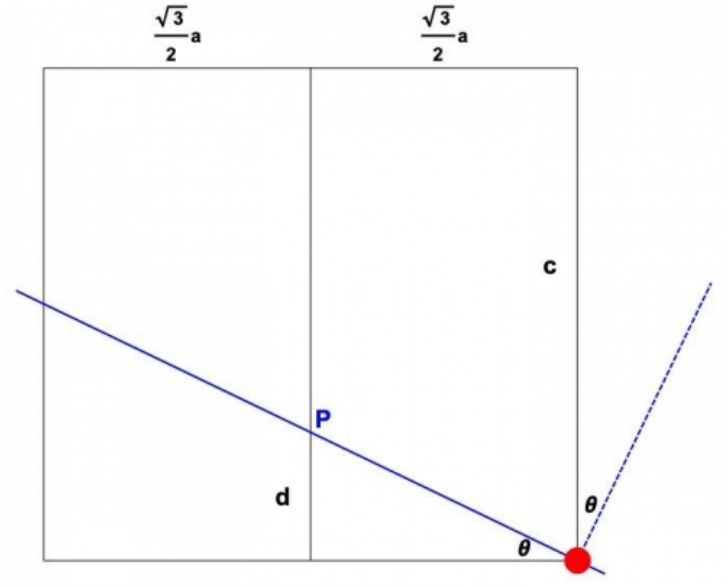


Figure 6.2.4.2: Cross-section of the wurtzite lattice. The dashed blue line indicates the **x'**-direction, which is normal to the interface (solid blue line).

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:

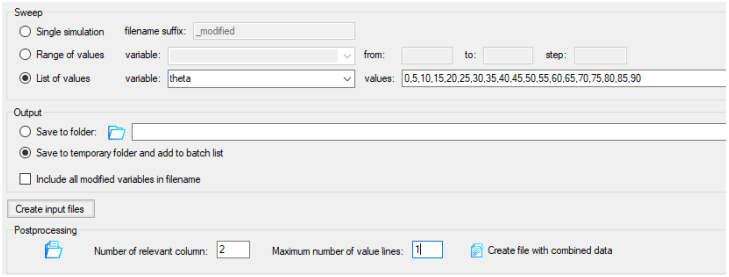
```
$gamma = $c_InGaN / $a_InGaN # c/a ratio
# ideal c/a ratio in wurtzite is SQRT(8/3)=1.63299
$h = floor(100*sin(theta))
$l = floor(100*2*gamma*cos(theta)/sqrt(3))
x_hkl = [$h, 0, $l]  # x axis perpendicular to (hkl) plane = (hkil) plane
```

Since nextnano[3] does not support variables in the Miller-Bravais indices, we explicitly give the indices in the input file using statements like `!IF %ORIENTATION40 hkil-x-direction = 64 0 -64 143`.

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.



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*

Strain

Figure 6.2.4.3 and Figure 6.2.4.4 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].

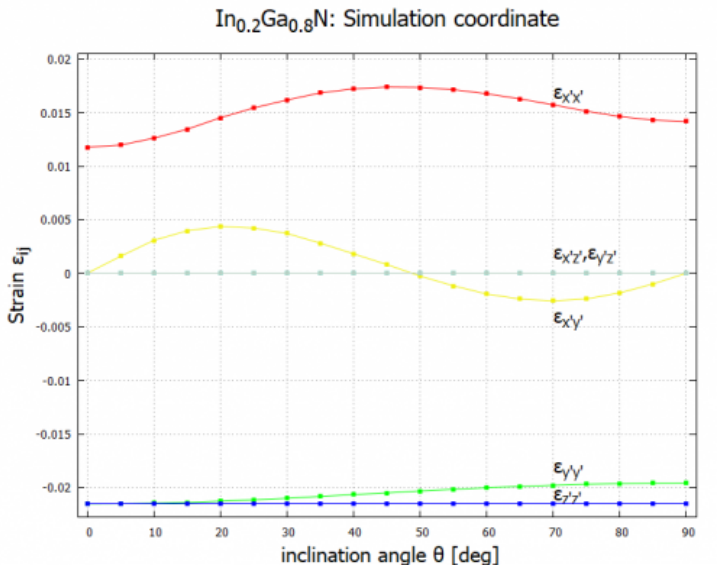


Figure 6.2.4.3: Elastic strain tensor components as a function of c-axis inclination angle θ in **simulation** coordinate system.

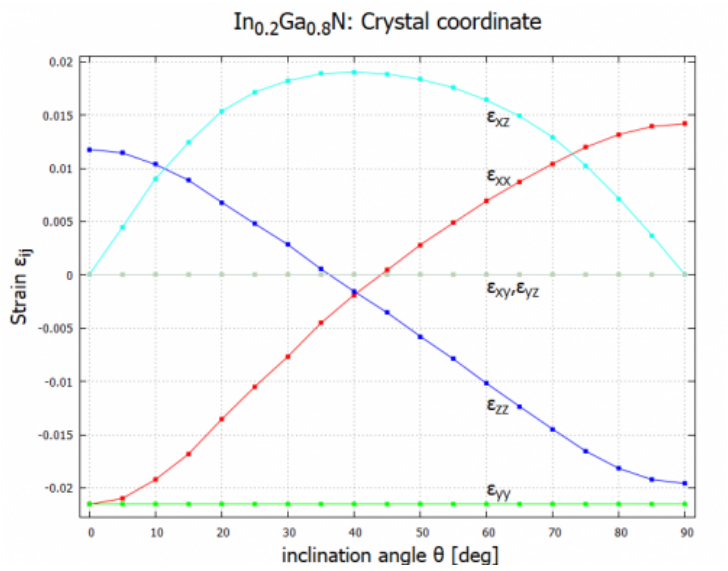


Figure 6.2.4.4: Elastic strain tensor components as a function of c-axis inclination angle θ in **crystal** coordinate system.

Piezoelectric effect (first-order)

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

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

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 in both the *nextnano++* and *nextnano*^{3} tools with corresponding material parameters in the respective database. The following flags export the strain tensor components and piezoelectric polarization vector in **crystal** and **simulation** coordinate systems (cf. *nextnano++* and nextnano3 documentation). 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{
```

(continues on next page)

(continued from previous page)

```
        crystal_system    = yes
        simulation_system = yes
    }

    output_polarization_vector{
        crystal_system    = yes
        simulation_system = yes
    }

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

```
$output-strain
    strain-simulation-system = yes
    strain-crystal-system    = yes
    polarization-vector      = yes
$end_output-strain
```

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

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

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 6.2.4.5 compares the results of the nextnano software with the results of [Romanov2006] and [Schulz2015], respectively. The analytical results in Figure 6.2.4.5 are the plot of the equation above, with an interchange of **x'**- and **z'**-axes.

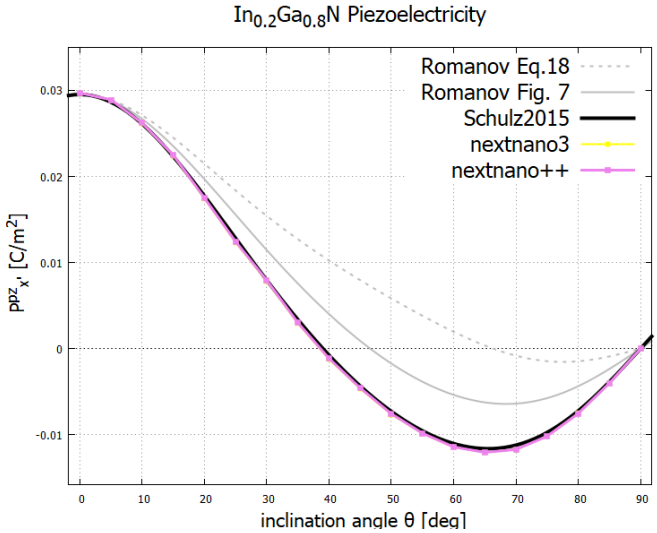


Figure 6.2.4.5: Piezoelectric polarization as a function of inclination angle. The gray dotted curve contains a typo $e_{33} \leftrightarrow e_{31}$ and misses the factor 2. When the first typo is fixed, the gray solid curve is obtained and looks to be consistent with Figure 7(a) in [Romanov2006]. With the factor 2 the result becomes the black curve. Both *nextnano*[3] and *nextnano++* reproduce the black curve and are thus consistent to [Schulz2015].

From the results in Figure 6.2.4.5 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.

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`.

- 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*

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 6.2.4.5 shown above, this mistake is not relevant for 0 and 90 degrees.

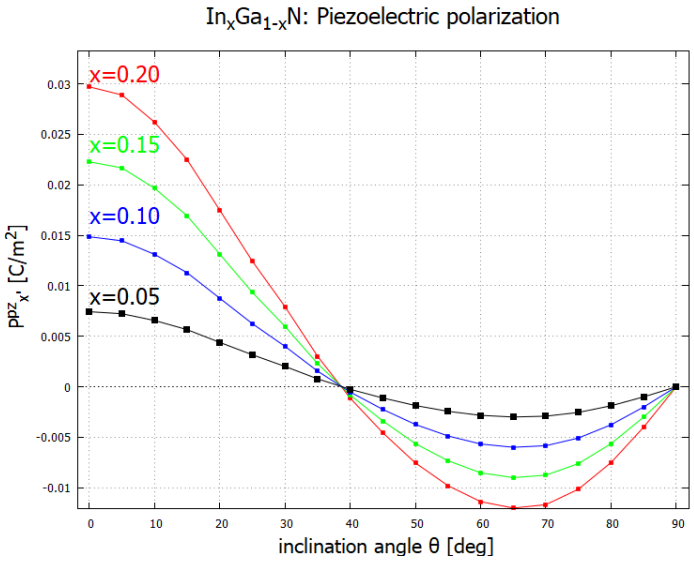


Figure 6.2.4.6: Alloy content dependence of the piezoelectric polarization for $In_xGa_{1-x}N$/GaN structure. $In_xGa_{1-x}N$ is under biaxial compressive strain with respect to GaN.

AlGaN

- Input file: *Romanov_AlGaN_theta_nnp.in*

Similarly, piezoelectric polarization of $Al_xGa_{1-x}N$/GaN structure is calculated and shown in Figure 6.2.4.7. 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 6.2.4.5 shown above, this mistake is not relevant for 0 and 90 degrees.

The sign of the piezoelectric polarization in Figure 6.2.4.7 is opposite to the case of InGaN/GaN composition (Figure 6.2.4.6). 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, $In_xGa_{1-x}N$ layer is subject to compressive strain, whereas $Al_xGa_{1-x}N$ is under tensile strain [Romanov2006].

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]

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

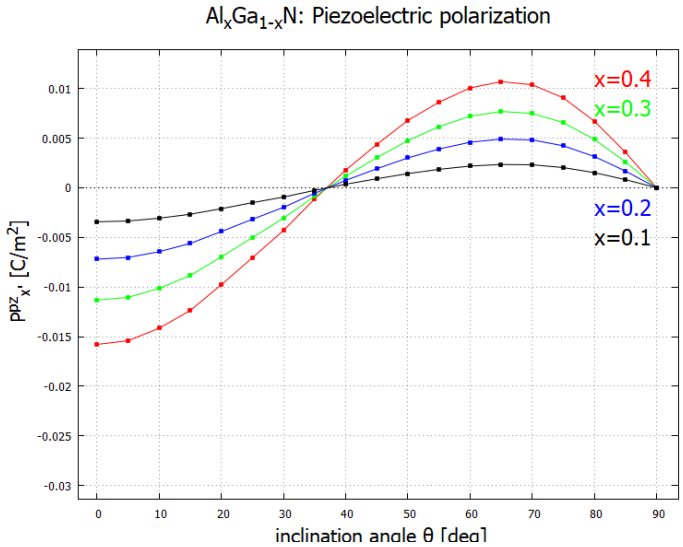


Figure 6.2.4.7: Alloy content dependence of the piezoelectric polarization for $Al_xGa_{1-x}N$/GaN structure. $Al_xGa_{1-x}N$ is under biaxial tensile strain with respect to GaN.

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], which is also implemented in *nextnano++* and *nextnano*^{3}.

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

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

and in nextnano3,

```
! nextnano3
$numeric-control
    ...
    piezo-second-order = 2nd-order    ! [no/2nd-order]
$end_numeric-control
```

Figure 6.2.4.8 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 $In_{0.2}Ga_{0.8}N$/GaN. The pink curve is different from the one in Figure 6.2.4.5 because we employed the material parameters used in [Patra2017]. The *nextnano*^{3} tool also produces a consistent result (input file: `Romanov_InGaN_theta_nn3_2nd.in`). Within *nextnano*^{3} one can also use different formulae for the second-order effect, cf. nextnano3 documentation.

Last update: nn/nn/nnnn

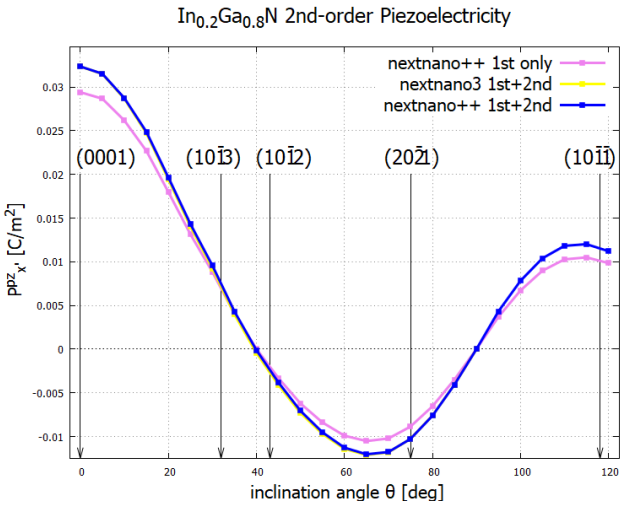


Figure 6.2.4.8: Second-order piezoelectricity. The second-order term enhances the piezoelectric polarization. The *nextnano*[3] result (yellow) is consistent to the *nextnano++* result (blue). Interface planes are indicated at corresponding angles.

6.2.5 General scheme of the optical device analysis

Here we summarize the models and equations that is used for the optical device analysis in *nextnano++*.

Table of contents

Related tutorials

- Semi-classical model
  - *GaAs solar cell*
  - *InGaAs Multi-quantum well laser diode*
  - *UV LED: Quantitative evaluation of the effectiveness of EBL*
  - *UV LED: Quantitative evaluation of the effectiveness of superlattice structure in p-region*
- Quantum model
  - *Optical absorption for interband and intersubband transitions*
  - *Optics: Optical gain of InGaAs quantum wells with different strain*
  - *Optics: Optical gain and spontaneous emission rate of strained GaN quantum well*
  - *Intersubband absorption of a GaAs cylindrical quantum wire*
  - *Interband absorption of a GaAs cylindrical quantum wire*
  - *Absorption of a GaAs spherical quantum dot*
  - 3D Optics: Interband absorption spectrum of a GaAs spherical quantum dot (*Coming soon.*)

Determination of carrier densities and current densities

Quantum mechanical calculation of charge carrier densities

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

$$p^i_{l,j}(\mathbf{x}) = \sum_\mu \left| (F_\mu)^i_{l,j}(\mathbf{x}) \right|^2 . \tag{6.2.5.1}$$

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

$$n(\mathbf{x}) = \sum_{i\in\mathrm{CB}} g^i_{\mathrm{c}} \sum_j p^i_{\mathrm{c},j}(\mathbf{x})\; f\Big([E^i_{\mathrm{c},j} - E_{\mathrm{F},n}(\mathbf{x})]/kT \Big) \tag{6.2.5.2}$$

$$p(\mathbf{x}) = \sum_{i\in\mathrm{VB}} g^i_{\mathrm{v}} \sum_j p^i_{\mathrm{v},j}(\mathbf{x})\; f\Big([-E^i_{\mathrm{v},j} + E_{\mathrm{F},n}(\mathbf{x})]/kT \Big) \tag{6.2.5.3}$$

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 $\mathbf{k}_\parallel$ in the lateral 2D direction and the quantized wave function in the growth direction, which has the $\mathbf{k}_\parallel$-dependency. Then the charge carrier densitiy is obtained by the following integral over $\mathbf{k}_\parallel$:

$$n(x) = \sum_{i\in\mathrm{CB}} g^i_{\mathrm{c}} \sum_j \frac{1}{(2\pi)^2} \int_{\Omega_{BZ}} d^2\mathbf{k}_\parallel\; p^i_{\mathrm{c},j}(x,\mathbf{k}_\parallel)\; f\Big([E^i_{\mathrm{c},j}(\mathbf{k}_\parallel) - E_{\mathrm{F},n}(x)]/kT \Big) \tag{6.2.5.4}$$

$$p(x) = \sum_{i\in\mathrm{VB}} g^i_{\mathrm{v}} \sum_j \frac{1}{(2\pi)^2} \int_{\Omega_{BZ}} d^2\mathbf{k}_{\parallel}\; p^i_{\mathrm{v},j}(x,\mathbf{k}_{\parallel})\; f\Big([-E^i_{\mathrm{v},j}(\mathbf{k}_{\parallel}) + E_{\mathrm{F},n}(x)]/kT\Big) \tag{6.2.5.5}$$

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

$$n(\mathbf{x}) = \sum_{i\in\mathrm{CB}} g^i_{\mathrm{c}} \sum_j \frac{1}{2\pi} \int_{\Omega_{BZ}} d\mathrm{k}\; p^i_{\mathrm{c},j}(\mathbf{x},\mathrm{k})\; f\Big([E^i_{\mathrm{c},j}(k) - E_{\mathrm{F},n}(\mathbf{x})]/kT\Big) \tag{6.2.5.6}$$

$$p(\mathbf{x}) = \sum_{i\in\mathrm{VB}} g^i_{\mathrm{v}} \sum_j \frac{1}{2\pi} \int_{\Omega_{BZ}} d\mathrm{k}\; p^i_{\mathrm{v},j}(\mathbf{x},\mathrm{k})\; f\Big([-E^i_{\mathrm{v},j}(k) + E_{\mathrm{F},p}(\mathbf{x})]/kT\Big) \tag{6.2.5.7}$$

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 (6.2.5.4) to (6.2.5.7) 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:

$$n(\mathbf{x}) = \sum_{i\in\mathrm{CB}} g^i_{\mathrm{c}} \left(\frac{m_{\mathrm{dos,e}} kT}{2\pi\hbar^2}\right)^{(3-d)/2} \sum_j p^i_{\mathrm{c},j}(\mathbf{x})\; \mathcal{F}_{(1-d)/2}\Big([E^i_{\mathrm{c},j} - E_{\mathrm{F},n}(\mathbf{x})]/kT\Big) \tag{6.2.5.8}$$

$$p(\mathbf{x}) = \sum_{i\in\mathrm{VB}} g^i_{\mathrm{v}} \left(\frac{m_{\mathrm{dos,h}} kT}{2\pi\hbar^2}\right)^{(3-d)/2} \sum_j p^i_{\mathrm{v},j}(\mathbf{x})\; \mathcal{F}_{(1-d)/2}\Big([-E^i_{\mathrm{v},j} + E_{\mathrm{F},p}(\mathbf{x})]/kT\Big) \tag{6.2.5.9}$$

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 $m^i_{\mathrm{dos},\lambda}$ is so-called density-of-states mass defined as

$$m^i_{\mathrm{dos},\lambda} = \left(\det \bar{m}^{*i}_{\lambda}\right) \quad \lambda = \mathrm{e,h} \tag{6.2.5.10}$$

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 $n(\mathbf{x},\phi,E_{\mathrm{F},n})$ and $p(\mathbf{x},\phi,E_{\mathrm{F},p})$ makes it easy to understand what the self-consistent calculation is actually doing.

Classical calculation of charge carrier densities

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 $E^i_{\mathrm{c}}(\mathbf{x})$ and $E^i_{\mathrm{v}}(\mathbf{x})$, 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

$$n(\mathbf{x}) = \sum_{i\in\mathrm{CB}} N^i_{\mathrm{c}}(T)\; \mathcal{F}_{1/2}\Big([-E^i_{\mathrm{c}}(\mathbf{x}) + e\phi(\mathbf{x}) + E_{\mathrm{F},n}(\mathbf{x})]/kT\Big) \tag{6.2.5.11}$$

$$p(\mathbf{x}) = \sum_{i\in\mathrm{VB}} N^i_{\mathrm{v}}(T)\; \mathcal{F}_{1/2}\Big([E^i_{\mathrm{v}}(\mathbf{x}) - e\phi(\mathbf{x}) - E_{\mathrm{F},p}(\mathbf{x})]/kT\Big). \tag{6.2.5.12}$$

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

$$N_l^i(T) = g_l^i \left(\frac{m_{\mathrm{dos},\lambda}^i kT}{2\pi\hbar} \right)^{2/3} \qquad (l,\lambda) = (\mathrm{v,h}),\ \text{or}\ (\mathrm{c,e}). \tag{6.2.5.13}$$

Here $m_{\mathrm{dos},\lambda}^i$ is the density-of-mass for $d = 3$ defined in (6.2.5.10).

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.

Poisson equation

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:

$$-\nabla \cdot [\varepsilon_0 \varepsilon_r(\mathbf{x}) \nabla \cdot \phi(\mathbf{x})] = \rho(\mathbf{x}, \phi) \tag{6.2.5.14}$$

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:

$$\rho(\mathbf{x}, \phi) = e[-n(\mathbf{x}, \phi) + p(\mathbf{x}, \phi) + N_D^+(x) - N_A^-(\mathbf{x}) + \rho_{\mathrm{pz}}(\mathbf{x}) + \rho_{\mathrm{py}}(\mathbf{x})] \tag{6.2.5.15}$$

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 (6.2.5.11) and (6.2.5.12) 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**.

Ionized donor/acceptor densities

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

$$N_{\mathrm{D}}^+(\mathbf{x}) = \sum_{i \in \mathrm{Donors}} \frac{N_{\mathrm{D},i}(\mathbf{x})}{1 + g_{\mathrm{D},i} \exp((E_{\mathrm{F},n}(\mathbf{x}) - E_{\mathrm{D},i}(\mathbf{x}))/k_{\mathrm{B}}T)} \tag{6.2.5.16}$$

$$N_{\mathrm{A}}^-(\mathbf{x}) = \sum_{i \in \mathrm{Acceptors}} \frac{N_{\mathrm{A},i}(\mathbf{x})}{1 + g_{\mathrm{A},i} \exp((E_{\mathrm{A},i}(\mathbf{x}) - E_{\mathrm{F},p}(\mathbf{x}))/k_{\mathrm{B}}T)} \tag{6.2.5.17}$$

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 $E^{\mathrm{ion}}_{\mathrm{D},i}, E^{\mathrm{ion}}_{\mathrm{A},i}$, the bulk conduction and valence band edges (including shifts due to strain) and the electrostatic potential.

$$E_{\mathrm{D},i}(\mathbf{x}) = E_\mathrm{c}(\mathbf{x}) - e\phi(\mathbf{x}) - E^{\mathrm{ion}}_{\mathrm{D},i}(\mathbf{x}) \tag{6.2.5.18}$$

$$E_{\mathrm{A},i}(\mathbf{x}) = E_\mathrm{v}(\mathbf{x}) - e\phi(\mathbf{x}) + E^{\mathrm{ion}}_{\mathrm{A}}(\mathbf{x}) \tag{6.2.5.19}$$

Piezoelectric and pyroelectric charge densities

ρ_{pz} and ρ_{py} are calculated according to the result of strain equation. (*TO be updated*)

Current equation

Current equation

The continuity equations in the presence of creation (generation, G) or annihilation (recombination, R) of electron-hole pairs read

$$\begin{aligned} -e\frac{\partial n}{\partial t} + \nabla \cdot \big(-e\mathbf{j}_n(\mathbf{x})\big) &= -e\big(G(\mathbf{x}) - R(\mathbf{x})\big), \\ e\frac{\partial p}{\partial t} + \nabla \cdot e\mathbf{j}_p(\mathbf{x}) &= e\big(G(\mathbf{x}) - R(\mathbf{x})\big), \end{aligned} \tag{6.2.5.20}$$

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

$$\begin{aligned} \mathbf{j}_n(\mathbf{x}) &= -\mu_n(\mathbf{x})n(\mathbf{x})\nabla E_{\mathrm{F},n}(\mathbf{x}), \\ \mathbf{j}_p(\mathbf{x}) &= \mu_p(\mathbf{x})p(\mathbf{x})\nabla E_{\mathrm{F},p}(\mathbf{x}). \end{aligned} \tag{6.2.5.21}$$

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

$$\begin{aligned} \nabla \cdot \mu_n(\mathbf{x})n(\mathbf{x})\nabla E_{\mathrm{F},n}(\mathbf{x}) &= -(G(\mathbf{x}) - R(\mathbf{x})), \\ \nabla \cdot \mu_p(\mathbf{x})p(\mathbf{x})\nabla E_{\mathrm{F},p}(\mathbf{x}) &= G(\mathbf{x}) - R(\mathbf{x}), \end{aligned} \tag{6.2.5.22}$$

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 $\big(n(\mathbf{x}, \phi, E_{\mathrm{F},n}), p(\mathbf{x}, \phi, E_{\mathrm{F},p})\big)$ 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 satifies the convergence criteria, which can be tuned by the users from *run{ current_poisson{ } }* or *run{ quantum_current_poisson{ } }*.

Recombination/Generation

The recombination mechanisms that *nextnano++* takes into account for the right-hand-side of (6.2.5.20) 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*.)

Optoelectronic characteristics based on the semi-classical model

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.

- **Spontaneous emission rate**

$$R_{\mathrm{rad}}^{spon}(\mathbf{x}, E) = C(\mathbf{x}) \int dE_{\mathrm{h}} \int dE_{\mathrm{e}}\, n(\mathbf{x}, E_{\mathrm{e}}) p(\mathbf{x}, E_{\mathrm{h}}) \delta(E_{\mathrm{e}} - E_{\mathrm{h}} - E). \tag{6.2.5.23}$$

Here $C(x)$ [$\mathrm{cm^3 s^{-1}}$] 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 $R_{\mathrm{rad}}^{spon}(\mathbf{x}, E)$.

- **Stimulated emission rate**

Stimulated emission rate is calculated here as the net emission rate containing both the generation by the stimulated absorption and the recombination by the spontaneous and stimulated emission according to the following equation:

$$R_{rad,net}^{\mathrm{stim}}(\mathbf{x}, E) = \left(1 - e^{\frac{E - (E_{\mathrm{F}n} - E_{\mathrm{F}p})}{k_{\mathrm{B}} T}}\right) R_{\mathrm{rad}}^{spon}(\mathbf{x}, E) \tag{6.2.5.24}$$

The reference equation is eq.(9.2.39) of *[ChuangOpto1995]*.

$R_{\mathrm{rad}}^{spon}(E)$ and $R_{\mathrm{rad,net}}^{stim}(E)$ are output on *Optical/semiclassical_spectra_photons_~.dat* and *stim_emission_photons_~.dat* as the integral of the above two quantities over $\mathbf{x}$, i.e.

$$R_{\mathrm{rad}}^{spon}(E) = \int d\mathbf{x}\, R_{rad}^{\mathrm{spon}}(\mathbf{x}, E), \quad R_{\mathrm{rad,net}}^{stim}(E) = \int d\mathbf{x}\, R_{\mathrm{rad,net}}^{stim}(\mathbf{x}, E) \tag{6.2.5.25}$$

On the other hand, $R_{\mathrm{rad,net}}^{stim}(\mathbf{x})$ is obtained as the integral of $R_{\mathrm{rad,net}}^{stim}(\mathbf{x}, E)$ over the photon energy, which is written as **"radiative"** in the output file *recombination.dat*, i.e.

$$R_{\mathrm{rad,net}}^{stim}(\mathbf{x}) = \int dE\, R_{\mathrm{rad,net}}^{stim}(\mathbf{x}, E). \tag{6.2.5.26}$$

Note: Precisely speaking, $R_{\mathrm{rad,net}}^{stim}(\mathbf{x})$ is not directly integrated from $R_{\mathrm{rad,net}}^{stim}(\mathbf{x}, E)$ but calculated according to the equation

$$R_{\mathrm{rad,net}}^{stim}(\mathbf{x}) = C(x) n(x) p(x) \left(1 - e^{\frac{E_{\mathrm{F}p} - E_{\mathrm{F}n}}{k_{\mathrm{B}} T}}\right).$$

This is meanwhile equivalent to (6.2.5.26).

- **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 don't specify either of them, this recombination rate is always 0.

$$R_{\text{fixed}}(\mathbf{x}) = -\big(G(\mathbf{x}) \qquad \text{specified from structure}\big) - \big(\int dE\, G(E,\mathbf{x}) \text{ calculated according to the configuration in classical}\big). \tag{6.2.5.27}$$

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

- **photocurrent**

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

$$I_{\text{photo}} = e \cdot \left(\int d\mathbf{x}\, R^{stim}_{\text{rad,net}}(\mathbf{x}) + \int d\mathbf{x}\, R_{\text{fixed}}(\mathbf{x}) \right) \tag{6.2.5.28}$$

- **internal quantum efficiency**

is calculated as

$$\eta_{IQE} = \frac{I_{\text{photo}}}{I_{\text{total}}} \tag{6.2.5.29}$$

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

- **volume quantum efficiency**

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

$$\eta_{VQE} = \frac{R^{stim}_{\text{rad,net}} + R_{\text{fixed}}}{R_{\text{total}}} \tag{6.2.5.30}$$

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 $\eta_{\text{IQE1}} > \eta_{\text{IQE2}}$

Note: If you have any comments on the terminologies and definitions of these quantities, please send to support [at] nextnano.com..

Moreover, the electrical power and optical power are calculated and output in *power.dat*:

- **Power**

$$\sum_i V_{\text{i-th contact}} \cdot I_{\text{i-th contact}} \tag{6.2.5.31}$$

- **Absorbed-power**

$$\int dE d\mathbf{x}\, E \cdot G(E,\mathbf{x}) \tag{6.2.5.32}$$

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

- **Emitted-power**

$$\int dE dx\, E \cdot R^{spon}_{\text{rad}}(E,x) \tag{6.2.5.33}$$

Optoelectronic characteristics based on the quantum model

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

These quantities are now supported

- 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*.

References

- [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)
- [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
- [ChuangOpto1995]
  Physics of Optoelectronic Devices
  S. L. Chuang
  John Wiley & Sons, Inc., New York (1995)

Last update: nn/nn/nnnn

6.2.6 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

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 $\mu_{const}^{n,p}$. $\mu_{max}^{n,p}$ is the mobility due to bulk phonon (lattice) scattering. For all semiconductors the temperature dependent lattice mobility is modeled by a power law:

$$\mu_{const}^{n,p}(T) = \mu_{max}^{n,p} \cdot \left(\frac{T}{T_0}\right)^{-exponent}, \tag{6.2.6.1}$$

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 :

$$\mu^{n,p} = \mu_{min1}^{n,p} \cdot e^{-\frac{P_c^{n,p}}{N_D+N_A}} + \frac{\mu_{const}^{n,p} - \mu_{min2}^{n,p}}{1 + \left(\frac{N_D+N_A}{C_r^{n,p}}\right)^{\alpha^{n,p}}} - \frac{\mu_1^{n,p}}{1 + \left(\frac{C_s^{n,p}}{N_D+N_A}\right)^{\beta^{n,p}}} \tag{6.2.6.2}$$

with the reference mobility parameters $\mu_{min1}^{n,p}$, $\mu_{min2}^{n,p}$ 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 $\mu_{const}^{n,p}$ is determined by equation (6.2.6.1) (constant mobility-model), i.e. the values in the database under keyword *mobility_constant{}* are the same as under this keyword.

The values of the parameters were taken from the DESSIS documentation (2001).

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:

$$\mu^{n,p} = \mu_{min}^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_m^{n,p}} + \frac{\mu_d^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_d^{n,p}}}{1 + \left(\frac{N_D+N_A}{N_0^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_N^{n,p}}}\right)^{A_a^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_a^{n,p}}}}, \tag{6.2.6.3}$$

with the reference mobility parameter $\mu_{min}^{n,p}(T_0)$, 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 $N_A + N_D$.

The values of the parameters were taken from the DESSIS documentation (2001).

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:

$$\mu^{n,p} = \mu_{min}^{n,p} + \frac{\mu_{const}^{n,p} - \mu_{min}^{n,p}}{1 + \left(\frac{N_D+N_A}{N_0^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_N^{n,p}}}\right)^{A_a^{n,p} \cdot \left(\frac{T}{T_0}\right)^{\alpha_a^{n,p}}}}, \tag{6.2.6.4}$$

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 $\mu_{const}^{n,p}$ is determined by the constant mobility-model: equation (6.2.6.1). The formulas for the reference mobility parameter $\mu_{const}^{n,p}$ are

$$\mu_{min}^{n,p}(T) = \mu_{min}^{n,p}(T_0) \left(\frac{T}{T_0} \right)^{\alpha_m^{n,p}} \tag{6.2.6.5}$$

$$\mu_{min}^{n,p}(T) = \mu_{min}^{n,p}(T_0) \cdot \left(\frac{2}{3} \right)^{\alpha_m^{n,p}} \left(\frac{T}{200K} \right)^{\alpha_{m2}^{n,p}}, \tag{6.2.6.6}$$

where (6.2.6.5) applies to temperatures $T \geq 200K$ and (6.2.6.6) to temperatures $T < 200K$. The value $T = 200K$ can be changed by T_{Switch}. By setting $\alpha_m^{n,p} = \alpha_{m2}^{n,p}$ and $\alpha_a^{n,p} = 0$, (6.2.6.6) reduces to (6.2.6.5) 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.)

Simba

Attention: These models are implemented only in *nextnano*[3]

This is one possible model for the mobility parameter μ^n (for electrons) and μ^p (for holes) that is used in the drift-diffusion model. The model is taken from the SIMBA documentation *[CaugheyThomas1967]*. In this model the mobility depends on the three quantities: doping density, temperature and **E**-field. The contributions of these quantities to the mobility are calculated in the following order:

1) Doping concentration:

$$\mu^{n,p}(N_A + N_D) = \mu_{min}^{n,p} + \frac{\mu_D^{n,p}}{1 + \left(\frac{N_A + N_D}{N_{ref}^{n,p}} \right)^{\alpha^{n,p}}} \tag{6.2.6.7}$$

with minimum mobility $\mu_{min}^{n,p}$, reference doping density $N_{ref}^{n,p}$, reference mobility $\mu_D^{n,p}$, exponent $\alpha^{n,p}$ and concentration of ionized acceptors N_A and donors N_D. Note that the nominal doping concentration, as specified in the input file and not the ionized one, is used in *nextnano*[3].

2) Temperature:

$$\mu^{n,p}(T) = \mu_{max}^{n,p} \cdot \left(\frac{T}{T_0} \right)^{-\gamma^{n,p}}, \tag{6.2.6.8}$$

with temperature T, reference temperature T_0 and exponent for temperature dependence $\gamma^{n,p}$.

3) Electric field (*perpendicular*):

$$\mu^{n,p}(\mathbf{E}_\perp) = \frac{\mu^{n,p}}{\sqrt{1 + \frac{\|\mathbf{E}_\perp\|}{E_T^{n,p}}}} \tag{6.2.6.9}$$

with perpendicular electric field parameter $E_T^{n,p}$. It is possible to include/ exclude the perpendicular **E**-field dependence.

4) Electric field (*parallel*):

There are six different SIMBA models for including the impact of the parallel electric field:

Model 0

no dependence on parallel electric field

Model 1

$$\mu^{n,p}(\mathbf{E}) = \frac{\mu^{n,p}}{\left[1 + \left(\frac{\|\mathbf{E}\|}{E_p^{n,p}}\right)^{\alpha^{n,p}}\right]^{\beta^{n,p}}} \tag{6.2.6.10}$$

with exponents $\alpha^{n,p}$ and $\beta^{n,p}$. The temperature dependency of peak electric field is described by:

$$E_p^{n,p}(T) = E_0^{n,p} - d_E^{n,p} \cdot (T - T_0), \tag{6.2.6.11}$$

with temperature T, peak electric field $E_0^{n,p}$, temperature dependence parameter of peak electric field $d_E^{n,p}$ and reference temperature T_0.

Model 2

$$\mu^{n,p}(\mathbf{E}) = \frac{\mu^{n,p}}{\left[1 + \left(\mu^{n,p}\frac{\|\mathbf{E}\|}{v_s^{n,p}}\right)^{\kappa^{n,p}}\right]^{1/\kappa^{n,p}}} \tag{6.2.6.12}$$

with exponent $\kappa^{n,p}$ adn saturation velocity v_s^{n,p}(T). Temperature dependency of saturation velocity is described by:

$$v_s^{n,p}(T) = v_0^{n,p} - d_v^{n,p} \cdot (T - T_0), \tag{6.2.6.13}$$

with reference saturation velocity $v_0^{n,p}$, temperature dependence parameter of saturation velocity $d_v^{n,p}$ and reference Temperature $T_0^{n,p}$.

Model 3

$$\mu^{n,p}(\mathbf{E}) = \frac{\mu^{n,p} + v_s^{n,p}\frac{\|\mathbf{E}\|^3}{E_p^{n,p}}}{\left[1 + \left(\frac{\|\mathbf{E}\|}{(E_p^{n,p})^4}\right)^{\alpha^{n,p}}\right]^{\beta^{n,p}}} \tag{6.2.6.14}$$

Model 4

$$\mu^{n,p}(\mathbf{E}) = \frac{2\mu^{n,p}}{1 + \left[1 + \left(\mu^{n,p}\frac{\|\mathbf{E}\|}{v_s^{n,p}}\right)^{\kappa^{n,p}}\right]^{1/\kappa^{n,p}}} \tag{6.2.6.15}$$

Model 5

$$\mu^{n,p}(\mathbf{E}) = \frac{\mu^{n,p} + v_s^{n,p}\frac{(\|\mathbf{E}\|)^{\alpha^{n,p}-1}}{(E_p^{n,p})^{\alpha^{n,p}}}}{\left[1 + \left(\frac{\|\mathbf{E}\|}{E_p^{n,p}}\right)^{\alpha^{n,p}}\right]^{\beta^{n,p}}} \tag{6.2.6.16}$$

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*

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

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.*.

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

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.*.

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

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: nn/nn/nnnn

6.2.7 Model for optical spectra within Fermi's golden rule

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 - Intraband matrix elements and selection rules*
- *Optical absorption for interband and intersubband transitions*

6.2.8 Doping

Activation Energies

Table 6.2.8.1: Donor levels (n-type) in units of eV relative to conduction band edge

Donor Name	Energy	Source
n-As-in-Si	0.054	DESSIS
n-As-in-Si	0.049	American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
n-P-in-Si	0.045	DESSIS, American Institute of Physics Handbook, 3rd ed., McGraw-Hill, New York (1972)
n-Sb-in-Si	0.039	DESSIS
n-N-in-Si	0.045	DESSIS
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	DESSIS
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 in the *nextnano*[3] database file *database_nn3.in* or at this link

Table 6.2.8.2: Acceptor levels (p-type) in units of eV relative to valence band edge

Acceptor Name	Energy	Source
p-In-in-Si	0.16	DESSIS
p-B-in-Si	0.045	DESSIS, 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	DESSIS
p-C-in-GaAs	0.027	Landolt-Boernstein 1982

More parameters can be found in the *nextnano*[3] database file *database_nn3.in* or at this link

6.3 Material Database

Note: This section is under construction

6.3.1 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++*.

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.

```
668 binary_zb{
669     name    = Si
670     valence = IV_IV
671
672     # Some other groups
673 }
```

```
1596 binary_zb {
1597     name    = GaAs
1598     valence = III_V
1599
1600     # Some other groups
1601 }
```

```
7226 binary_wz {
7227     name    = GaN
7228     valence = III_V
7229
7230     # Some other groups
7231 }
```

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.

```
8855 binary_wz {
8856     name    = ZnO
8857     valence = II_VI
8858
8859     lattice_consts{
8860         # Some parameters
```

(continues on next page)

(continued from previous page)

```
    }

    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 don't change it here.

> **Attention:** Every single parameter for the model must be defined in these groups.

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.

```
1419 ##############################################################################
     ↪##############
1420 #          T E R N A R Y     A L L O Y S     --    IV - IV       V A L E N␣
     ↪C E
1421 ##############################################################################
     ↪#############
```

```
3181 ##############################################################################
     ↪##############
3182 #          T E R N A R Y     A L L O Y S     --    III - V       V A L E N␣
     ↪C E
3183 ##############################################################################
     ↪##############
```

```
6341 ##############################################################################
     ↪##############
6342 #          T E R N A R Y     A L L O Y S     --    II - VI       V A L E N␣
     ↪C E
6343 ##############################################################################
     ↪##############
```

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
1436     binary_x   = Ge
1437     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"
```

(continues on next page)

(continued from previous page)

```
    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: nn/nn/nnnn

6.3.2 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.

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.

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
1629         defpot_uniaxial_b = -2.0   defpot_uniaxial_d = -4.8
1630
1631         delta_SO          = 0.341
1632     }
1633
1634     # The rest of parameters
1635 }
```

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{
            delta_SO   =  0.15
        }
    }
}
```

or

```
database{
    ternary_zb {
        name = "Ga(1-x)In(x)As"
        valence_bands{
            delta_SO   =  0.15
        }
    }
```

(continues on next page)

(continued from previous page)

```
}
```

or with one of the remaining names, in your input file.

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.

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"
    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
} : {
```

(continues on next page)

(continued from previous page)

```
    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: nn/nn/nnnn

6.3.3 Interpolation Schemes

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, $Si_xGe_{1-x}$is a binary alloy (two elements), while $Ga_xIn_{1-x}As$ 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 Si_xGe_{1-x} and $Ga_xIn_{1-x}As$ 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.

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

$$P_{\mathrm{AB}}(x) = x \cdot P_{\mathrm{A}} + [1-x] \cdot P_{\mathrm{B}},$$

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.

III-III-V and II-II-VI
For alloys of type $A_xB_{1-x}C$, the scheme reads

$$P_{\mathrm{ABC}}(x) = x \cdot P_{\mathrm{AC}} + [1-x] \cdot P_{\mathrm{BC}},$$

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.

III-V-V and II-VI-VI
For alloys of type AB_xC_{1-x}, the scheme reads

$$P_{\mathrm{ABC}}(x) = x \cdot P_{\mathrm{AB}} + [1-x] \cdot P_{\mathrm{AC}},$$

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.

Syntax
Let's 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
    valence = III_V

    # All the parameters of InAs here (P_B)
}

ternary_zb{
```

(continues on next page)

(continued from previous page)

```
    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

$$P_{\mathrm{AB}}(x) = x \cdot P_{\mathrm{A}} + [1-x] \cdot P_{\mathrm{B}} - x\,[1-x] \cdot b_{\mathrm{AB}},$$

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, 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

$$P_{\mathrm{ABC}}(x) = x \cdot P_{\mathrm{AC}} + [1-x] \cdot P_{\mathrm{BC}} - x\,[1-x] \cdot b_{\mathrm{ABC}},$$

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, 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

$$P_{\mathrm{ABC}}(x) = x \cdot P_{\mathrm{AB}} + [1-x] \cdot P_{\mathrm{AC}} - x\,[1-x] \cdot b_{\mathrm{ABC}},$$

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, 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
    valence = III_V

    # All the parameters of InAs here (P_B)
}
```

(continues on next page)

(continued from previous page)

```
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 $\mathrm{A_xB_{1-x}}$, 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 $\mathrm{A_xB_{1-x}}$ 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 $\mathrm{A_xB_{1-x}C}$, 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 $\mathrm{A_xB_{1-x}C}$ 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 $\mathrm{AB_xC_{1-x}}$, 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 $\mathrm{AB_xC_{1-x}}$ 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's 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:

$$b_{\mathrm{AlGaAs}\to\mathrm{GaAs}} = b_{\mathrm{AlGaAs}}(0) = -0.127 + 1.310 \cdot 0 = -0.127,$$

and at x=1 to describe the interpolation for small amounts of Ga, near AlAs:

$$b_{\mathrm{AlGaAs}\to\mathrm{AlAs}} = b_{\mathrm{AlGaAs}}(1) = -0.127 + 1.310 \cdot 1 = 1.183.$$

Finally, the fraction-dependent bowing parameter is given by

$$b_{\mathrm{AlGaAs}}(x) = x \cdot b_{\mathrm{AlGaAs}\to\mathrm{AlAs}} + [1-x] \cdot b_{\mathrm{AlGaAs}\to\mathrm{GaAs}}$$

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.

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: Si_xGe_y Sn_{1-x-y}, $Al_xGa_yIn_{1-x-y}N$, and $GaP_xAs_ySb_{1-x-y}$.

IV-IV-IV

For alloys of type $A_xB_yC_{1-x-y}$, 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:

$$\begin{aligned} b'_{\mathrm{AB}}(x,y) &= \frac{x \cdot b_{\mathrm{AB}\to\mathrm{A}} + y \cdot b_{\mathrm{AB}\to\mathrm{B}}}{x+y} \\ b'_{\mathrm{AC}}(x,w) &= \frac{x \cdot b_{\mathrm{AC}\to\mathrm{A}} + w \cdot b_{\mathrm{AC}\to\mathrm{C}}}{x+w} \\ b'_{\mathrm{BC}}(y,w) &= \frac{y \cdot b_{\mathrm{BC}\to\mathrm{B}} + w \cdot b_{\mathrm{BC}\to\mathrm{C}}}{y+w}. \end{aligned}$$

The b_{ABC} is a three-component bowing parameter.

III-III-III-V and II-II-II-VI

For alloys of type $A_xB_yC_{1-x-y}D$, 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 $P'_{ABD}(x,y)$, $P'_{ACD}(x,w)$, and $P'_{BCD}(y,w)$ are linear combinations of parameters P_{AD}, P_{BD}, and P_{CD}.

$$\begin{aligned} P'_{ABD}(x,y) &= x \cdot P_{\mathrm{AD}} + y \cdot P_{\mathrm{BD}} \\ P'_{ACD}(x,w) &= x \cdot P_{\mathrm{AD}} + w \cdot P_{\mathrm{CD}} \\ P'_{BCD}(y,w) &= y \cdot P_{\mathrm{BD}} + w \cdot P_{\mathrm{CD}} \end{aligned}$$

The $b'_{\mathrm{ABD}}(x,y)$, $b'_{\mathrm{ACD}}(x,w)$, and $b'_{\mathrm{BCD}}(y,w)$ are two-component bowing parameters. They can be equal zero, constant or dependent on mole fraction as:

$$\begin{aligned} b'_{\mathrm{ABD}}(x,y) &= \frac{x \cdot b_{\mathrm{ABD}\to\mathrm{AD}} + y \cdot b_{\mathrm{ABD}\to\mathrm{BD}}}{x+y} \\ b'_{\mathrm{ACD}}(x,w) &= \frac{x \cdot b_{\mathrm{ACD}\to\mathrm{AD}} + w \cdot b_{\mathrm{ACD}\to\mathrm{CD}}}{x+w} \\ b'_{\mathrm{BCD}}(y,w) &= \frac{y \cdot b_{\mathrm{BCD}\to\mathrm{BD}} + w \cdot b_{\mathrm{BCD}\to\mathrm{CD}}}{y+w}. \end{aligned}$$

The b_{ABCD} is a three-component bowing parameter.

III-V-V-V and II-VI-VI-VI

For alloys of type $AB_xC_yD_{1-x-y}$, having $w = 1 - x - y$, the scheme reads

$$\begin{aligned} P_{\mathrm{ABCD}}(x,y) = {} & P'_{ABC}(x,y) + P'_{ABD}(x,w) + P'_{ACD}(y,w) \\ & - xy \cdot b_{\mathrm{ABC}}(x) - xw \cdot b_{\mathrm{ABD}}(x) - yw \cdot b_{\mathrm{ACD}}(x) \\ & - xyw \cdot b_{\mathrm{ABCD}}, \end{aligned}$$

The $P'_{ABC}(x,y)$, $P'_{ABD}(x,w)$, and $P'_{ACD}(y,w)$ are linear combinations of parameters P_{AB}, P_{AC}, and P_{AD}.

$$\begin{aligned} P'_{ABC}(x,y) &= x \cdot P_{\mathrm{AB}} + y \cdot P_{\mathrm{AC}} \\ P'_{ABD}(x,w) &= x \cdot P_{\mathrm{AB}} + w \cdot P_{\mathrm{AD}} \\ P'_{ACD}(y,w) &= y \cdot P_{\mathrm{AC}} + w \cdot P_{\mathrm{AD}} \end{aligned}$$

where b_{ABC}, b_{ABD}, and b_{ACD} are two-component bowing parameters and b_{ABCD} is a three-component bowing parameter.

The $b'_{\mathrm{ABC}}(x,y)$, $b'_{\mathrm{ABD}}(x,w)$, and $b'_{\mathrm{ACD}}(y,w)$ are two-component bowing parameters. They can be equal zero, constant or dependent on mole fraction as:

$$\begin{aligned} b'_{\mathrm{ABC}}(x,y) &= \frac{x \cdot b_{\mathrm{ABC}\to\mathrm{AB}} + y \cdot b_{\mathrm{ABC}\to\mathrm{AC}}}{x+y} \\ b'_{\mathrm{ABD}}(x,w) &= \frac{x \cdot b_{\mathrm{ABD}\to\mathrm{AB}} + w \cdot b_{\mathrm{ABD}\to\mathrm{AD}}}{x+w} \\ b'_{\mathrm{ACD}}(y,w) &= \frac{y \cdot b_{\mathrm{ACD}\to\mathrm{AC}} + w \cdot b_{\mathrm{ACD}\to\mathrm{AD}}}{y+w}. \end{aligned}$$

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 $Si_xGe_y\ Sn_{1-x-y}$. 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.

Four-component alloys

Four-component alloys with a stoichiometry $A_xB_{1-x}C_yD_{1-y}$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 $Ga_xIn_{1-x}P_xAs_{1-y}$.

III-III-V-V and II-II-VI-VI

For alloys of type $A_xB_{1-x}C_yD_{1-y}$, having $u = 1 - x$ and $v = 1 - y$, the scheme reads

$$\begin{aligned} P_{\mathrm{ABCD}}(x,y) = {} & xy \cdot P_{\mathrm{AC}} + uy \cdot P_{\mathrm{BC}} + xv \cdot P_{\mathrm{AD}} + uv \cdot P_{\mathrm{BD}} \\ & - xuy \cdot b'_{\mathrm{ABC}}(x,u) - xuv \cdot b'_{\mathrm{ABD}}(x,u) - xyv \cdot b'_{\mathrm{ACD}}(y,v) - uyv \cdot b'_{\mathrm{BCD}}(y,v) \\ & - xuyv \cdot b_{\mathrm{ABCD}} \end{aligned}$$

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 $A_xB_yC_zD_{1-x-y-z}$, having $w = 1 - x - y - z$, the scheme reads

$$\begin{aligned} P_{\mathrm{ABCD}}(x,y) = {} & x \cdot P_{\mathrm{A}} + y \cdot P_{\mathrm{B}} + z \cdot P_{\mathrm{C}} + w \cdot P_{\mathrm{D}} \\ & - xy \cdot b'_{\mathrm{AB}}(x,y) - xz \cdot b'_{\mathrm{AC}}(x,z) - xw \cdot b'_{\mathrm{AD}}(x,w) \\ & - yz \cdot b'_{\mathrm{BC}}(y,z) - yw \cdot b'_{\mathrm{BD}}(y,w) - zw \cdot b'_{\mathrm{CD}}(z,w) \\ & - xyz \cdot b'_{\mathrm{ABC}} - xyw \cdot b'_{\mathrm{ABD}} - xzw \cdot b'_{\mathrm{ACD}} - yzw \cdot b'_{\mathrm{BCD}} \\ & - xyzw \cdot b_{\mathrm{ABCD}} \end{aligned}$$

```
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
}
```

Six-component alloys

III-III-III-V-V and II-II-II-VI-VI

For alloys of type $A_xB_yC_{1-x-y}D_zE_{1-z}$, having $u = 1 - x - y$ and $w = 1 - z$, the scheme reads

$$
\begin{aligned}
P_{\mathrm{ABCDE}}(x,y,z) = {} & xz \cdot P_{\mathrm{AD}} + yz \cdot P_{\mathrm{BD}} + uz \cdot P_{\mathrm{CD}} \\
& + xw \cdot P_{\mathrm{AE}} + yw \cdot P_{\mathrm{BE}} + uw \cdot P_{\mathrm{CE}} \\
& - xyz \cdot b'_{\mathrm{ABD}}(x,y) - xuz \cdot b'_{\mathrm{ACD}}(x,u) - yuz \cdot b'_{\mathrm{BCD}}(y,u) \\
& - xyw \cdot b'_{\mathrm{ABE}}(x,y) - xuw \cdot b'_{\mathrm{ACE}}(x,u) - yuw \cdot b'_{\mathrm{BCE}}(y,u) \\
& - xzw \cdot b'_{\mathrm{ADE}}(z,w) - yzw \cdot b'_{\mathrm{BDE}}(z,w) - uzw \cdot b'_{\mathrm{CDE}}(z,w) \\
& - xyzw \cdot b'_{\mathrm{ABDE}} - xuzw \cdot b'_{\mathrm{ACDE}} - yuzw \cdot b'_{\mathrm{BCDE}} \\
& - xyuz \cdot b'_{\mathrm{ABCD}} - xyuw \cdot b'_{\mathrm{ABCE}} \\
& - xyuzw \cdot b_{\mathrm{ABCDE}}
\end{aligned}
$$

```
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)
```

(continues on next page)

(continued from previous page)

```
    quaternary_abc_d{ TYPE=string }      #  A(x)B(y)C(1-x-y)D
    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
}
```

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: nn/nn/nnnn

6.3.4 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

Insulators and Metals

- SiO_2
- HfO_2
- Air
- Air_wz
- Al_2O_3 (sapphire)

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

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}
- $Zn_xCd_{1-x}S$

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

- $Zn_{1-x-y}Be_xMn_ySe$

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: nn/nn/nnnn

6.3.5 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 6.3.5.1. 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.

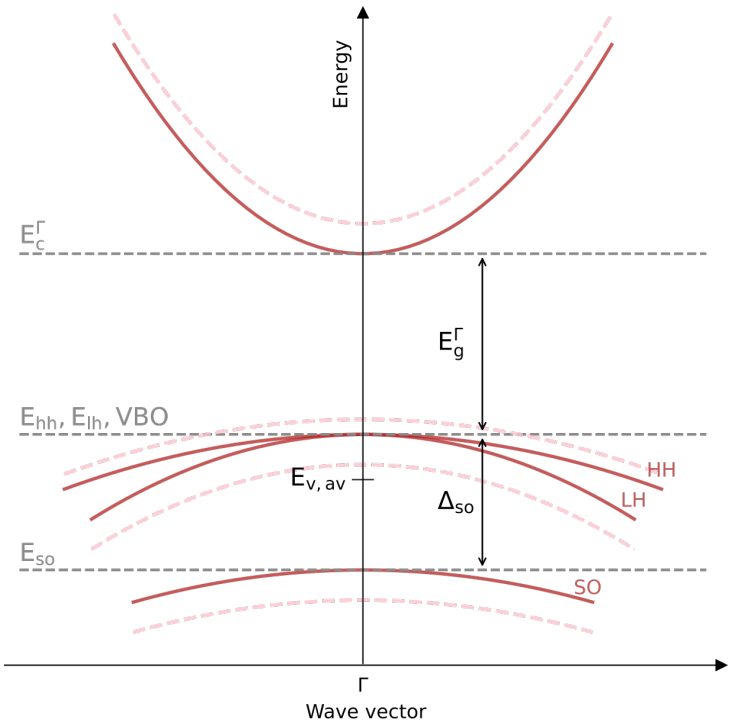


Figure 6.3.5.1: Band structure of freestanding (red solid lines) and compressively strained (pink dashed lines) in vicinity of Γ point.

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 6.3.5.2 and Figure 6.3.5.3.

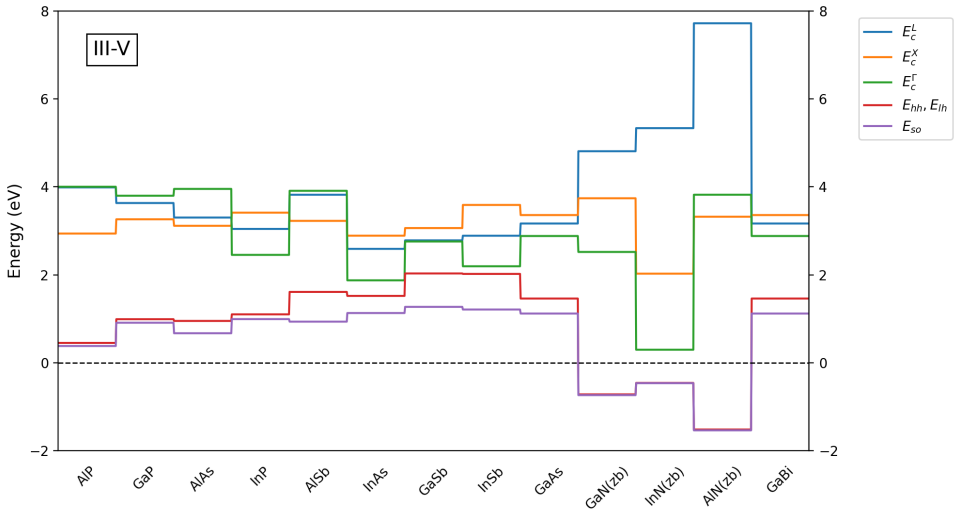


Figure 6.3.5.2: Band offsets of III-V zincblende binary compounds calculated with default parameters predefined in our database

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:

$$\begin{aligned}
E_c^X &= E_{v,av} + \frac{1}{3}\Delta_{so} + E_g^X \\
E_c^L &= E_{v,av} + \frac{1}{3}\Delta_{so} + E_g^L \\
E_c^{\Gamma} &= E_{v,av} + \frac{1}{3}\Delta_{so} + E_g^{\Gamma} \\
E_{hh} = E_{lh} &= E_{v,av} + \frac{1}{3}\Delta_{so} \\
E_{so} &= E_{v,av} - \frac{2}{3}\Delta_{so}
\end{aligned}$$

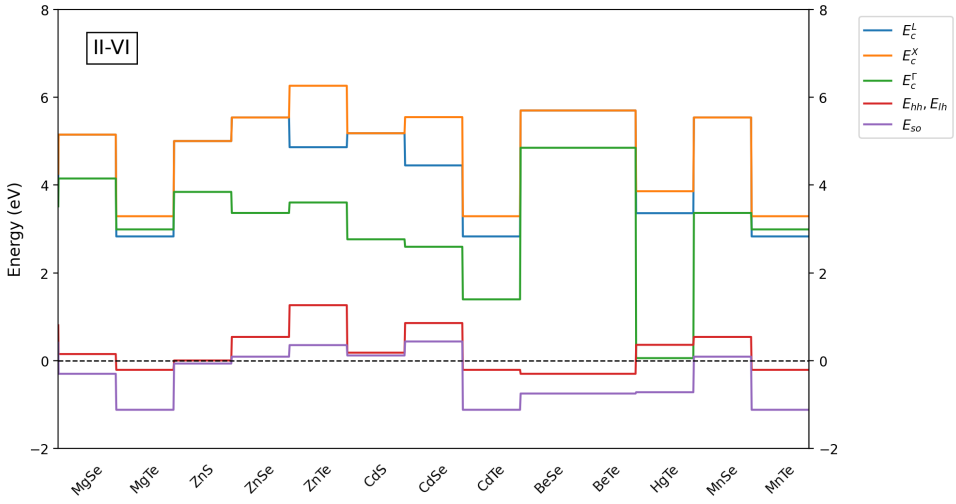


Figure 6.3.5.3: Band offsets of II-VI zincblende binary compounds calculated with default parameters predefined in our database

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 ($Ga_xIn_{1-x}As$, $Al_xGa_{1-x}As$, and $In_xAl_{1-x}As$) within full mole fraction ranges are shown in Figure 6.3.5.4. As visible for $Al_xGa_{1-x}As$, 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.

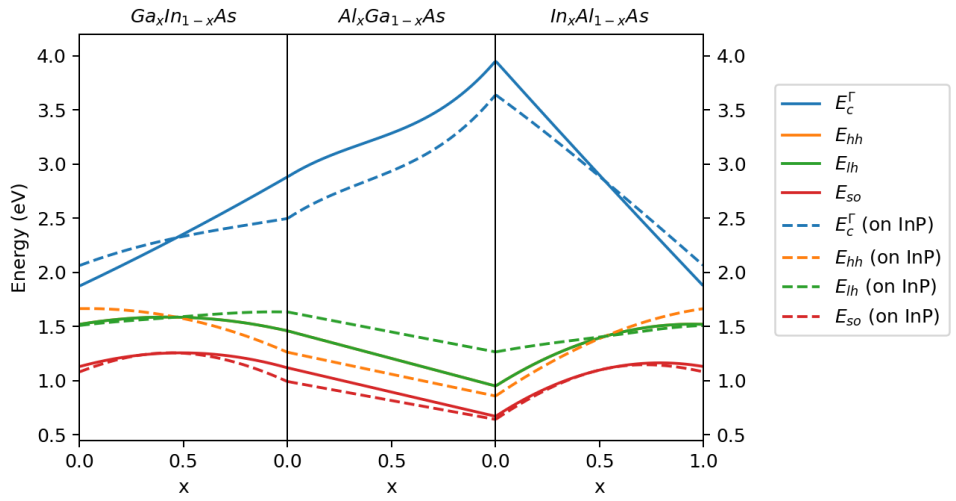


Figure 6.3.5.4: Interpolated band edges of $Ga_xIn_{1-x}As$, $Al_xGa_{1-x}As$, and $In_xAl_{1-x}As$ without strain (solid lines) and strained as grown on [1 0 0] plane of InP (dashed lines).

Last update: nn/nn/nnnn

6.4 Tutorials

6.4.1 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.

6.4.2 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.

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 6.4.2.1.

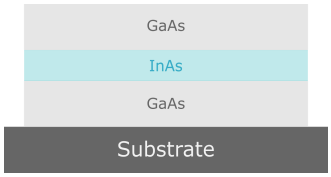


Figure 6.4.2.1: A schematic of a GaAs/InAs/GaAs heterostructure

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
```

(continues on next page)

(continued from previous page)

```
                            # 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 6.4.2.2 shows schematically the process of defining the grid.

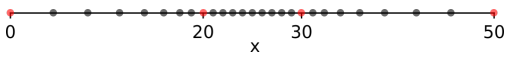


Figure 6.4.2.2: Schematics of the simulation grid with four "lines" defined (red circles). Interpolated grid points between lines are depicted with black circles.

One can also view the grid spacing using *nextnanomat*, see Figure 6.4.2.3.

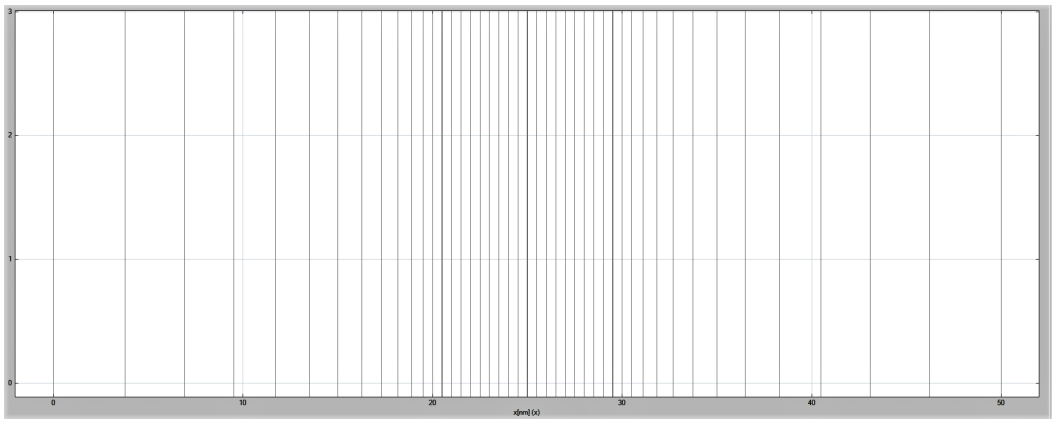


Figure 6.4.2.3: The numerical grid in the simulation.

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
```

(continues on next page)

(continued from previous page)

```
↪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.

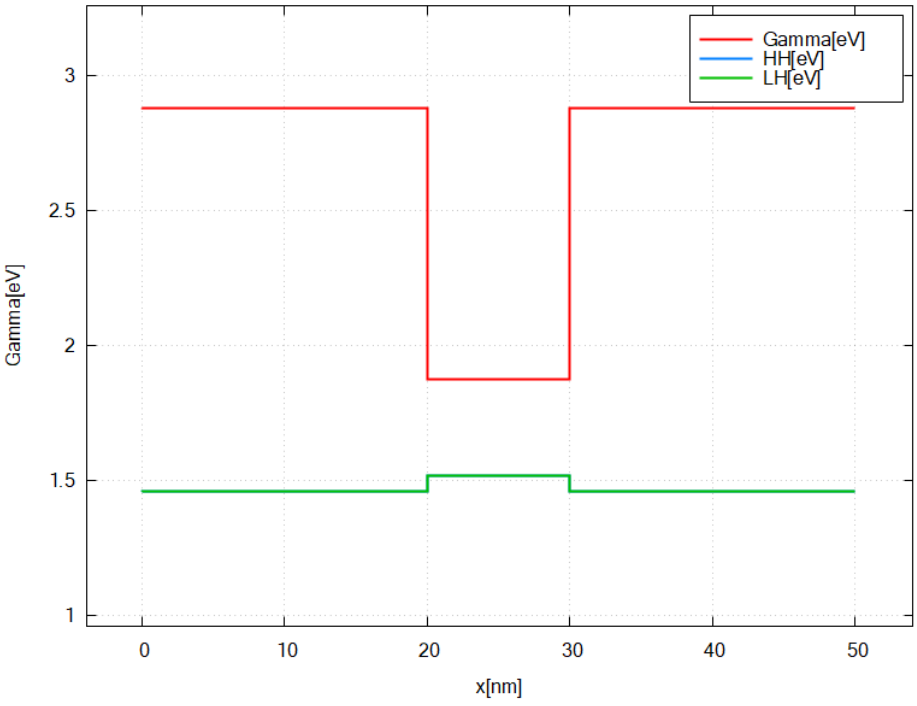


Figure 6.4.2.4: Energy profile of GaAs/InAs/GaAs heterostructure without considering strain.

Last update: 16/07/2024

— FREE — Finite Periodic Structures

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 6.4.2.5.

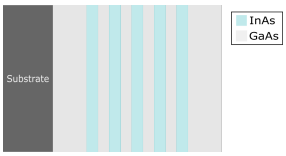


Figure 6.4.2.5: shows multiple GaAs/InAs quantum wells, which forms a finite superlattice

In the second part, we extend the input file of part one, and create the structure shown in Figure 6.4.2.6.

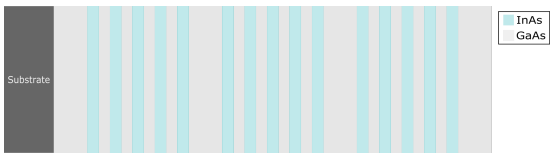


Figure 6.4.2.6: shows a sequence of three GaAs/InAs superlattices

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 $x = 20\text{nm}$ to $x = 30\text{nm}$. 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
```

(continues on next page)

(continued from previous page)

```
        }
                                # from x=-20.0 nm to x=70.0 nm further grid points
                                # are created according to the interpolated spacing (0.5 -
↪> 0.5)
                                # (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 6.4.2.7.

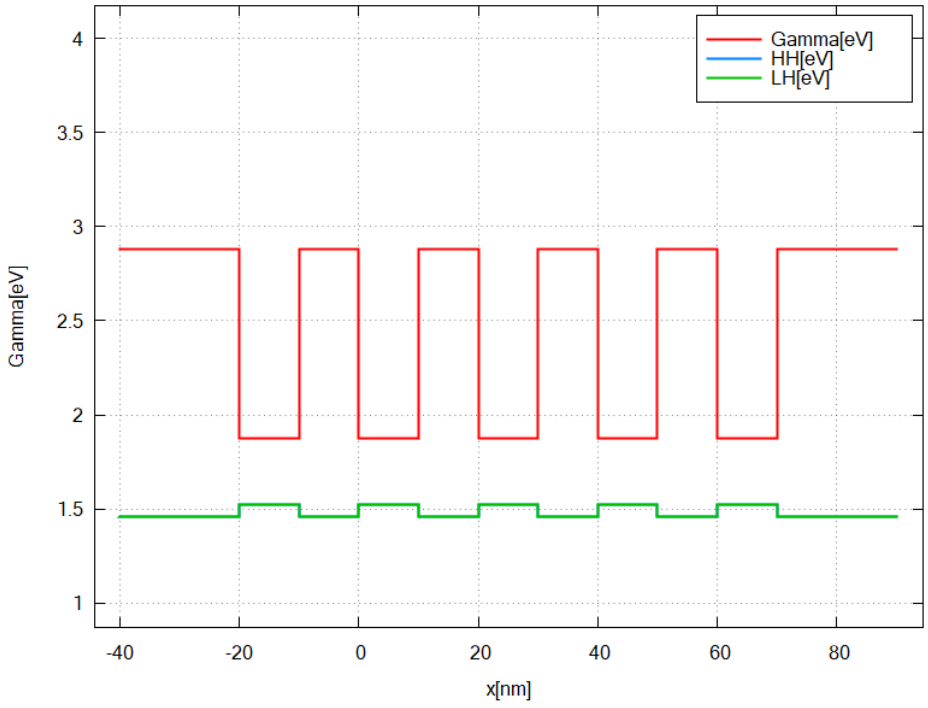


Figure 6.4.2.7: shows energy profile of multiple quantum well structure

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)
60                 shift = 120.0        # shifted by 120.0 nm
61                 max = 2              # two copies are created at x=-20.0 nm+i*shift (1
↪<=i<=max)
62             }
63         }
64
65         line{                        # fixed grid points are created in the middle of␣
↪two multiple QW structures to change grid spacing
66             pos = 85.0               # position: x=85.0 nm
67             spacing = 4.0            # grid spacing 4.0 nm
68
69             array{                   # fixed grid point at x=85.0 nm is duplicated␣
↪(including spacing)
70                 shift = 120.0        # shifted by 150.0 nm
71                 max = 1              # one copy is created at x=85.0 nm+max*shift
72             }
73         }
74
75         line{                        # fixed grid points are created at the top GaAs/
↪InAs interfaces of every multiple QW structure
76             pos = 70.0               # top InAs/GaAs interface at x=70.0 nm
77             spacing = 0.5            # grid spacing 0.5 nm
78
79             array{                   # fixed grid point at x=70.0 nm is duplicated␣
↪(including spacing)
80                 shift = 120.0        # shifted by 120.0 nm
81                 max = 2              # two copies are created at x=70.0 nm+i*shift (1
↪<=i<=max)
82             }
83         }
84                                      # from x=310.0 nm to x=340.0 nm further grid␣
↪points
85                                      # are created according to the spacings (0.5 -> 4.
↪0),
86                                      # which is interpolated (no equidistant spacing)
87         line{
88             pos = 340.0              # end device at x=340.0 nm
89             spacing = 4.0            # grid spacing 4.0 nm
90         }
91     }
92 }
```

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 6.4.2.8.

Just for demonstration, Figure 6.4.2.9 shows a screenshot of the employed grid.

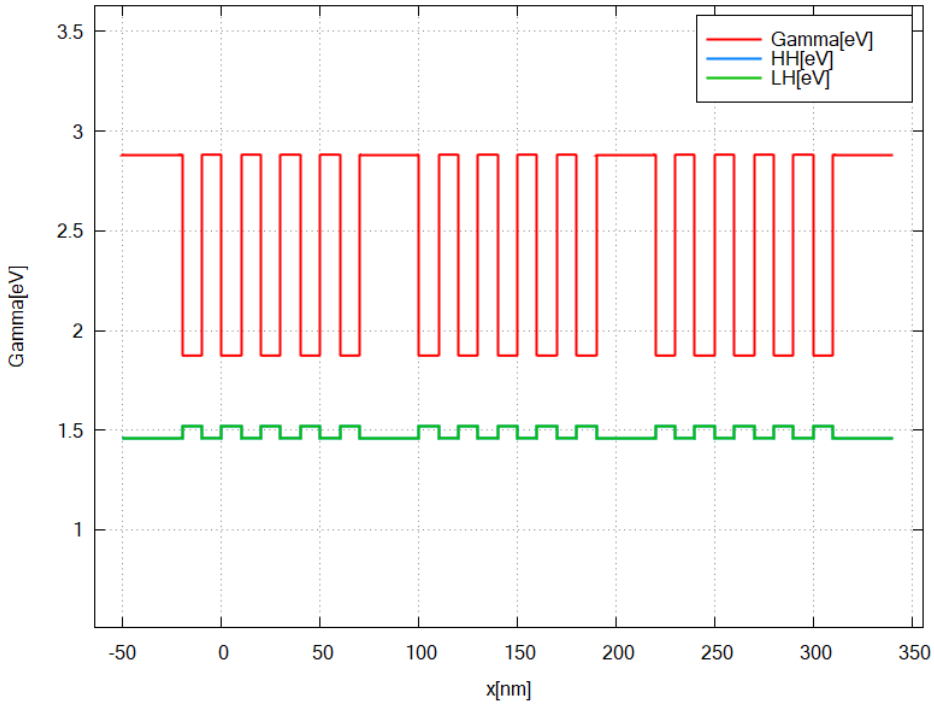


Figure 6.4.2.8: shows the band edges of conduction band at gamma point (Gamma), heavy hole (HH) and light hole (LH) of the complete structure

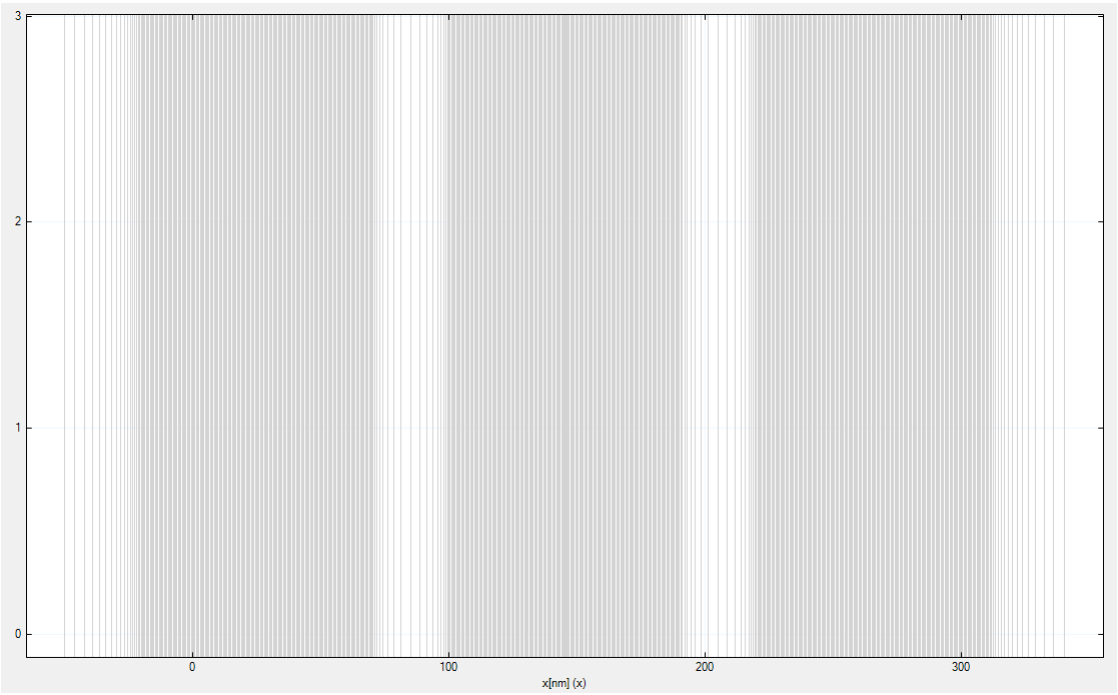


Figure 6.4.2.9: Numerical grid (gray).

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`
- Creating a sequence of periodic structures with `array2{}` works equivalently to `array{}`
- Don't 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 6.4.2.10 shows the two structures that will be created in this tutorial.

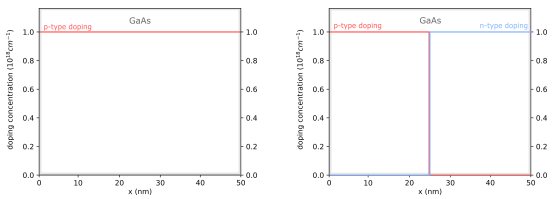


Figure 6.4.2.10: shows p-doped GaAs (left) and p-doped/ n-doped GaAs (right).

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 6.4.2.10. 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 6.4.2.11.

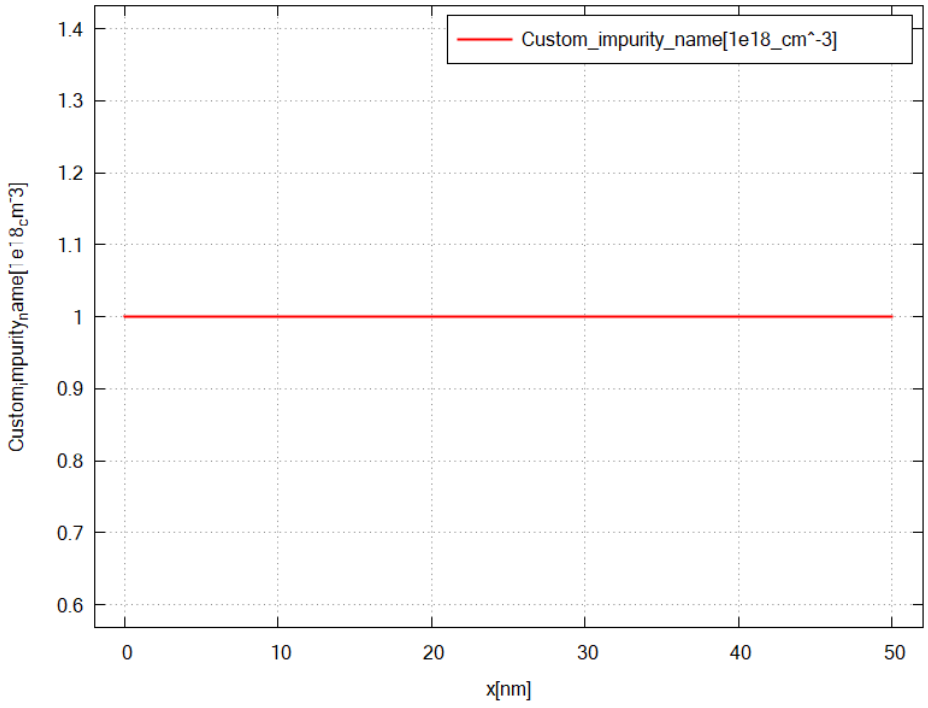


Figure 6.4.2.11: shows the doping concentration of acceptors along the x direction.

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 6.4.2.10 (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]

    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)
```

(continues on next page)

(continued from previous page)

```
        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 6.4.2.12 and Figure 6.4.2.13.

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

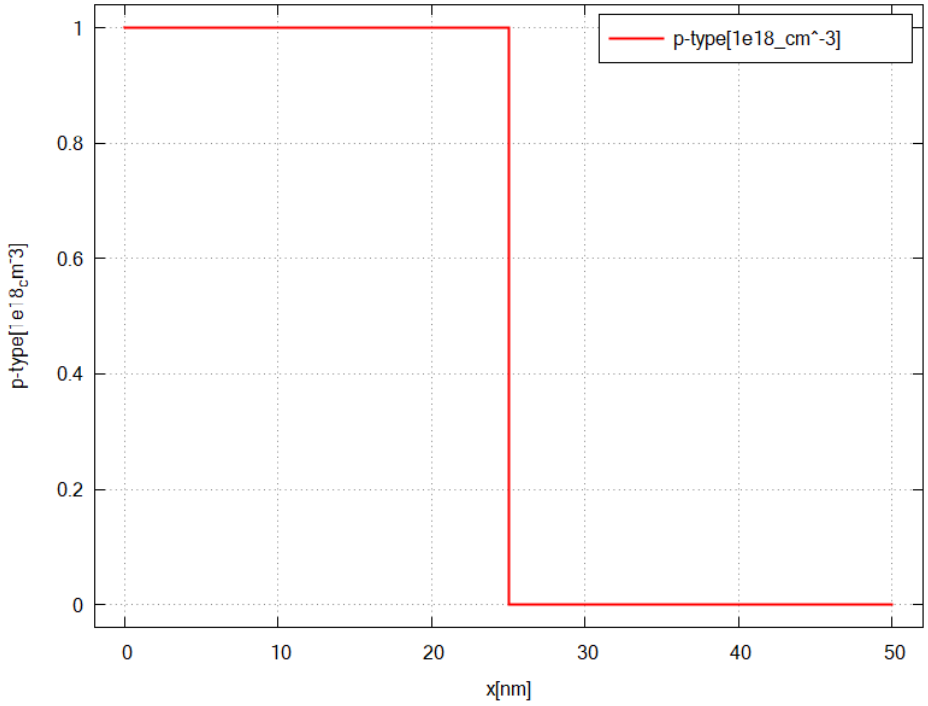


Figure 6.4.2.12: shows the doping concentration of acceptors along the x direction (p-doped region).

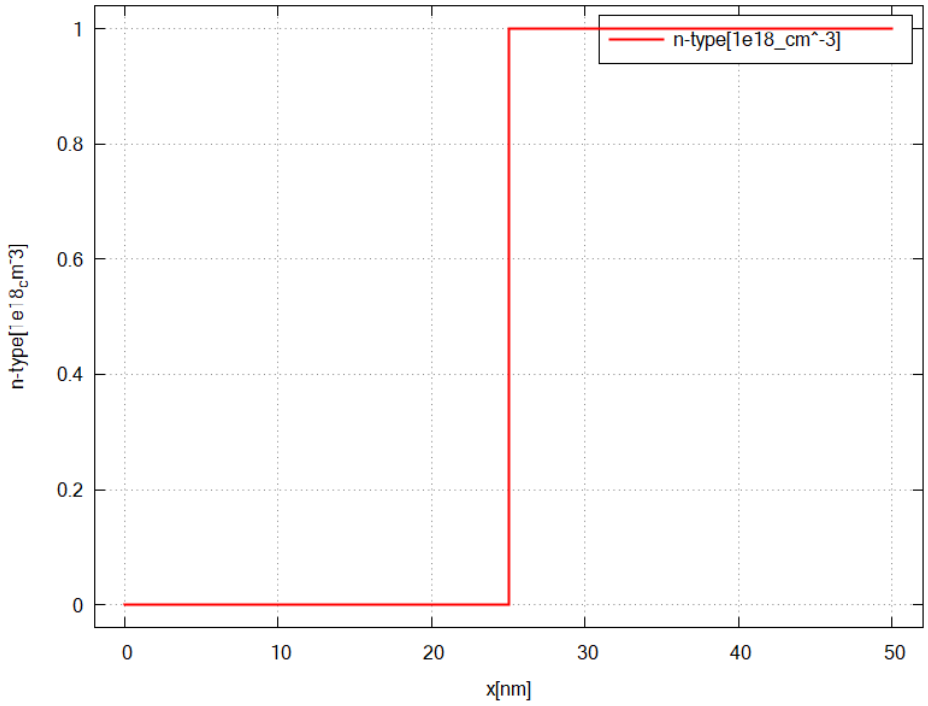


Figure 6.4.2.13: shows the doping concentration of donors along the x direction (n-doped region).

— FREE — Adding and Replacing Doping

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 6.4.2.14.

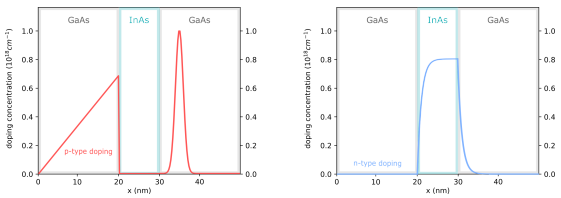


Figure 6.4.2.14: GaAs/InAs/GaAs heterostructure with p-type doping (left) and with different doping (right)

1. Replace and remove doping

We will now consider the structure in Figure 6.4.2.14 (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]

```

(continues on next page)

(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=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 6.4.2.15

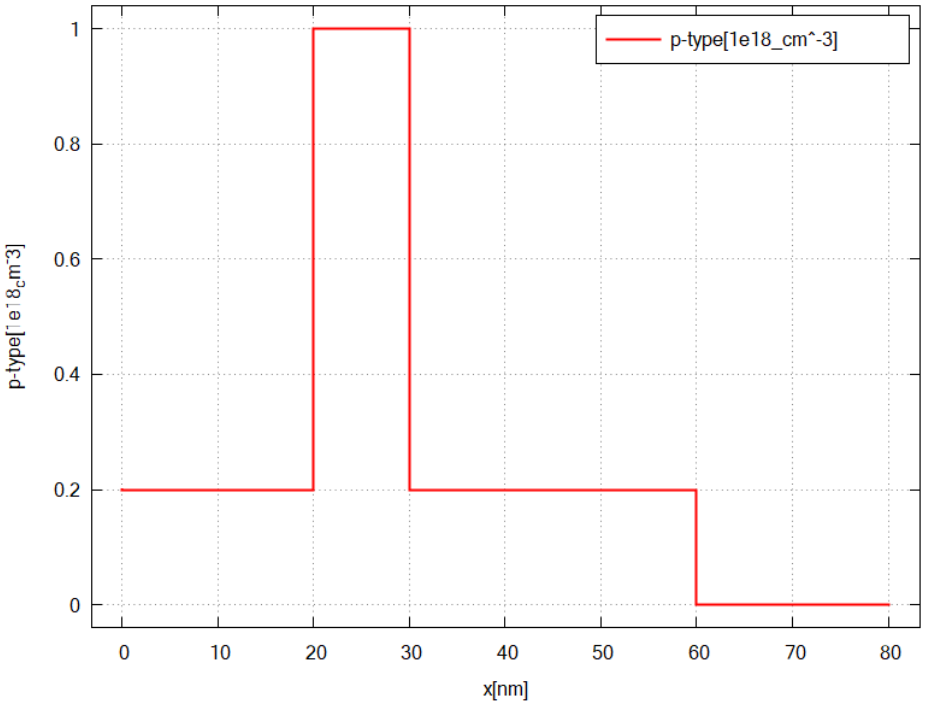


Figure 6.4.2.15: Doping concentration of donors along the x direction.

2. Add different dopants

We will now consider the structure in Figure 6.4.2.14 (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]
            }
        }
    }

    region{
```

(continues on next page)

(continued from previous page)

```
        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 6.4.2.16

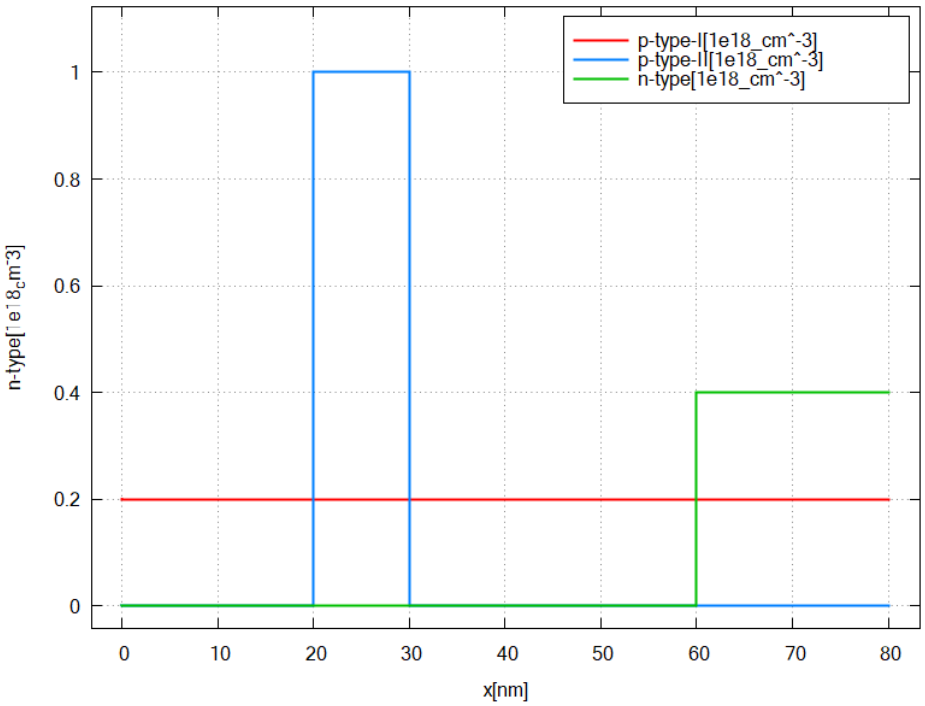


Figure 6.4.2.16: Doping concentration of donors/ acceptors along the x direction.

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 6.4.2.17 shows all the structures that will be created in this tutorial.

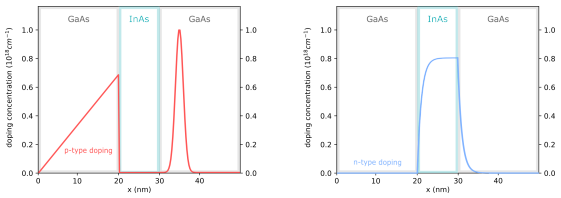


Figure 6.4.2.17: shows doping profiles including linear and Gaussian functions (left) and user defined functions (right).

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 6.4.2.17 (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
```

(continues on next page)

(continued from previous page)

```
    }

    #-------
    # 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
                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 6.4.2.18.

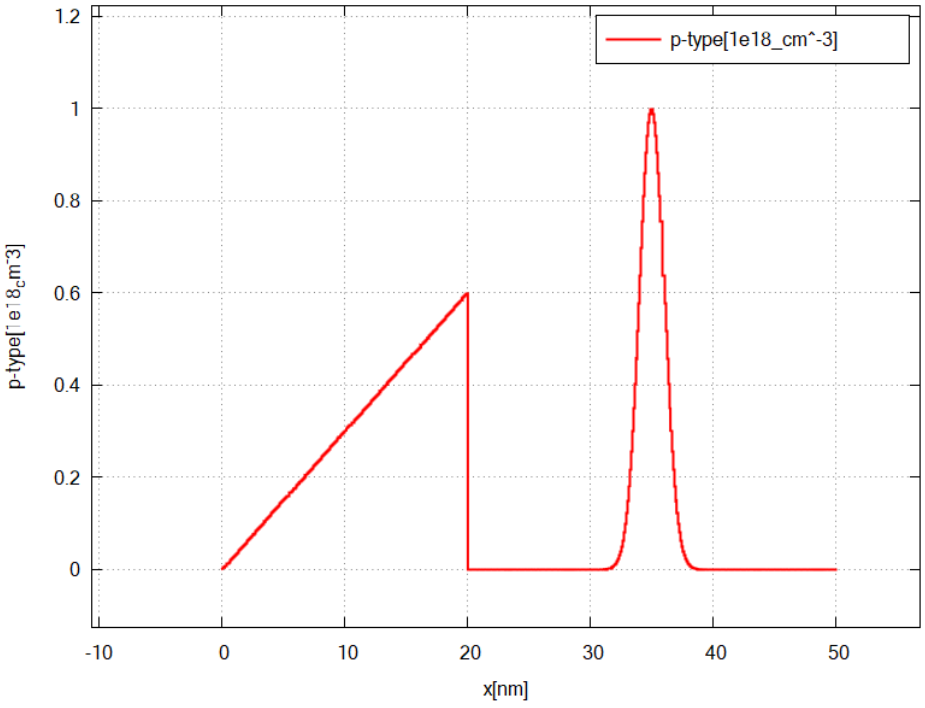


Figure 6.4.2.18: shows the doping concentration of donors along x.

2. Using custom doping profiles

In this example we introduce custom defined doping profiles. For that we consider the set up in Figure 6.4.2.17 (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
    }
    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
    #---------

```

(continues on next page)

(continued from previous page)

```
    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
    #-------

    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 6.4.2.19.

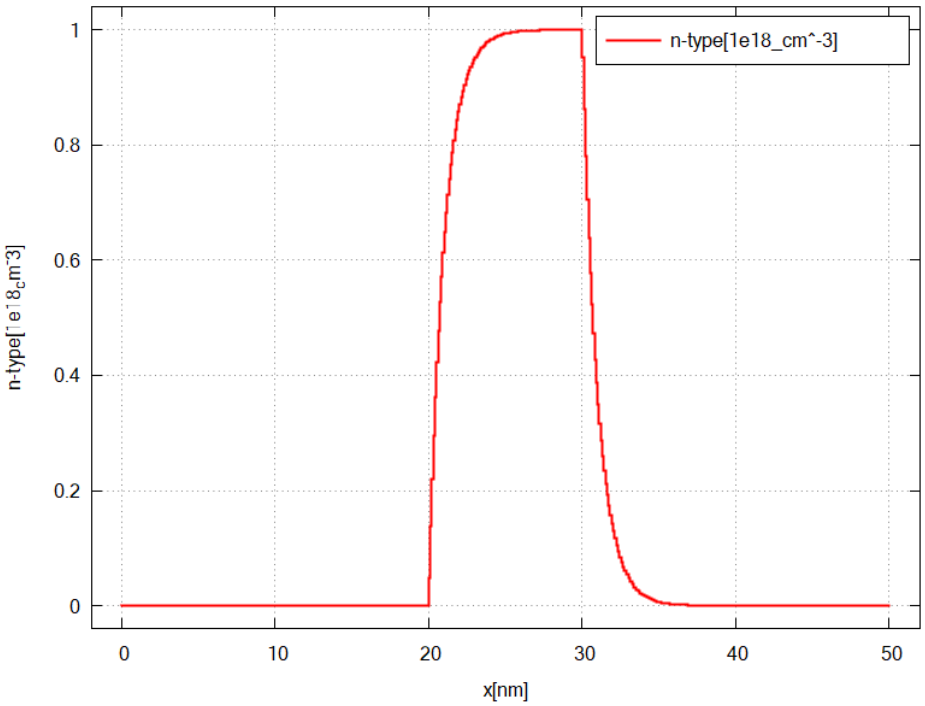


Figure 6.4.2.19: The doping concentration of donors along the x direction.

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

— 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 6.4.2.20). The device structure is shown in Figure 6.4.2.20.

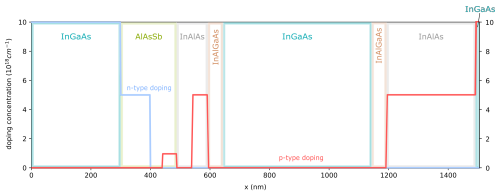


Figure 6.4.2.20: shows structure with doping profile

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
```

(continues on next page)

(continued from previous page)

```
    }

    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
```

(continues on next page)

(continued from previous page)

```
                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]
            }
        }
    }

    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)
        }
    }
}
```

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 6.4.2.21). 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`.

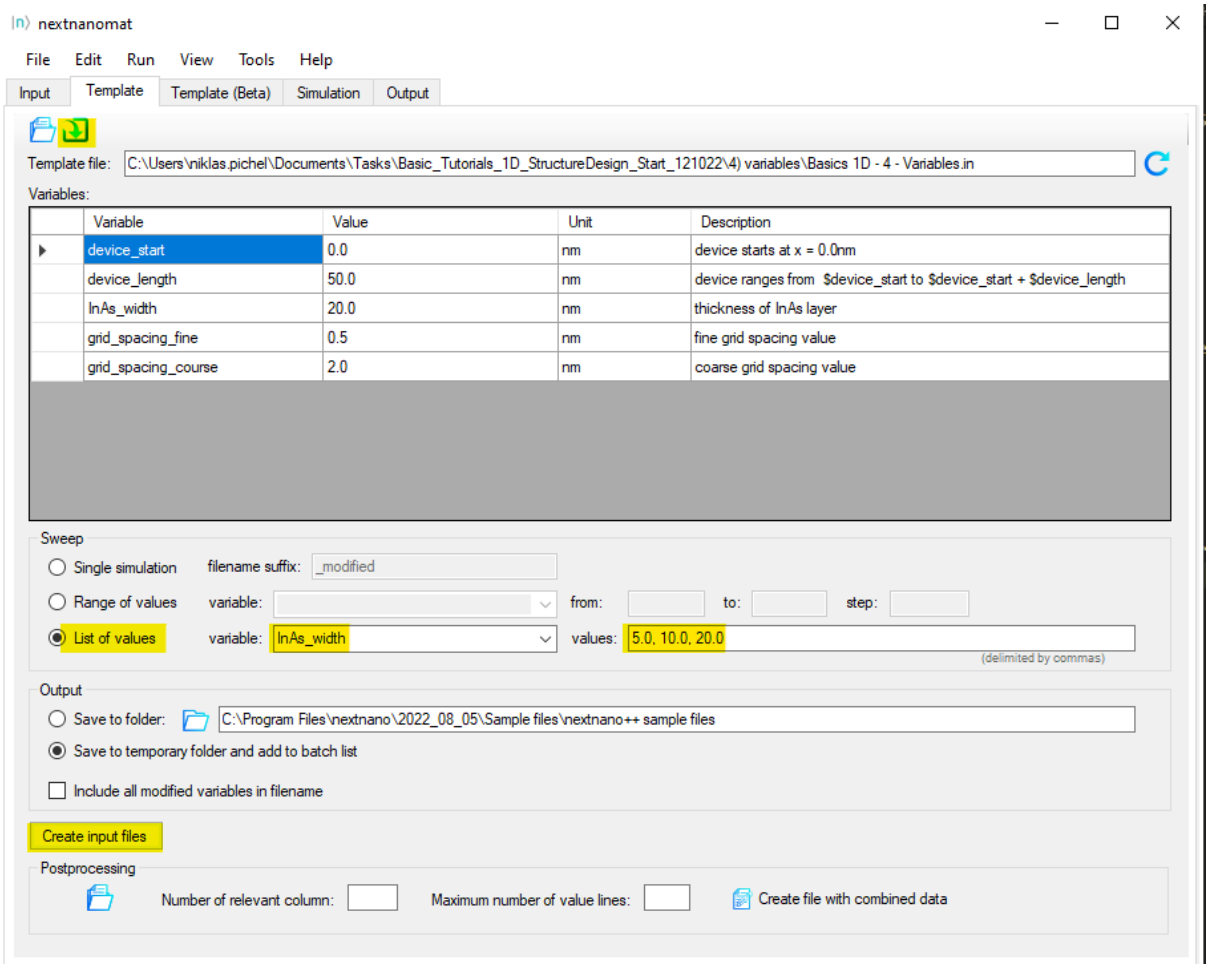


Figure 6.4.2.21: Screenshot showing *nextnanomat* interface to initialize the sweep:

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

After running the simulation you should find an output folder for every sweep value: `basics_1D_variables_InAs_width_<SweepValue>`. Figure 6.4.2.22 shows the overlay of energy profiles from every sweep.

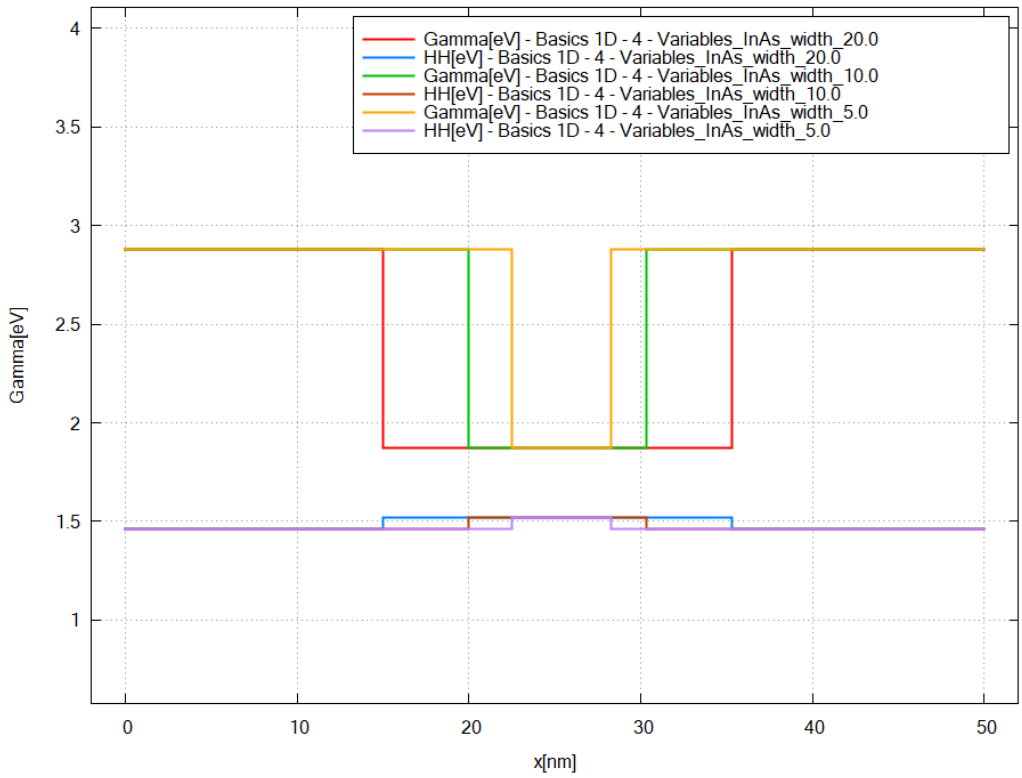


Figure 6.4.2.22: Overlay of energy profiles (conduction band at Γ and heavy hole valence band) corresponding to different InAs layer widths

Important things to remember

- Variables are defined by "$" + "Name of variable"

Last update: 16/07/2024

— SOON — Importing files

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's 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's 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's 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
        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{

    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
```

(continues on next page)

(continued from previous page)

```
↪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 6.4.2.23 shows the potential defined in the import files.

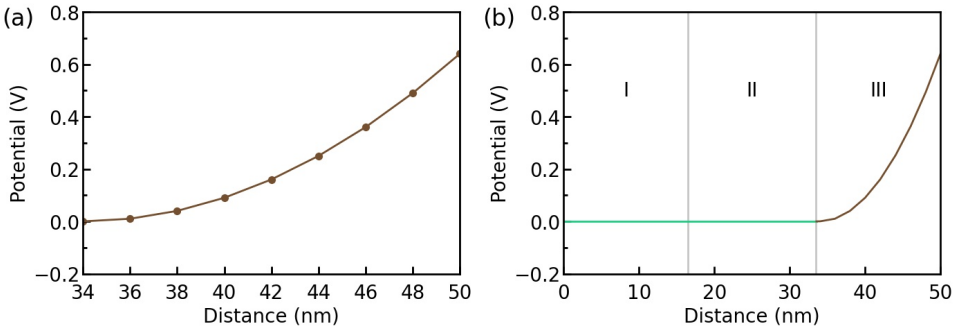


Figure 6.4.2.23: The potential introduced from the import file. The resulting potential in the entire structure.

Strain tensor

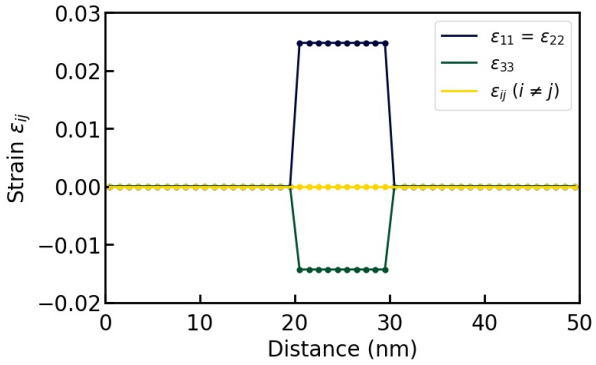


Figure 6.4.2.24: Imported strain tensor.

Alloy compositions

Figure 6.4.2.25 shows the alloy compositions in each region defined in the import files (a), (b) and the input file (c).

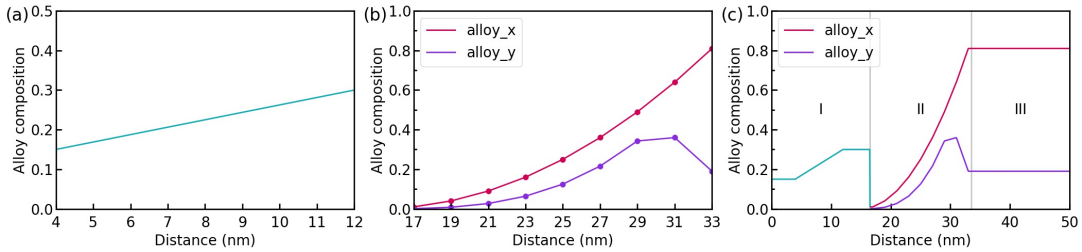


Figure 6.4.2.25: The alloy composition of Al(x)Ga(1-x)As is shown in (a). The alloy composition of Al(x)Ga(y)In(1-x-y)As is shown in (b) (The violet line: x, The purple line: y). (c) shows the alloy compositions in the whole structure. Region I is Al(x)Ga(y)In(1-x-y)As, region II is Al(x)Ga(y)In(1-x-y)As, and region III is Al(x)Ga(1-x)As(y)Sb(1-y), respectively.

The grid points in Figure 6.4.2.25 are originated from the import files.

There are some important points you can see from Figure 6.4.2.25 (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 6.4.2.26).

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.

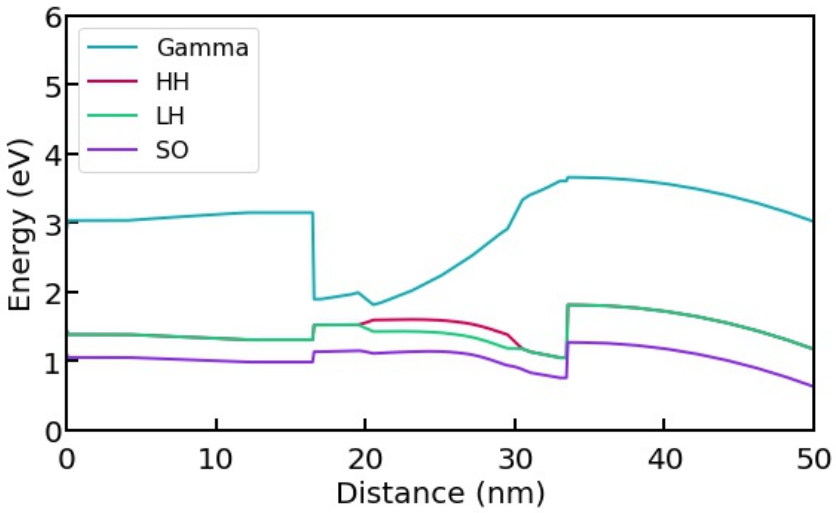


Figure 6.4.2.26: The band edges of the structure. The HH band and the LH band are degenerated in the region where there is no strain.

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 6.4.2.27 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

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.

— FREE — Schottky Barrier

- *Header*
- *Introduction*
- *Schottky Barrier*

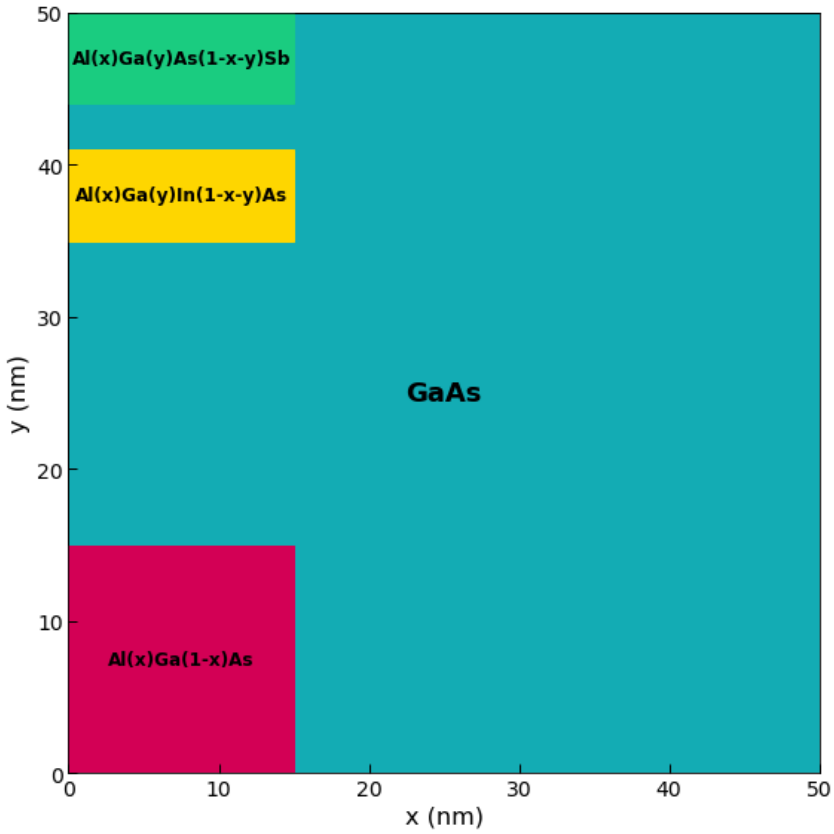


Figure 6.4.2.27: The geometry of the materials used in the 2D simulation. The dashed line is along x = 7.5 nm.

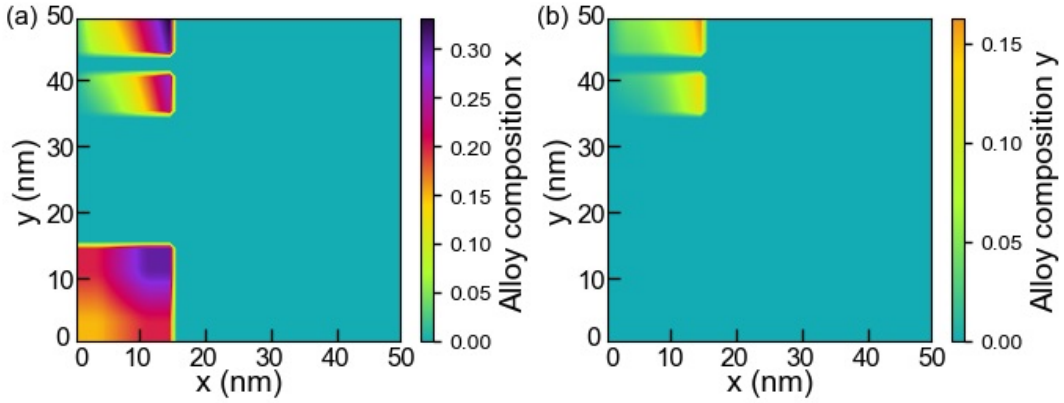


Figure 6.4.2.28: The alloy compositions of (a) x and (b) y of the materials used in the 2D simulation.

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

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.

The Schottky barrier model implemented in *nextnano++* corresponds to pinning of the Fermi level and does not take into account the work function of the metal. Due to such definition, the barrier height is independent of the metal work function and is entirely determined by the surface states and the doping.

```
contacts{
    schottky{                   # Schottky barrier (Fermi level pinning)
        name = contact
        bias = 0.0              # [V] apply voltage
        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 6.4.2.29, 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 6.4.2.29. At the left boundary, however, the band profile is affected by the type of contact.

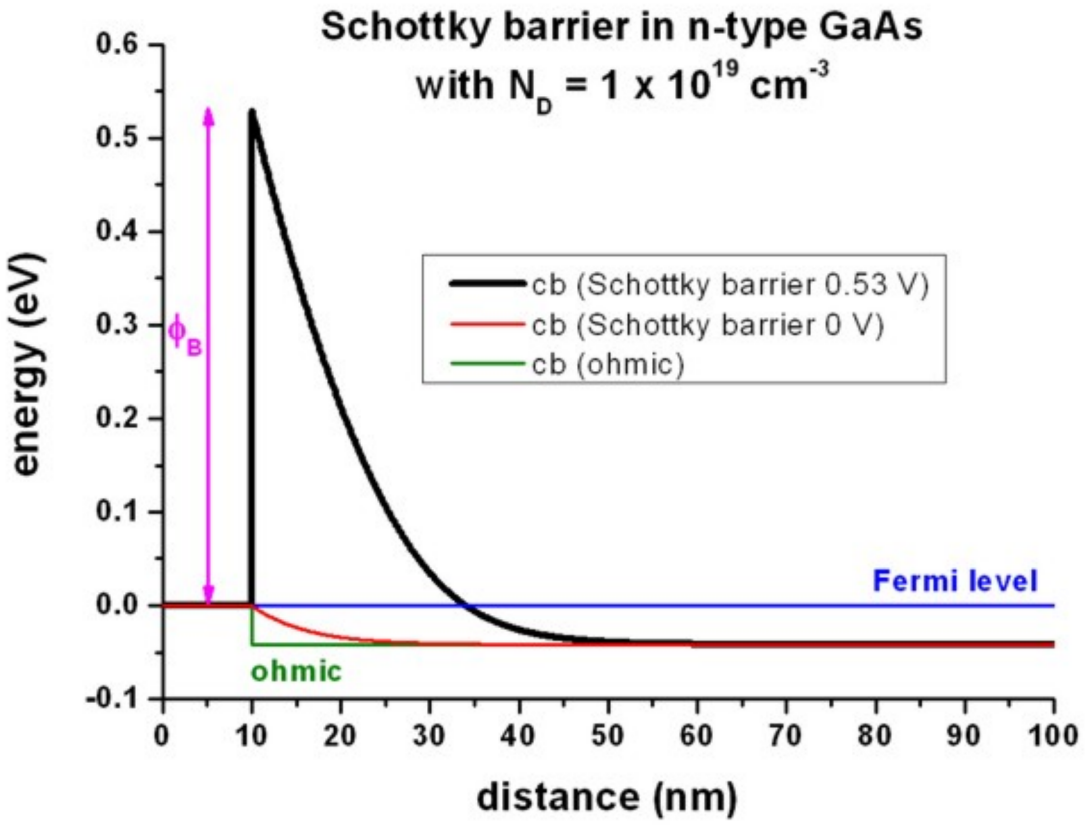


Figure 6.4.2.29: Calculated conduction band profile

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

— FREE — Surface Charges

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{
    ...
    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 6.4.2.30 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*

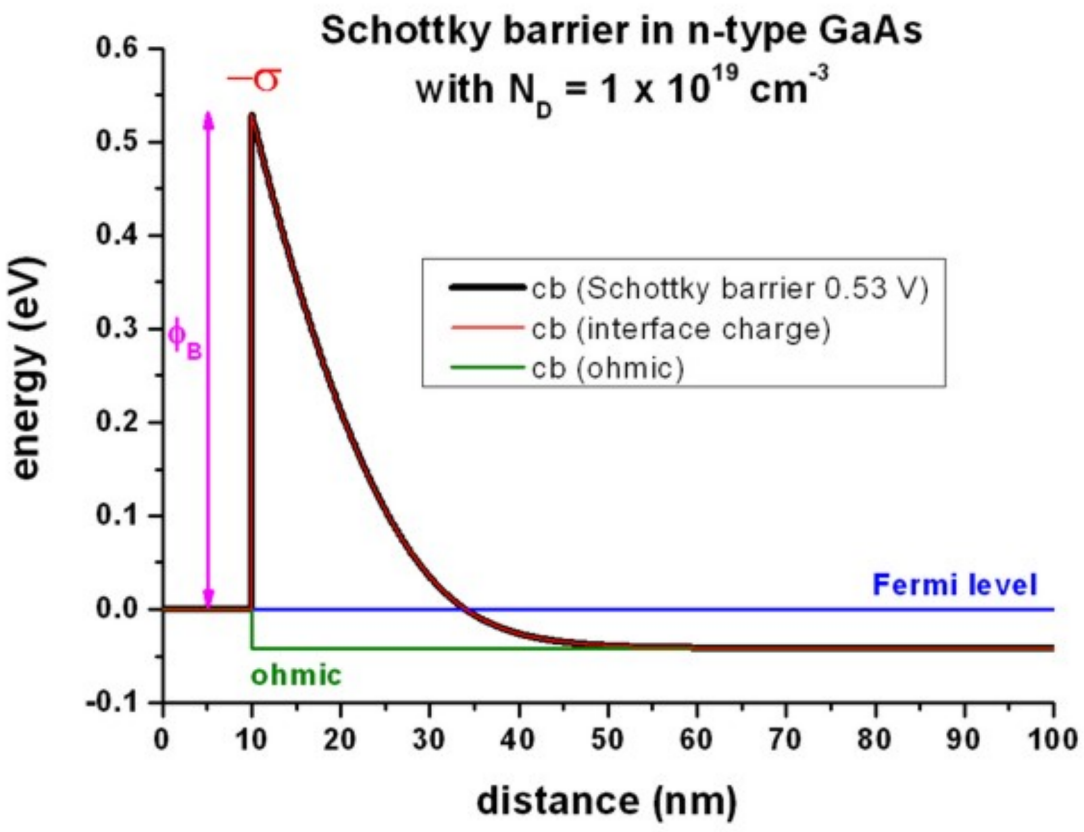


Figure 6.4.2.30: Calculated conduction band profile

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 $\cdot 10^{18}$ cm^{-3}. This corresponds to a sheet charge density of 8.4796 $\cdot 10^{12}$ cm^{-2} where we assume the states to have realistic *activation energies*.

```
impurities{
    ...
    acceptor{ # p-type
        name = "impurity_p"
        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 6.4.2.30 for the interface charges.

Last update: 16/07/2024

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 6.4.2.31 and Figure 6.4.2.32) 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^-).

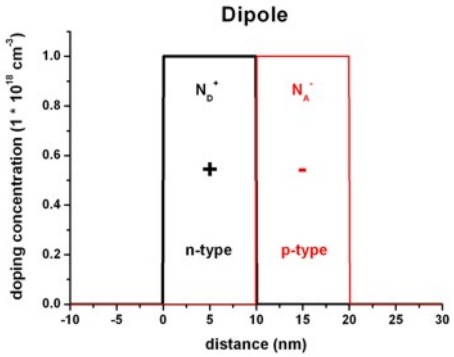


Figure 6.4.2.31: Doping distribution

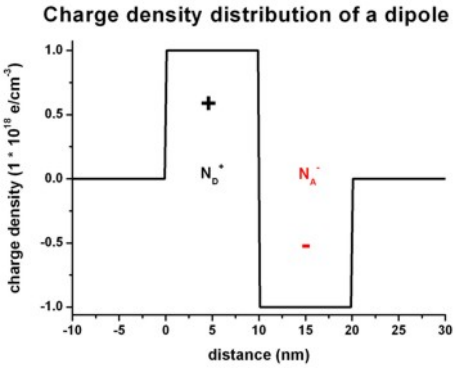


Figure 6.4.2.32: Charge density distribution

We have to solve the Poisson equation:

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

Figure 6.4.2.33 shows the corresponding electric field distribution and Figure 6.4.2.34 shows the electrostatic potential profile

The electric field is given by

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

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 $GaAs$.

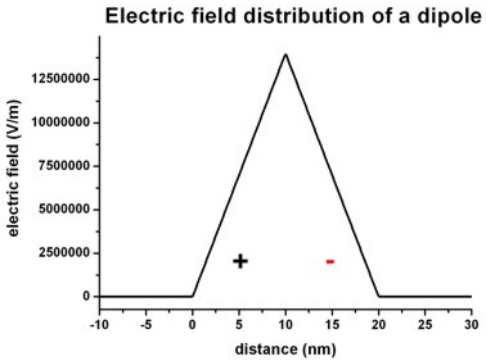


Figure 6.4.2.33: Electric field distribution

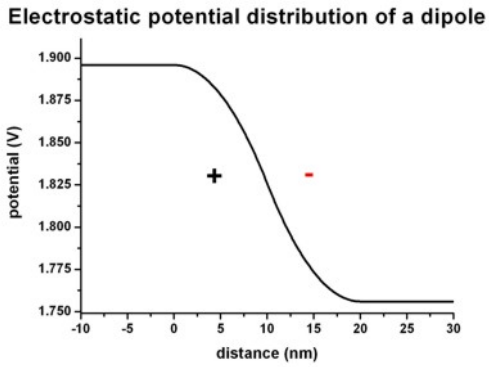


Figure 6.4.2.34: Electrostatic potential distribution

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 6.4.2.35 and Figure 6.4.2.36) 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 6.4.2.37 shows the corresponding electric field distribution and Figure 6.4.2.38 shows the electrostatic potential profile

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

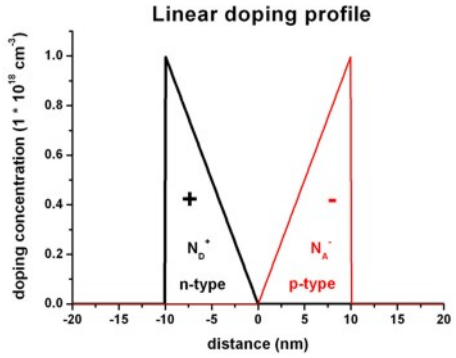


Figure 6.4.2.35: Doping profile

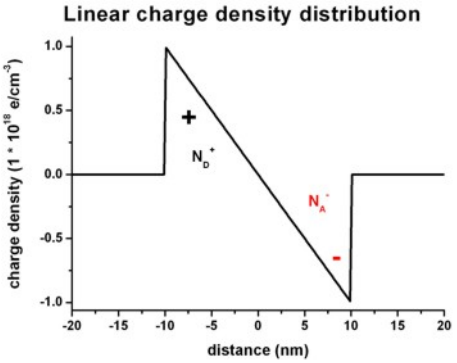


Figure 6.4.2.36: Charge density distribution

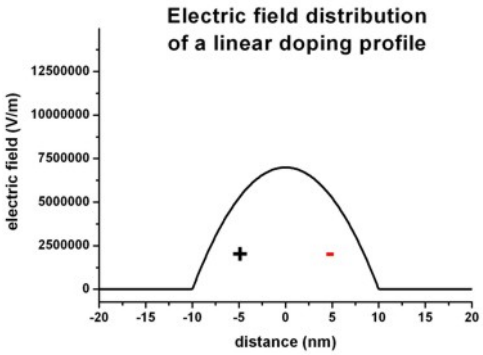


Figure 6.4.2.37: Electric field distribution

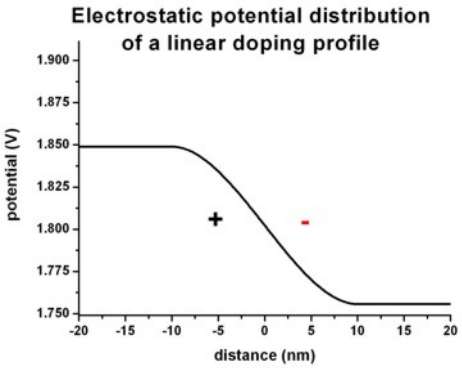


Figure 6.4.2.38: Electrostatic potential

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

Input file: *1D_Poisson_delta_nnpp.in*

The following figures (Figure 6.4.2.39 and Figure 6.4.2.40) 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^-).

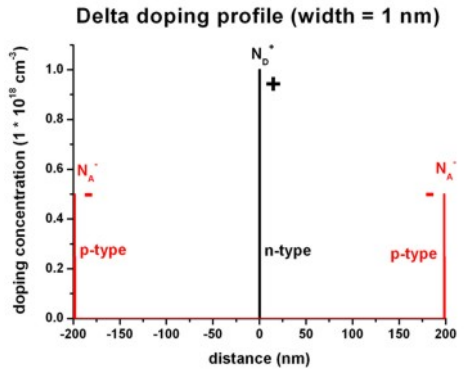


Figure 6.4.2.39: Doping profile

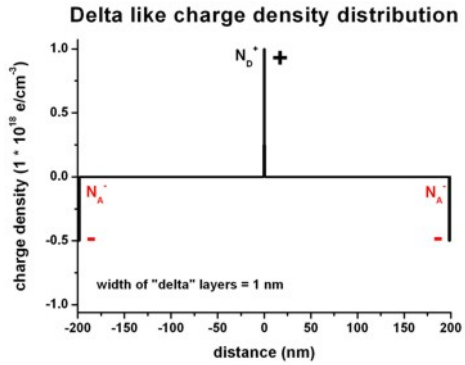


Figure 6.4.2.40: Charge density distribution

Figure 6.4.2.41 shows the corresponding electric field distribution and Figure 6.4.2.42 shows the electrostatic potential profile

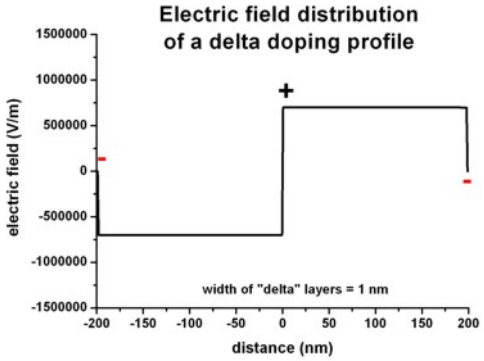


Figure 6.4.2.41: Electric field distribution

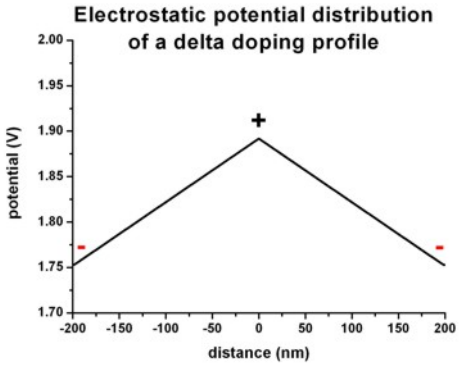


Figure 6.4.2.42: Electrostatic potential

Last update: nn/nn/nnnn

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:

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

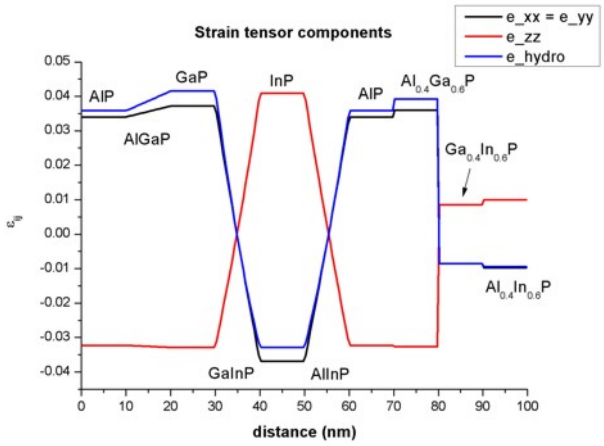


Figure 6.4.2.43: Strain tensor components

band gaps

Figure 6.4.2.44 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 6.4.2.45 compares the overall band gap to the case where band gap bowing has been neglected.

The *nextnano++* tool supports quaternaries in comparison with *nextnano*[3]:

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

(continues on next page)

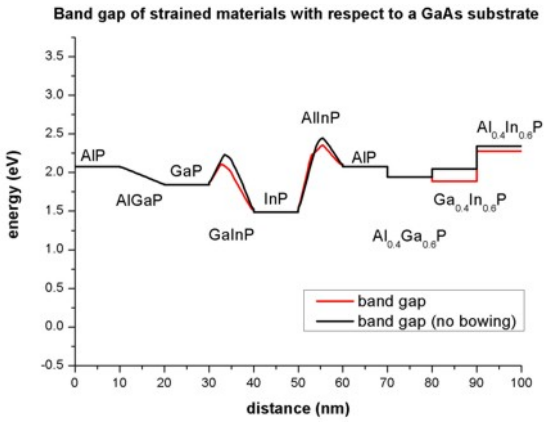


Figure 6.4.2.44: Band edge and band gap profile

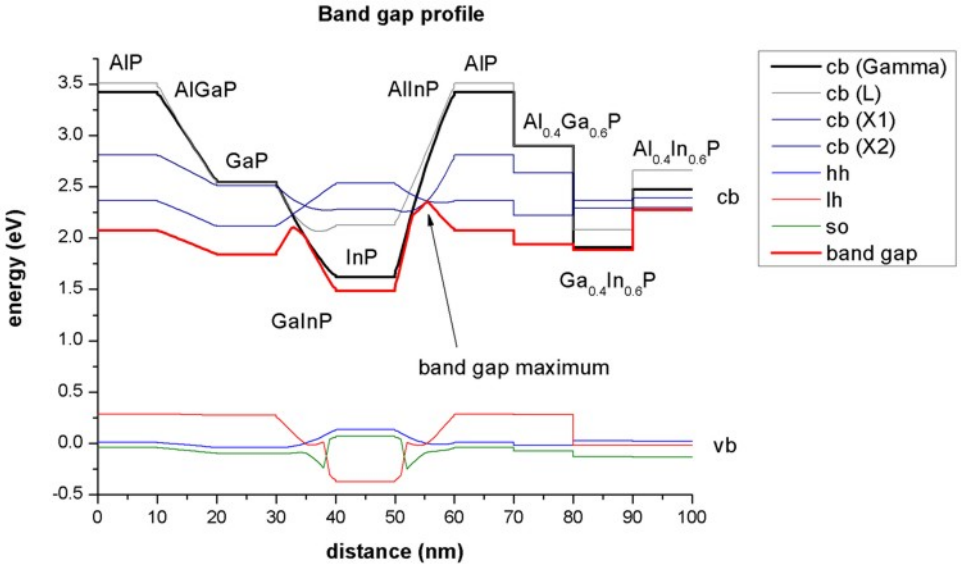


Figure 6.4.2.45: Ban dgap profile

(continued from previous page)

```
    alloy_y = 0.255
}
```

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

Last update: nn/nn/nnnn

— SOON/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*

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 $000\bar{1}$ 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 6.4.2.46 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.

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 opposite for GaN. This mechanism is explained in *[Chuang1996]*. Note that heavy and light hole are not degenerate under no-strain condition, unlike in zincblende crystals.

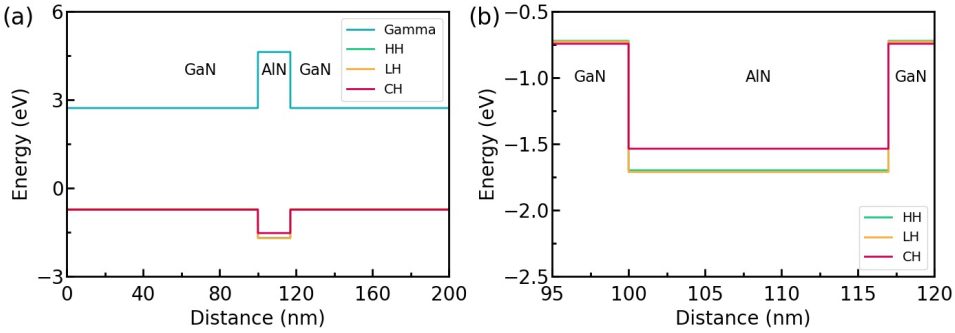


Figure 6.4.2.46: Calculated conduction and valence band structures without strain effects. (a) Full energy profile. (b) Valence band edges of AlN. (Run *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_offsets.in* to reproduce.)

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, $a_{\mathrm{AlN}} = 0.3112$ nm, smaller than the one of GaN, $a_{\mathrm{GaN}} = 0.3189$ nm, it becomes strained as follows.

The biaxial (in-plane) strain is tensile.

$$\varepsilon_{\parallel} = (a_{\mathrm{substrate}} - a)/a = 0.0247429$$

The uniaxial (growth direction) strain is compressive.

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

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

$$\varepsilon_{\mathrm{hy}} = Tr(\varepsilon_{\mathrm{ij}}) = (2\varepsilon_{\parallel} + \varepsilon_{\perp}) = 0.0351575$$

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

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

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.

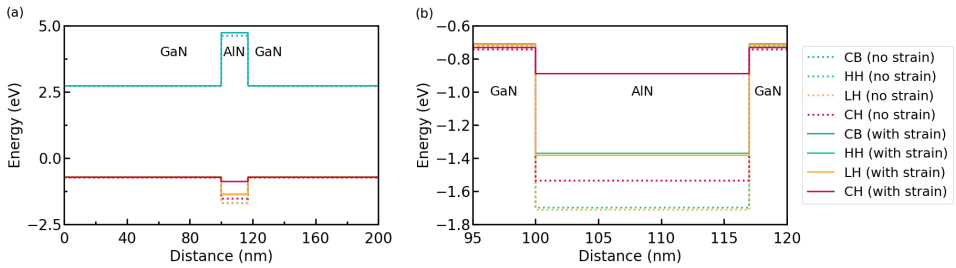


Figure 6.4.2.47: Calculated conduction and valence band structures with strain effects. (a) Full energy profile. (b) Valence band edges of AlN. (Run *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain.in* to reproduce.)

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 additive, 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 1st interface (GaN/AlN) at 100 nm:

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

2nd interface (AlN/GaN) at 117 nm:

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

The interface charge of $-0.056\ \mathrm{C/m^2}$ corresponds to $34.952 \times 10^{12}\ \mathrm{electrons/cm^2}$.

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.

$$P_{\mathrm{pz,x}}(\mathrm{AlN}) = e33\,\varepsilon_{\perp} + e31\left[\varepsilon_{\parallel} + \varepsilon_{\parallel}\right] = 1.79 \times [-0.0143283] - 0.50 \times 2 \cdot 0.0247429 = -0.050390\ \mathrm{C/m^2}$$

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

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

and

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

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, $P_{\mathrm{pz,x}}\left(\mathrm{GaN}\right)$, and AlN, $P_{\mathrm{pz,x}}\left(\mathrm{AlN}\right)$.

The 1st interface (GaN/AlN) at 100 nm:

$$-\left[P_{\mathrm{pz,x}}\left(\mathrm{AlN}\right)-P_{\mathrm{pz,x}}\left(\mathrm{GaN}\right)\right]=P_{\mathrm{pz,x}}\left(\mathrm{GaN}\right)-P_{\mathrm{pz,x}}\left(\mathrm{AlN}\right)=0+0.050390=0.050390\ \mathrm{C/m^2}$$

2nd interface (AlN/GaN) at 117 nm:

$$-\left[P_{\mathrm{pz,x}}\left(\mathrm{GaN}\right)-P_{\mathrm{pz,x}}\left(\mathrm{AlN}\right)\right]=P_{\mathrm{pz,x}}\left(\mathrm{AlN}\right)-P_{\mathrm{pz,x}}\left(\mathrm{GaN}\right)=-0.050390-0=-0.050390\ \mathrm{C/m^2}$$

The interface charge of $-0.050390\ \mathrm{C/m^2}$ corresponds to $31.451\times10^{12}\ \mathrm{electrons/cm^2}$.

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 6.4.2.48 (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)

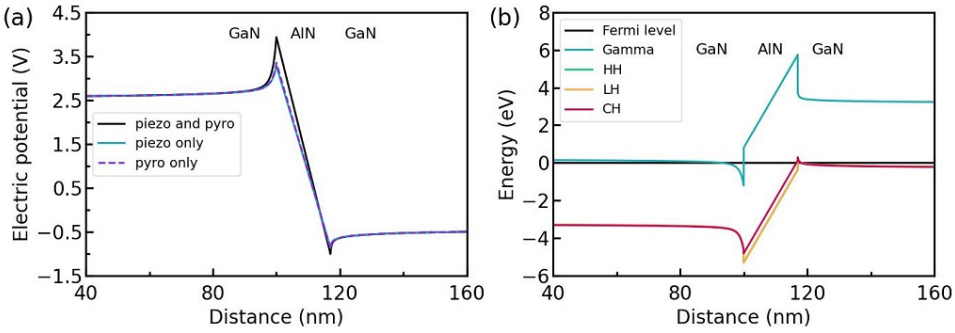


Figure 6.4.2.48: Electrostatic potential and energy profiles for Ga-face polarity. (a) The electrostatic potential with pyroelectric (py) and piezoelectric (pz) charges. (b) Conduction and valence band energy profiles under strain with all polarization charges included. (Run *piezo-pyro-charges_wz_GaN-AlN_1D_nnp_strain-pyro.in*, *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 6.4.2.48 (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 6.4.2.49 shows again the electrostatic potential and the energy profiles, as before, but for both, Ga-face and N-face polarities.

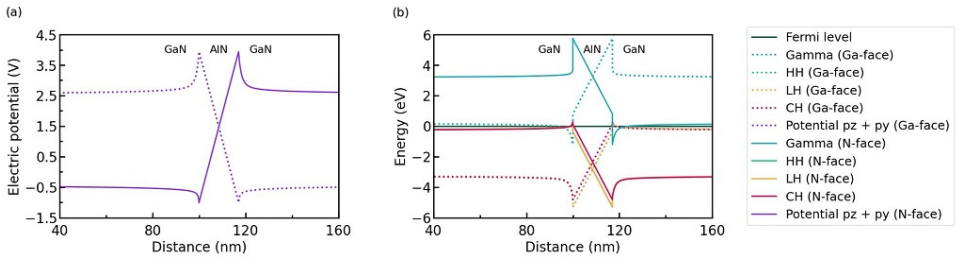


Figure 6.4.2.49: Electrostatic potential and energy profiles for Ga-face (dotted) and N-face polarities (solid). (a) The electrostatic potential with pyroelectric (py) and piezoelectric (pz) charges. (b) Conduction and valence band energy profiles under strain with all polarization charges included.

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

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:

$$v_{\mathrm{d,e}} = \mu \cdot E = \mu \cdot \frac{U}{d} = 1222.58\ \mathrm{cm^2/Vs} \cdot \frac{1\mathrm{V}}{1\ \mathrm{cm}} = 1222.58\ \mathrm{cm/s} = 12.23\ \mathrm{m/s}$$

The drift velocities of electrons and holes at each grid point (in units of $\mathrm{cm/s}$) 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:

$$t_{\mathrm{eff,e}} = \mu \cdot \frac{m_{\mathrm{e,cond}}}{e} = 1222.58\ \mathrm{cm^2/Vs} \cdot 0.258 \frac{m_0}{e} = 1.79 \cdot 10^{-13}\ s = 0.18\ \mathrm{ps}$$

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

$$l_{\mathrm{mfp}} = v_{\mathrm{d},e} \cdot t_{\mathrm{eff},e} = 0.0022\ \mathrm{nm}.$$

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

$$j = 19507\ \mathrm{A/m^2} = 1.9507\ \mathrm{A/cm^2}.$$

Taking into account the dimensions of the Si sample ($A = 1\ \mathrm{cm^2}$), this corresponds to a total current I of

$$I = 19507\ \mathrm{A/m^2} \cdot 1\ \mathrm{cm} \cdot 1\ \mu\mathrm{m} = 1.9507 \cdot 10^{-4}\ \mathrm{A} = 0.2\ \mathrm{mA}.$$

The ohmic resistance is thus given by

$$R = \frac{U}{I} = \frac{1\ V}{1.9507 \cdot 10^{-4}\ \mathrm{A}} = 5105.2\ \Omega = 5.1\ \mathrm{k\Omega}.$$

The conductivity σ is given by

$$\sigma = \frac{j}{E} = \mu_e\, n\, e = \frac{19507\ \mathrm{A/m^2}}{1\ \mathrm{V/cm}} = 195\ \Omega\mathrm{m}.$$

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, $w = 1\ \mu\mathrm{m}$.

Further Exercises

1. Repeat the calculations for InSb assuming electron mobility $\mu_{e,\mathrm{InSb}} = 4 \cdot 10^5\ \mathrm{cm}^2/\mathrm{Vs}$ 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 $\mu_{e,\mathrm{2DEG}} = 10^7\ \mathrm{cm}^2/\mathrm{Vs}$ 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 $4 \cdot 10^5$ 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 $\cdot\ m_0$) 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: nn/nn/nnnn

— 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: $5\,\mathrm{nm}$

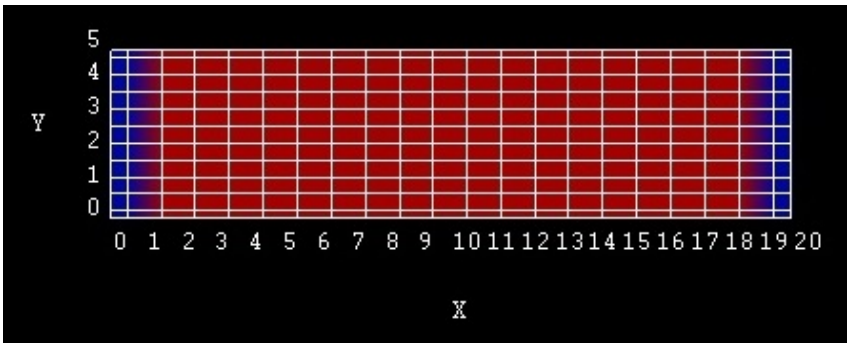


Figure 6.4.2.50: Simulated structure consisting of a left and right contact (blue) and n-doped Si layer (red).

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:

$$E_{\mathrm{c}} = k_{\mathrm{B}}T\ln\left(\frac{N_{\mathrm{c}}}{n}\right) = 0.026\,\mathrm{eV}\cdot\ln\left(\frac{2.738\cdot10^{19}\,\mathrm{cm}^{-3}}{13.478\cdot10^{18}\,\mathrm{cm}^{-3}}\right) = 0.026\,\mathrm{eV}\cdot\ln(2.031) = 18.3\,\mathrm{meV},$$
$$E_{\mathrm{v}} = -k_{\mathrm{B}}T\ln\left(\frac{N_{\mathrm{v}}}{p}\right) = -0.026\,\mathrm{eV}\cdot 42.538 = -1.099\,\mathrm{eV}.$$

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 6.4.2.51.

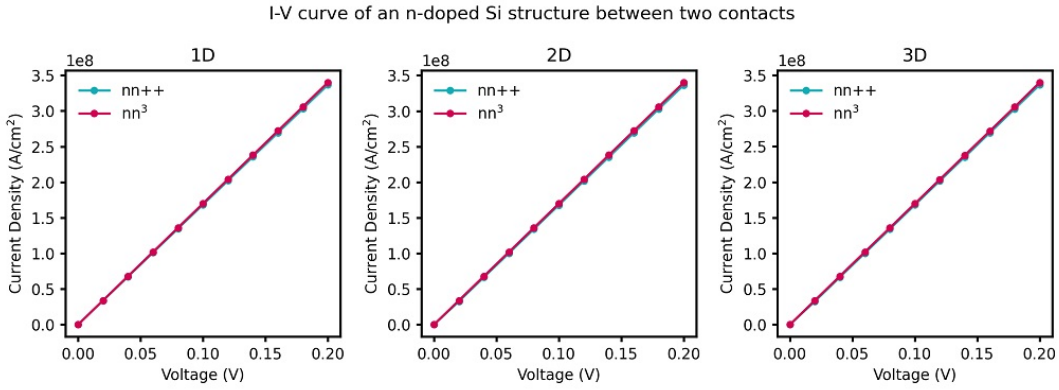


Figure 6.4.2.51: Simulated I-V characteristics of an n-doped Si structure using constant mobility model.

The *nextnano++* results are in agreement with the I-V characteristics obtained with *nextnano*[3]. 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 6.4.2.52. The width of the intrinsic region is 14 nm, and the n-doped regions are both 2 nm wide.

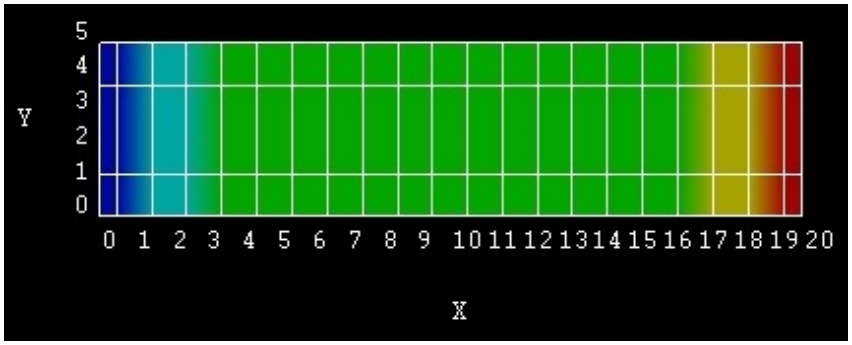


Figure 6.4.2.52: Simulated n-i-n structure consisting of a left contact (dark blue), n-doped Si (light blue), intrinsic Si (green), n-doped Si (yellow) and right contact (red).

Results

In Figure 6.4.2.53 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*.

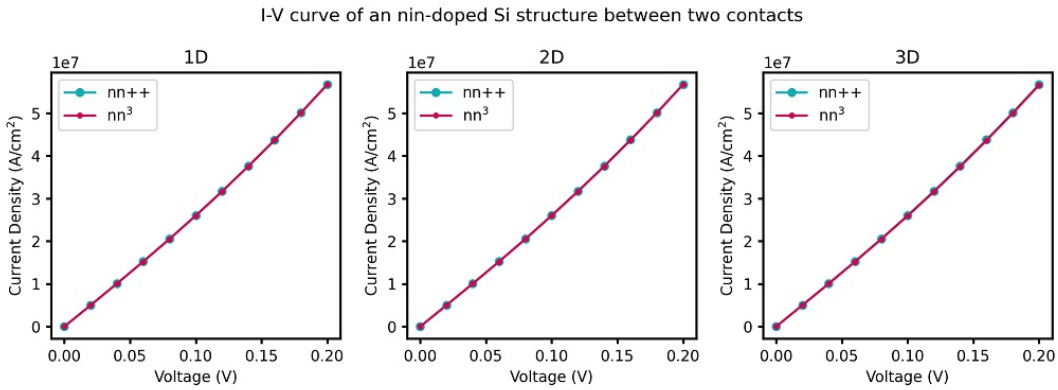


Figure 6.4.2.53: Simulated I-V characteristics of the n-i-n doped Si structure using constant mobility model.

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 $[\mathrm{A}/\mathrm{cm}^2]$. The obtained results are in perfect agreement.

Figure 6.4.2.54 shows the conduction band profile (*bias_xxxxx/bandedges.dat*) for different voltages.

Last update: 17/07/2024

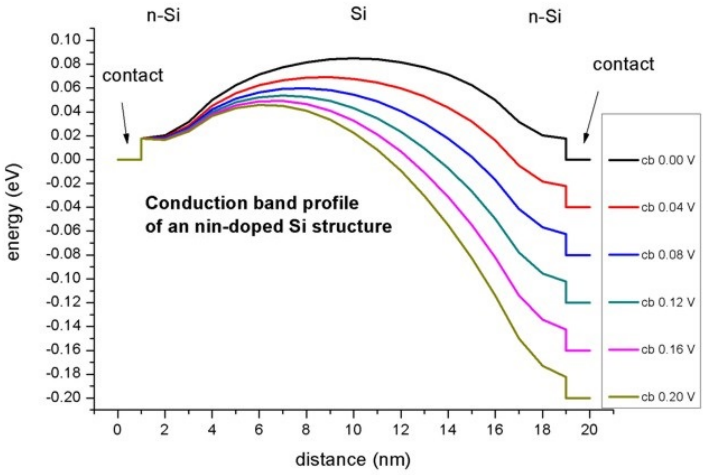


Figure 6.4.2.54: Simulated conduction band profile of the n-i-n Si structure for different voltages.

— 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	1×10^{18} cm^{-3}
contact	10	

As you see, the GaN is n-type doped with a donor concentration of $N_D = 1 \times 10^{18}$ cm^{-3}. The energy level is chosen to be 0.01507 eV below the conduction band edge.

```
impurities{
    donor{ name = "Si_donor" degeneracy = 2 energy = 0.01507 }
}
```

This leads to the electron density of 5.2846×10^{17} cm^{-3}. This is also equivalent to the concentration of the ionized donors. The result obtained by another commercial software is 5.355×10^{17} 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: $\mu_n = 100$ cm^2/Vs. The mobility model that is applied is called `constant` and described as below.

```
currents{
    mobility_model = constant
    recombination_model{
```

(continues on next page)

(continued from previous page)

```
        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 6.4.2.55 shows the IV curve obtained by *nextnano++*.

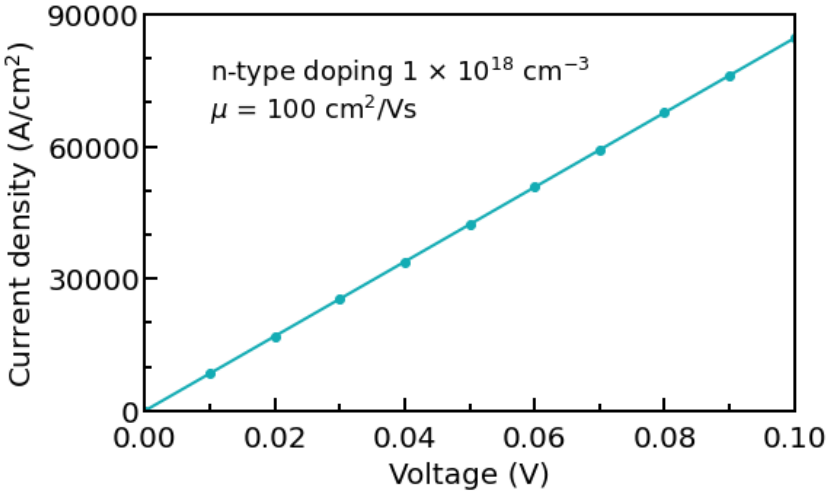


Figure 6.4.2.55: IV curve of an n-doped GaN single layer.

The figure shows that the GaN layer is an ohmic resistor. From Figure 6.4.2.55, you can obtain a resistivity of the n-GaN layer of 1.1819×10^{-6} Ωcm^2. Another commercial software results in 1.43×10^{-6} Ωcm^2.

A good check is the analytic formula given above. From this, you can obtain:

$$\sigma_n = e\mu_n n = 1.6022 \times 10^{-19} \text{ As} \times 100 \text{ cm}^2/\text{Vs} \times 5.2846 \times 10^{17} \text{ cm}^{-3} = 8.4670 \text{ A/Vcm}$$

$$\rho = d/\sigma = 100 \text{ nm}/(8.46700 \text{ A/Vcm}) = 1.1811 \times 10^{-6} \ \Omega\text{cm}^2$$

Another analytical result with the other commercial software is 1.168×10^{-6} Ωcm^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 $200\ \mathrm{nm}$. 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 $200\ \mathrm{nm}$), 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 $200\ \mathrm{nm}$.

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 $[\mathrm{A/cm^2}]$.

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 6.4.2.56). The intrinsic region and the n-doped regions are each 100 nm wide. At both ends of the device there are ohmic contacts.

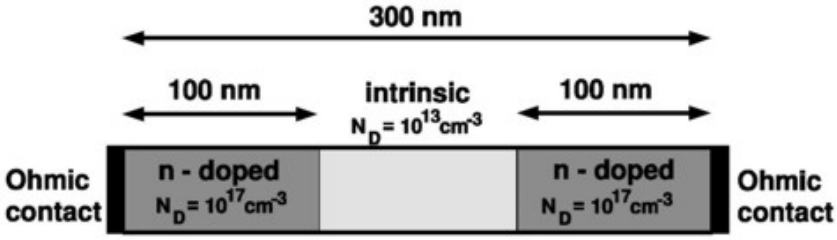


Figure 6.4.2.56: Geometry of the n-i-n Si resistor

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

$$n_{\mathrm{i}} = (N_{\mathrm{c}} N_{\mathrm{v}})^{\frac{1}{2}} \exp\left(-\frac{E_{\mathrm{gap}}}{2k_{\mathrm{B}}T}\right), \tag{6.4.2.1}$$

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 (6.4.2.1), 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 6.4.2.57 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.

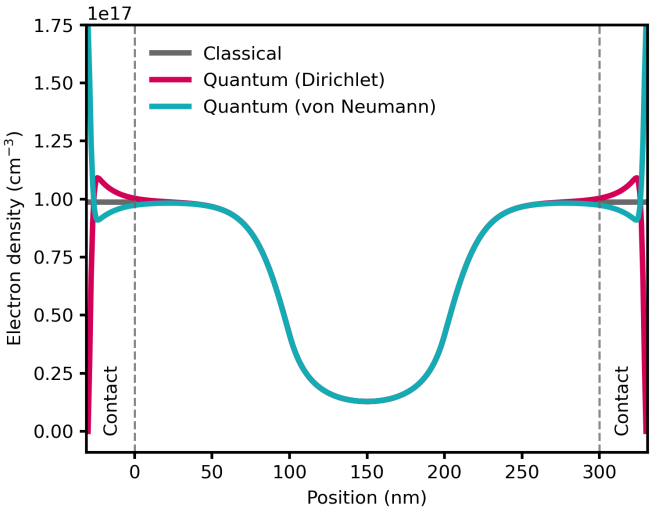


Figure 6.4.2.57: Comparison between classical and quantum mechanical electron densities for the n-i-n resistor. Quantum mechanical simulations using Dirichlet and von Neumann boundary conditions are shown.

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.

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 6.4.2.59.

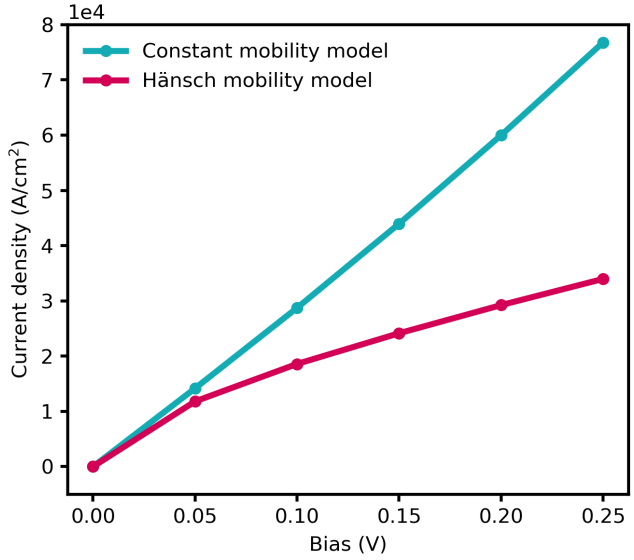


Figure 6.4.2.58: IV characteristics of the 300 nm Si n-i-n resistor for the constant mobility model and high field mobility model Hänsch (classical simulations).

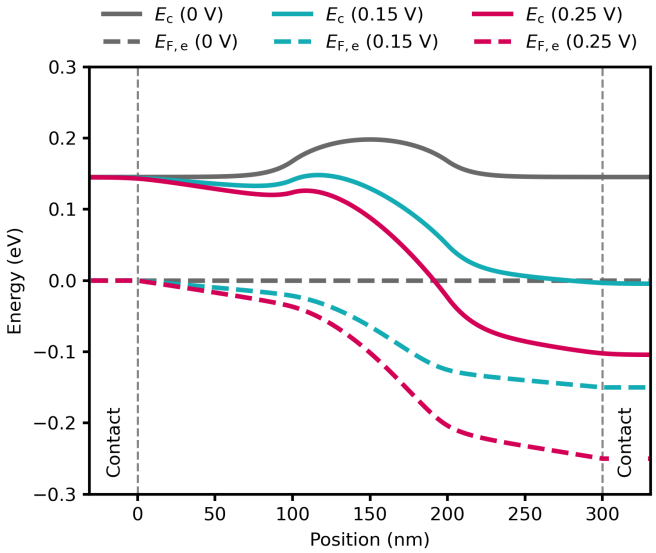


Figure 6.4.2.59: Conduction band edge profile E_c and electron quasi-Fermi levels $E_{F,e}$ at bias points of $0\,V$, $0.15\,V$ and $0.25\,V$.

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 6.4.2.60 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).

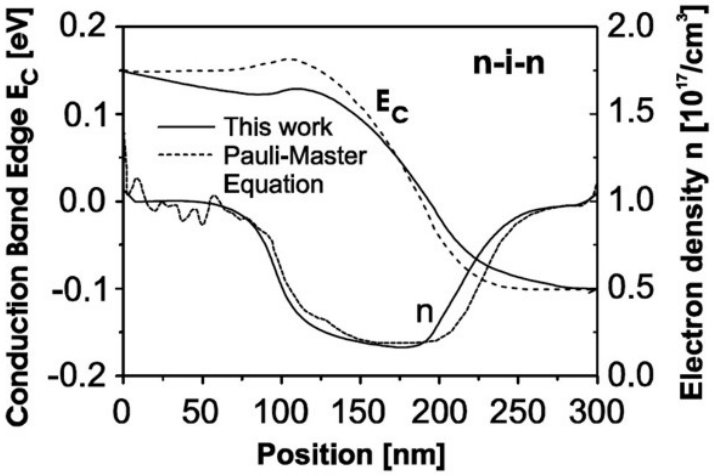


Figure 6.4.2.60: Calculated conduction band edges E_c and the electron densities n of the n-i-n structure as a function of position inside the structure. The results obtained from the Pauli master equation *[Fischetti1998]* are compared to our quantum mechanical results (full lines).

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: nn/nn/nnnn

Other

— DEV/EDU — Interpolation of 2-component alloys

- *Header*
- *Introduction*
- *How to set up simulations and why*
- *Interpolations*

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*

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 ($E_{g,InGaAs}^\Gamma$), the following formula holds.

$$E_{g,InGaAs}^\Gamma(x) = x \times E_{g,InAs}^\Gamma + [1 - x] \times E_{g,GaAs}^\Gamma$$

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
            }
        }
        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
```

(continues on next page)

(continued from previous page)

```
        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
        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,

$$P_{InGaAs}(x) = x \times P_{InAs} + [1-x] \times P_{GaAs} - x[1-x] \times b_{InGaAs}(x)$$

$$b_{InGaAs}(x) = x \times b_{In(x)Ga(1-x)As \rightarrow InAs} + [1-x] \times b_{In(x)Ga(1-x)As \rightarrow GaAs}$$

$b_{InGaAs}(x)$ is a composition-dependent bowing parameter. The $b_{In(x)Ga(1-x)As \rightarrow InAs}$ is a constant bowing parameter for nearly pure InAs ($x = 1$), while the $b_{In(x)Ga(1-x)As \rightarrow GaAs}$ is also a constant bowing parameter for nearly pure GaAs ($x = 0$).

To define $b_{In(x)Ga(1-x)As \rightarrow InAs}$, and $b_{In(x)Ga(1-x)As \rightarrow GaAs}$, 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
        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
```

(continues on next page)

(continued from previous page)

```
        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,

$$b_{In(x)Ga(1-x)As \to InAs} = b_{InGaAs}(1) = 0.359 + 0.491 \cdot (1-1) + 0.580 \cdot (1-1)^2 = 0.359$$

$$b_{In(x)Ga(1-x)As \to GaAs} = b_{InGaAs}(0) = 0.359 + 0.491 \cdot (1-0) + 0.580 \cdot (1-0)^2 = 1.43$$

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 6.4.2.61).

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} 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.

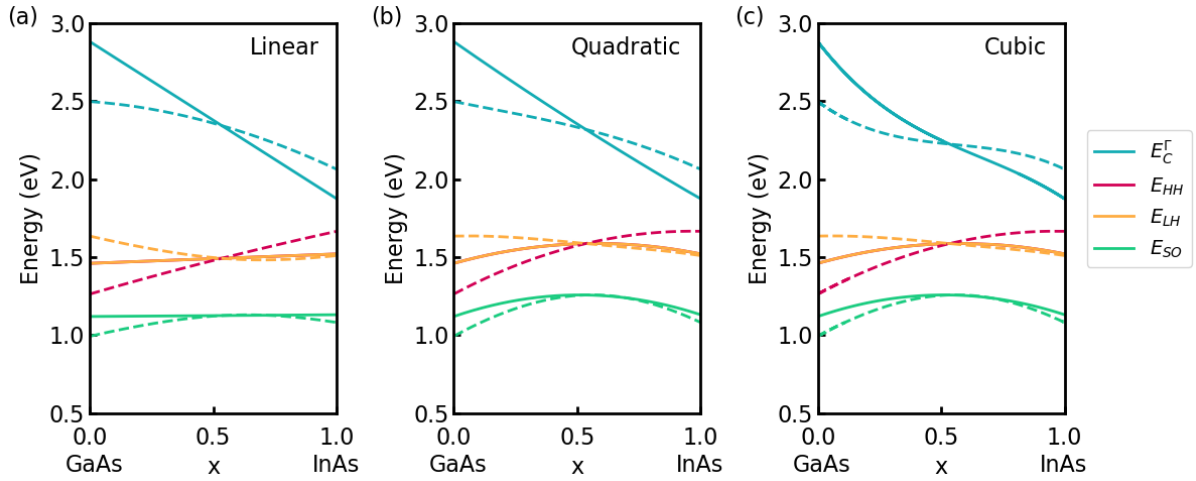


Figure 6.4.2.61: Band edges of In(x)Ga(1-x)As with a linear scheme in (a). (b) is with a quadratic scheme. (c) is with a cubic scheme. The band edges without strain are plotted with solid lines. The ones with strain are plotted with dotted lines.

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

6.4.3 p-n Junctions & Solar Cells

— 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*

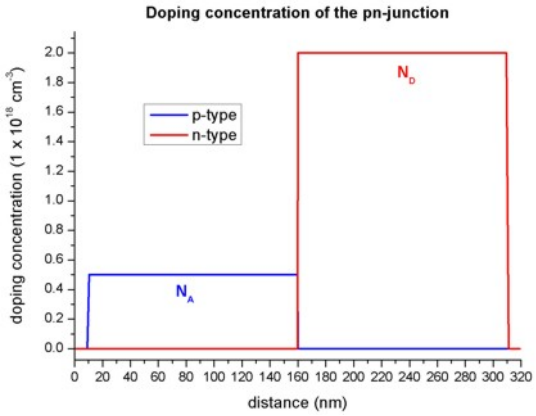
This tutorial discusses the *nextnano++* input file. Identical results can be achieved with the *nextnano*^{3} input files listed above.

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

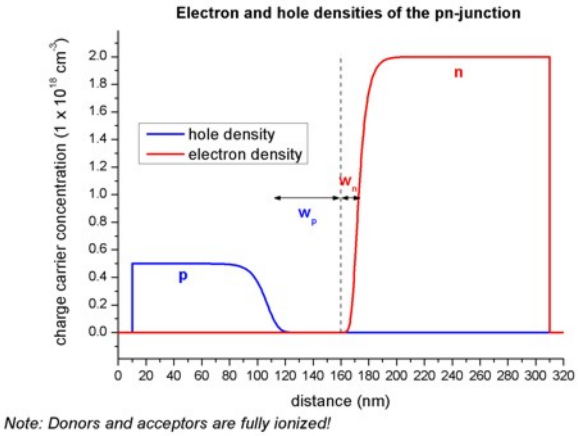
The structure consists of 300 nm GaAs. At the left and right boundaries, metal contacts are connected to the GaAs semiconductor (i.e. from 0 nm to 10 nm, and from 310 nm to 320 nm). The structure is p-type doped from 10 nm to 160 nm and n-type doped from 160 nm and 310 nm.

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

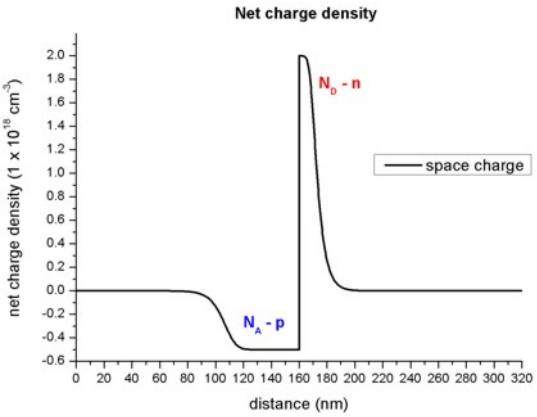
The equilibrium condition for a p-n junction is achieved by a small transfer of electrons from the n region to the p region, where they recombine with holes. This leads to a **depletion region** (depletion width = $w_p + w_n$), i.e. the region around the p-n junction only has very few free carriers left. 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!

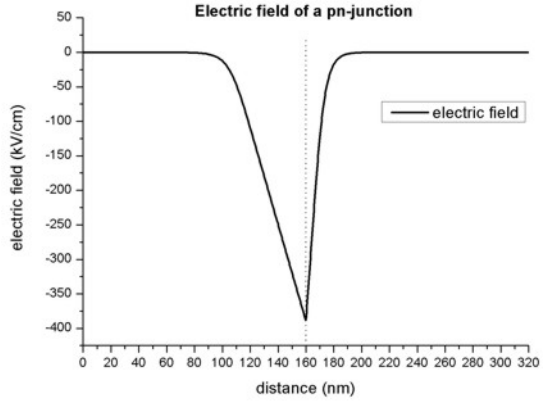
Net charges (space charge)

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:

$$F_{max} = \frac{-eN_A w_p}{\epsilon\epsilon_0} = -6.997 \times 10^{14}\text{V/m}^2 w_p = 387\text{kV/cm}$$
$$= \frac{-eN_D w_n}{\epsilon\epsilon_0} = -2.799 \times 10^{15}\text{V/m}^2 w_n = 386\text{kV/cm}$$

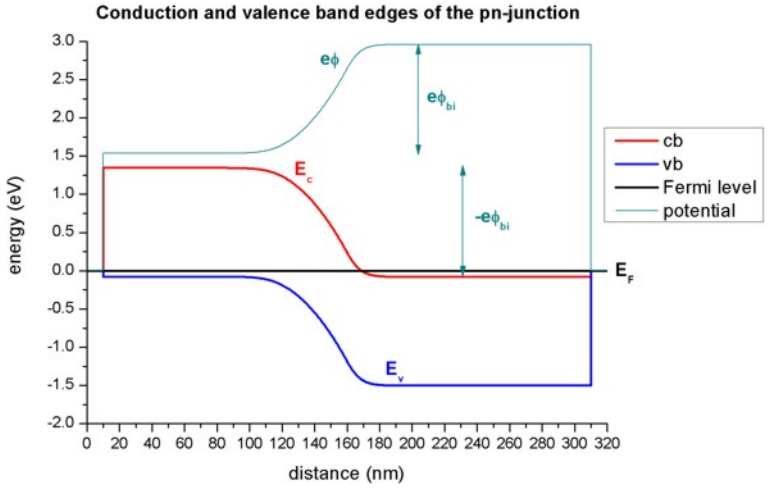
Symbol	Value
e	1.6022×10^{-19}As
ϵ	12.93 (Dielectric constant of GaAs)
ϵ_0	8.854×10^{12}As/(Vm)
N_A	$0.5 \times 10^{18}\text{cm}^{-3}$
N_D	$2.0 \times 10^{18}\text{cm}^{-3}$
w_p	55.3 nm
w_n	13.8 nm

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 $F_{\max} = 387$kV/cm, this would result in a depletion width: $w_p + w_n = 73.7$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

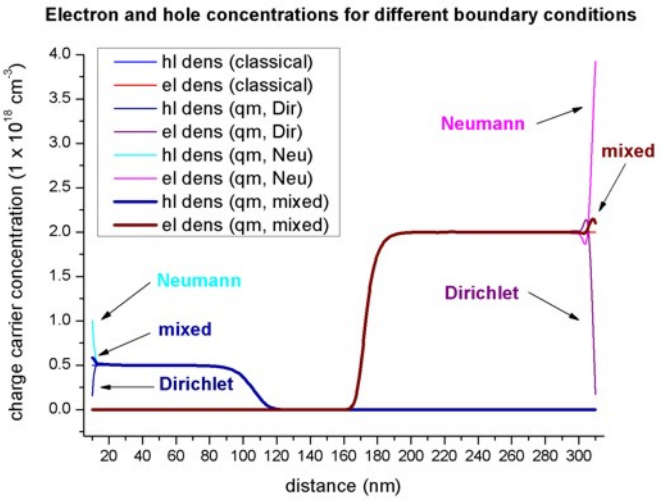
Using the *nextnano*[3] input file *pn_junction_GaAs_1D_QM_nn3.in*, we can solve the Schrödinger equation for the electrons, light and heavy holes in the single-band approximation over the whole device, rather than classically. We calculate up to 300 eigenvalues for each band. Thus the electron and hole 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.

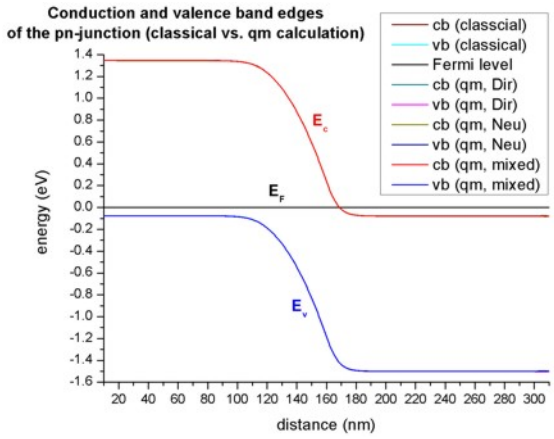
The following figure shows the band edges of the p-n junction for the four cases:

- 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.

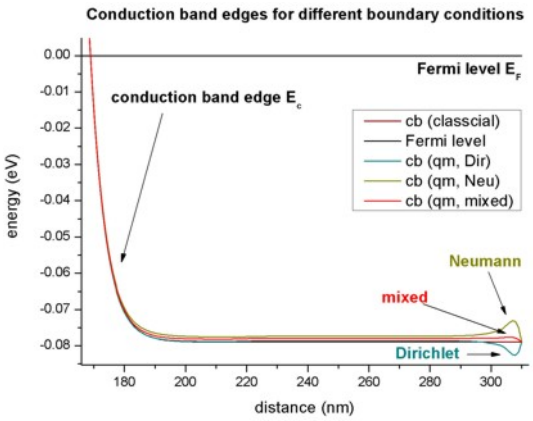


This figure is a zoom into the right boundary of the conduction band edge. On this scale, the tiny deviations for the different boundary conditions can be clearly seen.

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 ($E_{Fn} = E_{Fp} = 0$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_nn3.in / *_nnp.in* - input file for the *nextnano*[3] and *nextnano++* tools
- *pn_junction_GaAs_3D_nn3.in / *_nnp.in* - input file for the *nextnano*[3] and *nextnano++* tools

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                                                   ␣
↪!
# --------------------------                                                   ␣
↪!
#                                                                              ␣
↪!
# This is an input file for nextnano++ to calculate the band edges of a        ␣
↪!
# simple p-n junction with classical charge densities.                         ␣
↪ !
#                                                                              ␣
↪!
# It's part of the 1D p-n junction tutorial which can be found at:             ␣
↪ !
# https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_pn_junction.htm       ␣
↪ !
#                                                                              ␣
↪!
# For help on the individual keywords please go to                             ␣
↪!
# https://www.nextnano.com/nextnanoplus/software_documentation/input_file.
↪htm !
#                                                                              ␣
↪!
```

(continues on next page)

(continued from previous page)

```
# nextnano (c) nextnano GmbH
# This input file is (c) Stefan Birner, nextnano GmbH.
# This file is protected by applicable copyright laws. You may use it
# within your research or work group, but you are not allowed to give
# copies to other people without explicit permission.
#
# Documentation: https://www.nextnano.com/nextnanoplus/
# Support:       support@nextnano.com

#####################################################################################

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  }
```

(continues on next page)

(continued from previous page)

```
        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{
                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{
```

(continues on next page)

(continued from previous page)

```
        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 }    # '-
↪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{ }
}
```

Input Files for *nextnano*[3]:

- pn_junction_GaAs_1D_nn3.in
- pn_junction_GaAs_1D_QM_nn3.in
- pn_junction_GaAs_2D_nn3.in
- pn_junction_GaAs_3D_nn3.in

Last update: nn/nn/nnnn

— DEV — 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 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 = $[V_{\mathrm{start}}, V_{\mathrm{end}}]$). 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{
    mobility_model = minimos                    # mobility model
    recombination_model{                        # recombination model
        SRH       = yes
        Auger     = yes
        radiative = yes
    }
    minimum_density = $MINIMUMDENSITY           # convergence parameter
    maximum_density = 1e14                      # convergence parameter

    # output settings
    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.

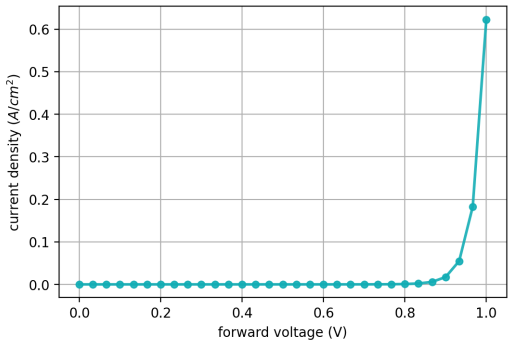


Figure 6.4.3.1: Current density as function of applied bias (1d simulation)

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 6.4.3.1, can be directly visualized in *nextnanomat*.

Input Files for *nextnano*[3]:

- *pn_junction_GaAs_1D_ForwardBias_nn3.in*

Last update: nn/nn/nnnn

— SOON/EDU — p-n junction in the dark

Attention: This tutorial is under construction

Header

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

- *p-n-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 *— SOON/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 6.4.3.2 shows 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 6.4.3.3 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 6.4.3.3 (b) and (c).

Figure 6.4.3.4 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.

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.

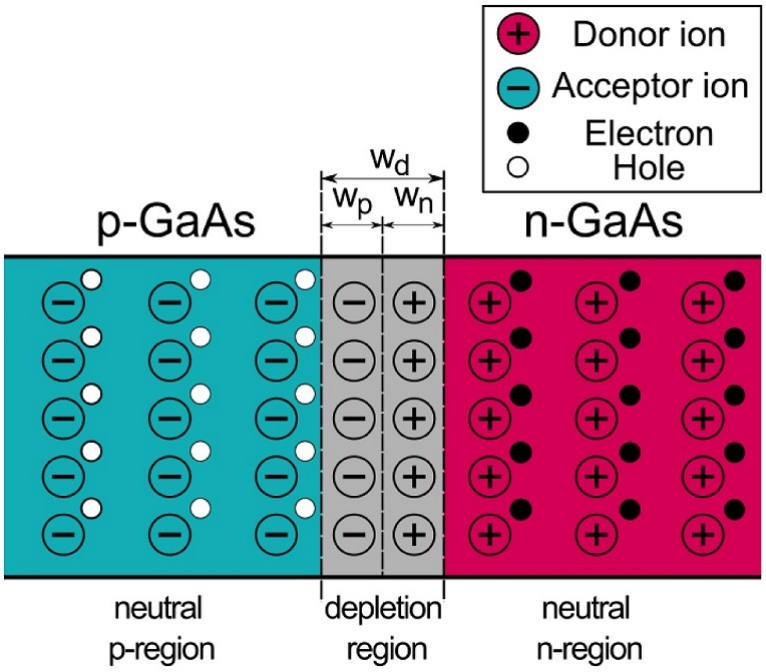


Figure 6.4.3.2: The schematic illustration of the p-n junction.

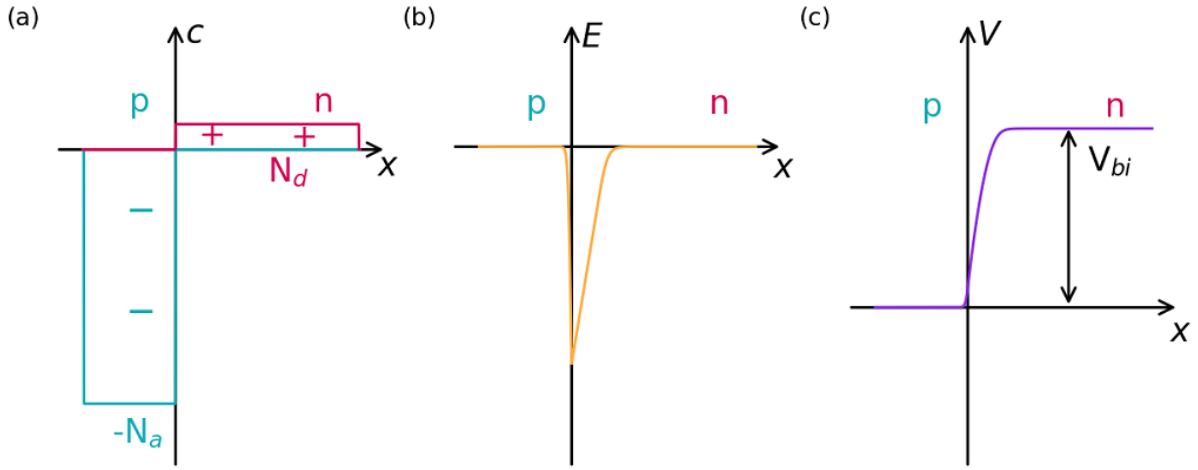


Figure 6.4.3.3: Some characteristics are shown across a p-n junction. (a) shows the dopant profile. (b) and (c) are the electric field and the potential across the space charge region, respectively.

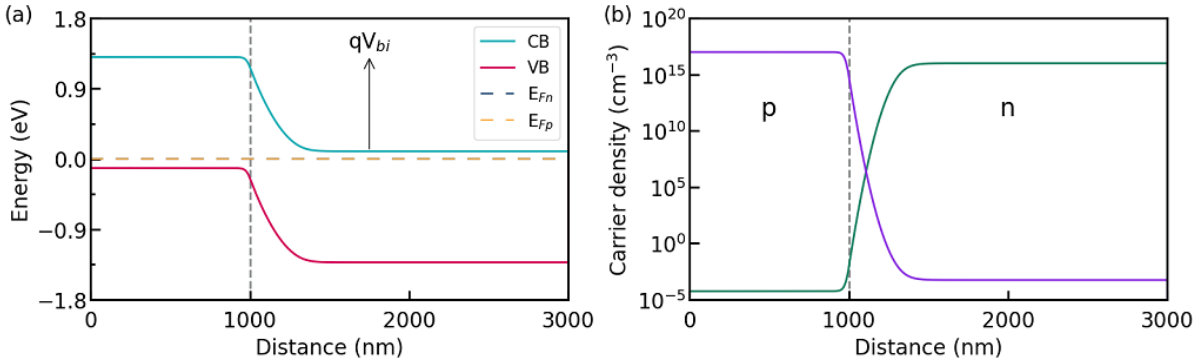


Figure 6.4.3.4: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

Since

$$w_p = \frac{1}{N_a}\sqrt{\frac{2\varepsilon_0\varepsilon V_{bi}}{q\left(\frac{1}{N_a}+\frac{1}{N_d}\right)}} \tag{6.4.3.1}$$

and

$$w_n = \frac{1}{N_d}\sqrt{\frac{2\varepsilon_0\varepsilon V_{bi}}{q\left(\frac{1}{N_a}+\frac{1}{N_d}\right)}}, \tag{6.4.3.2}$$

Thus,

$$w_{scr} = w_p + w_n = \sqrt{\frac{2\varepsilon_0\varepsilon}{q}\left(\frac{1}{N_a}+\frac{1}{N_d}\right)V_{bi}} \tag{6.4.3.3}$$

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:

$$w_{scr} = \sqrt{\frac{2 \cdot 8.854 \times 10^{-14}\ \text{C/(Vcm)} \cdot 12.93 \cdot 1.207\ \text{V}}{1.6022 \times 10^{-19}\ \text{C}} \frac{1.0 \times 10^{17}\ \text{cm}^{-3} + 1.0 \times 10^{16}\ \text{cm}^{-3}}{1.0 \times 10^{17}\ \text{cm}^{-3} \cdot 1.0 \times 10^{16}\ \text{cm}^{-3}}} = 4.356 \times 10^{-5}\ \text{cm} = 435.6\ \text{nm}$$

From the equation (6.4.3.3), 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 6.4.3.5 shows animation of (a) the band profiles, (b) the electric field, and (c) the space charge, respect to the applied bias.

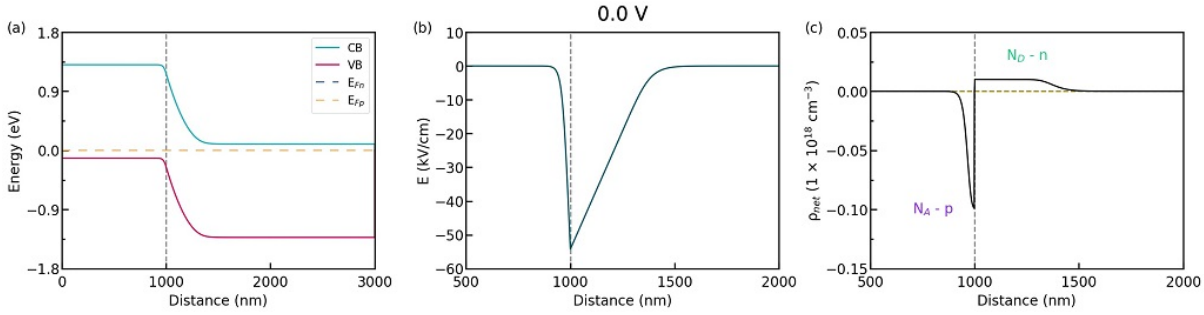


Figure 6.4.3.5: Some characters, (a) the band profiles, (b) the electric field, and (c) the space charge, respect to the applied bias.

As you see in Figure 6.4.3.5, the width w_{scr} decreases as the forward bias is applied. The width w_{scr} under the forward bias can be represented as follows.

$$w_{scr} = w_p + w_n = \sqrt{\frac{2\varepsilon_0\varepsilon}{q}\left(\frac{1}{N_a}+\frac{1}{N_d}\right)(V_{bi}-V)} \tag{6.4.3.4}$$

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.

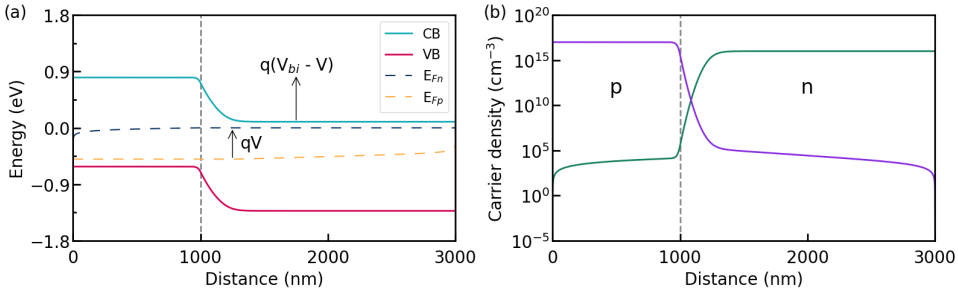


Figure 6.4.3.6: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

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,

$$qV = E_{Fn} - E_{Fp} \tag{6.4.3.5}$$

This relation can be seen from Figure 6.4.3.6 (a).

J-V curve

In this section, we sweep forward bias to acquire J-V curve. You can refer to *— DEV — I–V characteristic of GaAs p–n junction | 1D/2D/3D* to understand how to apply bias in *nextnano++*.

Figure 6.4.3.7 shows the J-V curve of the diode. *IV_characteristics.dat* is used to produce this figure.

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:

$$J_{scr}(V) = J_{scr,0}(\exp(qV/2k_BT) - 1), \tag{6.4.3.6}$$

where

$$J_{scr,0} = \frac{qn_i(w_p + w_n)}{\sqrt{\tau_n \tau_p}} \tag{6.4.3.7}$$

J_{diff} is called the diffusion current density and expressed in the following equation:

$$J_{diff}(V) = J_{diff,0}(\exp(qV/k_BT) - 1), \tag{6.4.3.8}$$

where

$$J_{diff,0} = q{n_i}^2 \left(\frac{D_n}{N_a L_n} + \frac{D_p}{N_d L_p} \right) \tag{6.4.3.9}$$

The parameters used in the expressions above are in the table.

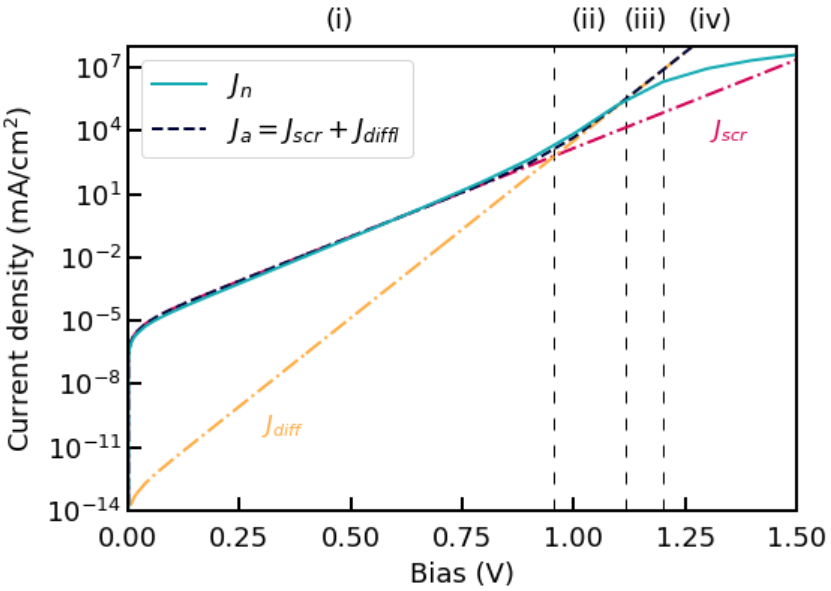


Figure 6.4.3.7: J-V curve of the diode. (i) space charge recombination current region, (ii) diffusion current region, (iii) high-injection region, (iv) series-resistence effect region.

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 6.4.3.7 (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 (6.4.3.6), 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 (6.4.3.8), 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 6.4.3.8 (b), where 1.2 V is applied to the diode.

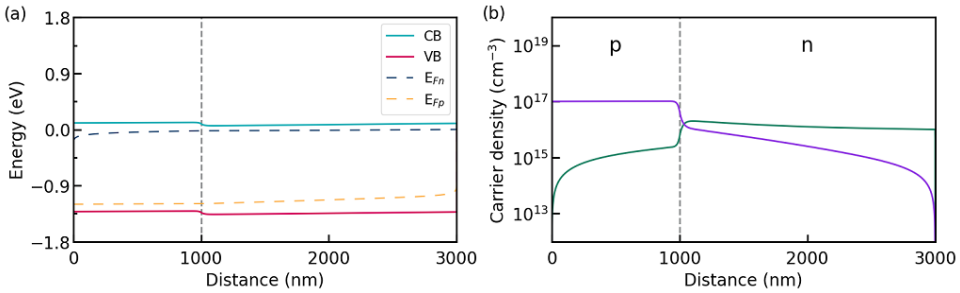


Figure 6.4.3.8: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

Then, the electron density must increase to maintain the neutrality. As a result, $n \approx p$ holds.

Because of the law of the junction,

$$np = n_i^2 \exp(qV/k_BT), \tag{6.4.3.10}$$

we acquire the equation as follows.

$$n = p = n_i \exp(qV/2k_BT) \tag{6.4.3.11}$$

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^*).

$$J_{diff,region(iv)} \approx J_{diff,0} \frac{qV^*}{k_BT}, \tag{6.4.3.12}$$

where

$$V^* = V - IR \tag{6.4.3.13}$$

Numerical control

Since we solve the current equation and the poisson equation (explanation: *General scheme of the optical device analysis*) self-consistently, we need some techniques to make the calculations more stable.

In this section, we introduce the effects of `currents{minimum_density}` and `currents{maximum_density}`.

You should also check *minimum_density*, *maximum_density*, and *Convergence* to understand the roles of the syntaxes.

In Figure 6.4.3.7, 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 = 1.0
    minimum_density_factor = [$min_density, $min_density]   # a minimum density for electrons and holes, respectively
    maximum_density = 1.0
    maximum_density_factor = [$max_density, $max_density]   # a maximum density for electrons and holes, respectively
}
```

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 `$max_density = 1.0E+20`. Similarly, you can set `$min_density`. Since the minimum electron and hole densities are about 1.0×10^{0} (cm^{-3}), `$min_density = 1.0E-2` is 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), `$min_density = 1.0E-2` is enough low as well. However, you have to take care of the magnitude of `$max_density`.

Figure 6.4.3.9 (a) shows the effect of the magnitude of `$max_density` on the current density under 0.5 V in the scheme (B).

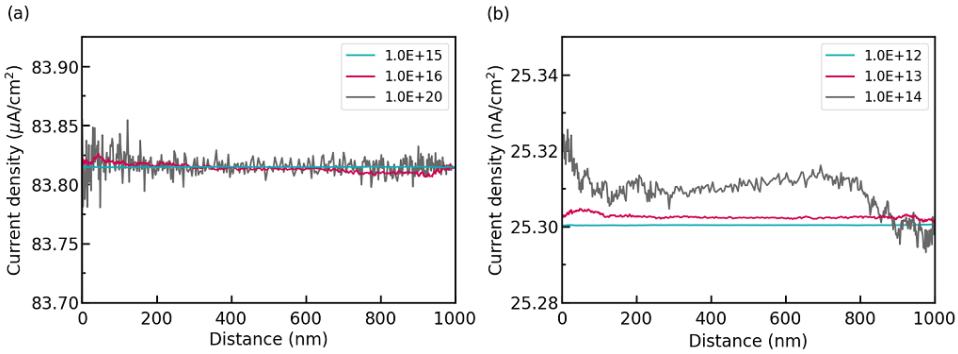


Figure 6.4.3.9: The effect of the magnitude of `$max_density` on the current density. (a) is under 0.5 V in the scheme (B). (b) is under 0.1 V in the scheme (A).

Although the current density has to be constant through the diode, it becomes unstable at `$max_density = 1.0E+16` and `$max_density = 1.0E+20`. Thus, you should set `$max_density` to `1.0E+15`, which shows the constant current density.

In the scheme (A), the same techniques should be applied. Figure 6.4.3.9 (b) shows the effect of the magnitude of `$max_density` on the current density under 0.1 V in the scheme (A). As you can see, `max_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

— SOON/EDU — p-n junction under illumination

Attention: This tutorial is under construction

Header

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

- *p-n-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 *— SOON/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 6.4.3.10 (a). The electric field impedes the diffusion of majority carriers as explained in *— SOON/EDU — p-n junction in the dark*.

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 6.4.3.10 (b)). As long as the junction is illuminated, the electron-hole pair is generated and constitutes the current (Figure 6.4.3.10 (c)).

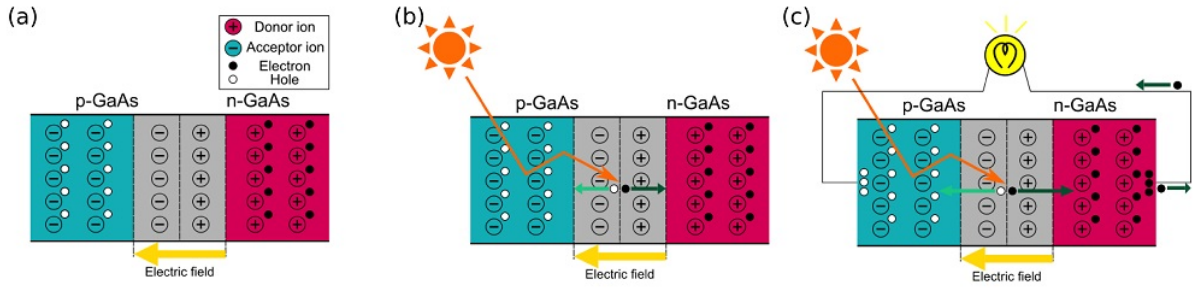


Figure 6.4.3.10: The schematic images showing the principles of a solar cell. (a) is the p-n junction at equilibrium. When it is illuminated, an electron-hole pair is generated at the junction (b). The current runs as long as the diode is illuminated (c). We assume the resistance of the light bulb is zero because of the short circuit.

Figure 6.4.3.11 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.

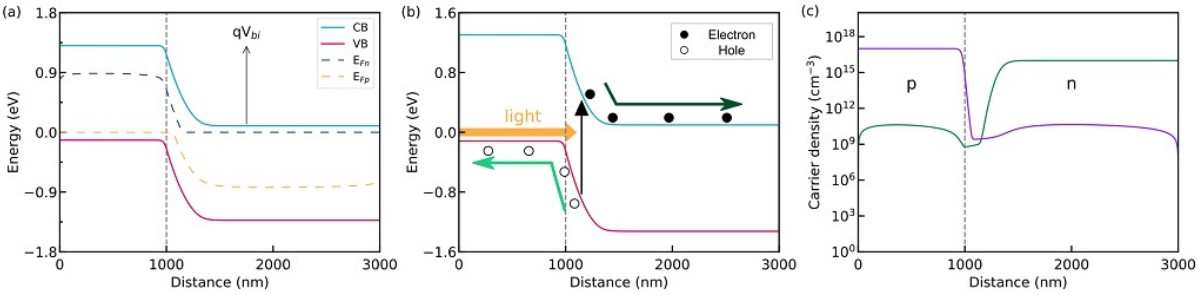


Figure 6.4.3.11: The band profiles are plotted in (a). When the light reaches the junction and the energy is bigger than the band gap, the electron-hole pair is generated as shown in (b). The carrier densities are plotted in (c). The hole density is shown in violet, whereas the electron density is in 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 6.4.3.11 (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 Photovoltaic 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 6.4.3.12.

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.

$$J(V) = J_{dark}(V) - J_{ph}(V) = J_{dark}(V) - J_{sc}, \tag{6.4.3.14}$$

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.

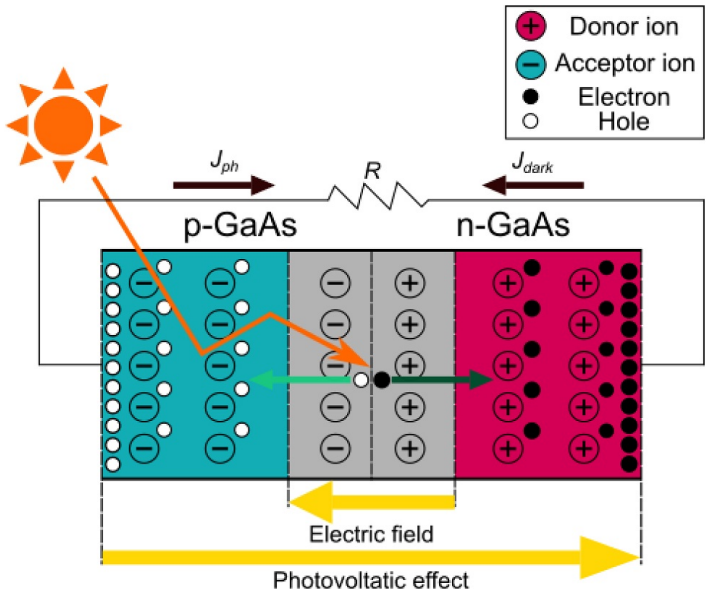


Figure 6.4.3.12: The circuit is connected to a resistive load. Note that J_{dark} flows in the opposite direction to J_{ph}.

$J_{dark}(V)$ can be expressed by the equation described in *— SOON/EDU — p-n junction in the dark*.

$$J_{dark}(V) = J_{m,0}(\exp(qV/mk_BT) - 1) \tag{6.4.3.15}$$

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,

$$J(V) = J_{m,0}(\exp(qV/mk_BT) - 1) - J_{sc} \tag{6.4.3.16}$$

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 6.4.3.13 shows the band profiles and the carrier densities of the diode under $0.5\ (V)$.

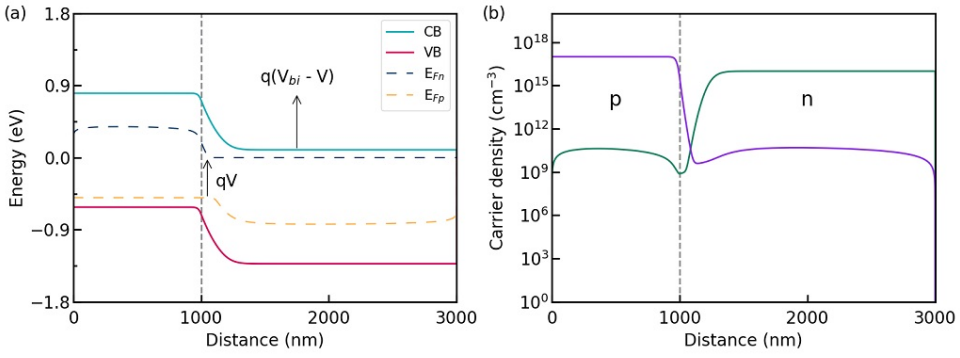


Figure 6.4.3.13: The band profiles are plotted in (a). The carrier densities are plotted in (b). The hole density is shown in violet, whereas the electron density is in green.

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 6.4.3.13 (a).

Open circuit

When the circuit is open (**open circuit**), the photovoltage V across the diode is called the open-circuit voltage V_{oc}.

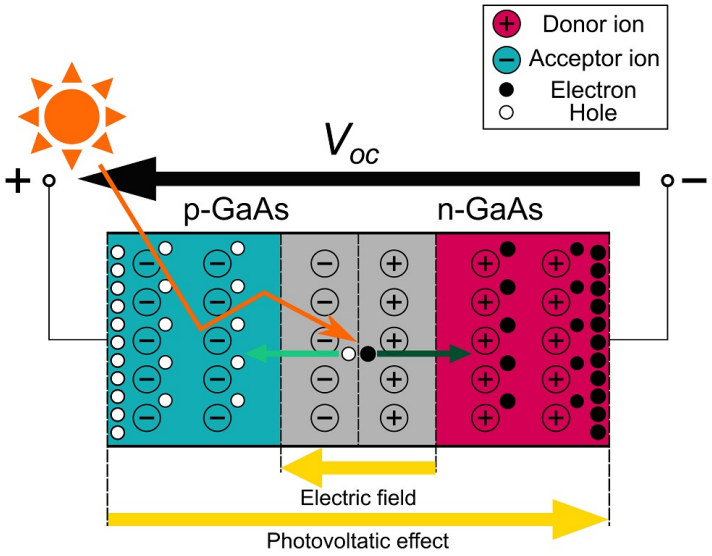


Figure 6.4.3.14: The circuit is open and the voltage V_{oc} is applied to the diode.

Since $J(V) = 0$ in (6.4.3.16) in this case,

$$V_{oc} = \frac{mk_BT}{q} \ln\left(\frac{J_{sc}}{J_0} + 1\right) \tag{6.4.3.17}$$

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 6.4.3.15 shows J-V curves of the solar cell under the illumination and under the dark condition.

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

$$P = JV = J_0V(\exp(qV/mk_BT) - 1) - J_{sc}V \tag{6.4.3.18}$$

The condition for the maximum power density is achieved when $dP/dV = 0$.

Thus,

$$V_m = \frac{1}{\beta} \ln\left[(J_{sc}/J_0) + 1\right] - \frac{1}{\beta} \ln\left(1 + \beta V_m\right) = V_{oc} - \frac{1}{\beta} \ln\left(1 + \beta V_m\right), \tag{6.4.3.19}$$

$$J_m = J_0 \beta V_m \exp(\beta V_m), \tag{6.4.3.20}$$

where $\beta = q/k_BT$.

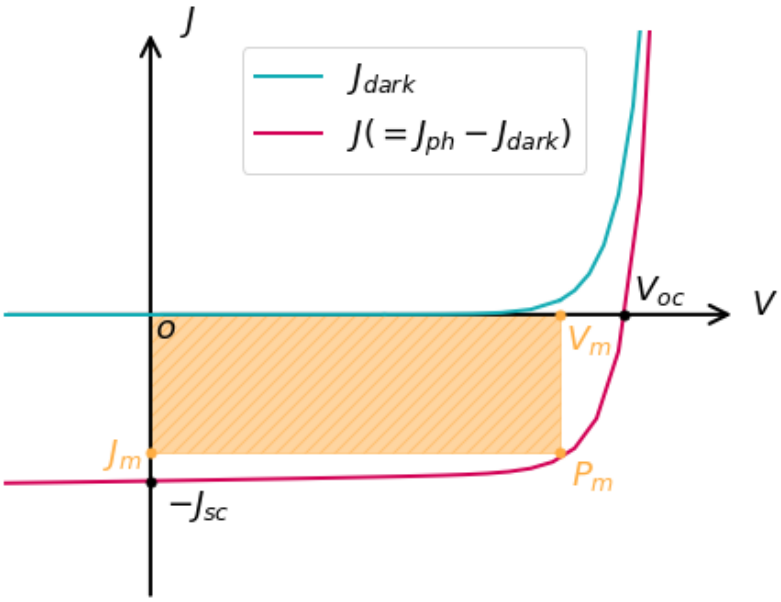


Figure 6.4.3.15: The J-V curves of the solar cell. The J-V curve under the illumination is shown in violet, whereas the J-V curve under the dark condition is in light-blue. The orange-filled area indicates the output of the maximum power density of the solar cell.

Therefore,

$$P_m = J_m V_m = J_0 \beta V_m^2 \exp(\beta V_m) = FF J_{sc} V_{oc}, \tag{6.4.3.21}$$

where FF is called the fill factor and the ratio to measure the sharpness of the J-V curve.

$$FF = \frac{J_m V_m}{J_{sc} V_{oc}} \tag{6.4.3.22}$$

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}.

$$\eta = \frac{P_m}{P_{in}} = \frac{I_m V_m}{P_{in}} = \frac{J_{sc} V_{oc} FF}{P_{in}} \tag{6.4.3.23}$$

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.

Effect of irradiation intensity

Figure 6.4.3.16 (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 (6.4.3.17), 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.

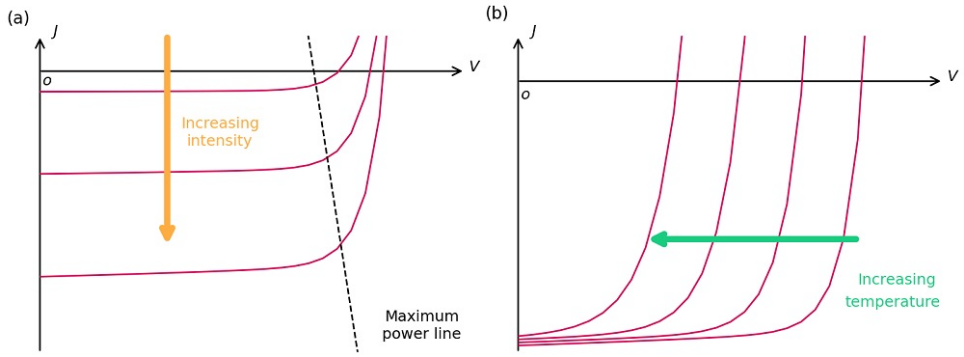


Figure 6.4.3.16: The J-V curves of the solar cell under incident light of various intensities is shown in (a). The J-V curves of the solar cell under different temperatures are shown in (b). The arrows indicate the direction of increasing intensity of the incident sunlight or the temperature.

Effect of temperature

The effect of the temperature on the J-V curve is shown in (b) in Figure 6.4.3.16. 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 (6.4.3.17), 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 *— SOON/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

GaAs solar cell

- *Header*
- *Input files*
- *Reference*
- *Structure*
- *Simulation procedure*
- *How does a solar cell work? & How do we simulate it?*
  - *1. Solar spectrum*
  - *1. Generation rate (internal calculation)*
  - *1. Generation rate (import)*
  - *4. Current-Voltage characteristics*
  - *5. Solar efficiency*

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 GmbH. 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.

In *1DGaAs_SolarCell_nn3.in* (*nextnano*[3]), you need to prepare the above three data as external files to calculate generation rate $G(E, x)$ internally. We have the external data files, whose data are identical with those used in *1DGaAs_SolarCell_nnp.in*, for *nextnano*[3].

You can also import the data of generation rate itself. In the simulation of *1DGaAs_SolarCell_nnp_import_generation.in* and *1DGaAs_SolarCell_nn3_import_generation.in*, the following output file of *nextnano*[3] 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 6.4.3.17 shows the band edges and quasi Fermi levels of the device. The device structure is as follows:

- 0-30 nm $Al_{0.8}Ga_{0.2}As$ 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.

The left side of the device (x=0 nm) is illuminated by the sun. As shown in Figure 6.4.3.22, 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.

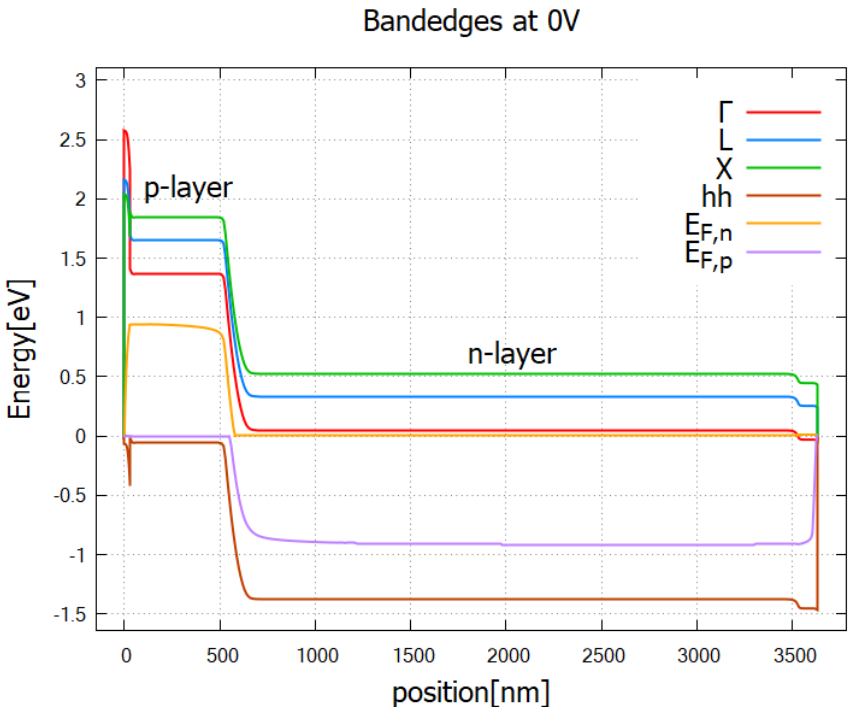


Figure 6.4.3.17: Band edges and quasi-Fermi levels of the solar cell at zero bias *bias_000000/bandedges.dat* (*nextnano++*) / *band_structure/BandEdges.dat* (*nextnano*[3])

Simulation procedure

The workflow of the simulation is summarized in Figure 6.4.3.18. 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 either import it into *nextnano++* or in *nextnano*[3].

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]:

$$\frac{2\pi \sin^2 \theta_{\text{sun}}}{h^3 c^2} \frac{E^2}{e^{E/k_B T_{\text{sun}}} - 1},$$

where E is the photon energy and $\theta_{\text{sun}} = 1.44 \times 10^{-3}\pi$[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

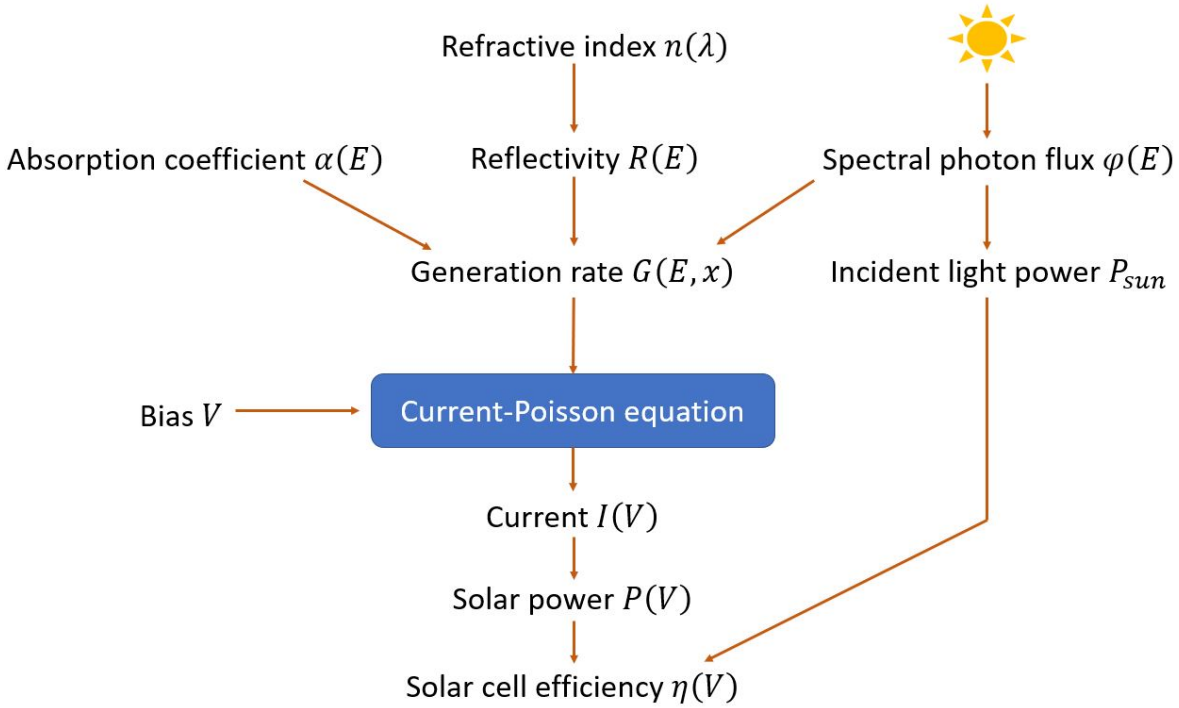


Figure 6.4.3.18: Workflow of solar cell simulation. Each quantity is explained in the following section.

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 $\phi(E)$ $[\mathrm{m}^{-2}\mathrm{s}^{-1}\mathrm{eV}^{-1}]$. 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 $[\mathrm{Wm}^{-2}\mathrm{eV}^{-1}]$. We have taken the AM1.5G spectral irradiance data from this website (Figure 6.4.3.19). If you have space applications in mind, please use the extraterrestrial spectrum, namely air mass zero (AM0).

The power of incident light

$$P_{\mathrm{sun}} = \int_0^\infty L(E)\mathrm{d}E = 1000\ \mathrm{Wm}^{-2},$$

is solely determined by the condition of the sun and the atmosphere of the earth (for AM0 $P_{\mathrm{sun}} = 1353\ \mathrm{Wm}^{-2}$). The ultimate challenge of solar cell research is to achieve the most efficient conversion of this energy input into electric power P_{out} $[\mathrm{Wm}^{-2}]$. The figure of merit is therefore defined as $\eta = \frac{P_{\mathrm{out}}}{P_{\mathrm{sun}}}$.

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 6.4.3.18):

$$G(E,x) = N\phi(E)\cdot(1-R(E))\cdot\alpha(E)e^{-\alpha(E)x},$$

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` (*nextnano++*) / `number-of-suns` (*nextnano*[3]).

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.

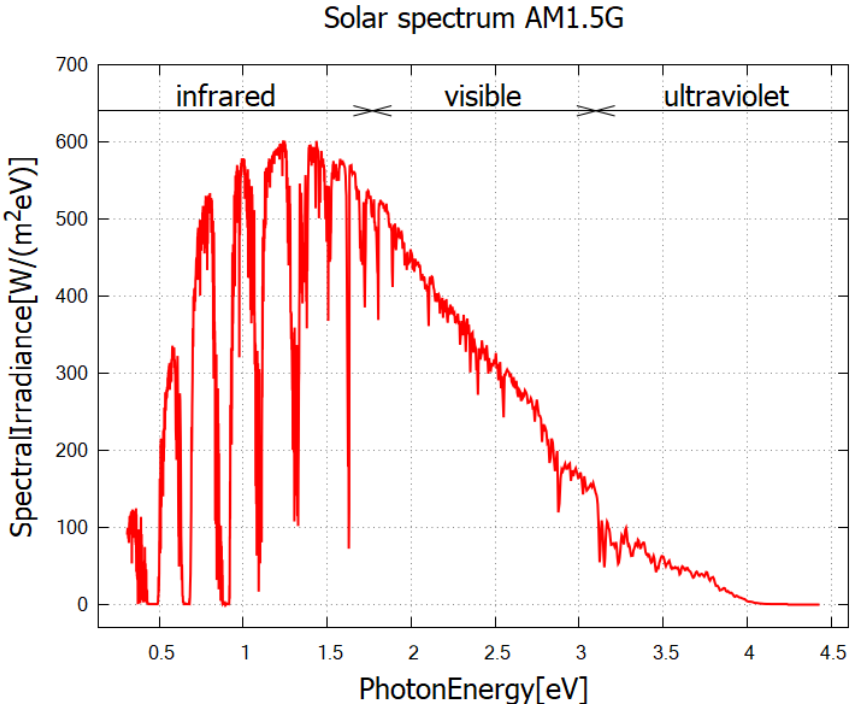


Figure 6.4.3.19: The AM1.5G spectral irradiance $L(E)$, that is, the solar spectrum measured on the earth. The *nextnano++* tool reads from the predifined data `Solar-ASTM-G173-global` and stores it in the output file *Irradiation/illumination_spectrum_power_eV.dat*. The *nextnano*[3] tool reads in the data file *ASTMG173_AM15G.dat* and stores it in the output file *optics/SolarSpectralIrradiance_eV.dat*.

```
# nextnano++
optics{
    ...

    global_illumination{
        direction_x = 1

        database_spectrum{
            name = "Solar-ASTM-G173-global"
            concentration = 1.0             # e.g. 1 sun
        }
    }

    global_reflectivity{
        database_spectrum{
            name = "Al0.80Ga0.20As"
        }
    }

    global_absorption_coeff{
        database_spectrum{
            name = "GaAs"
        }
    }
}
```

In the input file for *nextnano*[3], $\alpha(\lambda)$, $R(\lambda)$, $L(\lambda)$ and N are imported as specified in keyword *$optical-absorption*.

```
! nextnano3
$optical-absorption
 ...
 import-absorption-spectrum    = yes
 file-absorption-spectrum      = "(directory)\AbsorptionCoefficient_GaAs_300K.dat"

 import-reflectivity-spectrum  = yes
 file-reflectivity-spectrum    = "(directory)\Reflectivity_Al0.80Ga0.20As.dat"

 import-solar-spectrum         = yes
 file-solar-spectrum           = "(directory)\ASTMG173_AM15G.dat" ! G = global, i.e.␣
↪including diffuse light

!number-of-suns                = 0.0   ! switch off sun if generation rate is imported
 number-of-suns                = 1.0
!number-of-suns                = 100.0 ! optical concentration
$end_optical-absorption
```

If no reflectivity data is specified, perfect interface (zero reflection) is assumed. The reflectivity data of $Al_{0.8}Ga_{0.2}As$ used in this simulation (`optical_reflectivity` in *database_nnp_optional.in* for *nextnano++*, *Reflectivity_Al0.80Ga0.20As.dat* for *nextnano*[3]) 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 $Al_{0.8}Ga_{0.2}As$, 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 GmbH 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. For *nextnano*[3], they are stored in the output folder `optics` with file names *AbsorptionCoefficient.dat*, *Reflectivity.dat* and *SolarSpectralIrradiance.dat*, respectively.

The resulting generation rate is shown in Figure 6.4.3.20, Figure 6.4.3.21 and Figure 6.4.3.22.

1. Generation rate (import)

If the generation rate data $G(x) = \int G(E, x)\mathrm{d}E$ (Figure 6.4.3.22) 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* (*nextnano++*) and *1DGaAs_SolarCell_nn3_import_generation.in* (*nextnano*[3]), we import the data generated from *1DGaAs_SolarCell_nn3.in*.

```
# nextnano++
structure{
    region{
        everywhere{}
        generation{
            import{ import_from = "GenImportProfile" }
        }
    }
}

import{
    file{
```

(continues on next page)

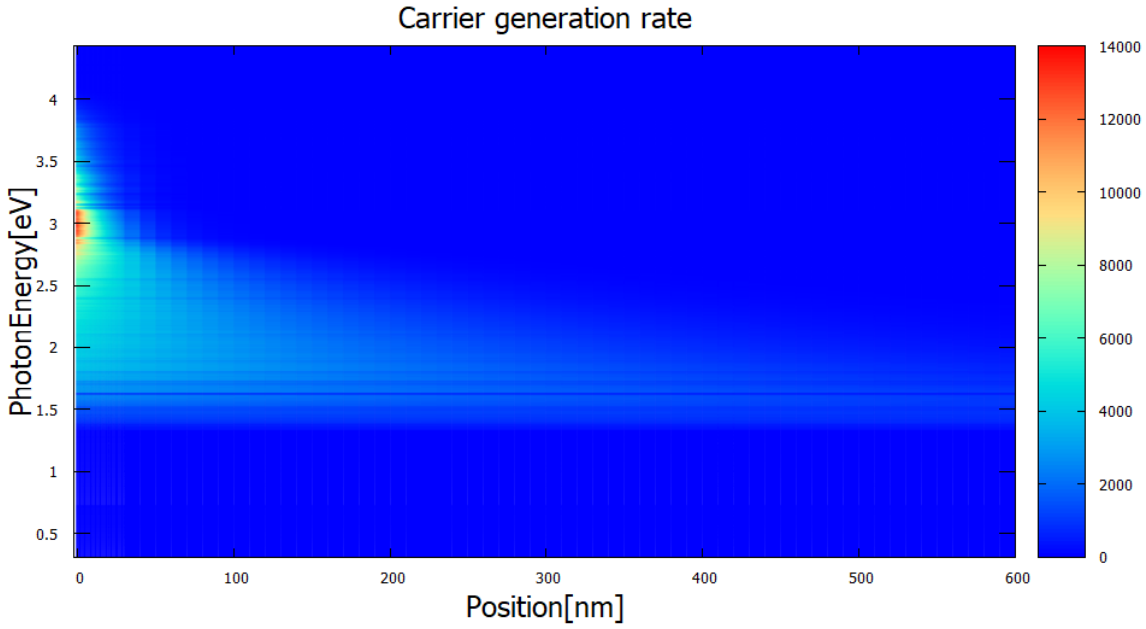


Figure 6.4.3.20: Generation rate as a function of position and energy *optics/GenerationRateLight_vs_Position_and_Energy_2Dplot_sun1.dat* (*nextnano*[3]) in units of $10^{18}\mathrm{cm}^{-3}\mathrm{eV}^{-1}\mathrm{s}^{-1}$. The corresponding file for *nextnano++* is *Irradiation/photo_generation_energy_resolved.fld*. This quantity is internally calculated using the absorption coefficient, reflectivity of the front surface and solar spectrum AM1.5G (Figure 6.4.3.19). 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.

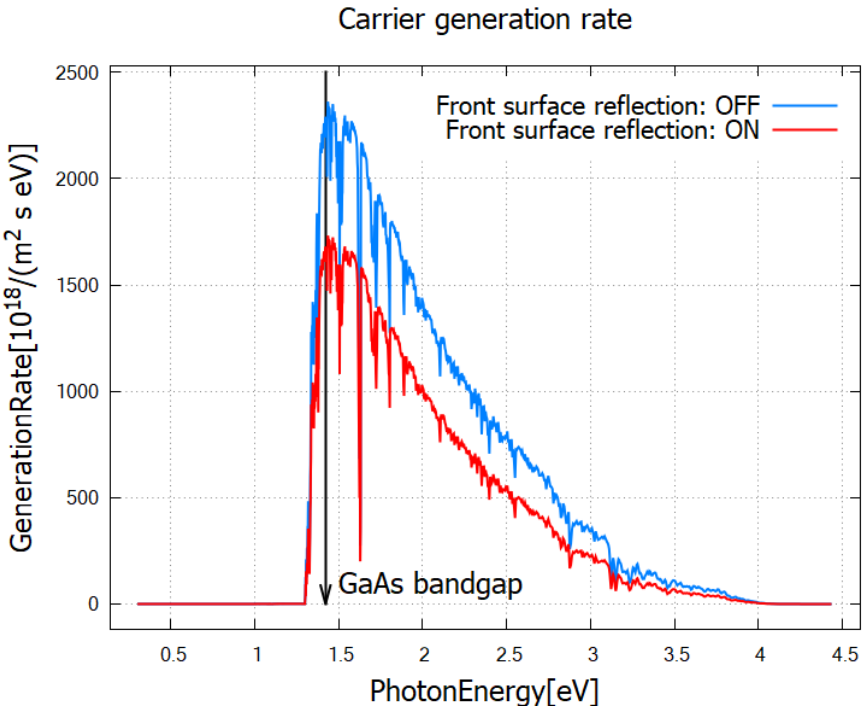


Figure 6.4.3.21: Generation rate as a function of energy *optics/GenerationRateLight_vs_Energy_sun1.dat* (*nextnano*[3]), red curve. The corresponding file for *nextnano++* is *Irradiation/photo_generation_integrated.dat*. Also shown is the result for zero reflection at the front surface (blue curve). Obviously, the generation rate becomes larger when the reflection at the front surface is neglected. One can also clearly see, by comparing with Figure 6.4.3.20, 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.

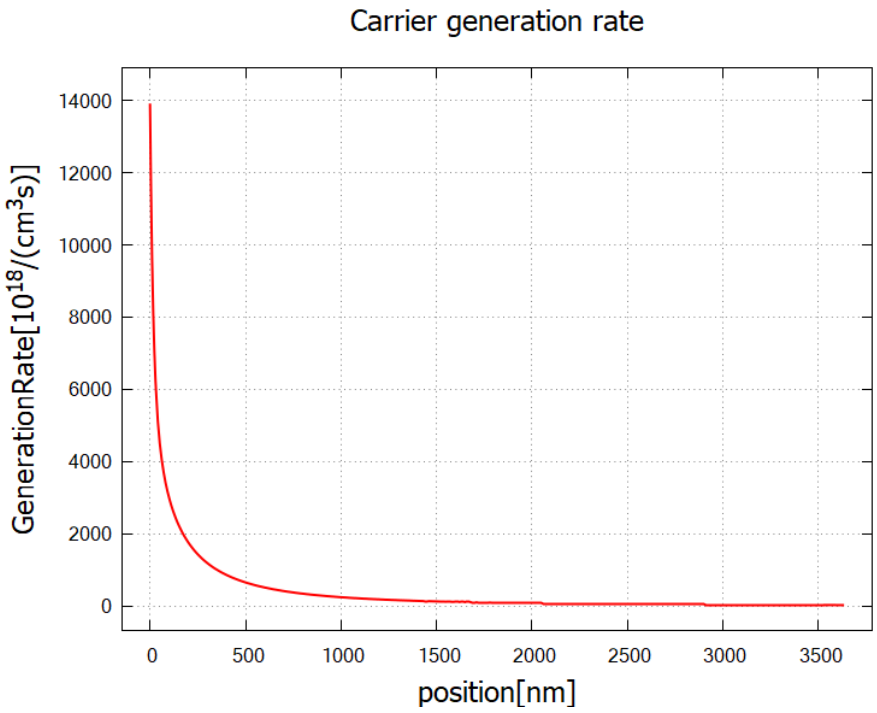


Figure 6.4.3.22: Generation rate as a function of position *optics/GenerationRateLight_vs_Position_sun1.dat* (*nextnano*^{3}). The corresponding file for (*nextnano++*) is *Irradiation/photogeneration.dat*. This data is obtained by integrating Figure 6.4.3.20 over energy E. When the photon flux travels 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).

(continued from previous page)

```
        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)
    }
}
```

```
! nextnano3
$import-data-on-material-grid
 source-directory    = "(directory path)"
 import-generation   = yes
 filename-generation = "GenerationRateLight_vs_Position_sun1.dat"
$end_import-data-on-material-grid

$optical-absorption
 ...
 number-of-suns      = 0.0 ! switch off sun if generation rate is imported
$end_optical-absorption
```

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 6.4.3.23 and Figure 6.4.3.24. For comparison, the dark current has been simulated by setting

```
# nextnano++
structure{
    region{
        generation{
            constant{ rate = 0.0 }
        }
    }
}
```

```
! nextnano3
$optical-absorption
 ...
 import-solar-spectrum   = yes
 number-of-suns          = 0.0
$end_optical-absorption
```

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{ }* when working with *nextnano++* or *$numeric-control* using *nextnano*[3].

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% (nextnano++) / 17.0% (nextnano3)** is achieved at the bias 0.9 V (Figure 6.4.3.25, Figure 6.4.3.26 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++) / 22.3% (nextnano3) for 100-sun concentration**, mainly due to the increase in open circuit voltage (Figure 6.4.3.25, Figure 6.4.3.26 blue). This means one cell operating under 100 suns can produce the same power output as $\frac{100 P_{sun} \times 0.216}{P_{sun} \times 0.158} = 137$ 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 and the file *solar_cell_info.txt* (*nextnano*[3]) contain additional properties of the solar cell.

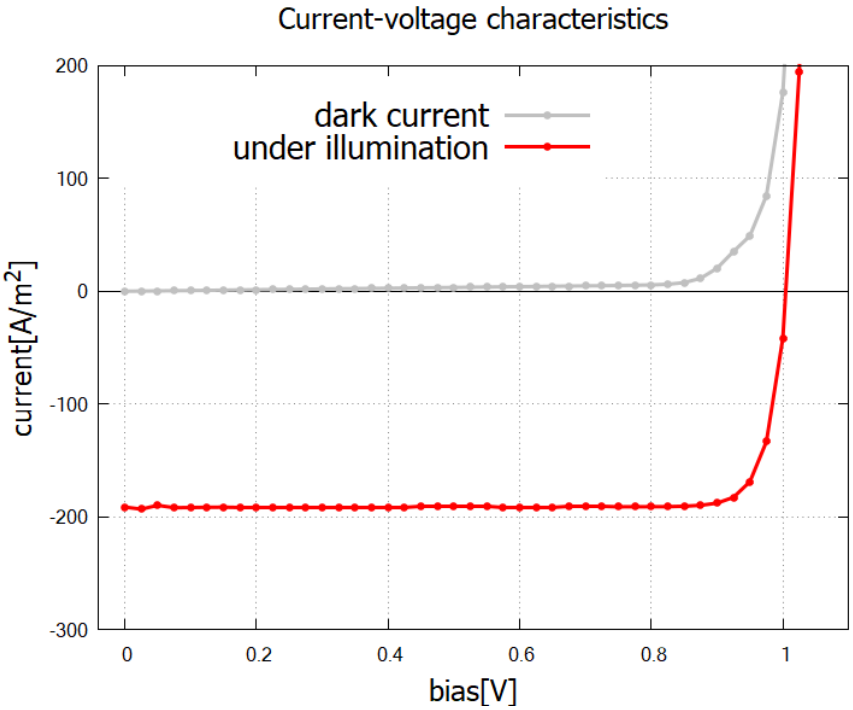


Figure 6.4.3.23: I-V characteristics of the solar cell *IV_characteristics.dat* (*nextnano++*). In the bias regime 0-1 V the system works as a solar cell.

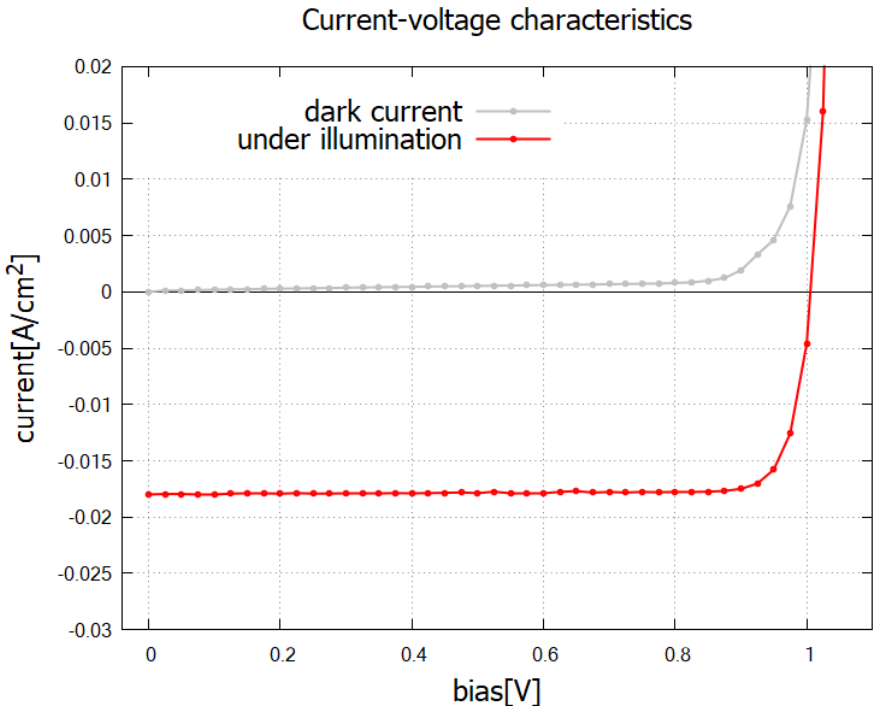


Figure 6.4.3.24: I-V characteristics of the solar cell *currentIV_characteristics_new.dat* (*nextnano*^{3}). In the bias regime 0-1 V the system works as a solar cell.

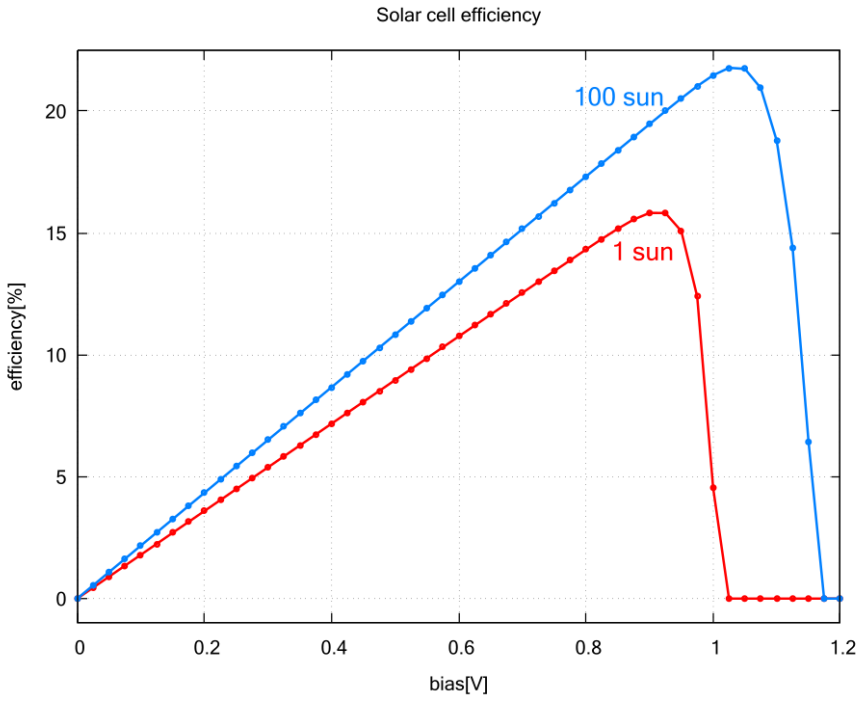


Figure 6.4.3.25: Solar cell efficiency η for no sunlight concentration (red) and 100-sun concentration (blue) by *nextnano++*. The data is contained in *solar_cell_efficiency.dat*.

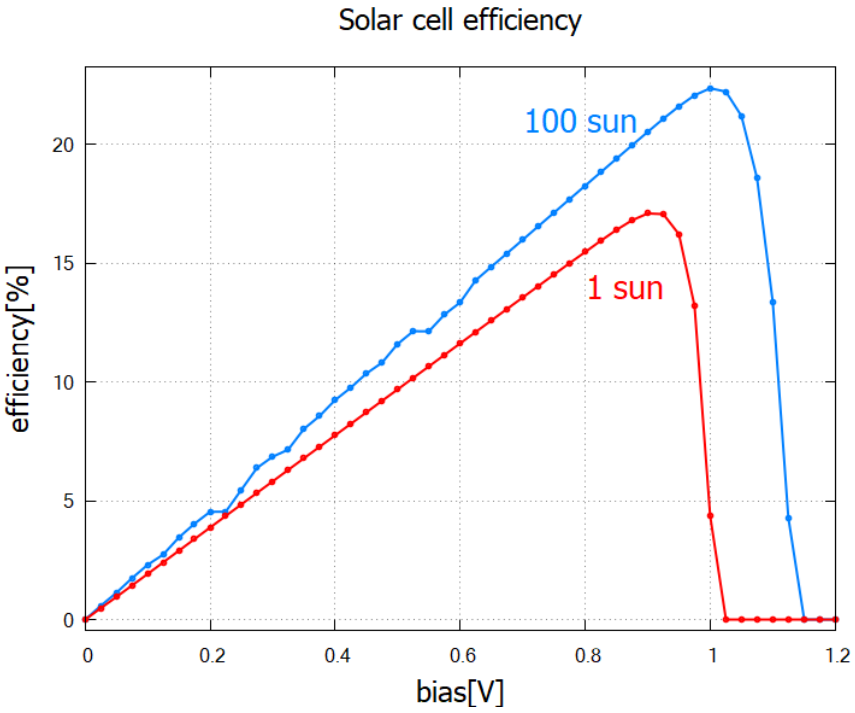


Figure 6.4.3.26: Solar cell efficiency η for no sunlight concentration (red) and 100-sun concentration (blue) by *nextnano*^{3}. The data is contained in *current/solar_cell_efficiency.dat*.

```
Solar cell results
*******************************************************************************************
short-circuit current:       I_sc    =       184.149021 [A/m^2]  (photo current: It␣
↪increases with smaller band gap.)
open-circuit voltage:        U_oc    =        -1.012500 [V]      (U_oc <= built-in␣
↪potential ~ band gap)
current at maximum power:    I_max   =       180.613633 [A/m^2]
voltage at maximum power:    U_max   =        -0.900000 [V]
maximum power output:        P_max   = U_max * I_max =      -162.552270 [W/m^2]␣
↪(condition for maximum power output: dP/dV = 0)
maximum extracted power:     P_solar = - P_max       =       162.552270 [W/m^2]
incident power:              P_in    =                      1000.369631 [W/m^2]
ideal conversion efficiency: eta     = P_max / P_in  =        16.249221 %
fill factor:                 FF      =         0.871824
In practice, a good fill factor is around 0.8.
All these results are approximations.
They are only correct if a lot of voltage steps have been used (i.e. a high␣
↪resolution of bias steps).
```

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{ } (nextnano++) or $numeric-control (nextnano³).

Input Files for *nextnano³*:

- *1DGaAs_SolarCell_nn3.in*
- *1DGaAs_SolarCell_nn3_import_generation.in*

Last update: nn/nn/nnnn

Cascade solar cell (Tandem solar cell)

Input Files:

- *1DCascadeSolarCell_nnp.in / *nn3.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)

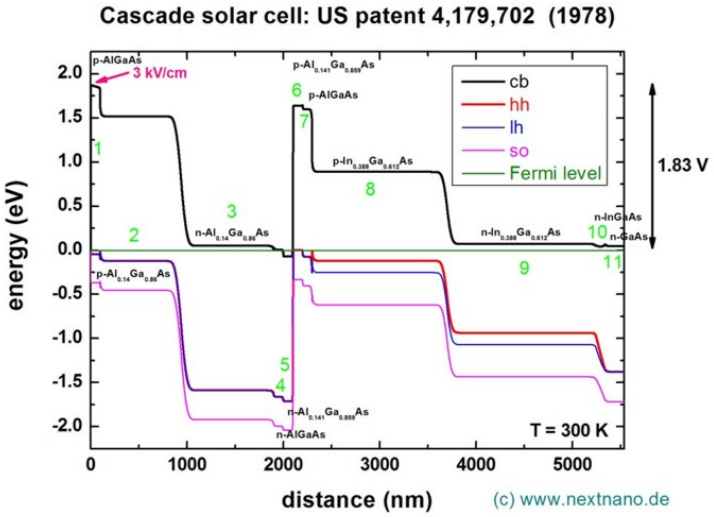
Input files used in this tutorial are followings:

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.

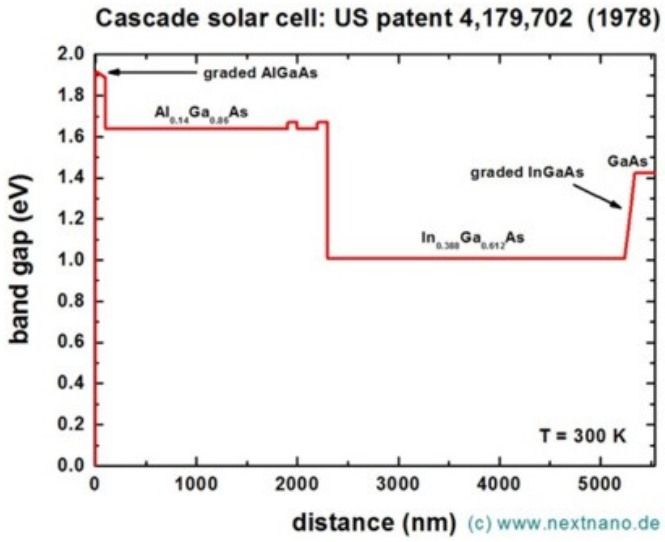
On the left side (region 1), a graded p-type AlGaAs layer has been used to generate an electric field of 3 kV / cm (= 30 meV / 100 nm). We assumed that all materials are strained with respect to the GaAs substrate, thus the degeneracy of heavy and light hole valence band edges is lifted, especially inside 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.

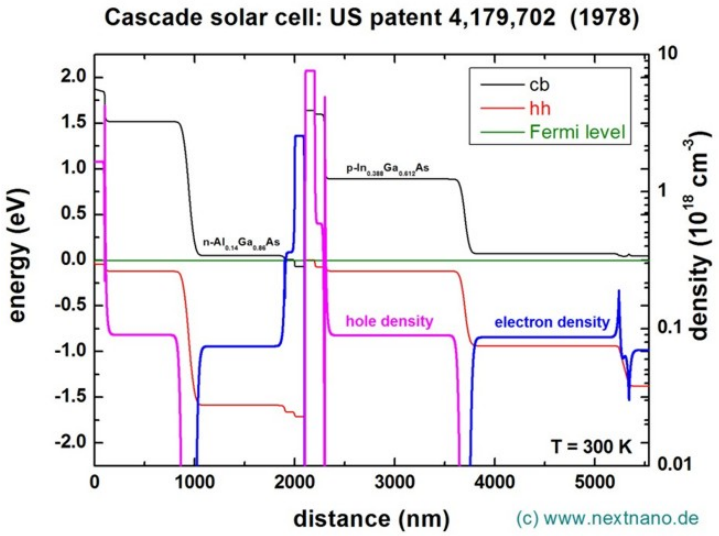
- *bias_00000/bandgap.dat* (*nextnano++*)
- *band_structure/BandGap1D.dat* (*nextnano*[3])



Electron and hole densities

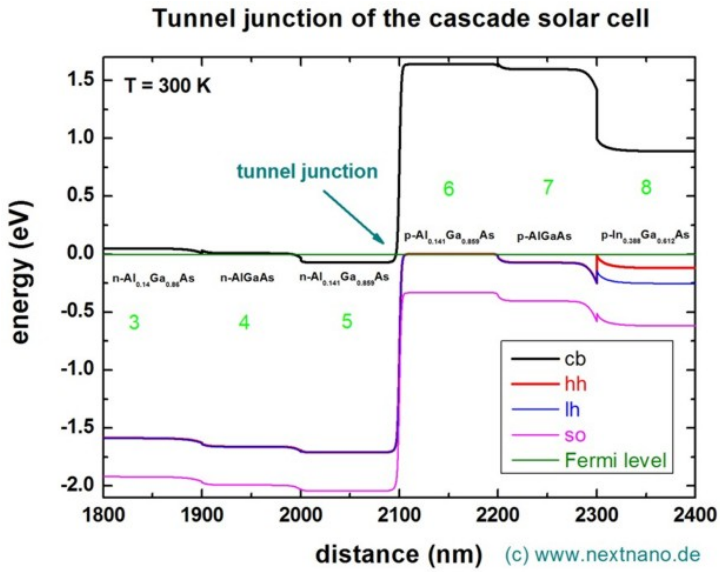
Here, the electron and hole densities are plotted. This data can be found in these files.

- *bias_00000/density_electron.dat*, *bias_00000/dentity_hole.dat* (*nextnano++*)
- *densities/density_el.dat*, *densities/density_hl.dat* (*nextnano*^{3})

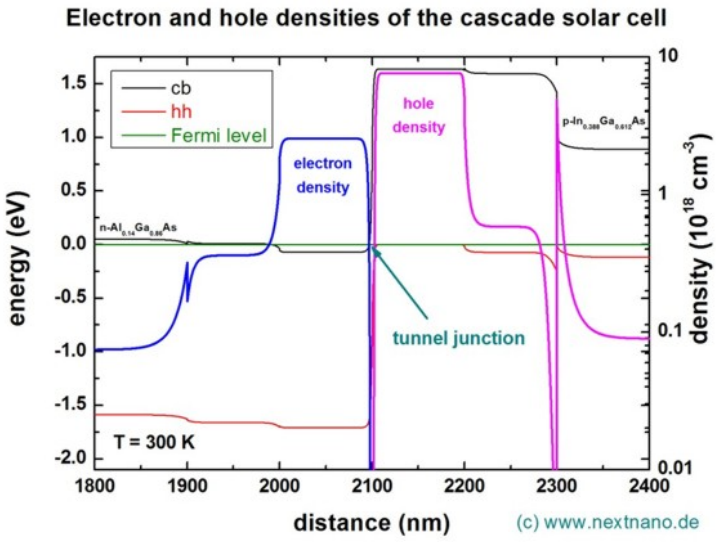


Tunnel junction

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).



What we can do on a solar cell using nextnano

We have the demonstration of the simulation for GaAs solar cell using nextnano GmbH 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++* and *nextnano*^{3} can calculate the generation rate $G(x)$ now. 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*.

Last update: 10/06/2024

6.4.4 Light-Emitting Diodes

InGaAs Multi-quantum well laser diode

- *Band structure*
- *Energy eigenstates and eigenvalues*
- *Charge densities*
- *Emission and absorption spectra*
- *Current and internal quantum efficiency*

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 6.4.4.1, 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.

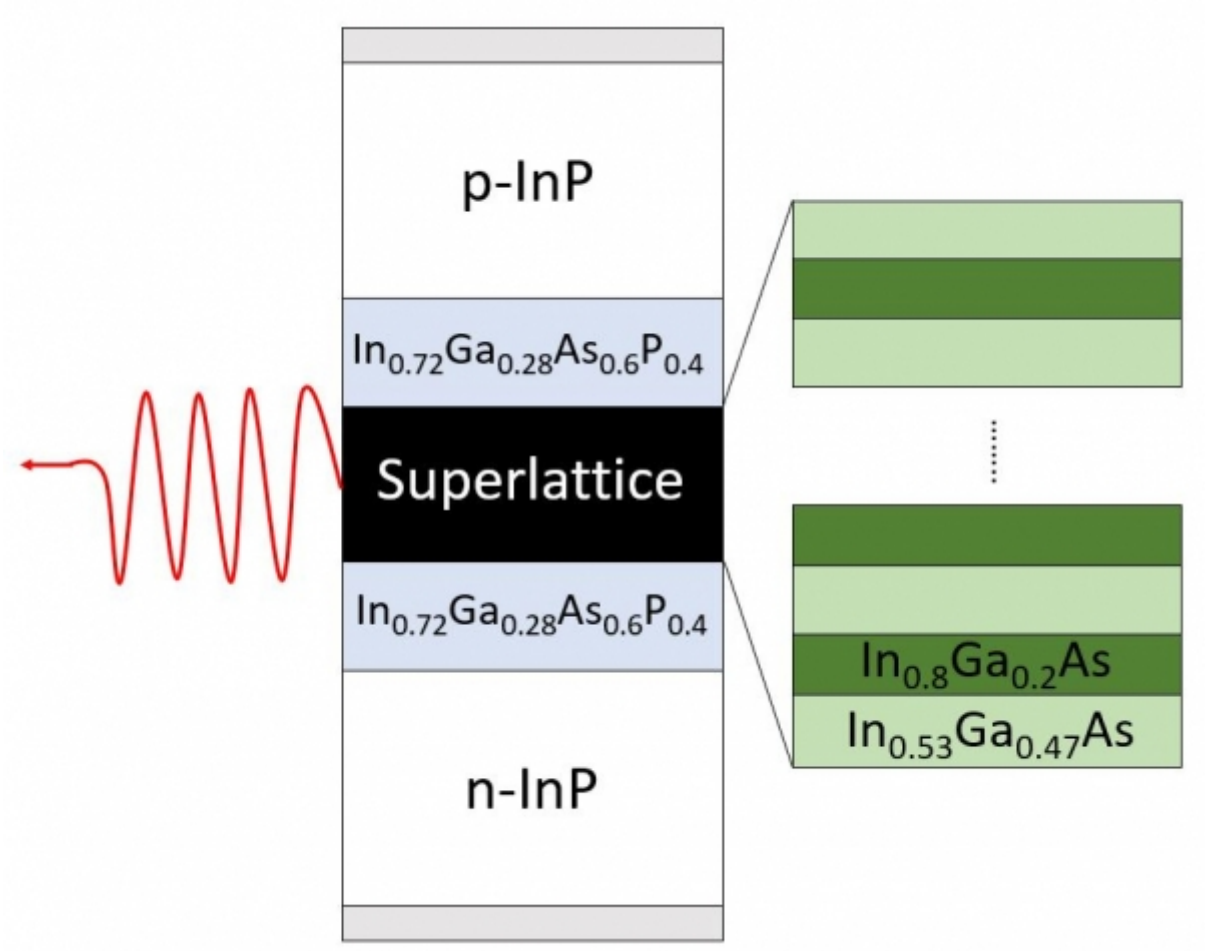


Figure 6.4.4.1: Structure overview

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

$$\begin{aligned} -e\frac{\partial n}{\partial t} + \nabla \cdot (-e\mathbf{j}_n(\mathbf{x})) &= -e(G(\mathbf{x}) - R(\mathbf{x})), \\ e\frac{\partial p}{\partial t} + \nabla \cdot e\mathbf{j}_p(\mathbf{x}) &= e(G(\mathbf{x}) - R(\mathbf{x})), \end{aligned} \tag{6.4.4.1}$$

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

$$\begin{aligned} \mathbf{j}_n(\mathbf{x}) &= -\mu_n(\mathbf{x})n(\mathbf{x})\nabla E_{F,n}(\mathbf{x}), \\ \mathbf{j}_p(\mathbf{x}) &= \mu_p(\mathbf{x})p(\mathbf{x})\nabla E_{F,p}(\mathbf{x}). \end{aligned} \tag{6.4.4.2}$$

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

$$\begin{aligned} \nabla \cdot \mu_n(\mathbf{x})n(\mathbf{x})\nabla E_{F,n}(\mathbf{x}) &= -(G(\mathbf{x}) - R(\mathbf{x})), \\ \nabla \cdot \mu_p(\mathbf{x})p(\mathbf{x})\nabla E_{F,p}(\mathbf{x}) &= G(\mathbf{x}) - R(\mathbf{x}), \end{aligned} \tag{6.4.4.3}$$

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

$$R_{\mathrm{rad}}^{spon}(\mathbf{x}, E) = C(\mathbf{x}) \int dE_h \int dE_e \; n(\mathbf{x}, E_e)p(\mathbf{x}, E_h)\delta(E_e - E_h - E), \tag{6.4.4.4}$$

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:

$$R_{rad,net}^{stim}(\mathbf{x}, E) = \left(1 - e^{\frac{E-(E_{Fn}-E_{Fp})}{k_B T}}\right) R_{rad}^{spon}(\mathbf{x}, E) \tag{6.4.4.5}$$

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 $R^{stim}_{rad,net}(E)$ as

$$\alpha(E) = \frac{\pi^2 \hbar^3 c^2}{n_r^2 E^2} \frac{R^{stim}_{rad,net}(E)}{V} \tag{6.4.4.6}$$

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 $R^{stim}_{rad,net}(E)$ 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 6.4.4.2.

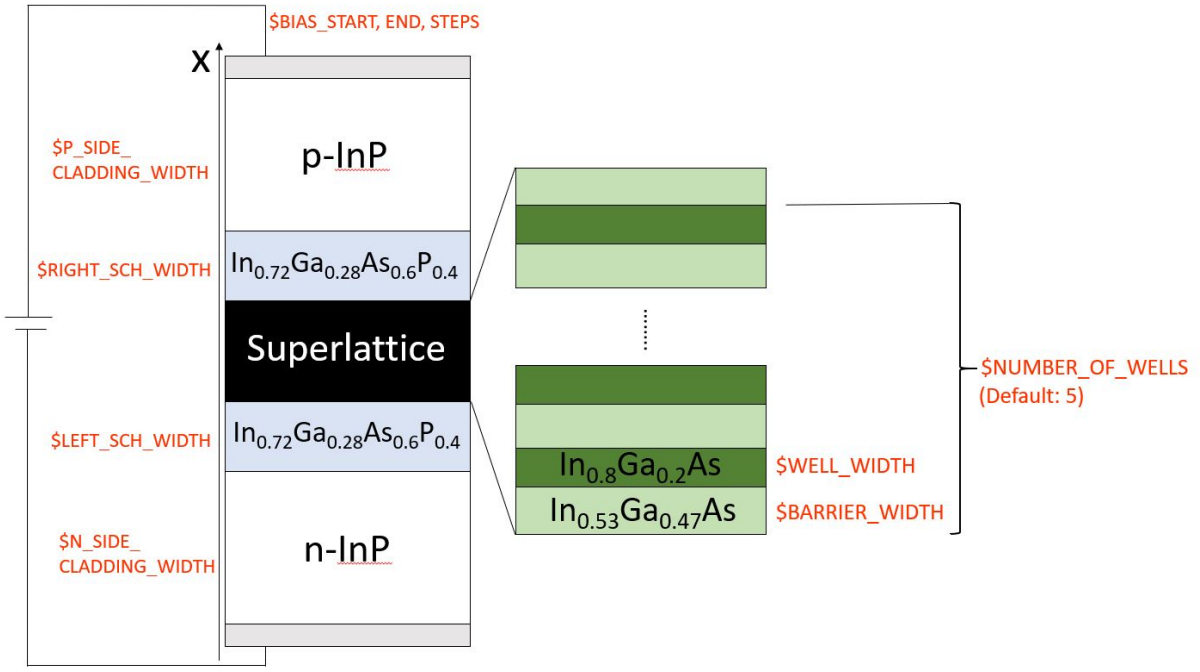


Figure 6.4.4.2: The definition of variables. The gray regions are contacts of 1nm thickness. `$NUMBER_OF_WELLS` determines the repetition of quantum wells. The program automatically sweeps the bias voltage starting from `$BIAS_START` until `$BIAS_END`, at intervals of `$BIAS_STEPS`.

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{
```

(continues on next page)

(continued from previous page)

```
        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_quantum_regions = yes
    }

    energy_resolved_density{
        only_quantum_regions = yes
        output_energy_resolved_densities{}
    }


    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
    }
}

# 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 6.4.4.3 shows the case for the bias 0.2 V. Here the quasi Fermi level of electrons is lower than the quantum wells.

For the bias 0.8 V (Figure 6.4.4.4), 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 6.4.4.12 shows the emission spectrum in this case. When the bias is too small, e.g. Figure 6.4.4.3, the intensity is much smaller, as can be seen in Figure 6.4.4.16.

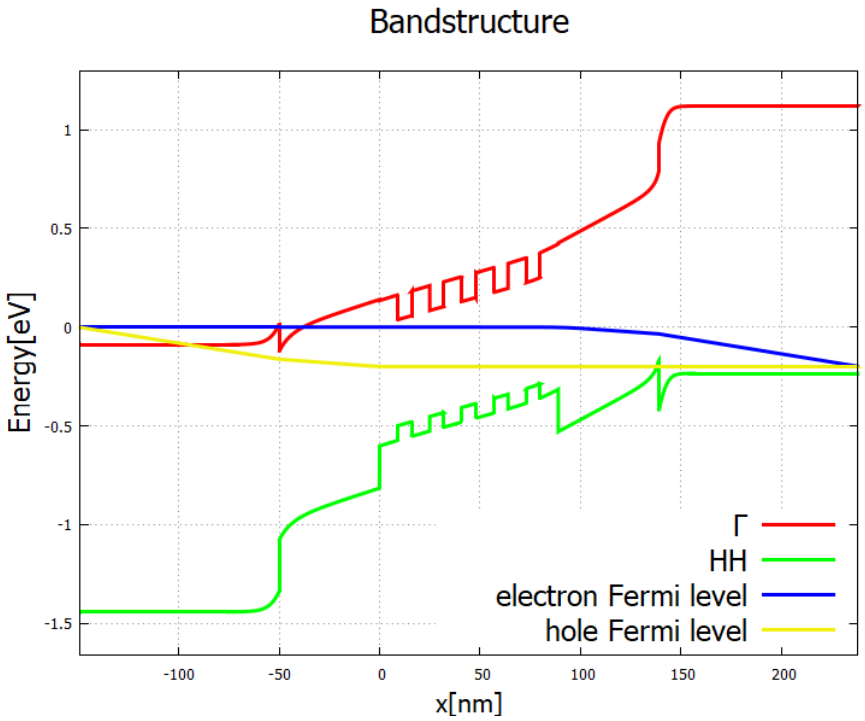


Figure 6.4.4.3: Band structure of the laser diode system for a low bias of 0.2 V.

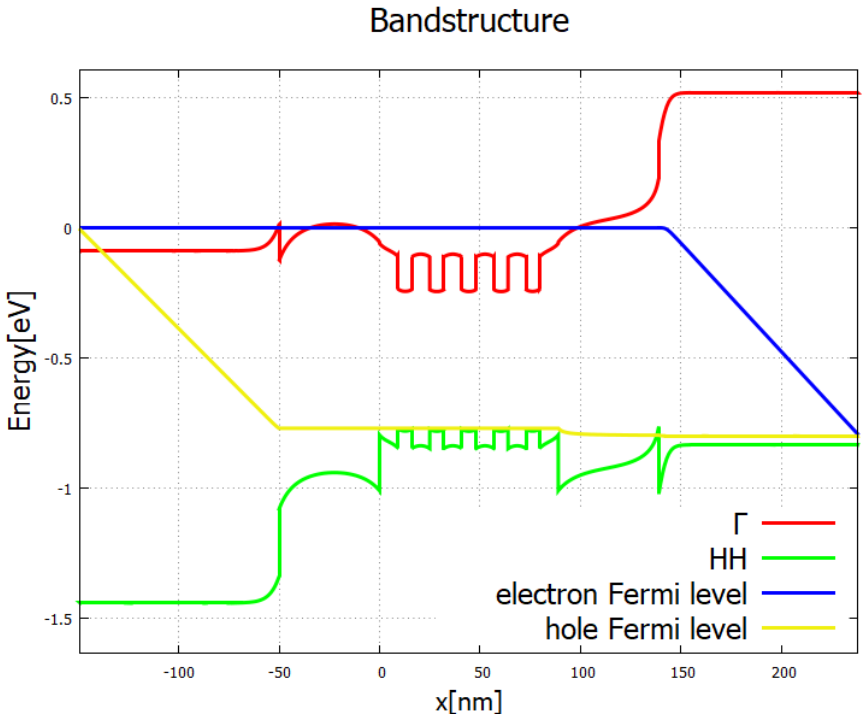


Figure 6.4.4.4: Band structure for a high bias 0.8 V. Electrons flowing from the left and holes from the right recombine in the active zone (multi-quantum well structure).

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 6.4.4.5 and Figure 6.4.4.6). The light hole and split-off states are out of the quantum wells and not of our interest here.

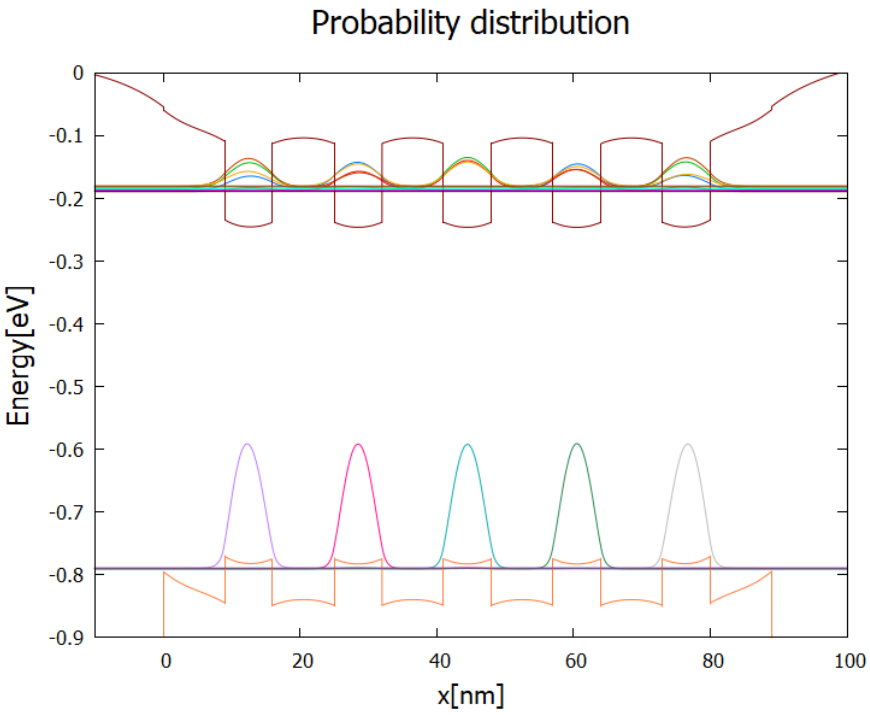


Figure 6.4.4.5: Probability distribution $|\psi(x)|^2$ of the lowest localized modes of electrons and holes for the band structure Figure 6.4.4.3. Horizontal lines are the corresponding eigenenergies.

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.

Note: These graphs are generated by *nextnanopy*.

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 6.4.4.12 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 6.4.4.13 with higher resolution.

The energy resolution in Figure 6.4.4.13 has been increased by a factor of 10 from Figure 6.4.4.12.

Note: Although these charge densities either with variable E or x are both obtained by integrating $n(x, E)$

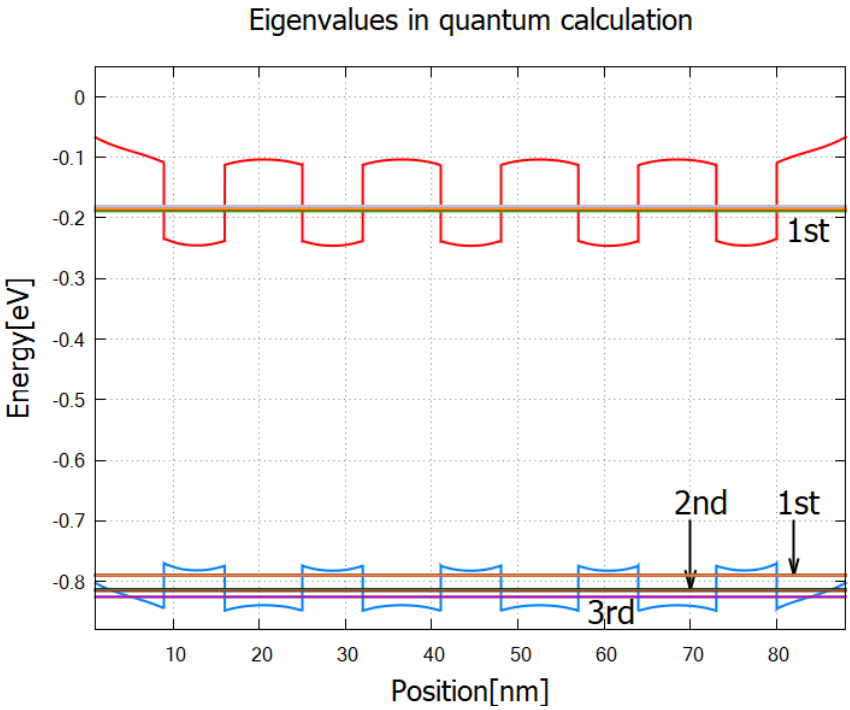


Figure 6.4.4.6: Eigenvalues of the Gamma-band up to 5th and heavy-hole-band states up to 13th in relation to band edges. The Eigenvalues above these are higher than the barrier energy of the quantum wells. The Gamma band has single "miniband", whereas the heavy-hole band has three. The 1st heavy-hole 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.

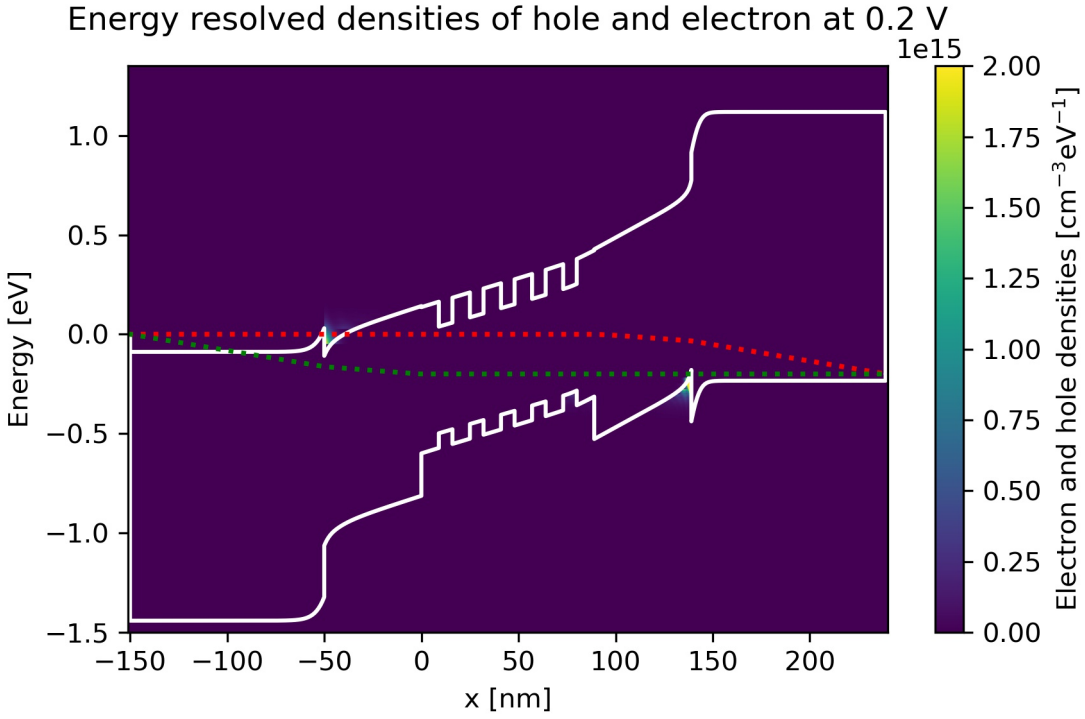


Figure 6.4.4.7: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.2 V in quantum calculation.

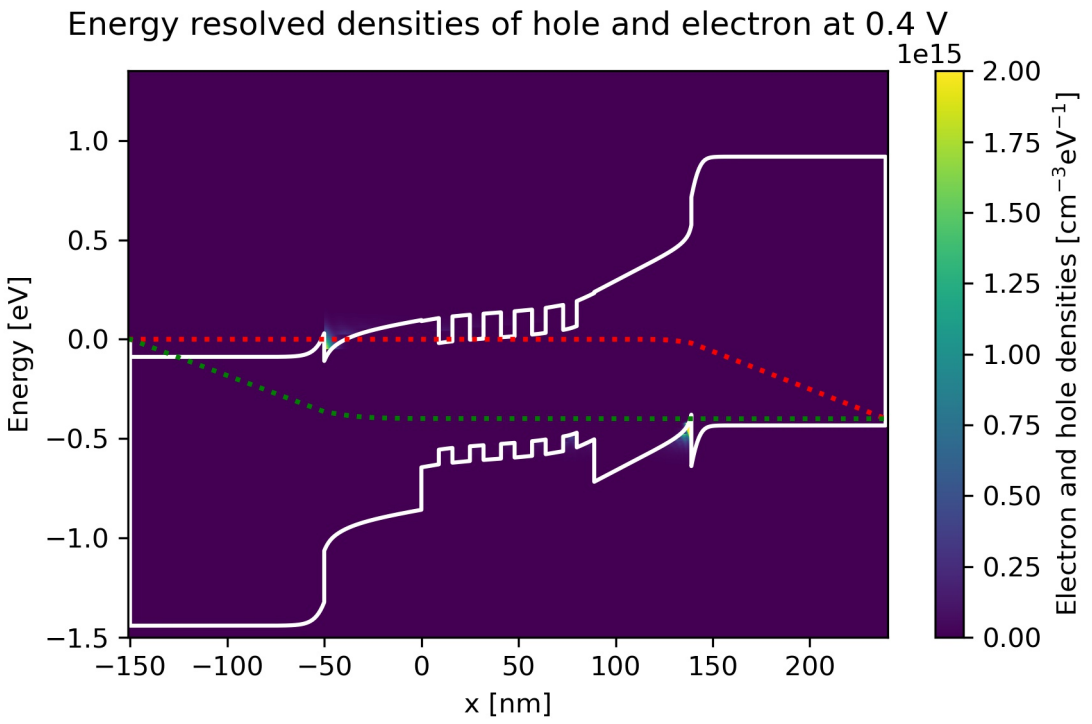


Figure 6.4.4.8: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.4 V in quantum calculation.

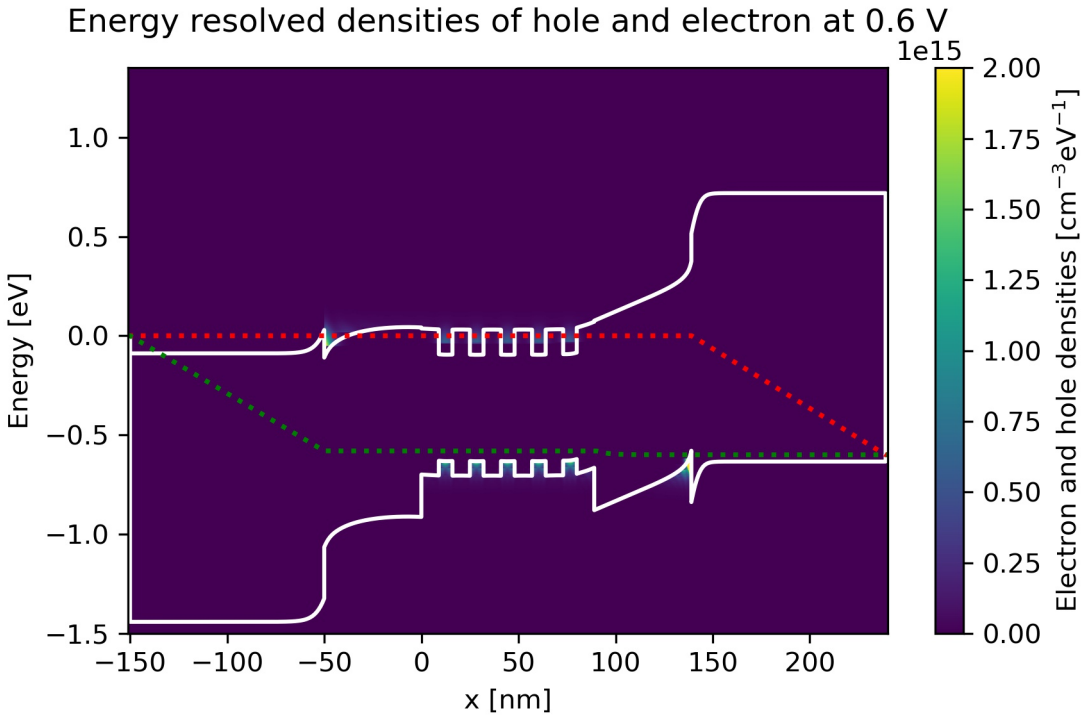


Figure 6.4.4.9: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.6 V in quantum calculation.

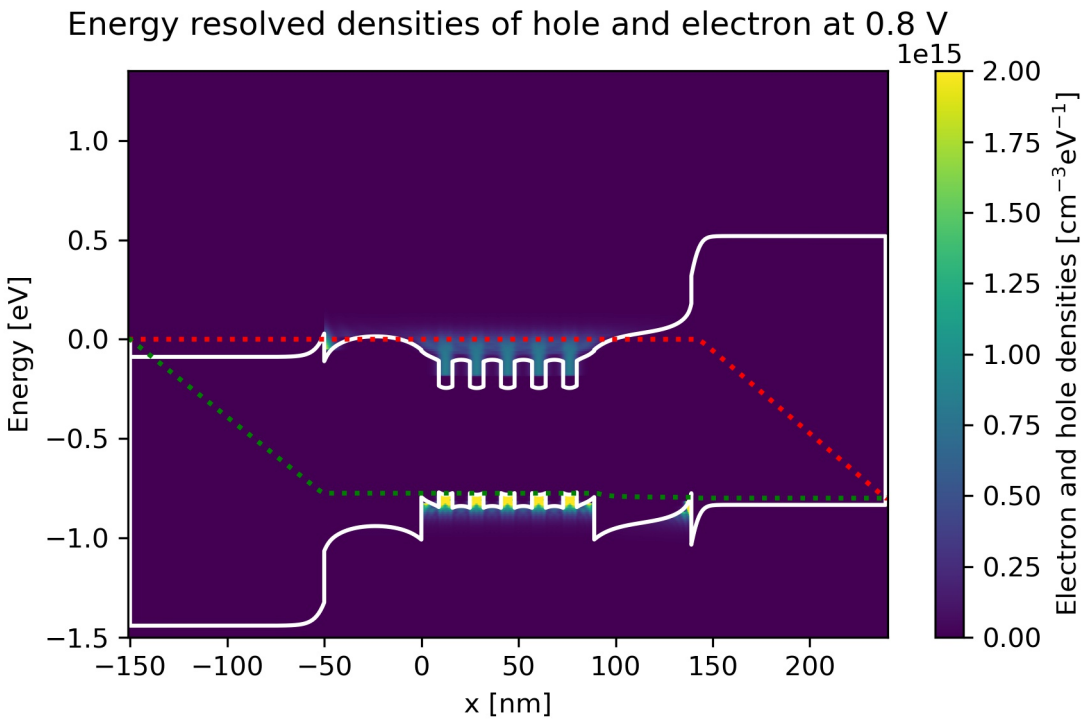


Figure 6.4.4.10: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 0.8 V in quantum calculation.

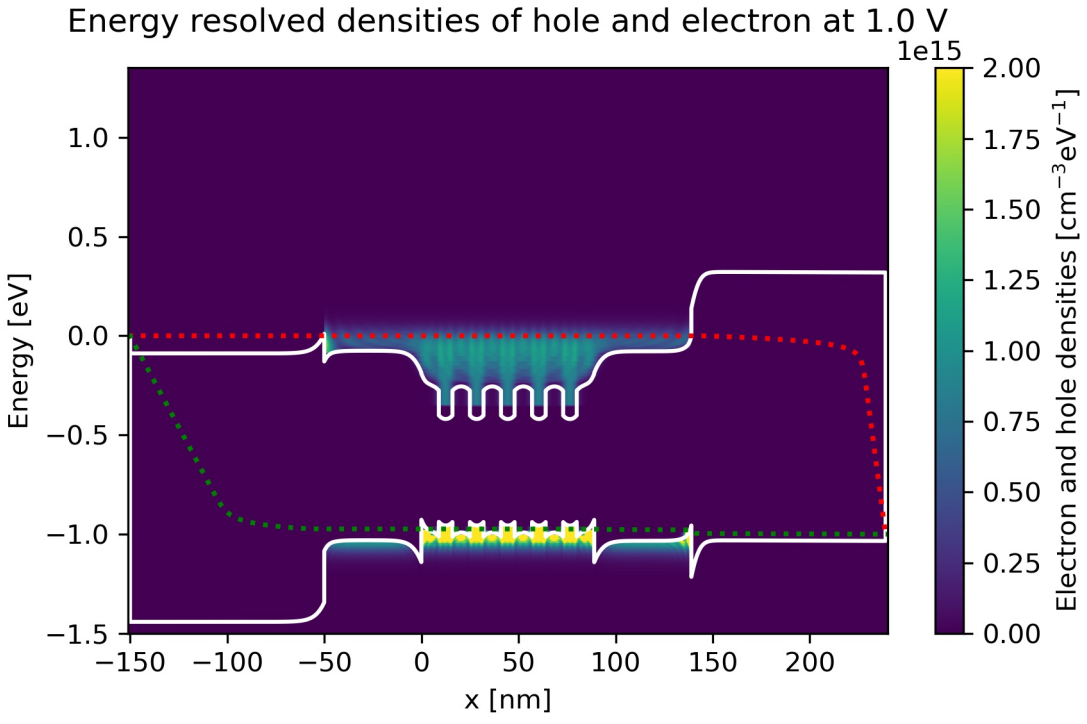


Figure 6.4.4.11: Energy-resolved electron and hole density, Gamma conduction band edge, HH valence band edge and quasi-Fermi levels at bias 1.0 V in quantum calculation.

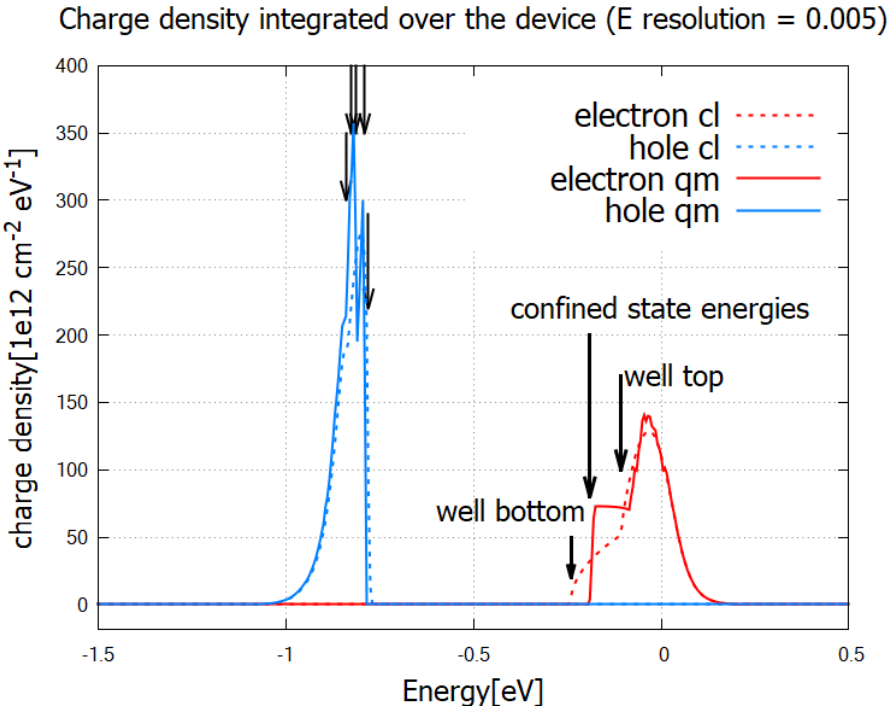


Figure 6.4.4.12: Electron (red) and hole (blue) densities integrated over the device as a function of energy.

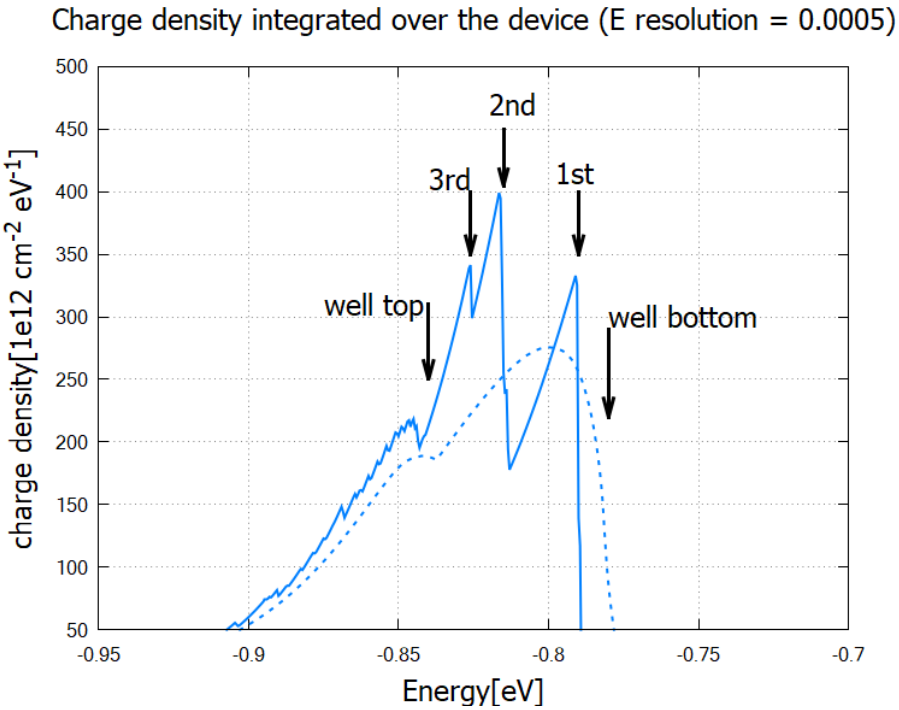


Figure 6.4.4.13: Hole density integrated over the device from classical (dashed) and quantum (solid) calculation.

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 6.4.4.14). The peak is at around 0.7-0.8eV, which is consistent with the charge distribution in Figure 6.4.4.12. 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*.

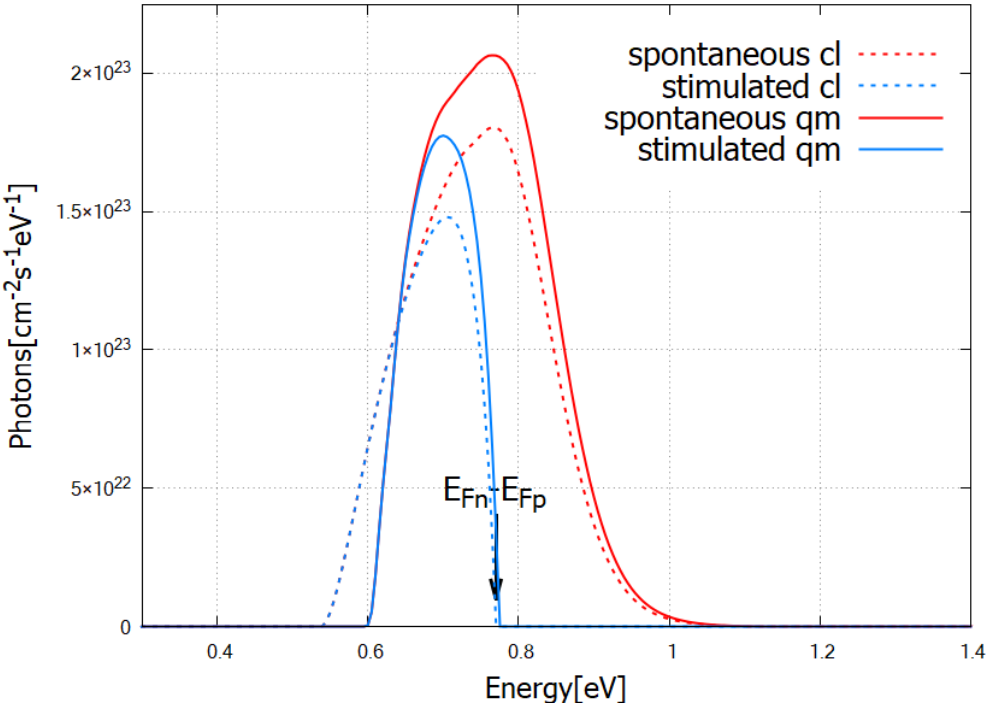


Figure 6.4.4.14: Emission spectrum of the laser diode for the bias 0.8 V.

The absorption spectra are calculated as

$$\alpha(E) = \frac{\pi^2 \hbar^3 c^2}{n_r^2 E^2} \frac{R_{rad,net}^{stim}(E)}{V}$$

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 $R_{rad,net}^{stim}(E)$ 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]*.

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{ }*.

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.

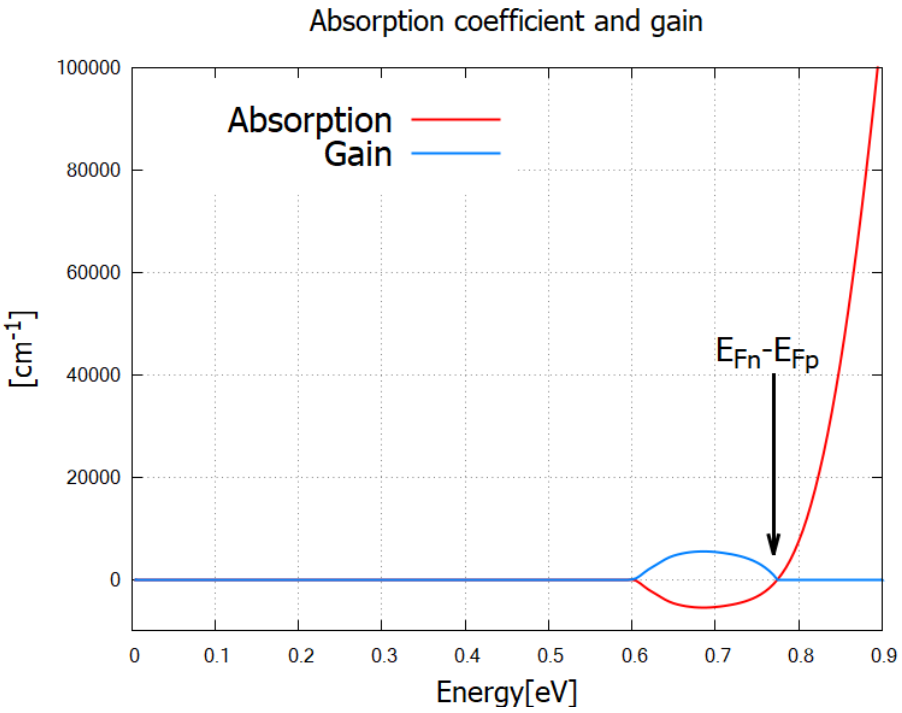


Figure 6.4.4.15: Classically calculated absorption and gain spectra. The sign of the spectrum switches at the energy corresponding to the quasi Fermi-level separation in the active region.

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 6.4.4.15, we compare the hole current and photocurrent.

Figure 6.4.4.16 clearly shows the consequence of the difference in band structures Figure 6.4.4.3 and Figure 6.4.4.4. 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*

$$\eta = \frac{I_{\text{photon}}}{I_{\text{charge}}}. \tag{6.4.4.7}$$

This quantity is written in `internal_quantum_efficiency.dat` and shown in Figure 16.

Last update: 16/07/2024

UV LED: Quantitative evaluation of the effectiveness of EBL

- *Header*
- *Structure*
- *Scheme*
- *Results*
  - *Current-voltage characteristics*
  - *Bandedges*
  - *Current Density*

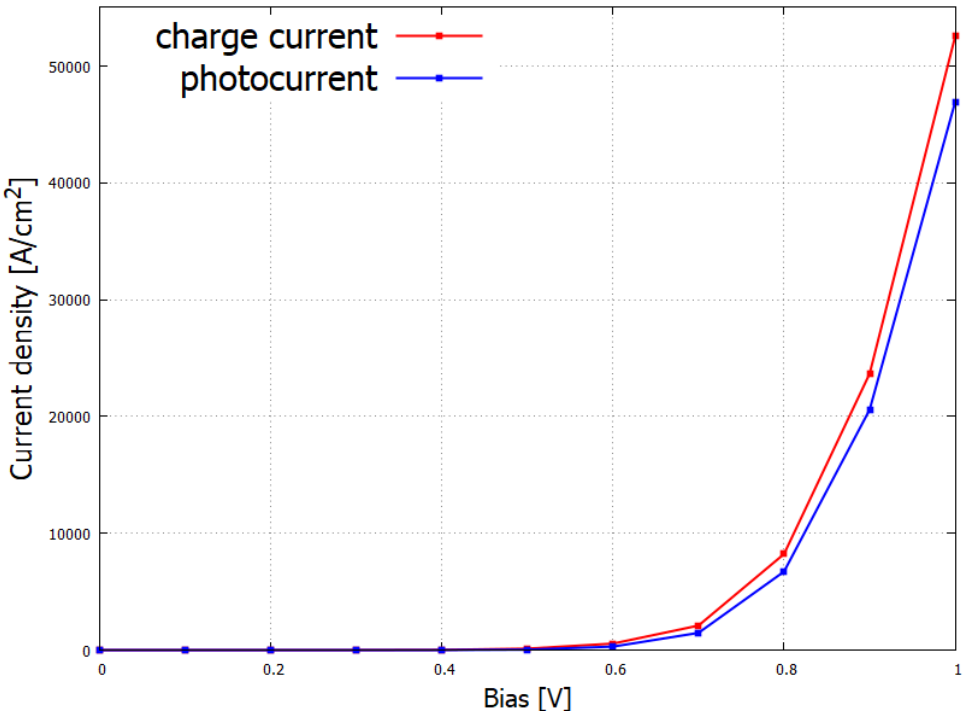


Figure 6.4.4.16: Charge current and photocurrent as a function of bias voltage (IV characteristics).

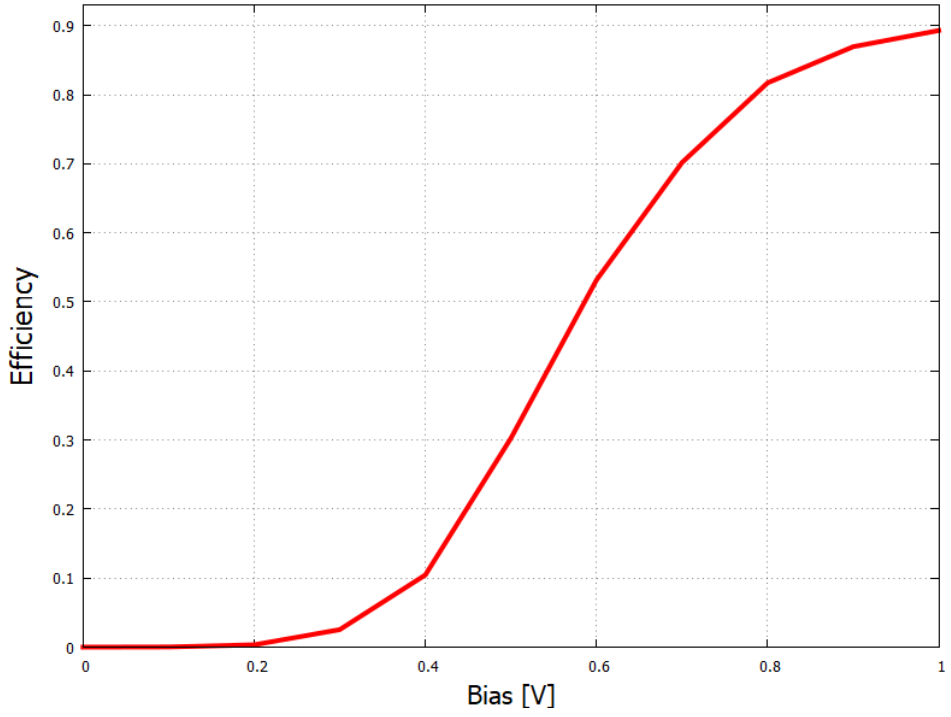


Figure 6.4.4.17: Conversion efficiency of the InGaAs laser diode.

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.

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{ }
}
```

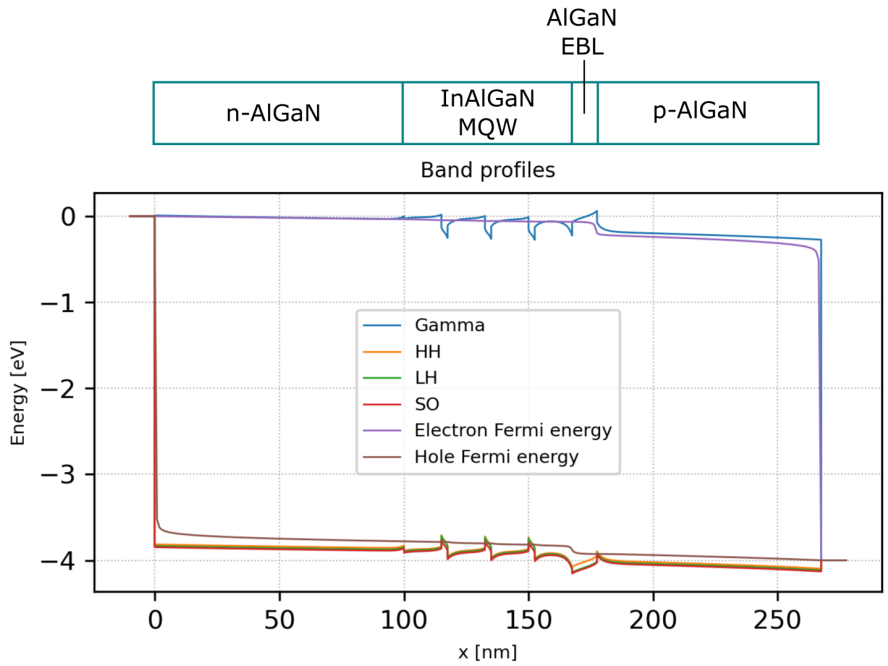


Figure 6.4.4.18: The band edges and Fermi levels for the structure with higher EBL (x=0.28, bias=4.00V, total current density=1.67×10^5 A/cm^2)

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 *General scheme of the optical device analysis*.

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 (6.2.5.28). 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.

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.

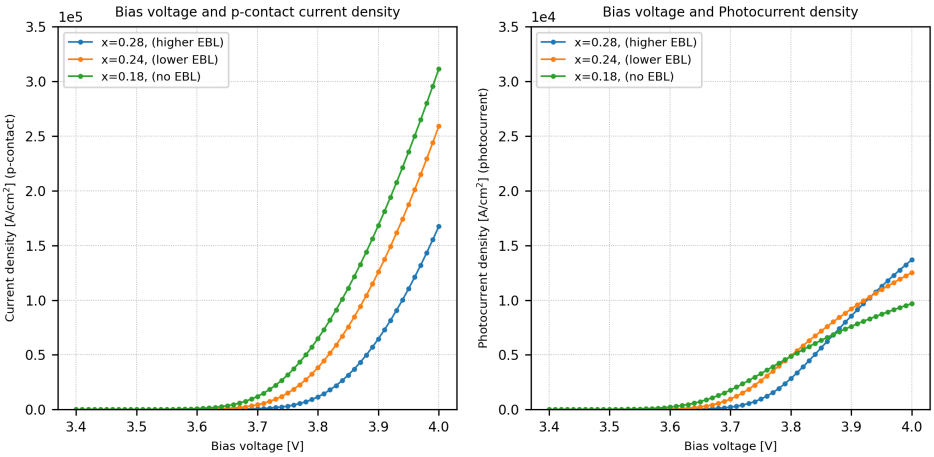


Figure 6.4.4.19: (*Left*:) The relationship between the p-contact current density and bias voltage. (*Right*:) The relationship between the photocurrent I_{photo} and bias voltage.

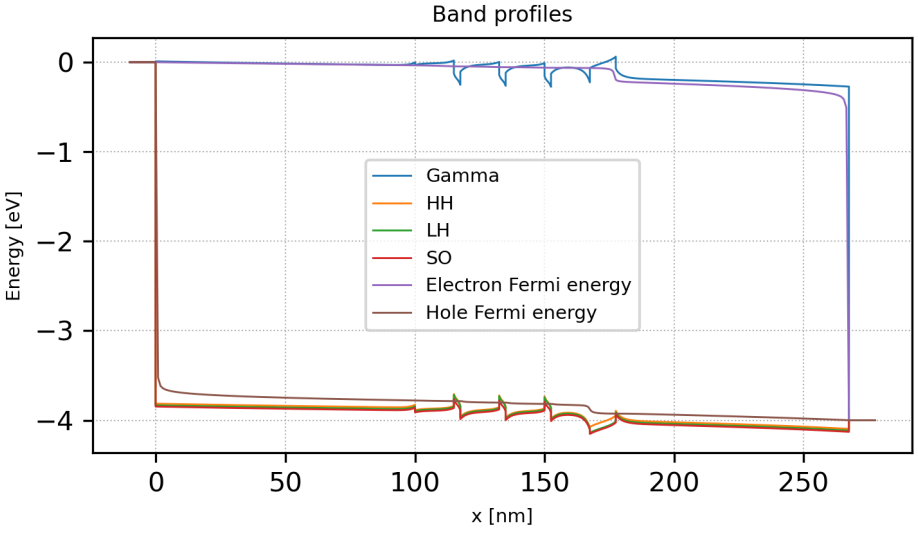


Figure 6.4.4.20: The band edges and Fermi levels for the structure with EBL (x=0.28, bias=4.00V, total current density=1.67×10^5 A/cm^2)

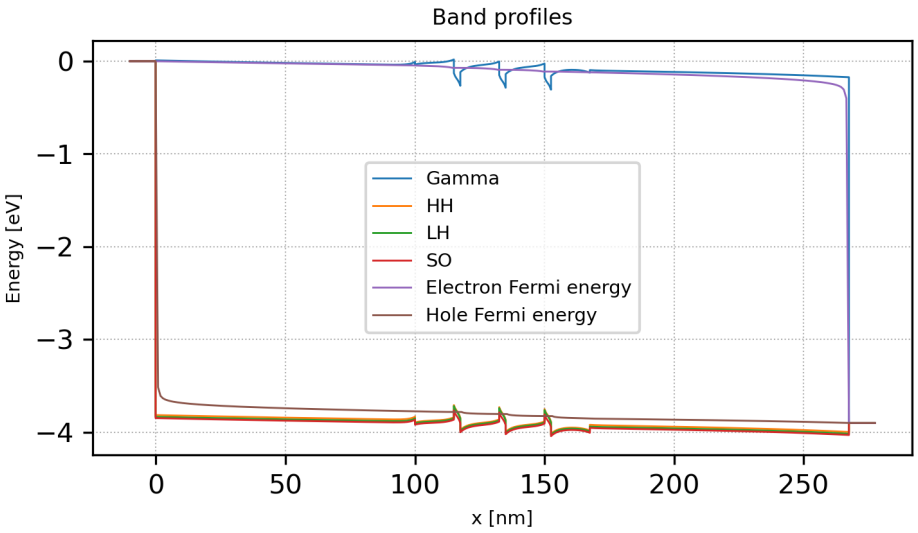


Figure 6.4.4.21: The band edges and Fermi levels for the structure without EBL (x=0.18, bias=3.90V, total current density=1.68×10^5 A/cm^2)

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 $\times$ 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 $\times$ 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 (6.2.5.33) and current density of p-contact for each structure.

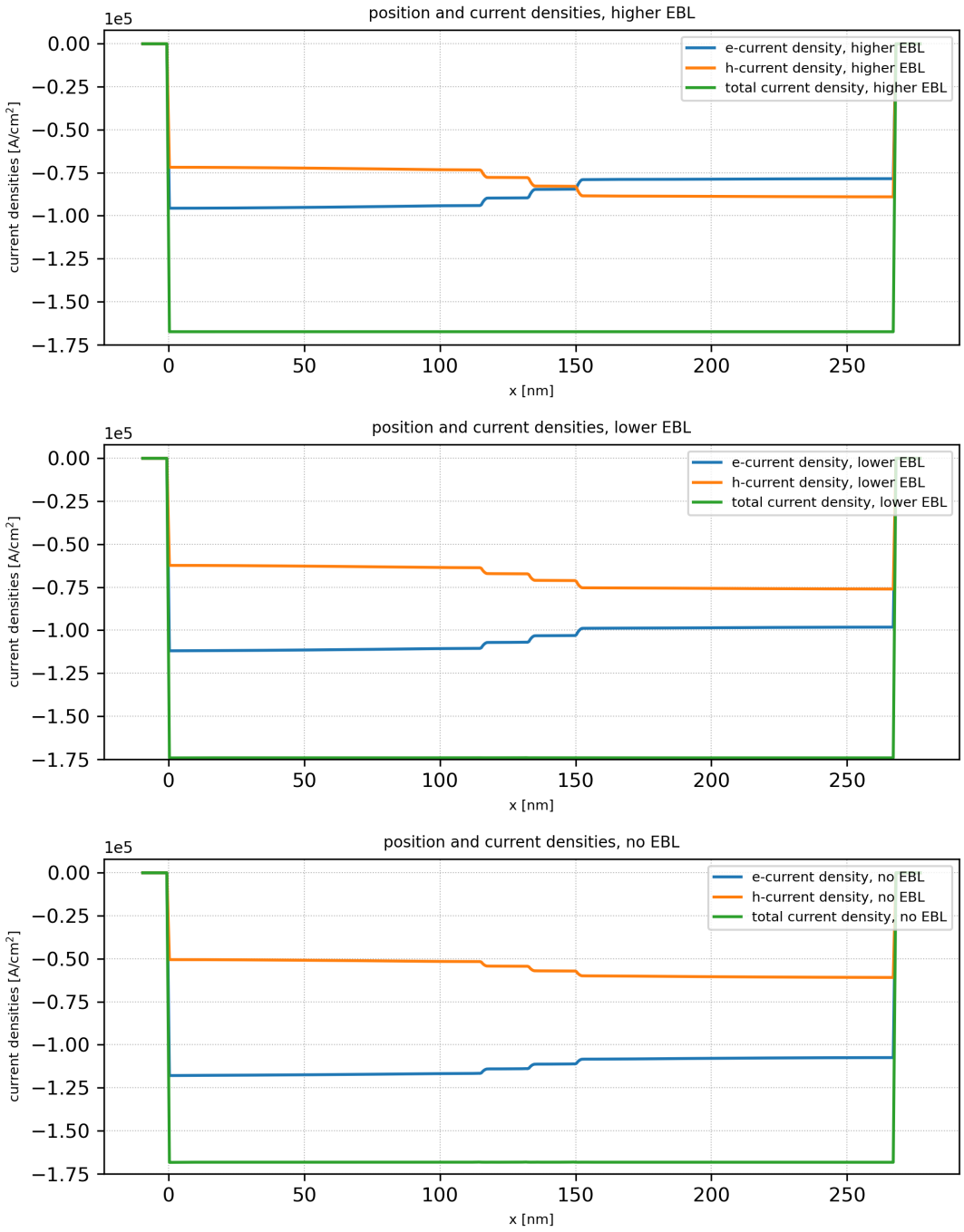


Figure 6.4.4.22: The current density profile for the the structures with higher EBL (*top*, 4.00 V, 1.67×10^5 A/cm^2), lower EBL (*middle*, 3.94 V, 1.74×10^5 A/cm^2), and no EBL (*bottom*, 3.90 V, 1.68×10^5 A/cm^2).

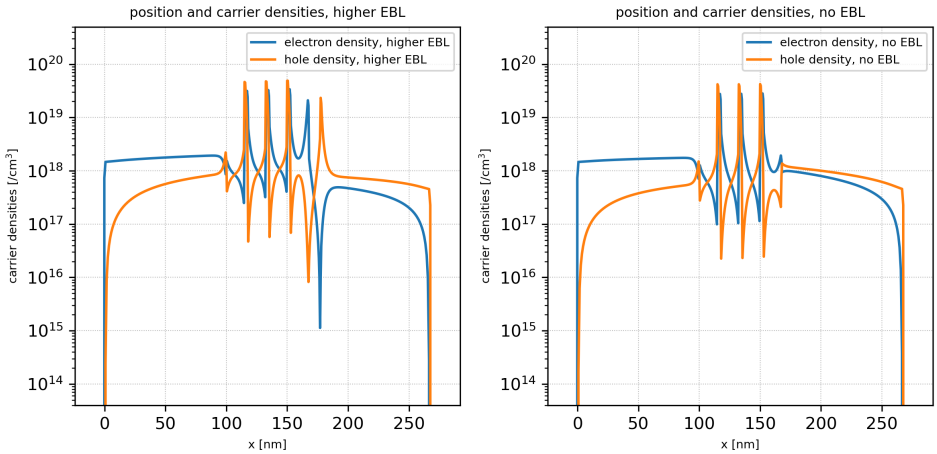


Figure 6.4.4.23: The electron and hole densities calculated in the structures with higher EBL (*left*, 4.00 V, 1.67× 10^5 A/cm^2) and no EBL (*right*, 3.90 V, 1.68× 10^5 A/cm^2).

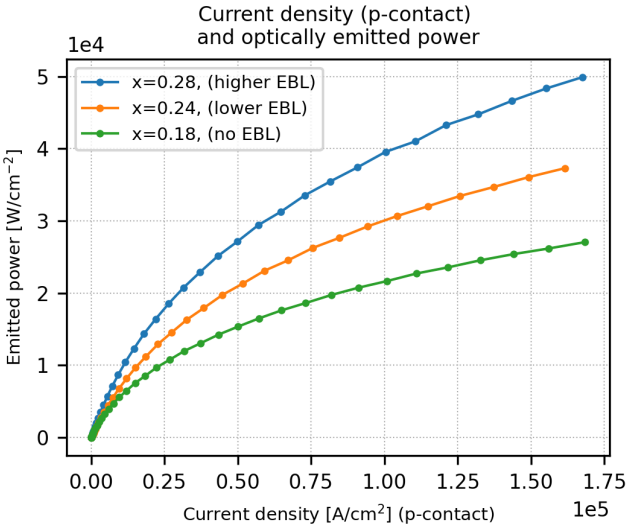


Figure 6.4.4.24: Current vs. power of light emission

Internal quantum efficiency

In *nextnano++*, the **internal quantum efficiency** is calculated as

$$\eta_{\mathrm{IQE}} = \eta_{\mathrm{VQE}} \cdot \eta_{\mathrm{IE}} = \frac{I_{\mathrm{photo}}}{I_{\mathrm{total}}} \tag{6.4.4.8}$$

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:

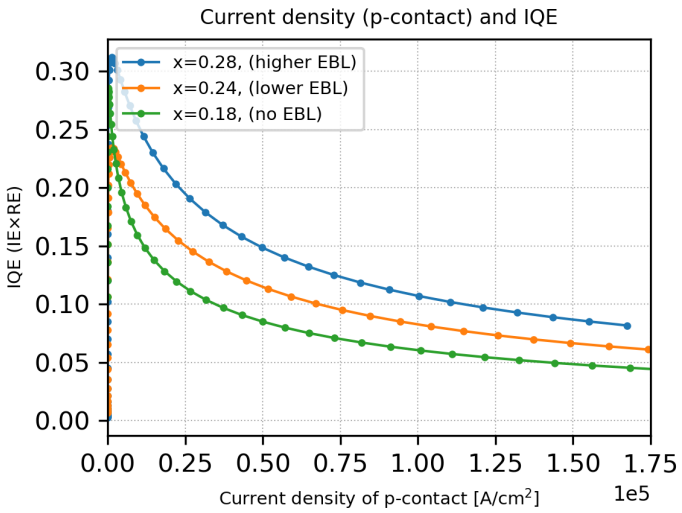


Figure 6.4.4.25: Current and internal quantum efficiency (IQE).

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

$$\eta_{\mathrm{VQE}} = \frac{R^{stim}_{\mathrm{rad,net}} + R_{\mathrm{fixed}}}{R_{\mathrm{total}}} \tag{6.4.4.9}$$

and also shows the improvement by the introduction of EBL:

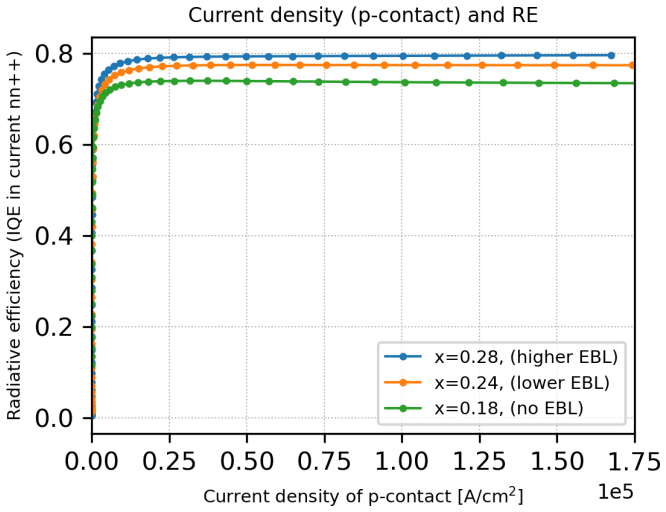


Figure 6.4.4.26: Current and volume quantum effciency (radiative efficiency).

The IQE can be decomposed like (6.4.4.8) 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} :

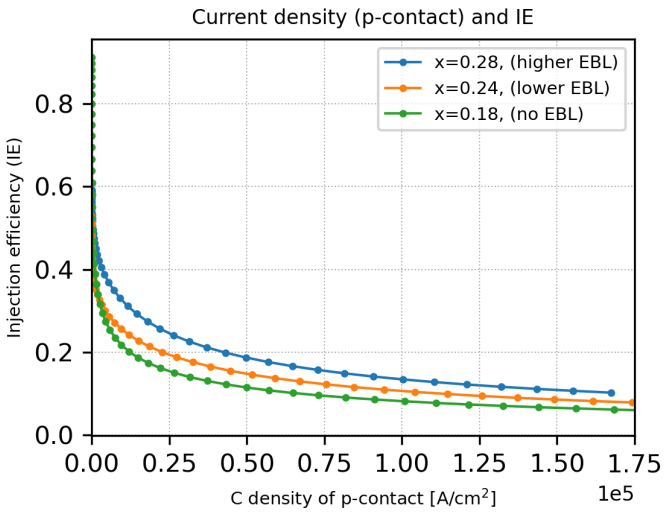


Figure 6.4.4.27: Current and injection efficiency (IE).

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

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	$L = L_{\text{well}} = L_{\text{barrier}}$	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.

The band edge tilting is due to the piezo- and pyro-electricity, which actually enhances the acceptor ionization as can be seen later.

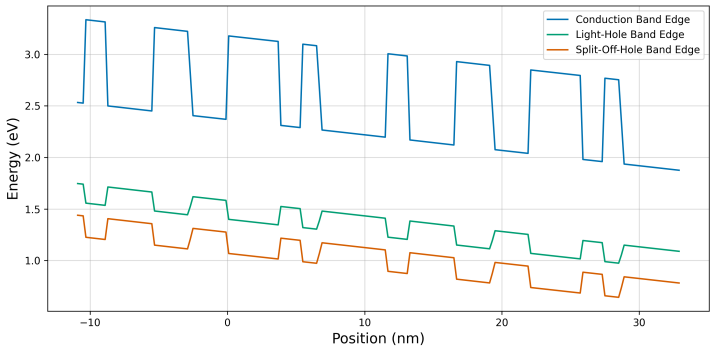


Figure 6.4.4.28: The band edge profile and the Fermi level

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 *General scheme of the optical device analysis*.

Ionization of dopant

The ionized donor and acceptor densities, N_D^+, N_A^- are calculated as

$$N_D^+(\mathbf{x}) = \sum_{i \in \text{Donors}} \frac{N_{D,i}(\mathbf{x})}{1 + g_{D,i} \exp((E_{F,n}(\mathbf{x}) - E_{D,i}(\mathbf{x}))/k_B T)} \tag{6.4.4.10}$$

$$N_A^-(\mathbf{x}) = \sum_{i \in \text{Acceptors}} \frac{N_{A,i}(\mathbf{x})}{1 + g_{A,i} \exp((E_{A,i}(\mathbf{x}) - E_{F,p}(\mathbf{x}))/k_B T)} \tag{6.4.4.11}$$

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

$$E_D(\mathbf{x}) = E_{c,0}(\mathbf{x}) - e\phi(\mathbf{x}) - E_D^{ion}(\mathbf{x}), \tag{6.4.4.12}$$

$$E_A(\mathbf{x}) = E_{v,0}(\mathbf{x}) - e\phi(\mathbf{x}) + E_A^{ion}(\mathbf{x}). \tag{6.4.4.13}$$

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 $E^{ion}_{D,i}, E^{ion}_{A,i}$ 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.

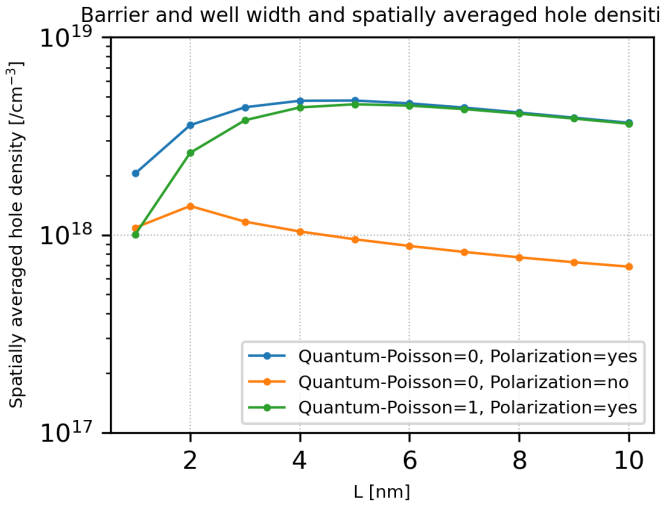


Figure 6.4.4.29: Barrier and well width L and spatially averaged hole densities.

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.

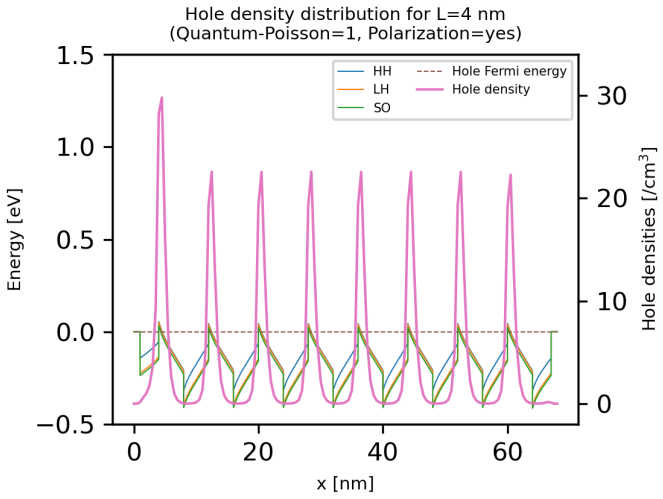


Figure 6.4.4.30: Hole density distribution calculated at L =4.0 nm by Schrödinger-Poisson equation including the polarization fields. The valence band edges are also displayed.

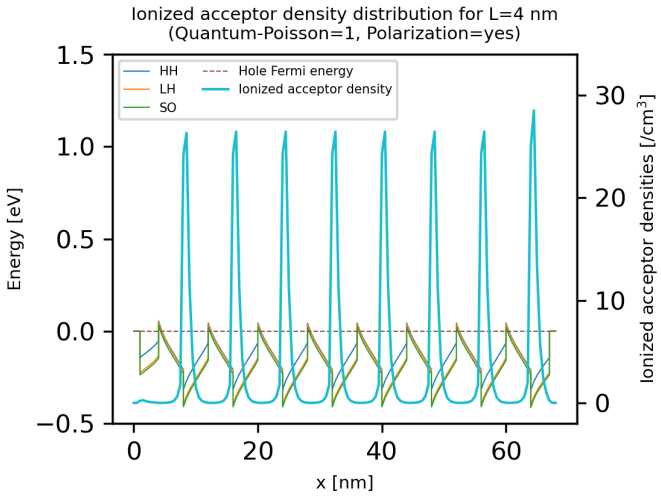


Figure 6.4.4.31: Ionized acceptor density distribution calculated at L =4.0 nm by Schrödinger-Poisson equation including the polarization fields. The valence band edges are also displayed.

IQE estimation

Structure

The simulation region consists of the following structure:

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.

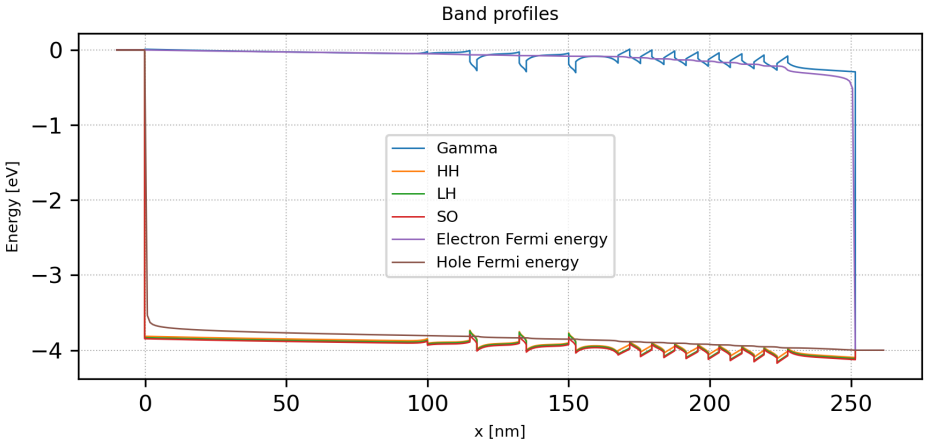


Figure 6.4.4.32: The band edges and Fermi levels for the structure with SL (bias=4.00 V, total current density=2.67×10^5 A/cm^2)

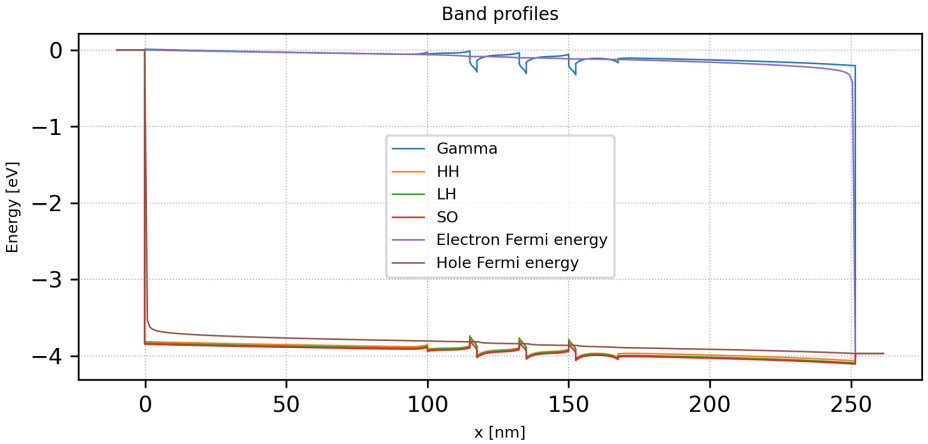


Figure 6.4.4.33: The band edges and Fermi levels for the structure with bulk p-region (bias=3.97 V, total current density=2.71×10^5 A/cm^2)

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 *General scheme of the optical device analysis*.

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.

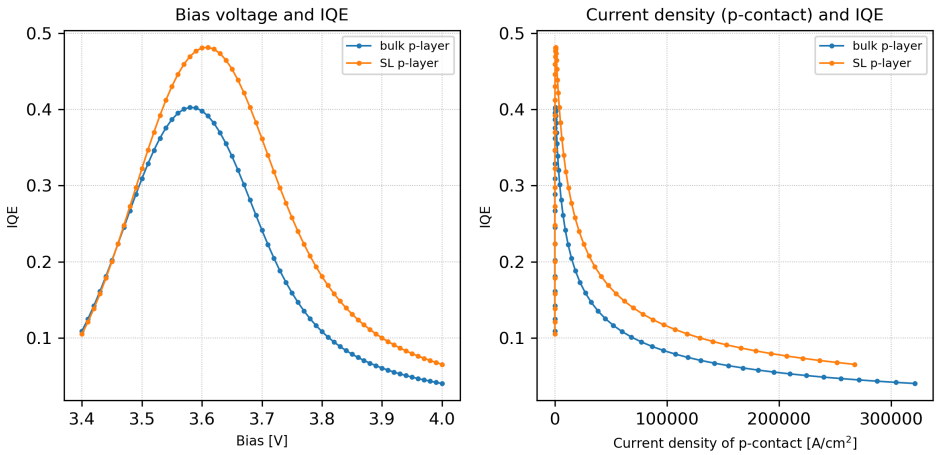


Figure 6.4.4.34: *Left*: Applied bias and IQE. *Right*: Current density at p-contact and IQE.

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.

The graphs shown in this tutorial are also generated by a python script using *nextnanopy*.

Last update: nn/nn/nnnn

6.4.5 Quantum Mechanics

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:

$$E_n = \left(n - \frac{1}{2}\right)\hbar\omega_0 \tag{6.4.5.1}$$

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 6.4.5.1 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.

The figure is in perfect agreement with Fig. 3.11 (p. 83) of *[HarrisonQWWD2005]*.

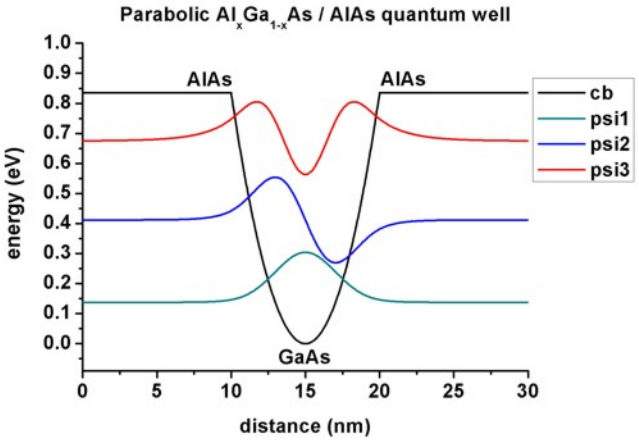


Figure 6.4.5.1: Calculated conduction band edge and the three lowest electron wave functions that are confined inside the parabolic QW. The energies were shifted so that the conduction band edge of $GaAs$ equals 0 eV.

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 (6.4.5.1) 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 6.4.5.2 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]*.

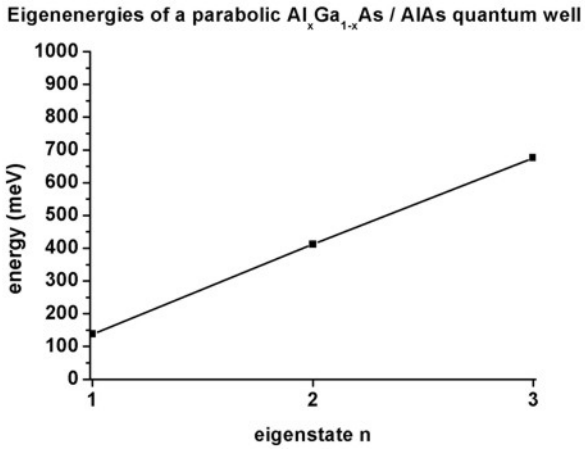


Figure 6.4.5.2: Eigenenergies for the three lowest states.

Matrix elements

The following matrix elements have been calculated:

- `interband_matrix_elements{}`: This spatial overlap matrix elements $\langle\psi_f|\psi_i\rangle$ simply returns the Kronecker delta as expected, because the wave functions are orthogonal.
- `intraband_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 6.4.5.3 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).

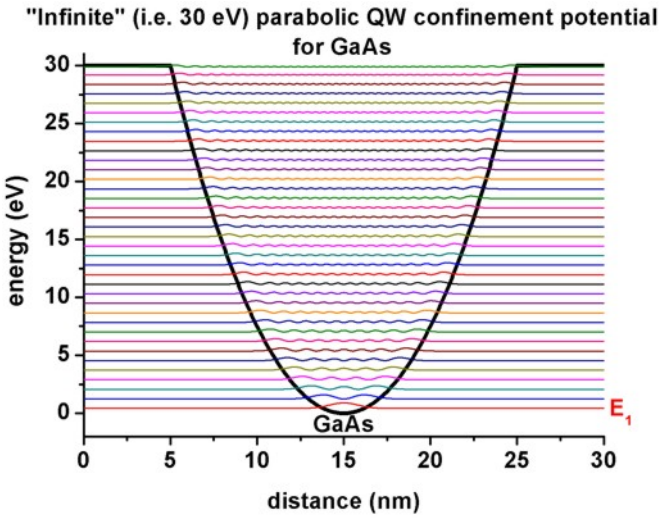


Figure 6.4.5.3: Calculated conduction band profile and probability densities (Ψ^2) for eigenstate n (n = 1, 2, ..., 37).

Figure 6.4.5.4 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 intraband 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\intraband_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.

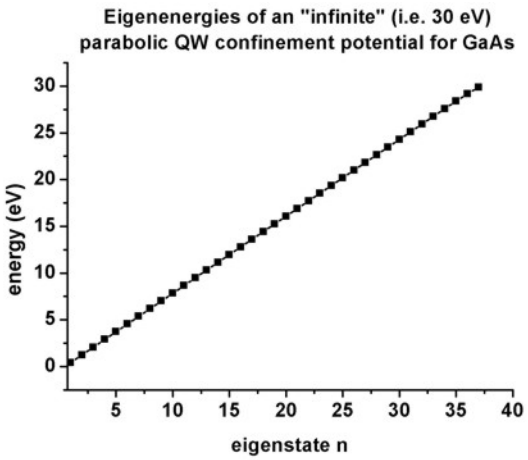


Figure 6.4.5.4: Eigenenergies of 37 eigenstates for an infinite parabolic QW.

"Infinite" (30 eV) half-parabolic QW confinement for GaAs

Input file: *1DGaAs_ParabolicQW_infinite_half_nnpp.in*

Figure 6.4.5.5 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).

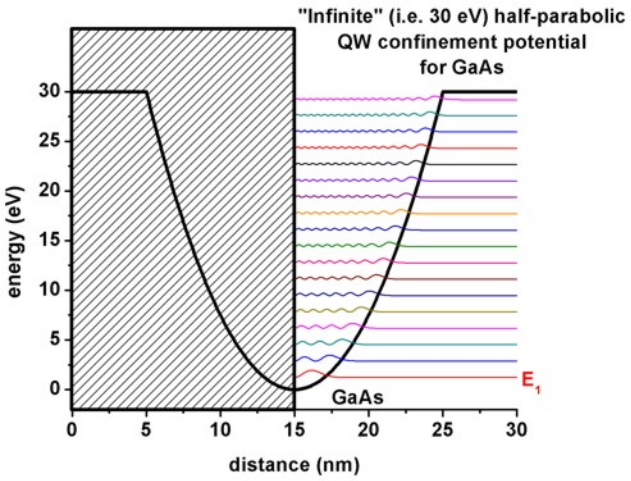


Figure 6.4.5.5: Calculated conduction band profile and probability densities (Ψ^2) for all confined eigenstates in an half-parabolic potential.

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 intraband 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\intraband_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: nn/nn/nnnn

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

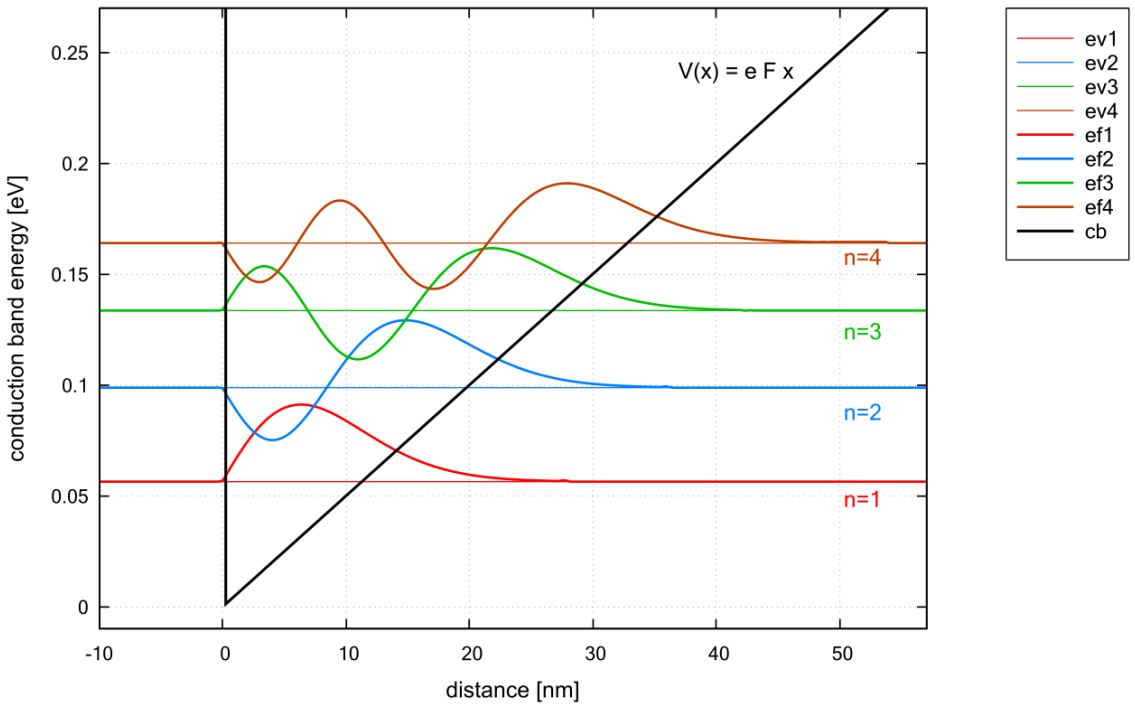
The Schrödinger equation for the transverse component of the electronic wave function has the following form inside the well:

$$\left[-\frac{\hbar^2}{2m^*}\frac{d^2}{dx^2} + eFx \right] \psi(x) = E\psi(x)$$

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):

$$c_n \simeq \left[\frac{3}{2} \pi \left(n - \frac{1}{4} \right) \right]^{2/3}$$

The *nextnano++* and *nextnano*^{3} eigenvalues for the lowest four eigenstates are in very good agreement with the analytic results:

	nextnano++ eigenvalue	*nextnano*^{3} eigenvalue	calculated eigenvalue	c_n (exact)	c_n (approximated)
n = 1	0.05647	0.05644	0.05664	$c_1 = 2.338$	$c_1 = 2.320251$
n = 2	0.09887	0.09882	0.09889	c_2	$c_2 = 4.081810$
n = 3	0.13358	0.13351	0.13365	c_3	$c_3 = 5.517164$
n = 4	0.16426	0.16416	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.

Last update: nn/nn/nnnn

— FREE — Double Quantum Well

Input files:

- *DoubleQuantumWell_6_nm_nnpp.in*
- *DoubleQuantumWell_6_nm_nn3.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.*

Input files for both the *nextnano++* and *nextnano*^{3} tools are available.

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".

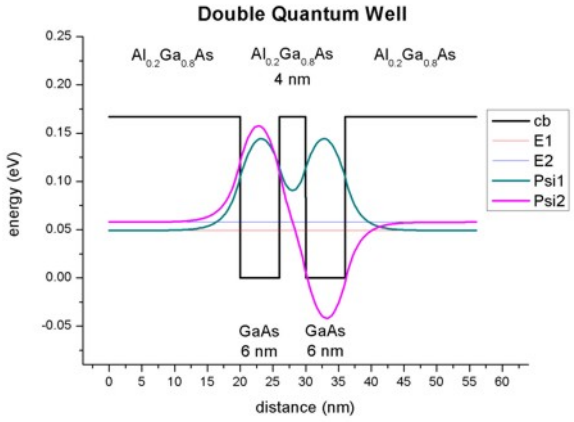
conduction band offset	$Al_{0.2}$ $Ga_{0.52}$ As / GaAs	0.167 eV
conduction band effective mass	$Al_{0.2}$ $Ga_{0.52}$ As	0.084 m_0
conduction band effective mass	GaAs	0.067 m_0

Results

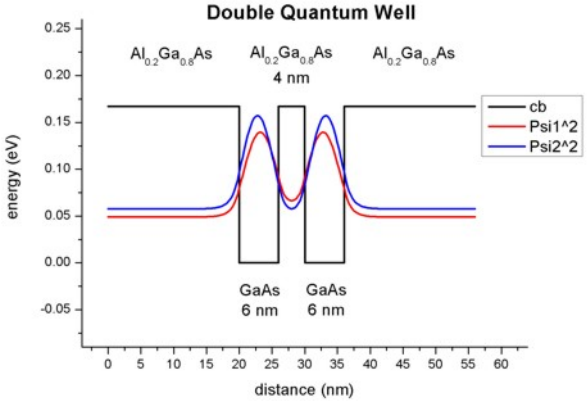
1. barrier width = 4 nm

- 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 (*nextnano++*)
band_structure/cb_Gamma.dat (*nextnano*^{3})

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 (*nextnano++*)
Schroedinger_1band/ev_cb1_sg1_deg1.dat (*nextnano*^{3})

These are the comparison of eigenvalues:

	nextnano++	*nextnano*^{3}	Harrison's book
ground state energy [eV]	0.04920	0.04920	0.04912
first excited state energy [eV]	0.05779	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 (*nextnano++*)
Schroedinger_1band/cb1_sg1_deg1_psi_shift.dat (*nextnano*^{3})

This file contains the eigenenergies and the squared wave functions (ψ^2):

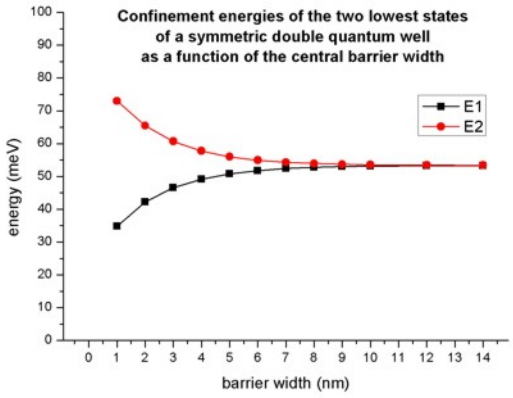
bias_00000/Quantum/wf_probabilities_shift_quantum_region_Gamma_0000.dat (*nextnano++*)
Schroedinger_1band/cb1_sg1_deg1_psi_squared_shift.dat (*nextnano*^{3})

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.

$$\psi_{\text{bonding}} = \frac{1}{\sqrt{2}}\psi_A + \psi_B \qquad \text{(lower energy)}$$
$$\psi_{\text{antibonding}} = \frac{1}{\sqrt{2}}\psi_A - \psi_B \qquad \text{(higher energy)}$$

- 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 (*nextnano++*)
Schredinger_1band/ev_cb1_sg1_deg1.dat (*nextnano*^{3})

For example, the values for the 1 nm barrier read:

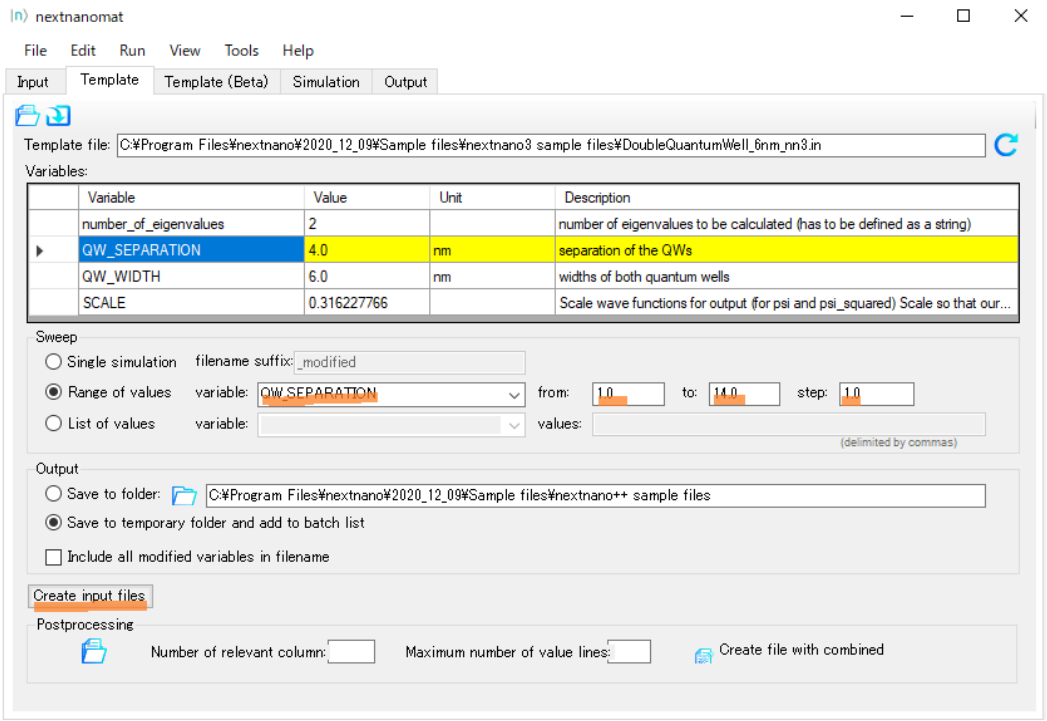
	nextnano++	*nextnano*^{3}	Harrison's book
ground state energy [eV]	0.03476	0.03476	0.03470
first excited state energy [eV]	0.07298	0.07298	0.07290

The values for the 14 nm barrier read:

	nextnano++	*nextnano*^{3}	Harrison's book
ground state energy [eV]	0.05332	0.05332	0.05323
first excited state energy [eV]	0.05338	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.



Another tutorial on coupled quantum wells can be found here .

Last update: nn/nn/nnnn

— 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

$$\phi_C(r) = \frac{1}{4\pi\varepsilon_0}\frac{q}{r}, \tag{6.4.5.2}$$

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

$$\left[-\frac{\hbar^2}{2m_0}\nabla^2 - q\,\phi_C(x,y,z)\right]\Psi(x,y,z) = E\,\Psi(x,y,z), \tag{6.4.5.3}$$

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's 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 }
```

(continues on next page)

(continued from previous page)

```
        }
      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 (6.4.5.2) introducing a regularizing parameter η aiming at removing the infinity.

$$\phi_C(r) \to \phi_\eta(r) = \frac{1}{4\pi\varepsilon_0} \frac{q}{\sqrt{x^2 + y^2 + z^2 + \eta^2}} \tag{6.4.5.4}$$

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's 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

$$\begin{aligned}\Psi_{1s}(r) &= 2\left(\frac{1}{a_B}\right)^{3/2}\exp\left(-\frac{r}{a_B}\right)\left(\frac{1}{4\pi}\right)^{1/2},\\ \Psi_{2s}(r) &= 2\left(\frac{1}{2a_B}\right)^{3/2}\left(1-\frac{r}{2a_B}\right)\exp\left(-\frac{r}{2a_B}\right)\left(\frac{1}{4\pi}\right)^{1/2}.\end{aligned} \tag{6.4.5.5}$$

Such comparison of s1 and s2 orbitals obtained with both methods is shown in Figure 6.4.5.6 a) and b), respectively.

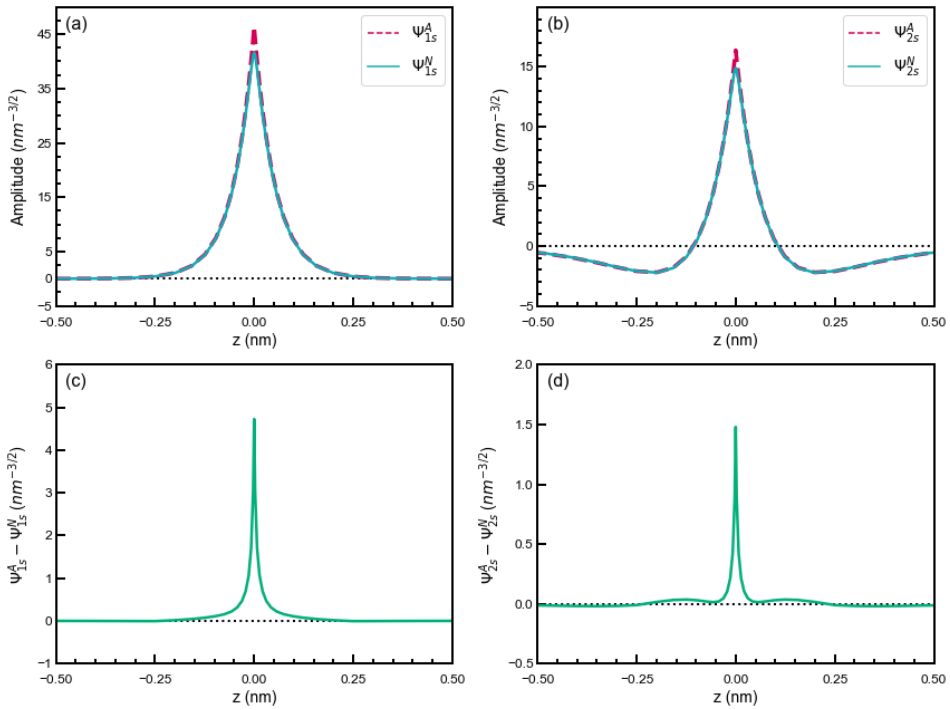


Figure 6.4.5.6: a), b) Comparison of 1s and 2s orbitals obtained from analytical formulas and numerical simulation. c), d) Difference between the analytical and numerical wave functions of s1 and s2 orbitals, respectively.

As seen in the Figure 6.4.5.6 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 6.4.5.7) 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.

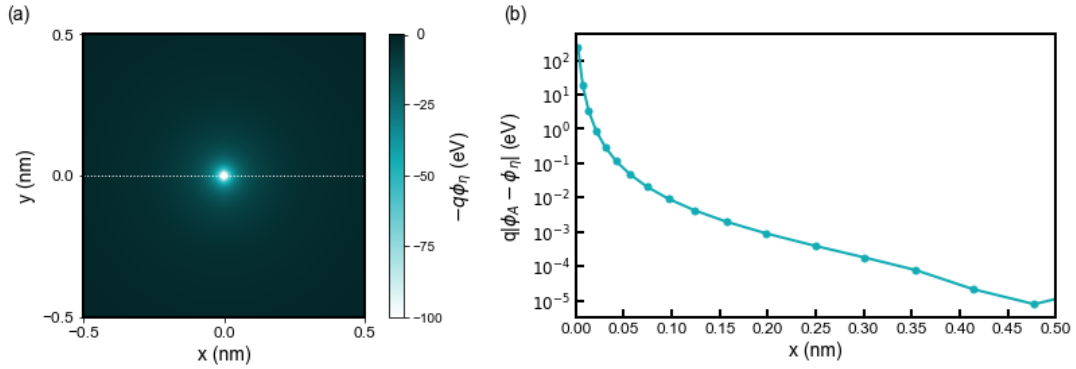


Figure 6.4.5.7: (a) shows the Coulomb energy distribution. (b) is the difference between the numerical Coulomb potential (V_η) calculated by *nextnano++* and the analytical potential (V_A) at $y = 0$, $z = 0$ along the white dash line in (a).

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]*

$$E_a = \frac{-m_0 q^4}{2(4\pi\varepsilon_0)^2\hbar^2}\frac{1}{n^2}, \tag{6.4.5.6}$$

where n is the principal quantum number, with computation using a fine grid (`$grid_medium=0.01`); see columns 4-6 in Table 6.4.5.1. 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 6.4.5.1: 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 6.4.5.1 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 6.4.5.8.

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 6.4.5.8 and the Table 6.4.5.1.

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

6.4.6 Quantum Wells

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*

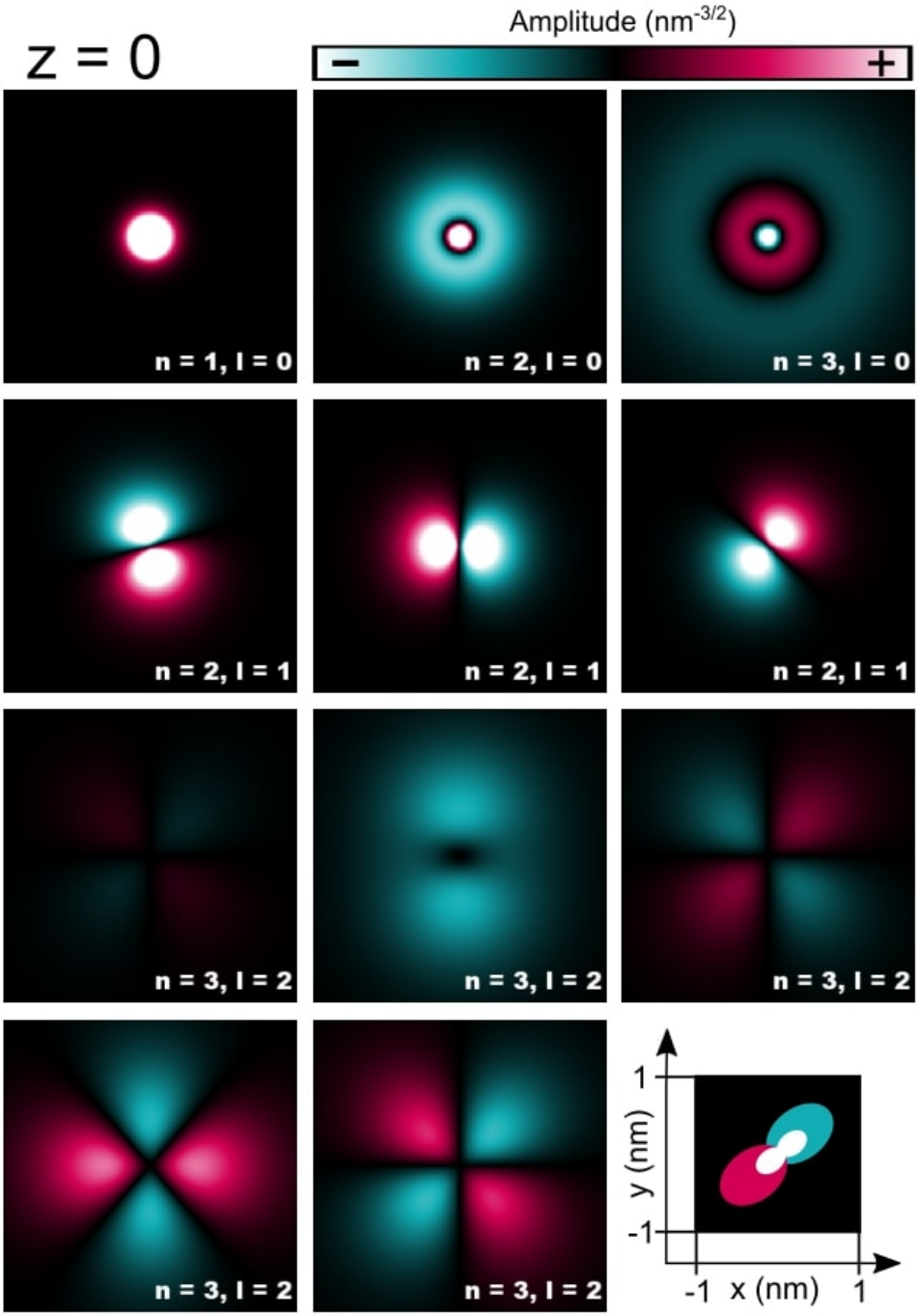


Figure 6.4.5.8: Cross-sections of 1s, 2s, 3s, 2p, and 3d orbitals computed for the Hydrogen atom.

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*

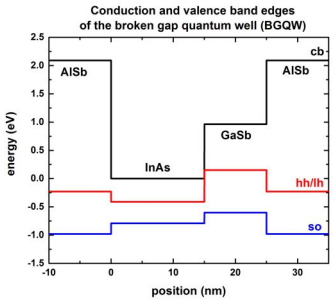
Material parameters used are taken from *Optical transitions in broken gap heterostructures*.

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.`

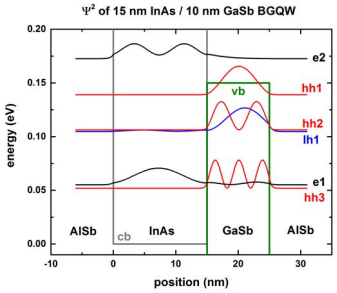
One can clearly see that the electron state (**e1**, **e2**) are confined in the InAs layer (left part of the figure), whereas the heavy (hh1, hh2, hh3) and light hole (lh1) states are confined in the GaSb layer (right part of the figure). One can see a slight hybridization of the **e1** and lh1 states, 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.

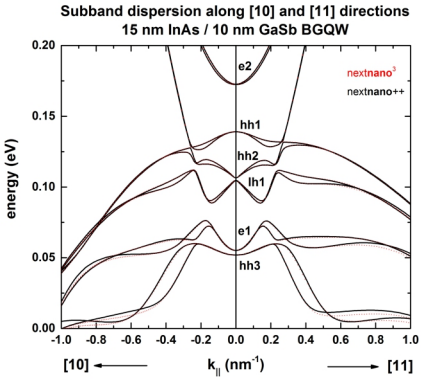
`Schroedinger_kp/kpar1D_disp_01_00el_8x8kp_ev_min001_ev_max020.dat` contains the $k_{||}$ dispersion from [00] to [01] because in the input file, it is specified that

```
dispersion{
    path{
        name = "kpar_01_00_10"
        point{ k = [0.0,  0.0,  1.0] }
        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: $E_n(k_{||}) = E_n(k_x, k_y) = E_n(0, k_y)$. 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.

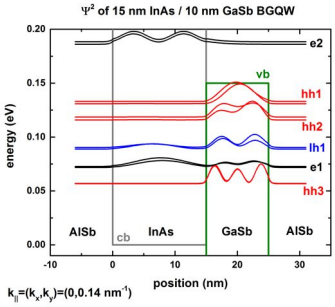
In asymmetric quantum wells, the double spin degeneracy is lifted at finite values of $\mathbf{k}_{||}$ because of spin-orbit interaction. This is the reason why we have two different dispersions $E(\mathbf{k}_{||})$ for "spin up" and "spin down" states. This also means that the wave functions at finite $\mathbf{k}_{||}$ 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 + 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.

Last update: nn/nn/nnnn

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 6.4.6.1 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.

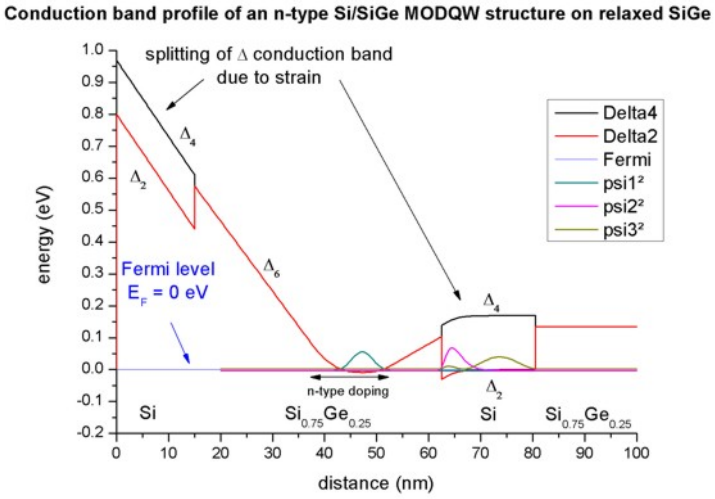


Figure 6.4.6.1: Calculated conduction band edge profile.

Figure 6.4.6.2 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.

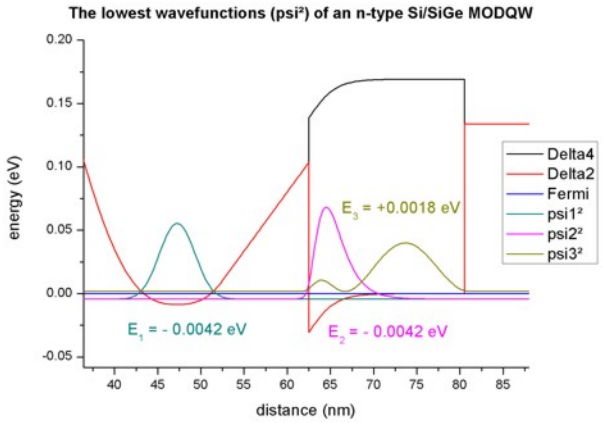


Figure 6.4.6.2: Calculated probability densities of the lowest electron states.

The electron density (in units of $1 \cdot 10^{18}$ cm^{-3}) is plotted in Figure 6.4.6.3. 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}.

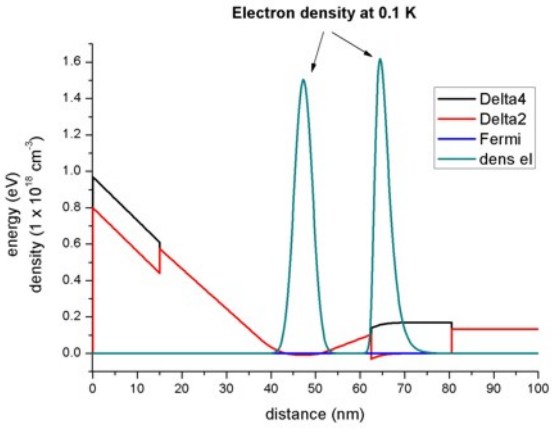


Figure 6.4.6.3: Calculated electron density profile.

Last update: nn/nn/nnnn

Exciton Binding Energy in an Infinite Quantum Well

Input File:
1D_exciton_binding_energy_infinite_QW_nnp.in 1D_exciton_binding_energy_infinite_QW_nn3.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

$$E_{\text{ex,b}} = -\frac{\mu e^4}{32\pi^2\hbar^2 e_{\text{r}}^2 e_0^2} = -\frac{\mu}{m_0 e_{\text{r}}^2} \cdot 13.61 \text{ eV},$$

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 ($e_{\text{r,GaAs}}$ = 12.93, $e_{\text{r,CdTe}}$ = 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

$$\frac{1}{\mu} = \frac{1}{m_{\text{e}}} + \frac{1}{m_{\text{h}}},$$

with the effective masses of electrons and holes: $\mathrm{m_e}$ and $\mathrm{m_h}$.

Example

The *reduced mass* of GaAs and CdTe are

$$\frac{1}{\mu_{\text{GaAs}}} = \frac{1}{0.067} + \frac{1}{0.5} \quad \Rightarrow \quad \mu_{\text{GaAs}} = 0.0591$$

$$\frac{1}{\mu_{\text{CdTe}}} = \frac{1}{0.096} + \frac{1}{0.6} \quad \Rightarrow \quad \mu_{\text{CdTe}} = 0.0828$$

with respective *Bohr radius*

$$\lambda_{\text{GaAs}} = 11.6 \text{ nm}$$

$$\lambda_{\text{CdTe}} = 6.8 \text{ nm}.$$

From the *3D bulk exciton binding energies*

$$E_{\text{ex,b}}(\text{GaAs}) = -4.8 \text{ meV}$$

$$E_{\text{ex,b}}(\text{CdTe}) = -10.0 \text{ meV}$$

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 (e_1–hh_1) of a 1D quantum well (type-I) are only obtainable in the following two limits:

- infinitely thin quantum well (*2D limit*)

$$E_{\mathrm{ex,QW}} = 4 \cdot E_{\mathrm{ex}}$$
$$\lambda_{\mathrm{ex,QW}} = \frac{\lambda_{\mathrm{ex}}}{2}$$

- infinitely thick quantum well (*3D bulk exciton limit*)

$$E_{\mathrm{ex,QW}} = E_{\mathrm{ex}}$$
$$\lambda_{\mathrm{ex,QW}} = \lambda_{\mathrm{ex}}$$

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. e_2–hh_2, e_1–lh_1).

Numerical calculation

Our numerical approach is based on a variational principle. We use the separable wave function

$$\psi(r) = \sqrt{\frac{2}{\pi}} \frac{1}{\lambda} \exp\left(-\frac{r}{\lambda}\right), \tag{6.4.6.1}$$

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 6.4.6.4) 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.

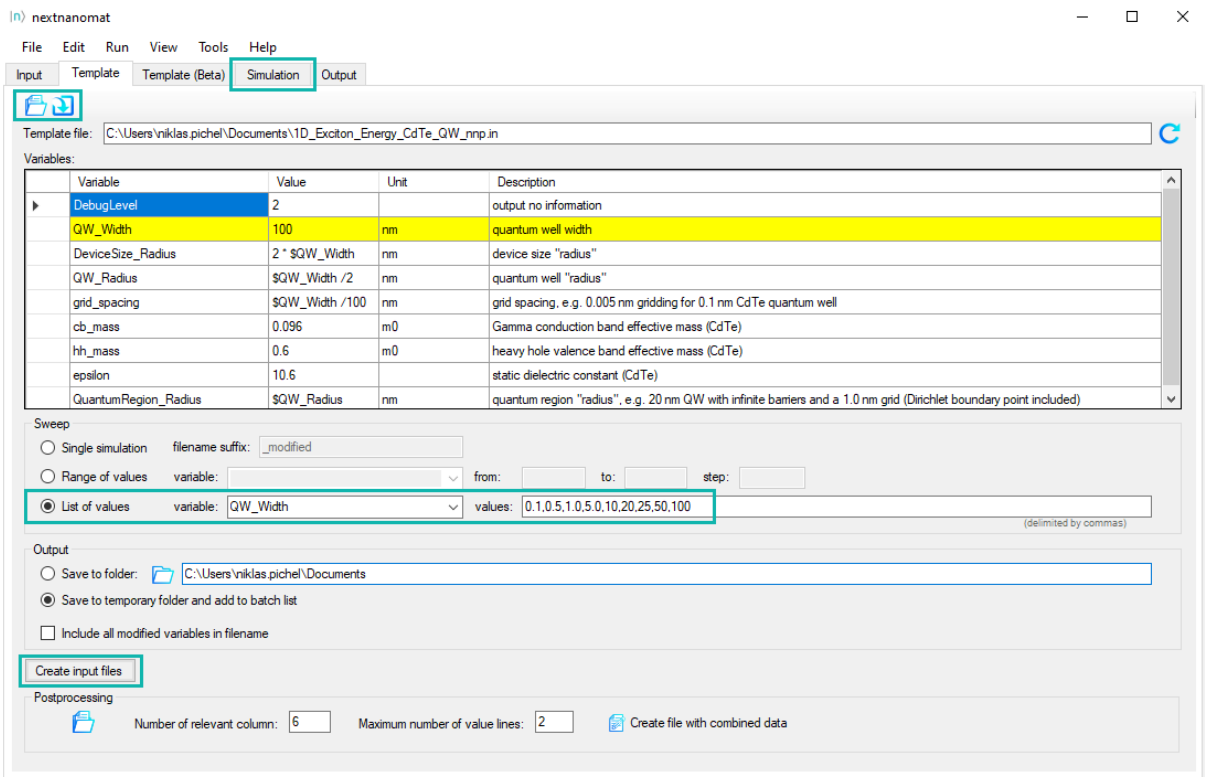


Figure 6.4.6.4: Initializing parameter sweep for QW_Width in tab Template

Initialization and execution of parameter sweep:

- 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 6.4.6.5 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 $\psi(r, z_{\mathrm{e}}, z_{\mathrm{h}})$ has to be used instead of (6.4.6.1).

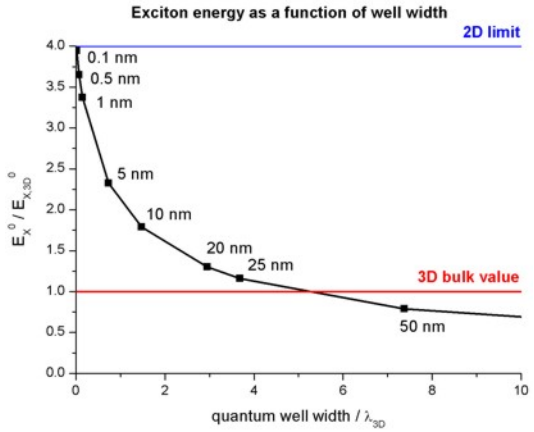


Figure 6.4.6.5: Exciton energy as a function of quantum well width

Figure 6.4.6.6 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.

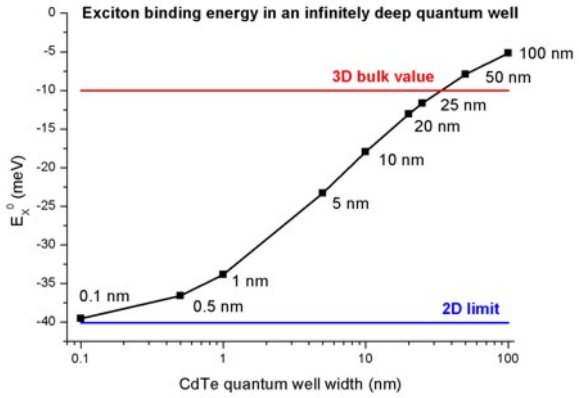


Figure 6.4.6.6: Exciton binding energy in an infinitely deep quantum well

Figure 6.4.6.7 shows the exciton Bohr radius a function of well width

The nextnano³ tool

The following comments are specific to the input file syntax of *nextnano³*.

The material parameters used are the following (*$binary-zb-default*).

```
!-----------------------------------------------------------!
! Here we are overwriting the database entries for CdTe.    !
!-----------------------------------------------------------!
$binary-zb-default                                          !
 binary-type                 = CdTe-zb-default              !
 apply-to-material-numbers   = 2                            !
```

(continues on next page)

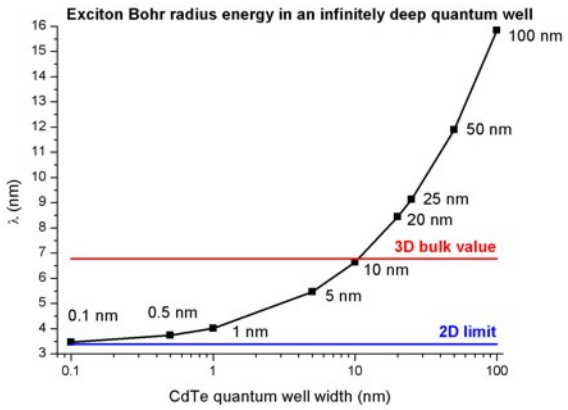


Figure 6.4.6.7: Exciton Bohr radius energy in an infinitely deep quantum well

(continued from previous page)

```
conduction-band-masses      = 0.096    0.096    0.096    ! Gamma      [m0]
                              ...                        !
                                                         !
valence-band-masses         = 0.6      0.6      0.6      ! heavy hole [m0]
                              ...                        !
static-dielectric-constants = 10.6     10.6     10.6     ! []
```

In order to calculate the exciton correction, the following flags have to be used:

```
$numeric-control
 simulation-dimension          = 1
 calculate-exciton             = yes   ! to switch on exciton correction
 exciton-electron-state-number = 1     ! electron ground state
 exciton-hole-state-number     = 1     ! hole     ground state
```

The output of the exciton binding energies can be found in this file: `Schroedinger_1band/exciton_energy1D.dat`

The output for the 5 nm CdTe QW looks like this:

```
Exciton correction for 1D quantum wells (type-I structures)
============================================================
static dielectric constant     =  10.6000000000 []
effective mass electron        =   0.0960000000 [m0]
effective mass hole            =   0.6000000000 [m0]
reduced mass                   =   0.0827586207 [m0]
Bulk Bohr radius of 3D exciton =   6.7778780735 [nm]
Bulk 3D exciton energy         = -10.0212560410 [meV]

lambda [nm]        exciton energy [meV]   exciton energy [Rydberg]
0.338893904E+001   -0.158496790E+002      0.158160603E+001
...
0.421888329E+001   -0.215591082E+002      0.215133793E+001
...
0.553296169E+001   -0.232757580E+002      0.232263879E+001
------------------------------------------------------------------
------------------------------------------------------------------
Calculated lambda and exciton energy:
0.546379967E+001   -0.232817837E+002      0.232324009E+001
------------------------------------------------------------------
```

The last iteration yields -23.28 meV for the exciton binding energy. `lambda` is the variational parameter λ which is equivalent to the exciton Bohr radius in units of `[nm]`.

Last update: nn/nn/nnnn

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{
        ...
        intraband_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{}
        }
```

(continues on next page)

(continued from previous page)

```
        lifetimes{ # calculate lifetimes
            Gamma{}
            phonon_energy = 0.036  # [eV]
        }
    }
}
```

The following two figures (Figure 6.4.6.8, Figure 6.4.6.9) 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.

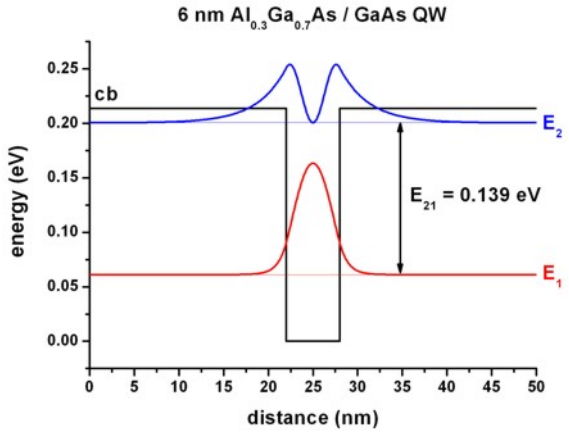


Figure 6.4.6.8: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1 and E_2)

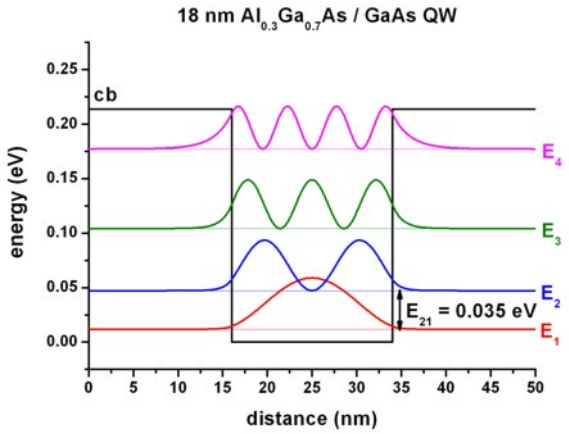


Figure 6.4.6.9: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1, E_2, E_3 and E_4).

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 6.4.6.10 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 *nextnano*[3] calculations are in good agreement with Fig. 3 of [FerreiraBastard1989].

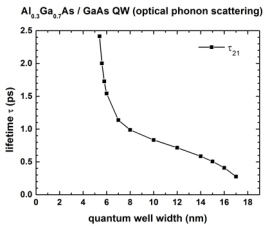


Figure 6.4.6.10: Calculated lifetimes τ as a function of quantum well width

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 6.4.6.11 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 *nextnano*[3] calculations presented in Figure 6.4.6.12 are in reasonable agreement with Fig. 5 in [FerreiraBastard1989].

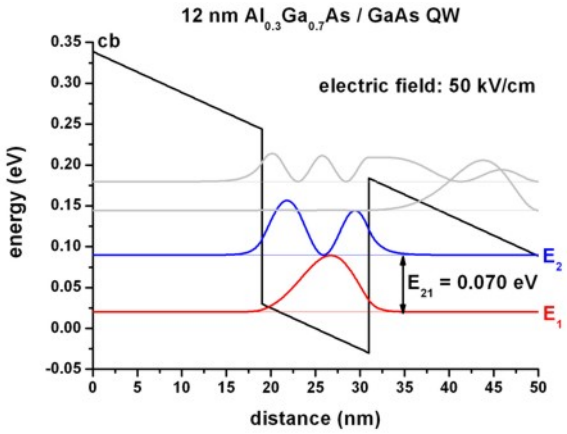


Figure 6.4.6.11: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1 and E_2), when electric field of -50 kV/cm is applied.

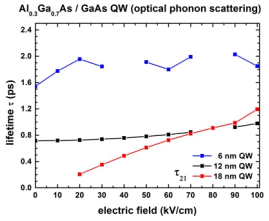


Figure 6.4.6.12: Calculated lifetimes τ in a single quantum well as a function of applied electric field.

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 6.4.6.13. The energy of the ground state is 50.4 meV.

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 6.4.6.14 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.

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:

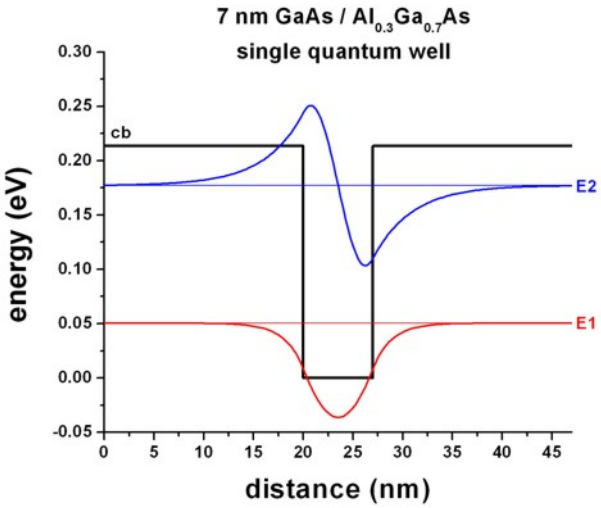


Figure 6.4.6.13: Calculated conduction band edge profile (black) and wave functions of confined electron states (E_1 and E_2)

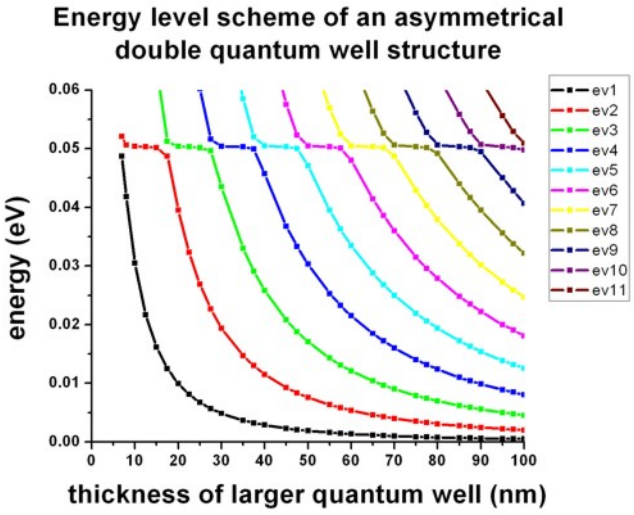


Figure 6.4.6.14: Calculated energy levels for different energy states as a function of $L_{\mathrm{QW,right}}$.

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 6.4.6.14) can be illustrated by plotting the wave functions for different values of $L_{\mathrm{QW,right}}$, see Figure 6.4.6.15, Figure 6.4.6.16, Figure 6.4.6.17 and Figure 6.4.6.18.

Figure 6.4.6.15 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.

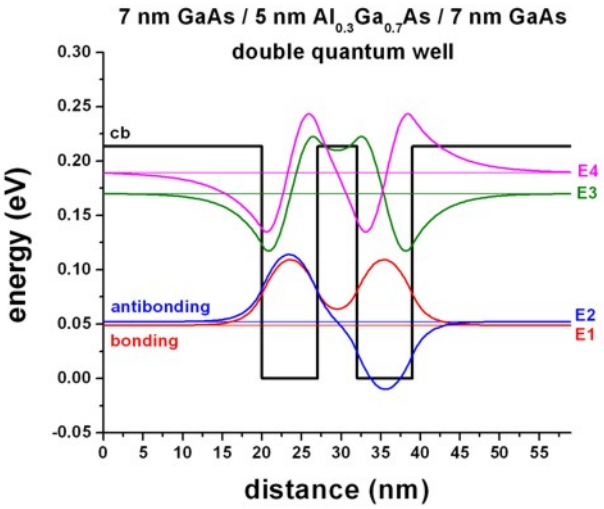


Figure 6.4.6.15: Calculated conduction band edge profile (black) for symmetric double quantum well: L_{left} = 7 nm and L_{right} = 7 nm, with wave functions of confined electron states.

Figure 6.4.6.16 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.

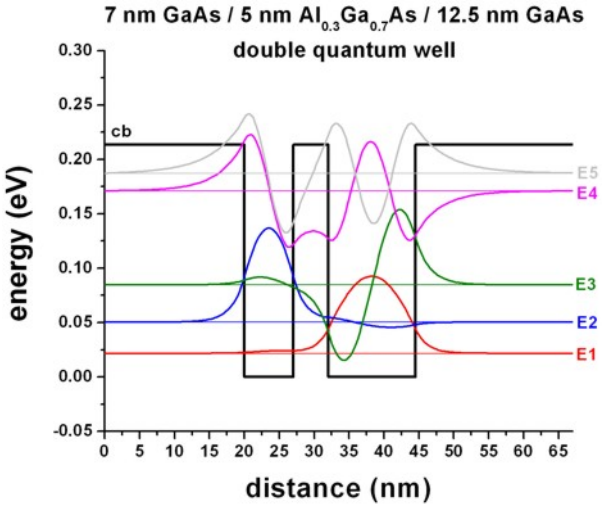


Figure 6.4.6.16: Calculated conduction band edge profile (black) for non-symmetric double quantum well: L_{left} = 7 nm and L_{right} = 12.5 nm, with wave functions of confined electron states.

Figure 6.4.6.17 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 6.4.6.16), but this time, the third state of moves

down in energy and couples to the 7 nm ground state of the left well (compare with Figure 6.4.6.16). This coupling leads to the formation of a bonding and an anti-bonding states.

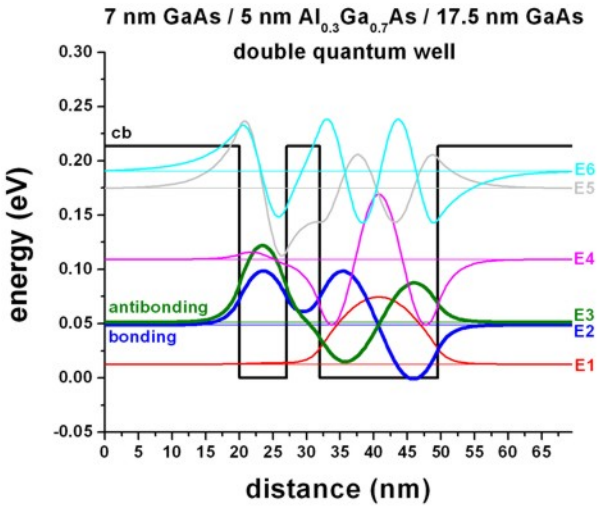


Figure 6.4.6.17: Calculated conduction band edge profile (black) for non-symmetric double quantum well: L_{left} = 7 nm and L_{right} = 17.5 nm, with wave functions of confined electron states.

Figure 6.4.6.18 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

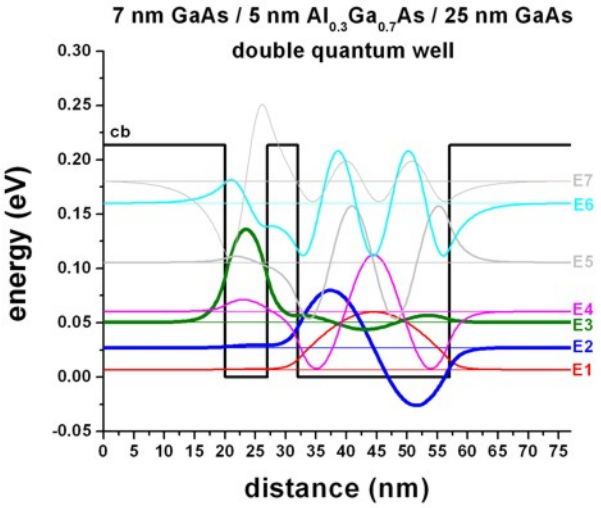


Figure 6.4.6.18: Calculated conduction band edge profile (black) for symmetric double quantum well: L_{left} = 7 nm and L_{right} = 25 nm, with wave functions of confined electron states.

Biased double quantum well

Input file: *1DGaAs_AlGaAs_DoubleQW_LifetimeFig12_field.in*

Figure 6.4.6.19 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.

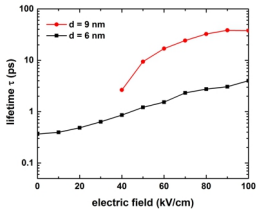


Figure 6.4.6.19: Calculated lifetimes τ in a single quantum well as a function of applied electric field.

Last update: nn/nn/nnnn

— 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 6.4.6.20 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).

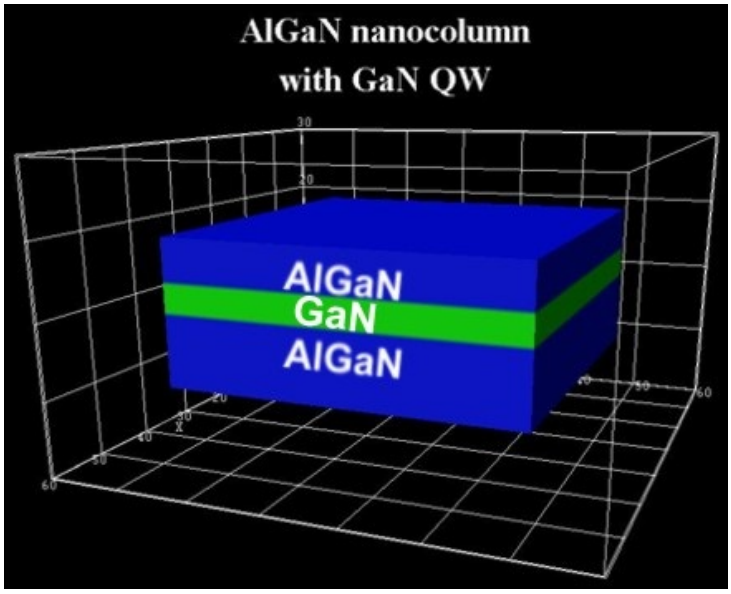


Figure 6.4.6.20: Simulated heterostructure consisting of a GaN (green) layer sandwiched in between two AlGaN (blue) layers.

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 6.4.6.21 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 6.4.6.22 shows the strain tensor component ϵ_{yy}. Similar to Figure 6.4.6.21, 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).

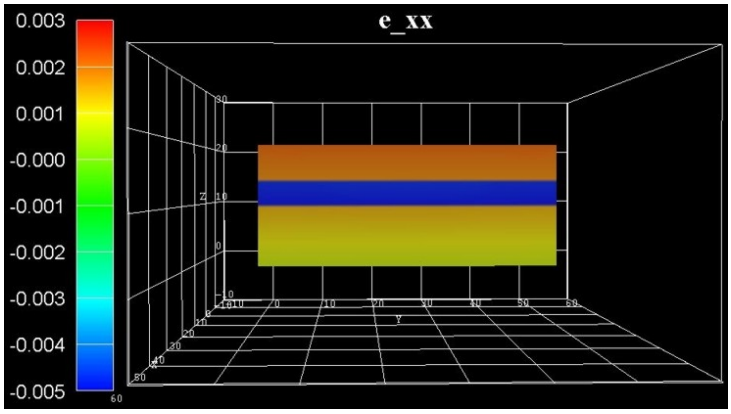


Figure 6.4.6.21: Calculated strain component ϵ_{xx} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

- Below and above the QW boundaries (red regions), the AlGaN takes on the lattice constant of ~GaN (tensile strain).

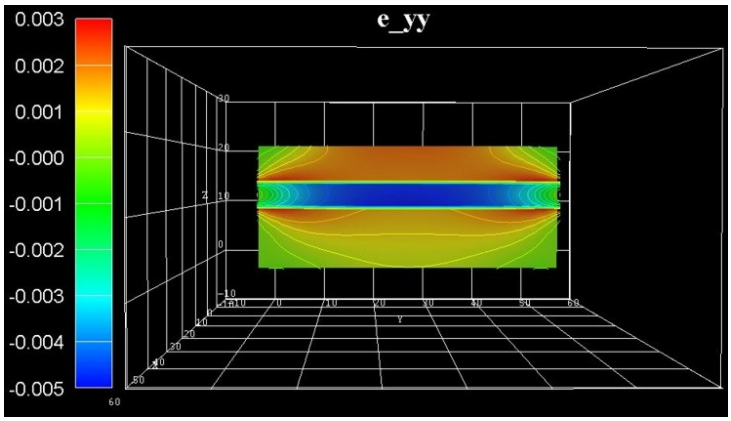


Figure 6.4.6.22: Calculated strain component ϵ_{yy} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

Figure 6.4.6.23 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 6.4.6.24 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 6.4.6.25 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 6.4.6.26 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*

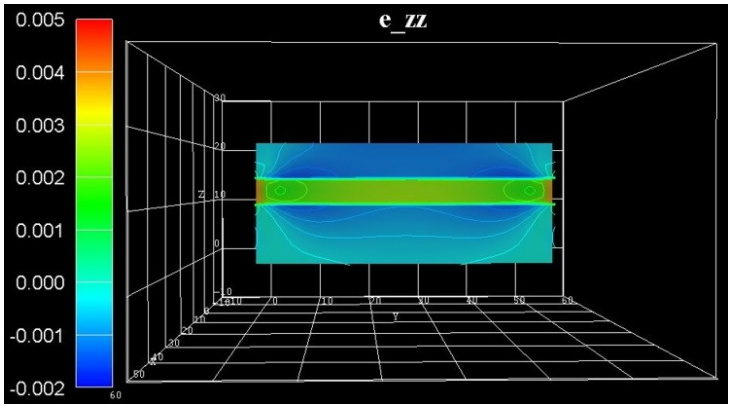


Figure 6.4.6.23: Calculated strain component ϵ_{zz} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

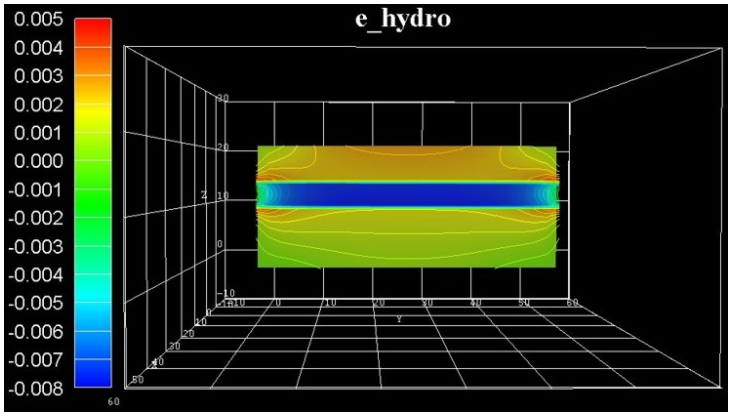


Figure 6.4.6.24: Calculated hydrostatic strain component $\epsilon_{\mathrm{hydro}} = \epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}$ in the y, z plane at $x = 25.0\,\mathrm{nm}$.

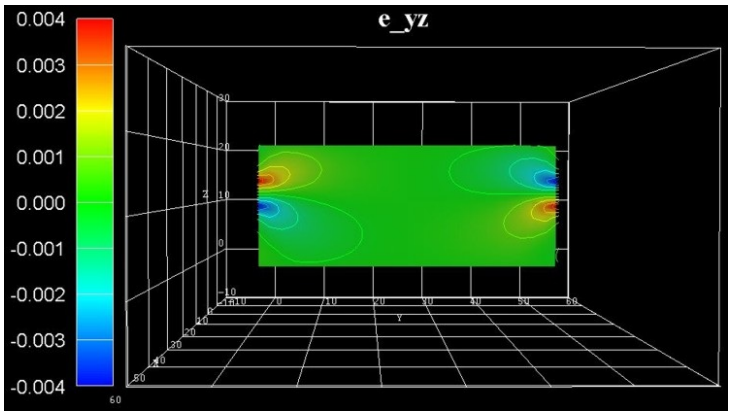


Figure 6.4.6.25: Calculated strain component ϵ_{yz} in the y, z plane at $x = 25.0\,\mathrm{nm}$.

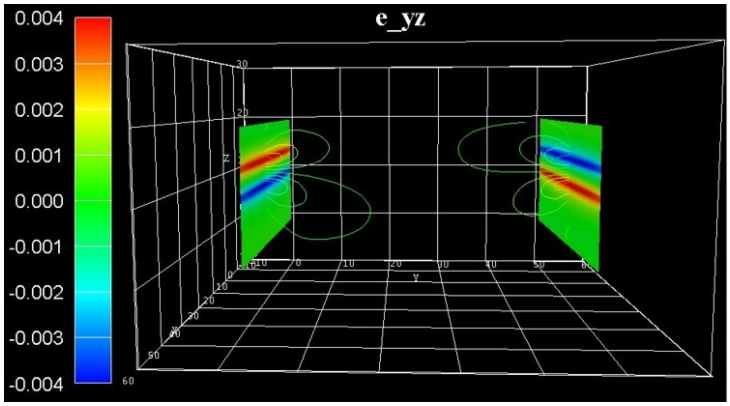


Figure 6.4.6.26: Calculated strain component ϵ_{yz} in the x, z plane at the boundaries.

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 6.4.6.27 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.

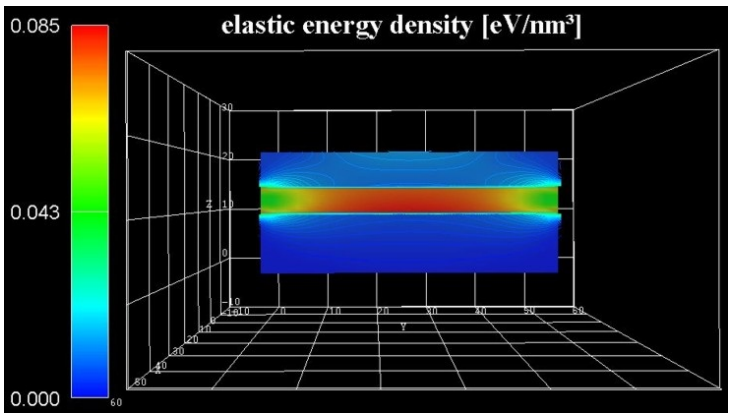


Figure 6.4.6.27: Elastic energy density in the x, z plane at $x = 25.0\,\mathrm{nm}$.

Last update: 17/07/2024

6.4.7 Quantum Wires

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 6.4.7.1 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.

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 6.4.7.2 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 6.4.7.3 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 surrounding material "air".

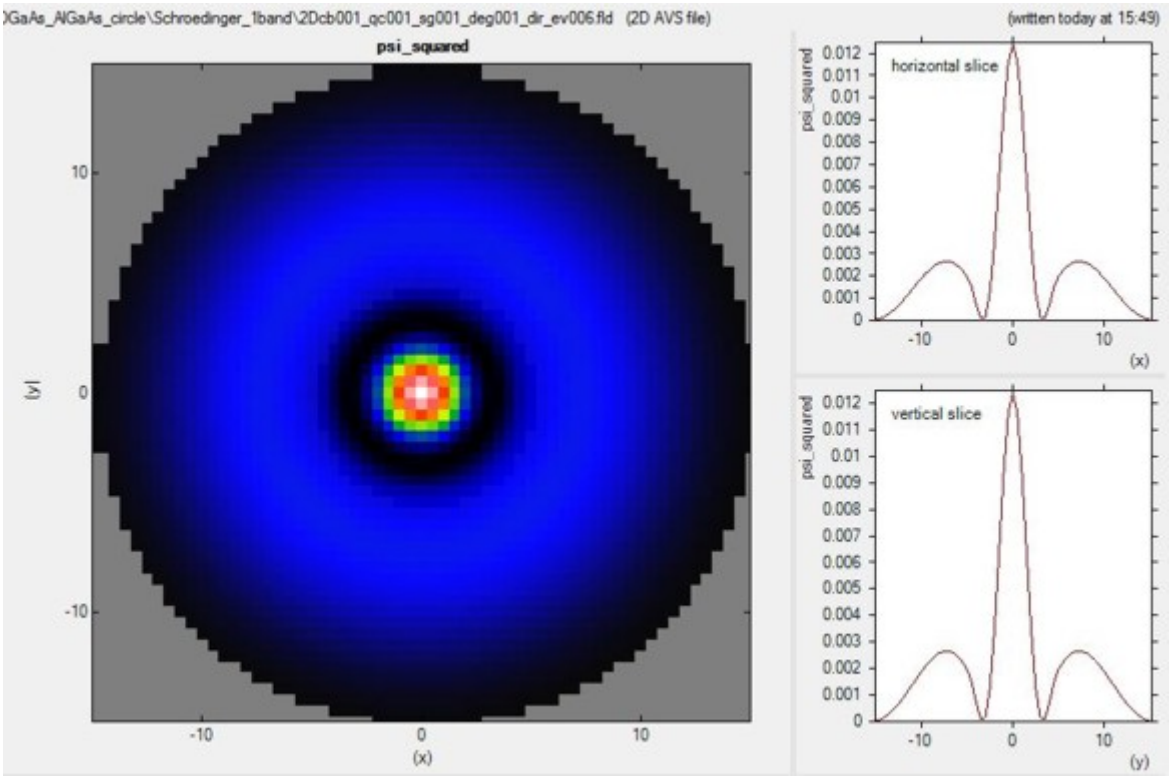


Figure 6.4.7.1: Ψ^2 of the 6th electron eigenstate.

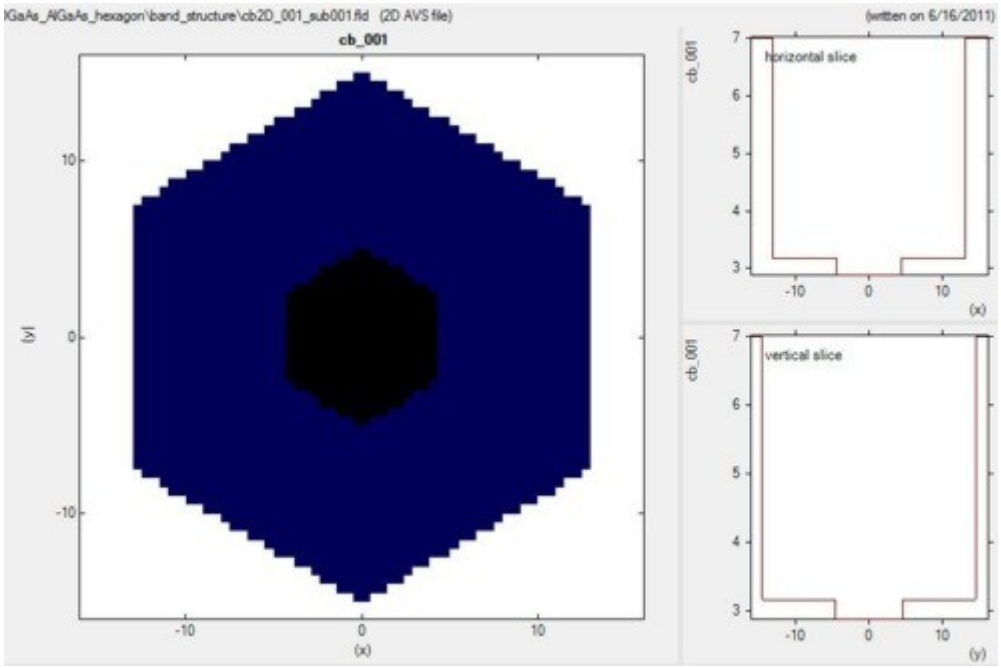


Figure 6.4.7.2: Conduction band edge profile of the hexagonal core-shell structure.

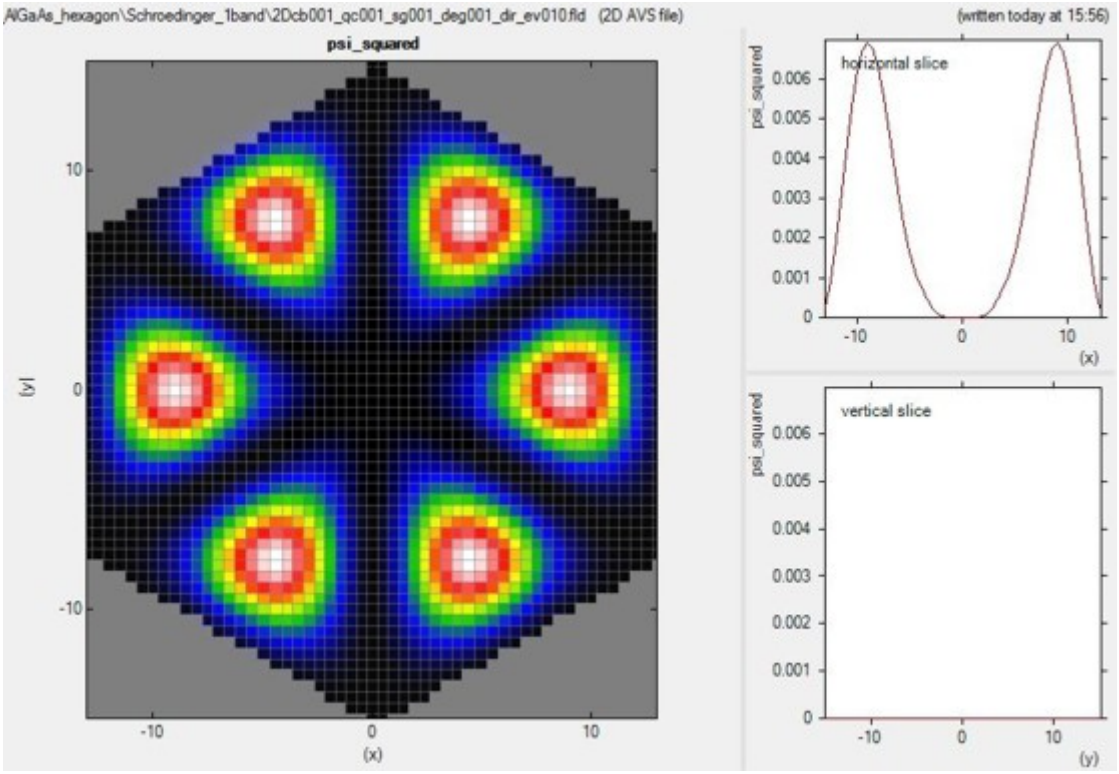


Figure 6.4.7.3: Ψ^2 of 10th electron eigenstate.

Alloy sweep

Note: In comparison with *nextnano*[3], *nextnano++* does not support an alloy sweep in the input file. However, you can use *nextnanomat*'s Template feature to perform an 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 6.4.7.4.

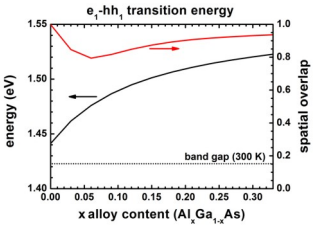


Figure 6.4.7.4: Transition energy and spatial overlap of e1 to h1 transition as a function of alloy content x.

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\intraband_matrix_elements_quantum_region_Gamma_001.txt*

```
alloy      type                          el-hl[eV]     el[eV]        hl[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 6.4.7.5.

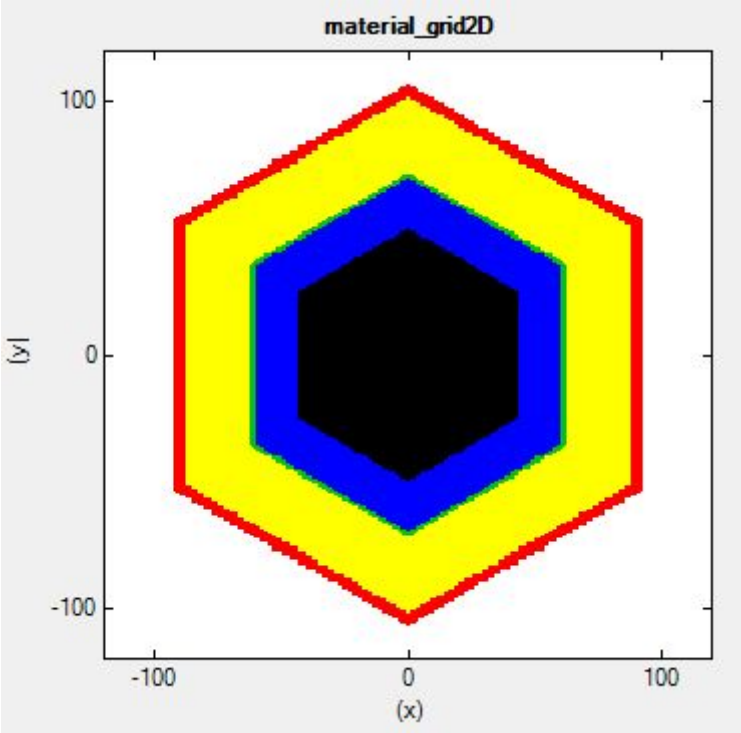


Figure 6.4.7.5: The material layers of the struture: $GaAs$ core (black), $AlGaAs$ spacer (blue), Si-doped $AlGaAs$ (green), $AlGaAs$ (yellow), $GaAs$ capping layer (red) and Schottky barrier contact (black) are shown. (The white layer itself is not included in the calculation. It only serves as a boundary condition)

The self-consistently calculated conduction band edge (*bandedges.fld*) is shown in Figure 6.4.7.6. 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 6.4.7.7. At the corners, the electron density is significantly higher, thus one-dimensional conducting channels are formed. Although the

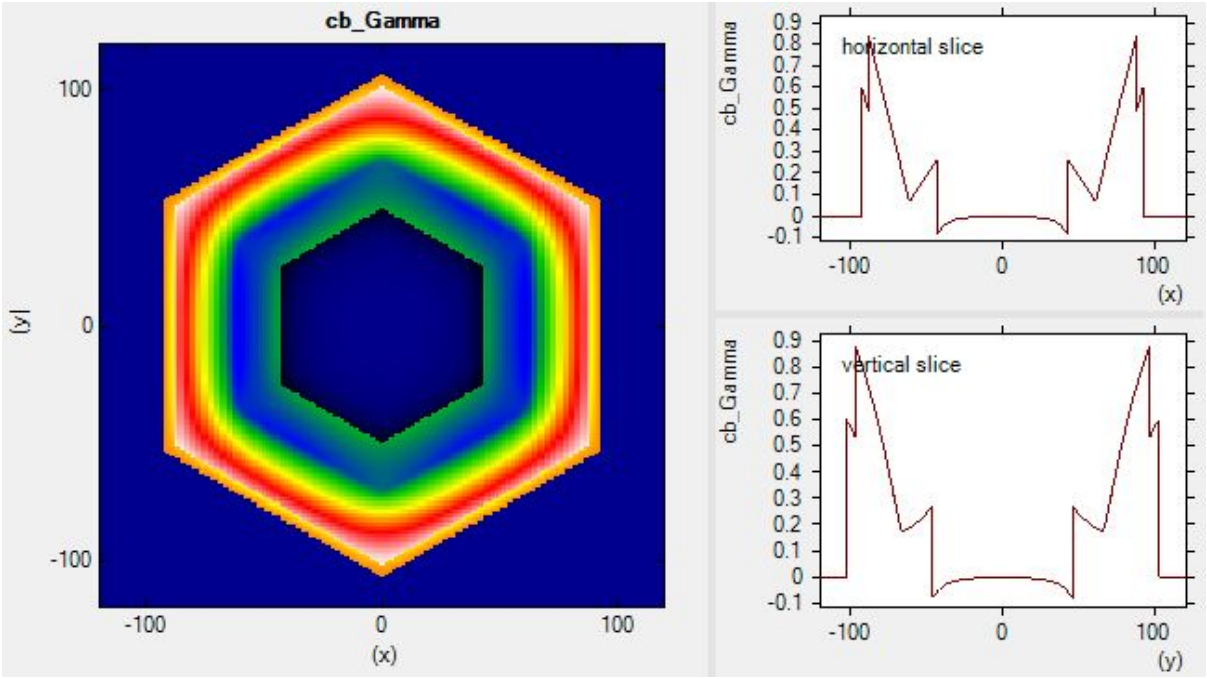


Figure 6.4.7.6: Conduction band edge profile.

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.

Last update: nn/nn/nnnn

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.

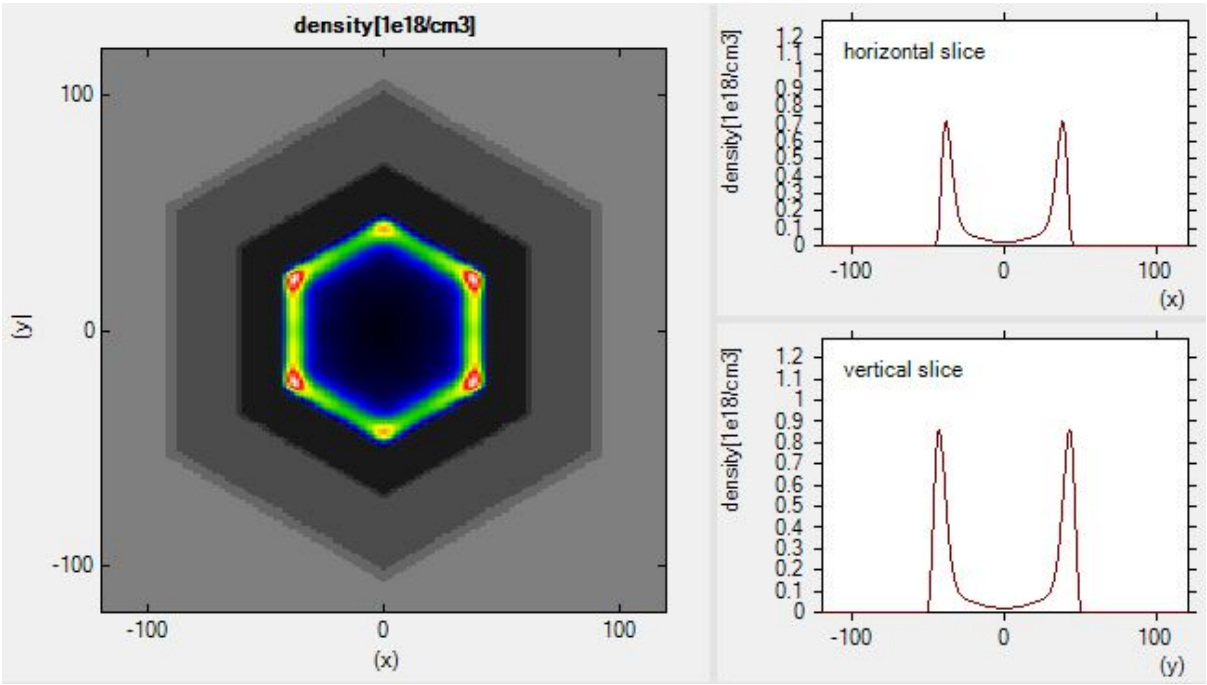
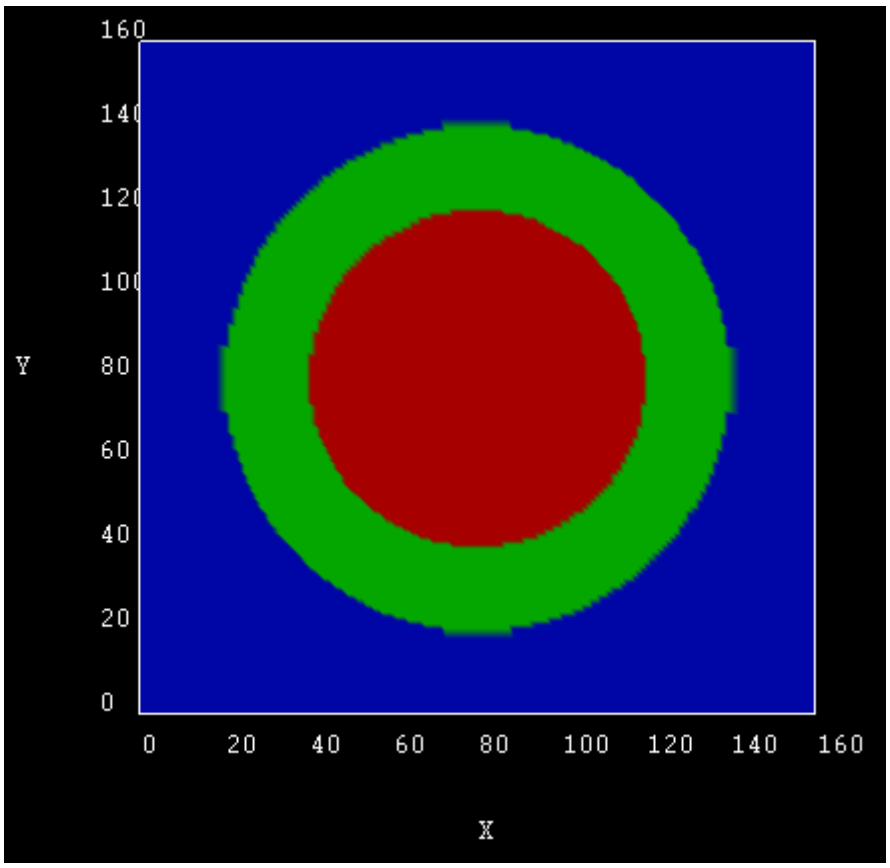


Figure 6.4.7.7: Charge density profile.



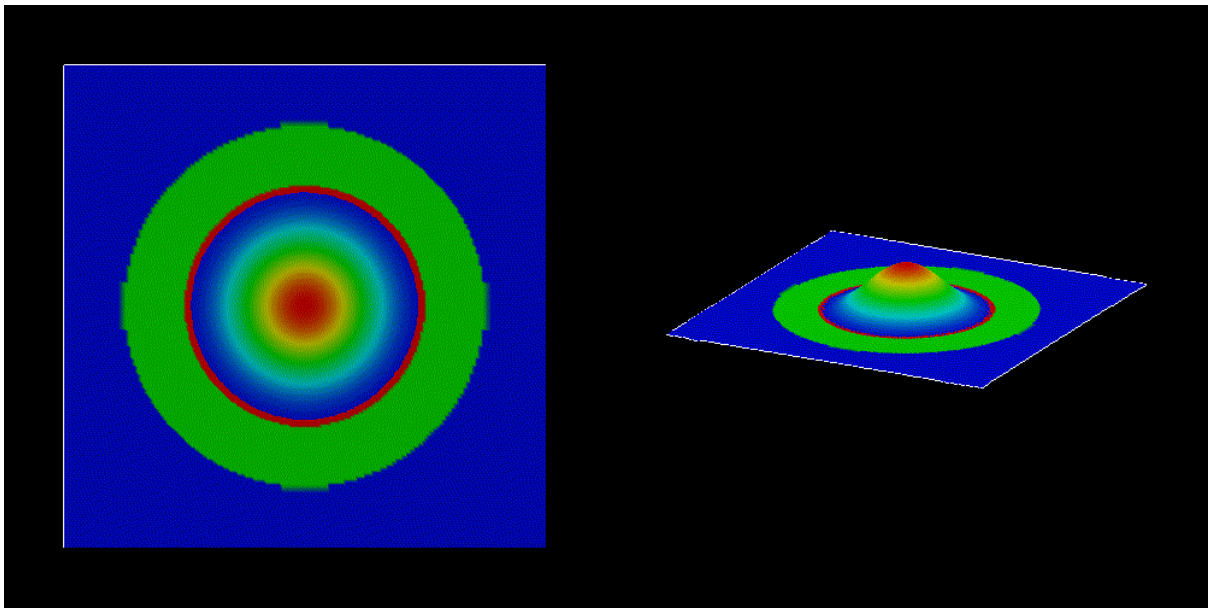
Simulation outcome

Electron wave functions

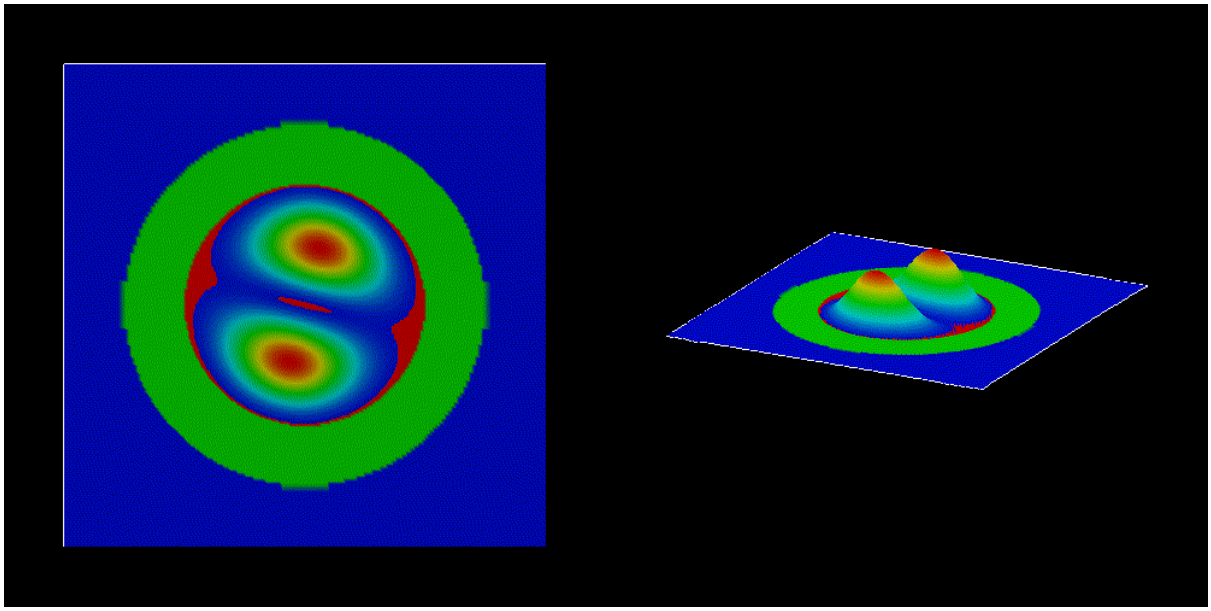
The size of the quantum cluster is a circle of diameter $2a = 80$ nm.

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.

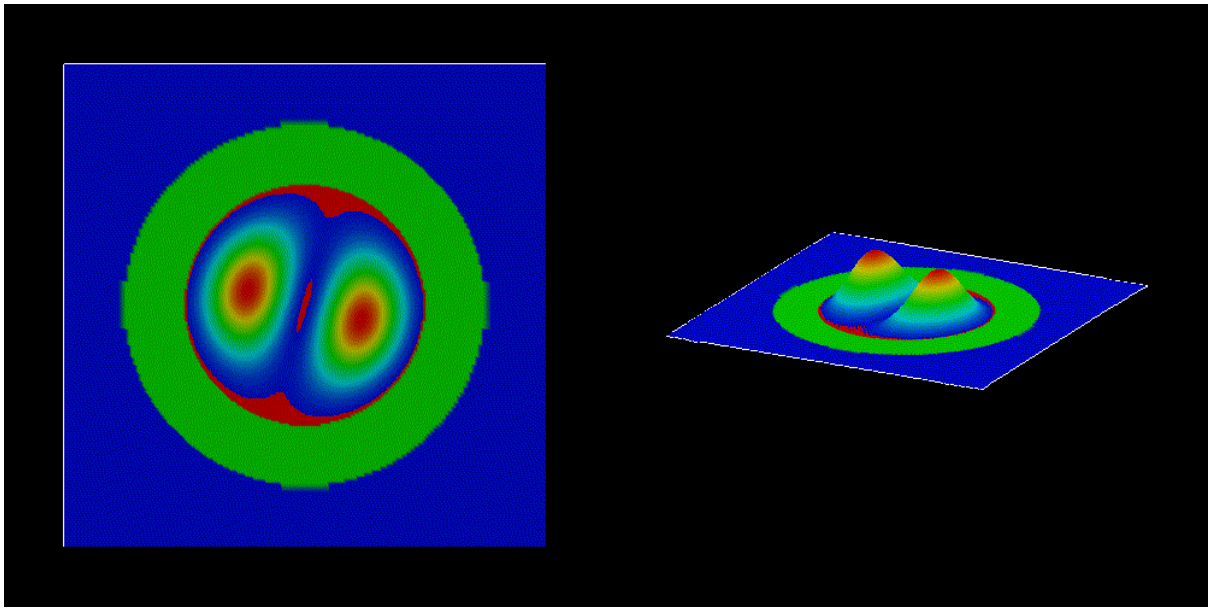
- 1st eigenstate, $(n,\ l) = (1,\ 0)$



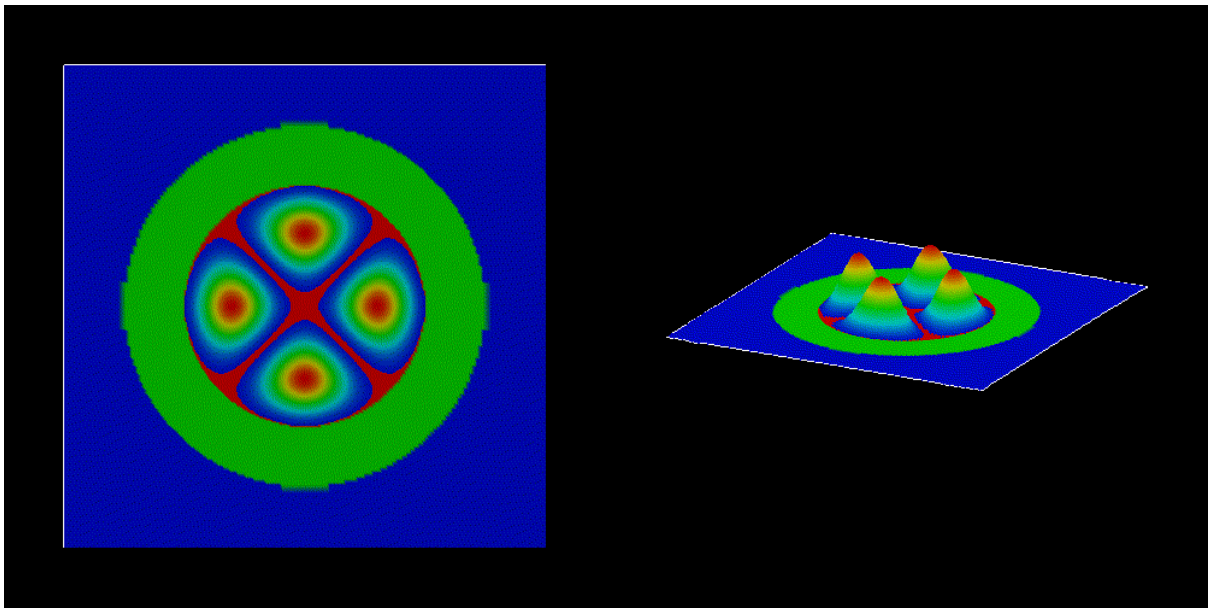
- 2nd eigenstate, $(n,\ l) = (1,\ 1)$



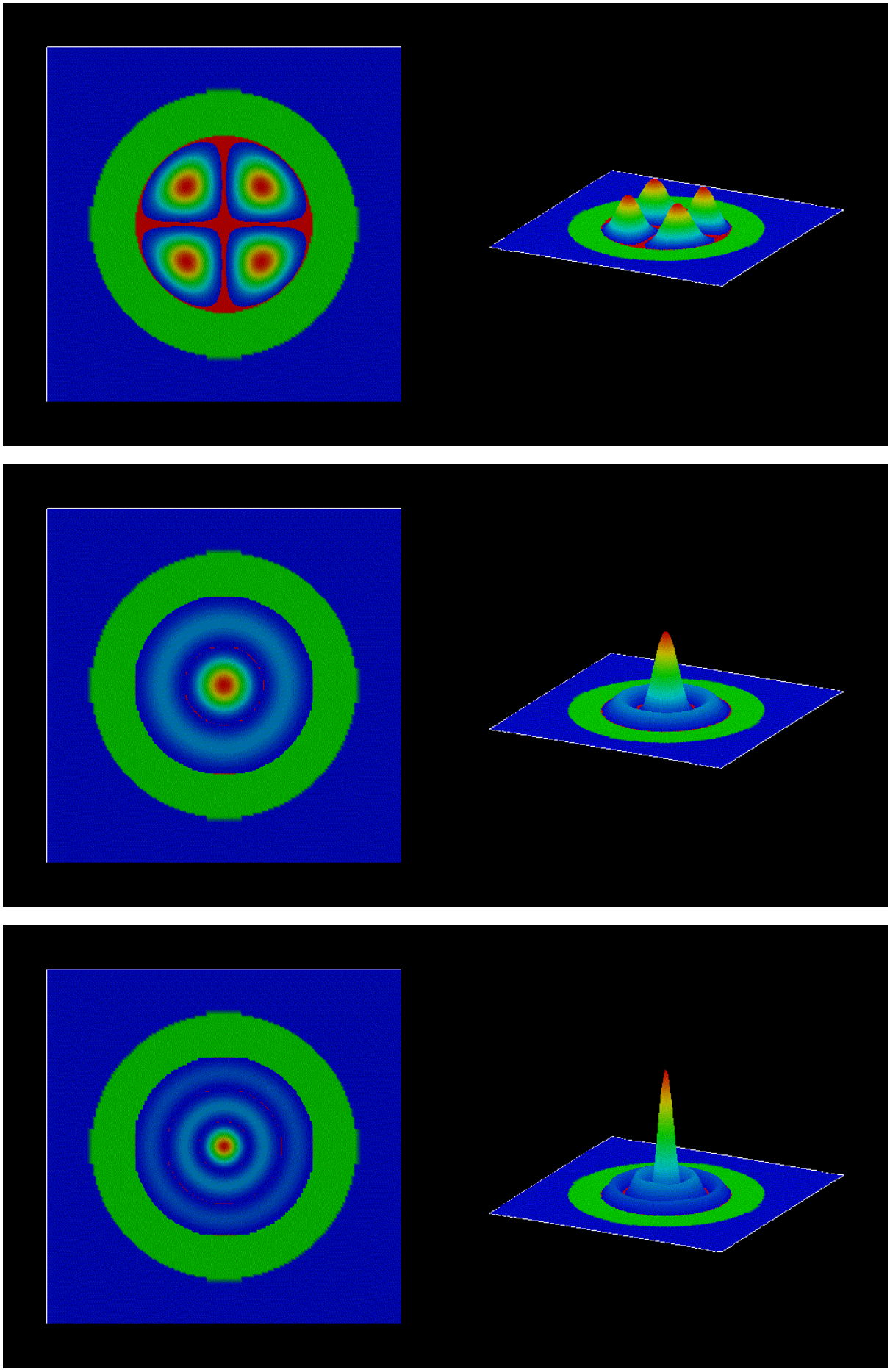
- 3rd eigenstate, $(n,\ l) = (1, -1)$

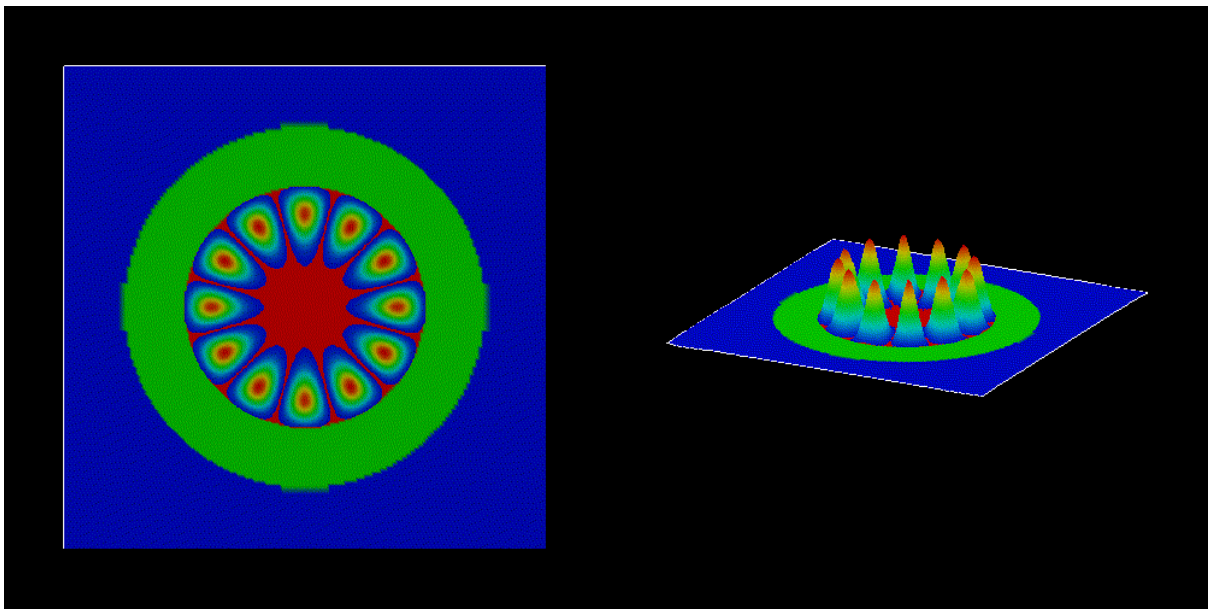


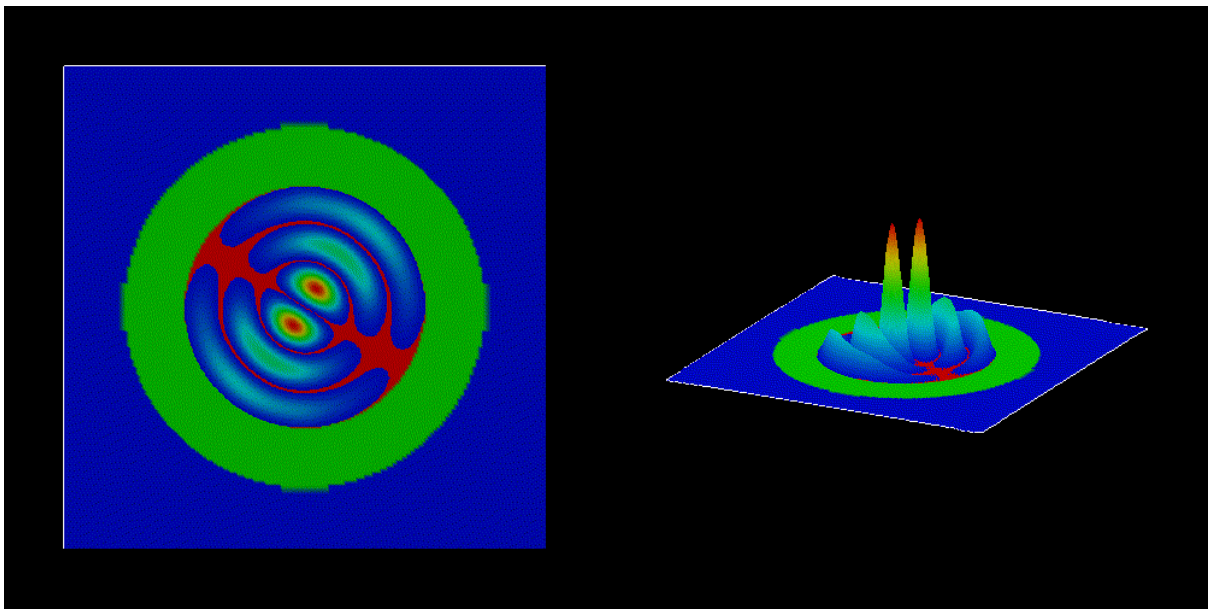
- 4th eigenstate, $(n,\ l) = (1,\ 2)$



- 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
- $m_e = 0.026 m_0$
- $V(r) = 0$ for $r < a$
- $V(r) = \infty$ for $r > a$

The analytical solution of the eigenstates of this quantum well is:

$$\psi_{n,l}(r,\theta) \propto J_l\left(\frac{j_{l,n} r}{a}\right)[A\cos(l\theta) + B\sin(l\theta)] \tag{6.4.7.1}$$

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)

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, *nextnano++* results and *nextnano*[3] results.

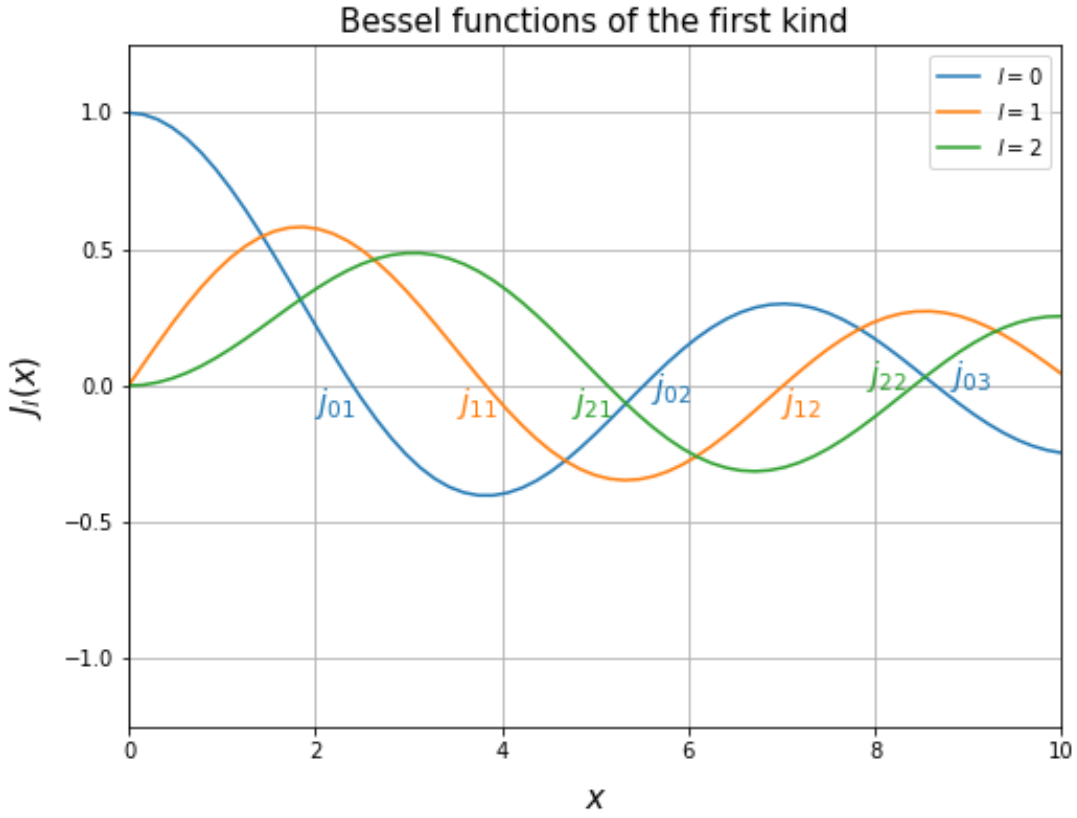
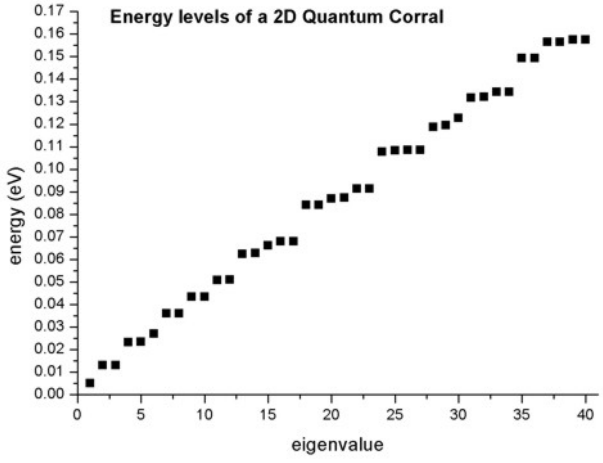


Figure 6.4.7.8: Bessel functions of the first kind for $l = 0, 1, 2$ generated by scipy.



	$[n, l]$	$j_{l,n}$	$j_{l,n}$ (approx.)	$E_{n,l}$ [eV]	$E_{n,l}$ [eV] (approx.)	$E_{n,l}$ [eV] (*nextnano++*)	$E_{n,l}$ [eV] (*nextnano*^{3})
1st	[1, 0]	2.405	$0.75\pi \simeq 2.356$	0.00530	0.00508	0.00510	0.00511
2nd	[1, 1]	3.832	$1.25\pi \simeq 3.926$	0.01345	0.01412	0.01294	0.01298
3rd	[1,-1]	3.832	$1.25\pi \simeq 3.926$	0.01345	0.01412	0.01294	0.01298
4th	[1, 2]	5.136	$1.75\pi \simeq 5.497$	0.02416	0.02768	0.02320	0.02325
5th	[1,-2]	5.136	$1.75\pi \simeq 5.497$	0.02416	0.02768	0.02329	0.02325
6th	[2, 0]	5.520	$1.75\pi \simeq 5.497$	0.02791	0.02767	0.02685	0.02693
7th	[2, 1]	7.016	$2.25\pi \simeq 7.067$	0.04508	0.04574	0.03584	0.03597

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)

Last update: nn/nn/nnnn

— DEV — Electron and hole wave functions in a T-shaped quantum wire grown by CEO (cleaved edge overgrowth)

> **Attention:** This tutorial is under construction

Input files:

- *T-QWR_GaAs-AlGaAs_Schuster_2005_2D_nnp.in*

Scope:

This tutorial aims to simulate the electron and hole wavefunctions of a T-shaped quantum wire (QWR). The tutorial is related to the PhD Thesis of R. Schuster *[SchusterPhD2005]*

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 $Al_{0.35}Ga_{0.65}As$ barriers (see Figure 6.4.7.9). 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 6.4.7.9 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.

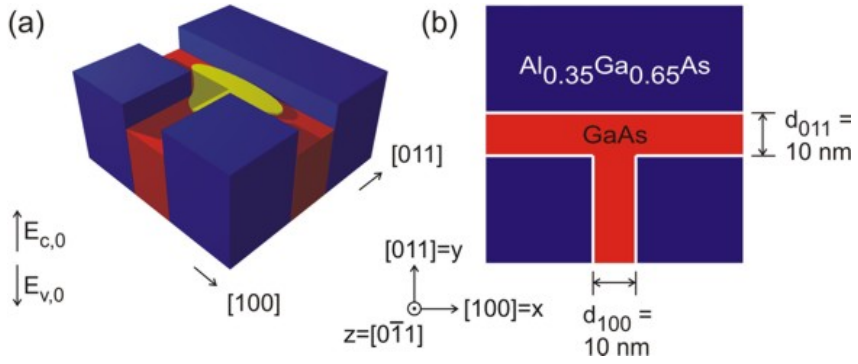


Figure 6.4.7.9: Two-dimensional conduction band edges of the T-shaped quantum wire.

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 $Al_{0.35}Ga_{0.65}As$:

$$\begin{aligned} \text{CBO} &= 0.2847\,\text{eV} \\ \text{VBO} &= -0.1926\,\text{eV} \\ E_{\text{gap,Al}_{0.35}\text{Ga}_{0.65}\text{As}} &= 2.2883\,\text{eV} \\ E_{\text{gap,GaAs}} &= 1.5193\,\text{eV} \end{aligned}$$

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 6.4.7.10 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.

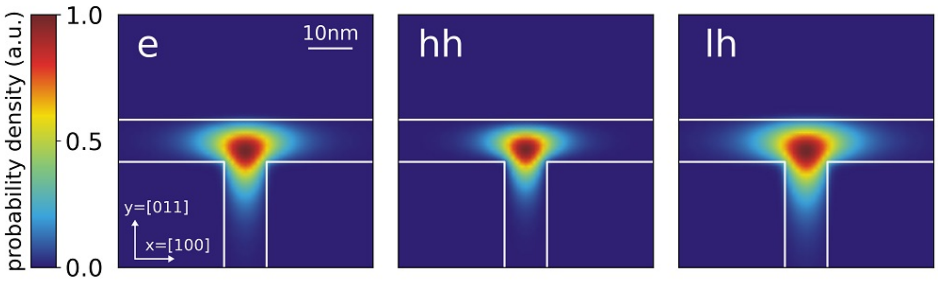


Figure 6.4.7.10: Probability densities of the electron (e), heavy hole (hh) and light hole (lh) state calculated using the effective mass approximation. The wavefunctions are normalized so that the maxima are equal to one.

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 6.4.7.11 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.

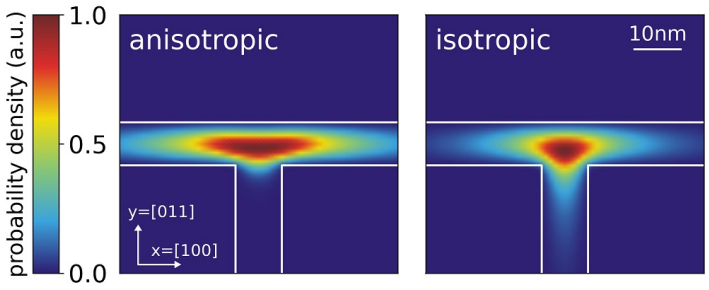


Figure 6.4.7.11: Probability density (ψ^2) for the hole ground state using anisotropic and isotropic k.p parameters.

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

Compared to *nextnano++*, *nextnano*[3] allows using anisotropic effective masses for solving the Schrödinger equation within the effective mass approximation. 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:

$$
\begin{aligned}
m^*_{\mathrm{hh},[100]} &= \frac{1}{\gamma_1 - 2\gamma_2}, \\
m^*_{\mathrm{hh},[011]} &= \frac{1}{\gamma_1 - 0.5 \cdot (\gamma_2 + 3\gamma_3)}, \\
m^*_{\mathrm{lh},[100]} &= \frac{1}{\gamma_1 + 2\gamma_2}, \\
m^*_{\mathrm{lh},[011]} &= \frac{1}{\gamma_1 + 0.5 \cdot (\gamma_2 + 3\gamma_3)}.
\end{aligned}
$$

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

$$
\begin{aligned}
m^*_{\mathrm{hh,isotropic}} &= \frac{1}{\gamma_1 - 0.8\gamma_2 - 1.2\gamma_3}, \\
m^*_{\mathrm{lh,isotropic}} &= \frac{1}{\gamma_1 + 0.8\gamma_2 + 1.2\gamma_3}.
\end{aligned}
$$

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^3 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 6.4.7.12 and Figure 6.4.7.13 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.

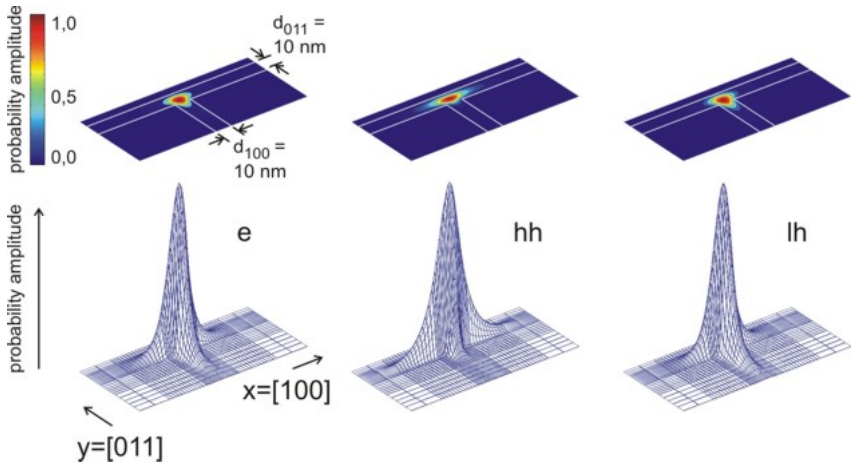


Figure 6.4.7.12: Probability amplitudes of the electron (e), heavy hole (hh) and light hole (lh) envelope functions at an unstrained T-shaped intersection of two 10 nm wide GaAs quantum wells embedded by $\mathrm{Al}_{0.35}\mathrm{Ga}_0.65\mathrm{As}$ barriers. The wave functions are normalized so that the maxima are equal to one.

These results are in very good qualitative agreement with the heavy hole and light hole wave functions calculated

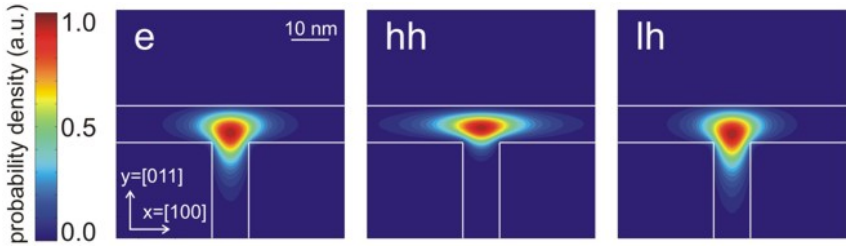


Figure 6.4.7.13: Contour diagram of the probability amplitudes of the electron (e), heavy hole (hh) and light hole (lh) eigenfunctions (same figures as Figure 6.4.7.12, but this time viewed from the top). The wave functions are normalized so that the maxima are equal to one.

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.

Last update: 17/07/2024

— DEV — Electron and hole wave functions in a T-shaped strained quantum wire grown by CEO (cleaved edge overgrowth)

Attention: This tutorial is under construction

Input files:

- *T-QWR-strained_GaAs-AlGaAs_Schuster_2005_2D_nnp.in*
- *Strained-QW_AlGaAs-InAlAs_1D_nnp.in*

Scope:
This tutorial treats strained quantum wires including a discussion of the strain calculation and the strain-induced piezoelectric fields (Poisson equation). The tutorial is related to the PhD Thesis of R. Schuster *[SchusterPhD2005]*

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 $In_{0.16}Al_{0.84}As$ barrier. The T-shaped intersection is surrounded by $Al_{0.3}Ga_{0.7}As$ which acts as a barrier to GaAs. The $In_{0.16}Al_{0.84}As$ barrier has a larger lattice constant than $Al_{0.3}Ga_{0.7}As$ 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 6.4.7.14 shows the sample layout.

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 $In_{0.16}Al_{0.84}As$ barrier (see Figure 6.4.7.15), in order to under-

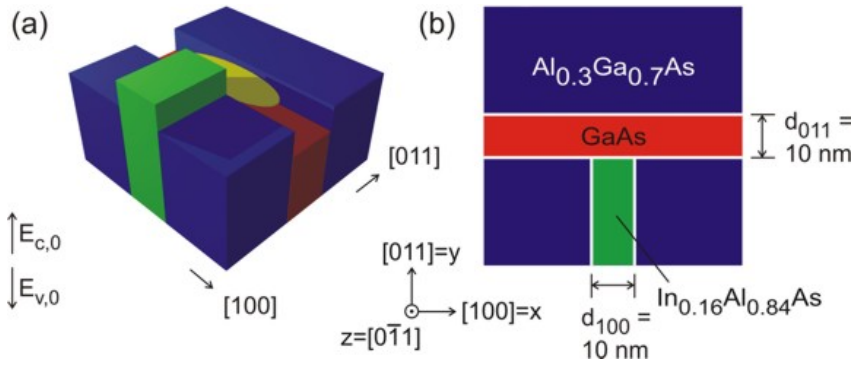


Figure 6.4.7.14: In (a) the two-dimensional conduction band edges of the T-shaped quantum wire without considering strain effects is shown. If one inverts the energy arrow then the left picture corresponds to the valence band edge. The wave function is indicated at the T-shaped intersection in yellow. 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.

stand the fundamental difference between these two layouts. As we see in from Figure 6.4.7.15 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 6.4.7.14 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.

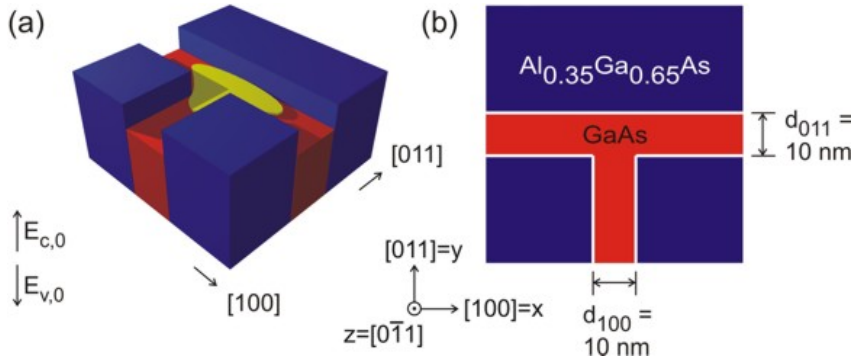


Figure 6.4.7.15: In (a) the two-dimensional conduction band edges of the T-shaped quantum wire (from the T-shaped quantum wire tutorial) without considering strain effects is shown. The wave function is indicated at the T-shaped intersection in yellow. In (b) a 60 nm x 60 nm extract of the schematic layout including the dimensions, the material composition and the orientation of the wire with respect to the crystal coordinate system is shown.

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 $\mathrm{In_{0.16}Al_{0.84}As}$ layer to adopt the lattice constant of $\mathrm{Al_{0.3}Ga_{0.7}As}$. The model for strain calculations can be specified inside the `strain{ }` group, where we choose the model: `minimized_strain{ }`.

In Figure 6.4.7.16 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 $\mathrm{In_{0.16}Al_{0.84}As}$ layer.

Note that in a one-dimensional example, which is provided in the input file *Strained-QW_AlGaAs-InAlAs_1D_nnp.in*, the strain tensor components of a $\mathrm{In_{0.16}Al_{0.84}As}$ layer that is strained pseudomorphically with

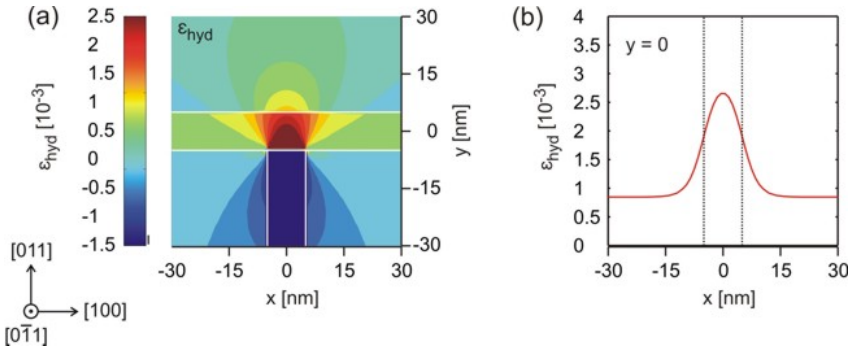


Figure 6.4.7.16: In (a) the hydrostatic strain ϵ_{hyd} inside the T-shaped quantum wire structure is shown. In (b) a cross-section of ϵ_{hyd} along x at $y = 0$ is shown.

respect to an $\mathrm{Al}_{0.30}\mathrm{Ga}_{0.7}\mathrm{As}$ 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 6.4.7.17 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 $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ is forced to have the same lattice constant as $\mathrm{Al}_{0.3}\mathrm{Ga}_{0.7}\mathrm{As}$ 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. $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$, which has a larger lattice constant than $\mathrm{Al}_{0.3}\mathrm{Ga}_{0.7}\mathrm{As}$ is strained compressively along the z direction. Similar to the 1D case, it is also expected that the ϵ_{yy} component inside the $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ barrier has a similar value to ϵ_{zz}, which is clearly the case. The dark blue area in Figure 6.4.7.17 (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 6.4.7.18). The same applies to the value of ϵ_{xx}, which is similar to the 1D value inside the $\mathrm{In}_{0.16}\mathrm{Al}_{0.84}\mathrm{As}$ 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.

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 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 6.4.7.19.

By comparing Figure 6.4.7.17 (a) and Figure 6.4.7.19 (a) we observe that $\epsilon_{\tilde{x}\tilde{x}} = \epsilon_{xx}$, because the x coordinate axes coincide. Symmetry arguments show that the following holds:

$$\begin{aligned}
\epsilon_{\tilde{y}\tilde{y}} &= \frac{1}{2}\left(\epsilon_{yy} + \epsilon_{zz}\right) \\
\epsilon_{\tilde{x}\tilde{y}} = \epsilon_{\tilde{x}\tilde{z}} &= \frac{1}{\sqrt{2}}\epsilon_{xy}
\end{aligned}$$

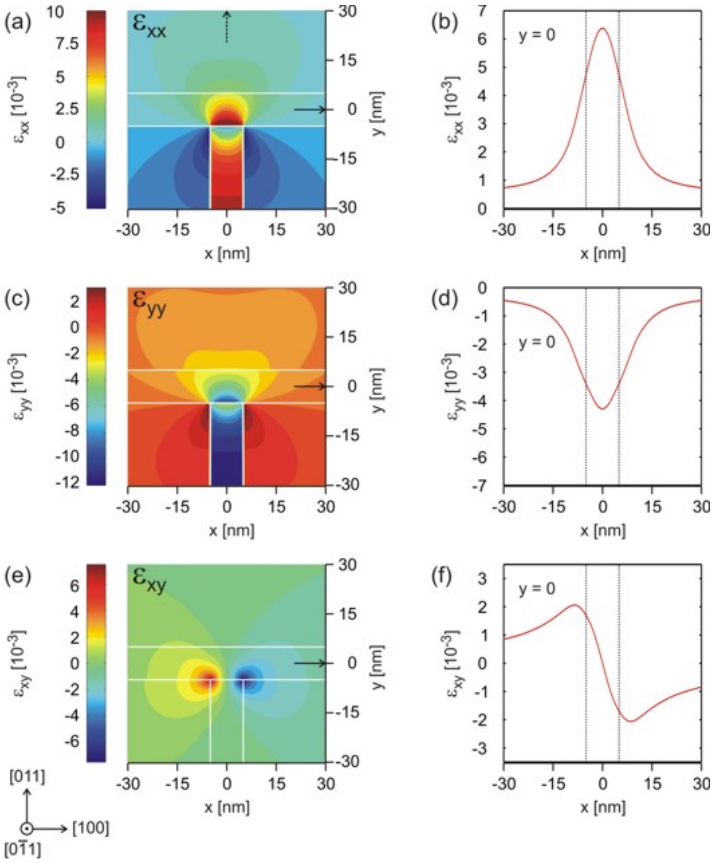


Figure 6.4.7.17: In (a), (c), (e) the strain components ϵ_{xx}, ϵ_{yy}, ϵ_{xy} are shown. In (b), (d), (f) a cut through the structure along x at $y = 0$ is shown.

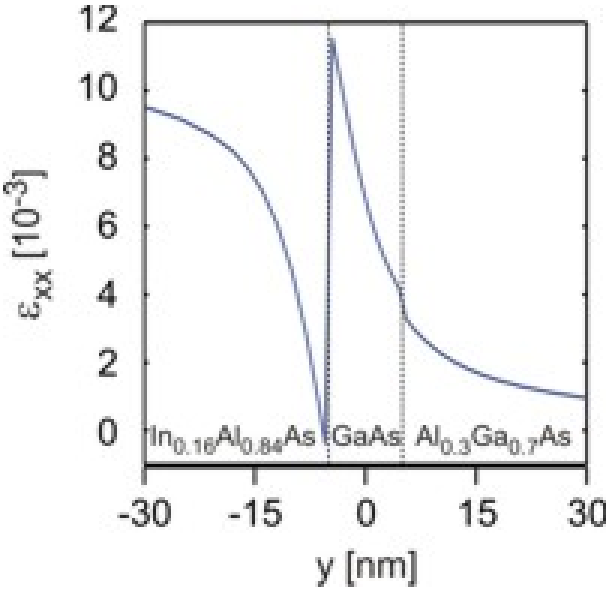


Figure 6.4.7.18: Strain tensor component ϵ_{xx} along y direction at position $x = 0$.

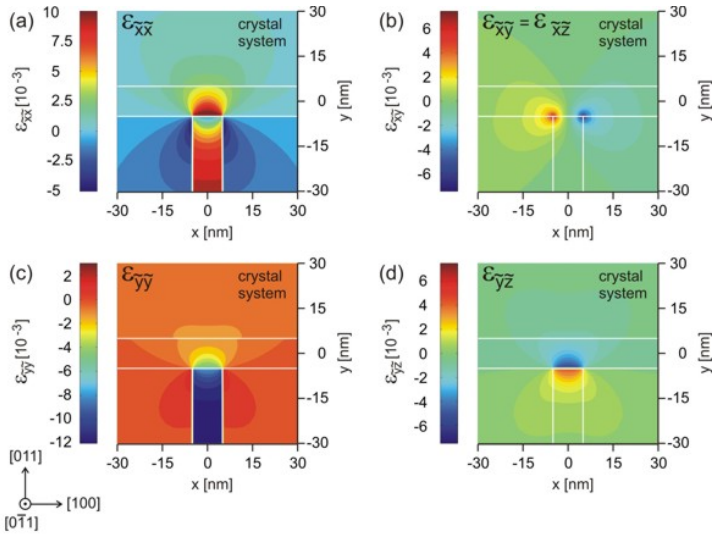


Figure 6.4.7.19: Strain tensor components $\epsilon_{\tilde{x}\tilde{x}}$, $\epsilon_{\tilde{y}\tilde{y}}$, $\epsilon_{\tilde{x}\tilde{y}} = \epsilon_{\tilde{x}\tilde{z}}$ and $\epsilon_{\tilde{y}\tilde{z}}$ with respect to the crystal coordinate system. The rotation with respect to the simulation system is a rotation of 45 degrees around the x axis, i.e. the [100] axis.

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 $\mathbf{P}_{\text{piezo}}$, given by

$$\mathbf{P}_{\text{piezo}} = e_{14} \begin{pmatrix} 2\epsilon_{\tilde{y}\tilde{z}} \\ 2\epsilon_{\tilde{x}\tilde{z}} \\ 2\epsilon_{\tilde{x}\tilde{y}} \end{pmatrix},$$

where e_{14} is the piezoelectric constant in units of $[\mathbf{C/m^2}]$. Once having determined the piezoelectric polarization, one is able to compute the piezoelectric charge density:

$$\rho_{\text{piezo}}(x, y) = -\text{div}\, \mathbf{P}_{\text{piezo}}(x, y).$$

In Figure 6.4.7.20 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.

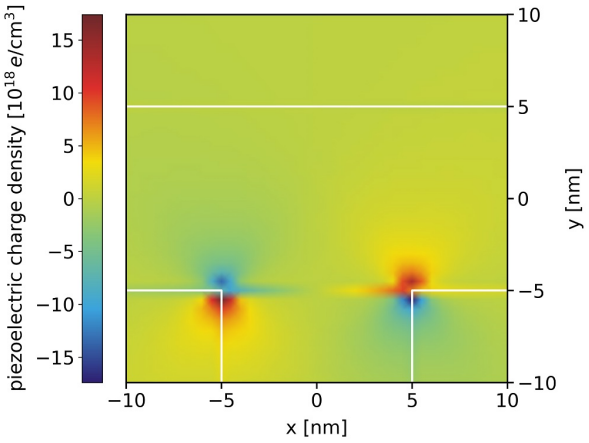


Figure 6.4.7.20: Piezoelectric charge density $\rho_{\text{piezo}}(x, y)$.

Calculation of the conduction and valence band edges

In Figure 6.4.7.21 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.

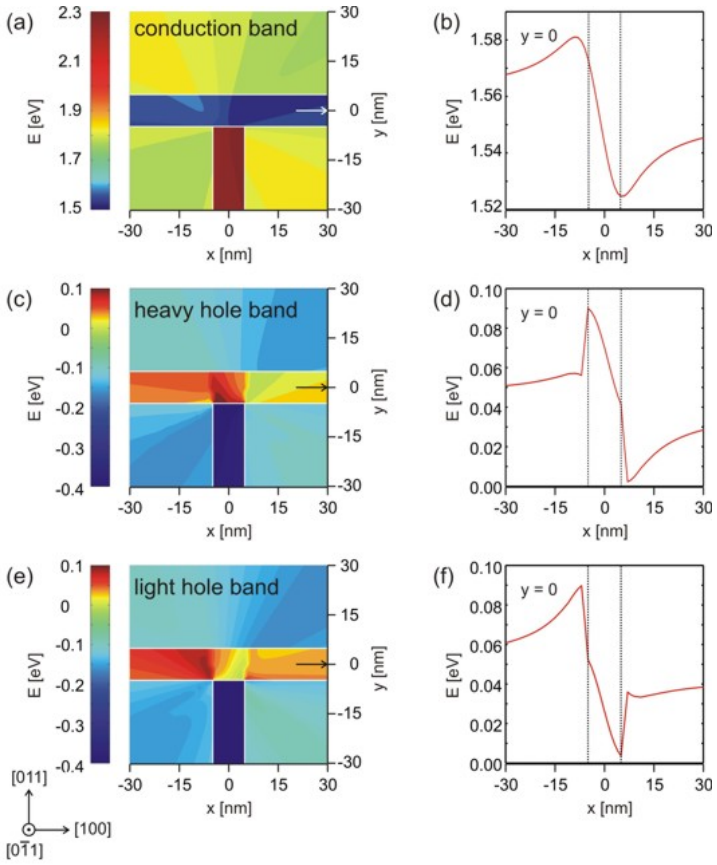


Figure 6.4.7.21: In (a), (c), (e) a 2D plot of the conduction, heavy hole and light hole band edge energies are shown. In (b), (d), (f) a cut through the conduction, heavy hole and light hole band edge energies at $y = 0$.

Electron and heavy hole wave functions

Figure 6.4.7.22 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 6.4.7.22 (a) the piezoelectric effect was not included. As one can clearly see in Figure 6.4.7.22 (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: 17/07/2024

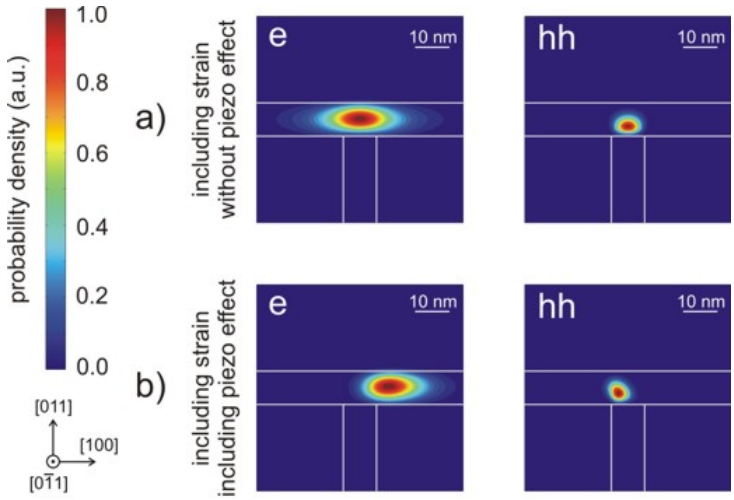


Figure 6.4.7.22: In (a) the contour diagram of the square of the electron (e) and heavy hole (hh) wave functions (i.e. ψ^2) for the case where strain is included in the simulations, but piezoelectricity is not. Subplot (b) shows the same results as in (a), but this time including the piezoelectric effect. Note that in the plot the wave functions are normalized so that the maximum equals one, respectively.

6.4.8 Quantum Dots

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:

$$\begin{aligned} E_{n_1,n_2,n_3} &= \frac{\hbar^2\pi^2}{2m_e}\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 1.4462697\cdot 10^{-17}\ \mathrm{eVm}^2\cdot\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 0.1446269\ \mathrm{eV}\cdot(n_1^2+n_2^2+n_3^2) \end{aligned} \tag{6.4.8.1}$$

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. (6.4.8.1) 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
```

(continues on next page)

(continued from previous page)

```
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 6.4.8.1 and Figure 6.4.8.2 show the isosurfaces of the electron wave function (Ψ^2) of the ground state and the 11th state, respectively. Both states are nondegenerate.

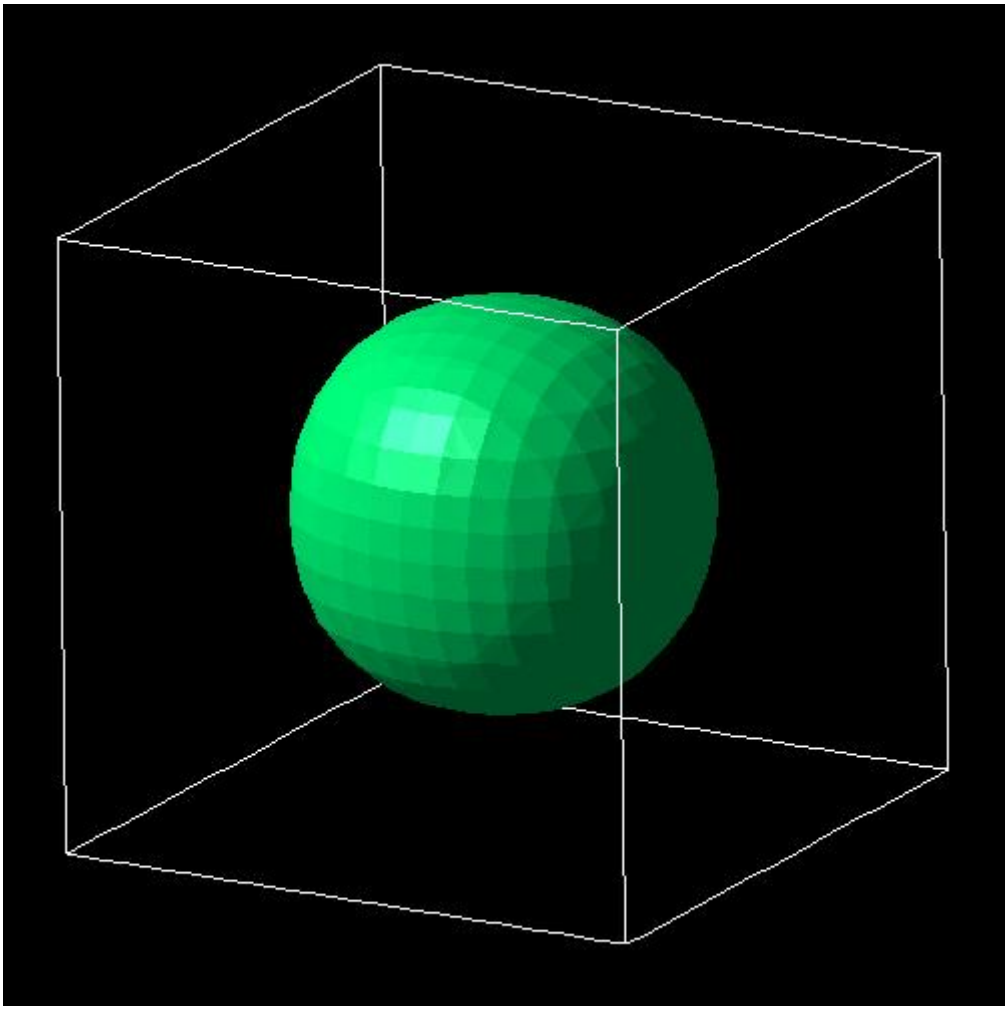


Figure 6.4.8.1: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm cubic quantum dot with infinite barriers for the ground state E_{111}.

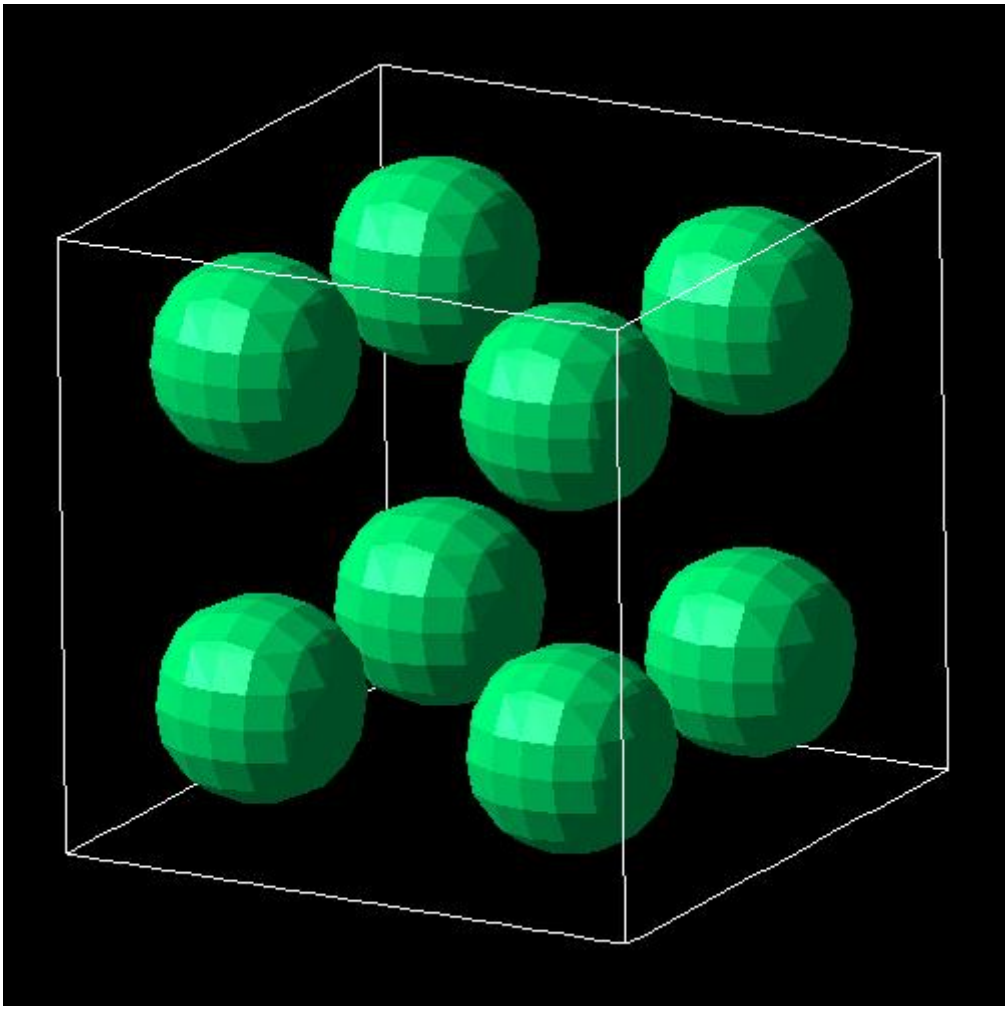


Figure 6.4.8.2: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm cubic quantum dot with infinite barriers for the 11th eigenstate E_{222}.

Intraband (= intersubband) transitions

```
quantum{
    region{
        ...
        intraband_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
        }
    }
}
```

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:

$$\begin{aligned} E_{n_1,n_2,n_3} &= \frac{\hbar^2\pi^2}{2m_e}\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 1.4462697\cdot 10^{-17}\ \mathrm{eVm}^2\cdot\left(\frac{n_1^2}{L_x^2}+\frac{n_2^2}{L_y^2}+\frac{n_3^2}{L_z^2}\right) \\ &= 0.1446269\ \mathrm{eV}\cdot(n_1^2+n_2^2)+0.5785079\ \mathrm{eV}\cdot n_3^2 \end{aligned} \tag{6.4.8.2}$$

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/
```

(continues on next page)

(continued from previous page)

```
↪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
30      4.794363              4.886326      = E332
...
34      5.121061              5.400036      = E441
```

The following figures: Figure 6.4.8.3, Figure 6.4.8.4, Figure 6.4.8.5 and Figure 6.4.8.6 show the isosurfaces of the electron wave function (ψ^2) of the 1st, 4th, 9th and 14th, respectively. All these states are nondegenerate.

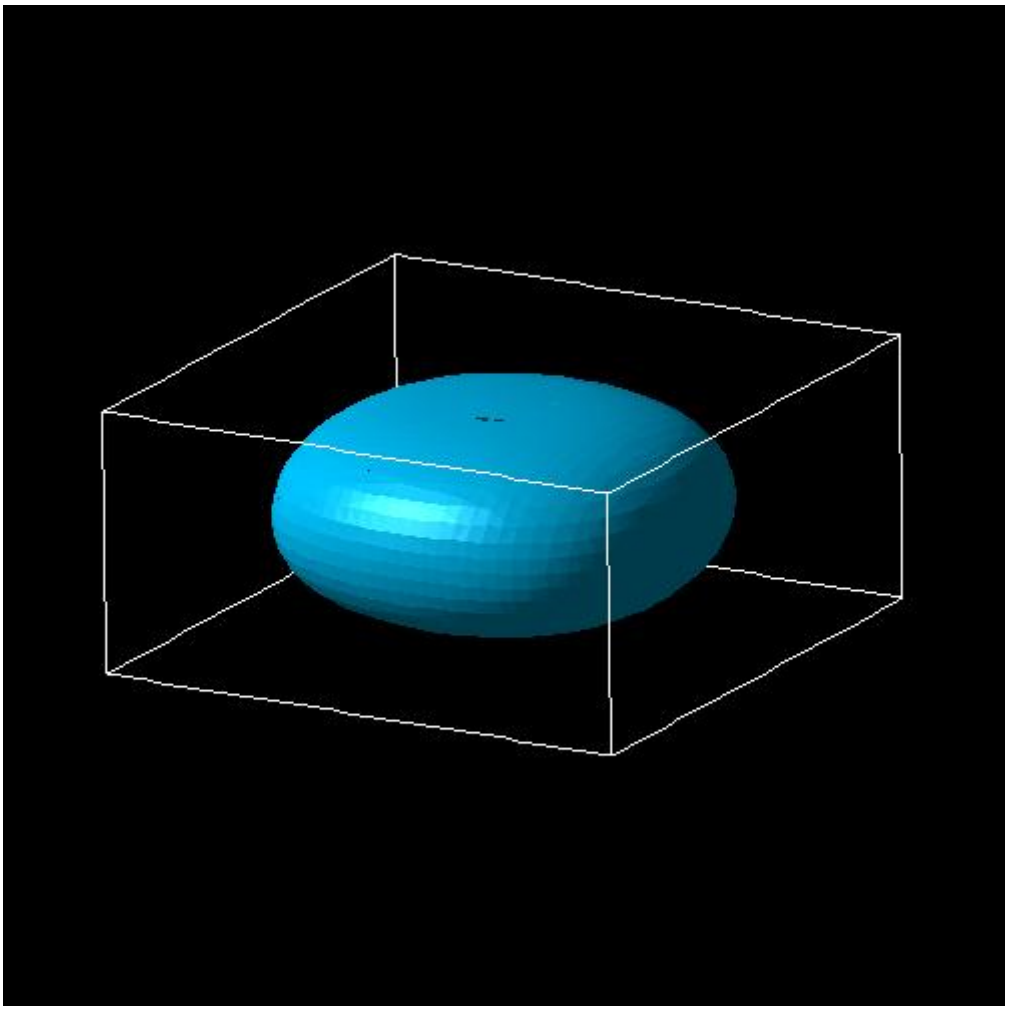


Figure 6.4.8.3: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm by 10 nm by 5 nm cuboidal shaped quantum dot with infinite barriers for the ground state E_{111}.

Last update: nn/nn/nnnn

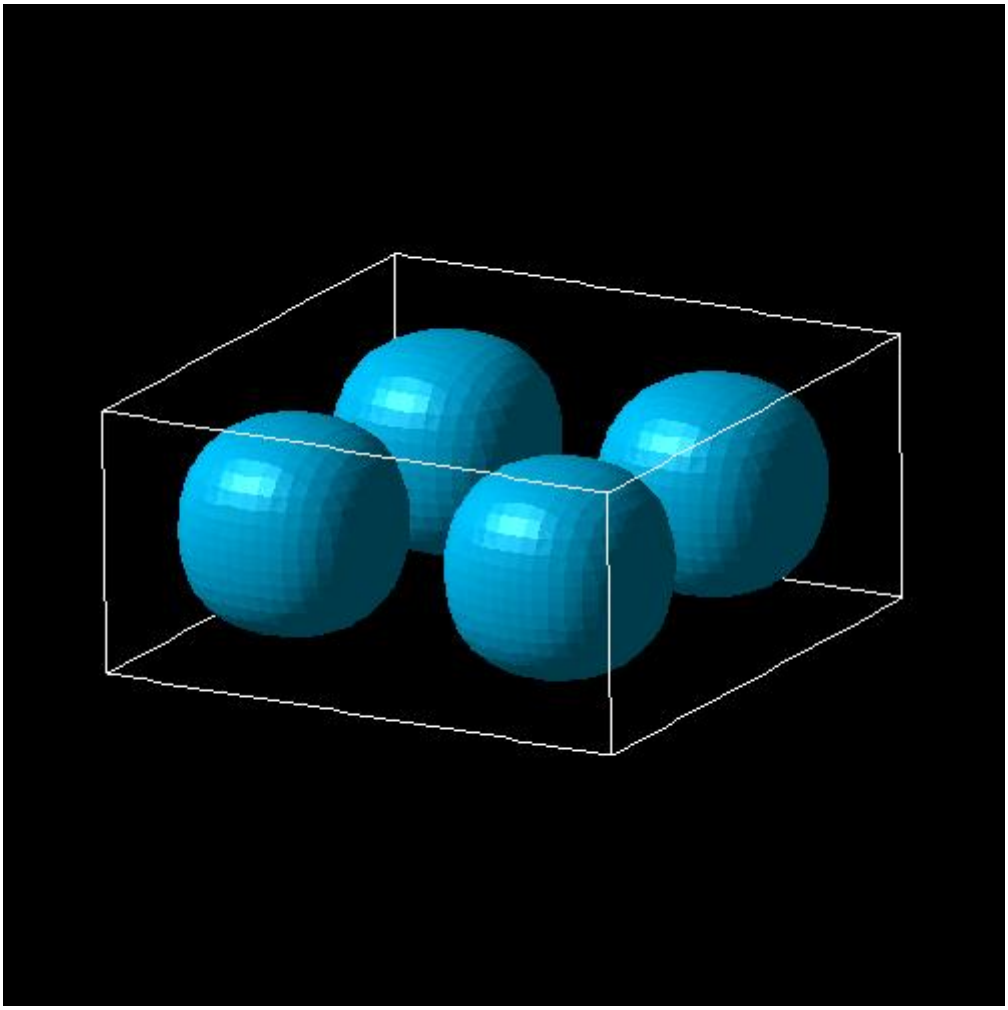


Figure 6.4.8.4: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm by 10 nm by 5 nm cuboidal shaped quantum dot with infinite barriers for the 4th state E_{221}.

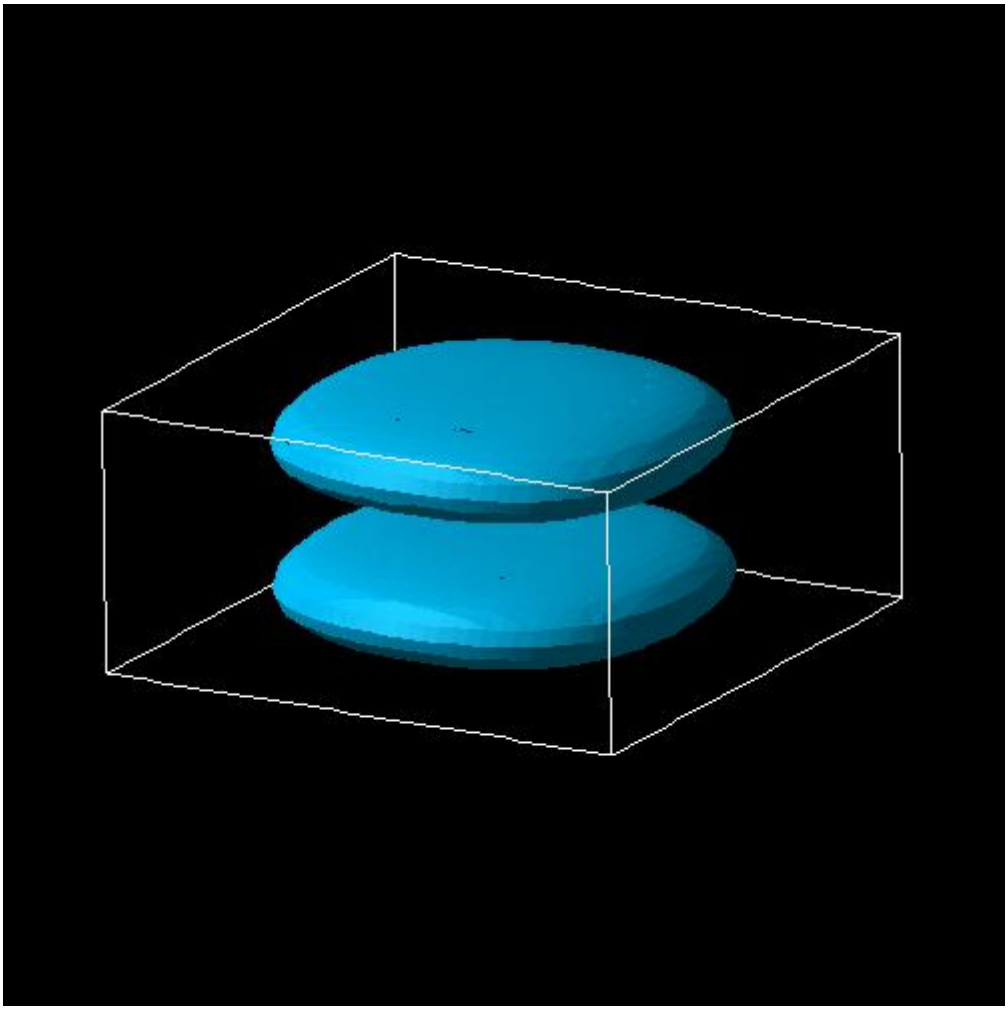


Figure 6.4.8.5: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm by 10 nm by 5 nm cuboidal shaped quantum dot with infinite barriers for the 9th state E_{112}.

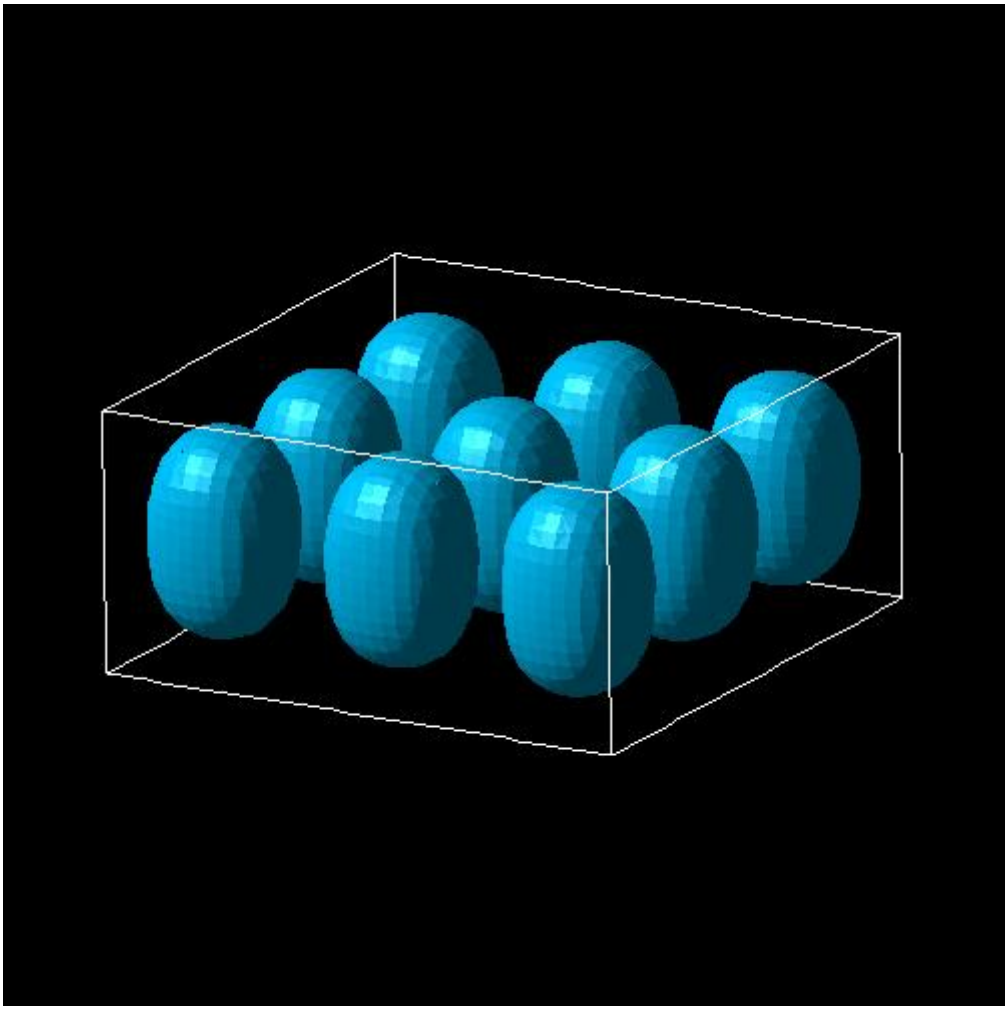


Figure 6.4.8.6: Isosurfaces of the electron wave function (Ψ^2) of a 10 nm by 10 nm by 5 nm cuboidal shaped quantum dot with infinite barriers for the 14th state E_{331}.

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:

```
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 6.4.8.7 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:

$$E_1 = -\frac{\hbar^2 \pi^2}{2 m_h R^2} = -0.314 eV$$

using m_h = 0.192 m_0 as *[Burdov2002]*, where he uses incorrect k.p parameters: In his definition L must be -5.8 and M = -3.43.

$$E_1 = -\frac{\hbar^2 \pi^2}{2 m_h R^2} = -0.254 eV$$

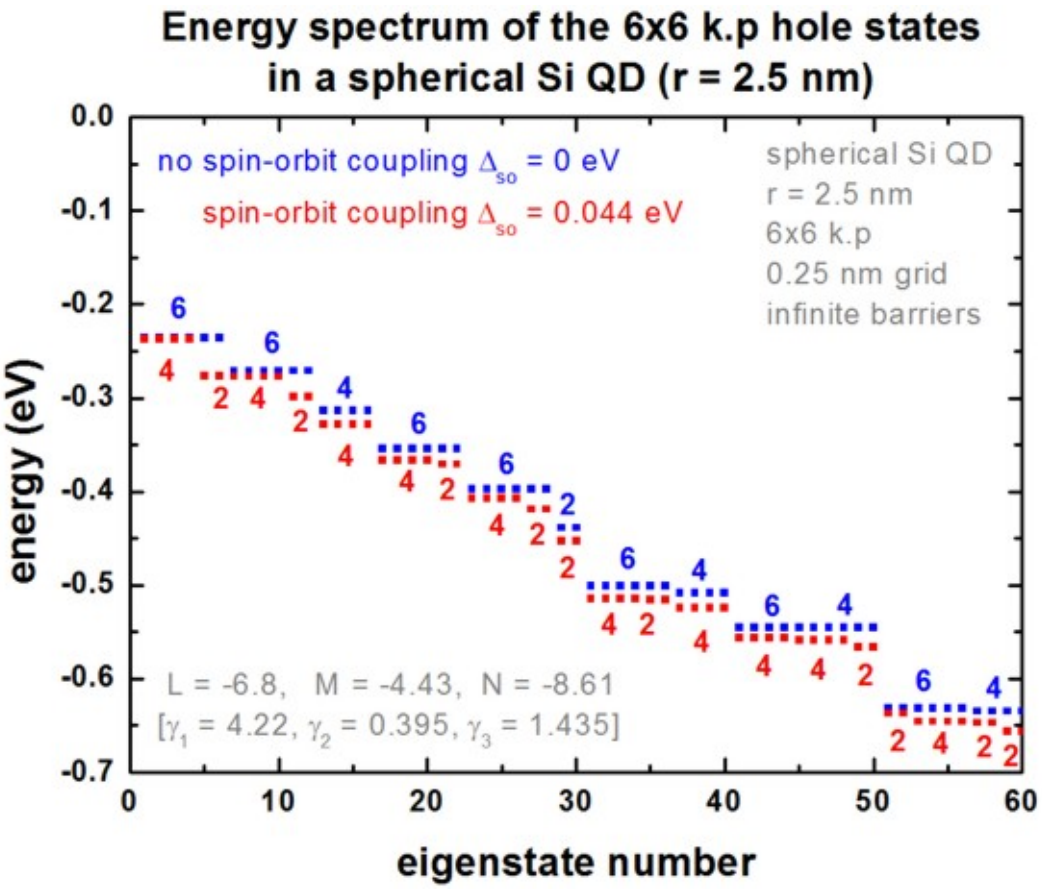


Figure 6.4.8.7: Energy spectrum of the 6-band k.p hole states in a spherical Si QD.

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 nextnano³ and nextnano++

Figure 6.4.8.8 compares the *nextnano³* results with the *nextnano++* results. The results of both simulators are in excellent agreement.

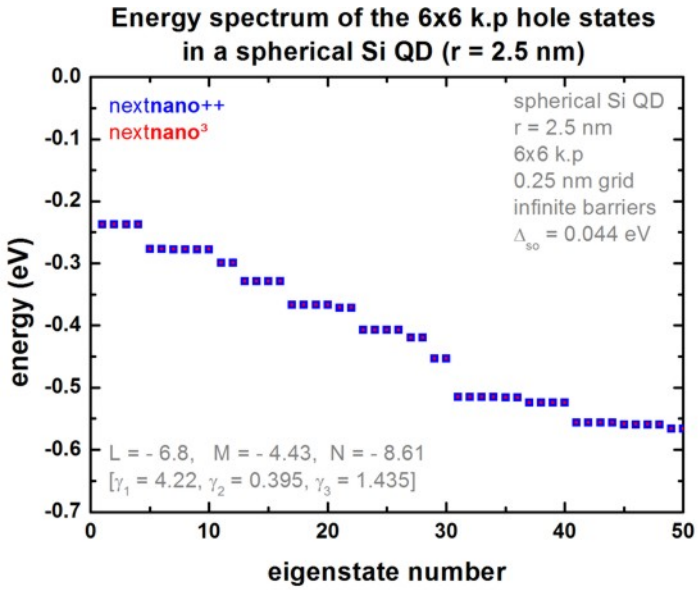


Figure 6.4.8.8: Energy spectrum of the 6-band k.p hole states in a spherical Si QD (Comparison *nextnano++* and *nextnano³*).

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.

Last update: nn/nn/nnnn

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).

i.e. they are the same as in the 3D CEO QD tutorial (apart from the effective masses).

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.

Results

Electron and heavy hole ground states

The following figures show the square of the electron (left side) and heavy hole (right side) wave functions (isovolumes of 25 % of ψ^2) for different applied electric fields (0 kV/cm, -17.5 kV/cm, -40 kV/cm). A slice through the conduction and valence band edges along the growth direction through the center of the QDs is also shown.

- Zero electric field

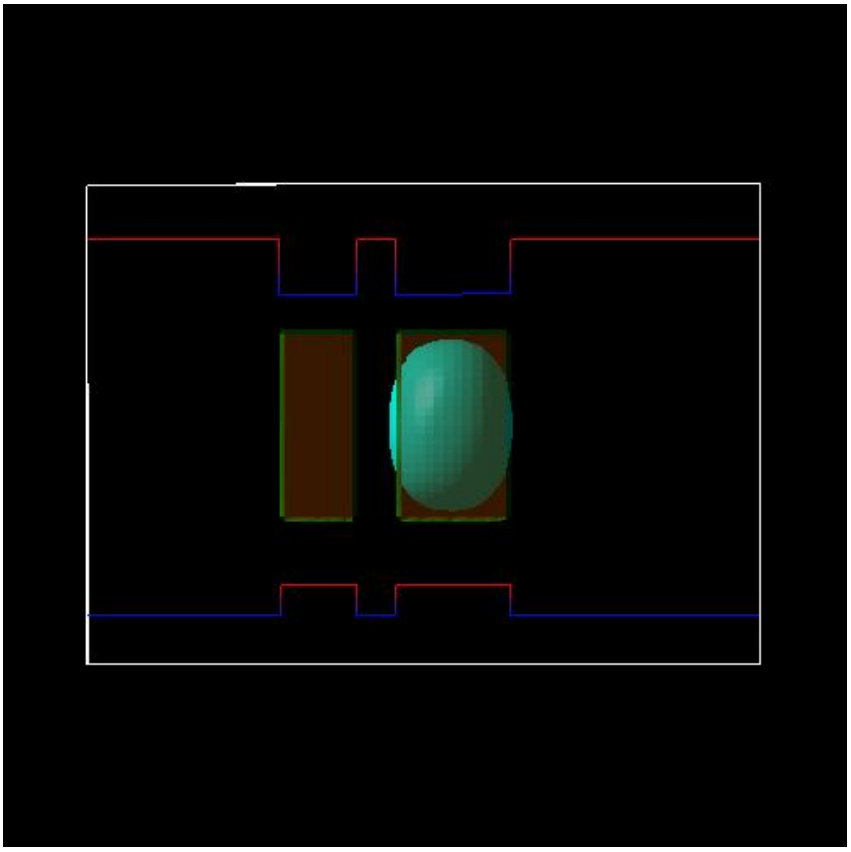


Figure 6.4.8.9: **electron** ground state for zero electric field

At zero applied electric field, both electron and heavy hole are located in the larger dot and form a **direct (bright) exciton**

- Electric field of -17.5 kV/cm

  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 forms a bonding (Figure 6.4.8.12) and an antibonding state (not shown) and is thus located in both wells (strong coupling). The exciton that is formed is something in between a direct and an indirect exciton.

- Electric field of -40 kV/cm

  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 6.4.8.23 below on spatial electron-hole overlap integrals).

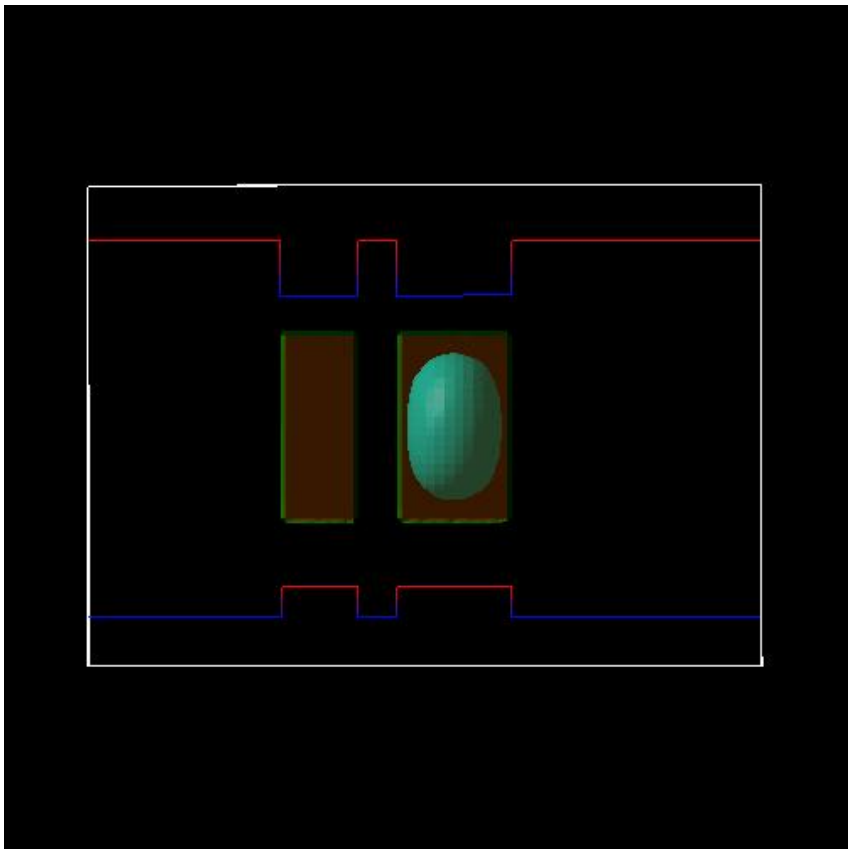


Figure 6.4.8.10: **heavy hole** ground state for zero electric field

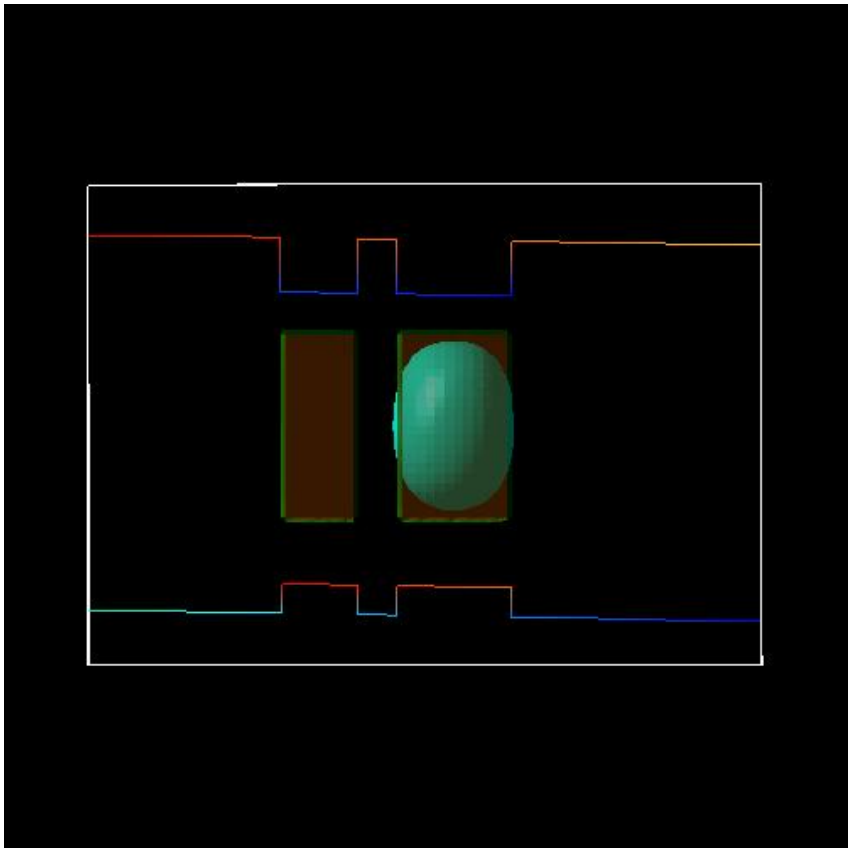


Figure 6.4.8.11: **electron** ground state for an electric field of -17.5 kV/cm

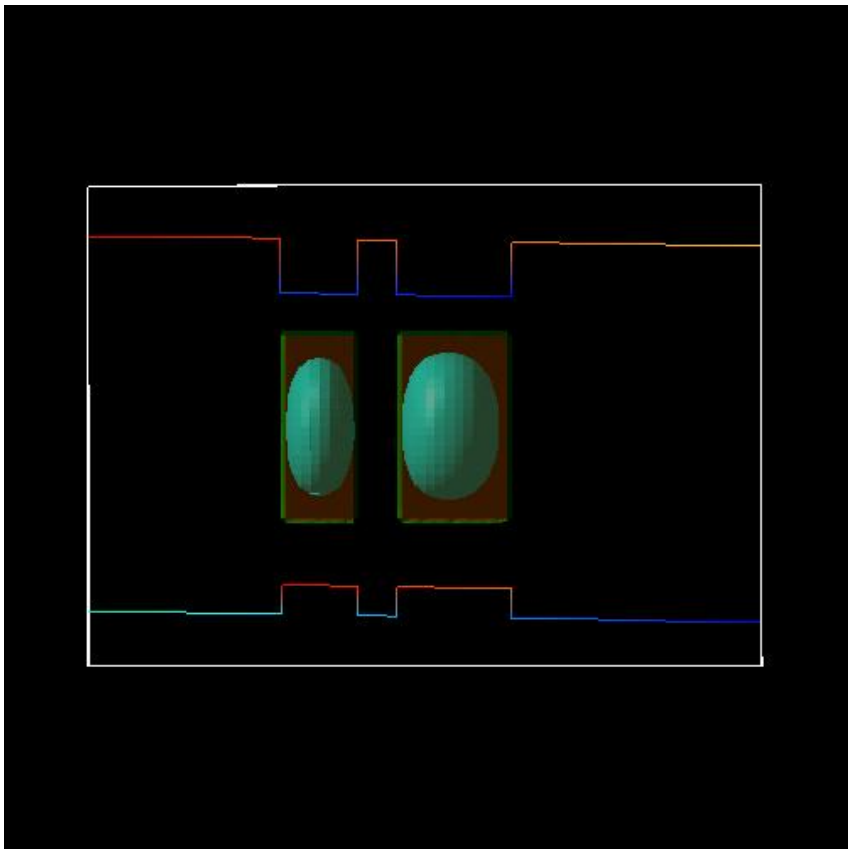


Figure 6.4.8.12: **heavy hole** ground state for an electric field of -17.5 kV/cm

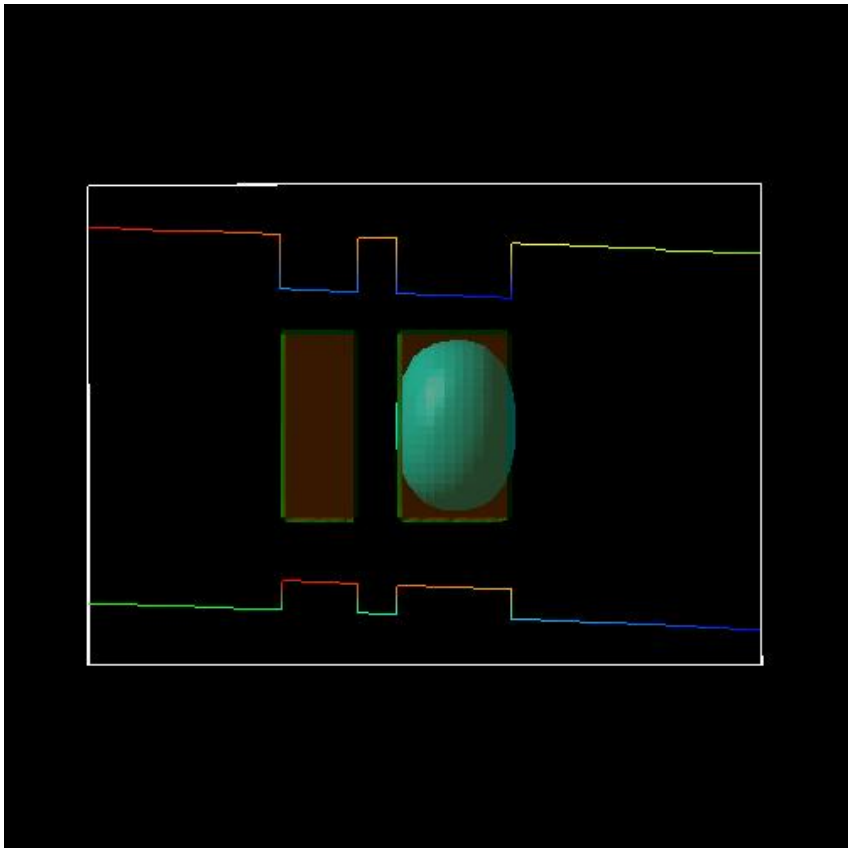


Figure 6.4.8.13: **electron** ground state for an electric field of -40 kV/cm

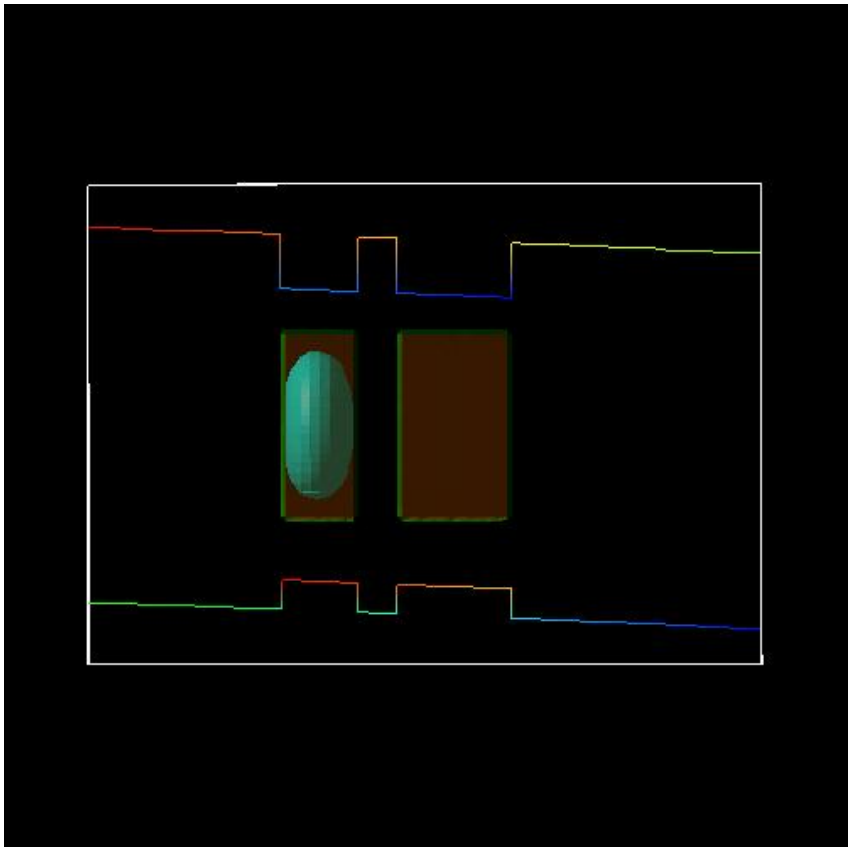


Figure 6.4.8.14: **heavy hole** ground state for an electric field of -40 kV/cm

Electron and heavy hole energies

Figure 6.4.8.15 shows the electron energies of the ground state (**e1**) and the first excited electron state (e2) of the QD molecule. The 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.

Figure 6.4.8.16 shows the heavy hole energies of the ground state (**h1**) and the excited hole states (h2, h3, h4, h5) of the QD molecule. 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 (see Figure 6.4.8.17 to Figure 6.4.8.21 of the hole wave functions further below).

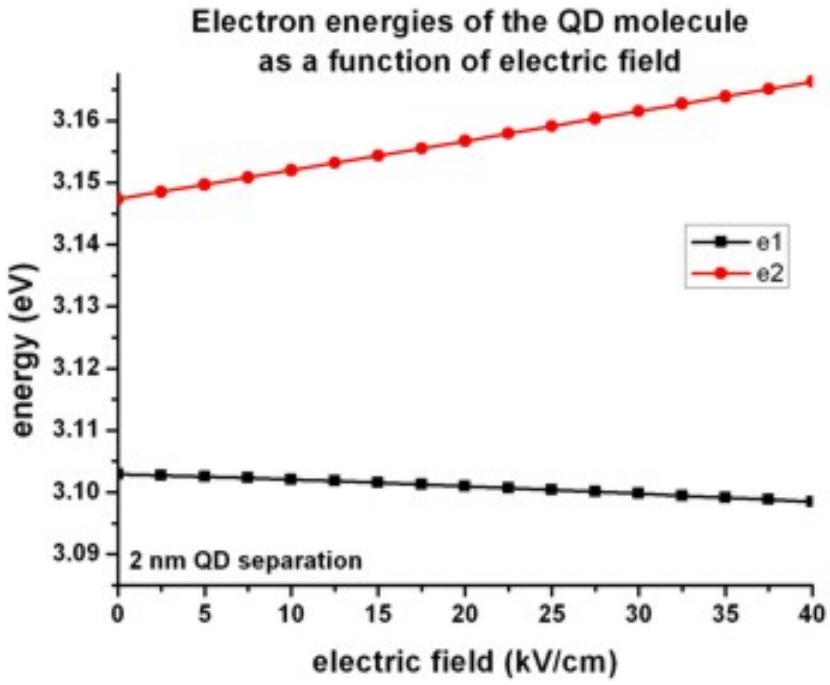


Figure 6.4.8.15: Electron energies

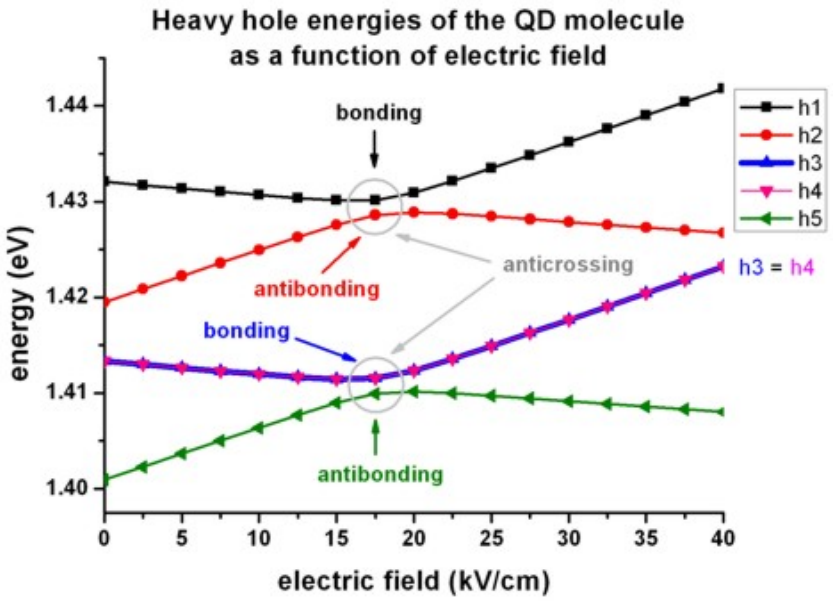


Figure 6.4.8.16: Heavy hole energies

Bonding and antibonding heavy hole state at anticrossing point

The following figures show the square of the lowest five hole wave functions (isovolumes of 2 % of ψ^2) at an electric field of -17.5 kV/cm.

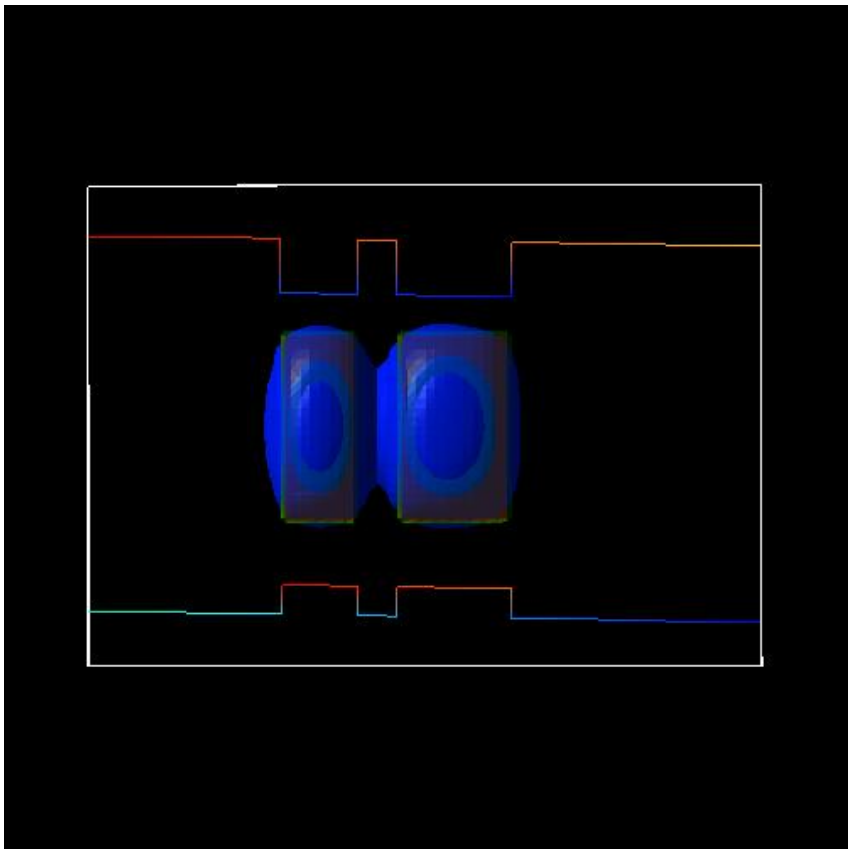


Figure 6.4.8.17: heavy hole (**h1**) ground state at -17.5 kV/cm (**bonding**)

Electron-hole transition energies

The following figure shows the five lowest electron-hole transition energies of the QD molecule as a function of electric field. 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**.

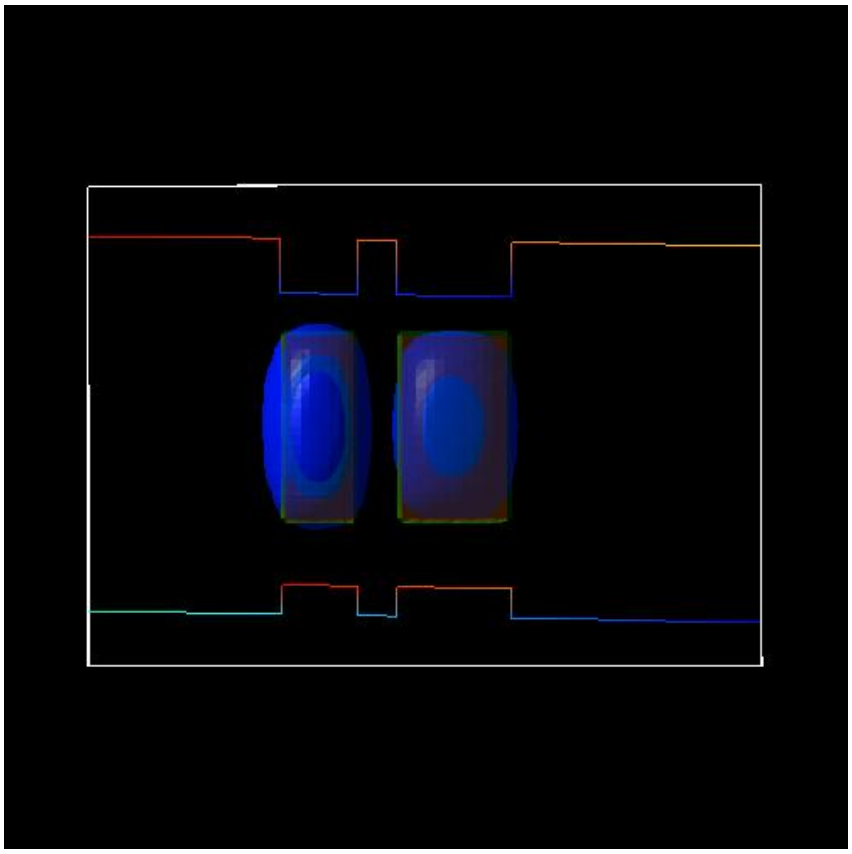


Figure 6.4.8.18: heavy hole (h2) first excited state at -17.5 kV/cm (antibonding)

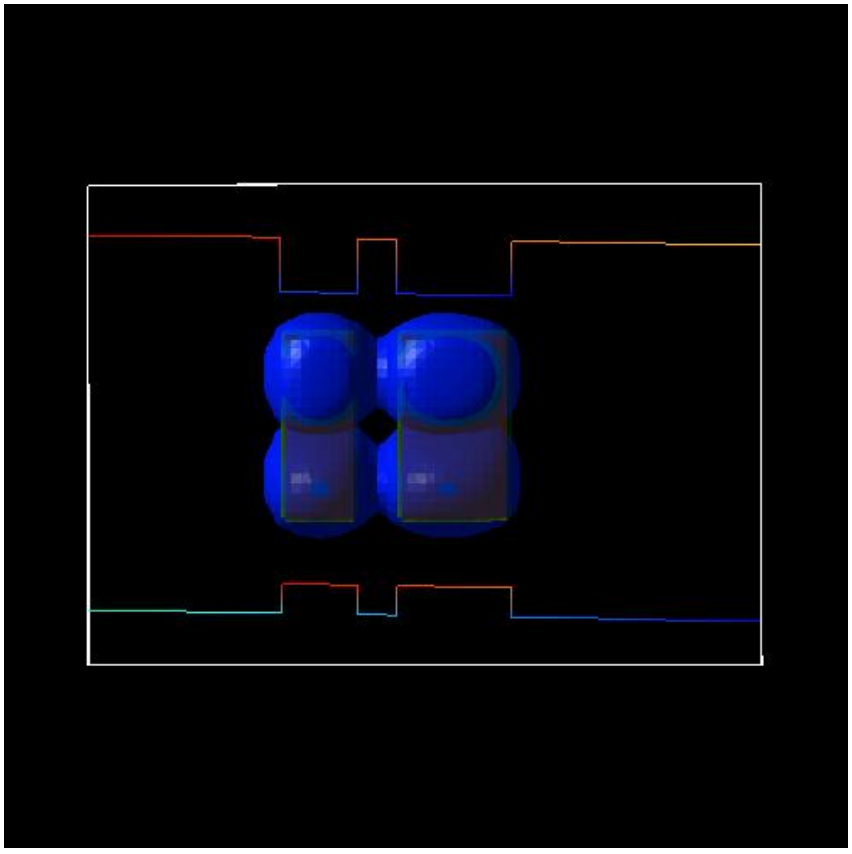


Figure 6.4.8.19: heavy hole (h3) state at -17.5 kV/cm (bonding)

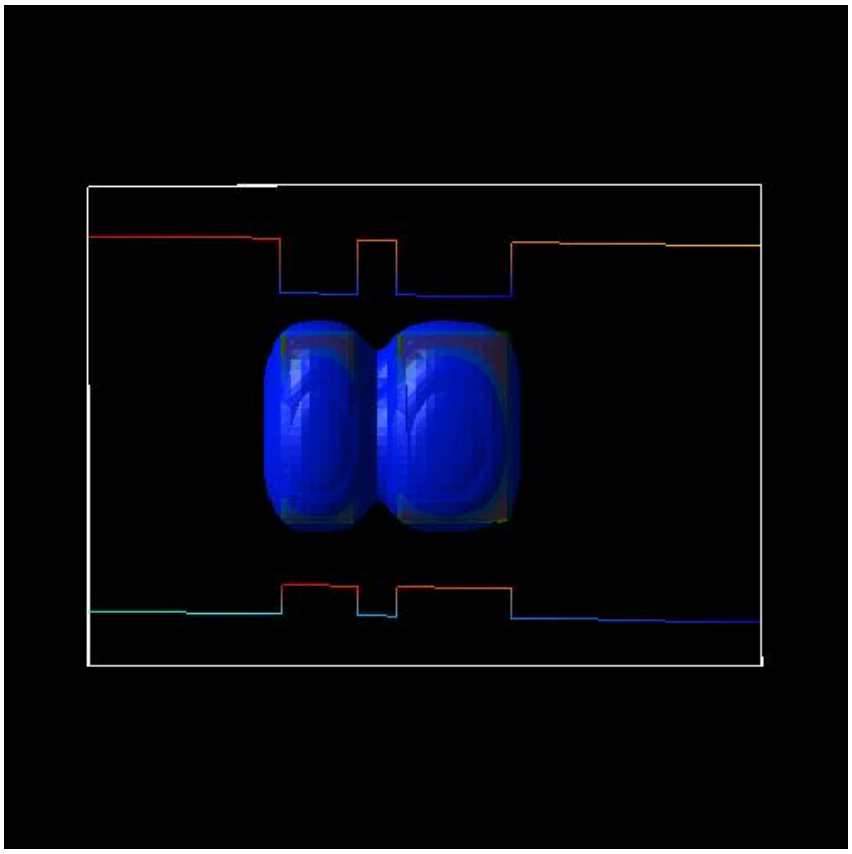


Figure 6.4.8.20: heavy hole (h4) state at -17.5 kV/cm (bonding)

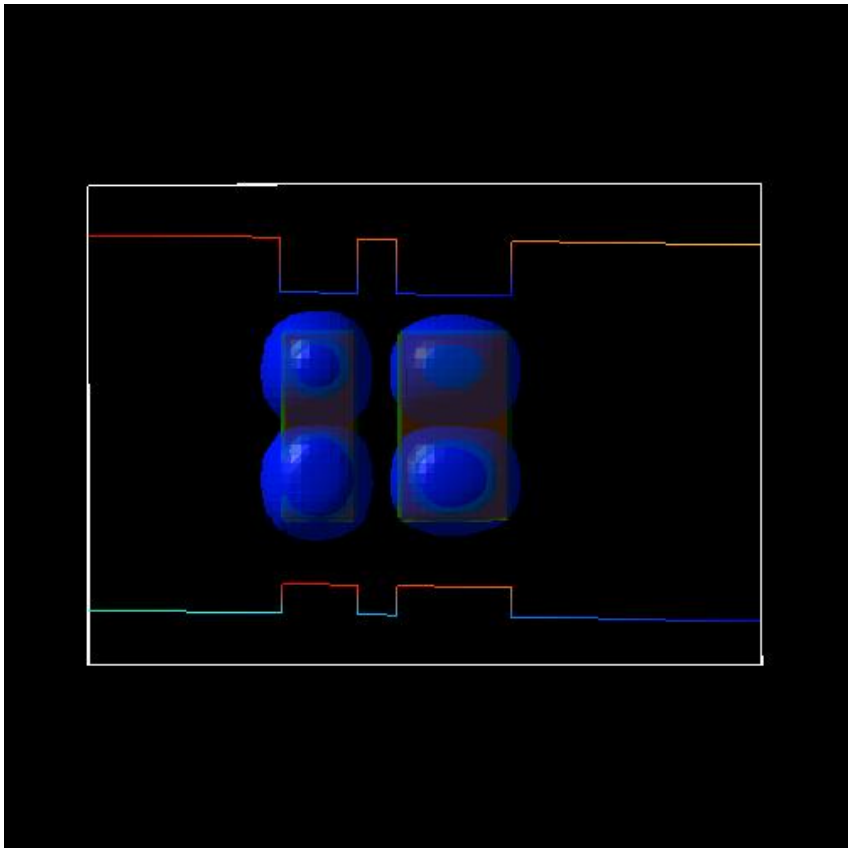


Figure 6.4.8.21: heavy hole (h5) state at -17.5 kV/cm (antibonding)

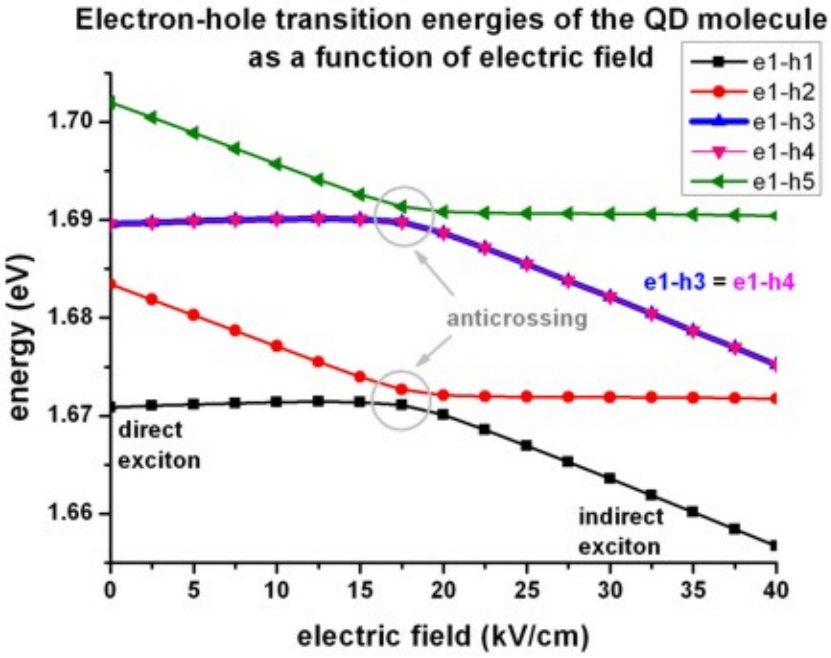


Figure 6.4.8.22: Electron-hole transition energies

Electron-hole 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.

$$\int \psi^*_{el,i}(x)\psi_{hl,j}dx$$

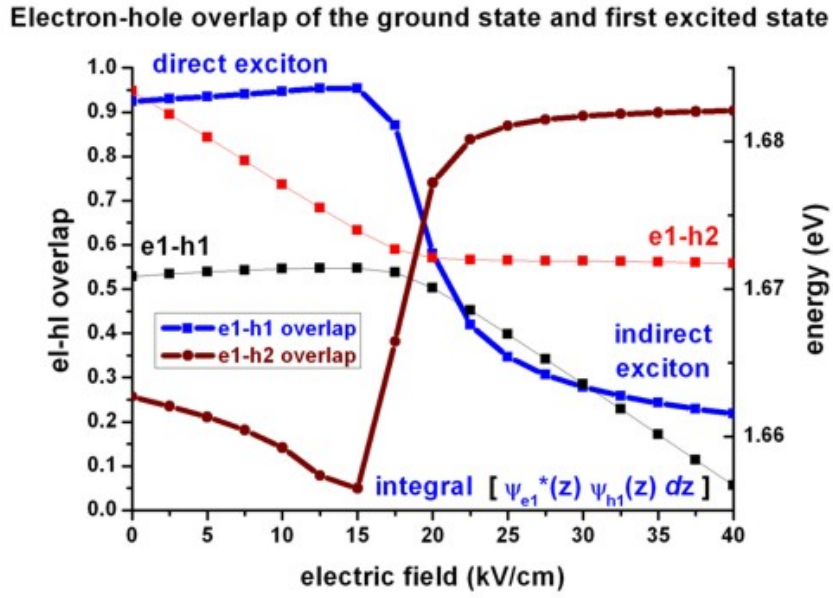


Figure 6.4.8.23: Spatial electron-hole overlap integrals

Last update: nn/nn/nnnn

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.

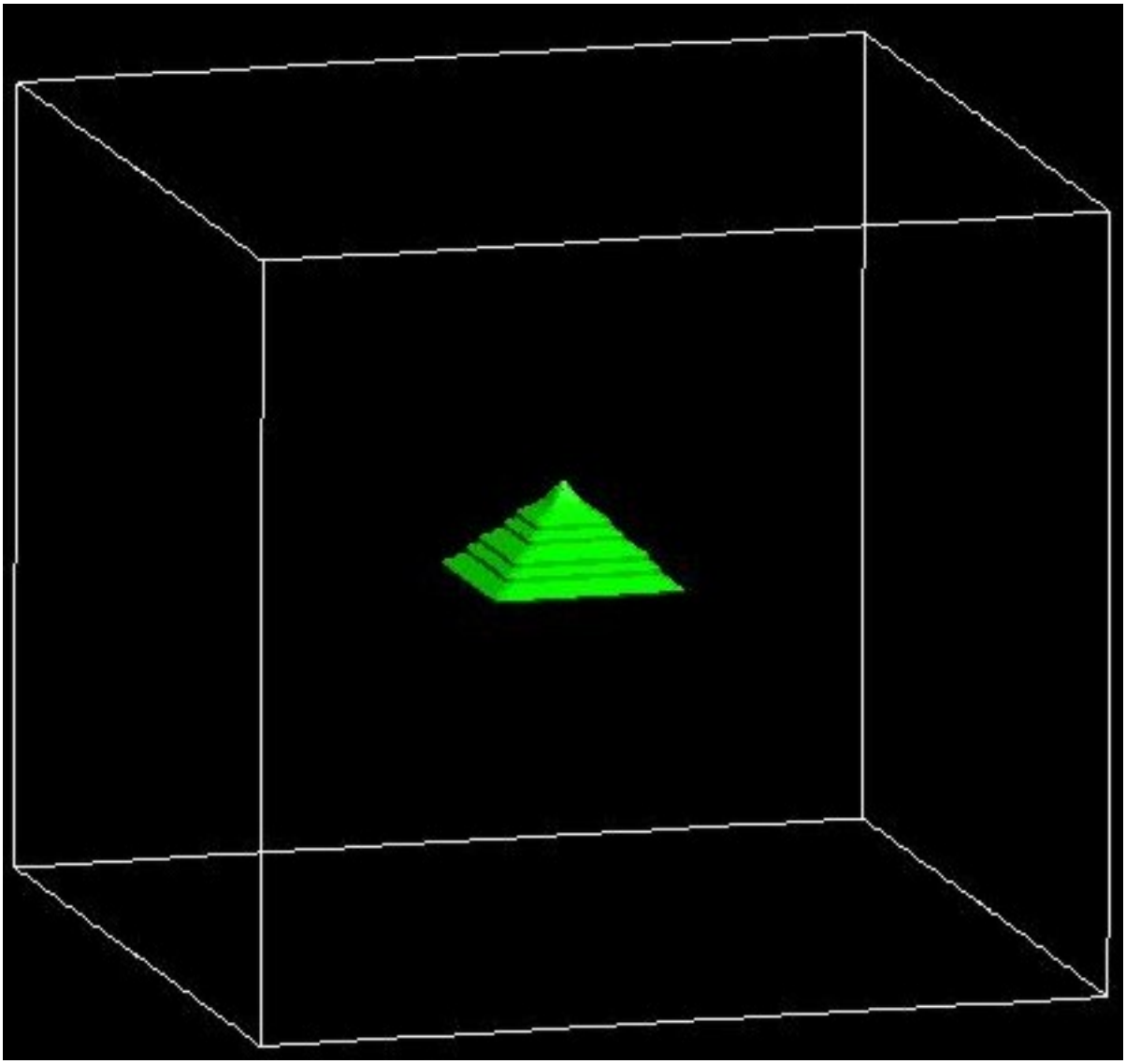
The whole simulation area has the dimensions 44 nm x 44 nm x 40 nm.

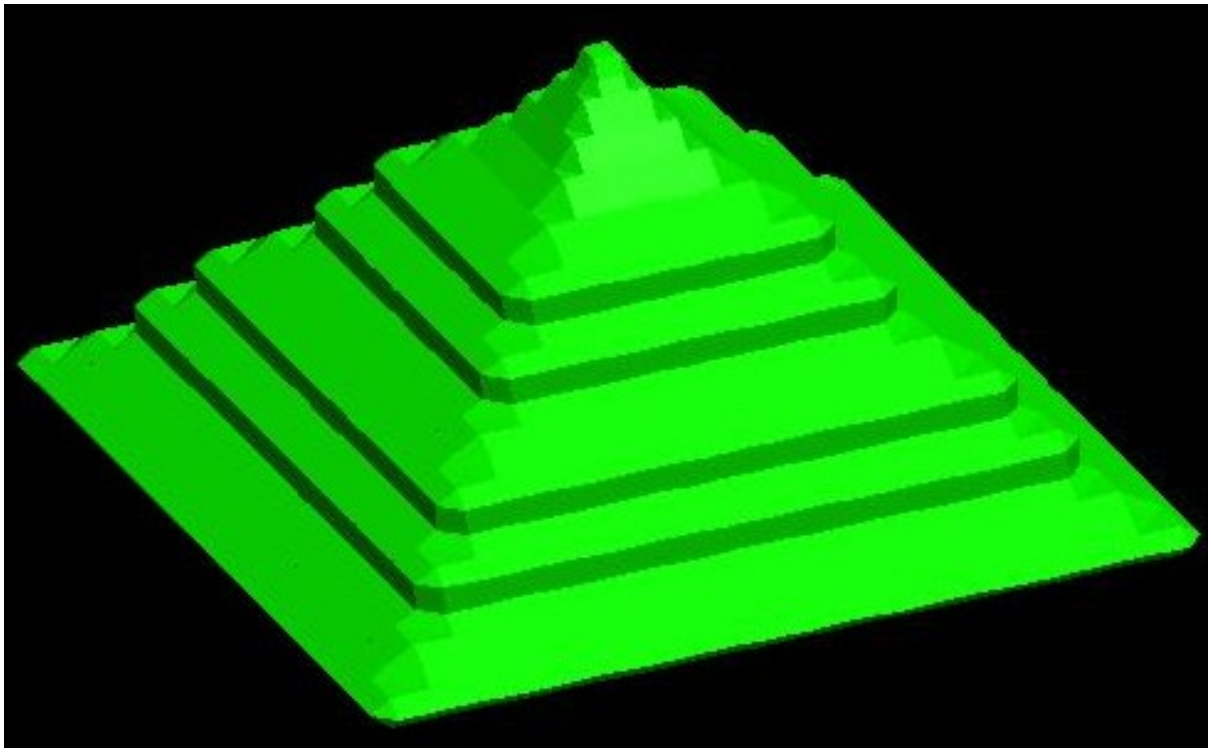
Conduction and valence band profiles

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 6.4.8.24 shows the band profile along the z axis through the center of the QD (x = y = 0 nm), and Figure 6.4.8.25 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.

The figures compare well with Figs. 2(a) and 2(b) of *[Pryor1998]*. However, there are some differences: Due to 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).





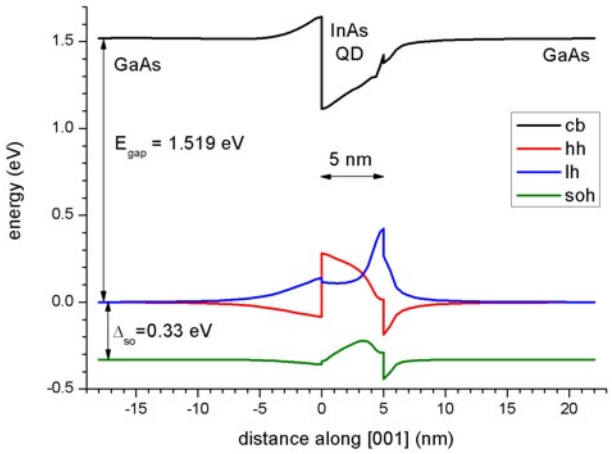


Figure 6.4.8.24: Calculated band edge profile along z axis.

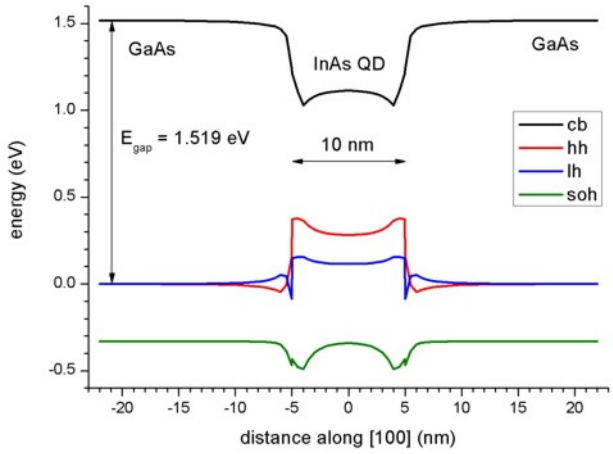


Figure 6.4.8.25: Calculated band edge profile along x axis.

Electron wave function of the ground state (single-band effective-mass approximation)

Figure 6.4.8.26 shows the probability distribution of the electron ground state (The data is contained in *bias_00000\Quantum\probability_shift_dot_Gamma_0001.avs.fld*)

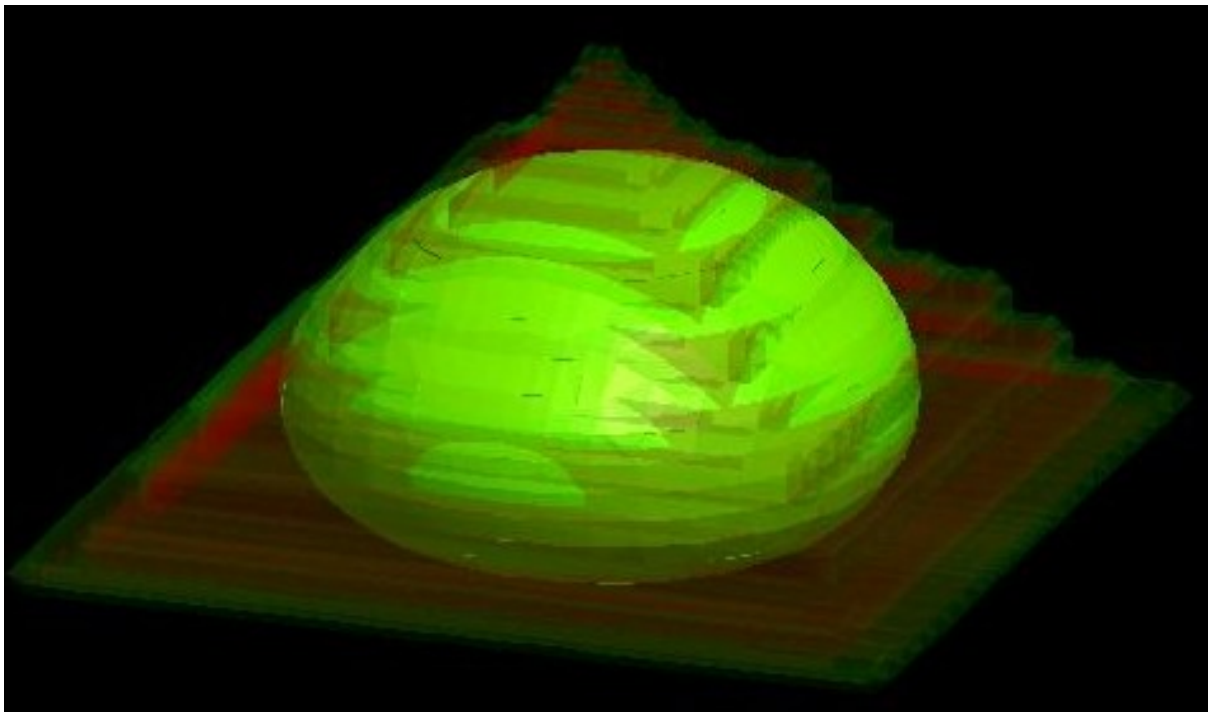


Figure 6.4.8.26: Ψ^2 of the electron ground state inside the quantum dot.

Note: The following sections are preliminary and yet to be updated.

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
-0.58734305
-0.59621434
-0.60757551
-0.62802221
-0.63650764
```

Last update: nn/nn/nnnn

— SOON — Hexagonal shaped GaN quantum dot embedded in AlN (wurtzite)

Attention: Figures in this tutorial are transferred from old documentation and are consistent with *nextnano*[3] input files. 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

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:

- In this tutorial we investigate the influence of strain and pyro-/ piezoelectric fields on the electronic structure of hexagonal shaped GaN/ AlN quantum dots. The tutorial is based on *[Andreev2000]*.

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 6.4.8.27 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.

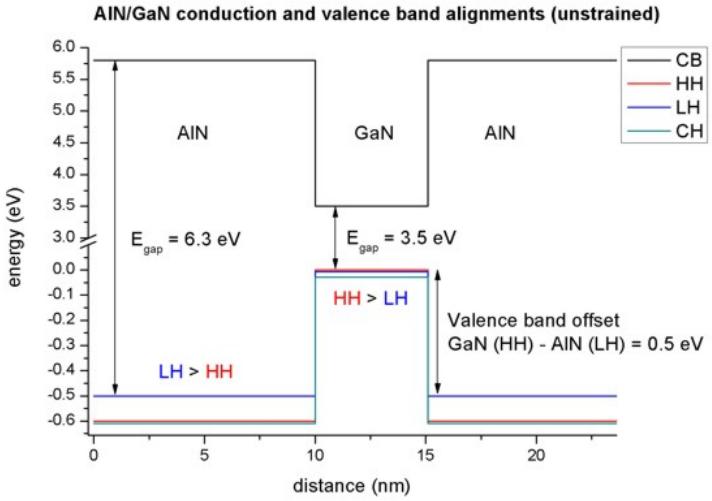


Figure 6.4.8.27: Conduction band edge (CB) and valance band edges (HH, LH, CH) of the 1D AlN/ GaN QD (unstrained).

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 6.4.8.28 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 6.4.8.27, 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.

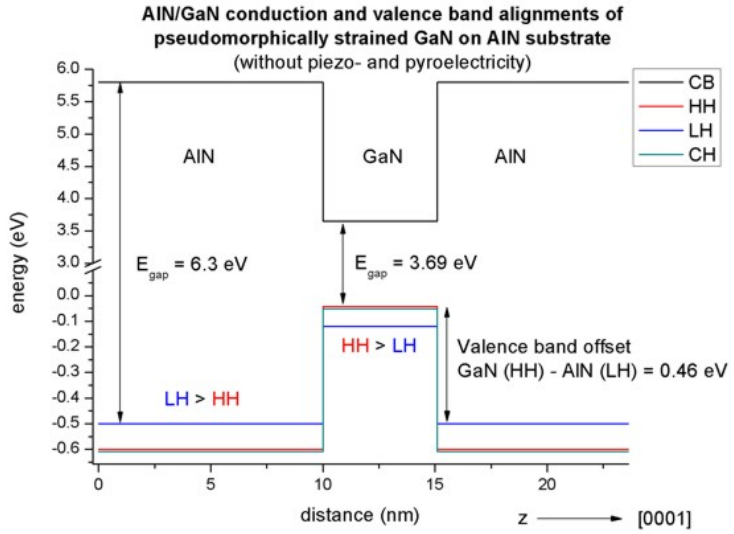


Figure 6.4.8.28: Conduction band edge (CB) and valance band edges (HH, LH, CH) of the 1D AlN/GaN QD (strained on AlN substrate).

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 6.4.8.29 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 6.4.8.29.

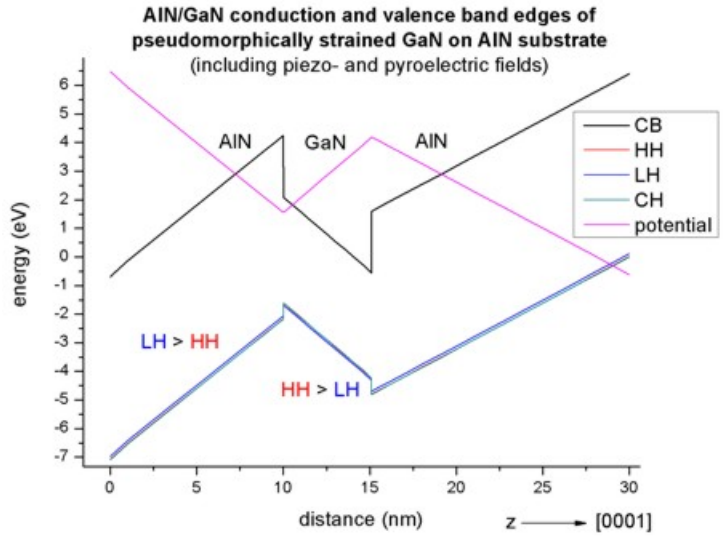


Figure 6.4.8.29: Conduction band edge (CB) and valance band edges (HH, LH, CH) of the 1D AlN/GaN QD (strained on AlN substrate) including piezo- and pyroelectric fields.

Electron and hole wave functions in AlN/GaN QWs

Figure 6.4.8.30 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.

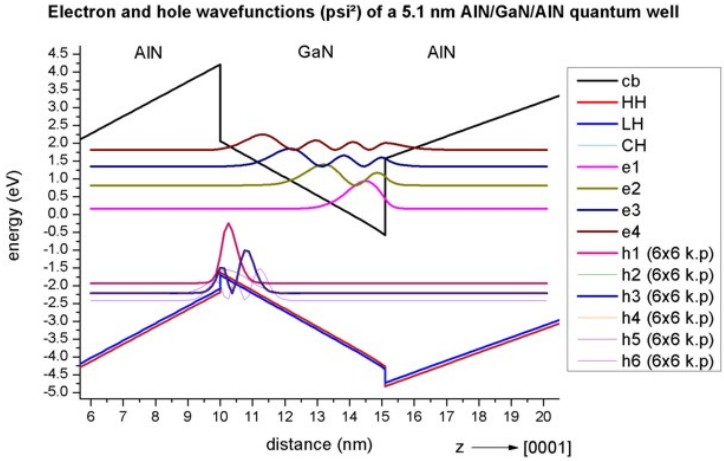


Figure 6.4.8.30: Electron and hole wavefunctions Ψ^2 of a 5.1 mm AlN/GaN/AlN quantum well.

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 6.4.8.31 and Figure 6.4.8.32, respectively.

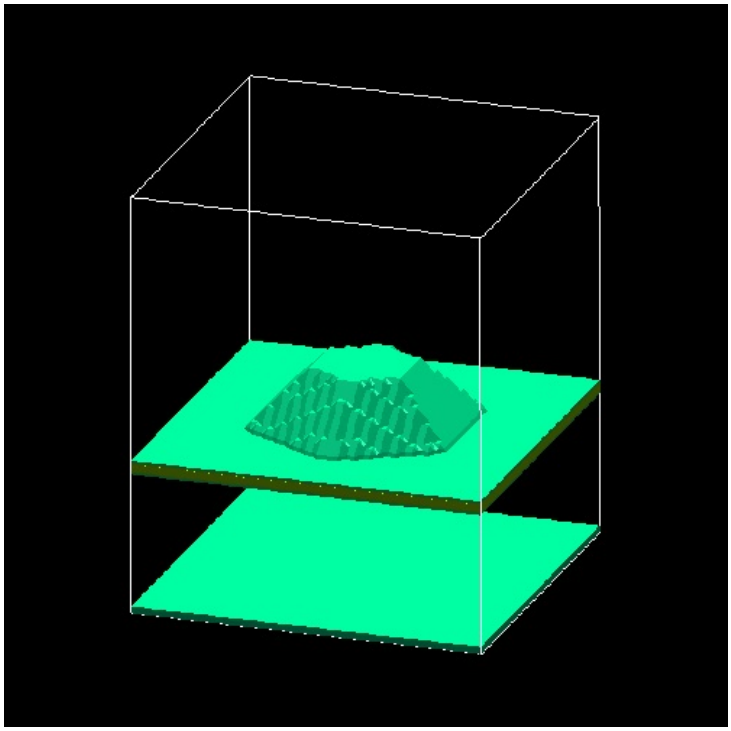


Figure 6.4.8.31: 3D AlN/GaN quantum dot.

The strain tensor components of a line through the center of the quantum dot along the z axis are shown in Figure 6.4.8.33. This figure is related to Fig. 2 (a) in *[Andreev2000]*.

Figure 6.4.8.34 and Figure 6.4.8.35 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]*.

The strain induced piezoelectric fields and the pyroelectric fields lead to the electrostatic potential which is shown in Figure 6.4.8.36 and Figure 6.4.8.37. The figures of the potential are related to Fig. 4 in *[Andreev2000]*. In Figure 6.4.8.36 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 6.4.8.37 shows a cut of the electrostatic potential through the wetting layer plane.

The conduction and valence band edges are shown in Figure 6.4.8.38 and Figure 6.4.8.39. 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 6.4.8.40 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 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]*.

Version of this tutorial for *nextnano*^{3} can be found here .

Last update: 11/07/2024

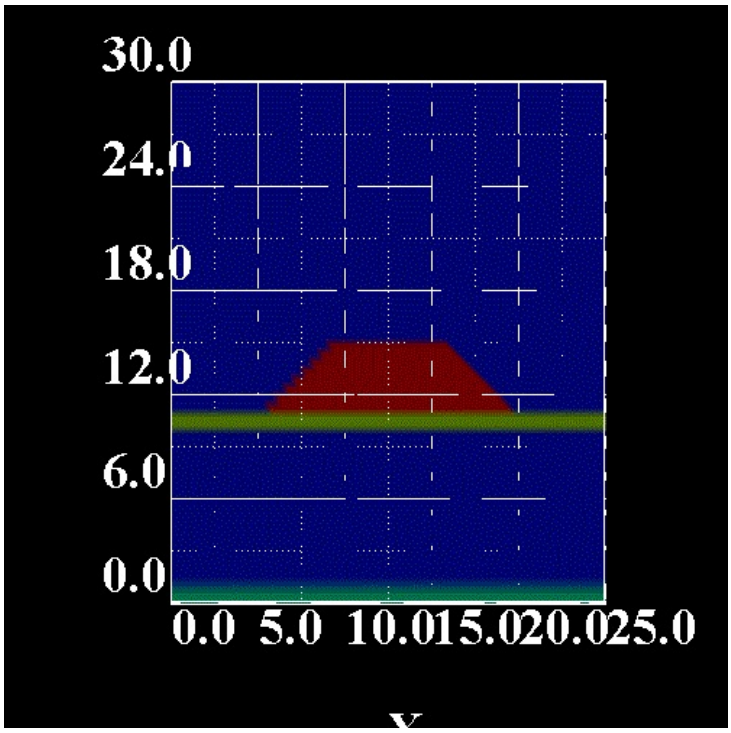


Figure 6.4.8.32: Cross-section of the hexagonal shaped AlN/GaN quantum dot.

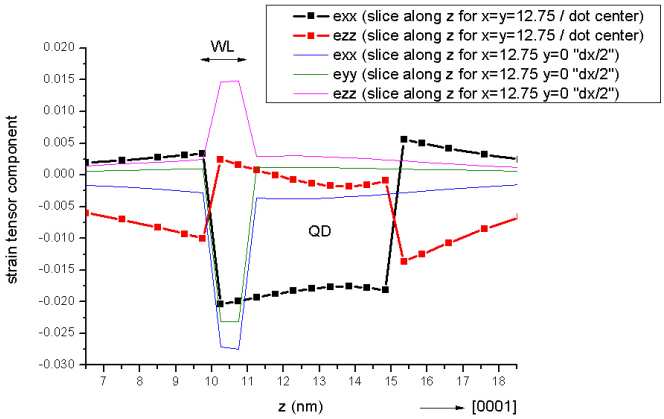


Figure 6.4.8.33: Strain tensor along the z-axis through the points (x, y) = (12.75, 12.75) nm and (x, y) = (12.75, 0) nm.

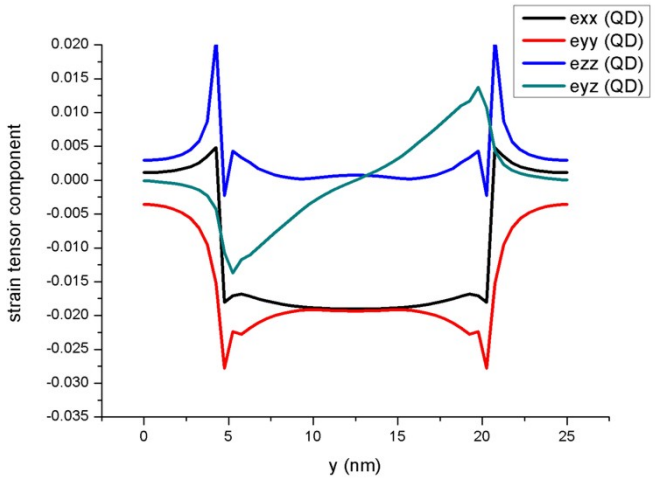


Figure 6.4.8.34: Strain tensor along the y-axis through the quantum dot (QD).

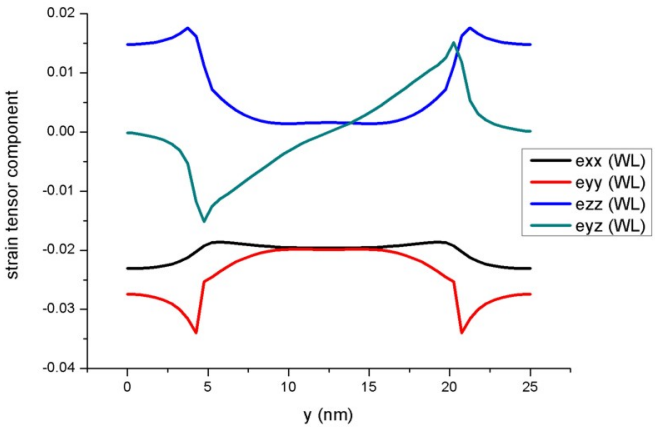


Figure 6.4.8.35: Strain tensor along the y-axis through the wetting layer (WL).

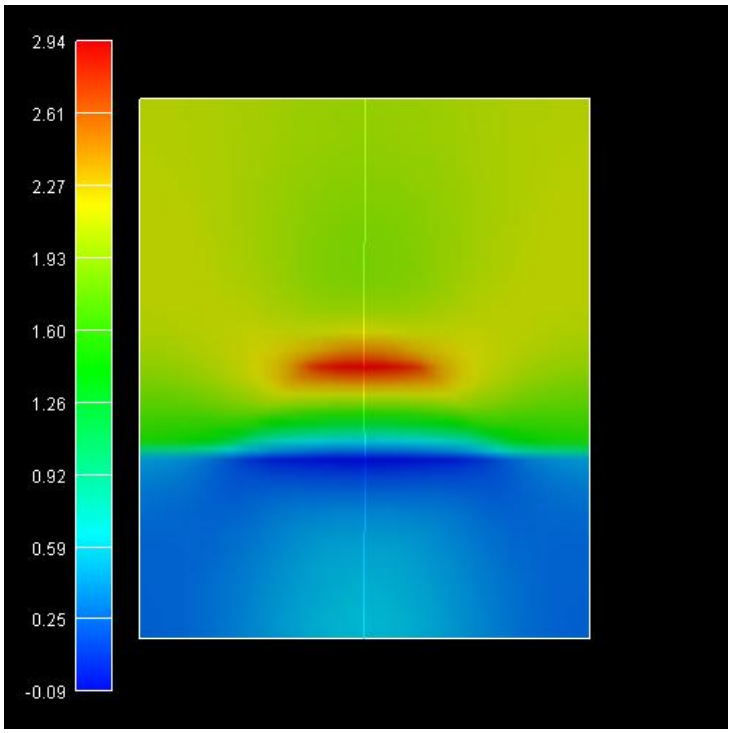


Figure 6.4.8.36: Electrostatic potential inside the quantum dot.

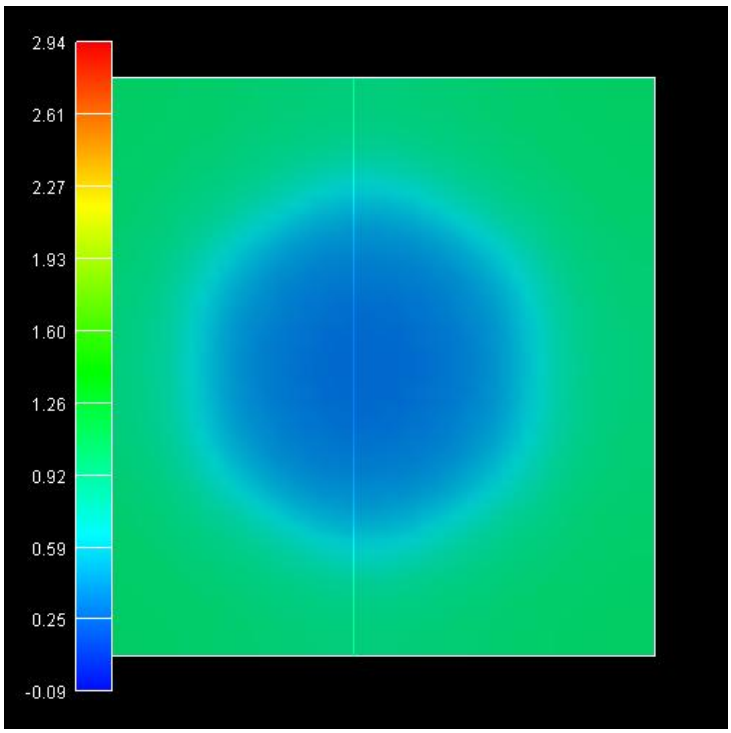


Figure 6.4.8.37: Electrostatic potential inside the wetting layer.

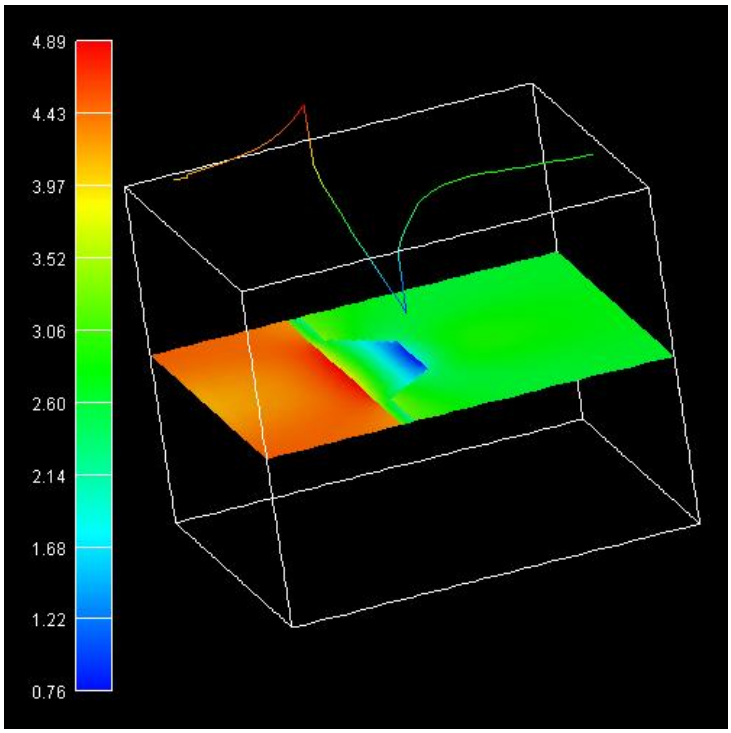


Figure 6.4.8.38: Conduction band edge of the QD.

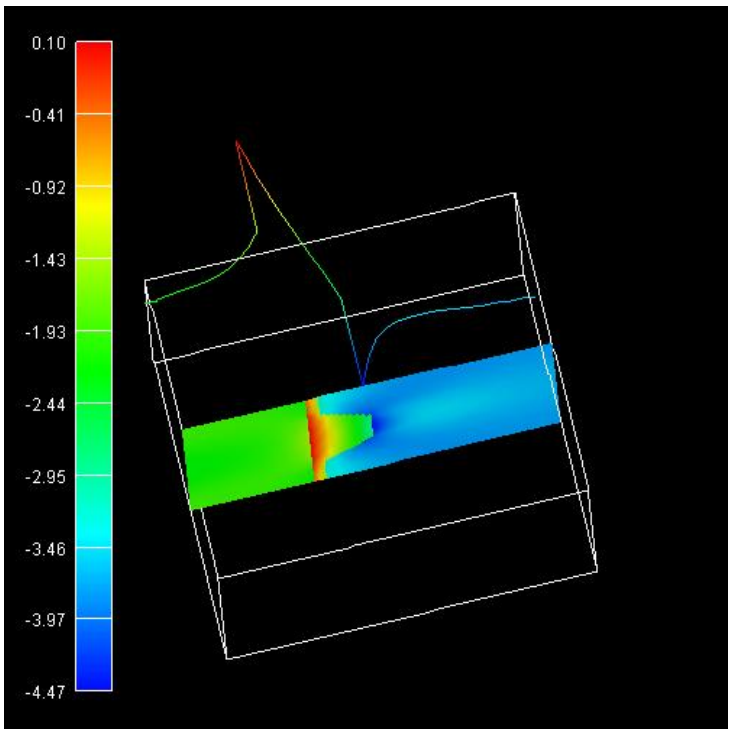


Figure 6.4.8.39: Valence band edge of the QD.

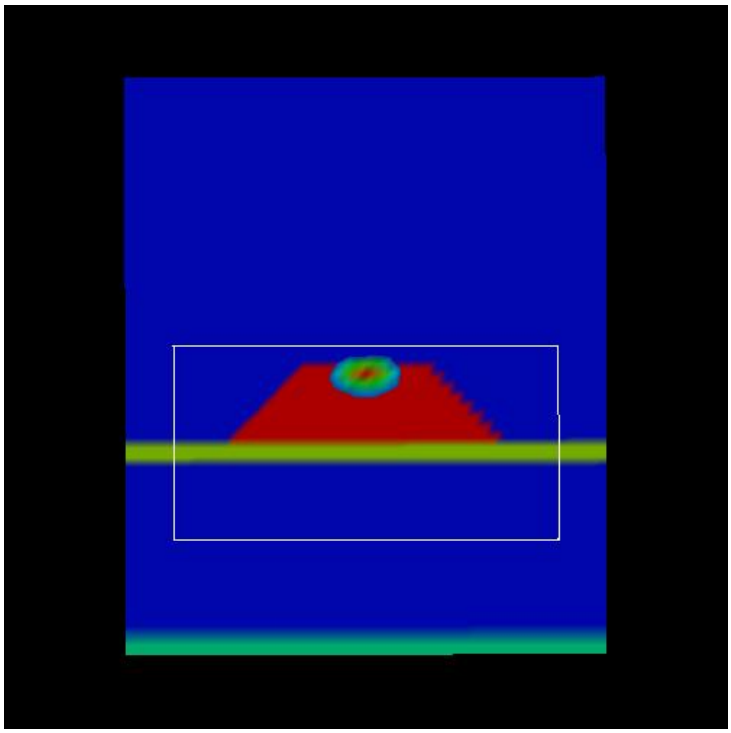


Figure 6.4.8.40: Electron ground state of the QD.

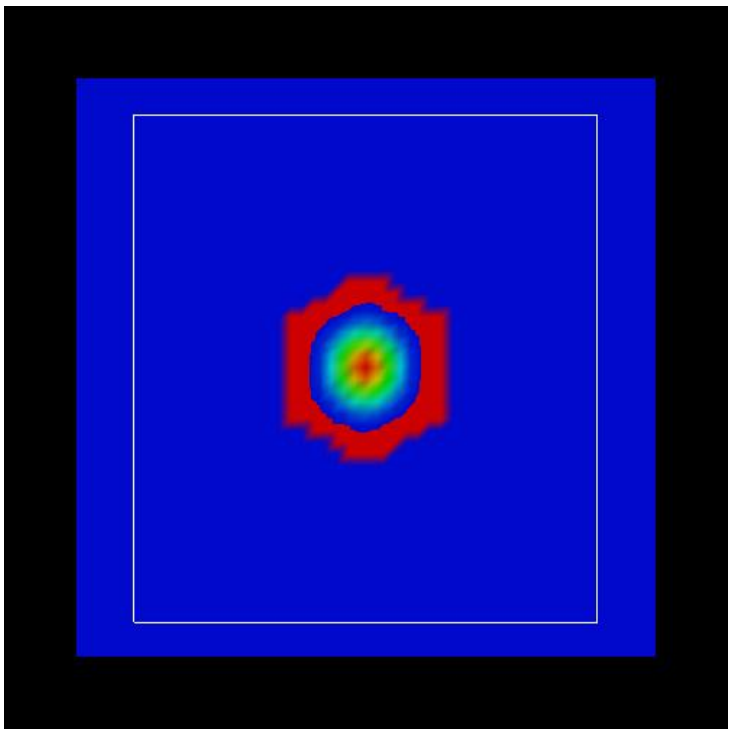


Figure 6.4.8.41: Probability density Ψ^2 of the 1^{st} electron state in the QD.

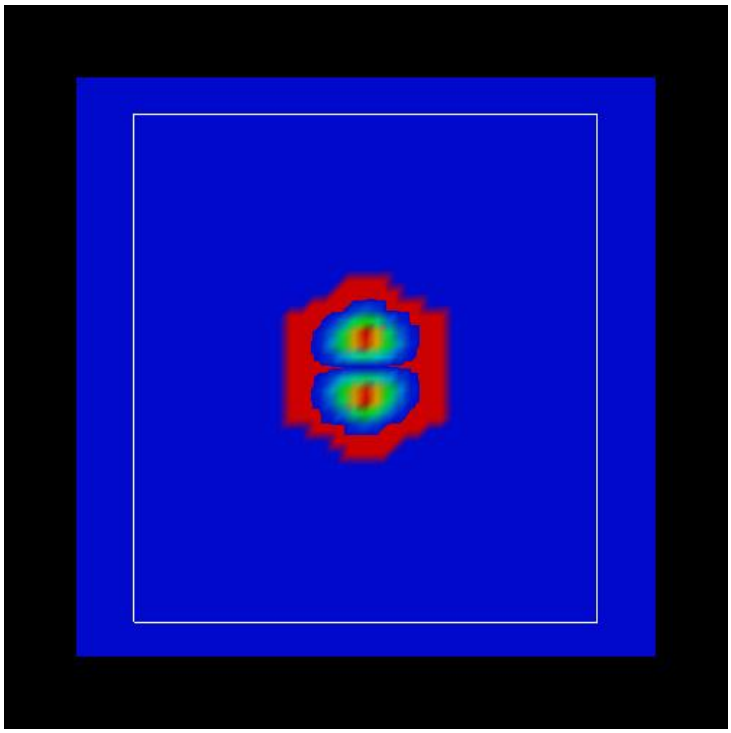


Figure 6.4.8.42: Probability density Ψ^2 of the 2^{nd} electron state in the QD.

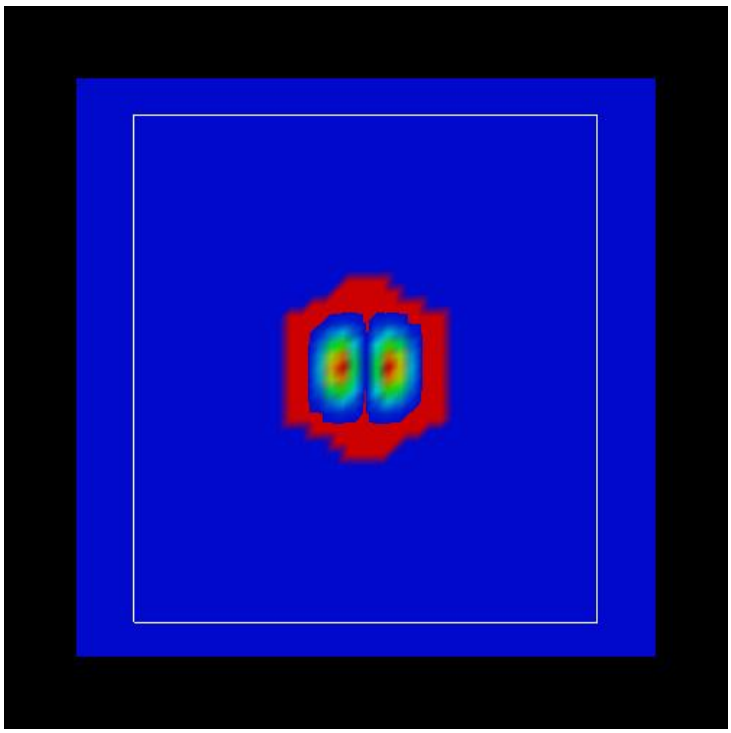


Figure 6.4.8.43: Probability density Ψ^2 of the 3^{rd} electron state in the QD.

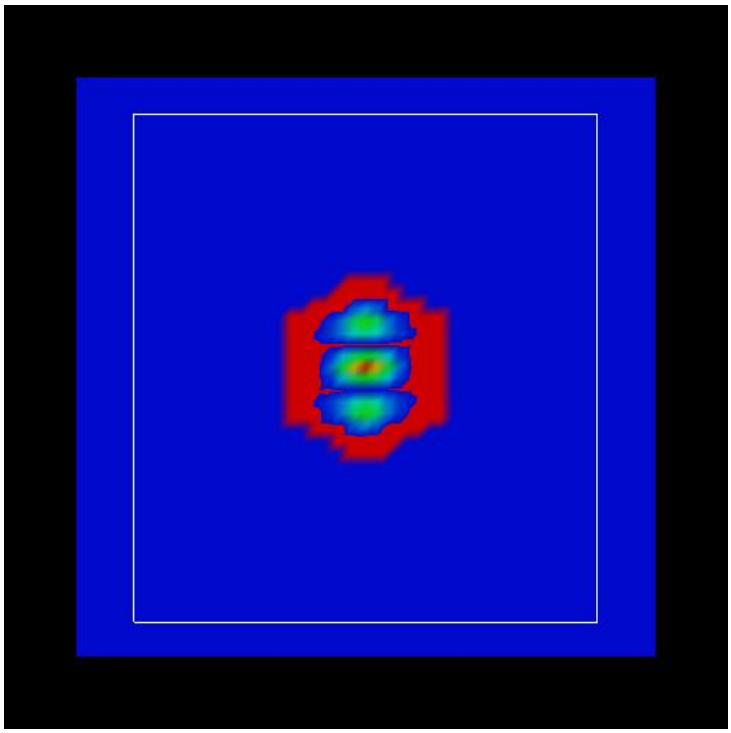


Figure 6.4.8.44: Probability density Ψ^2 of the 4^{th} electron state in the QD.

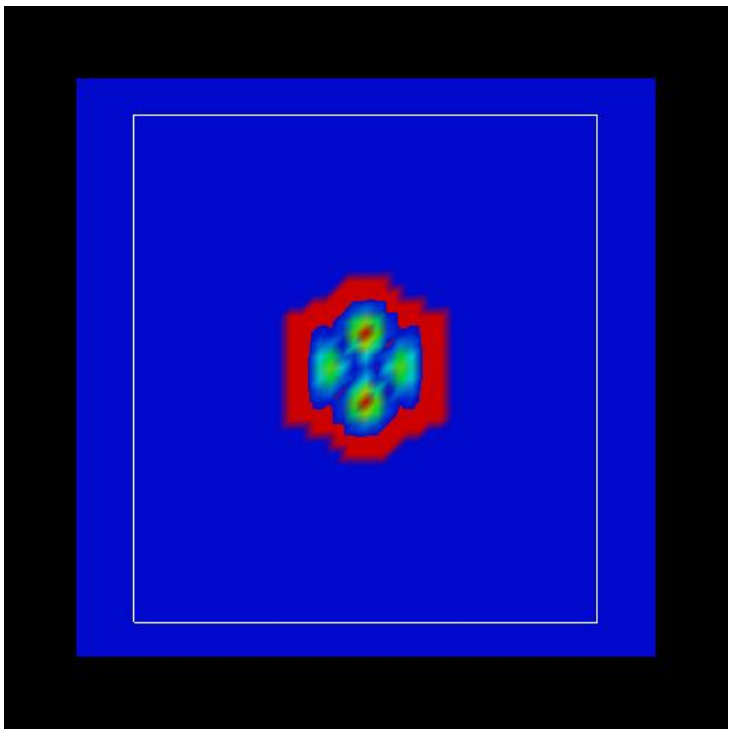


Figure 6.4.8.45: Probability density Ψ^2 of the 5^{th} electron state in the QD.

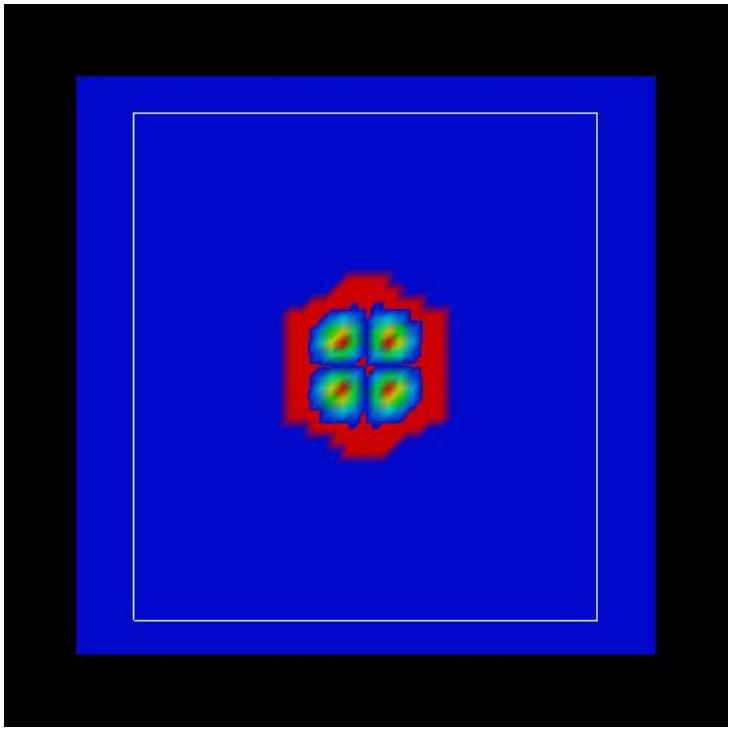


Figure 6.4.8.46: Probability density Ψ^2 of the 6^{th} electron state in the QD.

— 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\interband_matrix_elements_quantum_region_HH_Gamma.txt*

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 $r = 5$ nm shown in Figure 6.4.8.47.

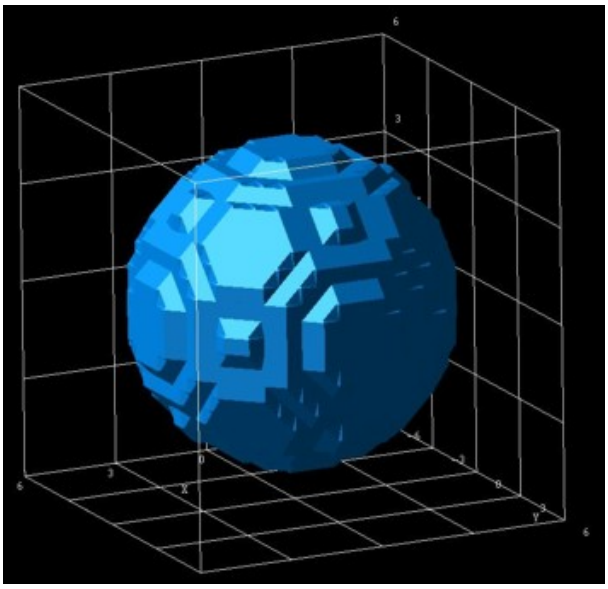


Figure 6.4.8.47: Spherical quantum dot.

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 6.4.8.48 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.

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 (interband matrix elements). 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 6.4.8.49 shows the calculated interband matrix elements as a function of energy. (Note: The figure has to be updated: Now we output the square of this matrix element.)

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 ($r_x = 5$ nm, $r_y = 5$ nm, $r_z = 10$nm), we calculate the lowest 30 eigenvalues.

The energy spectrum (degeneracy spectrum) looks very different from the spherical QD spectrum (c.f. Figure 6.4.8.51)

The interband matrix elements are shown in Figure 6.4.8.52 (Note: The figure has to be updated: Now we output the square of this matrix element.)

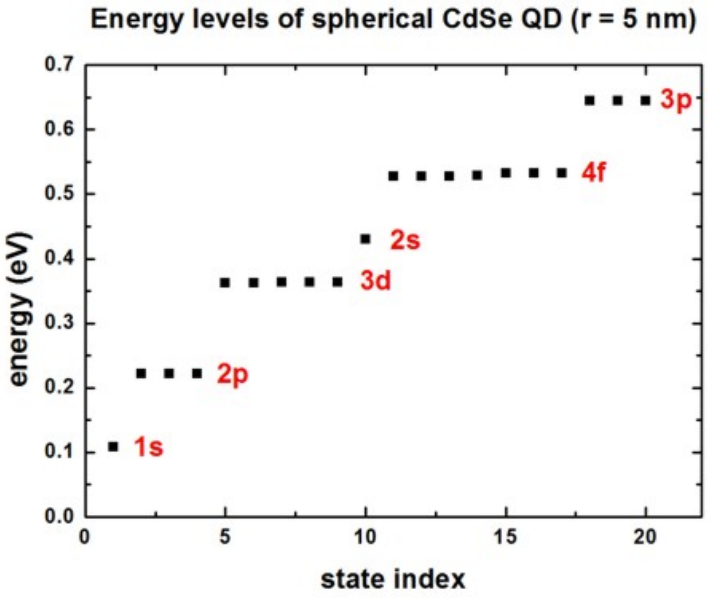


Figure 6.4.8.48: Eigenenergies of the lowest 20 states in the QD.

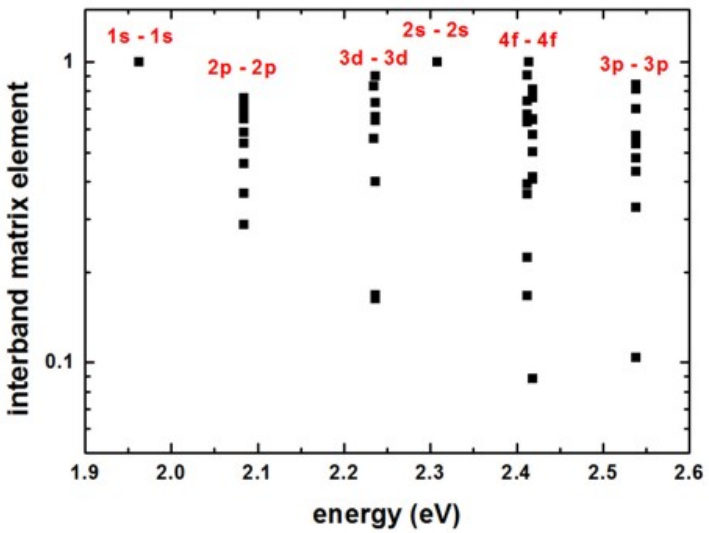


Figure 6.4.8.49: Interband matrix elements.

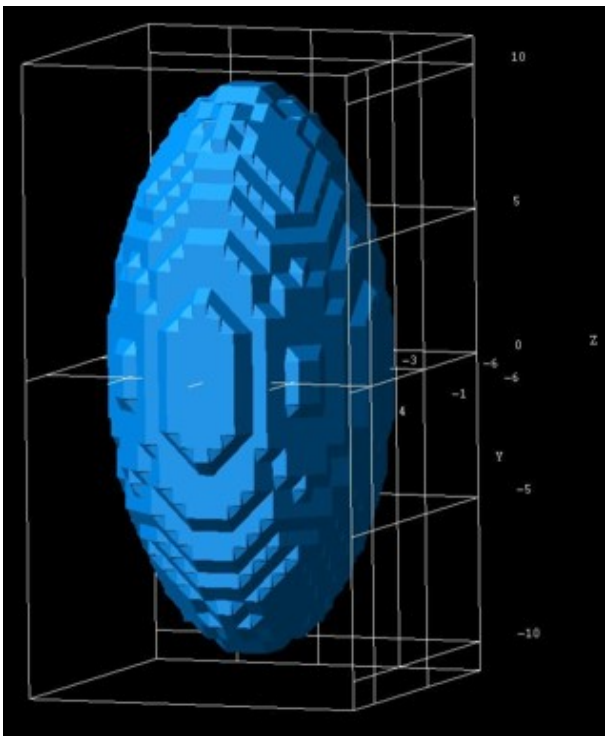


Figure 6.4.8.50: Ellipsoidal quantum dot.

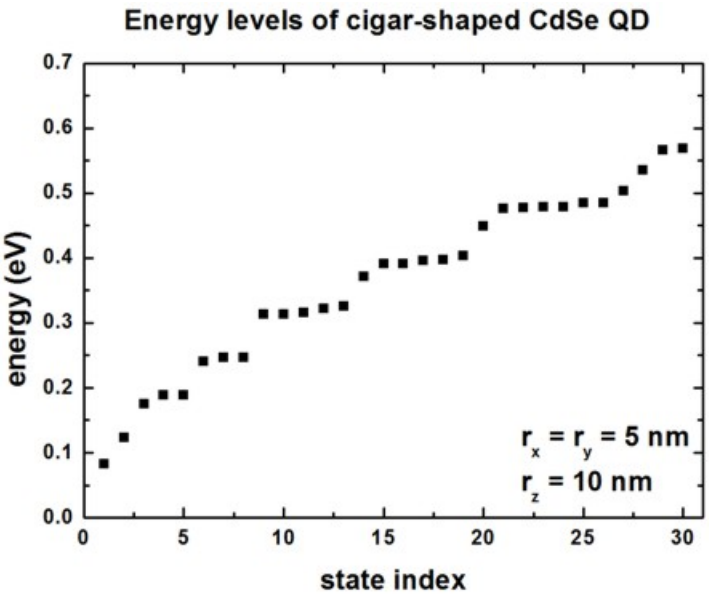


Figure 6.4.8.51: Eigenenergies of the lowest 30 states in the QD.

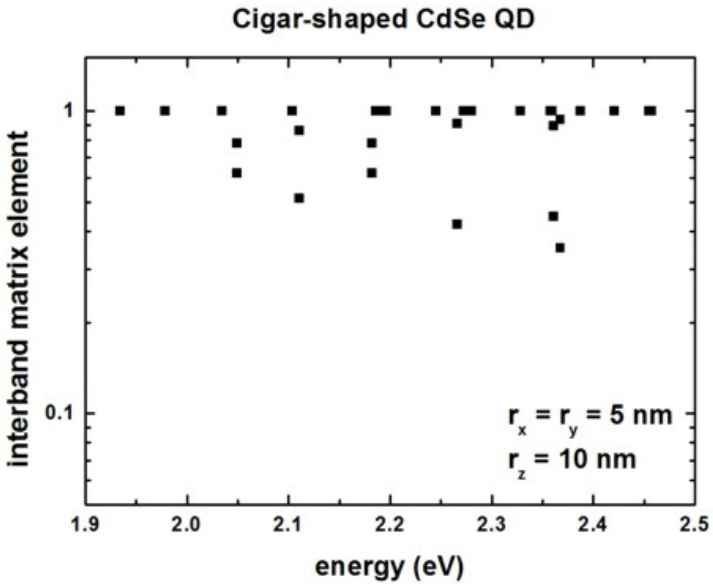


Figure 6.4.8.52: Interband matrix elements.

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.)

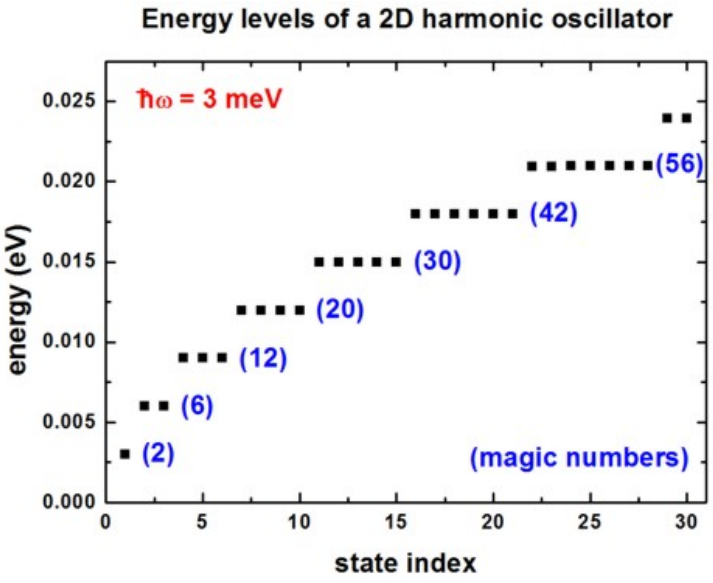


Figure 6.4.8.53: Eigenenergies of the lowest 30 states in a harmonic potential.

Version of this tutorial for *nextnano*[3] can be found here .

Last update: 17/07/2024

6.4.9 Electronic Band Structures

k.p dispersion in bulk GaAs (strained / unstrained)

Input files:

- *bulk_kp_dispersion_GaAs_nnp.in*
- *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 6.4.9.1 visualizes the results.

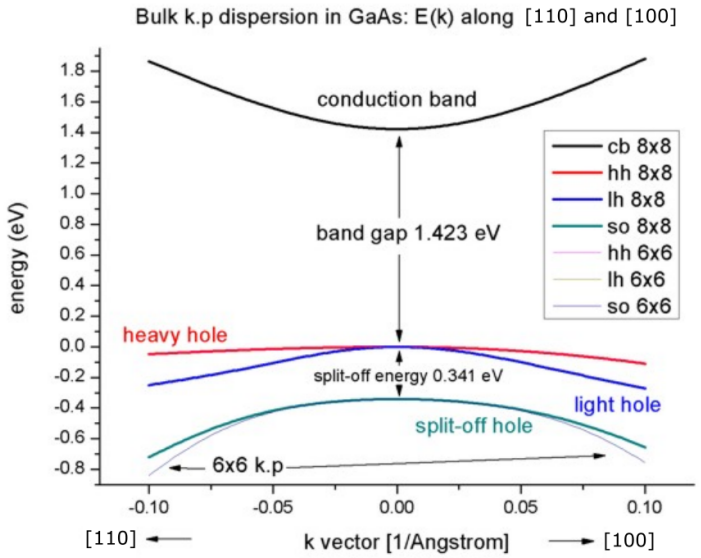


Figure 6.4.9.1: Bulk k.p dispersion in $GaAs$: $E(k)$ along [100] and [110].

The split-off energy of 0.341 eV is identical to the split-off energy as defined in the database:

```
...
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 6.4.9.2.

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 6.4.9.3.

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{ }
}
```

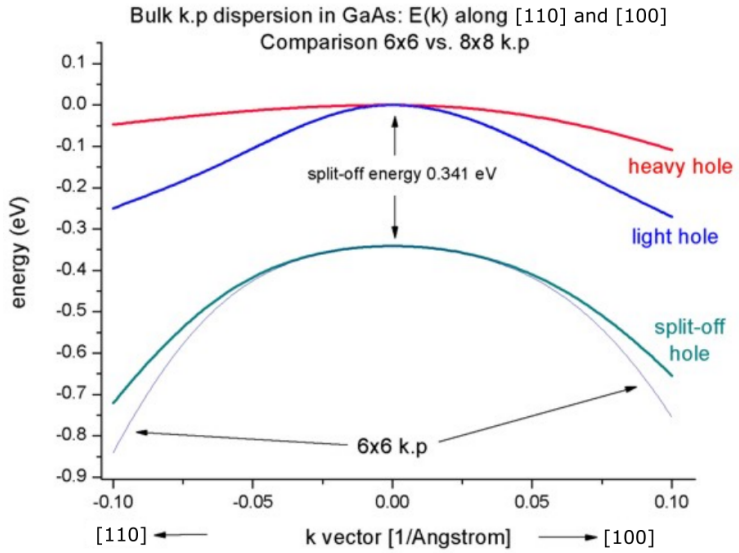


Figure 6.4.9.2: Bulk k.p dispersion in GaAs: $E(k)$ along [100] and [110] - Comparision between 6x6 and 8x8 k.p

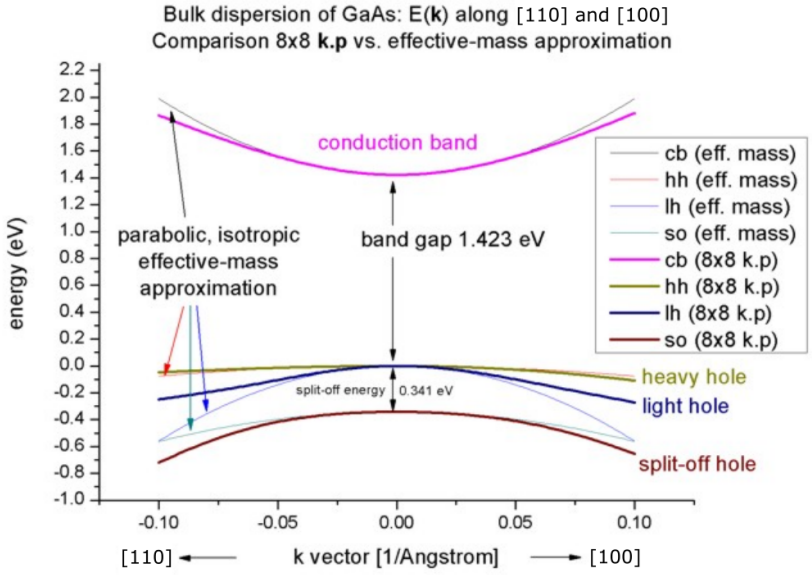


Figure 6.4.9.3: Bulk k.p dispersion in $GaAs$: $E(k)$ along [100] and [110] - Comparision between 8x8 k.p and effective-mass approximation

```
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]).

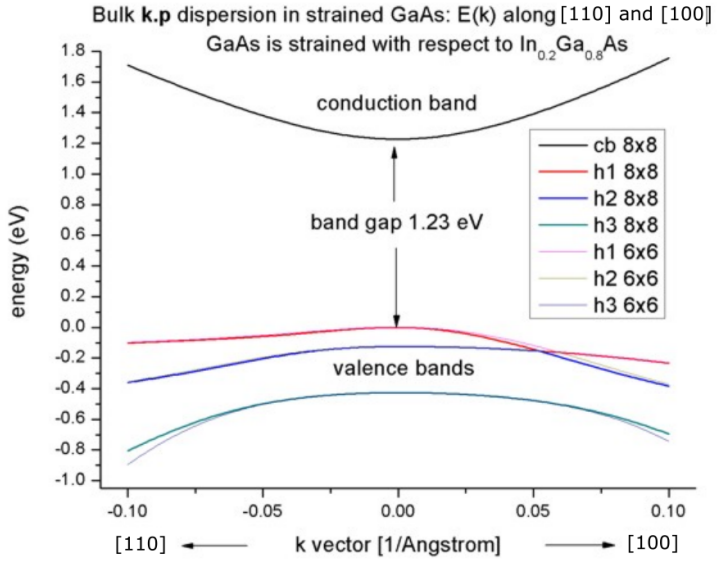


Figure 6.4.9.4: Bulk k.p dispersion in $GaAs$ strained with respect to $In_{0.2}Ga_{0.8}As$: $E(k)$ along [100] and [110].

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 6.4.9.5.

Note that in the strained case, the effective-mass approximation is very poor.

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
```

(continues on next page)

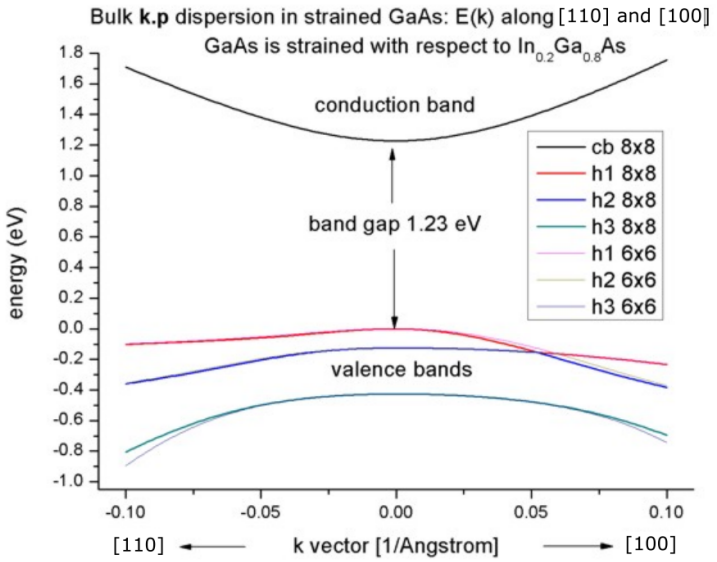


Figure 6.4.9.5: Bulk valence band k.p dispersion in $GaAs$ strained with respect to $In_{0.2}Ga_{0.8}As$: $E(k)$ along [100] and [110] - Comparision between 6x6 and 8x8 k.p approximation.

(continued from previous page)

```
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	$\|S\downarrow\rangle$
2	$\|S\uparrow\rangle$

- The hole eigenstates are 4-fold (heavy and light holes) and 2-fold degenerate (split-off holes).

3	$\left\|\frac{3}{2},\frac{3}{2}\right\rangle$ hh spin up	$\frac{1}{\sqrt{2}}\|(X+iY)\uparrow\rangle$
4	$\left\|\frac{3}{2},\frac{1}{2}\right\rangle$ lh	$\frac{1}{\sqrt{6}}\|(X+iY)\downarrow\rangle - \sqrt{\frac{2}{3}}\|Z\uparrow\rangle$
5	$\left\|\frac{3}{2},-\frac{1}{2}\right\rangle$ lh	$\frac{1}{\sqrt{6}}\|(X-iY)\uparrow\rangle - \sqrt{\frac{2}{3}}\|Z\downarrow\rangle$
6	$\left\|\frac{3}{2},-\frac{3}{2}\right\rangle$ hh spin down	$\frac{1}{\sqrt{2}}\|(X-iY)\downarrow\rangle$
7	$\left\|\frac{1}{2},\frac{1}{2}\right\rangle$ s/o split	$\frac{1}{\sqrt{3}}\|(X+iY)\downarrow\rangle - \frac{1}{\sqrt{3}}\|Z\uparrow\rangle$
8	$\left\|\frac{1}{2},-\frac{1}{2}\right\rangle$ s/o split	$\frac{1}{\sqrt{3}}\|(X-iY)\downarrow\rangle - \frac{1}{\sqrt{3}}\|Z\downarrow\rangle$

$\frac{1}{\sqrt{2}} = 0.707 \rightarrow \left(\frac{1}{2}\right)^2 = 0.5$
$\frac{1}{\sqrt{3}} = 0.577 \rightarrow \left(\frac{1}{3}\right)^2 = 0.333$
$\frac{1}{\sqrt{6}} = 0.408 \rightarrow \left(\frac{1}{6}\right)^2 = 0.166$

Last update: nn/nn/nnnn

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 6.4.9.6 shows the bulk k.p dispersion of unstrained GaN (wurtzite). The results are in excellent agreement to Fig. 4 (b) of *[KumagaiChuangAndoPRB1998]*.

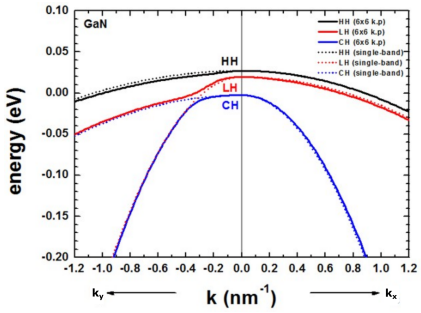


Figure 6.4.9.6: Calculated 1-band and k.p dispersion of HH, LH and CH valence bands (unstrained). The k_x direction corresponds to the c axis [0001]. The dispersion along k_y and k_z is identical (only k_y is shown), i.e. the dispersion in the (100) plane is isotropic.

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 6.4.9.7) vs. tensely (+0.5%, Figure 6.4.9.8) 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)

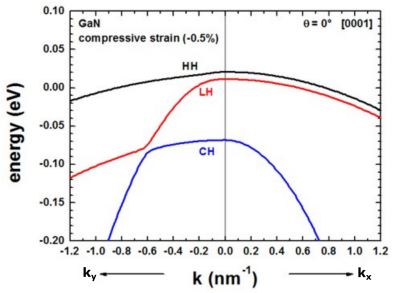


Figure 6.4.9.7: Calculated k.p dispersion of HH, LH and CH valence bands (compressive strain)

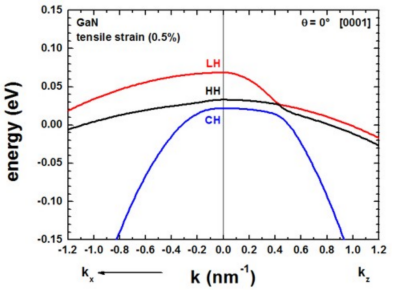


Figure 6.4.9.8: Calculated k.p dispersion of HH, LH and CH valence bands (tensile strain)

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.

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.

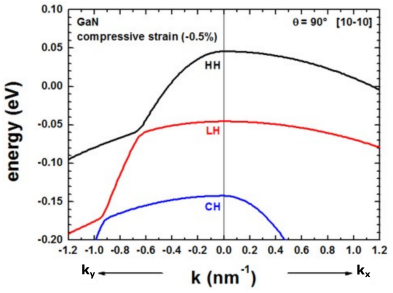


Figure 6.4.9.9: Calculated k.p dispersion of HH, LH and CH valence bands (compressive strain)

The results of Figure 6.4.9.9 and Figure 6.4.9.11 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 }
```

(continues on next page)

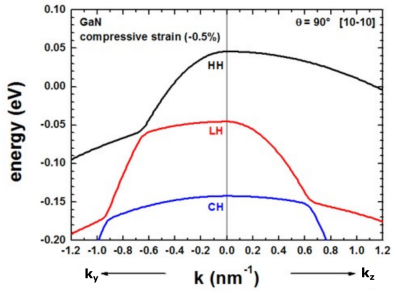


Figure 6.4.9.10: Calculated k.p dispersion of HH, LH and CH valence bands (compressive strain)

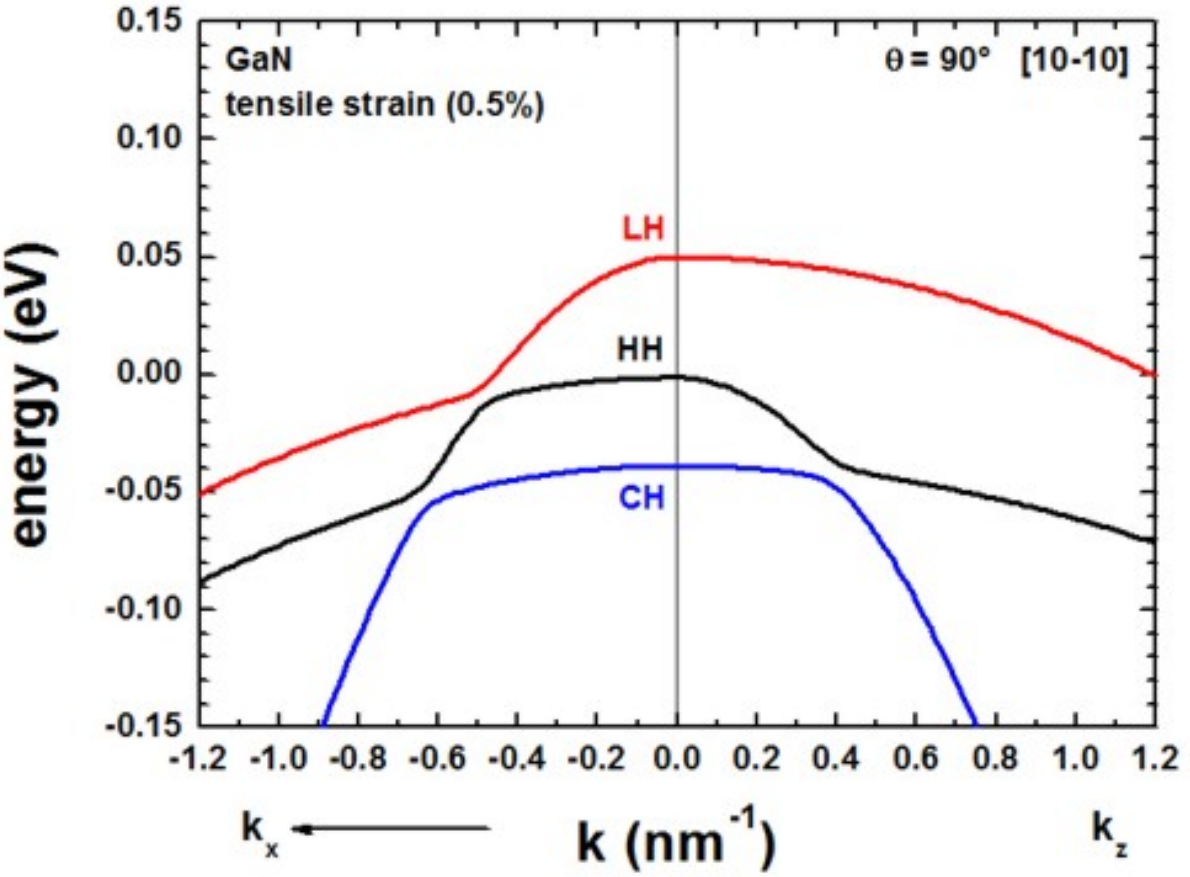


Figure 6.4.9.11: Calculated k.p dispersion of HH, LH and CH valence bands (tensile strain)

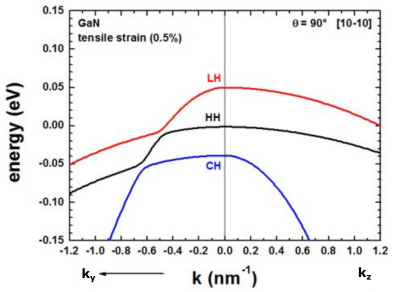


Figure 6.4.9.12: Calculated k.p dispersion of HH, LH and CH valence bands (tensile strain)

(continued from previous page)

```
        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 $\theta = 90°$, 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]*.

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 6.4.9.13 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).

Last update: nn/nn/nnnn

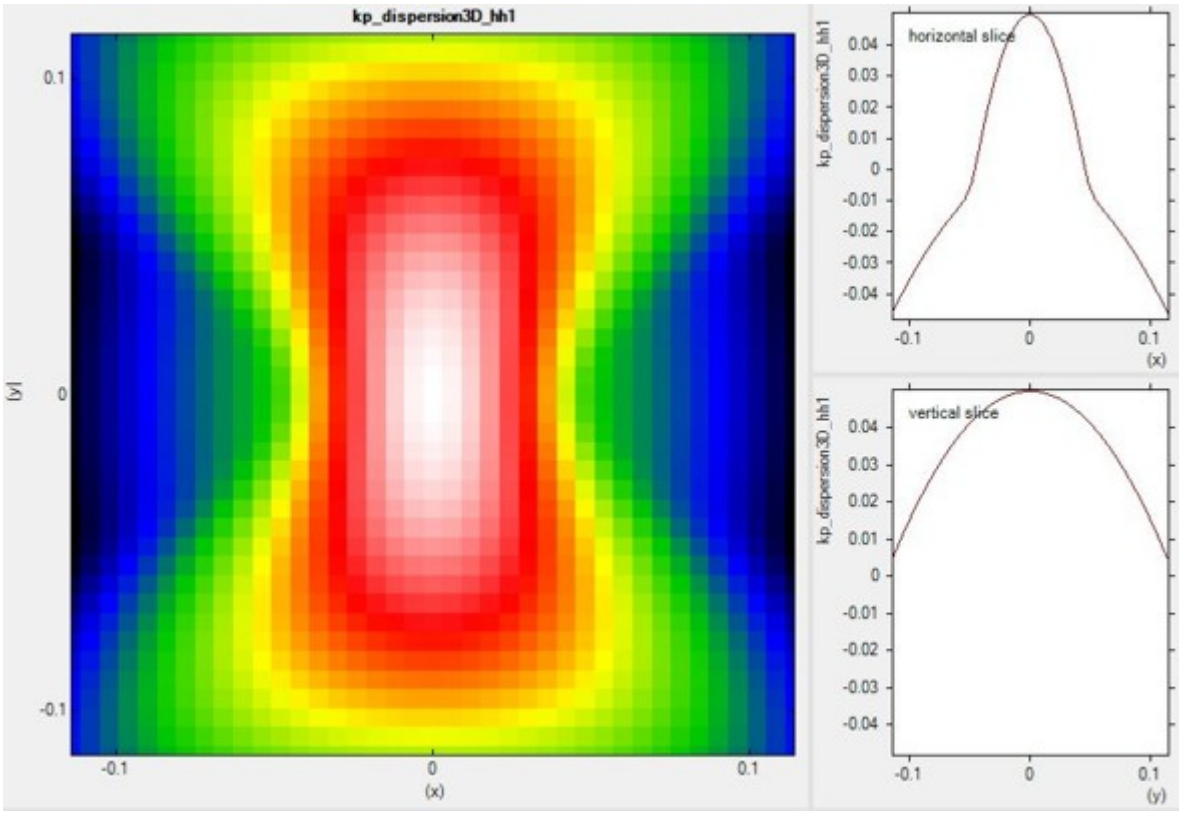


Figure 6.4.9.13: 2D slice at k_x = 0 of calculated 3D dispersion.

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 6.4.9.14, Figure 6.4.9.15 and Figure 6.4.9.16. The final figures will look like this (left: dispersion along [0001], right: dispersion along [110]):

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)
}
```

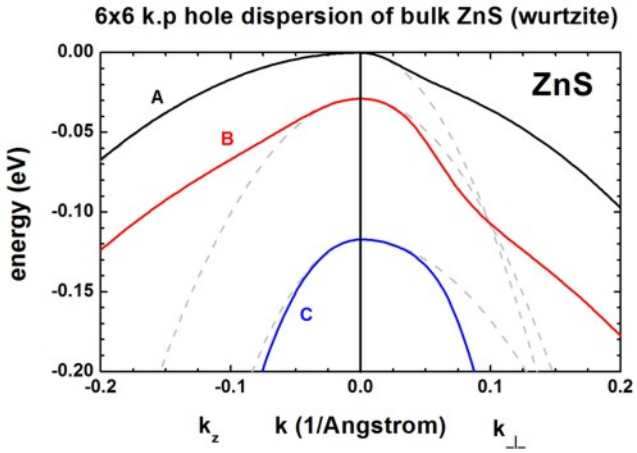


Figure 6.4.9.14: Calculated 1-band (dotted gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

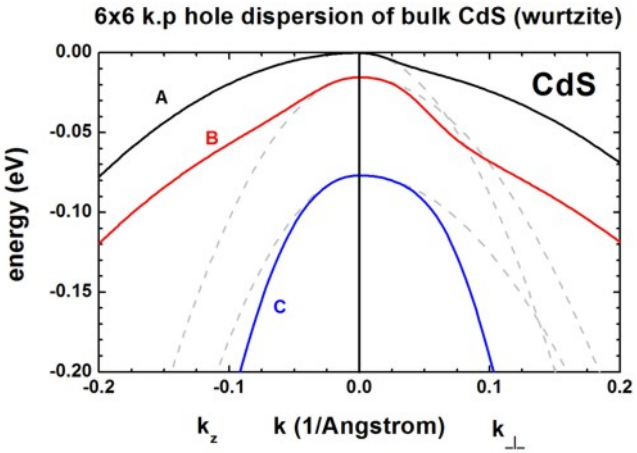


Figure 6.4.9.15: Calculated 1-band (dotted gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

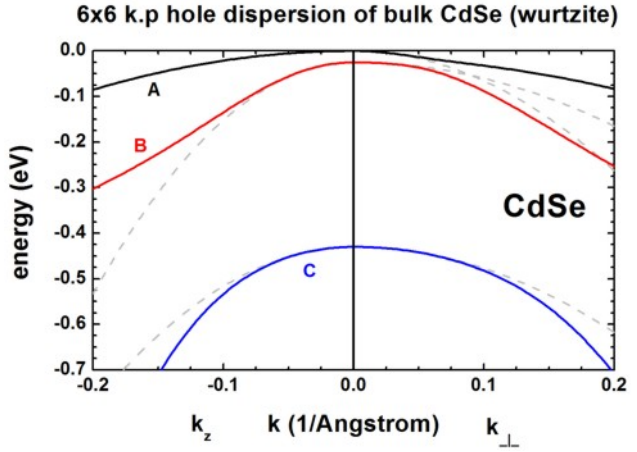


Figure 6.4.9.16: Calculated 1-band (dotted gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

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 6.4.9.17 shows the bulk 6-band k.p energy dispersion for ZnO. The gray lines are the dispersions assuming a parabolic effective mass.

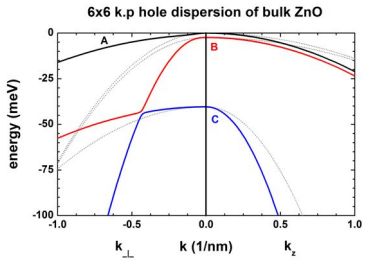


Figure 6.4.9.17: Calculated parabolic effective mass (dotted, gray) and k.p dispersion of HH (A, black), LH (B, red) and CH (C, blue) valence bands (unstrained).

The following files are plotted:

- *bulk_6x6kp_dispersion_as_in_inputfile_kxkykz_000_kxkykz.dat*
- *bulk_sg_dispersion.dat*

The files

- *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: nn/nn/nnnn

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 6.4.9.18 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.

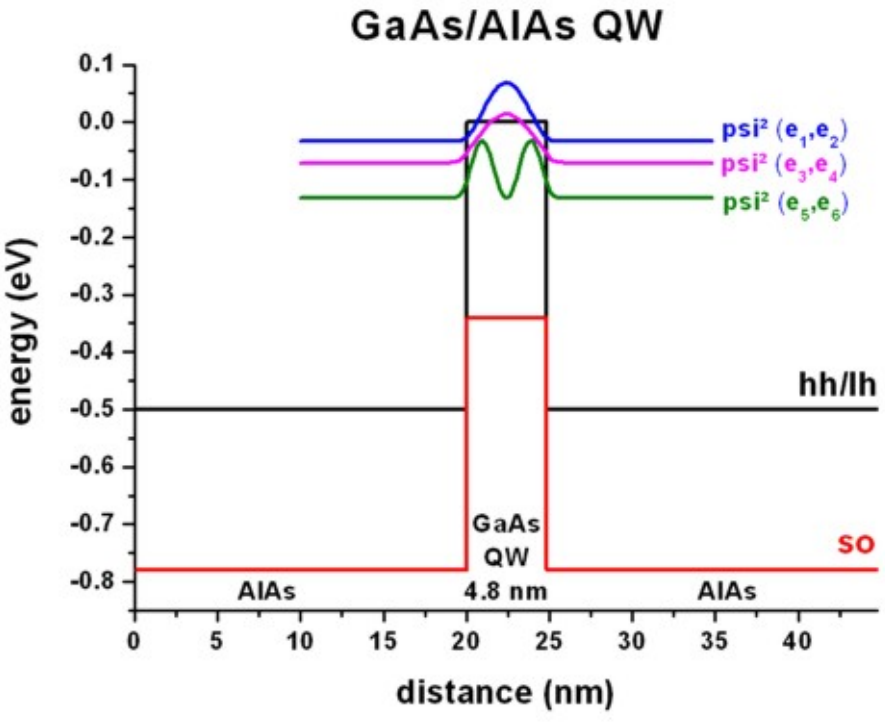


Figure 6.4.9.18: Calculated valence band edges with Ψ^2 of the lowest hole states.

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
            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]
        }

        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]
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

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 6.4.9.19, 3 and 4 to Figure 6.4.9.20 and 5 and 6 to Figure 6.4.9.21. 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.

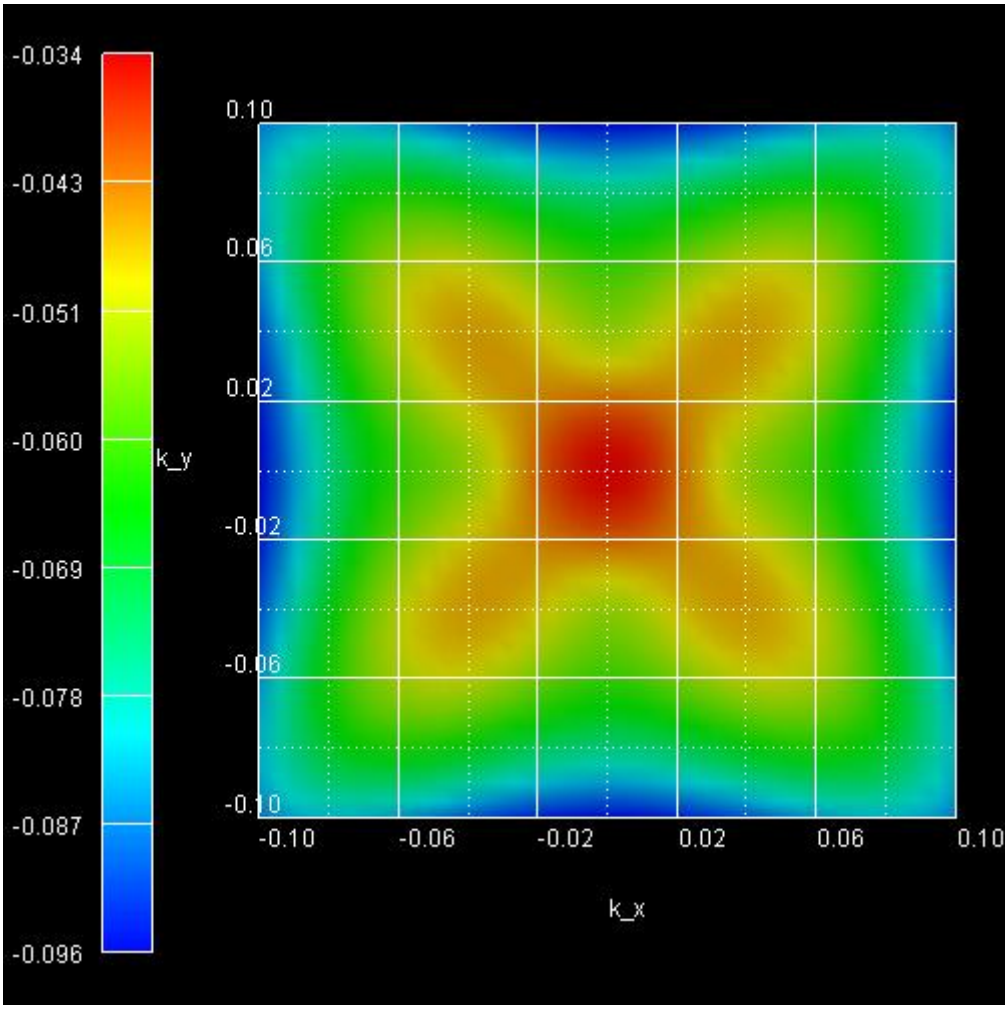


Figure 6.4.9.19: Subband 1 (eigenvalue 1 and 2)

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 6.4.9.22. This plot was obtained by plotting the following file: *dispersion_quantum_region_kp6_kpar_10_00_11.dat*.

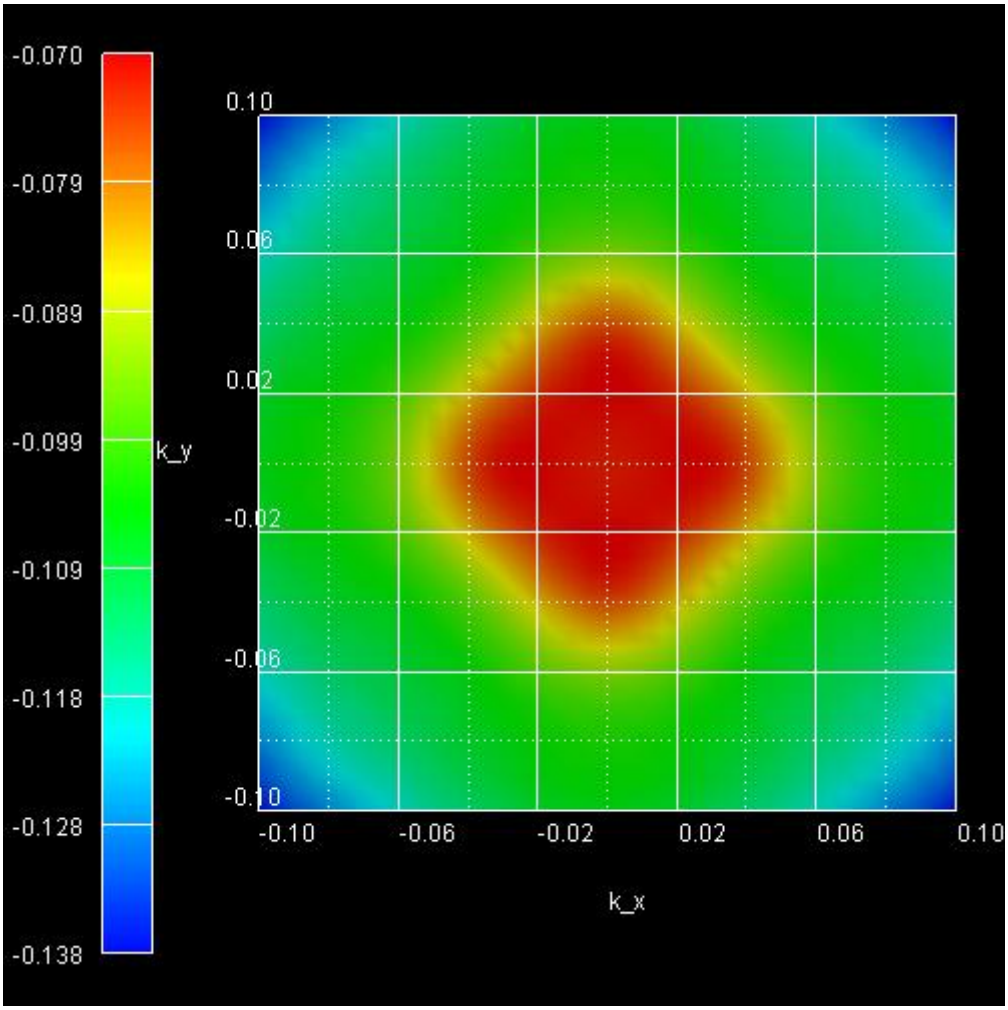


Figure 6.4.9.20: Subband 2 (eigenvalue 3 and 4)

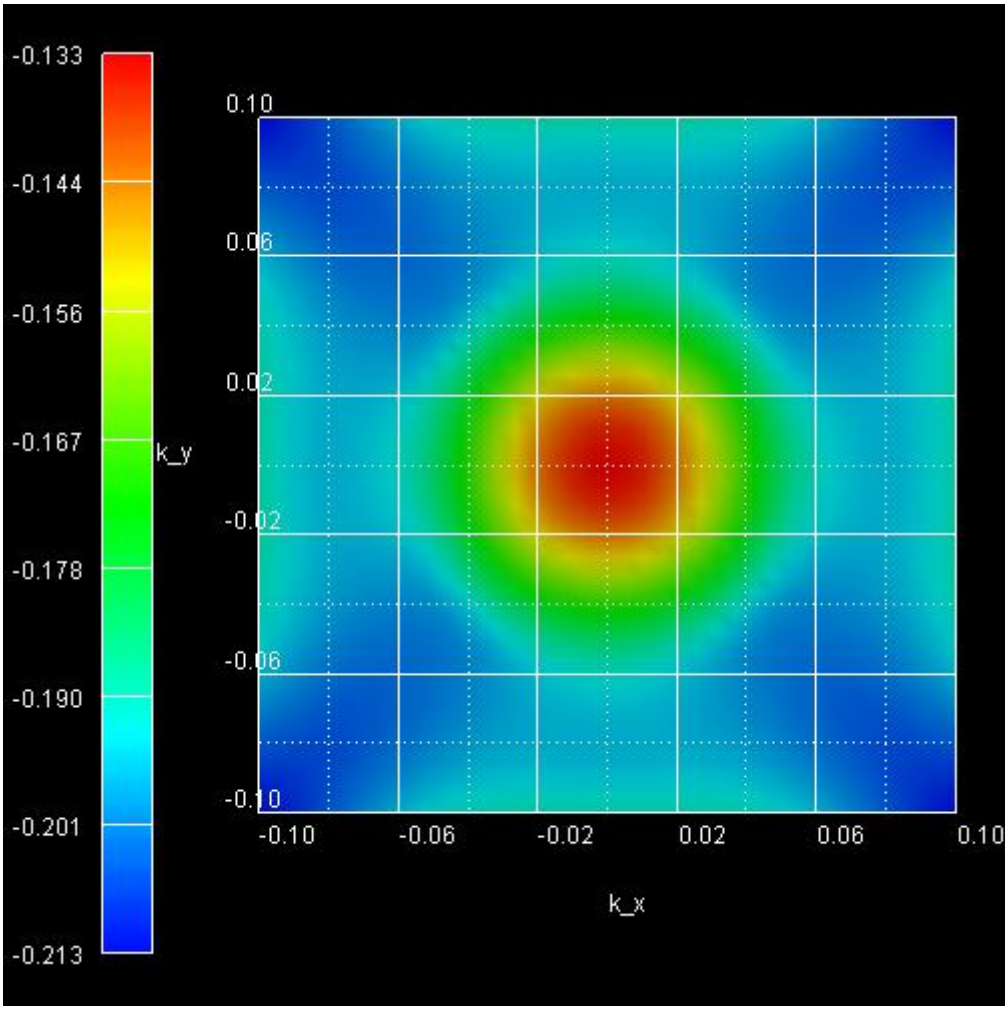


Figure 6.4.9.21: Subband 3 (eigenvalue 5 and 6)

The value of the abscissa is found as follows:

- From [10] to zero we just take $-k_x$.
- From zero to [11] we take $\sqrt{k_x^2 + k_y^2}$.

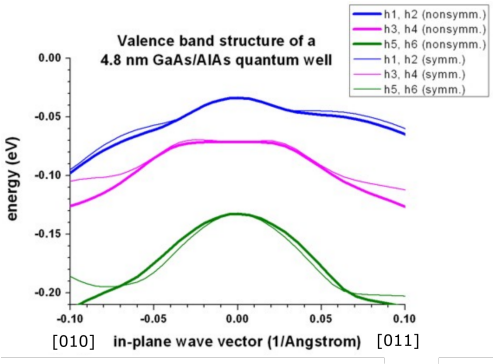


Figure 6.4.9.22: Calculated valence band structure of a $GaAs/AlAs$ QW.

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.

Symmetrized vs. nonsymmetrized k.p Hamiltonian

Note: In *nextnano++* the symmetrized k.p Hamiltonian is not implemented. Only *nextnano*[3] allows switching between the k.p dispersion for the nonsymmetrized and symmetrized k.p Hamiltonian by an explicit keyword.

```
$numeric-control
 simulation-dimension            = 1
 kp-vv-term-symmetrization       = no   ! nonsymmetrized k.p Hamiltonian
!kp-vv-term-symmetrization       = yes  !     symmetrized k.p Hamiltonian
```

b) Tensely strained $GaSb/AlSb$ quantum wells

Input file: *1Dwell_GaSb_AlSb_nnp.in*

Figure 6.4.9.23 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.

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.)

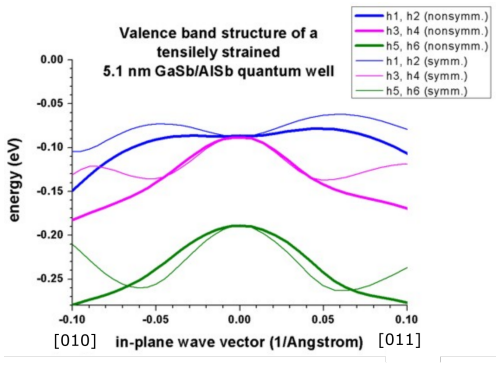


Figure 6.4.9.23: Calculated valence band structure of a tensely strained $GaSb/AlSb$ QW.

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.

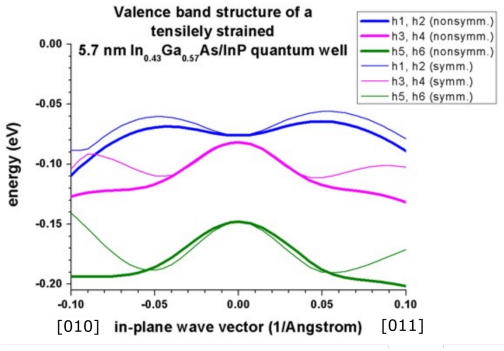


Figure 6.4.9.24: Calculated valence band structure of a tensely strained $In_{0.43}Ga_{0.57}As/InP$ QW.

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.

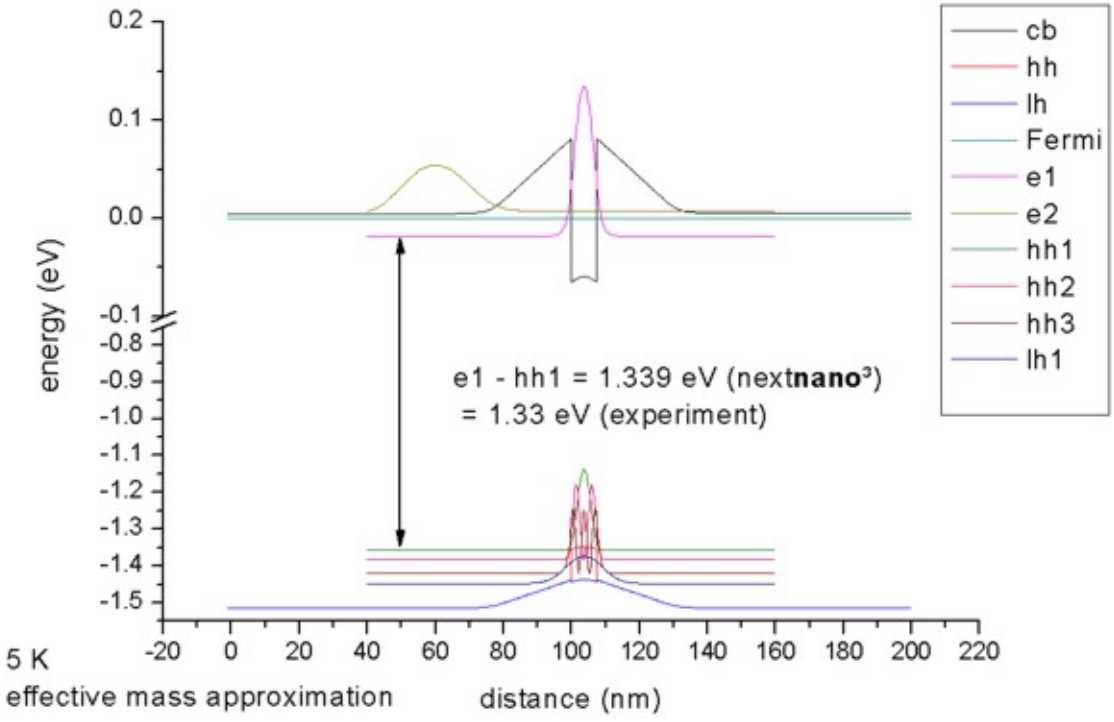


Figure 6.4.9.25: Calculated band edge profile of a compressively strained $In_{0.2}Ga_{0.8}As/GaAs$ QW (single-band Schrödinger equation).

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).

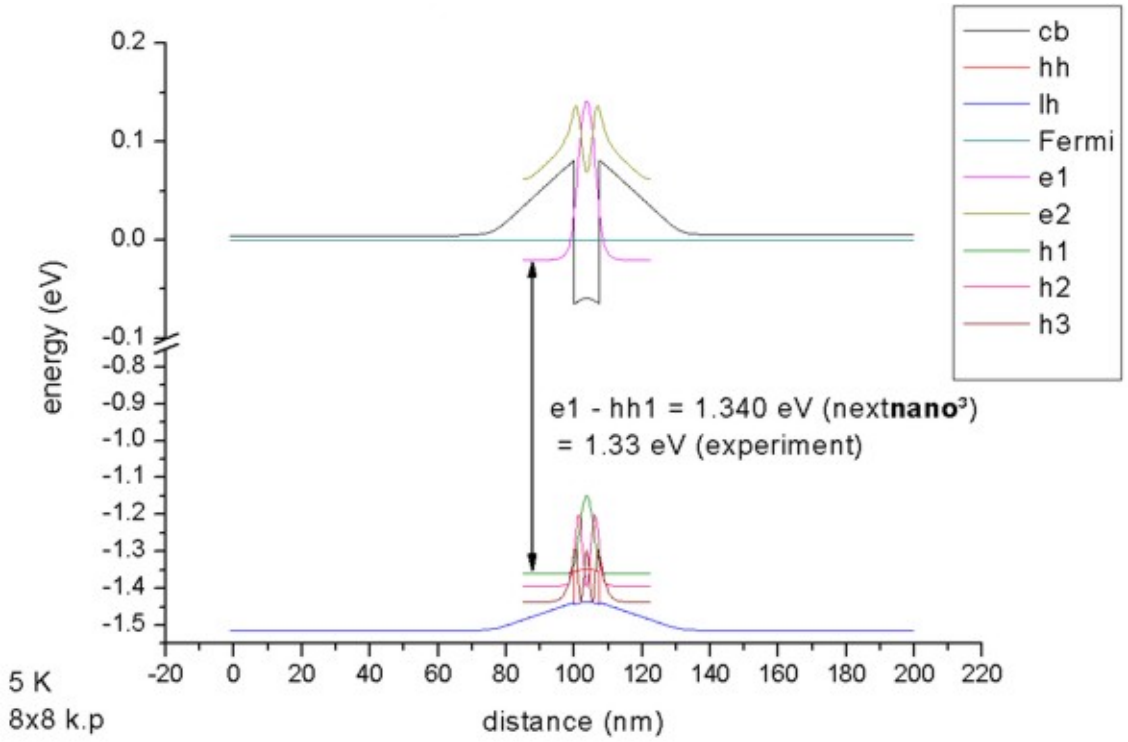


Figure 6.4.9.26: Calculated band edge profile of a compressively strained $In_{0.2}Ga_{0.8}As/GaAs$ QW (8-band k.p).

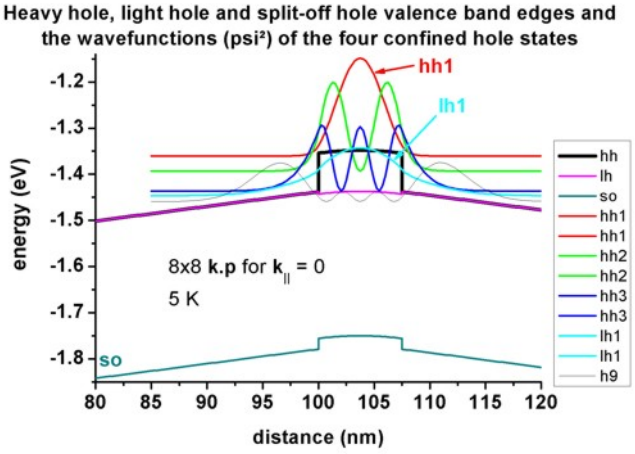


Figure 6.4.9.27: Calculated valence band structure and lowest hole states of a compressively strained $In_{0.2}Ga_{0.8}As/GaAs$ QW (8-band k.p).

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 6.4.9.28.

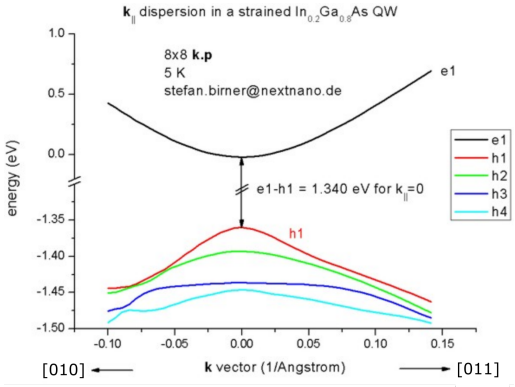


Figure 6.4.9.28: Calculated subband dispersions in $In_{0.2}Ga_{0.8}As/GaAs$ QW.

Figure 6.4.9.29 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: nn/nn/nnnn

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.

The structure is modeled as a superlattice (or multi quantum well, MQW), i.e. we apply periodic boundary conditions to the Poisson equation.

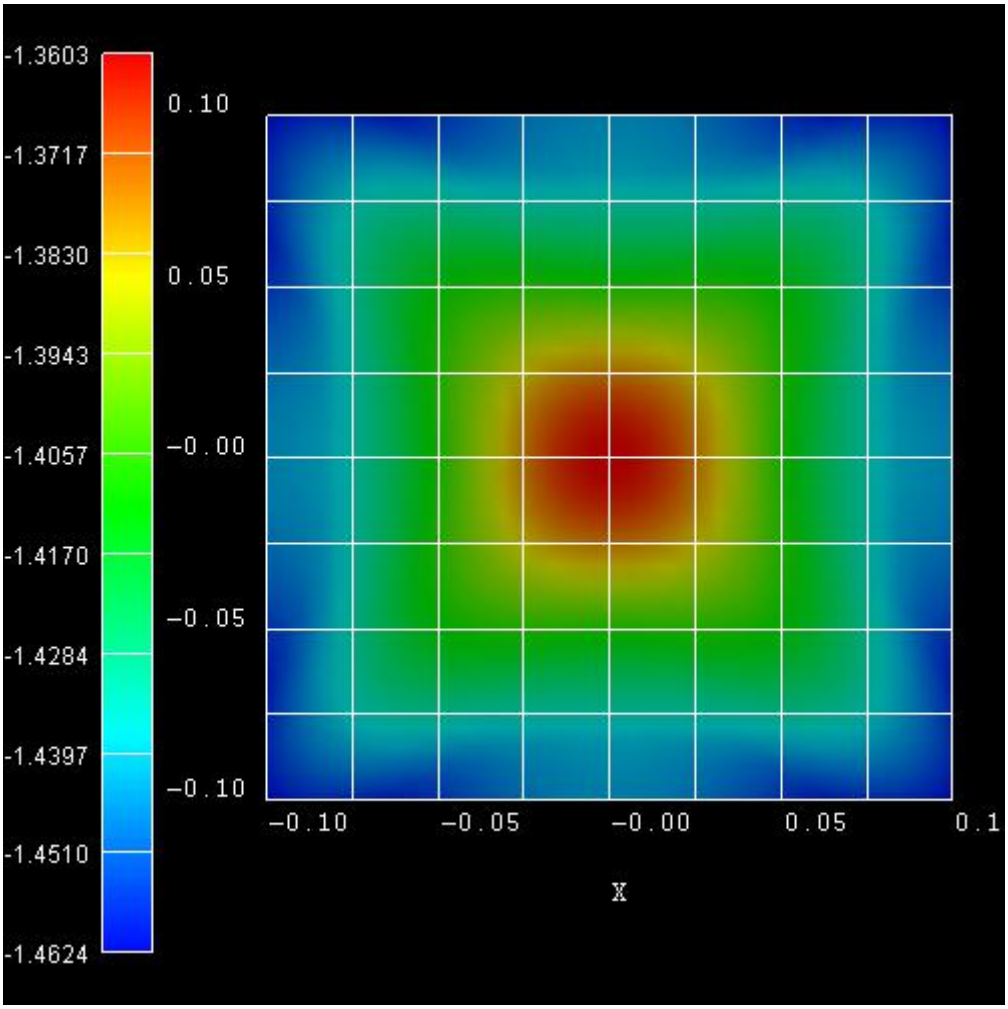


Figure 6.4.9.29: Calculated dispersion of h1 state in a $In_{0.2}Ga_{0.8}As/GaAs$ QW.

The growth direction is along the hexagonal axis, i.e. along [0001].

Conduction and valence band profile

Figure 6.4.9.30 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)

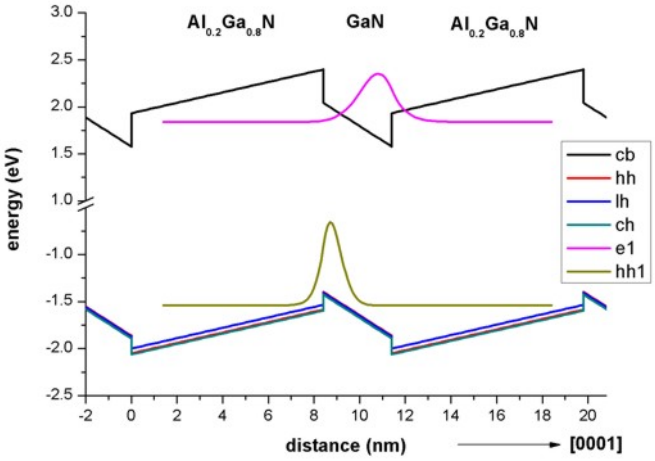


Figure 6.4.9.30: Calculated band edge profile.

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 6.4.9.31 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.

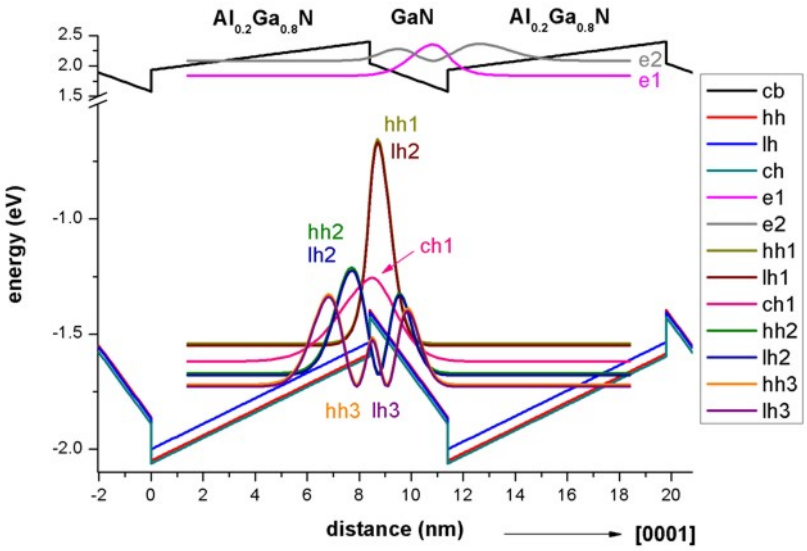


Figure 6.4.9.31: Calculated wave functions of lowest eigenstates.

$k_{||}$ 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 6.4.9.32 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 *[Park2000]* only the spin-up eigenstates are plotted because the splitting of the Kramer's degeneracy was assumed to be very small.

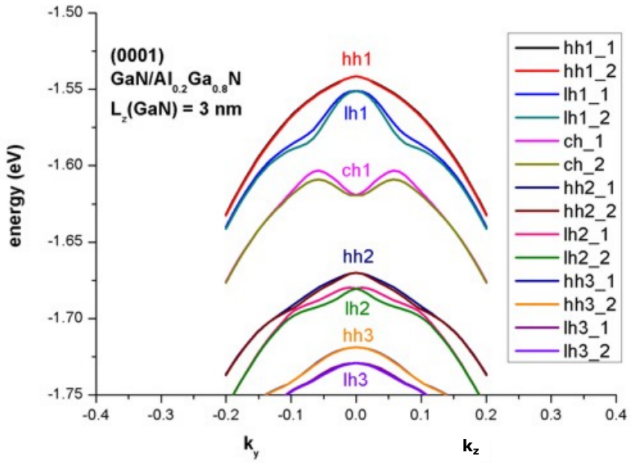


Figure 6.4.9.32: Calculated energy dispersion $E(k_{||}) = E(k_y, k_z)$.

[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 6.4.9.33 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.

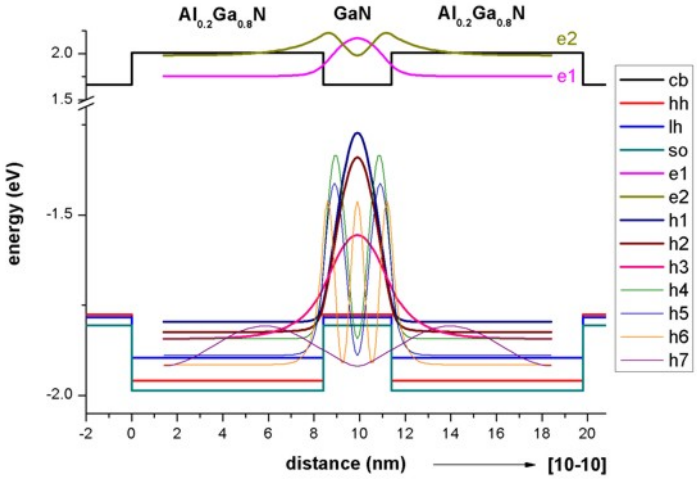


Figure 6.4.9.33: Calculated wave functions of lowest eigenstates.

$k_{||}$ 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 6.4.9.34, compare well with Fig. 3(c) of *[Park2000]*.

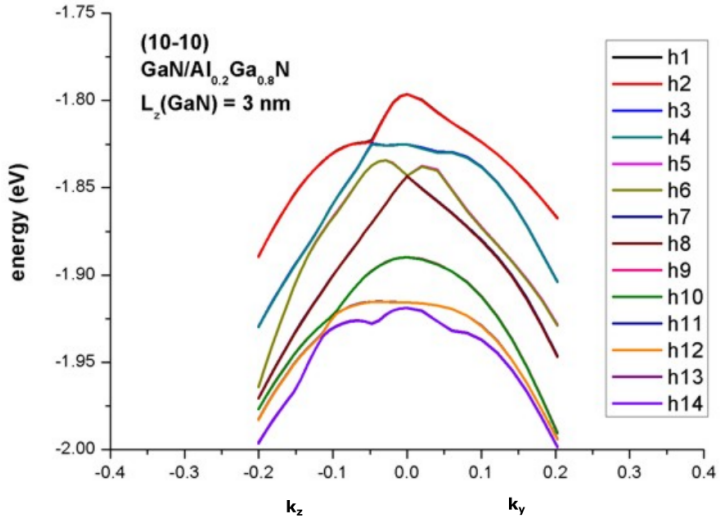


Figure 6.4.9.34: Calculated energy dispersion $E(k_{||}) = E(k_y, k_z)$.

Last update: nn/nn/nnnn

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 6.4.9.35. The GaN nanowire is discretized on a mesh with a grid resolution of 0.05 nm.

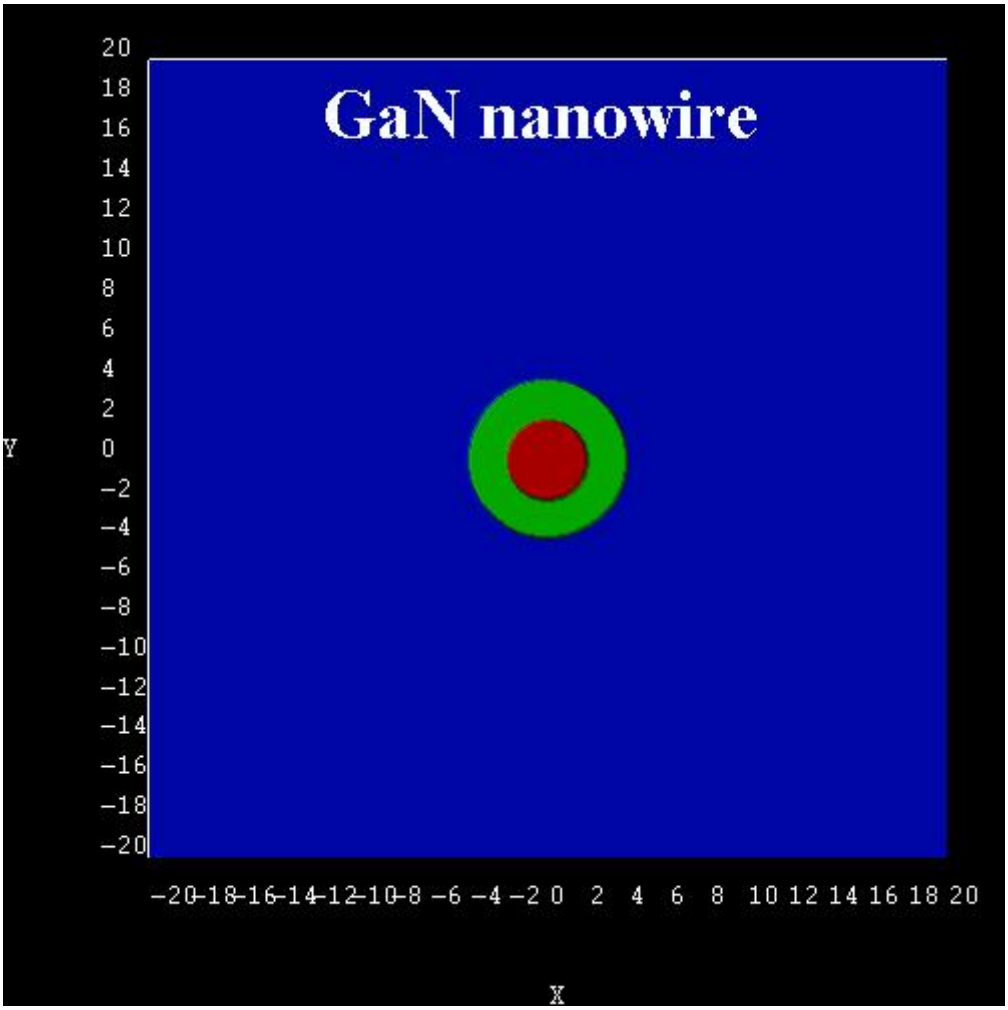


Figure 6.4.9.35: GaN nanowire structure.

Electrons

Figure 6.4.9.36 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).

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*

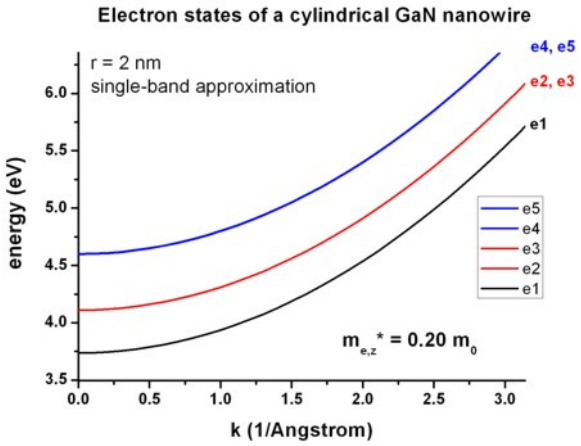


Figure 6.4.9.36: Energy dispersion $E(k)$ of electron states.

The wave function (Ψ^2) of the electron ground state at $k = 0$ is shown in Figure 6.4.9.37.

Holes

The following figures show the ground state wave function (psi^2) of the hole (Figure 6.4.9.38) and the 1st excited hole state (Figure 6.4.9.39) 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 don't have much overlap and the electron ground state has $L = 0$, whereas the hole ground state has $L = \pm 1$ (dark exciton effect).

Figure 6.4.9.40 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

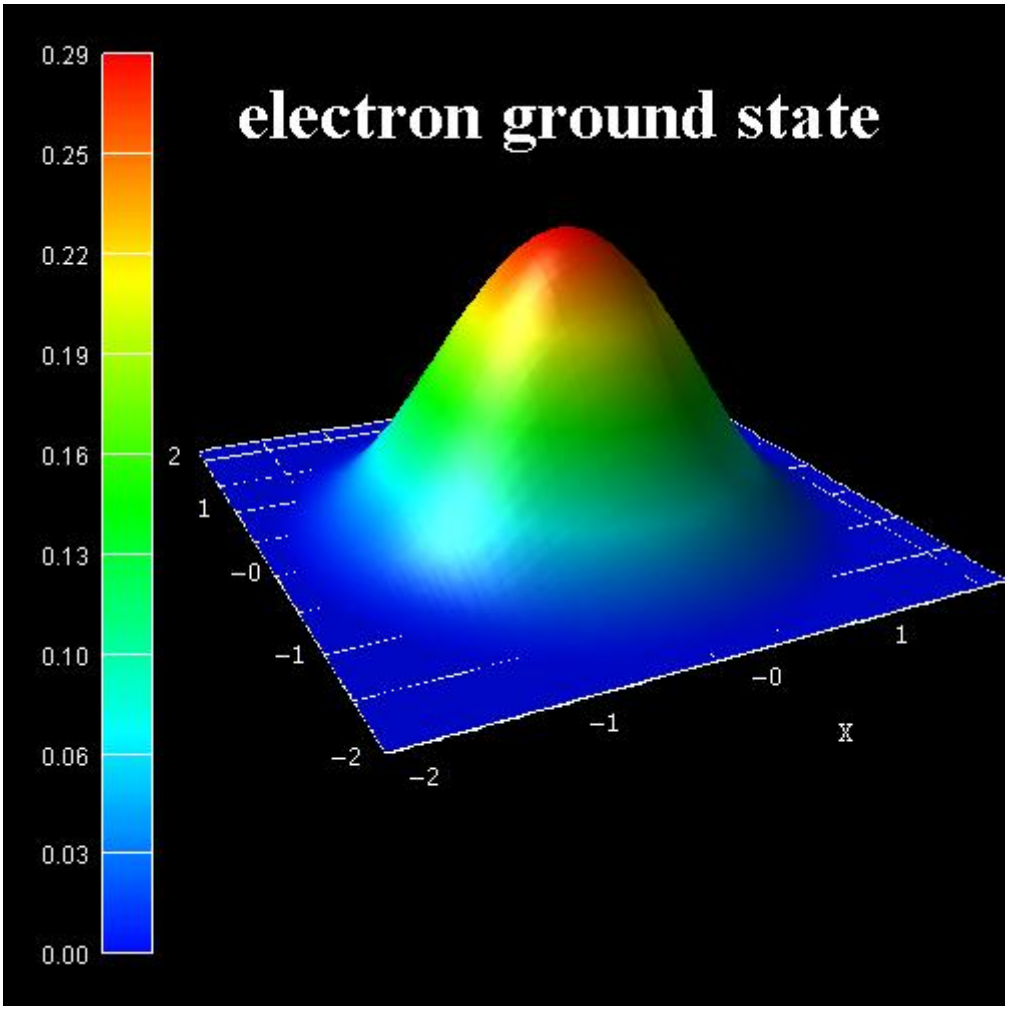


Figure 6.4.9.37: Ψ^2 of electron ground state.

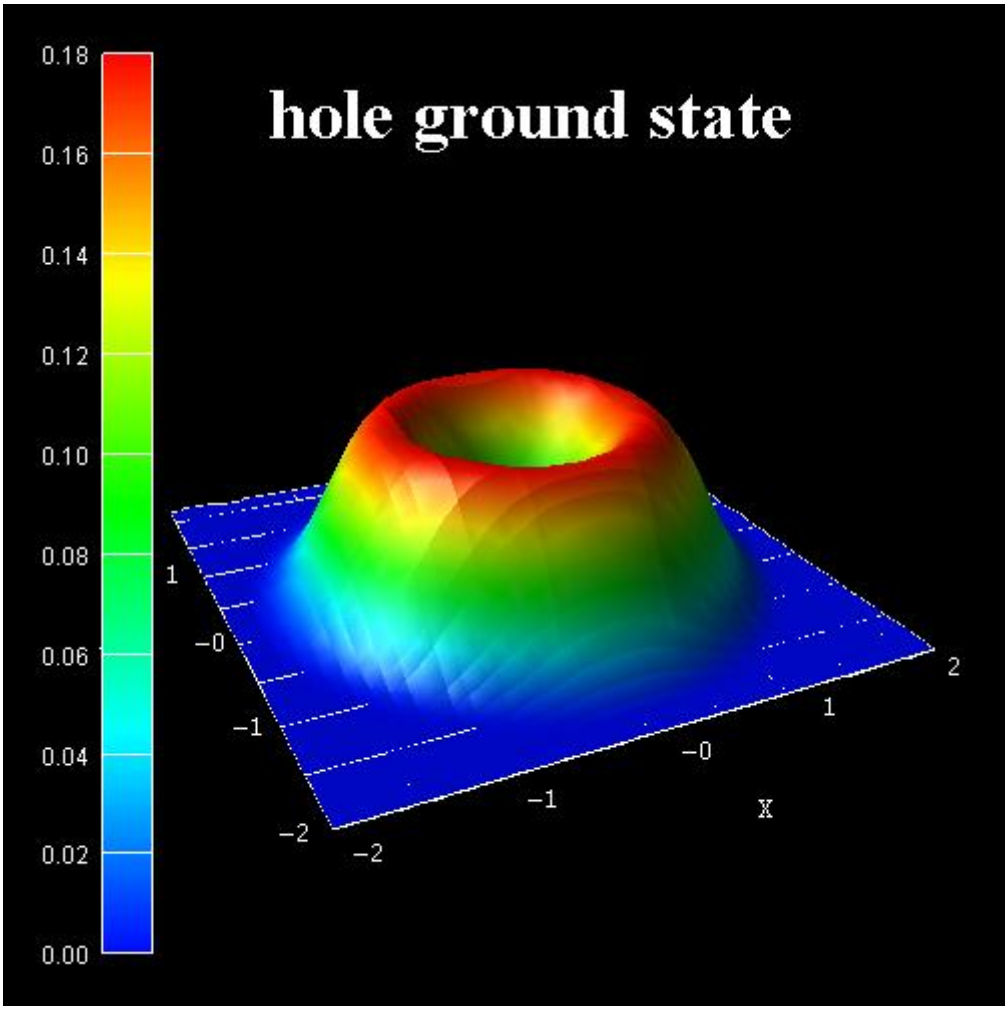


Figure 6.4.9.38: Ψ^2 of hole ground state.

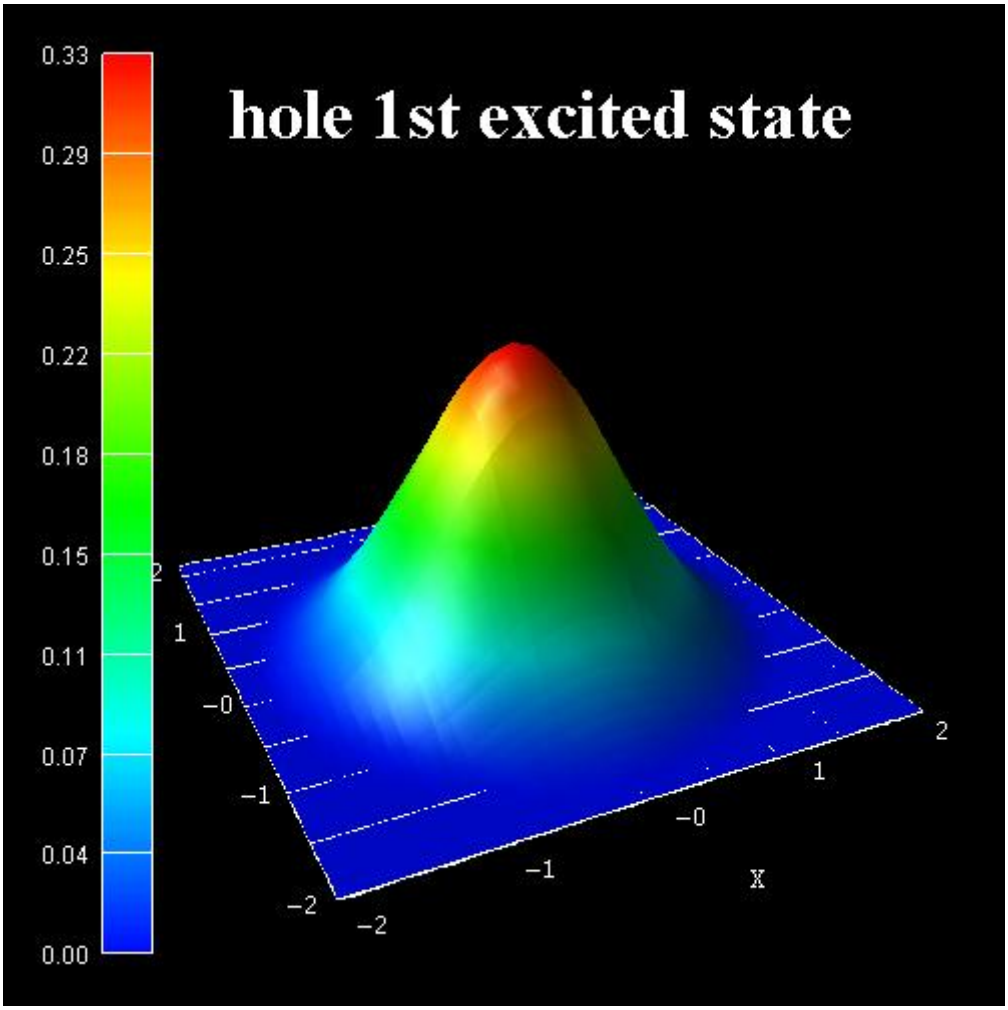


Figure 6.4.9.39: Ψ^2 of 1st excited hole state.

$$A_5 = \frac{1}{2}(L_1 - M_1)$$

is equal to

$$A_5 = \frac{1}{2}N_1.$$

However, they differ by 0.0064.

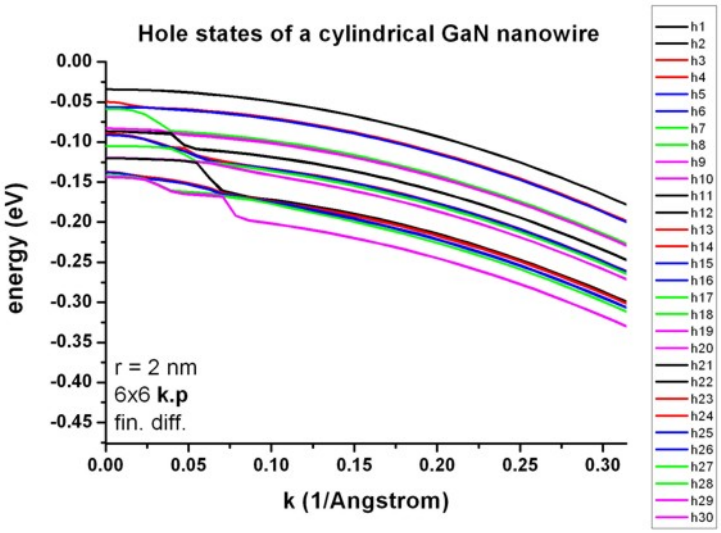


Figure 6.4.9.40: Energy dispersion $E(k)$ of hole states.

The data that has been plotted in Figure 6.4.9.40 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	$\Rightarrow$ The definition of the k.p Hamiltonians differs.
M = 0.1956	M_1 = -0.1956 - 1 = -1.1956	$\Rightarrow$ The definition of the k.p Hamiltonians differs.
N = 0.3813	M_2 = -0.3813 - 1 = -1.3813	$\Rightarrow$ 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	$\Rightarrow$ The definition of the k.p Hamiltonians differs
T = 7.3308	L_2 = -7.3308 - 1 = -8.3308	$\Rightarrow$ 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	$\Rightarrow$ The definition of the k.p Hamiltonians differs.
A_2 = M_3 + 1 = -1.4335 + 1 = -0.4335	$\Rightarrow$ 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	$\Rightarrow$ inconsistent to -3.06135
A_5 = 1/2 (N_1) = -3.06135	$\Rightarrow$ inconsistent to -3.05495
A_6 = $\sqrt{2}/2N_2$ = -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: nn/nn/nnnn

— SOON — 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 6.4.9.41) aim to reproduce Fig.1(a), and Fig.4(a) of *[FischettiJAP2003]*.

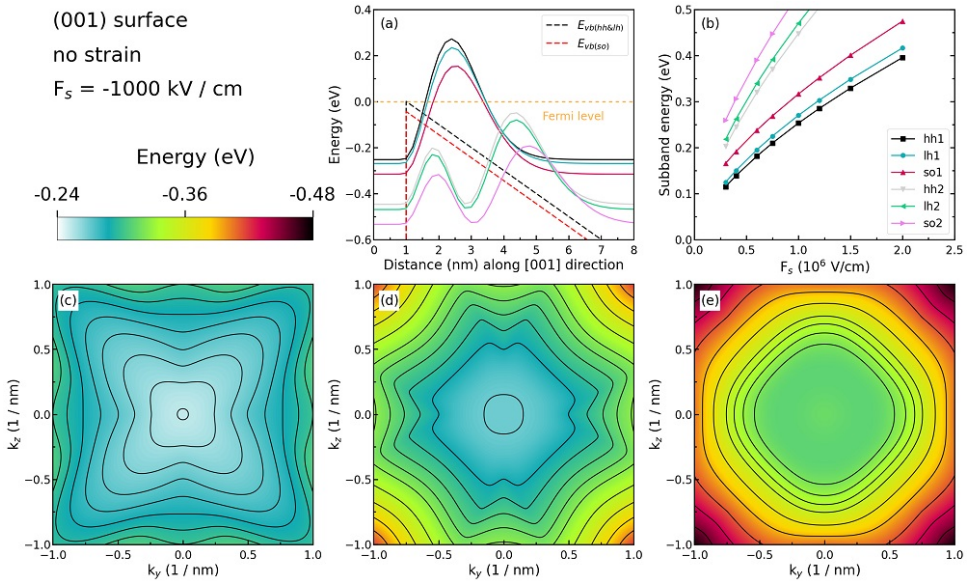


Figure 6.4.9.41: Some characteristics of the unstrained silicon inversion layer with (001) surface orientation.

Figure 6.4.9.41 (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

$$V(z') = eF_s z'$$

where F_s is the surface field. In the figure, the electric field is $F_s = -1000$ 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 6.4.9.41 (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 6.4.9.41 (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 6.4.9.41 (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 6.4.9.42) aim to reproduce Fig.2(a), and Fig.4(b) of *[FischettiJAP2003]*.

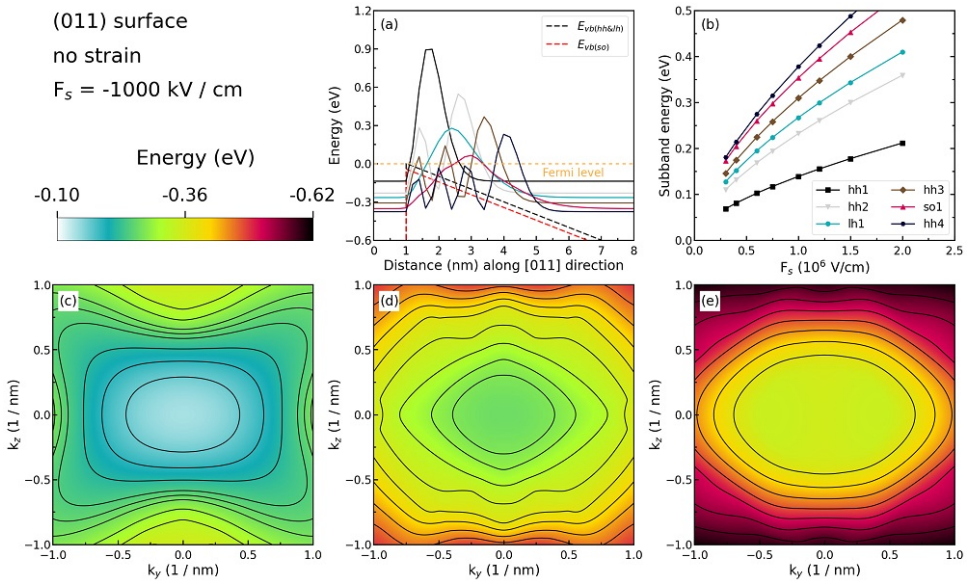


Figure 6.4.9.42: Some characteristics of the unstrained silicon inversion layer with (011) surface orientation.

Figure 6.4.9.42 (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 6.4.9.41, 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 6.4.9.42 (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 6.4.9.41 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 6.4.9.42 (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.

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 6.4.9.43) aim to reproduce Fig.3(a), and Fig.4(c) of *[FischettiJAP2003]*.

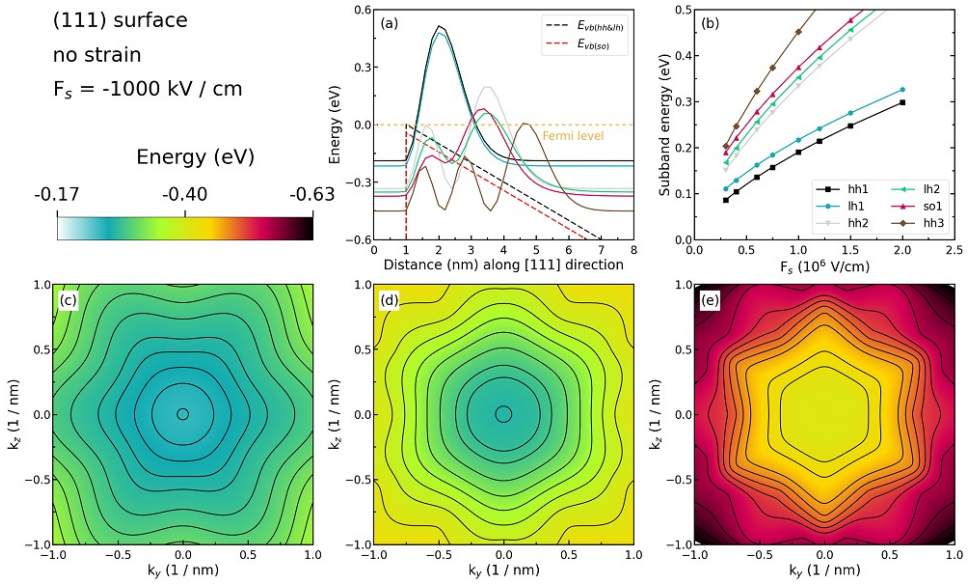


Figure 6.4.9.43: Some characteristics of the unstrained silicon inversion layer with (111) surface orientation.

Figure 6.4.9.43 (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 6.4.9.41, with a magnitude of $-1000\ \mathrm{kV/cm}$.

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 6.4.9.43 (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 6.4.9.41 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 6.4.9.43 (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 $\mathbf{k}$. 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 (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 6.4.9.44) aim to reproduce Fig.5(a), and Fig.7(a) of *[FischettiJAP2003]*.

Figure 6.4.9.44 (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 6.4.9.41, 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 6.4.9.44 (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

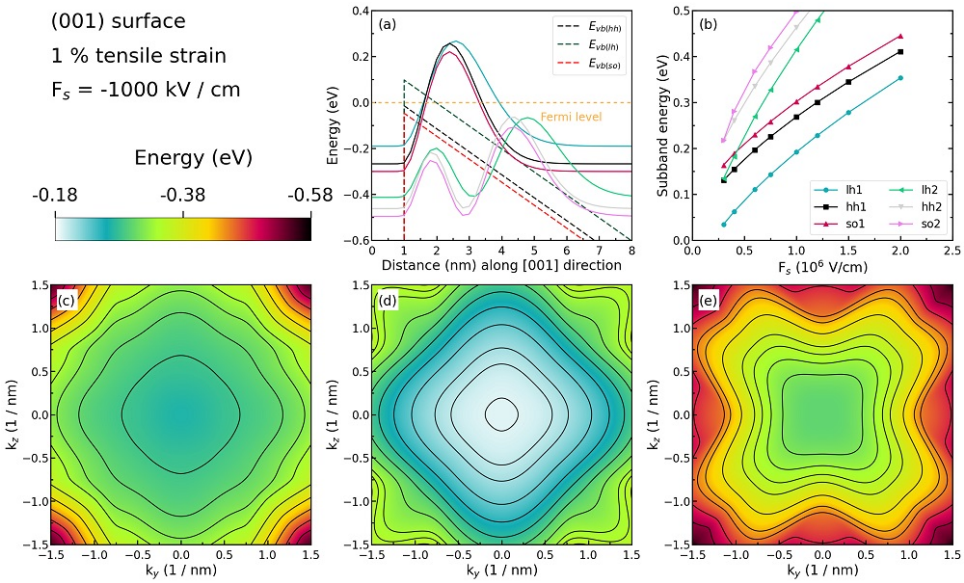


Figure 6.4.9.44: Some characteristics of the $1\ \%$ tensilely strained silicon inversion layer with (001) surface orientation.

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 6.4.9.41 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 6.4.9.44 (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 $\mathbf{k}$. 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 6.4.9.45) aim to reproduce Fig.6(a), and Fig.7(b) of *[FischettiJAP2003]*.

Figure 6.4.9.45 (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 6.4.9.41, 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 6.4.9.45 (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.

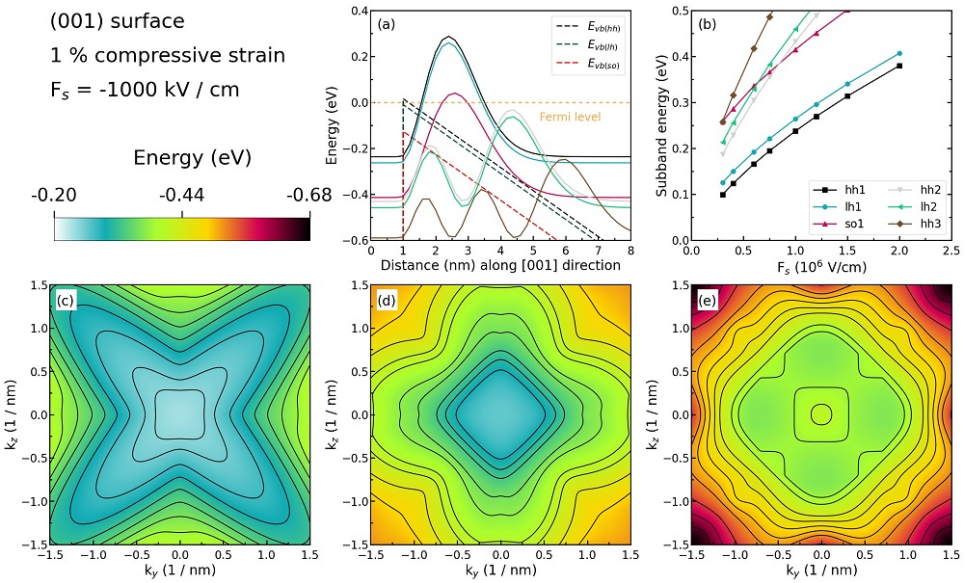


Figure 6.4.9.45: Some characteristics of the $1\ \%$ compressively strained silicon inversion layer with (001) surface orientation.

The plotting method is the same as in Figure 6.4.9.41 (b), and we also do not perform an analysis on the labels (**hh**, **lh**, and **so**) of each curve.

Figure 6.4.9.45 (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 6.4.9.46) show how the number of $k_{||}$ points affects the simulation results. The system is the same as the one we use in Figure 6.4.9.41, however, with different $k_{||}$ points. The equienergy lines are plotted for $E - E_0 = -25\ \mathrm{meV}$ where E_0 is the eigenvalue of corresponding subbands at $\mathbf{k} = (k_x, k_y) = 0$.

The grid points on the Figure 6.4.9.46 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

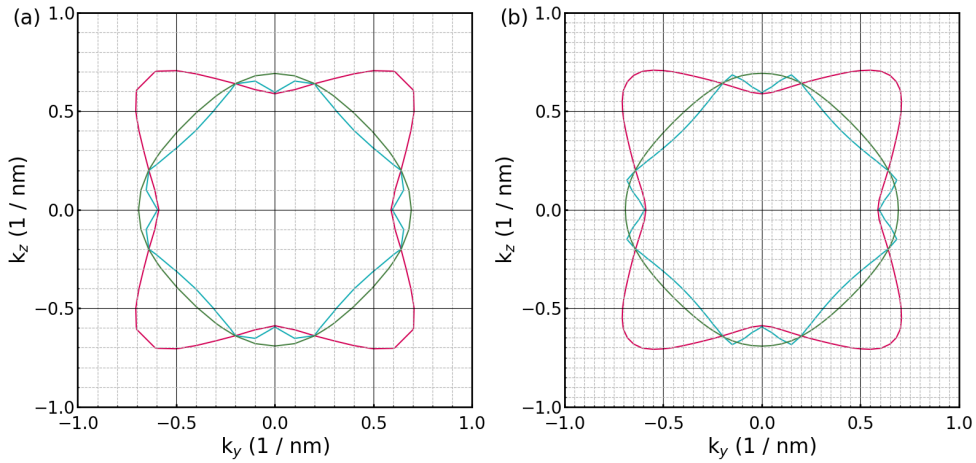


Figure 6.4.9.46: The dispersion for 441 $k_{||}$ points ((a)), and for 1681 $k_{||}$ points ((b)) for the (001) surface of the unstrained silicon under the electric field ($F_s = -1000$ kV/cm).

— NEW — 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 (6.4.9.1).

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{zx} \\ \sigma_{xy} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{12} & & & \\ C_{12} & C_{11} & C_{12} & & & \\ C_{12} & C_{12} & C_{11} & & & \\ & & & C_{44} & & \\ & & & & C_{44} & \\ & & & & & C_{44} \end{bmatrix} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ 2\varepsilon_{yz} \\ 2\varepsilon_{zx} \\ 2\varepsilon_{xy} \end{bmatrix} \tag{6.4.9.1}$$

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 *— SOON — 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,

$$\varepsilon_{xx} = -0.00727$$
$$\varepsilon_{yy} = \varepsilon_{zz} = 0.00277$$
$$\varepsilon_{yz} = \varepsilon_{zx} = \varepsilon_{xy} = 0$$

This data is contained at *band-structure-2DHG_Si_Wang_2004_1D_nnp_biax_tens_strain.dat*.

Simulation results

No stress applied

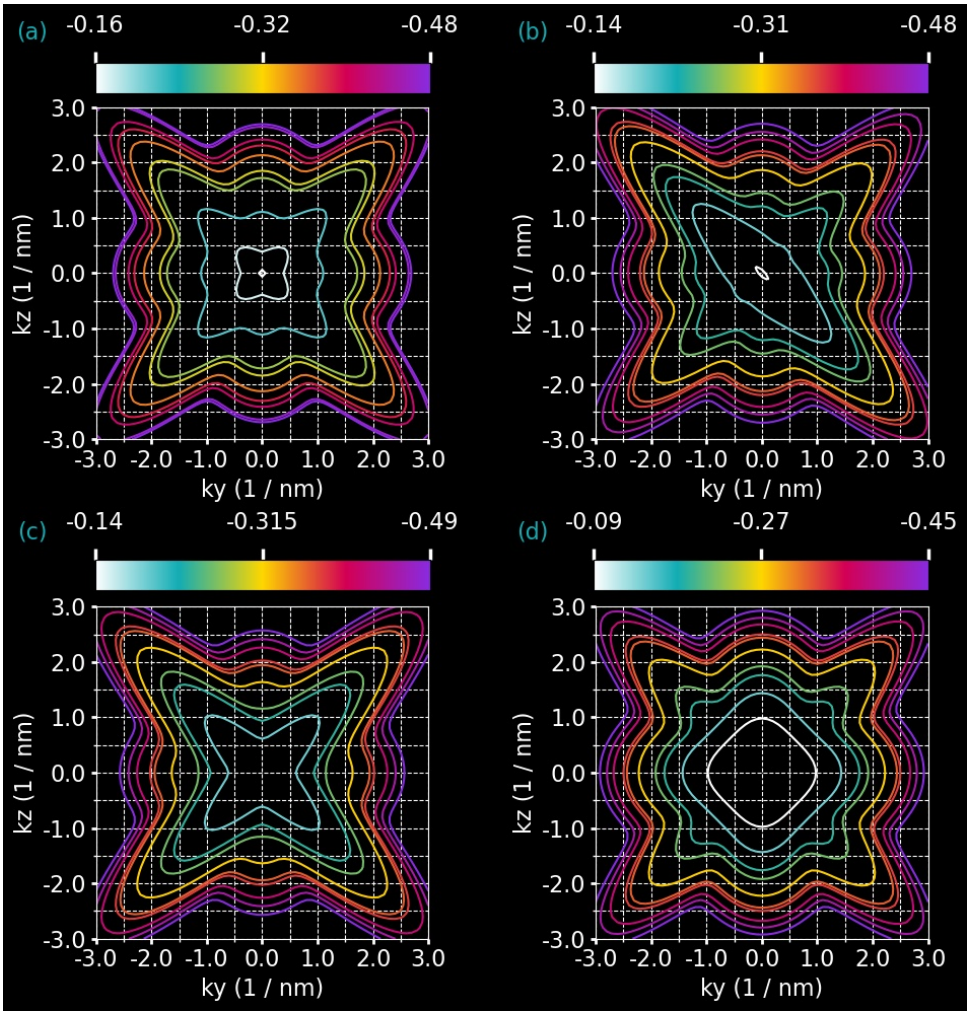


Figure 6.4.9.47: The calculated equienergy lines under no strain (a), under uniaxial strain (b), under biaxial compressive strain (c), and under biaxial tensile strain (d) are shown. Only one spin state is plotted for clarity. The axes represent k_y and k_z in units of $[1/\mathrm{nm}]$.

First, Figure 6.4.9.47 (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 6.4.9.47 (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 6.4.9.47 (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

6.4.10 Superlattices

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 $GaAs$ / ... . 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 6.4.10.1 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

$$\Psi(x + L) = e^{ik_x(x+L)} = e^{ik_x x} e^{ik_x L}$$

$$\Leftrightarrow \quad \Psi(x + L) = \Psi(x) e^{ik_x L}$$

k_x is the wave vector of the electron (or hole) along the growth direction of the infinite superlattice. In Figure 6.4.10.2 we plot the dispersion curve, i.e. the energy of the electron as a function of its superlattice wave vector

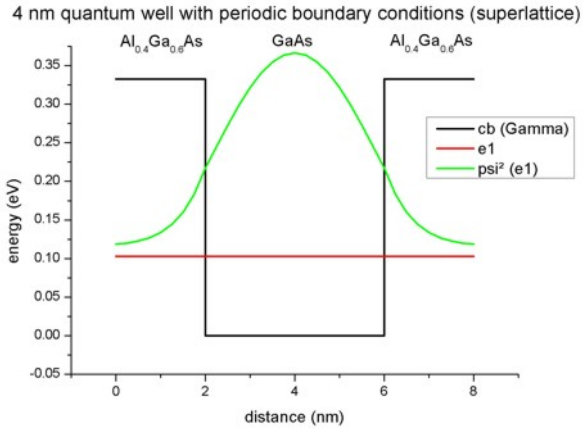


Figure 6.4.10.1: Calculated conduction band edge profile of single 4 nm GaAs QW with periodic boundary conditions.

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 $[-\pi/L, \pi/L]$.

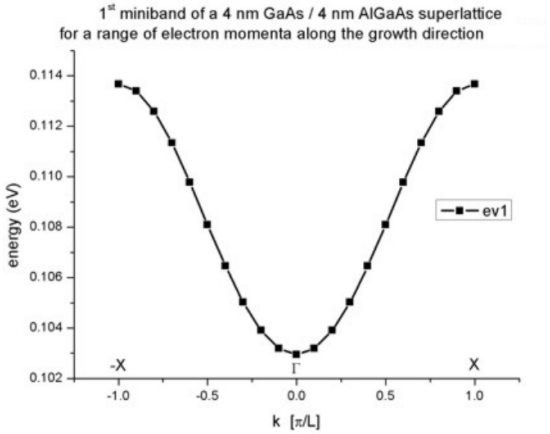


Figure 6.4.10.2: Calculated subband dispersion (= miniband)

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 $GaAs$ / ... . 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 6.4.10.3 shows the conduction band edge and the two lowest eigenstates that are confined inside the well and their corresponding probailiy 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.)

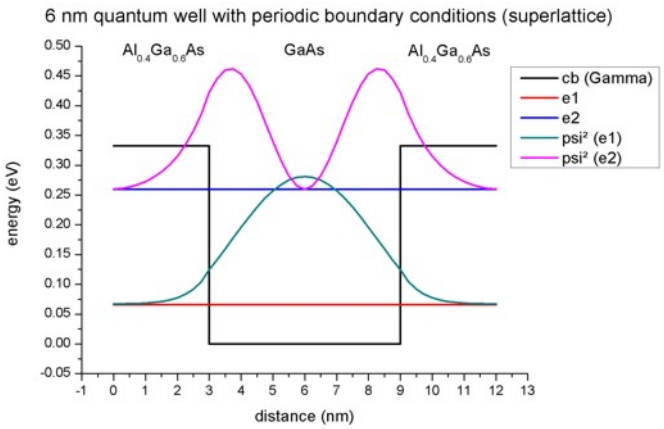


Figure 6.4.10.3: Calculated conduction band edge profile of single 6 nm $GaAs$ QW with periodic boundary conditions.

The following figure (Figure 6.4.10.4) 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).

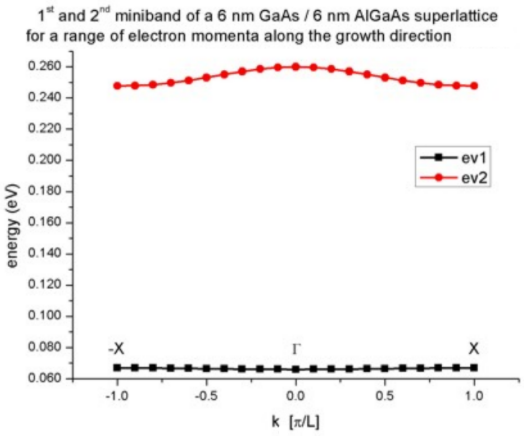


Figure 6.4.10.4: Calculated subband dispersion (= miniband)

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"
```

(continues on next page)

(continued from previous page)

```
                point{ k = [$left_dispersion,  0.0, 0.0] }
                point{ k = [$right_dispersion, 0.0, 0.0] }
                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 6.4.10.5 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.)

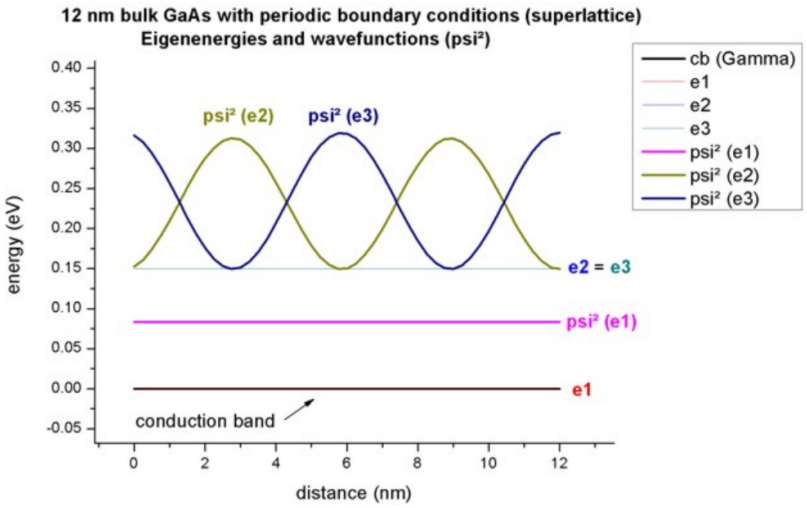


Figure 6.4.10.5: Calculated conduction band edge profile of bulk $GaAs$ and Ψ^2 of lowest electron eigenstates (periodic boundary conditions were used).

The following figure (Figure 6.4.10.6) 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 $GaAs$.

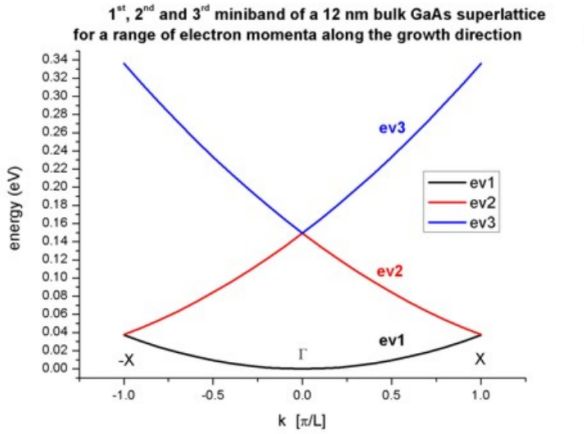


Figure 6.4.10.6: Calculated subband dispersion (= miniband)

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*.

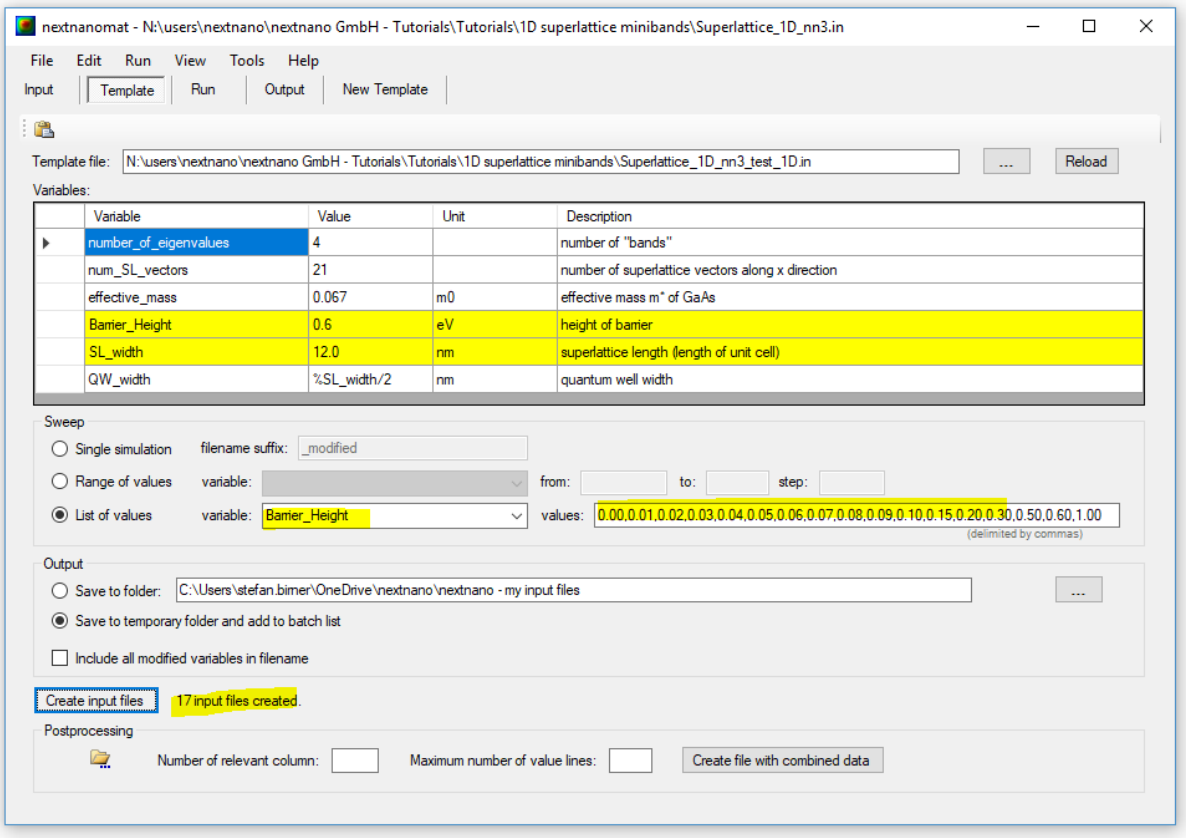


Figure 6.4.10.7 compares the dispersion of a superlattice for two different QW barrier heights.

Last update: nn/nn/nnnn

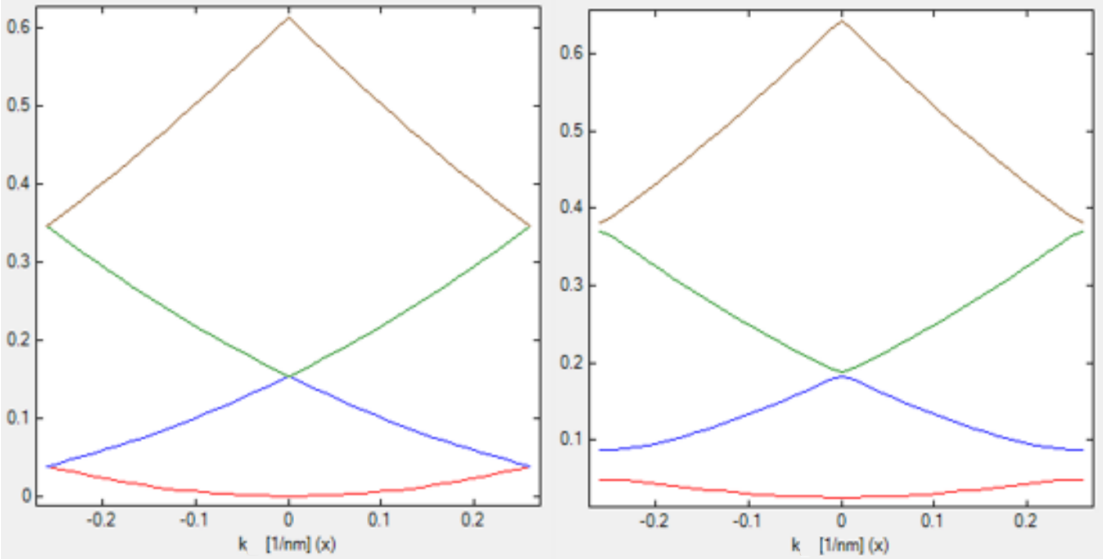


Figure 6.4.10.7: The left figure contains a quantum well superlattice with a barrier height of 0 eV, i.e. a bulk semiconductor, while the figure on the right shows the dispersion for a barrier height of 0.06 eV.

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.

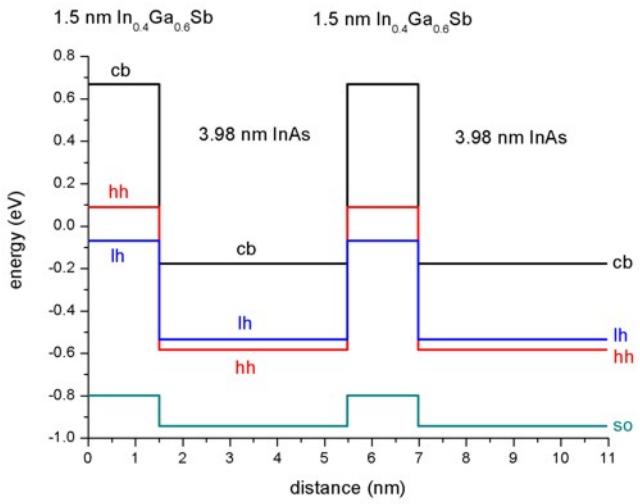
The structure has a **type-II** band alignment, i.e. the electrons are confined in the InAs layer, whereas the holes are confined in the $In_{0.4}Ga_{0.6}Sb$ layer.

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.

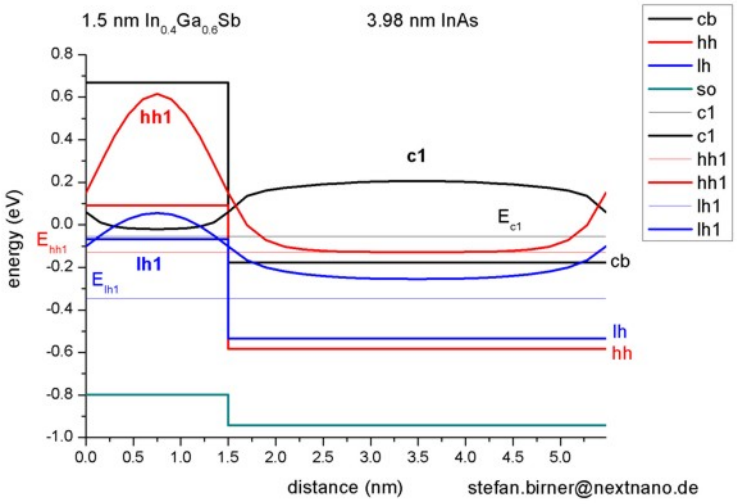


Electron and hole wave function for $k_{||} = 0$

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.

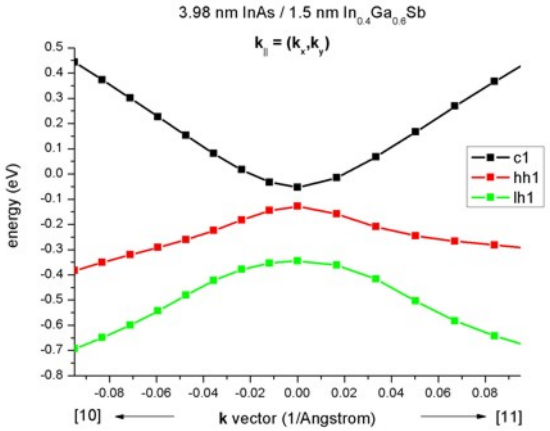
One can clearly see that the electron state (**c1**) is confined in the InAs layer (right part of the figure), whereas the heavy (hh1) and light hole (lh1) states are confined in the $In_{0.4}Ga_{0.6}Sb$ layer (left part of the figure).



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.

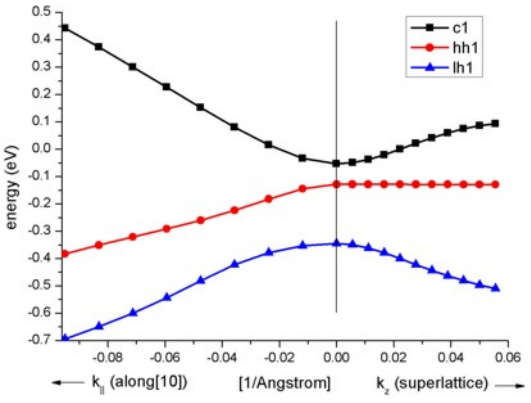
Electron and hole energies for $k_z \neq 0$

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.

nextnano3

Using `1DInAs_InGaSb_SL_k_parallel_superlattice.in`, one can generate a 3D plot of the energy dispersion E(k_x, k_y, k_{SL}) for each eigenvalue. The files are called `dispersion_k_parallel_k_SL_ev001.fld` where `ev001` indicates eigenvalue number 1.

Last update: nn/nn/nnnn

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{
```

(continues on next page)

(continued from previous page)

```
      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"
   }
}
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
```

(continues on next page)

(continued from previous page)

```
        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
        }

        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 following graphs were generated with *nextnanopy*. 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.

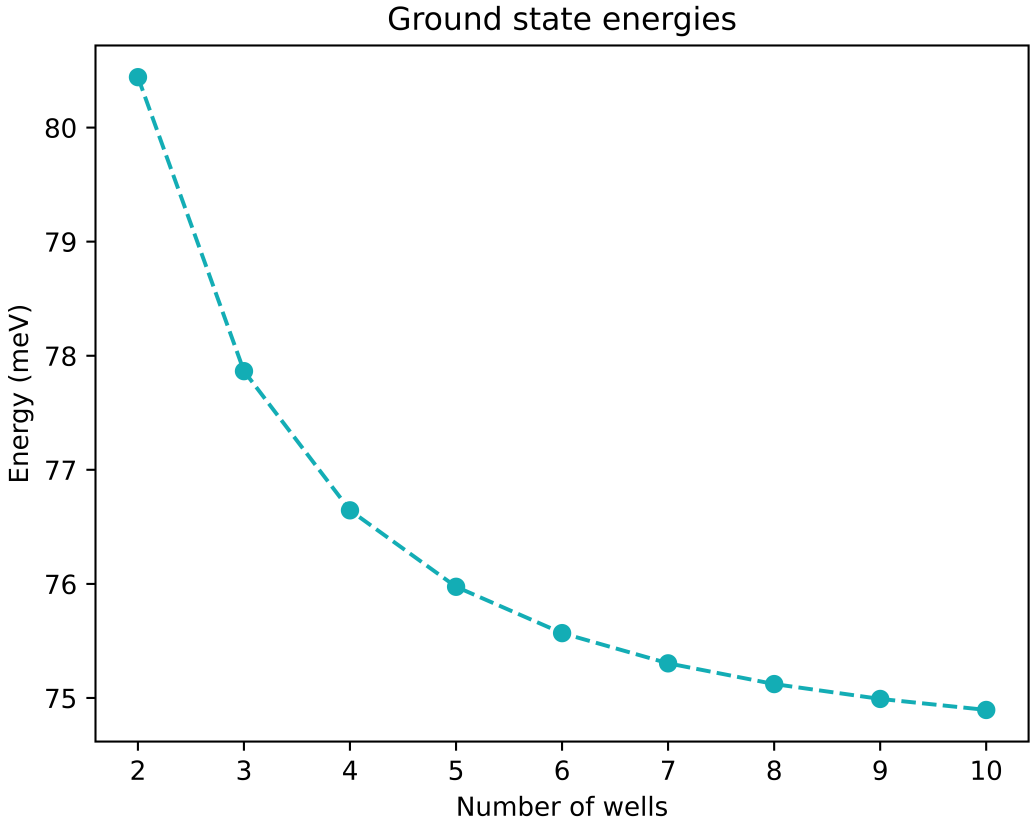


Figure 6.4.10.8: Ground state energies plotted as a function of N. Convergence at higher number of wells is observed.

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

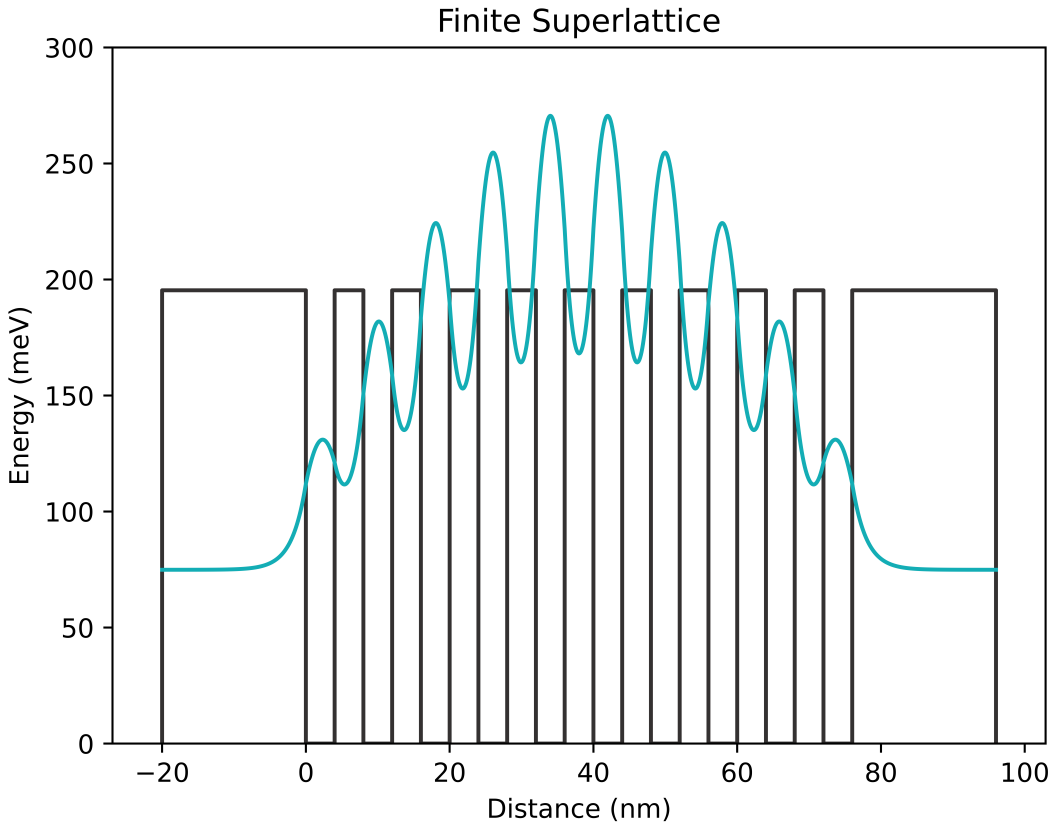


Figure 6.4.10.9: The wave function for a superlattice system

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: nn/nn/nnnn

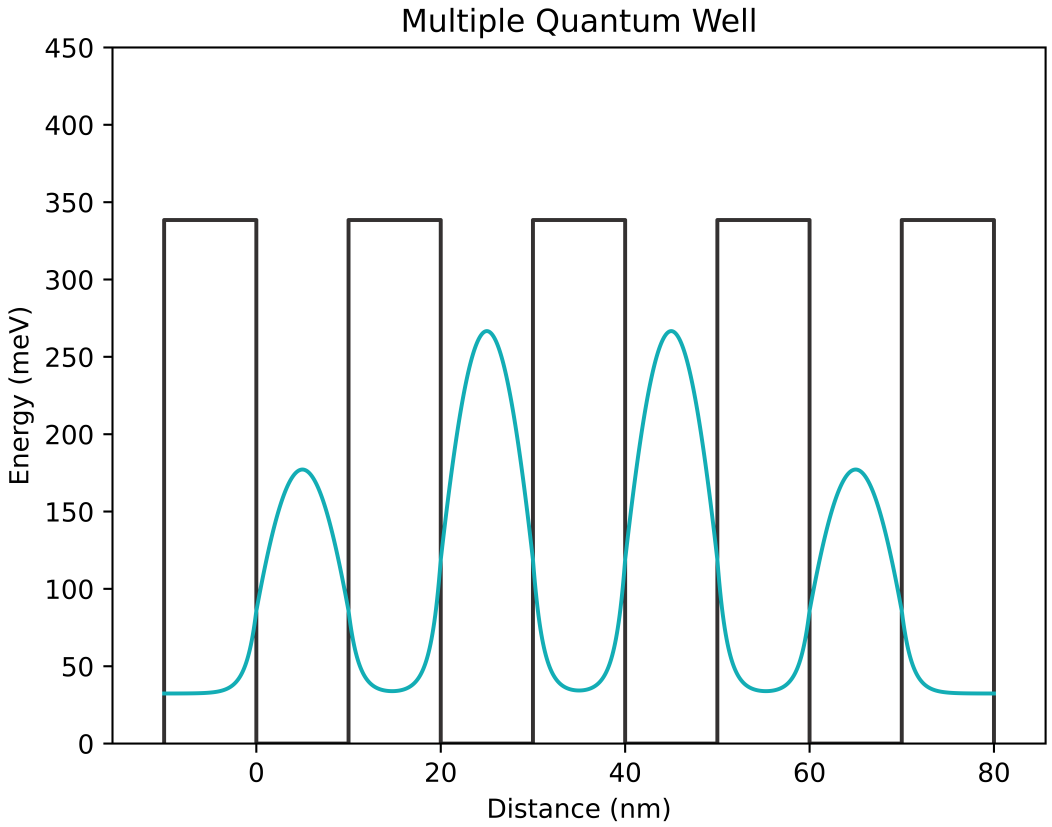


Figure 6.4.10.10: The wave function for a multiple quantum well system

— 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 $L_x = L_y = L_z = 6.5$ nm and are separated by a distance of $H_x = H_y = H_z = 1.5$ nm. We model only one QD and assume periodic boundary conditions along the x, y and z direction giving a superlattice period of $d_x = d_y = d_z = 8$ 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{}
           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"
```

(continues on next page)

(continued from previous page)

```
            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 6.4.10.11 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.

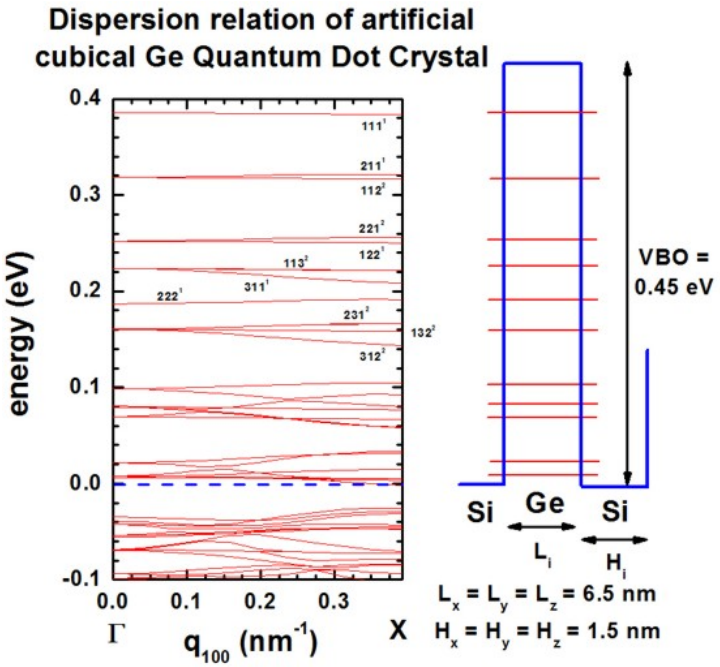


Figure 6.4.10.11: (left) Superlattice dispersion along [100] for heavy-hole states in an artificial cubic Ge quantum dot crystal and (right) valence-band profile through the center of the QD and eigenstates of the lowest heavy-hole states.

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 6.4.10.12 and Figure 6.4.10.13) 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 6.4.10.13).

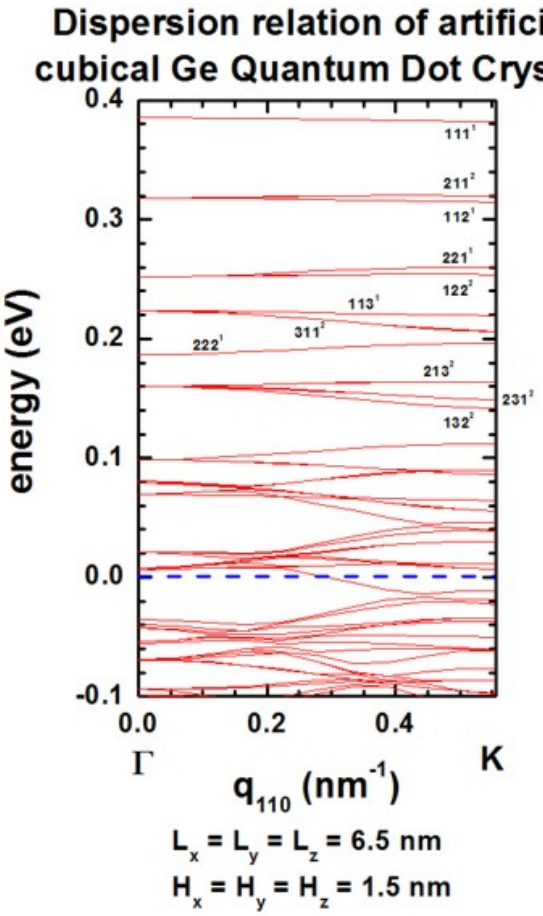


Figure 6.4.10.12: Superlattice dispersion along [110] for heavy-hole states in an artificial cubic Ge quantum dot crystal.

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.

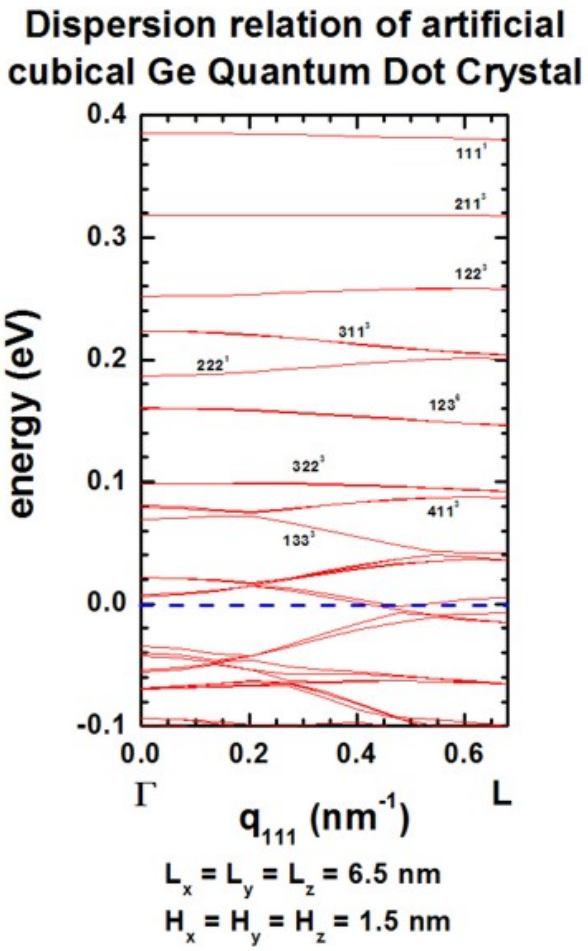


Figure 6.4.10.13: Superlattice dispersion along [111] for heavy-hole states in an artificial cubic Ge quantum dot crystal.

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]*.

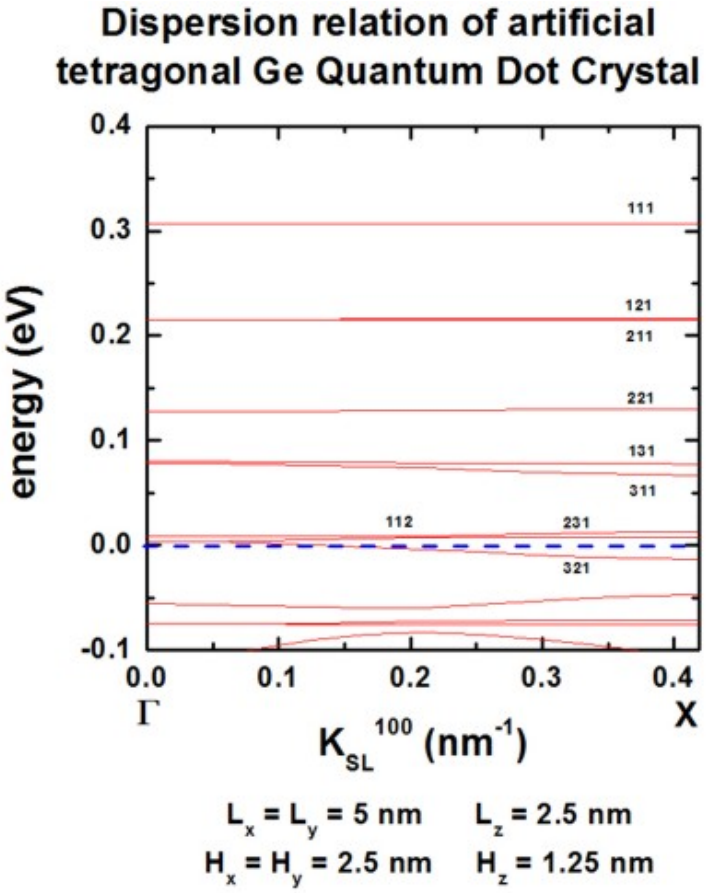


Figure 6.4.10.14: Superlattice dispersion along [100] for heavy-hole states in an artificial tetragonal Ge quantum dot crystal.

The version of this tutorial for *nextnano*[3] can be found here.

Last update: 17/07/2024

6.4.11 Cascade Structures

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.

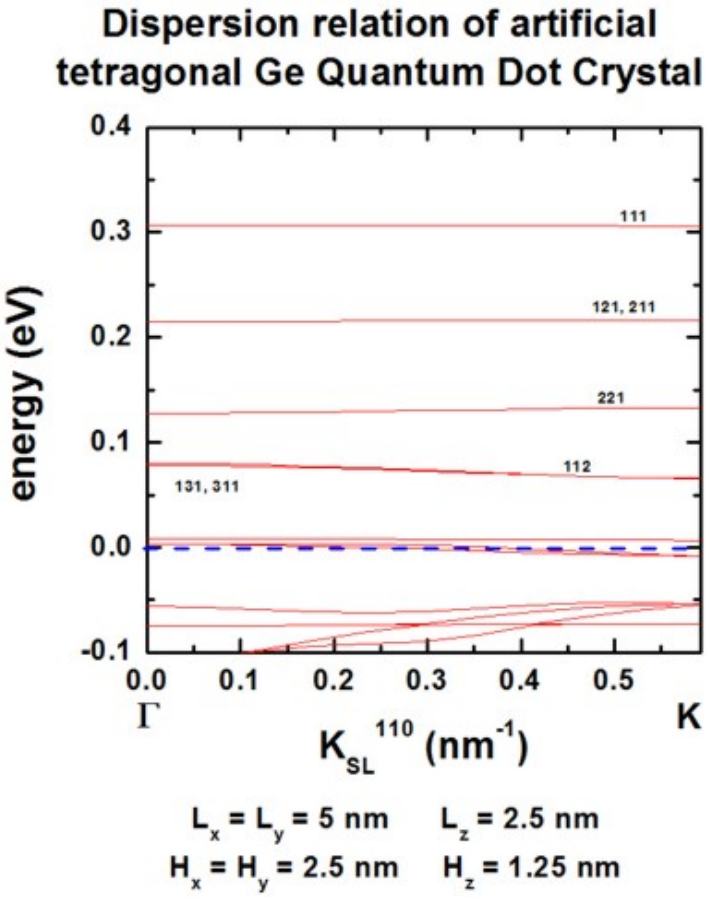


Figure 6.4.10.15: Superlattice dispersion along [110] for heavy-hole states in an artificial tetragonal Ge quantum dot crystal.

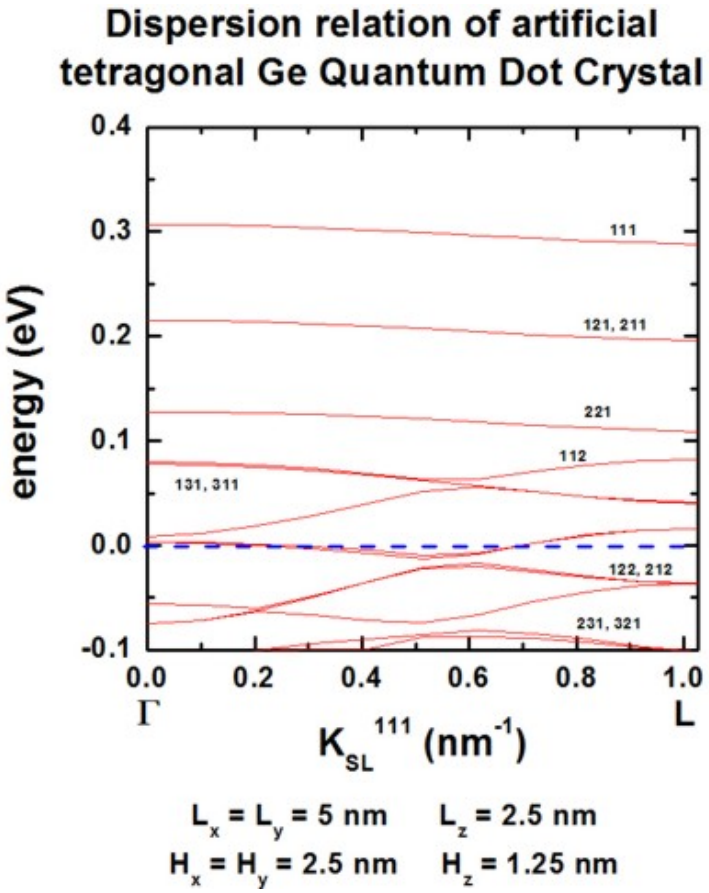
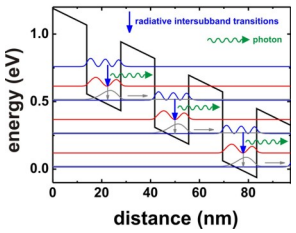


Figure 6.4.10.16: Superlattice dispersion along [111] for heavy-hole states in an artificial tetragonal Ge quantum dot crystal.



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.

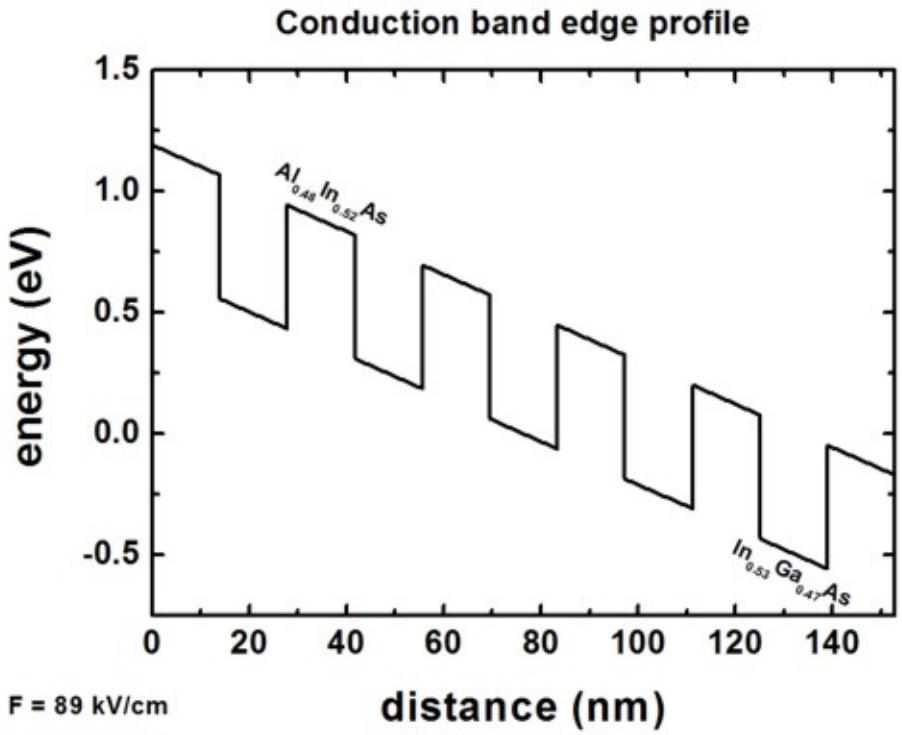
Bandedges

The conduction and valence band edges can be found in the following file:

- *bias_0000/Quantum/bandedges.dat* (*nextnano++*)
- *band_profile/cb_Gamma.dat* (*nextnano*[3], conduction band edge only)

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* (*nextnano++*)
- *wavefunctions/ev_cb1_sg1_deg1.dat* (*nextnano*^{3})

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* (*nextnano++*)
- *wavefunctions/cb1_sg1_deg1_psi_squared_shift.dat* (*nextnano*^{3})

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:

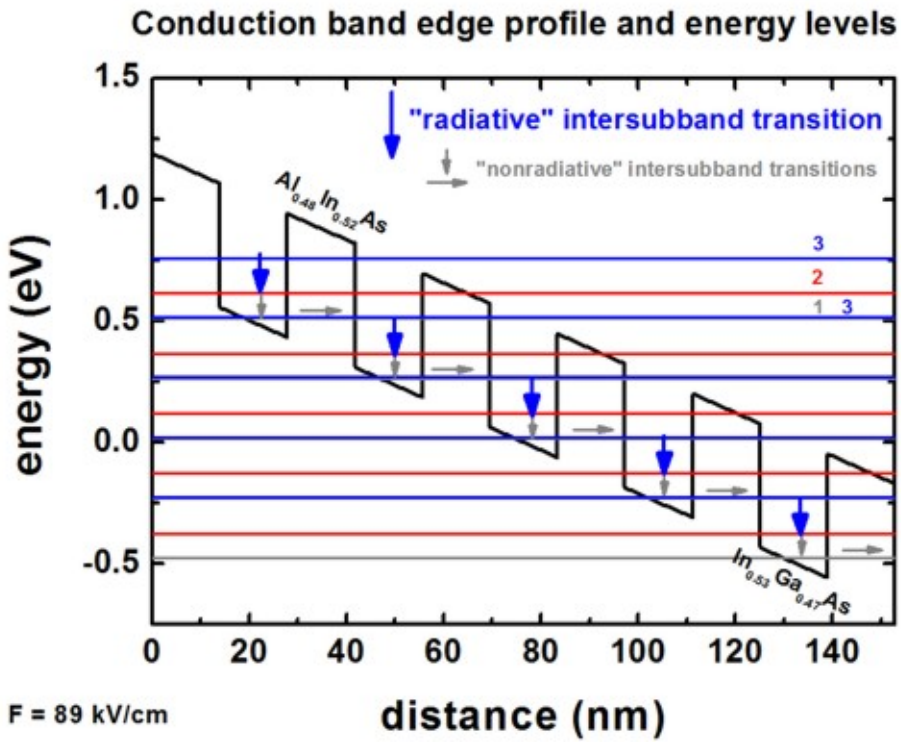
Note: The figure shows only the following energy levels: 1,2,3,4,5,9,10,12,16,18,20,26,27,30,37

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* (*nextnano++*)
- *wavefunctions/cb1_qc1_sg1_deg1_psi_squared_shift.dat* (*nextnano*^{3})

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: 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

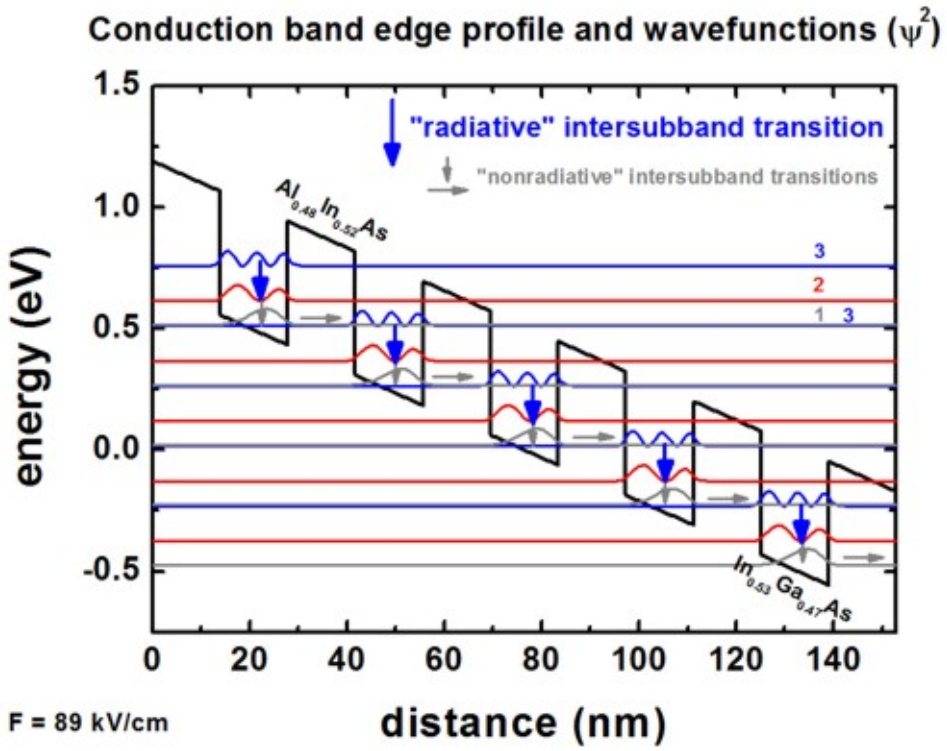
The effective masses that were used for each grid point can be found in these files.

- *Structure/charge_carrier_masses.dat* (*nextnano++*)
- *material_parameters/conduction_band_masses.dat* (*nextnano*^{3})

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:
  - *nextnano++*: 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.
  - *nextnano³*: effective mass tensor components of Gamma, L and X valleys in units of [m0].

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/intraband_matrix_elements_quantum_region_Gamma_100.txt* (*nextnano++*)
- *bias_0000/Quantum/transition_energies_quantum_region_Gamma_Gamma.txt* (*nextnano++*)

In the output file of the *nextnano³* sample file, we can already have them here:

- *wavefunctions/intraband_pz_cb1_sg1_deg1.txt* (*nextnano³*)

See *Model for optical spectra within Fermi's golden rule* for more information on the matrix elements.

Input Files for *nextnano*[3]:

- *1DQCL_simple_nn3.in*

Last update: 28/05/2024

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␣
↪to the same value as in nextnano3
```

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 6.4.11.1 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
}
```

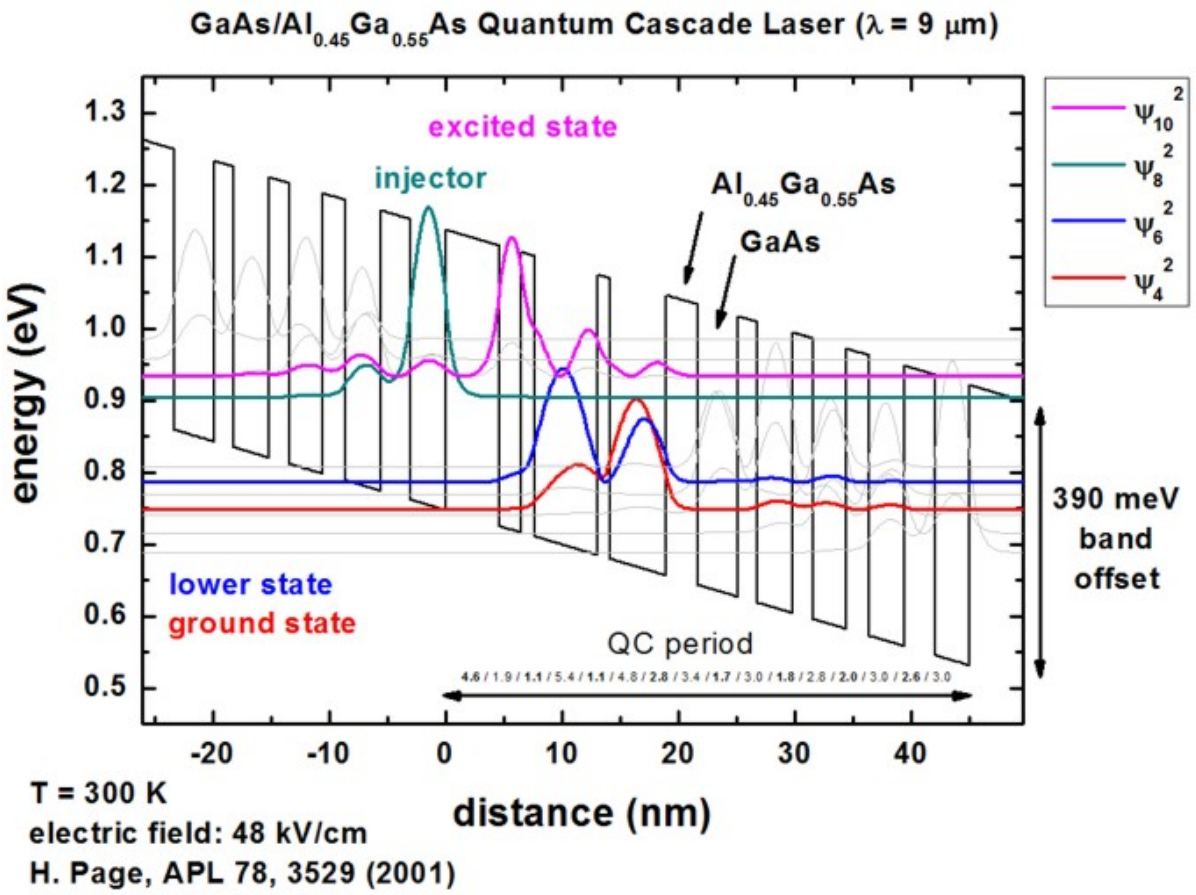


Figure 6.4.11.1: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 48 kV/cm applied. Also shown are the probability densities (Ψ^2) of four electron states, which are shifted by their corresponding eigenenergies.

Intraband matrix elements

The files:

- *bias_00000\Quantum\interband_matrix_elements_qr1_Gamma_100.txt*
- *bias_00000\Quantum\dipole_moment_matrix_elements_qr1_Gamma_100.txt*

contain the p_x and z intraband 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]*:

	nextnano[3]	*[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 6.4.11.1. The corresponding input is *1DQuantumCascadeLaser.in*.

2. λ = 9.4 μm or 132 meV

The simulated quantum-cascade structure, shown in Figure 6.4.11.2, is based on *[Sirtori1998]*. The corresponding input file is *1DQCL_AlGaAs_Sirtori_APL73_1998.in*.

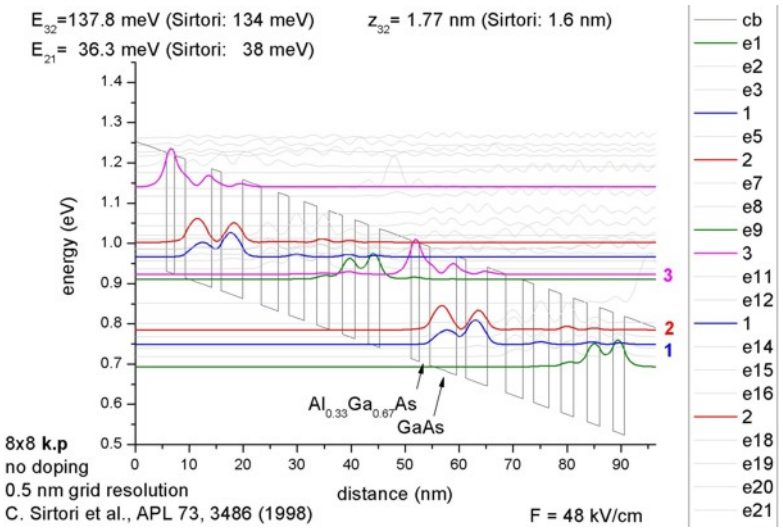


Figure 6.4.11.2: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 48 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

3. λ = 10 μm or 124 meV (77 K)

The simulated quantum-cascade structure, shown in Figure 6.4.11.3 and Figure 6.4.11.4, is based on *[Friedrich2005]*. The corresponding input file is *1DQCL_Andrea_Friedrich_NoInjector_InGaAs_APL86_2005_kp.in*.

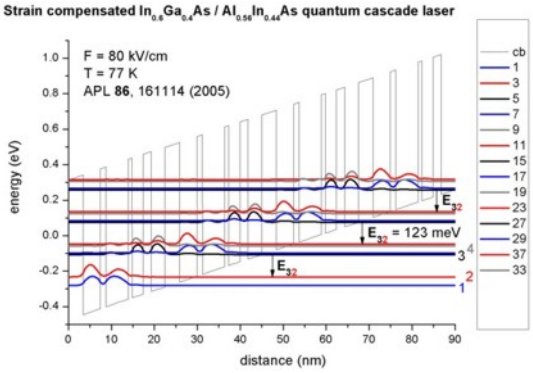


Figure 6.4.11.3: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 80 kV/cm applied (T = 77 K). Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

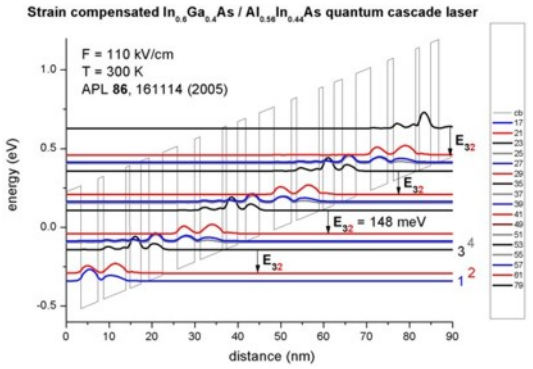


Figure 6.4.11.4: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 110 kV/cm applied (T = 300 K). Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

4. λ = 66 μm, i.e. 4.54 THz or 18.8 meV

The simulated quantum-cascade structure, shown in Figure 6.4.11.5, 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 6.4.11.6, is based on *[Hu2005]*. The corresponding input file is *1DQCL_THz_MIT_Sandia_SemicScTech20_2005.in*.

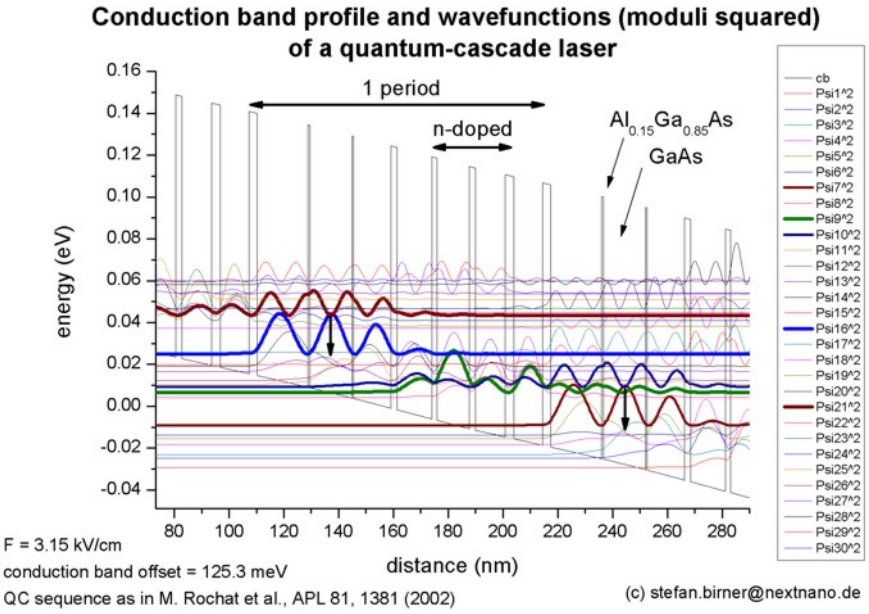


Figure 6.4.11.5: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 3.15 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

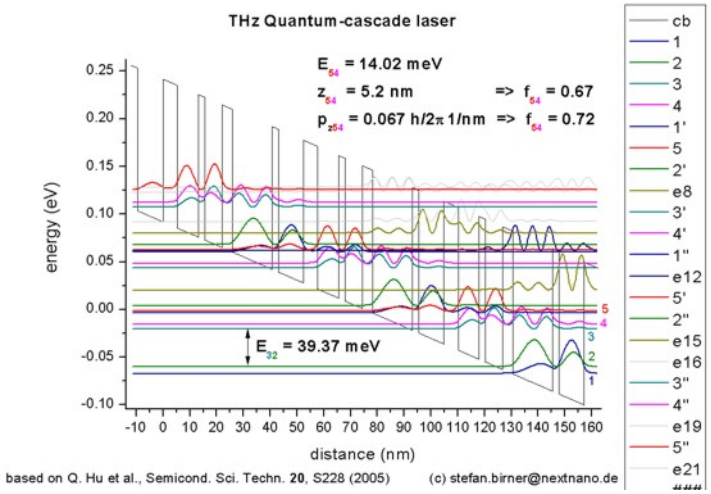


Figure 6.4.11.6: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 12.2 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

6. λ = 107 μm, i.e. 2.8 THz or 11 meV

The simulated quantum-cascade structure, shown in Figure 6.4.11.7, is based on *[Andrews2008]*. The corresponding input file is *THzQCL_Andrews_Vienna_MatSciEng2008_nnp.in*.

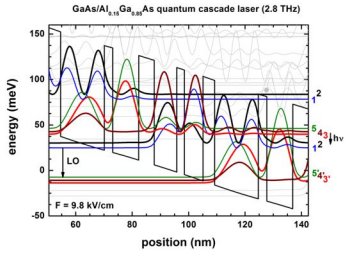


Figure 6.4.11.7: Calculated conduction band edge (black line) of the quantum-cascade structure with electric field of strength 9.8 kV/cm applied. Also shown are the probability densities (Ψ^2) of several electron states, which are shifted by their corresponding eigenenergies.

7. λ = 9.9 μm, i.e. 30.2 THz or 125 meV

The simulated quantum-cascade structure, shown in Figure 6.4.11.8, is based on *[Dehlinger2000]*. This corresponding input file is *1DQuantumCascadeLaserSiGe_nnpp.in*.

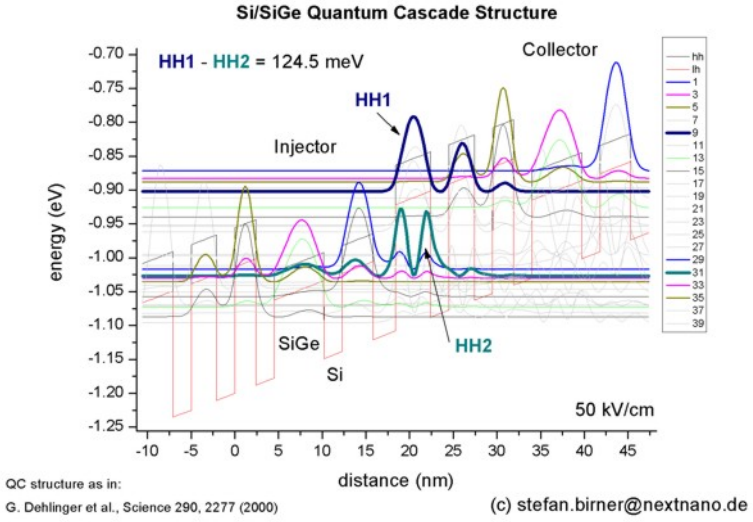


Figure 6.4.11.8: Calculated valance band edge (black line) of the quantum-cascade structure with electric field of strength 50 kV/cm applied. Also shown are the probability densities (Ψ^2) of several hole states, which are shifted by their corresponding eigenenergies.

Last update: nn/nn/nnnn

6.4.12 Optical Spectra and Transitions

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 $\mathbf{k \cdot p}$ 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

$\mathbf{k}_\parallel$ 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 $\mathbf{k}_\parallel = k_y\hat{y} + k_z\hat{z}$ (1D) and $\mathbf{k}_\parallel = k_z\hat{z}$ (2D). Under envelope function approximation the $\mathbf{k}\cdot\mathbf{p}$ model yields the following equation to determine the confined states in structured directions

$$\sum_{\mu=1}^{8} H_{\nu\mu}^{\mathrm{kp8}}(\mathbf{k}_\parallel, \mathbf{r}_\perp) f_{m,\mu}(\mathbf{r}_\perp) = E_m(\mathbf{k}_\parallel) f_{m,\nu}(\mathbf{r}_\perp) \quad (\nu = 1, \ldots, 8), \tag{6.4.12.1}$$

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 $\mathbf{k}_\parallel$ as

$$\Psi_n(\mathbf{k}_\parallel, \mathbf{r}) = \sum_{\mu=1}^{8} F_{m,\mu}(\mathbf{k}_\parallel, \mathbf{r}) u_\mu(\mathbf{r}) = \sum_{\mu=1}^{8} \frac{e^{i\mathbf{k}_\parallel\cdot\mathbf{r}_\parallel}}{\sqrt{A}} f_{m,\mu}(\mathbf{r}_\perp) u_\mu(\mathbf{r}), \tag{6.4.12.2}$$

where $u_\mu(\mathbf{r})$ is the Bloch function of the band μ at $\mathbf{k} = 0$ and $A = \int d\mathbf{r}_\parallel$. 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 $\mathbf{k}_\parallel$ 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
            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 $\mathbf{k}_\parallel = 0$. 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 $n \rightarrow m$) in a condensed matter is calculated from Fermi's golden rule *[ChuangOpto1995]*. The absorption coefficient has the dimension of (length)$^{-1}$.

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega} \frac{1}{V} \sum_{n>m} \sum_{\mathbf{k}_\parallel} |\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathbf{k}_\parallel)|^2 (f_m - f_n)\delta(E_n - E_m - \hbar\omega), \tag{6.4.12.3}$$

where the first sum runs over bands that fulfill $E_n > E_m$, and $f_m(\mathbf{k}_\parallel) = [1 + e^{[E_m(\mathbf{k}_\parallel)-E_F]/k_BT}]^{-1}$ is the occupation of eigenstate m. When `optics{ occupation_ignore=yes }` (default is `no`), the program assumes

$$\begin{cases} f_m(\mathbf{k}_\parallel) = 0 \text{ if } m \in \text{conduction band} \\ f_m(\mathbf{k}_\parallel) = 1 \text{ if } m \in \text{valence band} \end{cases}$$

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
      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

$$\vec{\pi}_{nm}(\mathbf{k}_\parallel) = \langle n|\vec{\pi}|m\rangle = \int d\mathbf{r} \begin{pmatrix} F_{n1}^* & \cdots & F_{n8}^* \end{pmatrix} \begin{pmatrix} & & \\ & \vec{\pi}_{\nu\mu}^{\mathrm{kp8}} & \\ & & \end{pmatrix} \begin{pmatrix} F_{m1} \\ \vdots \\ F_{m8} \end{pmatrix}, \tag{6.4.12.4}$$

where the 8×8 matrix representation of the momentum operator, $\vec{\pi}_{\nu\mu}^{\mathrm{kp8}}$, 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␣
↪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 $r_{\mathrm{opt}}, N_{\mathrm{opt}}, N'_{\mathrm{opt}}$) 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 6.4.12.1).

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 $2k_BT$ or $3k_BT$ should be sufficient. Usually $r_{\mathrm{opt}} = 0.3$ 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

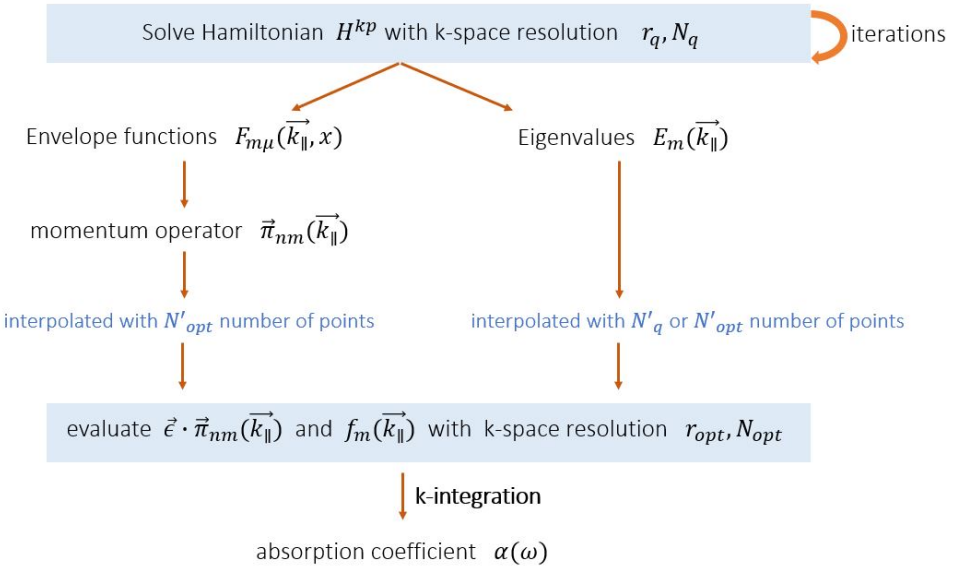
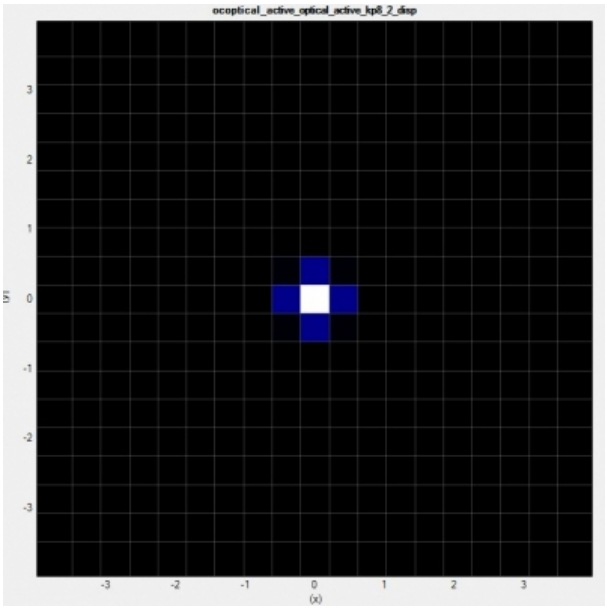


Figure 6.4.12.1: Calculation algorithm of optical absorption spectrum and its relation to the parameters in `k_integration{}`. r_q, N_q, N'_q and $r_{\rm opt}, N_{\rm opt}, N'_{\rm opt}$ are specified in `quantum{ }` and `optics{ }`, respectively. *To do; the energy dispersion is interpolated with* $N'_{\rm q}$ *or* $N'_{\rm opt}$*?*

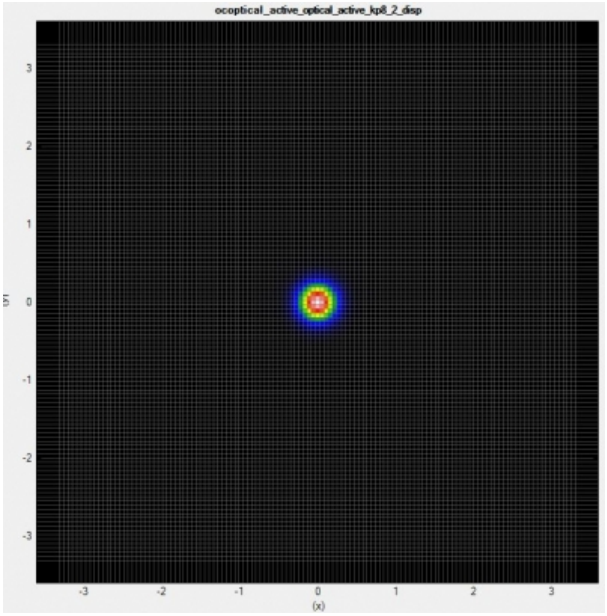
nextnanomat (see also *Output*). This shows the occupation $f_m(\mathbf{k}_\parallel)$ as a function of $\mathbf{k}_\parallel$. Let us consider 1D simulation and suppose you got the following:

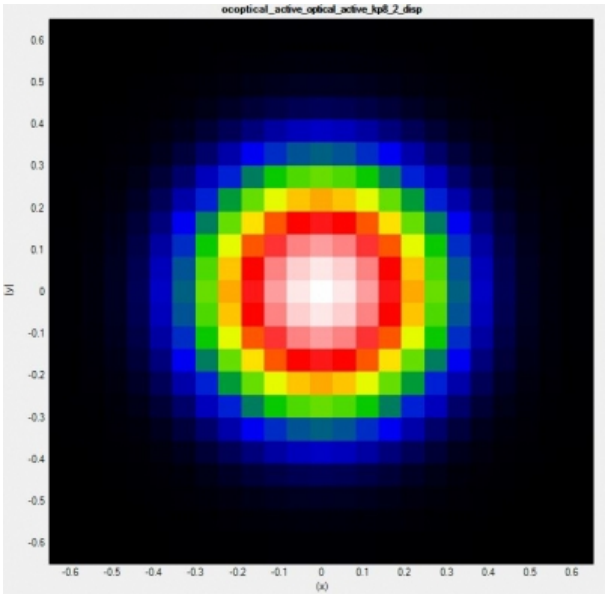


where $(r_{\rm opt}, N_{\rm opt}) = (0.3, 8)$. The horizontal- and vertical axes are k_y and k_z, respectively. The area $|k_{y,z}| \leq r_{\rm opt}\frac{\pi}{a}$ is shown with the k||-space gridding (thin white lines). The number of k|| points in one direction is $2N_{\rm opt} + 3$. The occupation profile is not smooth, and you might want a higher resolution by increasing the parameter $(r_{\rm opt}, N_{\rm opt}) \rightarrow (0.3, 60)$:

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)$\propto (r_{\rm opt}N_{\rm opt})^2$). 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 $r_{\rm opt}$ and $N_{\rm opt}$ in such a way that the ratio $r_{\rm opt}/N_{\rm opt}$ remains constant. This will cut down the irrelevant region without changing the resolution. For example, if you set $(r_{\rm opt}, N_{\rm opt}) = (0.05, 10)$, you obtain

and this should be sufficient for the k||-space integration.


ocoptical_active_optical_active_kp8_2_disp
(x)


ocoptical_active_optical_active_kp8_2_disp
(x)

After tuning the parameters r_{opt}, N_{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 6.4.12.2 shows that this parameter improves the absorption spectrum.

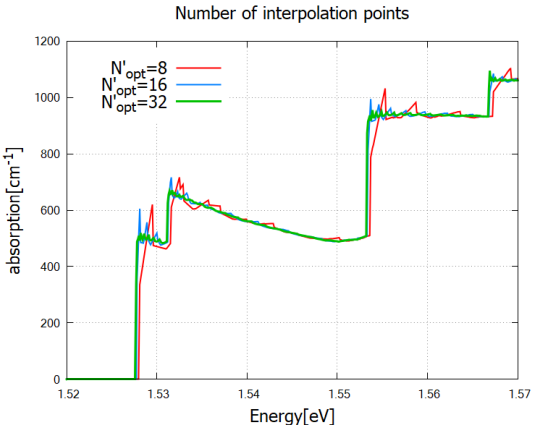


Figure 6.4.12.2: The effect of the parameter N'_{opt} specified in `optics{ k_integration{}}` on absorption spectrum output *\Optics\absorption*. Larger N'_{opt} smoothens the k||-dependence of the integrand, which leads to smoother spectrum.

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 6.4.12.3 and Figure 6.4.12.4, respectively.

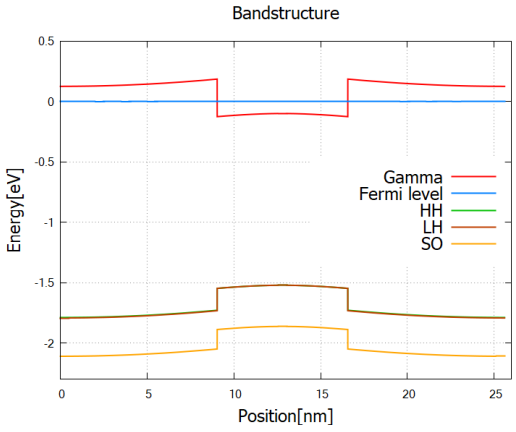


Figure 6.4.12.3: Single quantum well structure `\bandedges.dat`. The bias voltage between two contacts is set to 2mV.

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 6.4.12.7. 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,

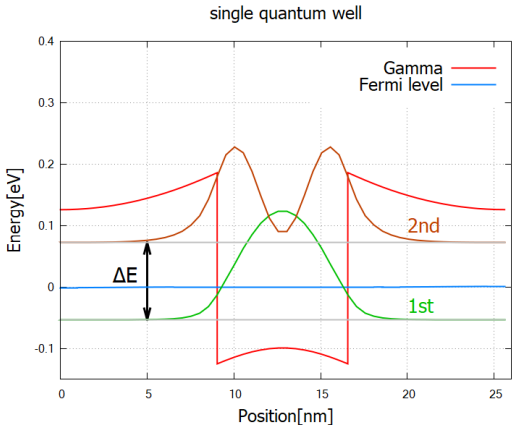


Figure 6.4.12.4: Probability distribution $|\psi(x)|^2$ of the confined states at $\mathbf{k}_{||} = 0$ (`\Quantum\ probabilities_shift_optical_active`). The wave functions here are the solution to the 8-band k.p model. The energy separation is ΔE=0.06960-(-0.05589)=0.1255[eV] according to the output data. The electron Fermi energy lies between two bound states.

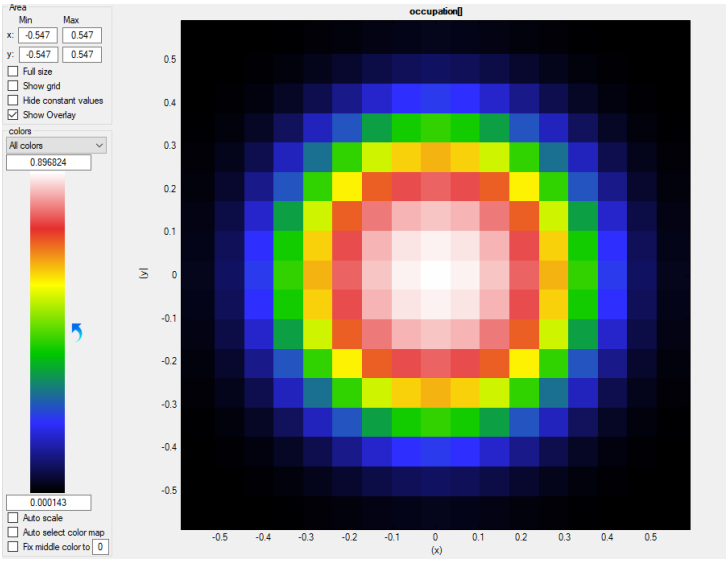


Figure 6.4.12.5: Occupation of the first (m=1) bound states as a function of $\mathbf{k}_{||}$.

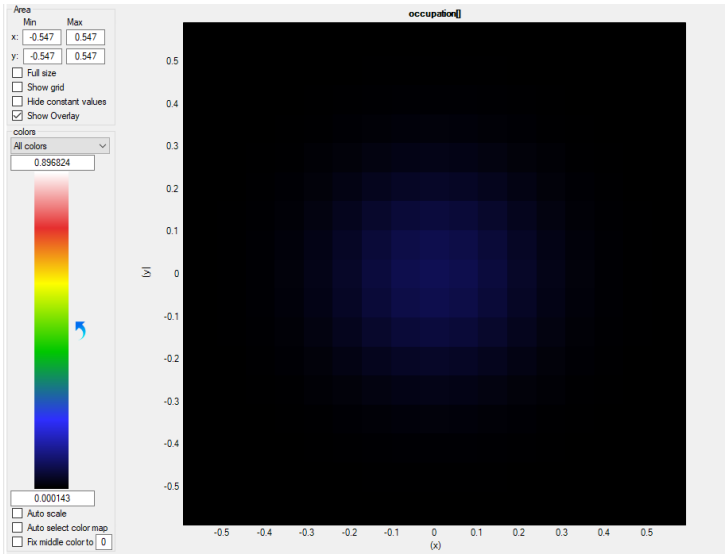


Figure 6.4.12.6: Occupation of the second (m=2) bound states as a function of $\mathbf{k}_{||}$.

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 $\hbar\omega < \Delta E$.

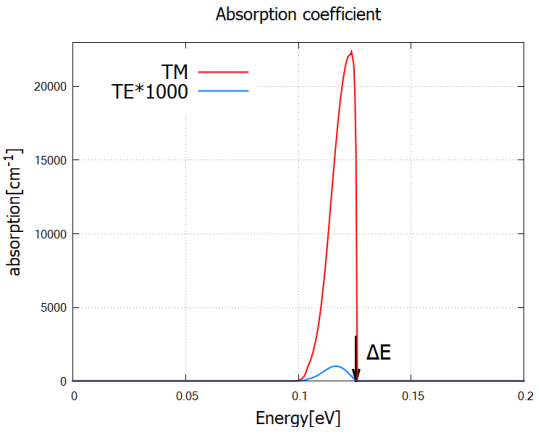


Figure 6.4.12.7: Absorption coefficient in `\Optics\absorption_~.dat` as a function of photon energy, for TE and TM. Black arrow points the energy separation ΔE. The broadening of the spectrum is due to the k||-dependence of wave functions and corresponding eigenvalues.

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 6.4.12.7). 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 6.4.12.8) leads to large band mixing. In fact, the output `\Quantum\spinor_composition_~.dat` shows that the ground states in Figure Figure 6.4.12.8 consists of 80.7% of conduction band and 19.3% of valence band contribution.

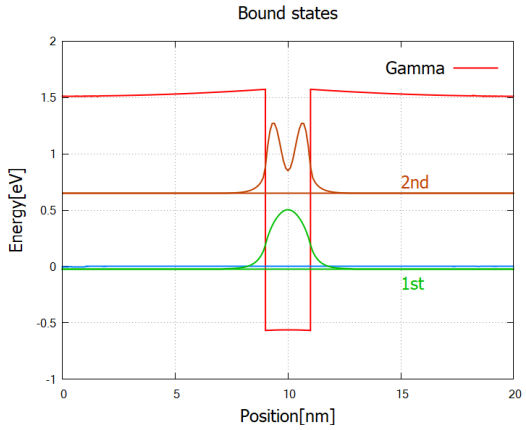


Figure 6.4.12.8: Confined states at $\mathbf{k}_{\parallel} = 0$ (`\Quantum\probabilities_shift_optical_active`) in a narrower and deeper quantum well. The blue line marks the electron Fermi energy (0eV).

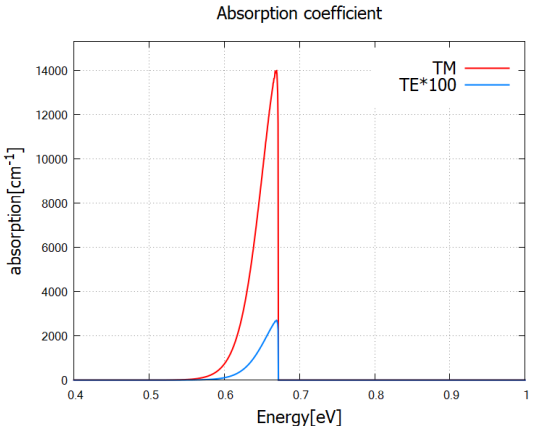


Figure 6.4.12.9: Absorption spectrum for TE and TM. TE absorption becomes relevant compared to Figure 6.4.12.7 because of the large band-mixing. Note that TE spectrum here is multiplied by a factor of 100, instead of 1000 in Figure 6.4.12.7.

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/$Al_xGa_{1-x}As$ 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 6.4.12.10. We have continuum states above the barriers as well as bound states in the superlattice (miniband).

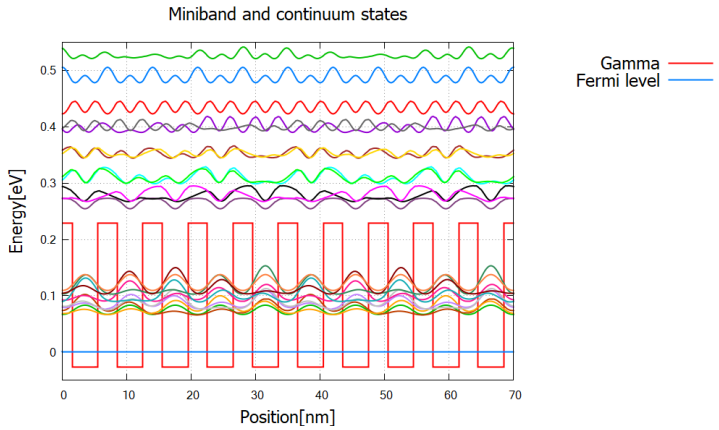


Figure 6.4.12.10: Gamma band profile and probability distribution of the bound miniband states and continuum states above the top of the barriers.

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 over all transitions). The total absorption spectrum for TE and TM polarization looks like this:

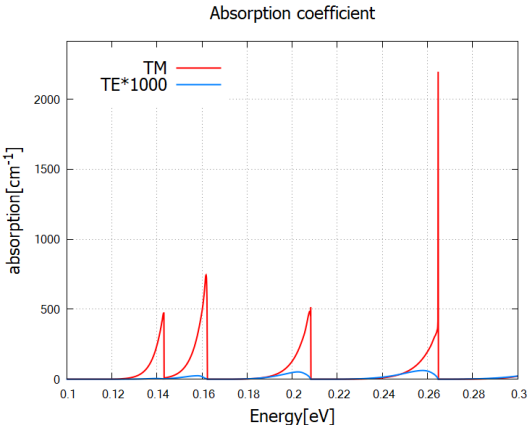


Figure 6.4.12.11: Absorption spectrum for TE ($\vec{\epsilon} = \hat{y}$) and TM ($\vec{\epsilon} = \hat{x}$) polarization. TE spectrum is magnified by factor of 1000. We observe that TM absorption is much larger than TE, while the peak positions are the same.

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 6.4.12.12).

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

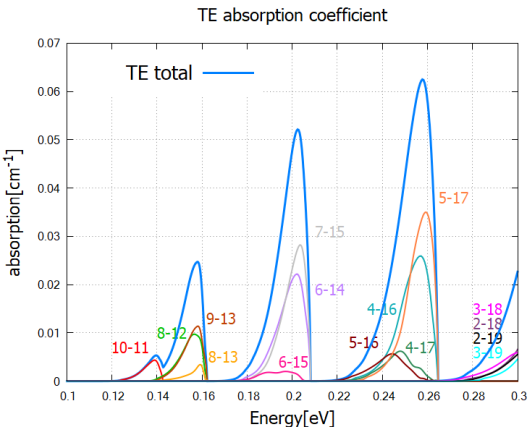


Figure 6.4.12.12: Contributions from different transitions to the total TE absorption spectrum.

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 6.4.12.12, we identify the first four peaks to the following four different transitions (Figure 6.4.12.13). We have confirmed that the peak energies in Figure 6.4.12.12 are consistent to the energy separation of the corresponding states.

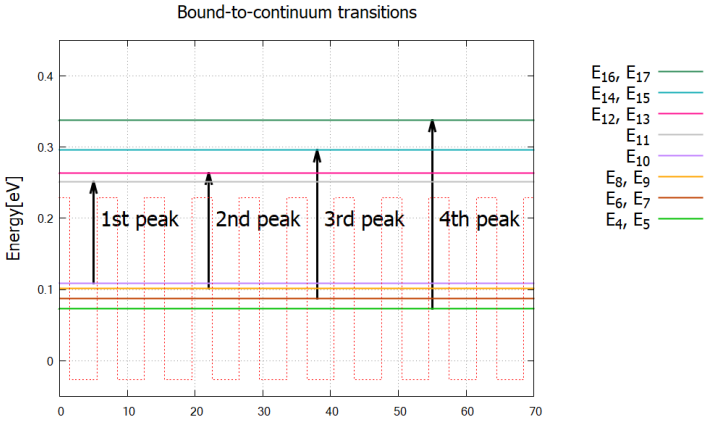


Figure 6.4.12.13: Eigenenergies of relevant bound- and continuum states. Many other transitions have little contribution due to the shape of the wave functions and/or occupation of the states. When we calculate for wider energy range, i.e. increase the parameter `$ENERGY_MAX`, there will be many more peaks that are attributed to higher energy transitions.

Lastly we check the occupation (Fermi-Dirac distribution) $f_m(\mathbf{k}_{\|})$. In the output `\Optics\eigenvaluespectrum` (Figure 6.4.12.14), occupation at k||=0 of m-th state, $f_m(\mathbf{k}_{\|} = 0)$, is plotted at 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 6.4.12.13 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.

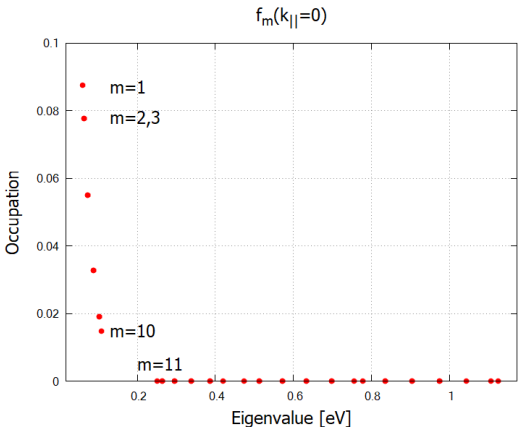


Figure 6.4.12.14: Occupation of eigenstates showing a noticeable difference for bound (m=1-10) and continuum (m=11,...) states.

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:

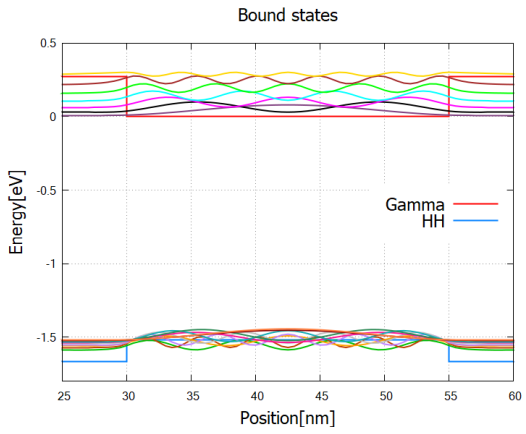


Figure 6.4.12.15: The conduction band edge profile (`bandedges.dat`) and wave functions of the bound states (`\Quantum\probabilities_shift`).

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 6.4.12.16 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 $E_n(\mathbf{k}_\parallel) - E_m(\mathbf{k}_\parallel)$ in interband transitions are concave upward $\sim +k^2$ (here we do not consider Type 2 semiconductors).

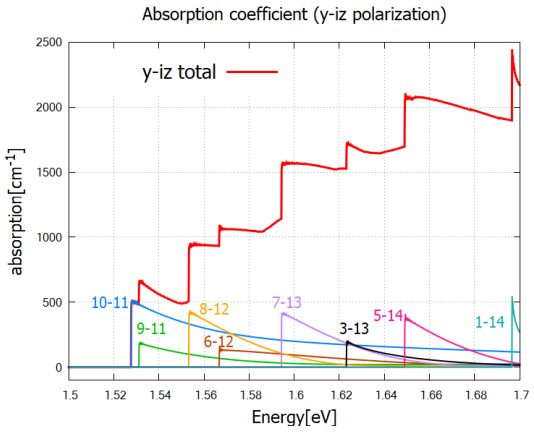
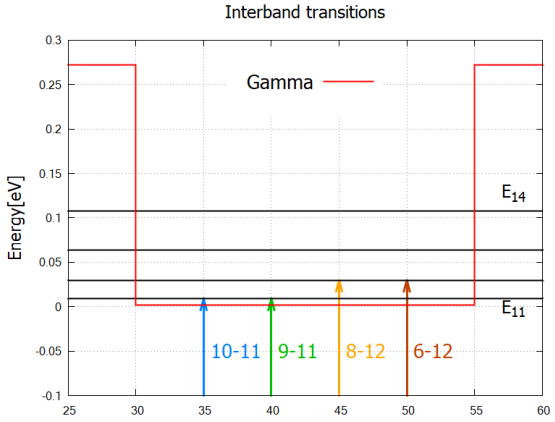


Figure 6.4.12.16: Absorption coefficient of circularly polarized lights. Numbers "m-n" denote each transition $m \to n$. The first four transitions are sketched in Figure 6.4.12.17 .

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 $\mathbf{k}_\parallel = 0$, the reliable 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

$d = \min_{\mathbf{k}_\parallel \in\, \Omega^* \text{ edge}} |E_n(\mathbf{k}_\parallel) - E_m(\mathbf{k}_\parallel)|$

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.

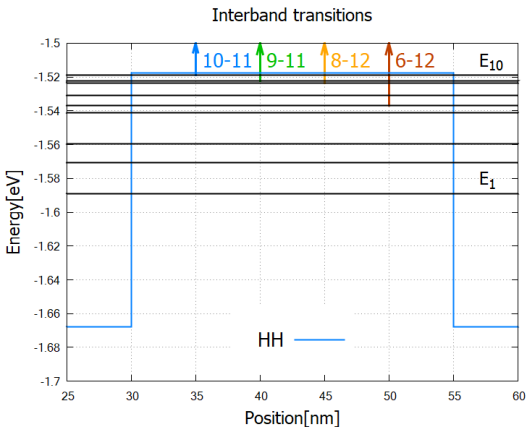


Figure 6.4.12.17: Eigenvalues (black) and transitions from valence-band to conduction band bound states (arrows) which are responsible for the first four steps in Figure 6.4.12.16. Here spin-degenerate states are counted as one state (eigenstate numbering in `optics{ }`).

Doping and Schottkey barrier

In the second input file `AlGaAs_QW_Frankenberger_Doping_schottky07_nnp.in`, we consider the following structure:

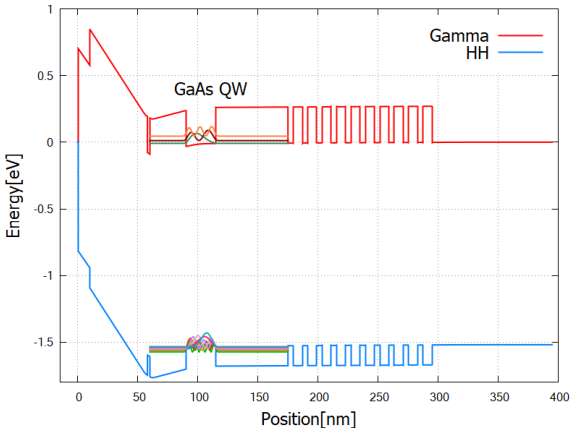


Figure 6.4.12.18: The band structure and eigen functions used for optics calculation. The Fermi level is at 0eV.

Figure 6.4.12.20 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 transitions to conduction band states just above the Fermi energy are prohibited because of the thermal distribution of electrons.

Last update: nn/nn/nnnn

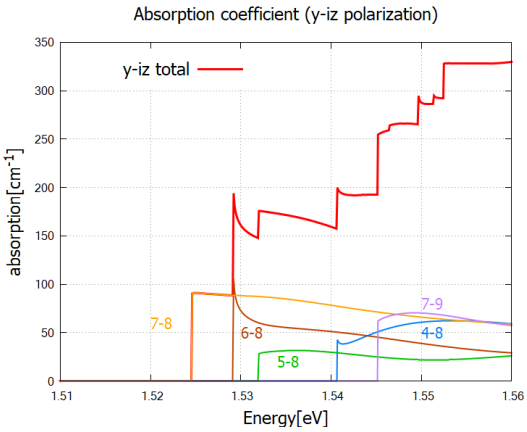


Figure 6.4.12.19: Absorption coefficient of circularly polarized lights. Numbers "m-n" denote each transition $m \to n$.

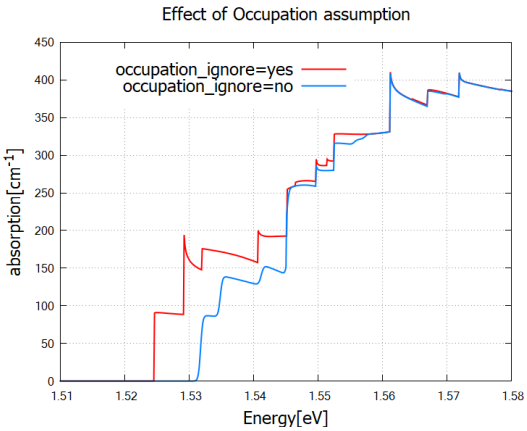


Figure 6.4.12.20: Absorption coefficient for different settings of occupation. The red curve is identical to the total absorption spectrum in Figure 6.4.12.19. When `occupation_ignore=no`, absorption of low energy photons is suppressed due to the occupation of the lowest conduction band states (also see Figure 6.4.12.18).

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 don't consider L and X bands here)
- 5 confined heavy hole states
- 2 confined light hole states
- 3 confined split-off hole states

Figure 6.4.12.21 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.

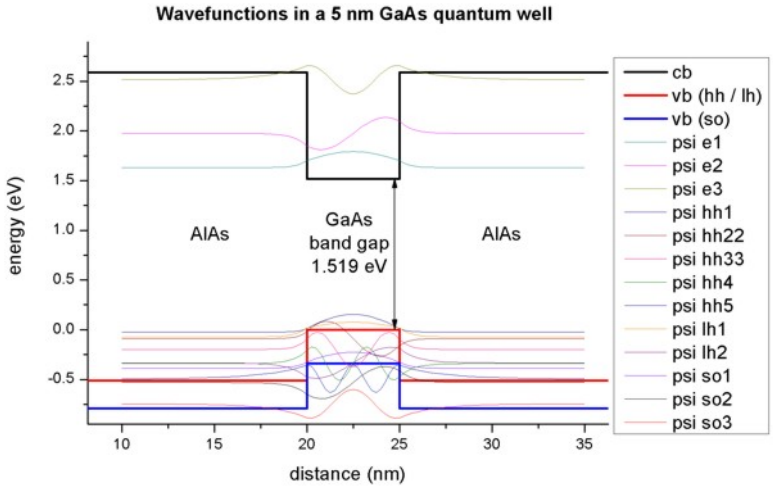


Figure 6.4.12.21: Calculated conduction band edge (black), hh/ lh valence bands (red) and split-off hole valence band (blue) with wave functions of lowest electron and hole states.

As one can see, the valence band looks rather messy. Thus, we zoom into it, see Figure 6.4.12.22 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.

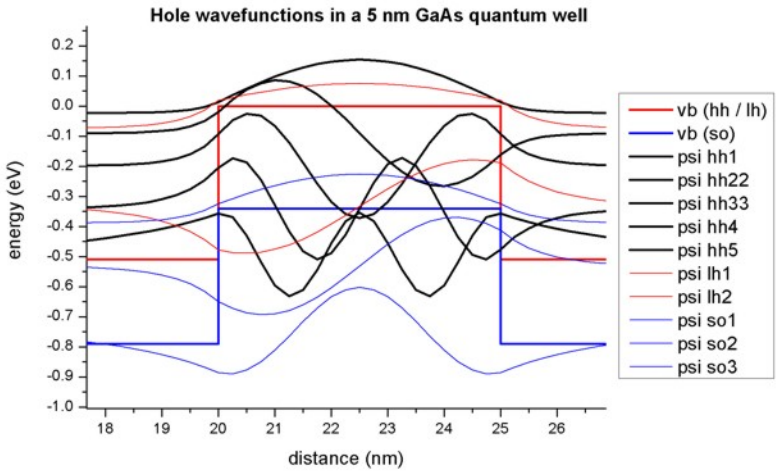


Figure 6.4.12.22: Calculated valence band edges and hole wave functions. The 5 heavy hole wave functions are indicated in black, the 2 light hole wave function in red and the 3 split-off hole wave functions in blue.

Interband matrix elements

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{
    ...
    interband_matrix_elements{ # output matrix elements
        HH_Gamma{}
        LH_Gamma{}
        SO_Gamma{}
    }
}
```

The spatial overlap integrals of the envelope functions are contained in these files:

- *bias_00000Quantuminterband_matrix_elements_quantum_region_HH_Gamma.txt* - (heavy hole)
- *bias_00000Quantuminterband_matrix_elements_quantum_region_LH_Gamma.txt* - (light hole)
- *bias_00000Quantuminterband_matrix_elements_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)
```

(continues on next page)

(continued from previous page)

```
<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 6.4.12.23 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)

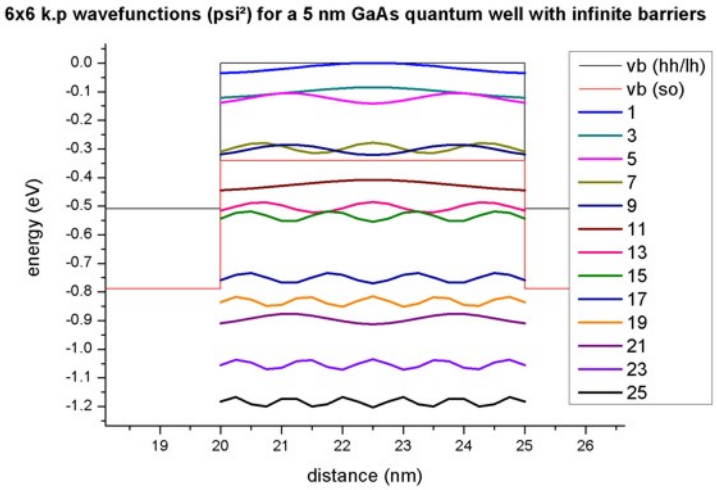


Figure 6.4.12.23: 6-band k.p wave functions (Psi^2) for a 5 nm $GaAs$ quantum well with finite barriers

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 6.4.12.24 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.

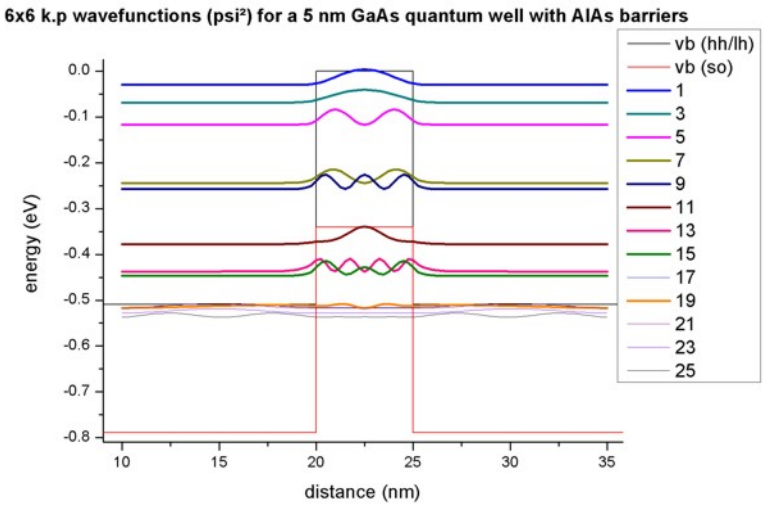


Figure 6.4.12.24: 6-band k.p wave functions (Psi^2) for a 5 nm $GaAs$ quantum well with $AlAs$ barriers

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.

Last update: nn/nn/nnnn

Optical intraband transitions in a quantum well - Intraband 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{
    ...
    intraband_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 6.4.12.25 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 = 2^2 E1)
E3 = 0.50831 eV    (0.50891 = 3^2 E1)
E4 = 0.90314 eV    (0.90473 = 4^2 E1)
```

(continues on next page)

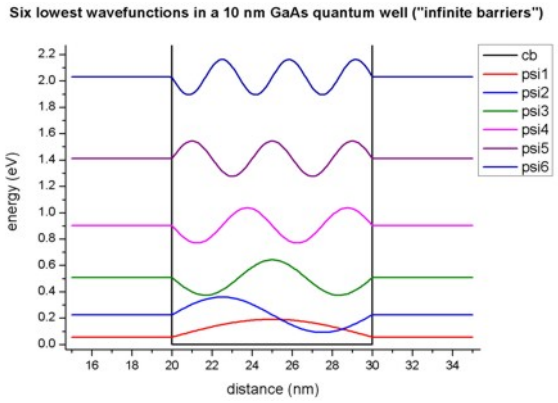


Figure 6.4.12.25: Calculated conduction band edge (black) and wave functions of confined electron states.

(continued from previous page)

```
E5 = 1.41011 eV    (1.41365 = 5^2 E1)
E6 = 2.02872 eV    (2.03565 = 6^2 E1)
```

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.

Intraband 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:

$$M_{fi} = \langle \psi_\text{f} | x | \psi_\text{i} \rangle = \int_{-\infty}^{\infty} \psi_f^*(x) x \psi_i(x) dx$$

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
```

(continues on next page)

(continued from previous page)

```
...
<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
<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} = 8ħ/3L = 0.2666 ħ/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\intraband_matrix_elements_quantum_region_Gamma_100.txt*
- *bias_00000\Quantum\intraband__matrix_elements_quantum_region_HH_100.txt*
- *bias_00000\Quantum\intraband_matrix_elements_quantum_region_LH_100.txt*
- *bias_00000\Quantum\intraband_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\intraband_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
```

(continues on next page)

(continued from previous page)

```
<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
<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
```

Last update: nn/nn/nnnn

— NEW — 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 6.4.12.26)

- *bias_00000\Quantum\probabilities_shift_quantum_region_kp8_00000.dat* - probability distributions (see Figure 6.4.12.26)
- *bias_00000\Quantum\absorption_coeff_quantum_region_TE_y_eV.dat* - absorption spectrum TE (see Figure 6.4.12.27)
- *bias_00000\Quantum\absorption_coeff_quantum_region_TM_z_eV.dat* - absorption spectrum TM (see Figure 6.4.12.27)

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 $k_{\parallel} = 0$) are showed in the Figure 6.4.12.26. 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*.

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

$$E_1 = 1.028\ \mathrm{eV} - [-0.275\ \mathrm{eV}] = 1.303\ \mathrm{eV},$$
$$E_2 = 1.118\ \mathrm{eV} - [-0.302\ \mathrm{eV}] = 1.420\ \mathrm{eV}.$$

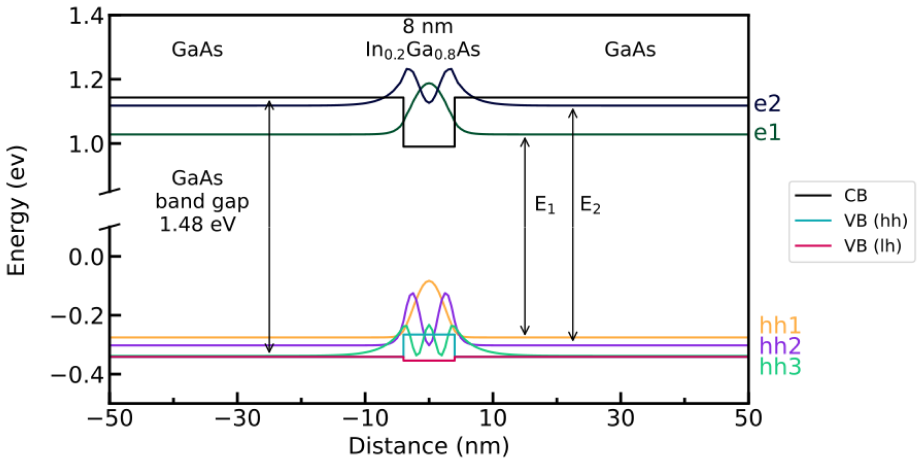


Figure 6.4.12.26: Energy profiles and probability distributions of confined electrons and holes states at $k_{||} = 0$. The conduction band is labeled as **CB**. The heavy-hole valence bands is denoted as **VB (hh)** while the light-hole valence band as **VB (lh)**. The first and the second electron 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**.

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 6.4.12.27 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 6.4.12.27.

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

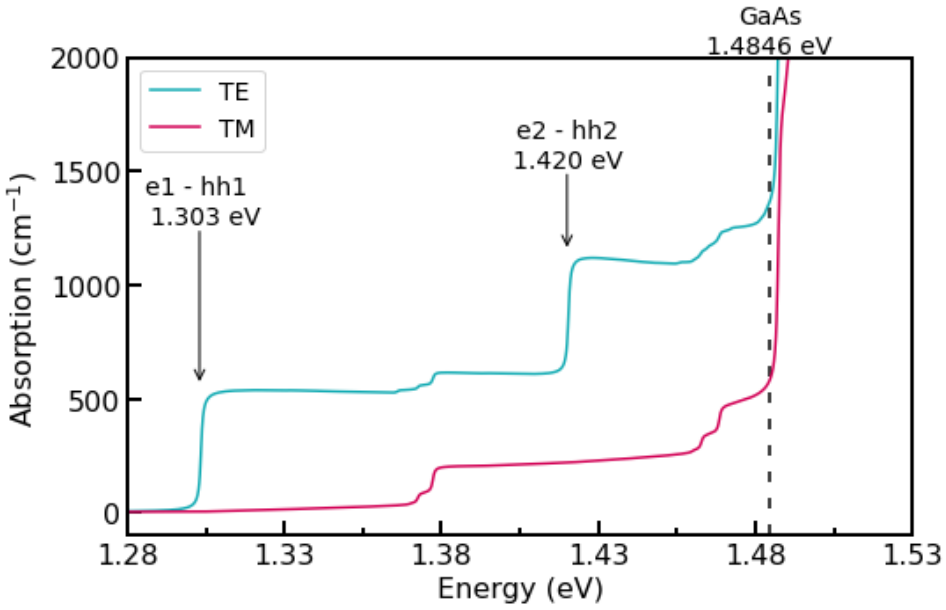


Figure 6.4.12.27: Absorption spectrum for TE (turquoise) and TM (magenta) modes of optical field.

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`.

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*[3]:

- *1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_sg_nn3.in*
- *1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_kp_nn3.in*

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.

Input files for both the *nextnano++* and *nextnano*[3] tools are available.

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 GmbH
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 inter-subband 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 GmbH
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 documentation and figures were generated automatically using *nextnanopy*.

The following Python script was used: `intersubband_InfiniteQW_nextnano3.py`

The following figures have been generated using *nextnano*[3]. 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)

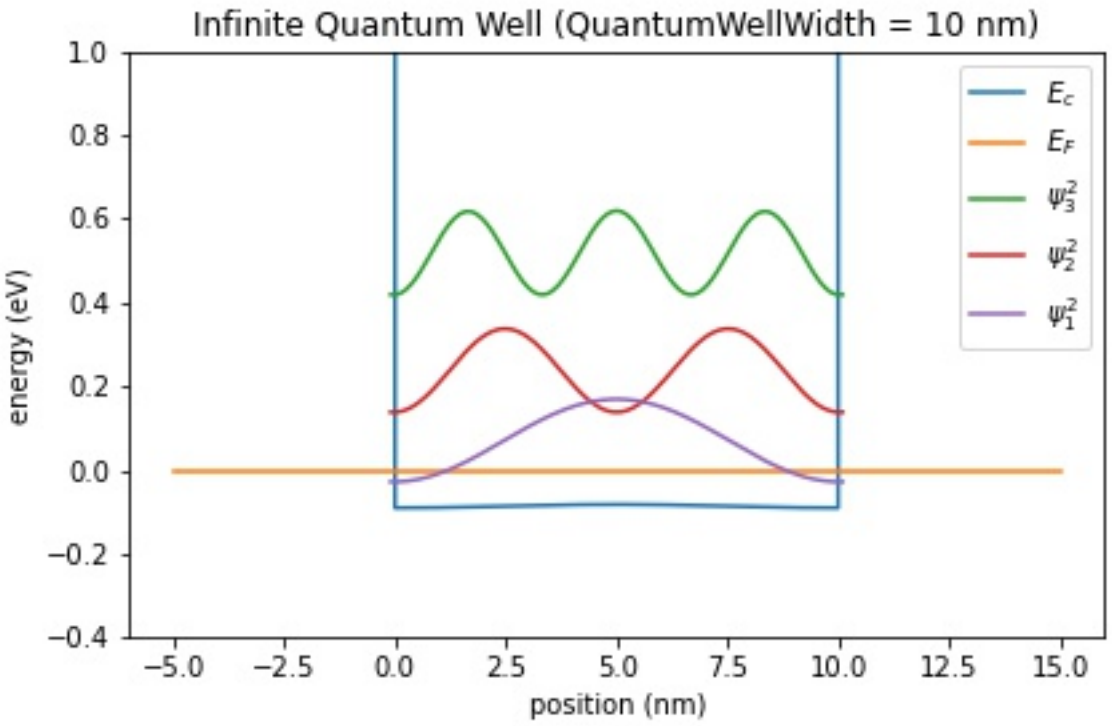


Figure 6.4.12.28: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 10 nm)

Infinite Quantum Well (QuantumWellWidth = 13 nm)

Infinite Quantum Well (QuantumWellWidth = 16 nm)

Infinite Quantum Well (QuantumWellWidth = 19 nm)

Parameter sweep: Well width

Figure 6.4.12.37 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 6.4.12.38 shows the absorption spectra for different **doping concentrations** (Variable: `$DopingConcentration`). The peak absorption coefficient increases with the doping concentration N_D.

Last update: nn/nn/nnnn

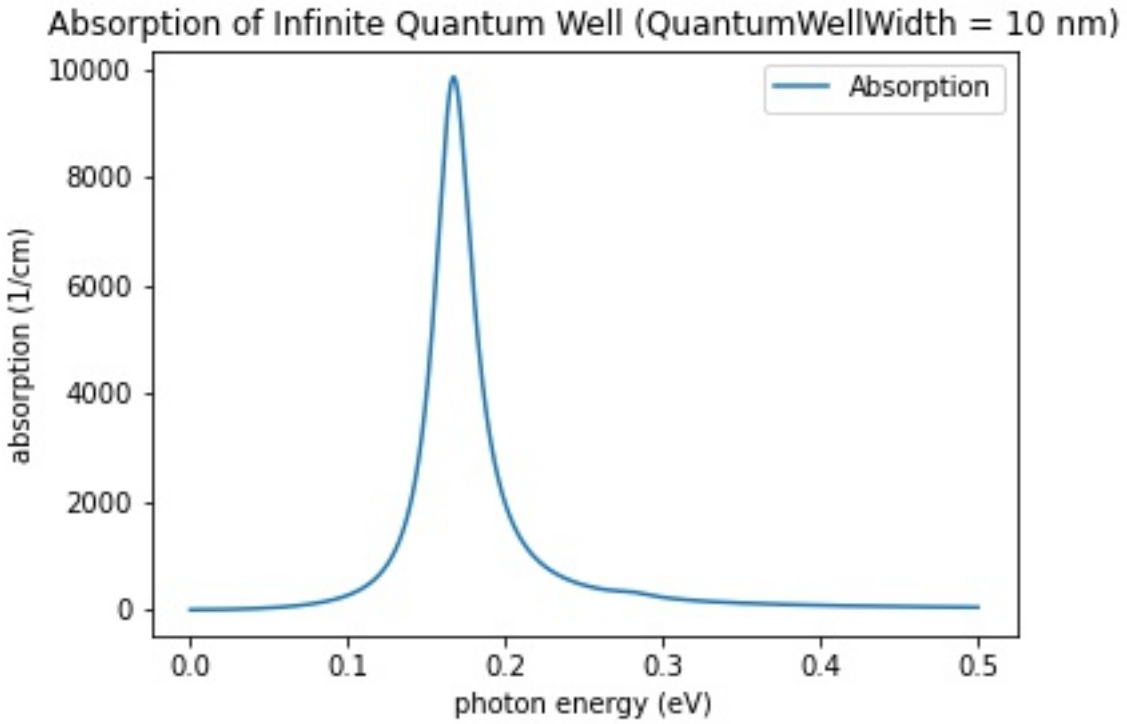


Figure 6.4.12.29: Calculated absorption spectra $\alpha(E)$ of an infinite quantum well (QuantumWellWidth = 10 nm)

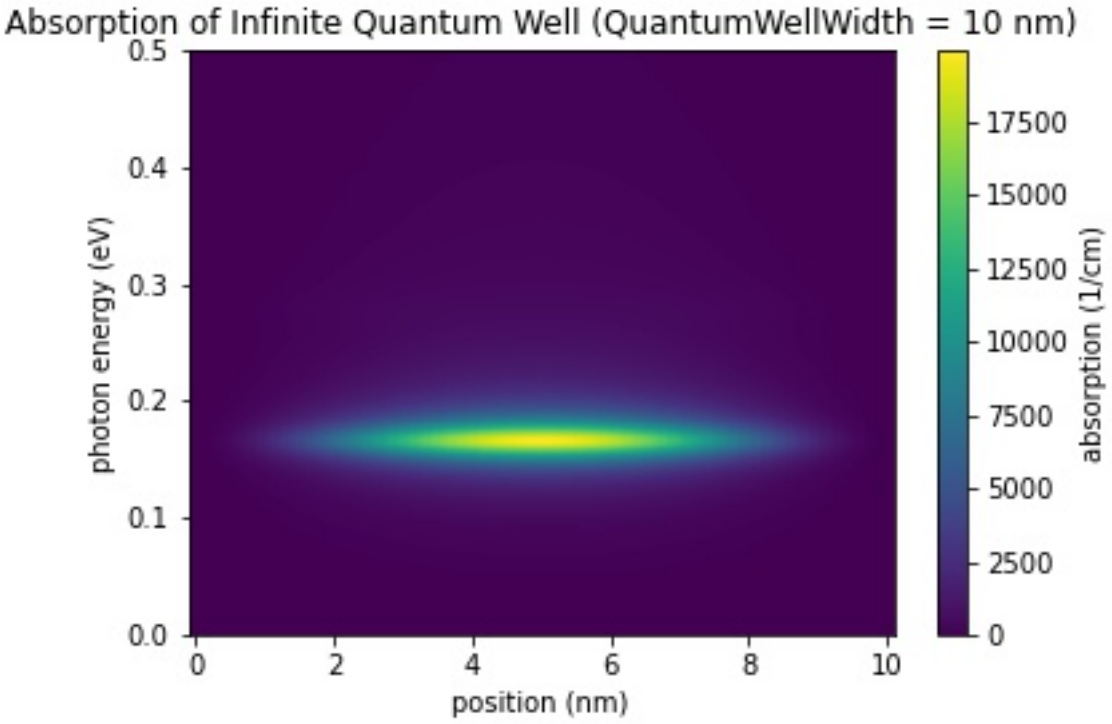


Figure 6.4.12.30: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 10 nm)

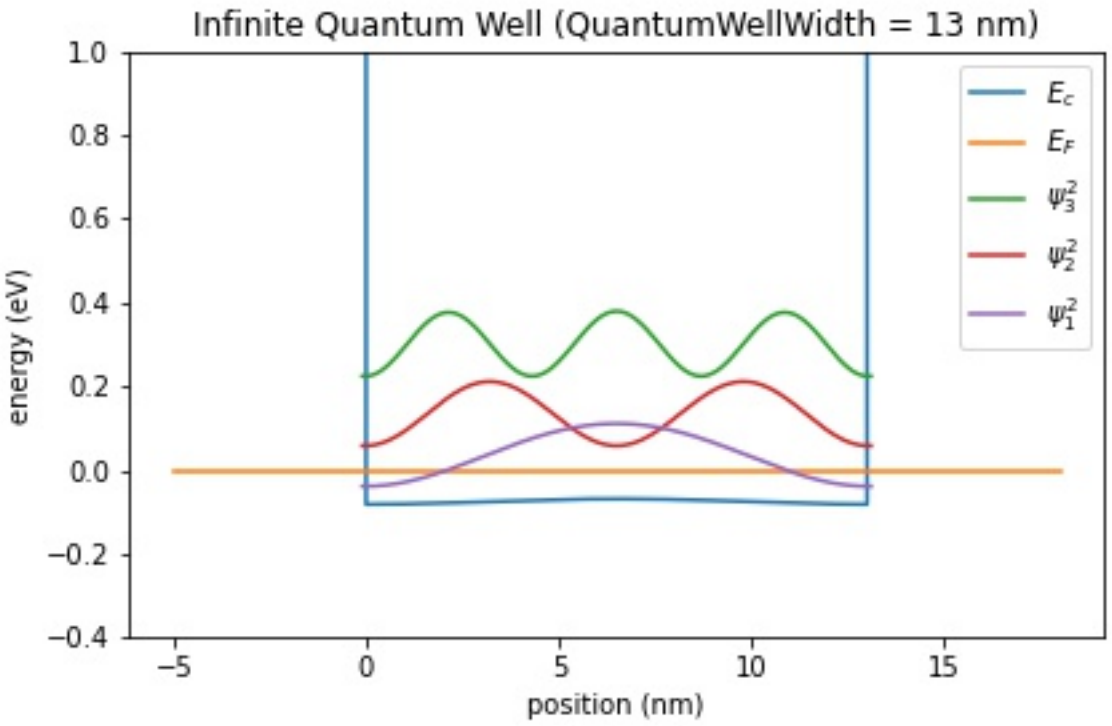


Figure 6.4.12.31: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 13 nm)

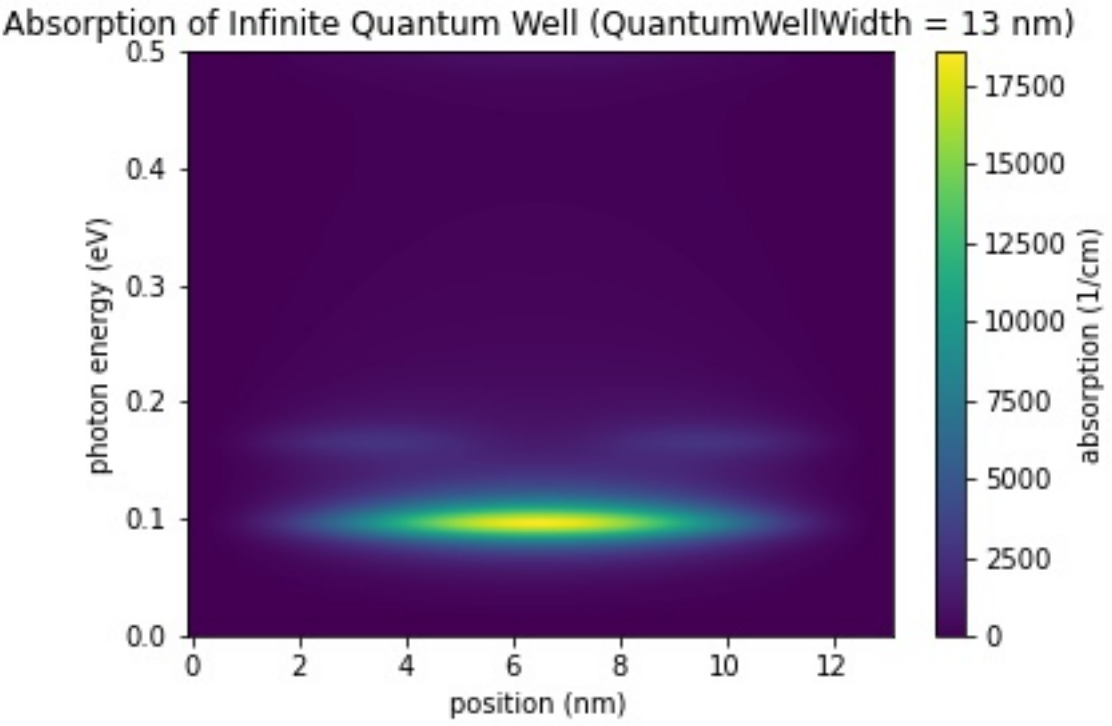


Figure 6.4.12.32: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 13 nm)

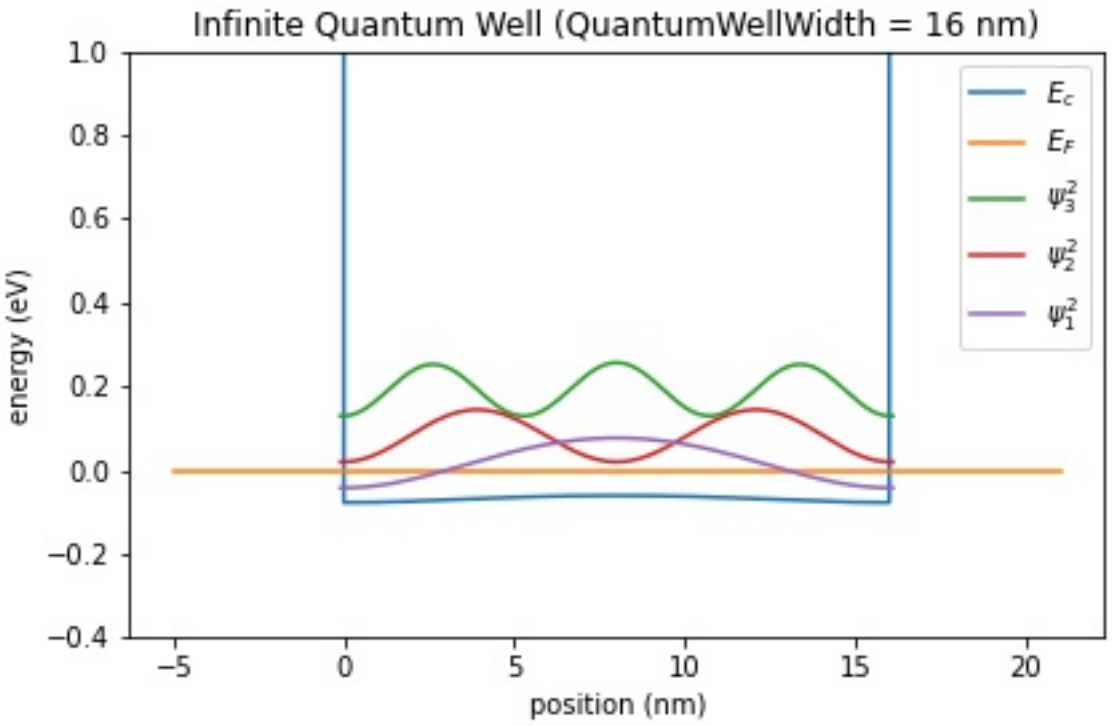


Figure 6.4.12.33: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 16 nm)

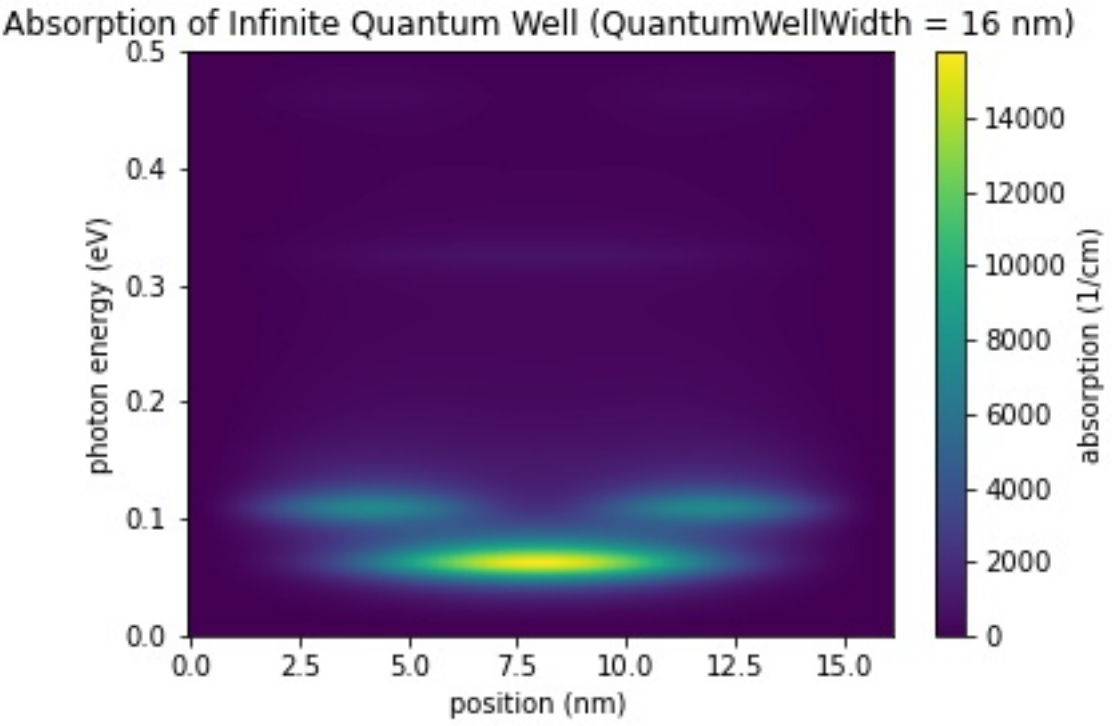


Figure 6.4.12.34: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 16 nm)

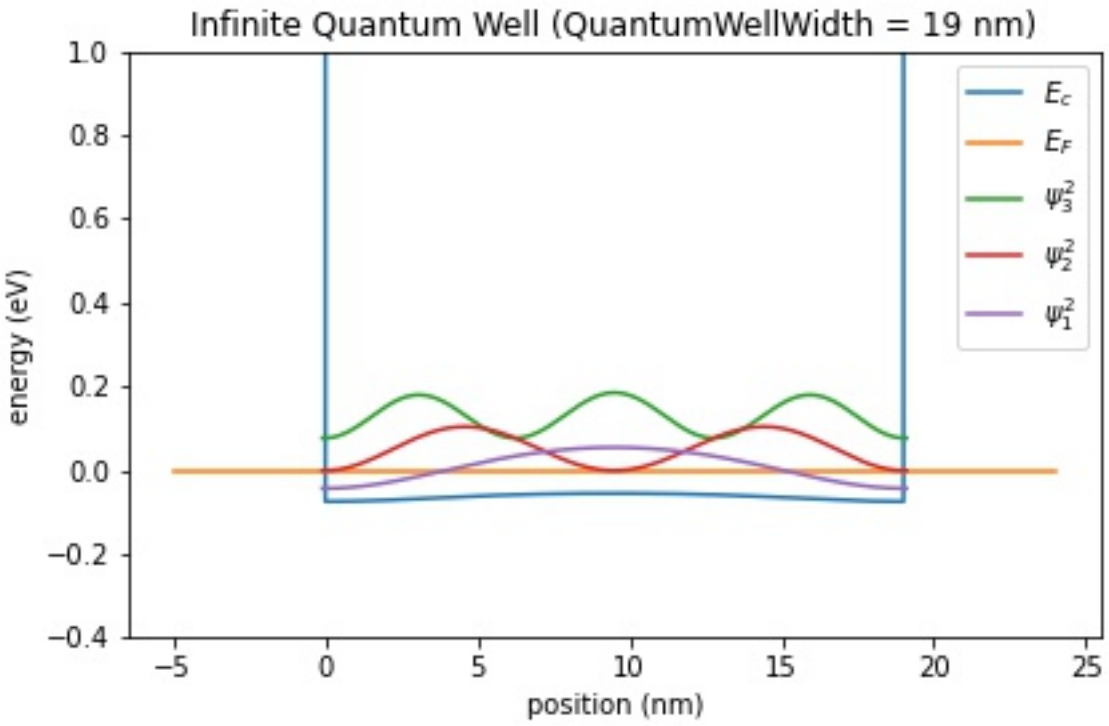


Figure 6.4.12.35: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 19 nm)

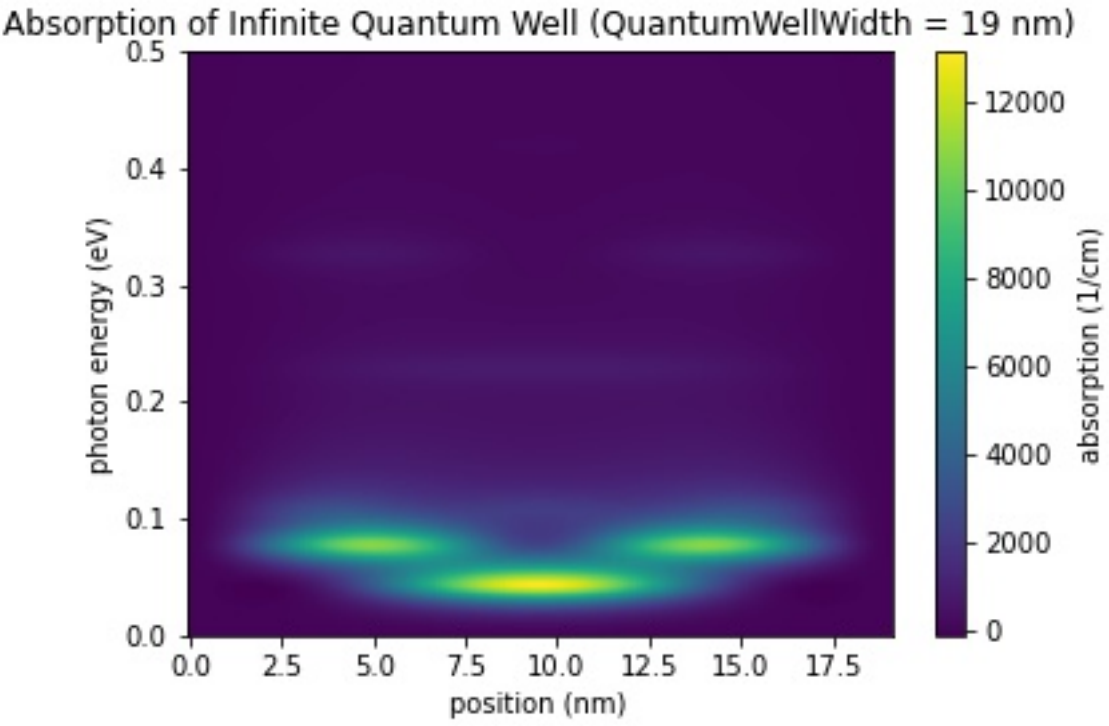


Figure 6.4.12.36: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 19 nm)

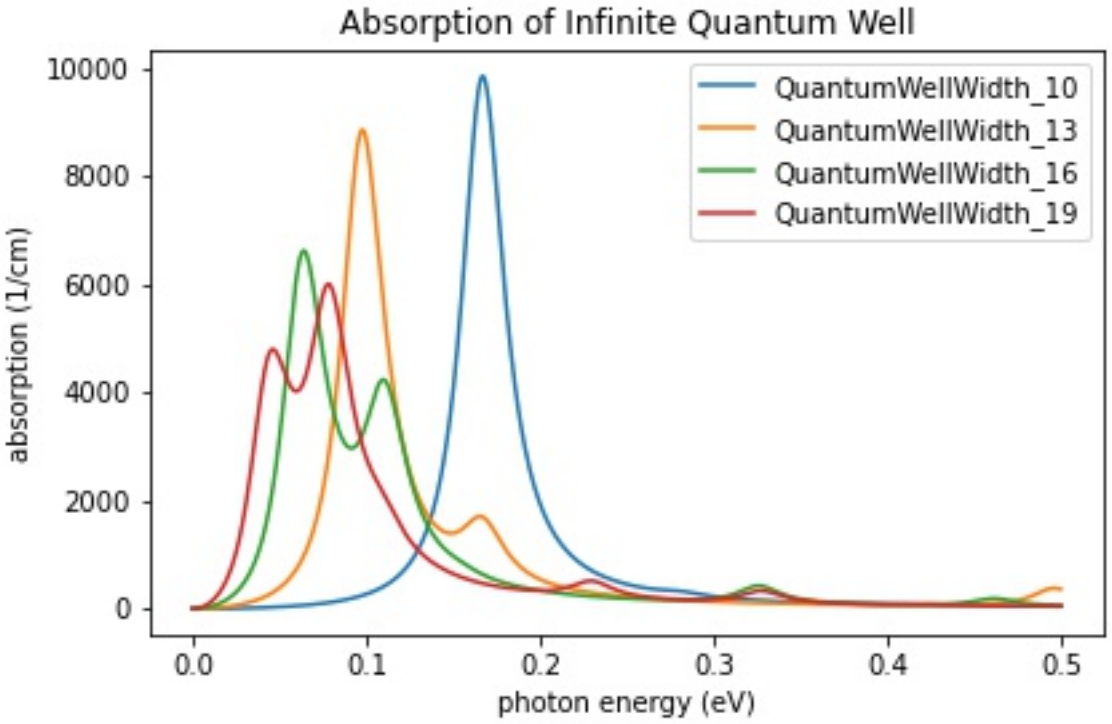


Figure 6.4.12.37: Calculated absorption spectra $\alpha(E)$ of an infinite quantum well for different well widths

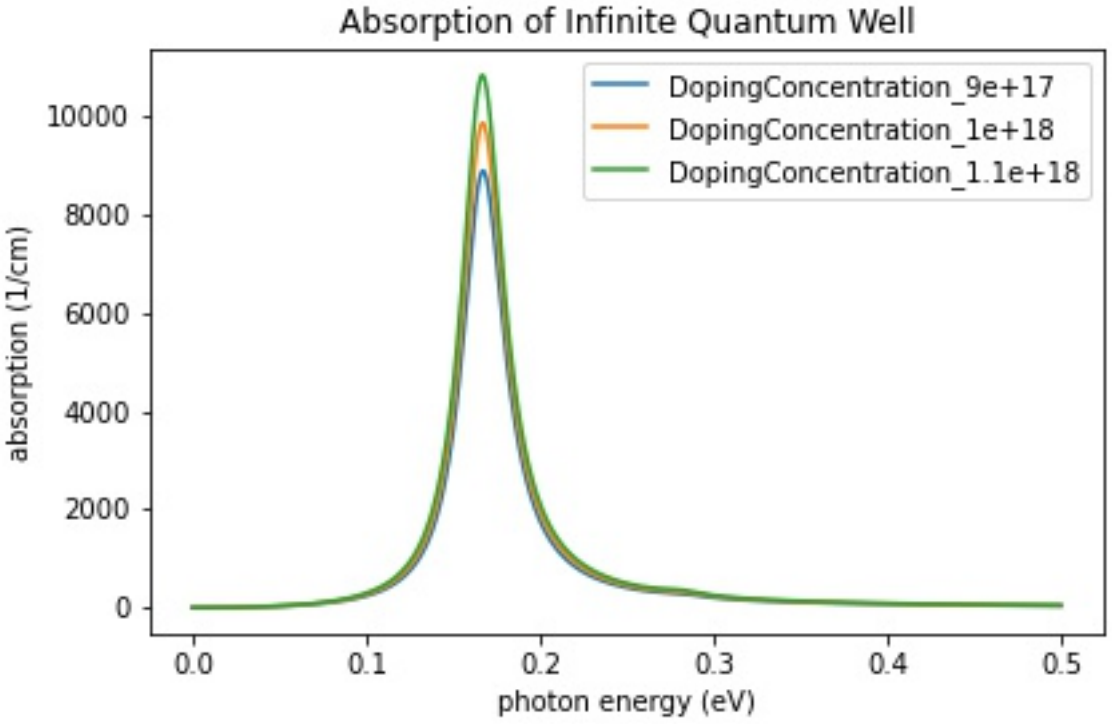


Figure 6.4.12.38: Calculated absorption spectra $\alpha(E)$ of an infinite quantum well for different doping concentrations

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.

Input files for both the *nextnano++* and *nextnano*^{3} tools are available.

The following input files were used:

- Single Quantum Well
  - `1DSirtoriPRB1994_OneWell_sg_self-consistent_nn*.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_OneWell_kp_self-consistent_nn*.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
  - `1DSirtoriPRB1994_OneWell_sg_quantum-only_nn*.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_OneWell_kp_quantum-only_nn*.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
- Two coupled Quantum Wells
  - `1DSirtoriPRB1994_TwoCoupledWells_sg_self-consistent_nn*.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_TwoCoupledWells_kp_self-consistent_nn*.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
  - `1DSirtoriPRB1994_TwoCoupledWells_sg_quantum-only_nn*.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_TwoCoupledWells_kp_quantum-only_nn*.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
- Three coupled Quantum Wells
  - `1DSirtoriPRB1994_ThreeCoupledWells_sg_self-consistent_nn*.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_ThreeCoupledWells_kp_self-consistent_nn*.in` (8-band $\mathbf{k} \cdot \mathbf{p}$)
  - `1DSirtoriPRB1994_ThreeCoupledWells_sg_quantum-only_nn*.in` (single-band effective mass approximation)
  - `1DSirtoriPRB1994_ThreeCoupledWells_kp_quantum-only_nn*.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
    }
```

For *nextnano*[3] we use:

```
# 'QUANTUM_ONLY':    solve Schrödinger and Poisson equations self-
↪consistently
# 'SELF_CONSISTENT': solve Schrödinger equation only
%QUANTUM_ONLY    = .FALSE.
%SELF_CONSISTENT = .TRUE.

$simulation-flow-control
 !IF %QUANTUM_ONLY    flow-scheme = 3   # Schrödinger only
 !IF %SELF_CONSISTENT flow-scheme = 2   # Schrödinger-Poisson

 !IF %QUANTUM_ONLY    raw-potential-in = yes
 !IF %SELF_CONSISTENT raw-potential-in = no
```

(continues on next page)

(continued from previous page)

```
$quantum-model-electrons
 ...
 model-name = effective-mass            # single-band

 number-of-eigenvalues-per-band = 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 $k_{\parallel} = 0$.

For *nextnano*[3] we use:

```
$simulation-flow-control
 !IF %QUANTUM_ONLY    flow-scheme = 3   # Schrödinger only

 !IF %QUANTUM_ONLY    raw-directory-in = raw_data/
 !IF %QUANTUM_ONLY    raw-potential-in = yes

$quantum-model-electrons
 ...
 model-name = 8x8kp                     # 8-band k.p

 number-of-eigenvalues-per-band = 6     # 6 eigenstates
```

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 6.4.12.39 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]*).

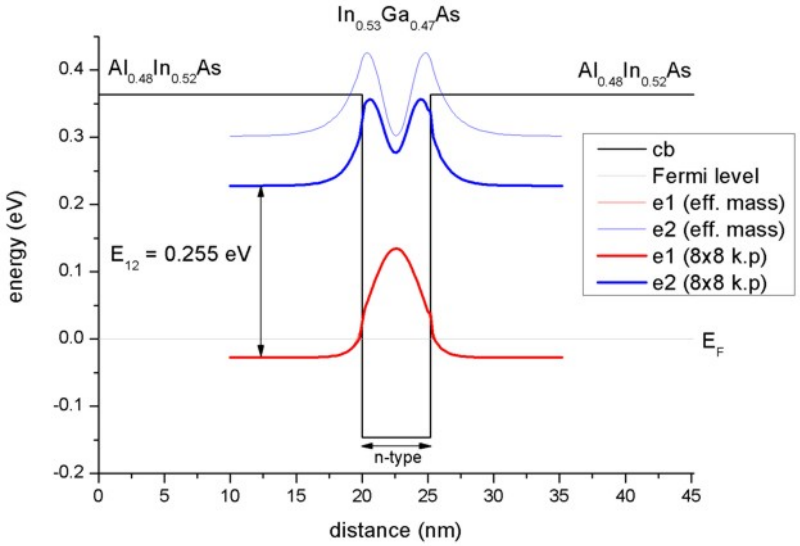


Figure 6.4.12.39: Conduction band edge, Fermi level and confined electron states of a quantum well

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 6.4.12.40 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)

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).

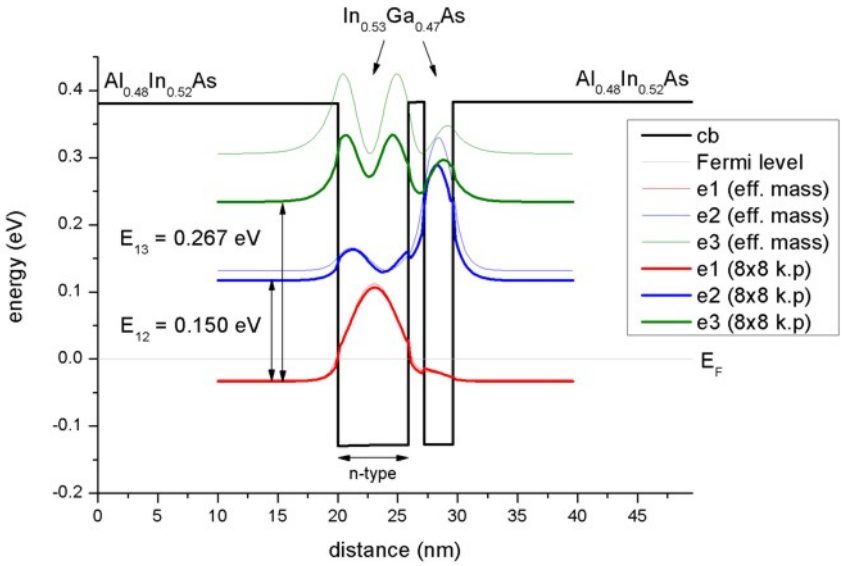


Figure 6.4.12.40: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

Three coupled quantum wells

Figure 6.4.12.41 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]*).

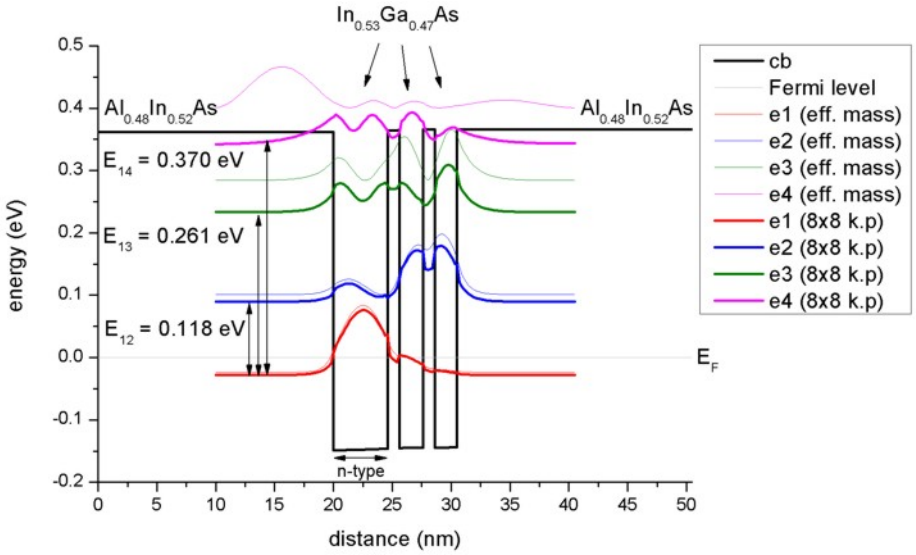


Figure 6.4.12.41: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells

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 —

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

The following documentation and figures were generated automatically using *nextnanopy*.

The following Python script was used: `intersubband_MQW_nextnano3.py`

The following figures have been generated using *nextnano*[3]. 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)

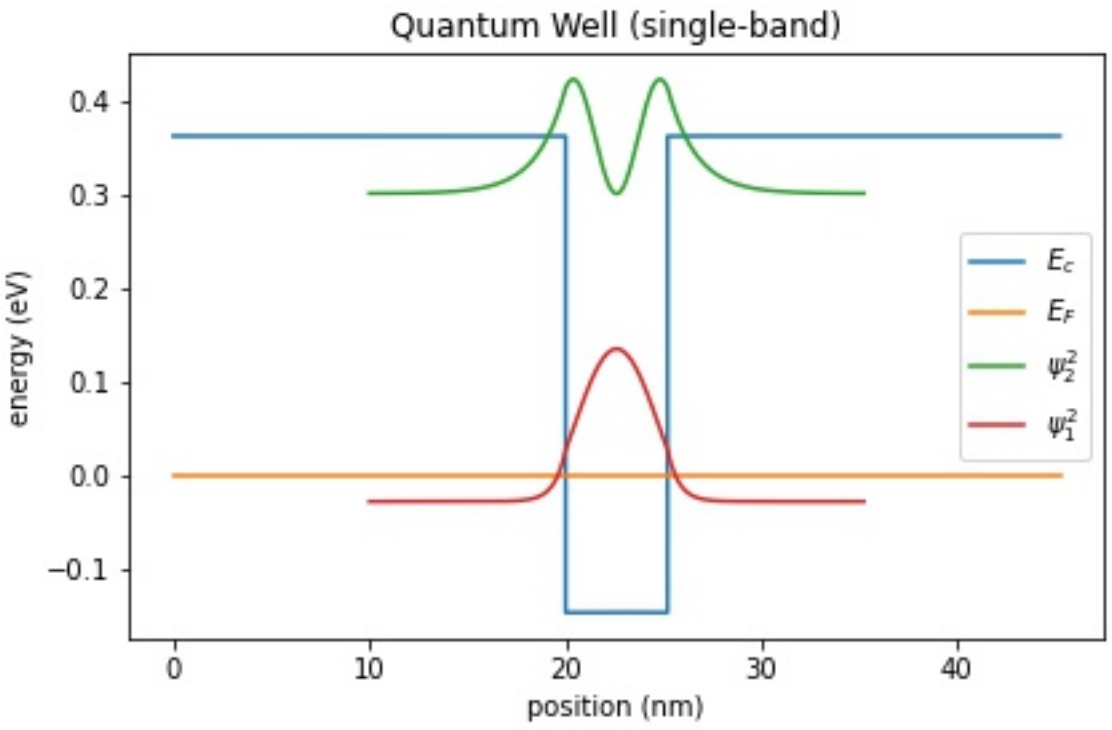


Figure 6.4.12.42: Conduction band edge, Fermi level and confined electron states of a quantum well

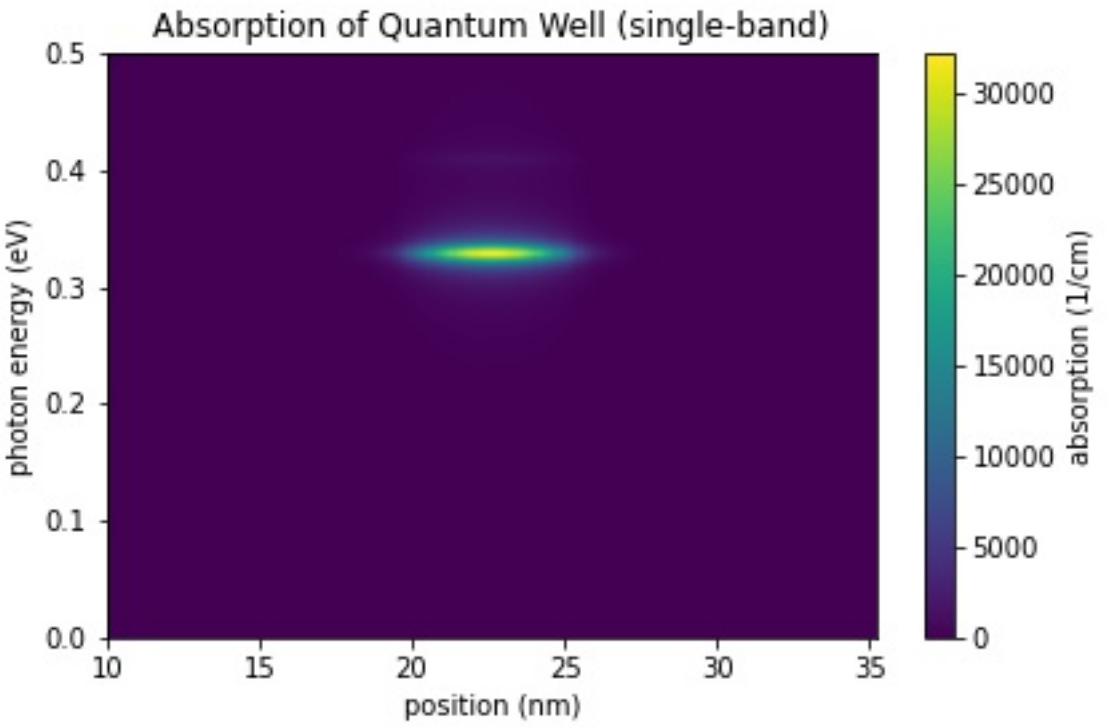


Figure 6.4.12.43: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of a quantum well

Quantum Well (k.p)

Two Coupled Quantum Wells (single-band)

Two Coupled Quantum Wells (k.p)

Three Coupled Quantum Wells (single-band)

Three Coupled Quantum Wells (k.p)

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

— End —

Last update: nn/nn/nnnn

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*

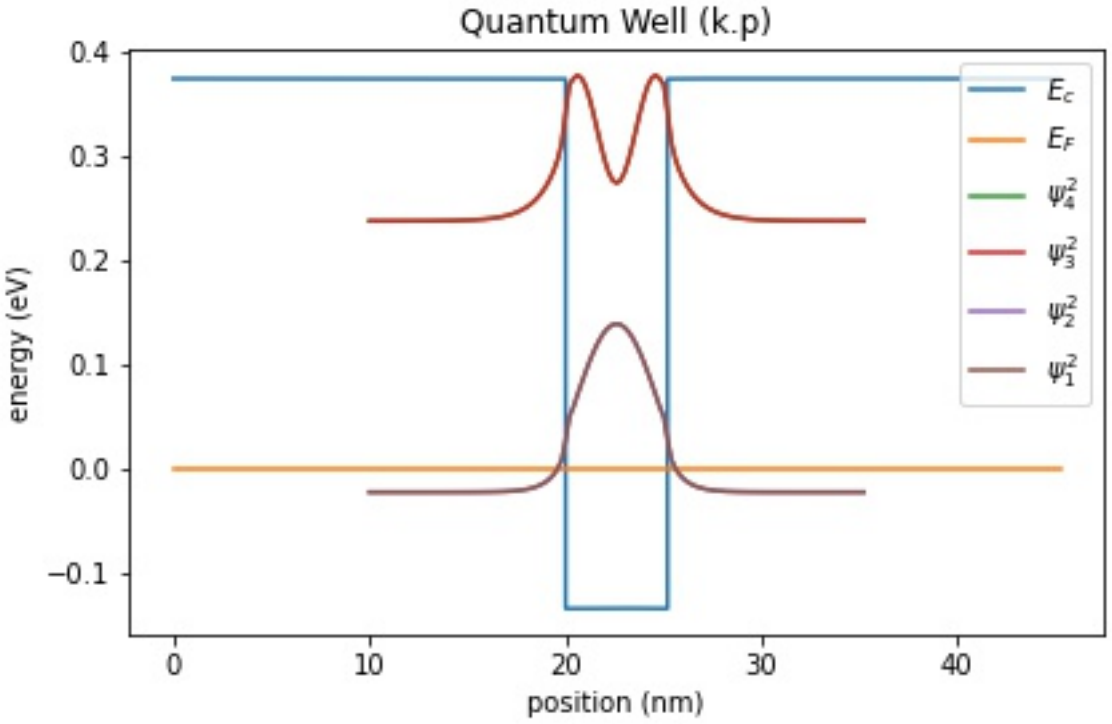


Figure 6.4.12.44: Conduction band edge, Fermi level and confined electron states of a quantum well

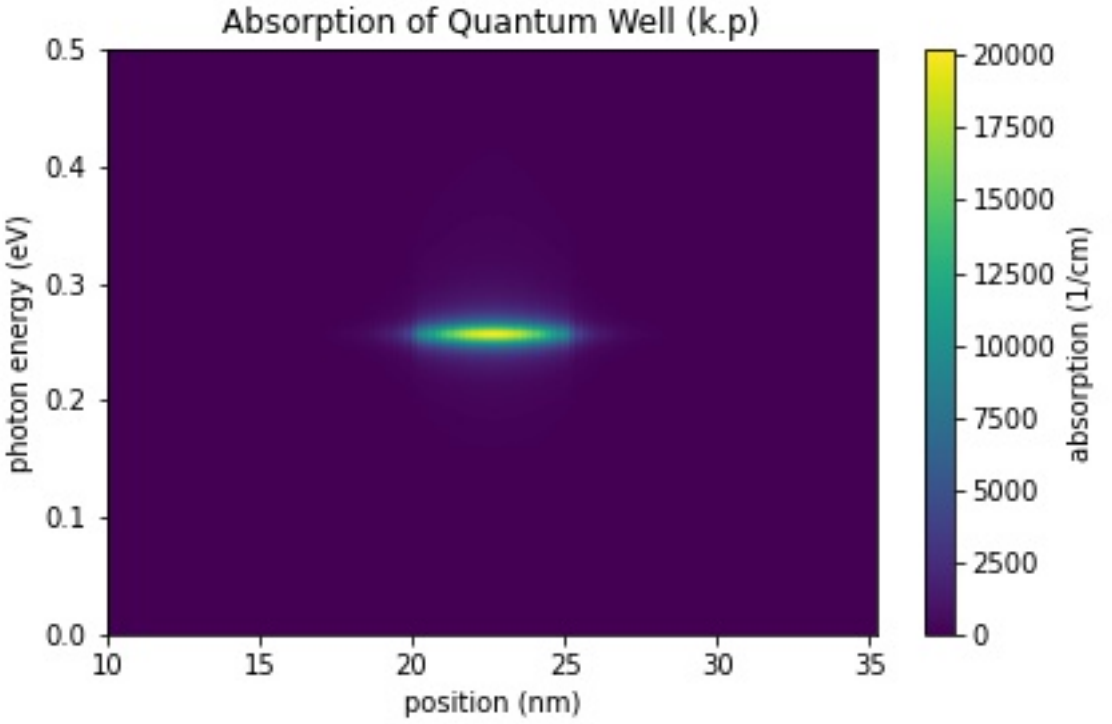


Figure 6.4.12.45: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of a quantum well

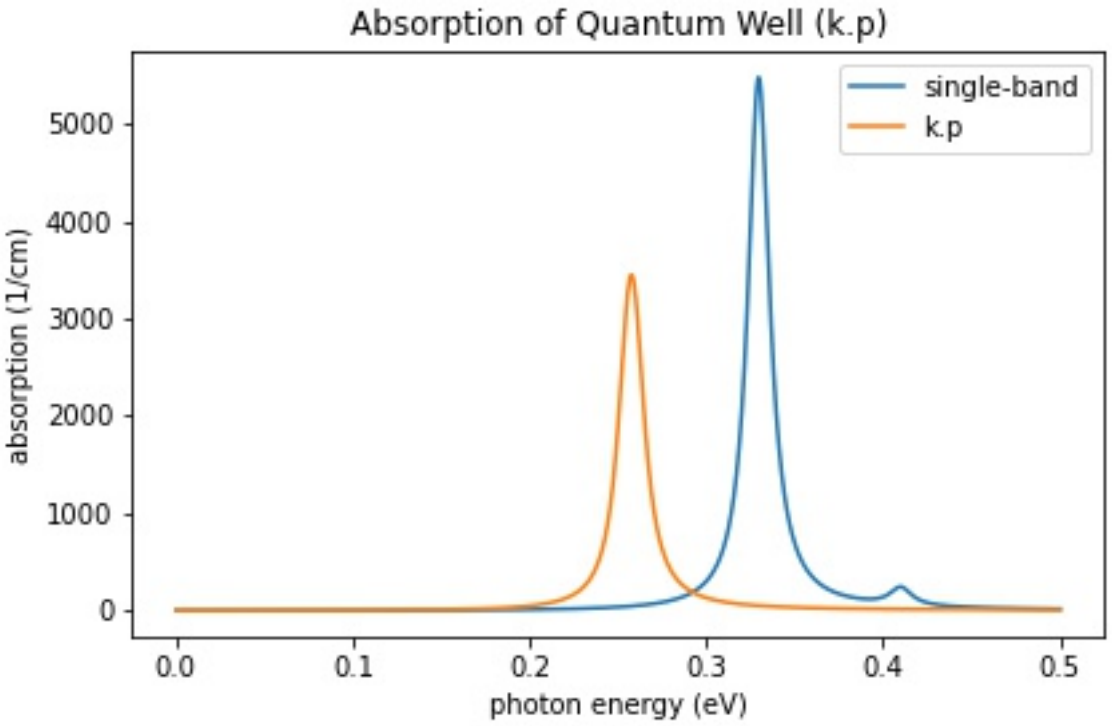


Figure 6.4.12.46: Calculated absorption spectra $\alpha(E)$ of a quantum well

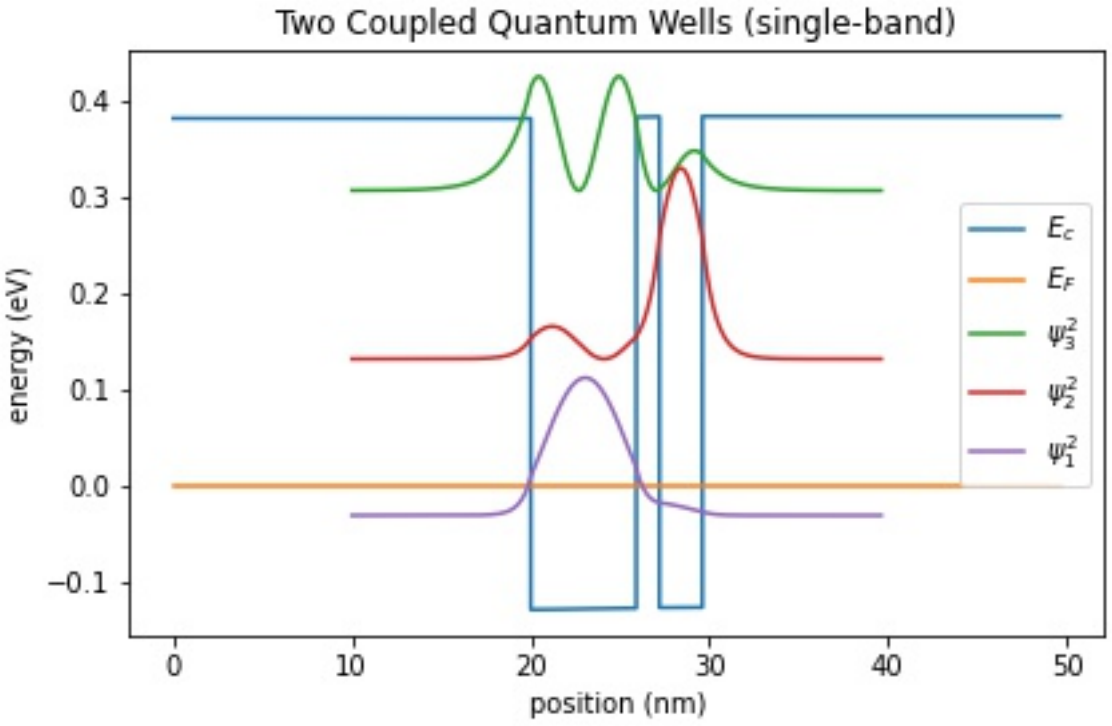


Figure 6.4.12.47: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

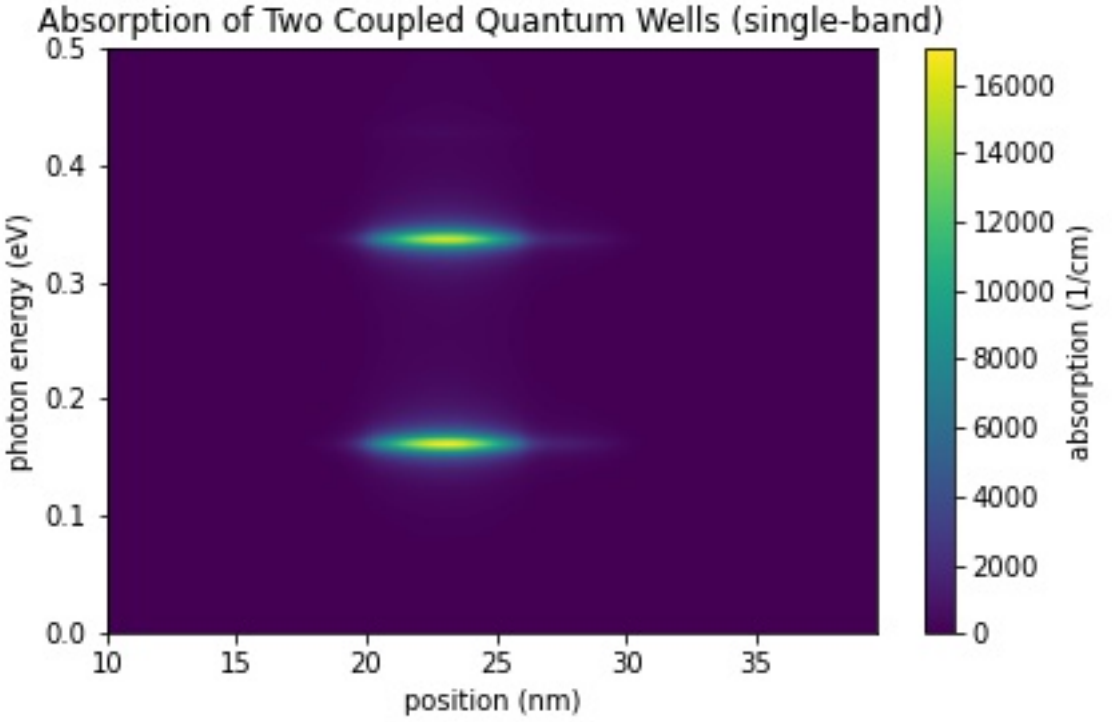


Figure 6.4.12.48: Calculated spatially resolved absorption spectrum:math:*alpha(x,E)* of two coupled quantum wells

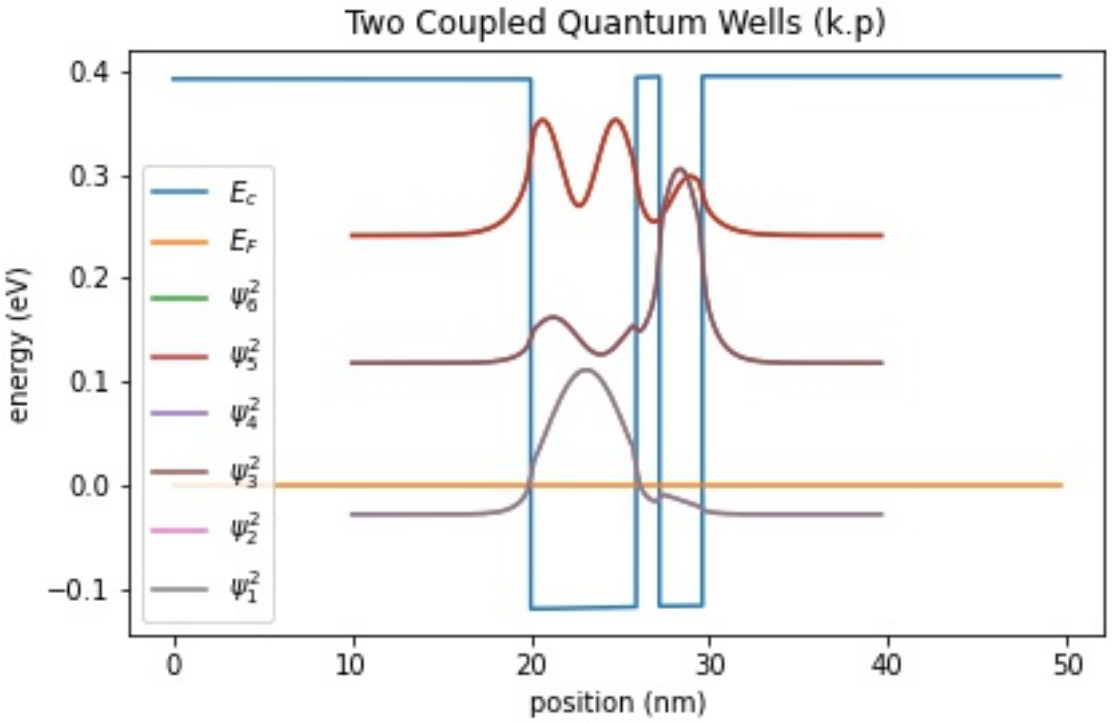


Figure 6.4.12.49: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

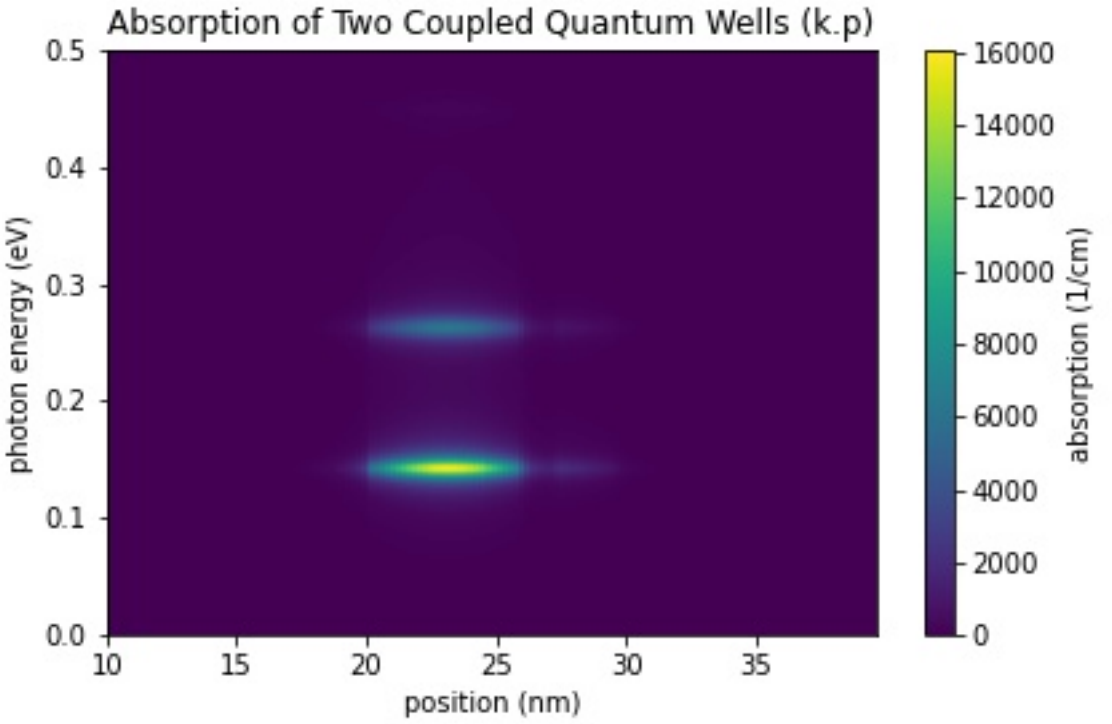


Figure 6.4.12.50: Calculated spatially resolved absorption spectrum:math:*alpha(x,E)* of two coupled quantum wells

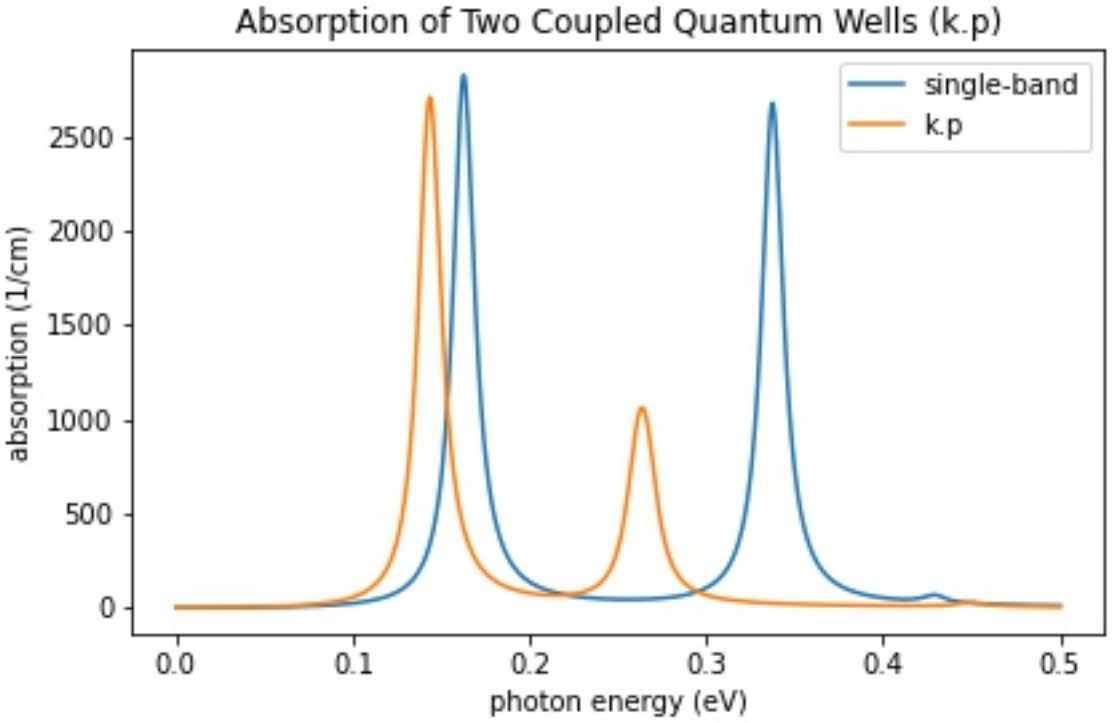


Figure 6.4.12.51: Calculated absorption spectra $\alpha(E)$ of two coupled quantum wells

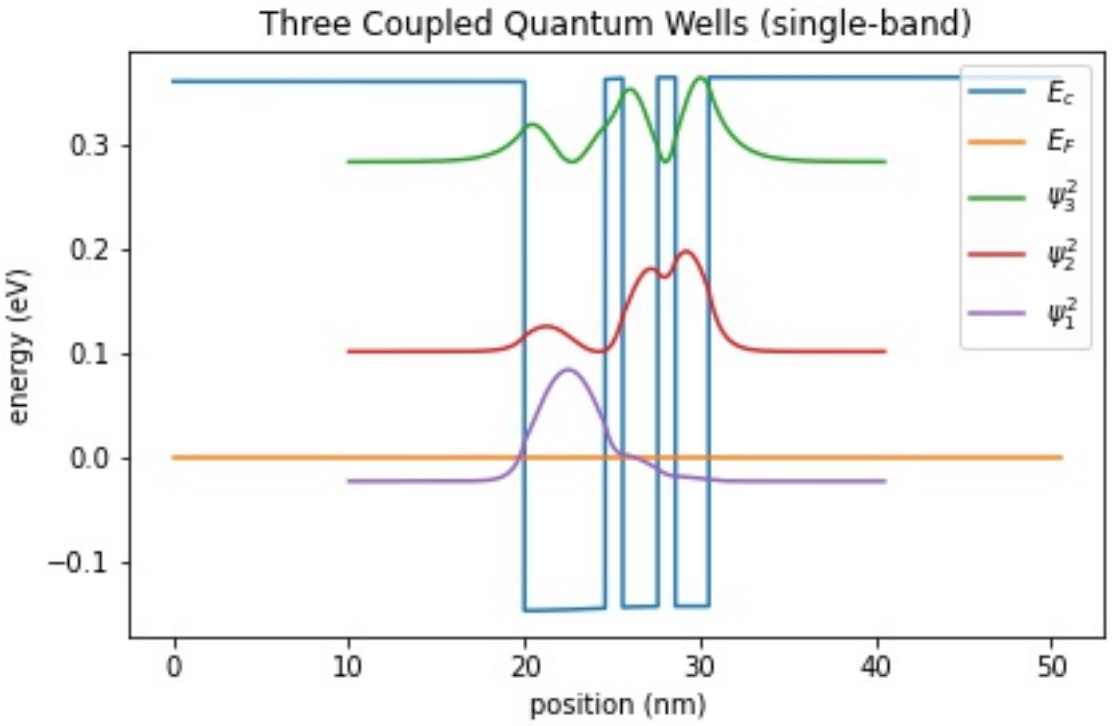


Figure 6.4.12.52: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells

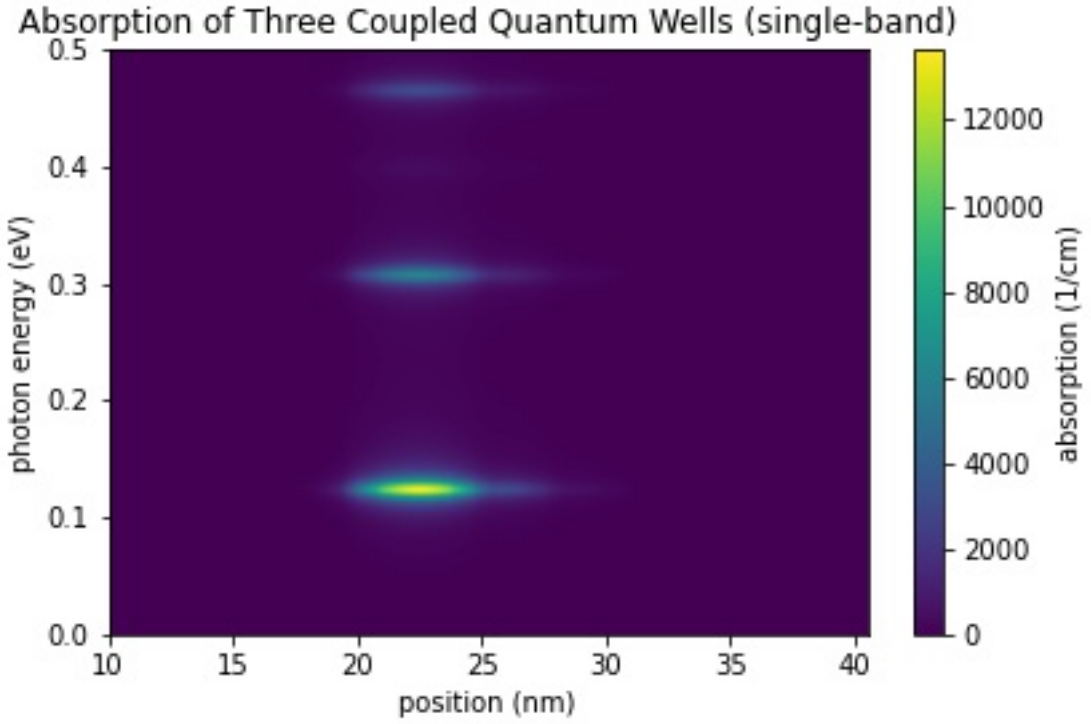


Figure 6.4.12.53: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of three coupled quantum wells

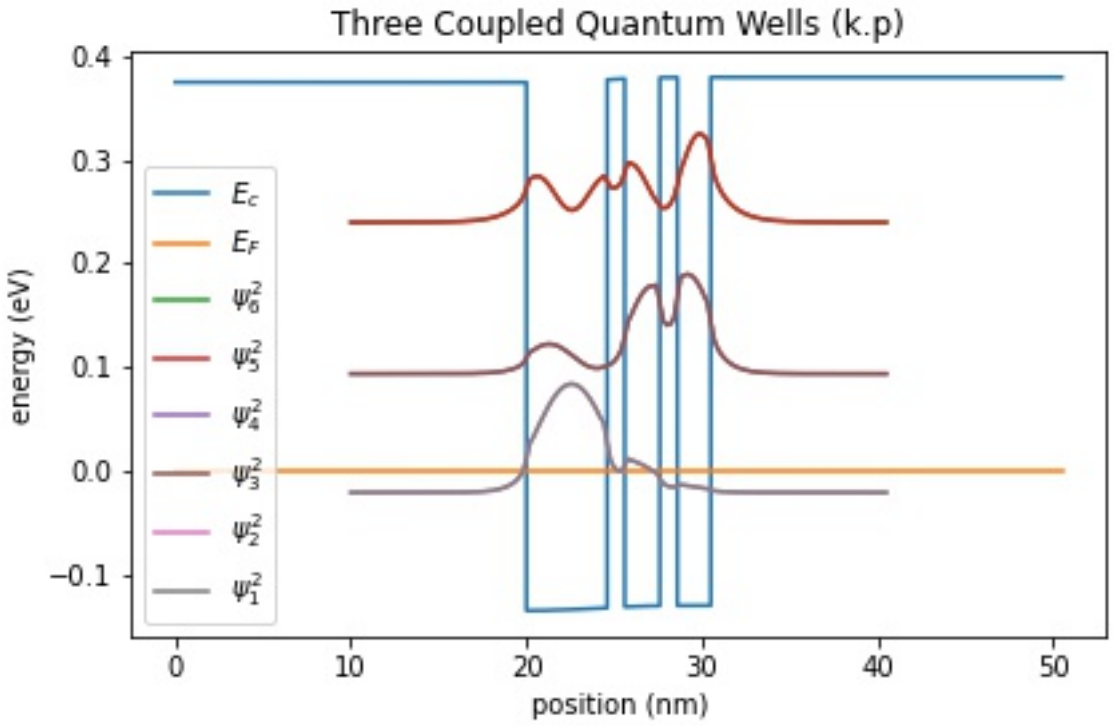


Figure 6.4.12.54: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells

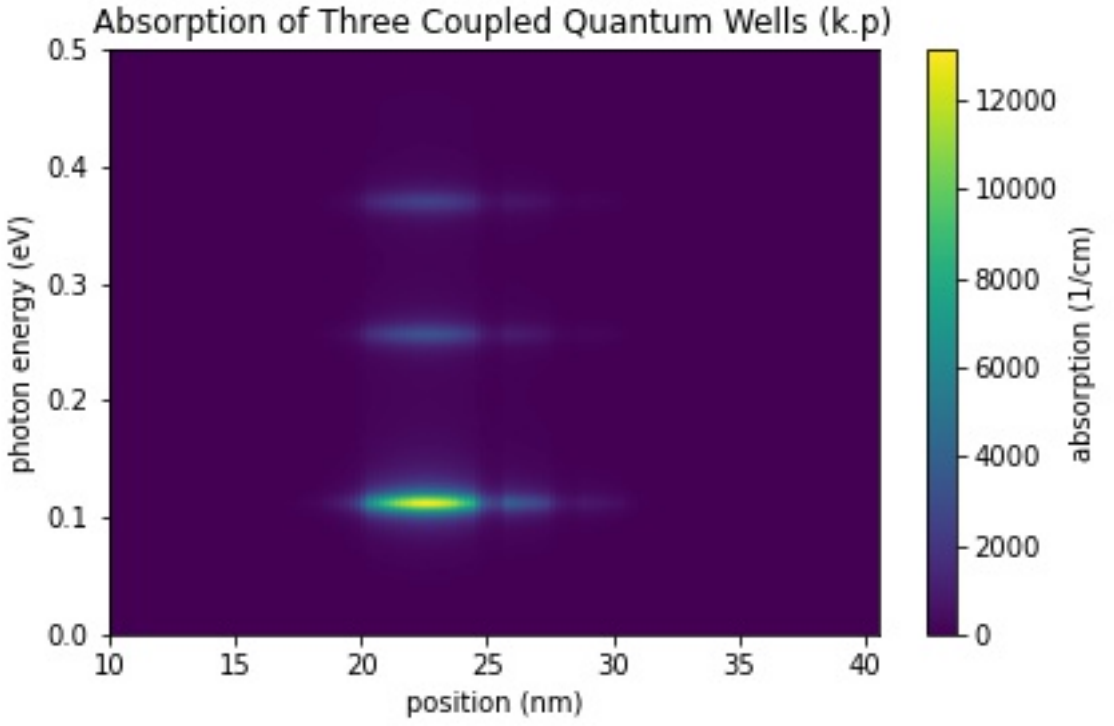


Figure 6.4.12.55: Calculated spatially resolved absorption spectrum $\alpha(x, E)$ of three coupled quantum wells

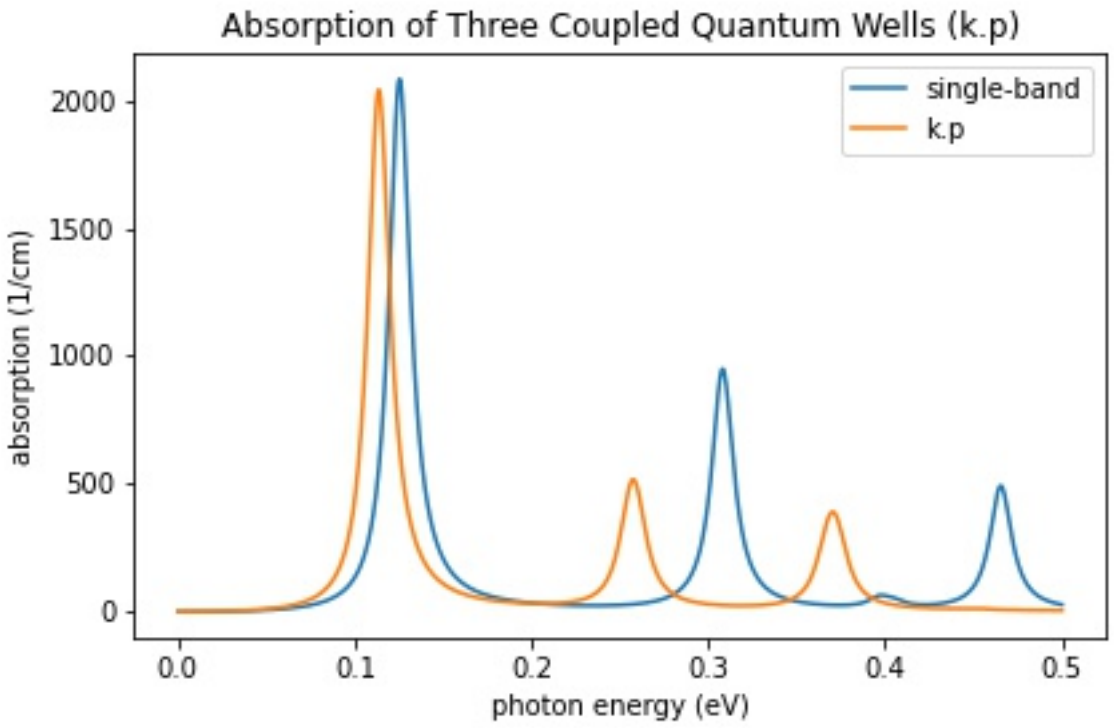


Figure 6.4.12.56: Calculated absorption spectra $\alpha(E)$ of three coupled quantum wells

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{ }
    quantum{ }
```

(continues on next page)

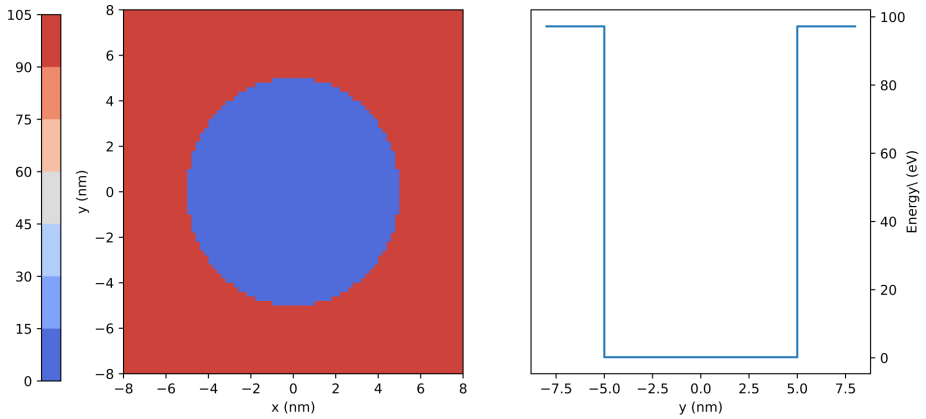


Figure 6.4.12.57: Left: Conduction band edge for cylindrical quantum wire. Right: Slice of the band edge along $x = 0$.

(continued from previous page)

```
    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 $n \rightarrow m$) in condensed matter is calculated on the basis of Fermi's golden rule [*ChuangOpto1995*] in the dimenstion of (length)$^{-1}$:

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega} \frac{1}{V} \sum_{n>m} \sum_{\mathrm{k}_z} |\vec{\epsilon} \cdot \vec{\pi}_{nm}(\mathrm{k}_z)|^2 (f_m(\mathrm{k}_z) - f_n(\mathrm{k}_z)) \mathcal{L}(E_n(\mathrm{k}_z) - E_m(\mathrm{k}_z) - \hbar\omega), \quad (6.4.12.5)$$

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 } }*,

    $$\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ \left(- \frac{(E_n - E_m - \hbar\omega)^2}{2\sigma^2} \right) \right\}$$

    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

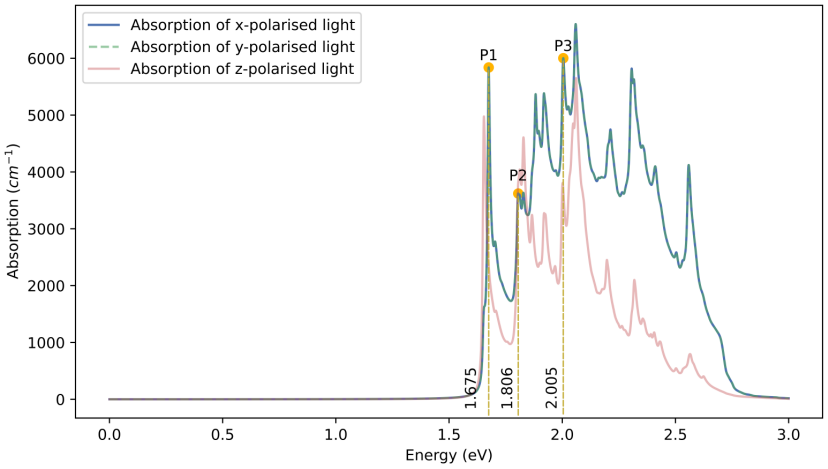


Figure 6.4.12.58: Calculated absorption spectrum $\alpha(\vec{\epsilon}, E)$ for $\vec{\epsilon} = \hat{x}, \hat{y}, \hat{z}$.

Figure 6.4.12.58 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

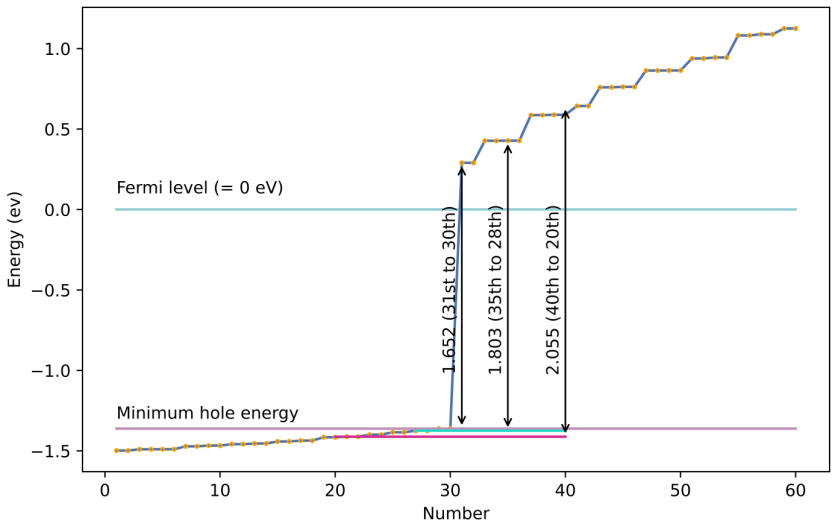


Figure 6.4.12.59: Calculated energy spectrum and the minimum hole energy.

Figure 6.4.12.59 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 6.4.12.59 with the purple line. It can be seen through a comparison with Figure 6.4.12.58 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 6.4.12.59 due to spin are put together as one eigenstate in *Optics*. Thus *\Optics\occupation_disp_~_kp8_16.dat* shows the 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 6.4.12.59.

At $T = 300\text{K}$, $k_B T \simeq 0.026$ eV , which is insufficient energy to excite electron carriers to the upper conduction band states.

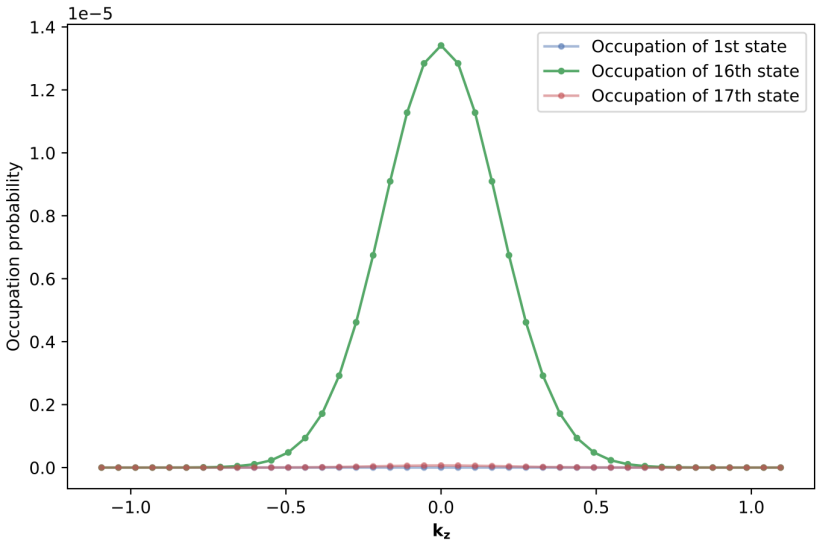


Figure 6.4.12.60: Calculated occupation probabilities for the ground state and 16th excited state as a function of k_z.

From the above data of eigenvalues and occupations, we could see which pair of states contributes to each peak in the absorption spectrum Figure 6.4.12.58. In order to understand the magnitude of the peaks and why some pairs of states don't appear as peaks, we will see the output data for $|\vec{\epsilon} \cdot \vec{\pi}_{nm}(k_z)|^2$ next.

Transition intensity (Momentum matrix element)

An important part of the calculation of optical absorption spectra is the transition intensity:

$$T_{nm}(\vec{\epsilon}, k_z) = \frac{2}{m_0} |\vec{\epsilon} \cdot \vec{\pi}_{nm}(k_z)|^2 \tag{6.4.12.6}$$

which has dimensions of energy [eV].

The intensity at $k_z = 0$ $\left(T_{nm}(\vec{\epsilon}, k_z = 0)\right)$ 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 6.4.12.59.

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. (6.4.7.1) 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.

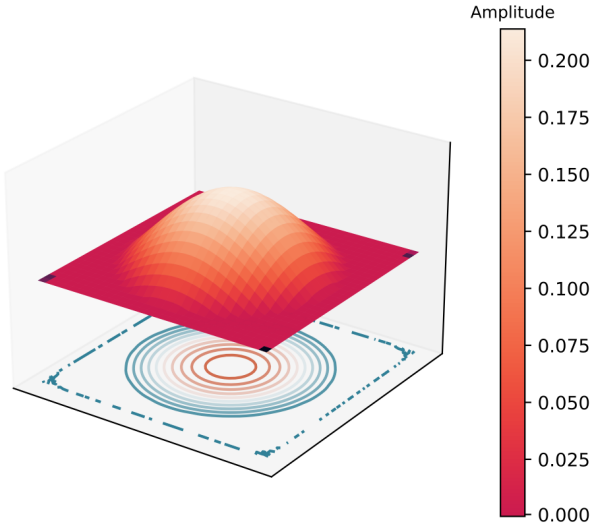


Figure 6.4.12.61: Wave function of the ground state. $(n, l) = (1, 0)$

Wave functions of the energy eigenstates calculated by the single-band model.

Last update: nn/nn/nnnn

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*

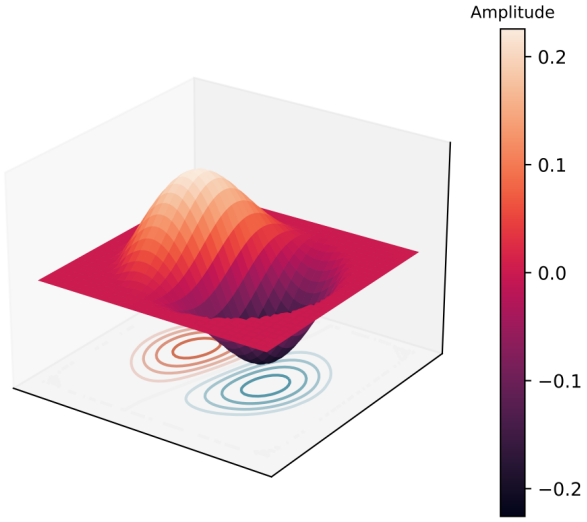


Figure 6.4.12.62: Wave function of the first excited state. $(n, l) = (1, \pm 1)$

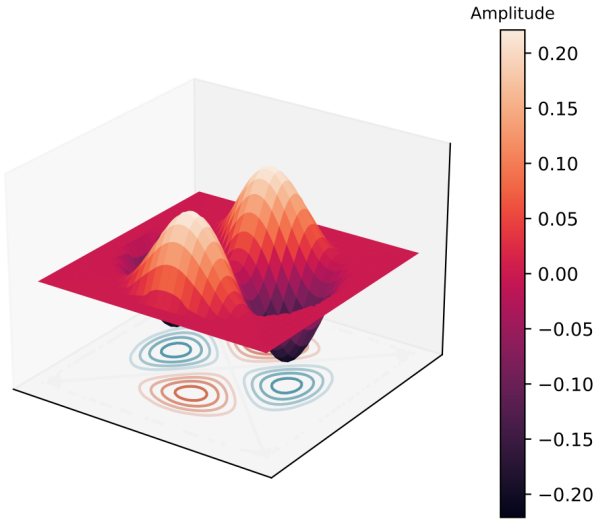


Figure 6.4.12.63: Wave function of the second excited state. $(n, l) = (1, \pm 2)$

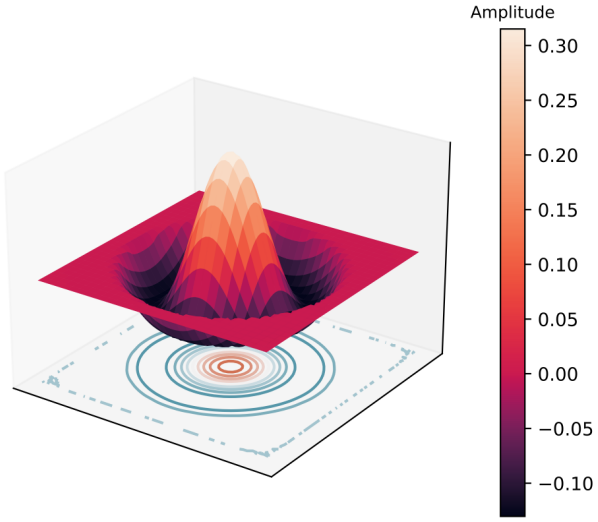


Figure 6.4.12.64: Wave function of the third excited state. $(n, l) = (2, 0)$

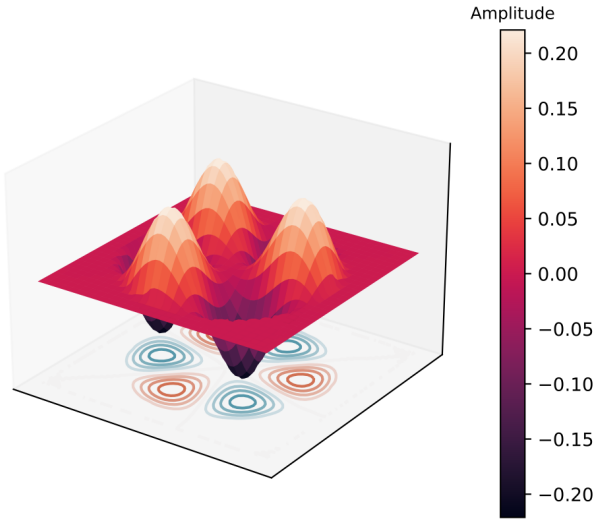


Figure 6.4.12.65: Wave function of the fourth excited state. $(n, l) = (1, \pm 3)$

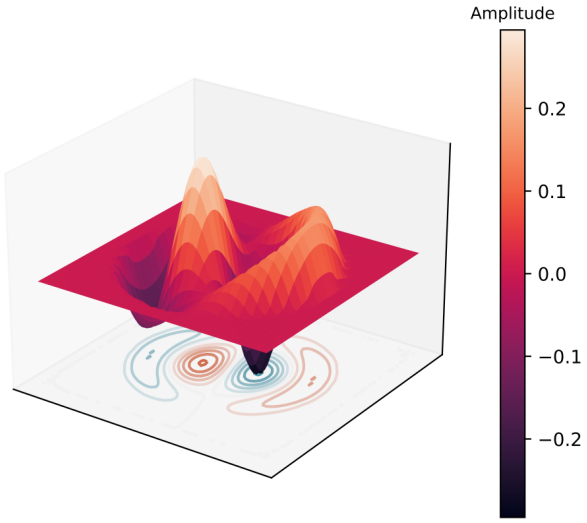


Figure 6.4.12.66: Wave function of the fifth excited state. $(n, l) = (2, \pm 1)$

- *Eigenvalues, transition energies, and occupations*
- *Transition intensity (Momentum matrix element)*
- *Eigenstates*

Input file:

- *2Dcircular_infinite_wire_GaAs_intra_nnp.in*

Structure

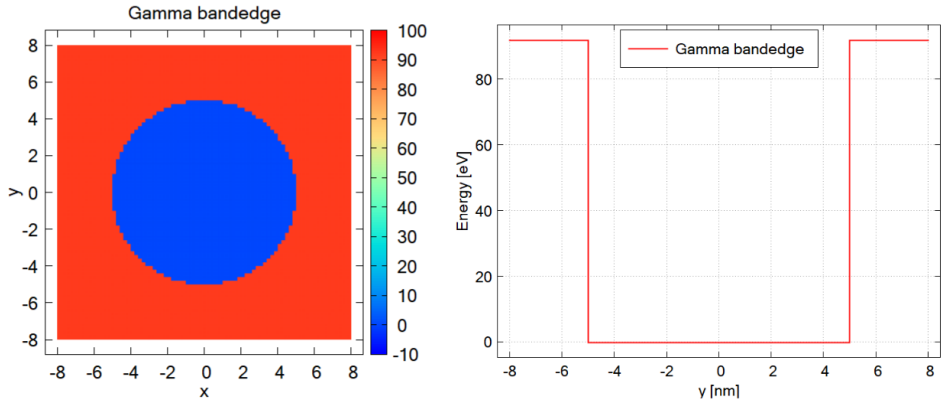


Figure 6.4.12.67: Left: Conduction band edge for a cylindrical quantum wire. Right: Slice of the band edge along $x = 0$.

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]

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 $n \rightarrow m$) in a condensed matter is calculated on the basis of Fermi's golden rule [*ChuangOpto1995*] in the dimenstion of (length)$^{-1}$:

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega} \frac{1}{V} \sum_{n>m} \sum_{\mathrm{k}_z} |\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathrm{k}_z)|^2 (f_m(\mathrm{k}_z) - f_n(\mathrm{k}_z)) \mathcal{L}(E_n(\mathrm{k}_z) - E_m(\mathrm{k}_z) - \hbar\omega), \quad (6.4.12.7)$$

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 } }*,

    $$\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ \left(- \frac{(E_n - E_m - \hbar\omega)^2}{2\sigma^2} \right) \right\}$$

    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

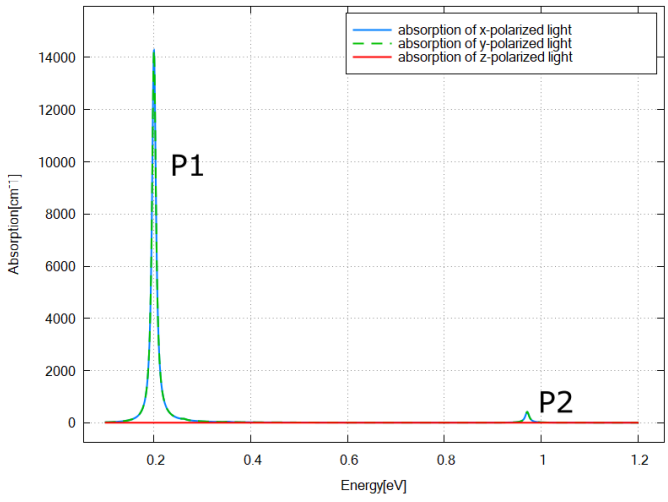


Figure 6.4.12.68: Calculated absorption spectrum $\alpha(\vec{\epsilon}, E)$ for $\vec{\epsilon} = \hat{x}, \hat{y}, \hat{z}$.

Figure 6.4.12.68 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.

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 6.4.12.68 is reasonable since the single-band model is emulated in this tutorial.

Eigenvalues, transition energies, and occupations

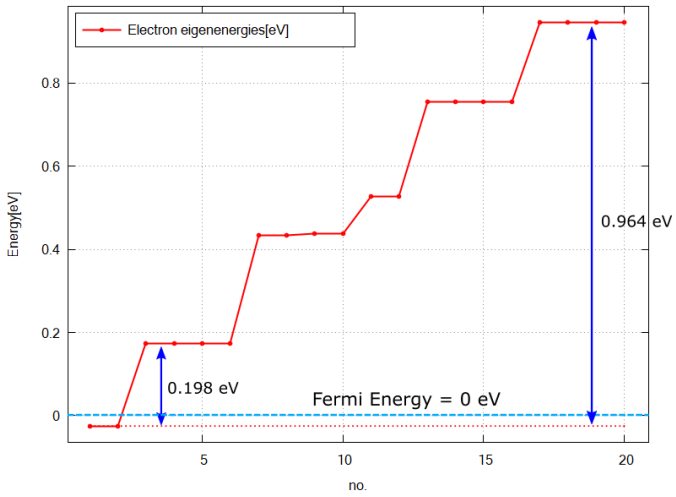


Figure 6.4.12.69: Calculated energy spectrum and Fermi energy (=0 eV).

Figure 6.4.12.69 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 6.4.12.68 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:

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 6.4.12.69 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.

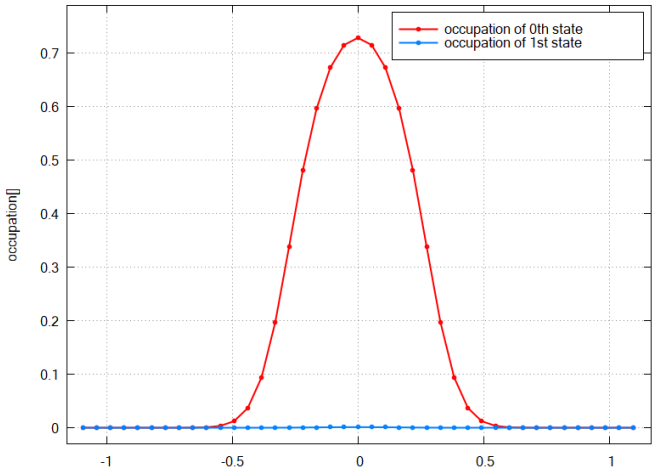


Figure 6.4.12.70: Calculated occupation probabilities for the ground state and 1st excited state as a function of k_z.

From the above data of eigenvalues and occupations, we could see which pair of states contributes to each peak in the absorption spectrum Figure 6.4.12.68. In order to understand the magnitude of the peaks and why some pairs of states don't appear as peaks, we will see the output data for $|\vec{\epsilon}\cdot\vec{\pi}_{nm}(k_z)|^2$ next.

Transition intensity (Momentum matrix element)

One of the key element for the calculation of absorption spectra is the transition intensity

$$T_{nm}(\vec{\epsilon},k_z)=\frac{2}{m_0}|\vec{\epsilon}\cdot\vec{\pi}_{nm}(k_z)|^2 \tag{6.4.12.8}$$

which has the dimension of energy [eV].

The intensity at $k_z=0$ $\left(T_{nm}(\vec{\epsilon},k_z=0)\right)$ 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 don't 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. (6.4.7.1) 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 $l = \pm 1$.

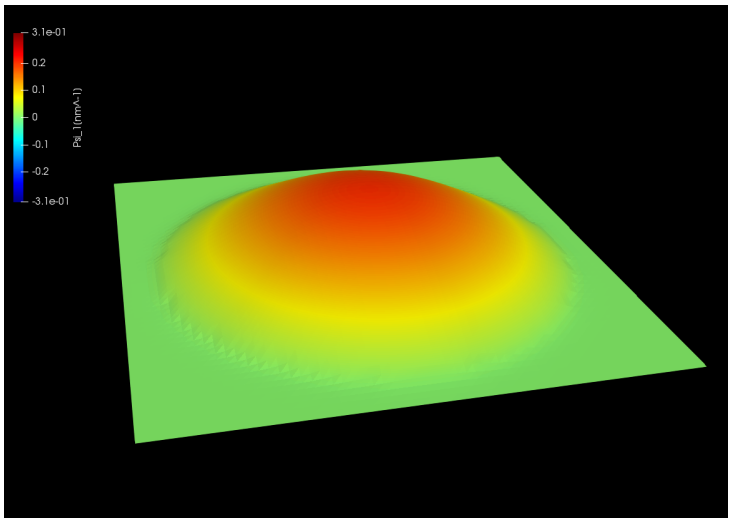


Figure 6.4.12.71: Wave function of the ground state. $(n, l) = (1, 0)$

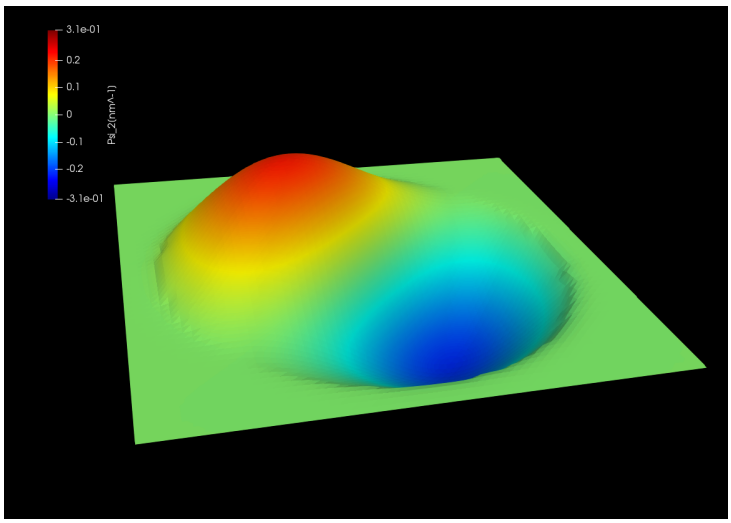


Figure 6.4.12.72: Wave function of the 1st excited state. $(n, l) = (1, \pm 1)$

Wave funstions of the energy eigenstates calculated by the single-band model.

Last update: nn/nn/nnnn

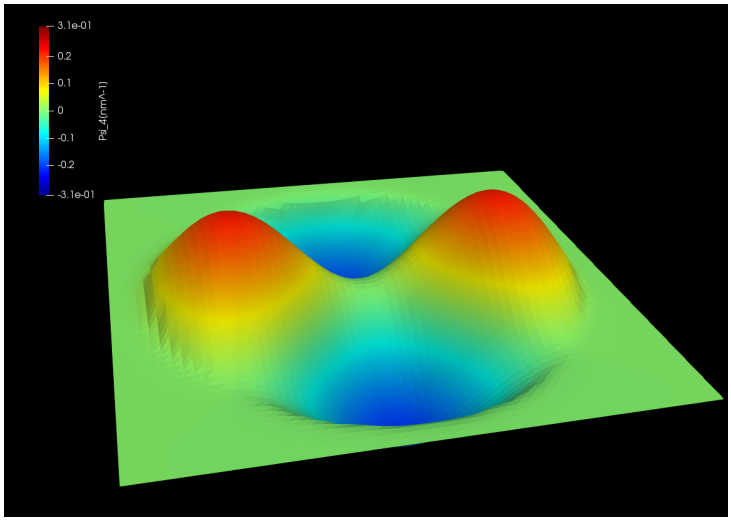


Figure 6.4.12.73: Wave function of the 2nd excited state. $(n, l) = (1, \pm 2)$

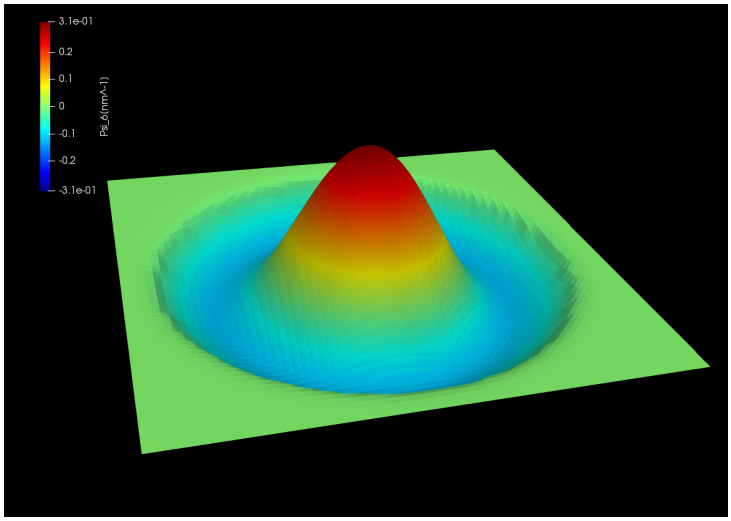


Figure 6.4.12.74: Wave function of the 3rd excited state. $(n, l) = (2, 0)$

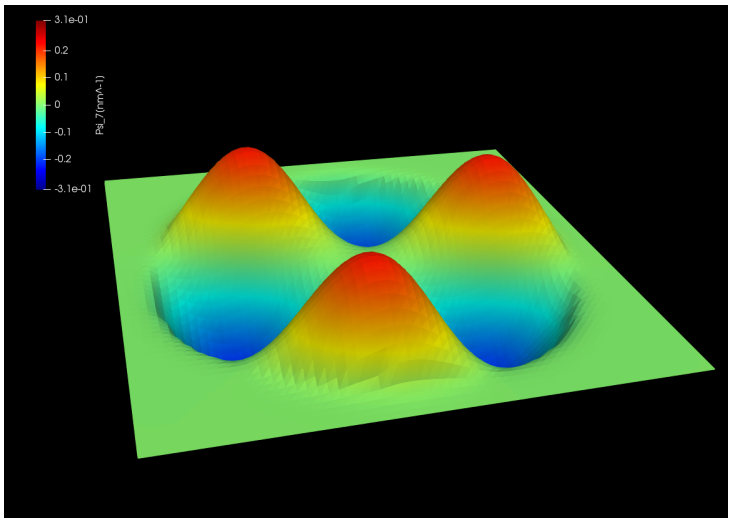


Figure 6.4.12.75: Wave function of the 4th excited state. $(n, l) = (1, \pm 3)$

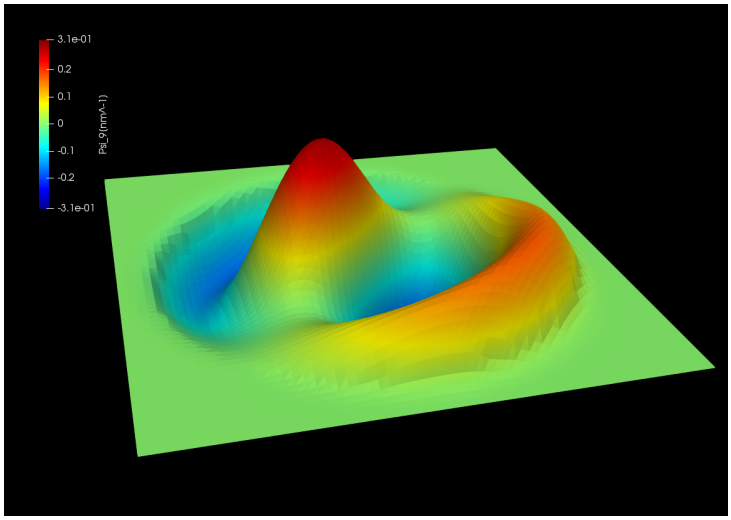


Figure 6.4.12.76: Wave function of the 5th excited state. $(n, l) = (2, \pm 1)$

Absorption of a GaAs spherical quantum dot

Section author: Naoki Mitsui

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

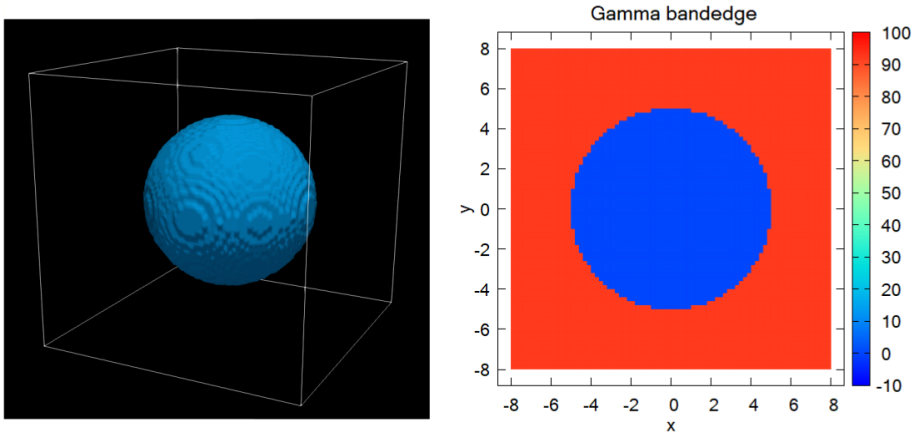


Figure 6.4.12.77: Left: GaAs region as a spherical quantum dot. Right: Slice of the Gamma band edge along $z = 0$.

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{ }
    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 $n \rightarrow m$) in a condensed matter is calculated on the basis of Fermi's golden rule [*ChuangOpto1995*] in the dimenstion of (length)$^{-1}$:

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{n_s c \varepsilon_0 m_0^2 \omega} \frac{1}{V} \sum_{n>m} |\vec{\epsilon}\cdot\vec{\pi}_{nm}|^2 (f_m - f_n)\mathcal{L}(E_n - E_m - \hbar\omega), \tag{6.4.12.9}$$

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 } }*,

    $\mathcal{L}(E_n - E_m - \hbar\omega) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{\left(-\frac{(E_n - E_m - \hbar\omega)^2}{2\sigma^2}\right)\right\}$

    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 don't have the k-summation like 1D and 2D cases.

Results

Absorption

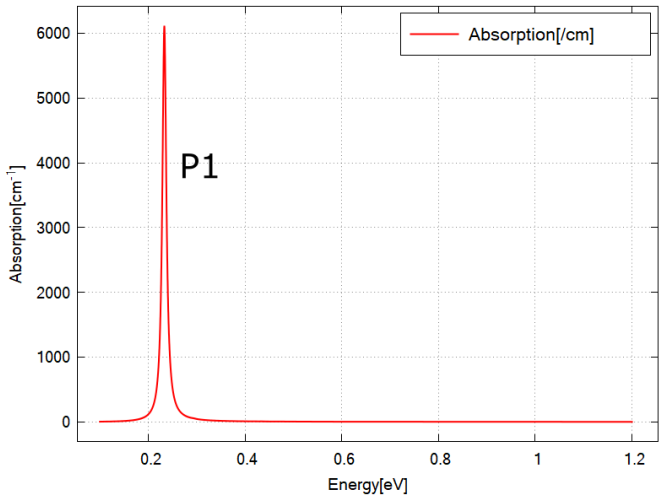


Figure 6.4.12.78: Calculated absorption spectrum $\alpha(\vec{\epsilon}, E)$ for $\vec{\epsilon} = \hat{x}$.

Figure 6.4.12.78 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 6.4.12.79 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 6.4.12.78 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 6.4.12.79 due to spin are put together as one eigenstate in *Optics*.

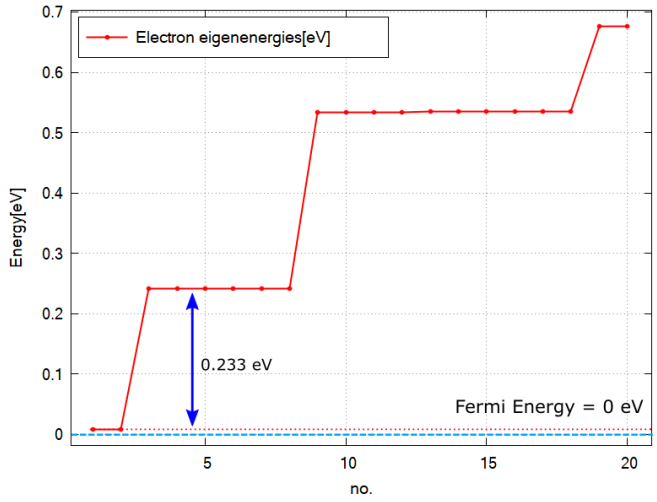


Figure 6.4.12.79: Calculated energy spectrum and Fermi energy (=0 eV).

From the above data of eigenvalues, we could see which pair of states contributes to the peak in the absorption spectrum Figure 6.4.12.78. In order to understand why some pairs of states don't 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

$$T_{nm}(\vec{\epsilon}) = \frac{2}{m_0} |\vec{\epsilon} \cdot \vec{\pi}_{nm}|^2 \tag{6.4.12.10}$$

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 don't 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.

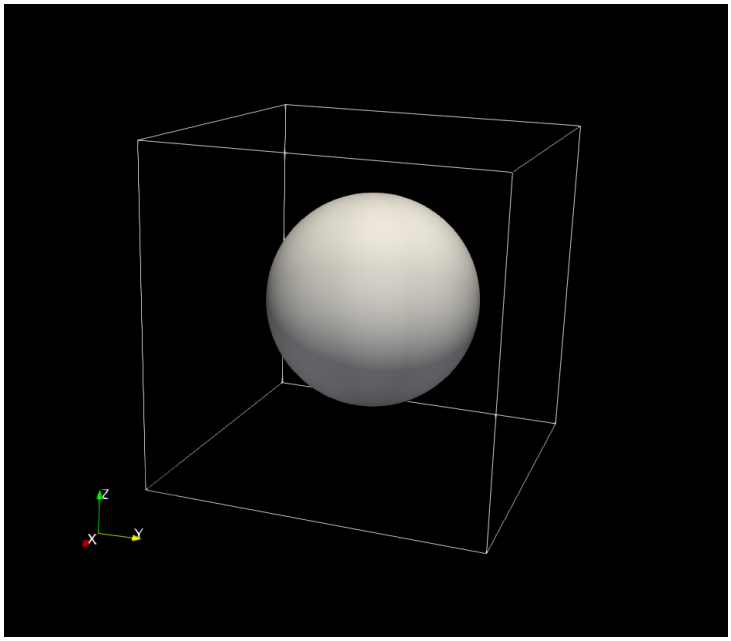


Figure 6.4.12.80: |wave function|^2 of the ground state. (s orbital, not degenerated.)

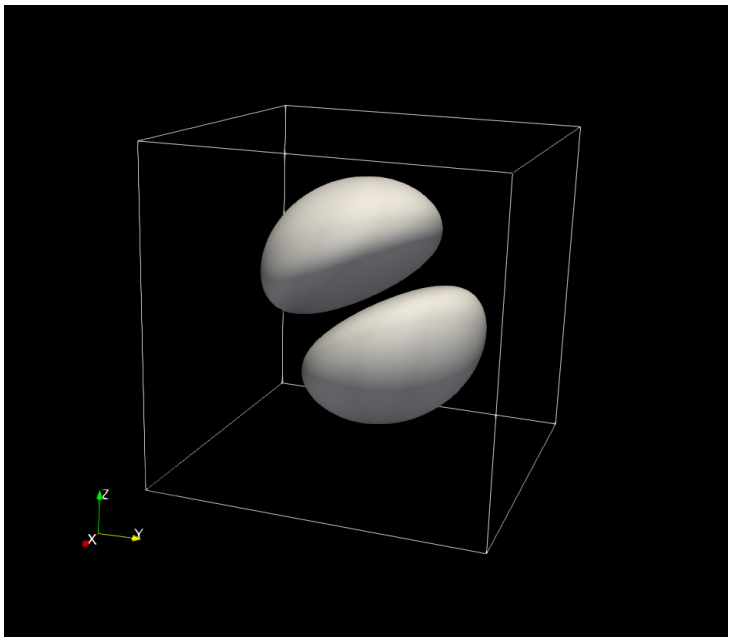


Figure 6.4.12.81: |wave function|^2 of the 1st excited state. (3 times degenerated, p orbital)

|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: nn/nn/nnnn

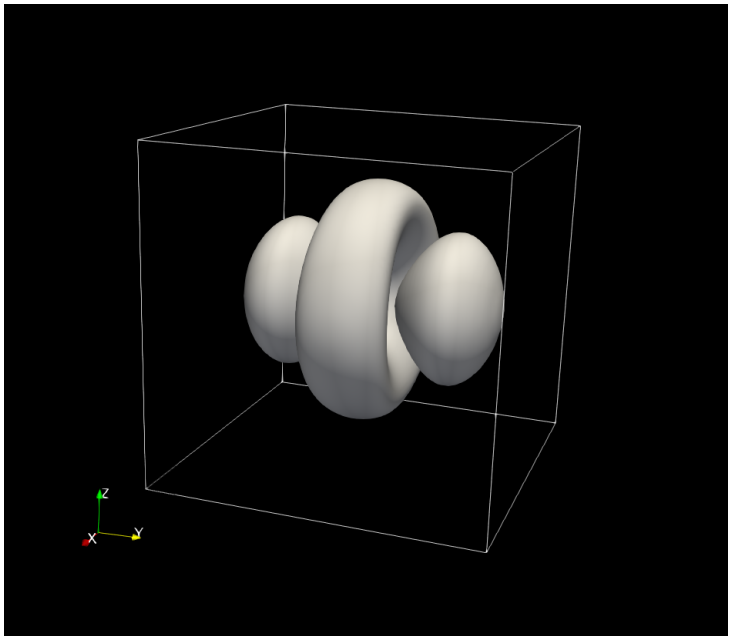


Figure 6.4.12.82: |wave function|^2 of the 2nd excited state. (5 times degenerated, d orbital)

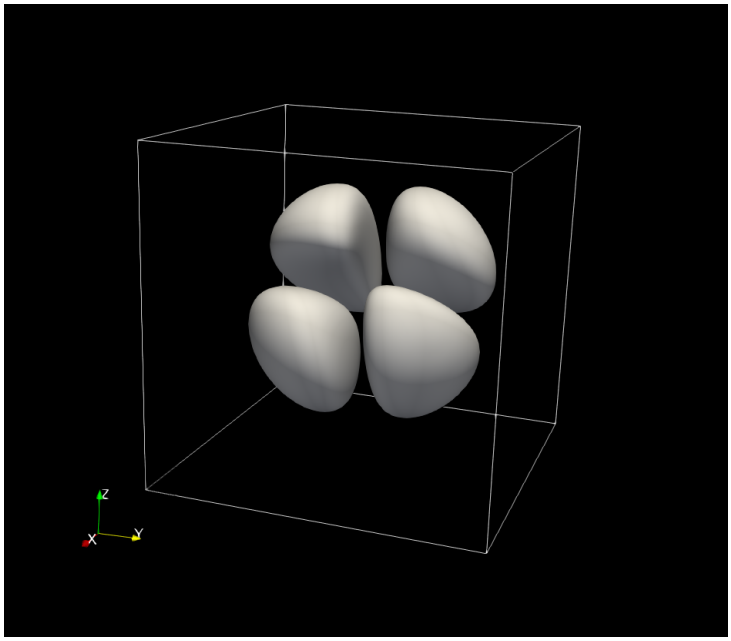


Figure 6.4.12.83: |wave function|^2 of the 2nd excited state. (d orbital)

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.

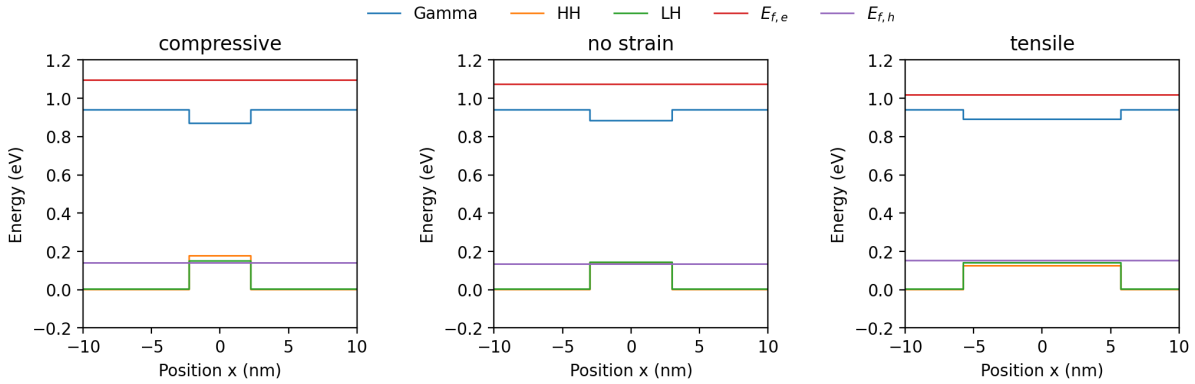


Figure 6.4.12.84: The band edges and Fermi levels of compressively strained QW (L_w = 4.5 nm, x = 0.41) (**left**), unstrained QW (L_w = 6.0 nm, x = 0.47) (**center**) and tensely strained QW (L_w = 11.5 nm, x = 0.50) (**right**). The band profile is shifted so that the valence band edge of the barrier is at 0 eV.

The band profiles for all three cases are depicted in Figure 6.4.12.84. 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.

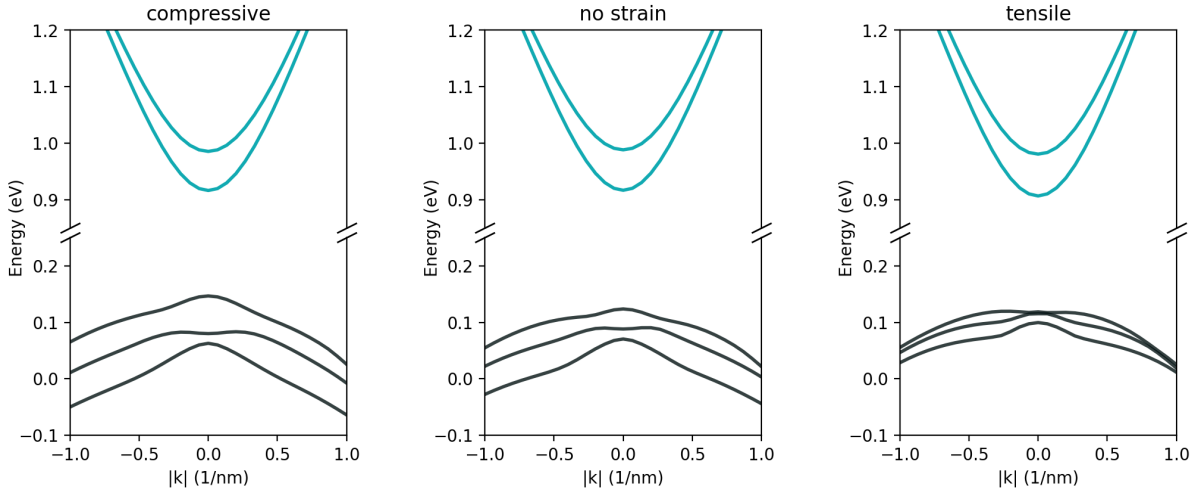


Figure 6.4.12.85: Calculated subband dispersions for the comressively strained QW (**left**), unstrained QW (**center**) and tensely strained QW (**right**). The ground and 1st excited states for electron (cyan), as well as the three highest hole states (black) are shown.

Energy dispersions for all three cases are shown in Figure 6.4.12.85. 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 6.4.12.86 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

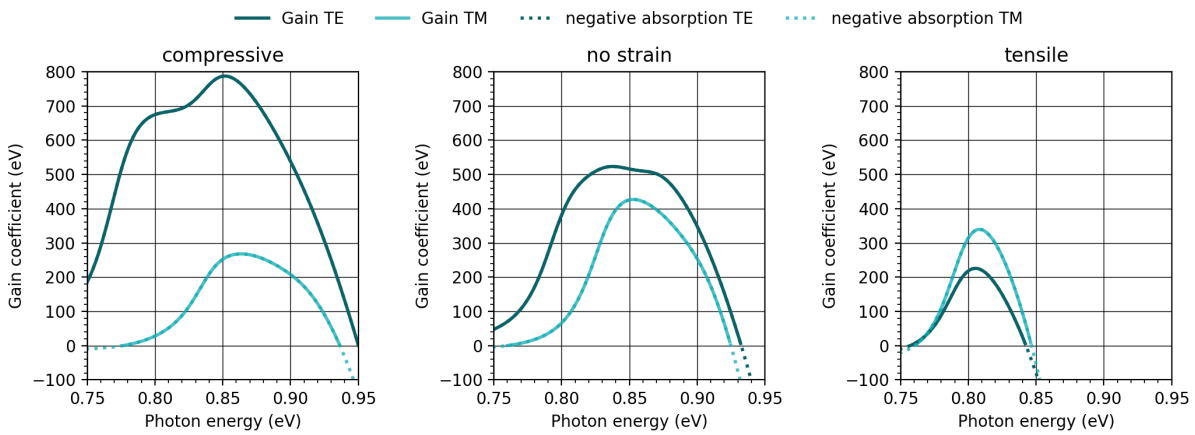


Figure 6.4.12.86: Calculated optical gain of TE and TM optical mode for compressively strained QW (**left**), unstrained QW (**center**) and tensily strained QW (**right**)

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,
- 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: nn/nn/nnnn

Optics: Optical gain and spontaneous emission rate of strained GaN quantum well

Section author: Naoki Mitsui

> **Warning:** This tutorial is under construction

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 (6.4.4.4).

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

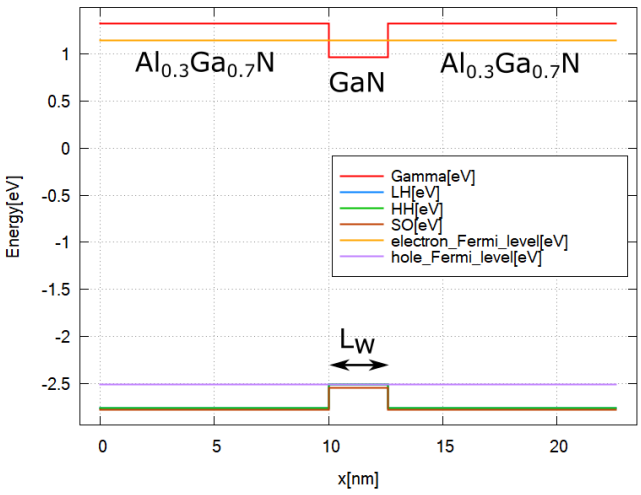


Figure 6.4.12.87: The band edges and Fermi energies for $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 3 \times 10^{19}$ cm^{-3} inside the well region.

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 $\times$ 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:

$$r^{spon}(\vec{\epsilon},\omega) = \frac{n_r e^2 E}{\pi\hbar^2 c^3 \varepsilon_0 m_0^2}\frac{2}{V}\sum_{n>m}\sum_{\mathbf{k}_\parallel}|\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathbf{k}_\parallel)|^2\mathcal{L}(E_n(\mathbf{k}_\parallel)-E_m(\mathbf{k}_\parallel)-E)f_n(\mathbf{k}_\parallel)(1-f_m(\mathbf{k}_\parallel)), \tag{6.4.12.11}$$

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]*.

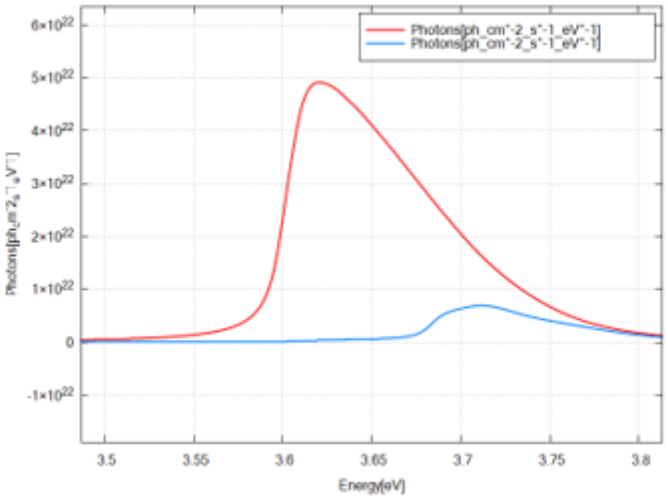


Figure 6.4.12.88: r^{spon} for an $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 2.6$ [nm]

When we don't apply the linewidth broadening, the result shows the exact energy where the emission by each pair of state starts.

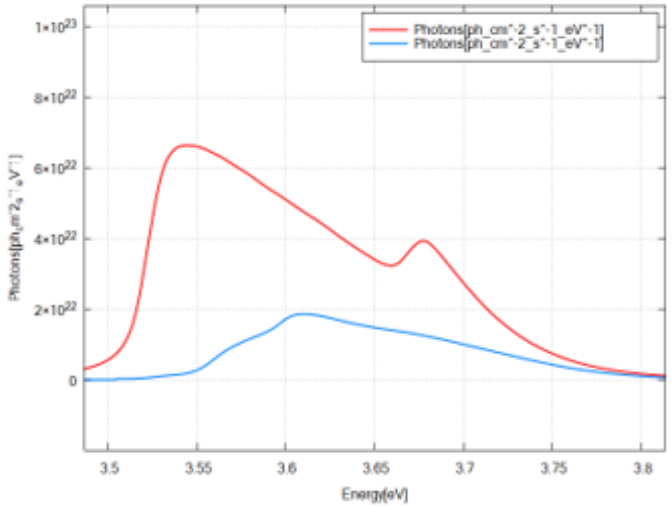


Figure 6.4.12.89: r^{spon} for an $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 5.0$ [nm]

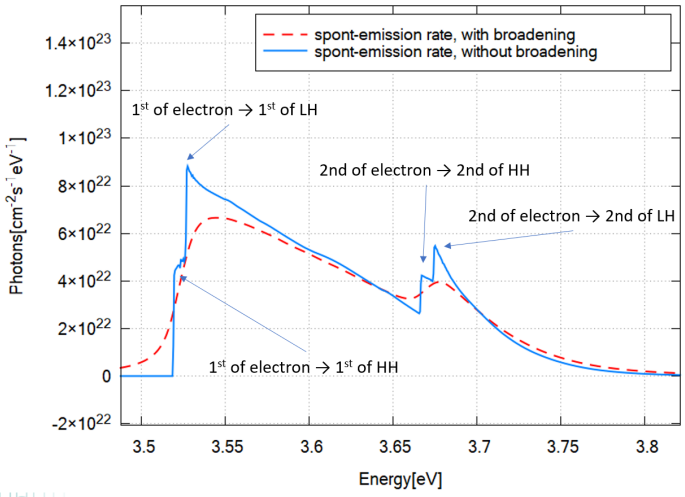


Figure 6.4.12.90: TE emission rate in Figure 6.4.12.89 with (red dashed line) and without (blue line) line width broadening.

Optical Gain

The optics section can calculate the absorption spectra $\alpha(\vec{\epsilon}, \omega)$. This can be understood as a negative gain, i.e.

$$\alpha(\vec{\epsilon}, \omega) = -g(\vec{\epsilon}, \omega) \tag{6.4.12.12}$$

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.

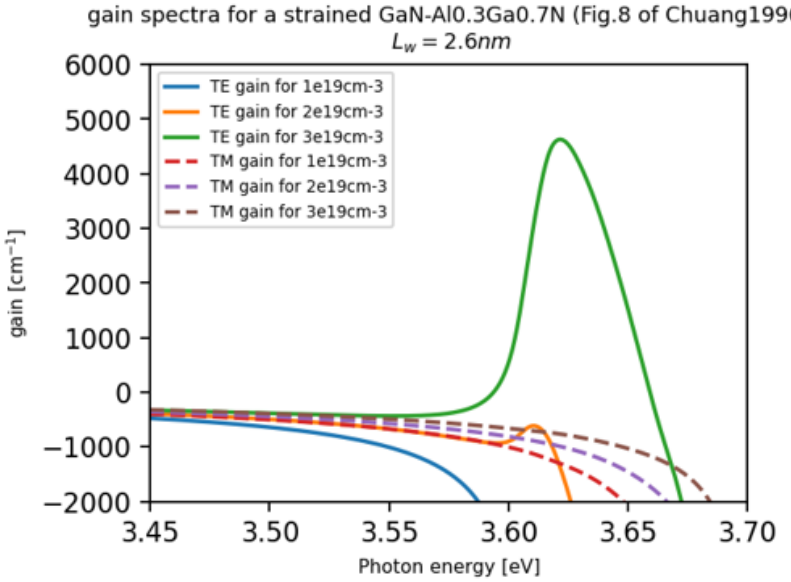


Figure 6.4.12.91: $g(\vec{\epsilon}, \omega)$ for a $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 1, 2, 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 2.6$ [nm]

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.

Last update: nn/nn/nnnn

Excitons

Exciton absorption in infinite quantum well

Input files:
1D_InterbandExcitonAbsorption_InfiniteWell_GaAs_8kp_nnp.in
1D_InterbandExcitonAbsorption_InfiniteWell_GaAs_effective_mass_nnp.in

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

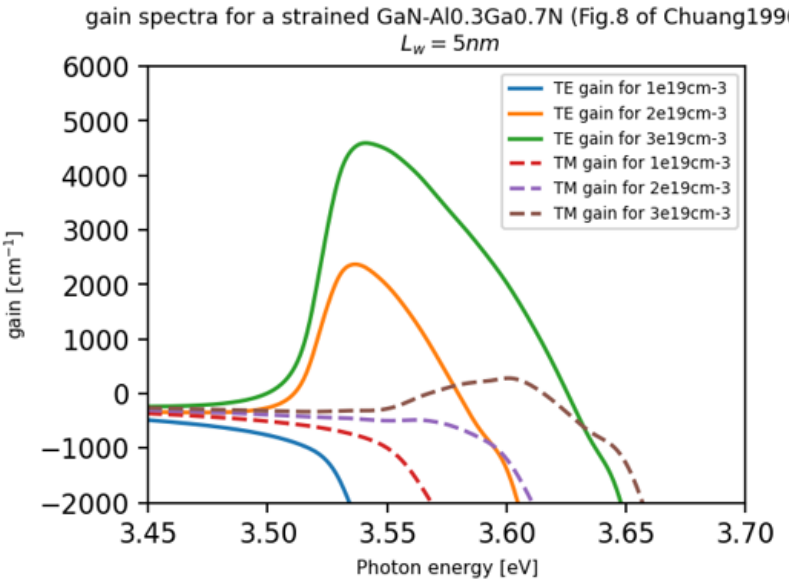


Figure 6.4.12.92: $g(\vec{\epsilon}, \omega)$ for a $Al_{0.3}Ga_{0.7}N$-GaN quantum well with the carrier concentration $n = 1, 2, 3 \times 10^{19}$ cm^{-3} on each polarization TE (x or y) and TM (z). $L_w = 5.0$ [nm]

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 6.4.12.93

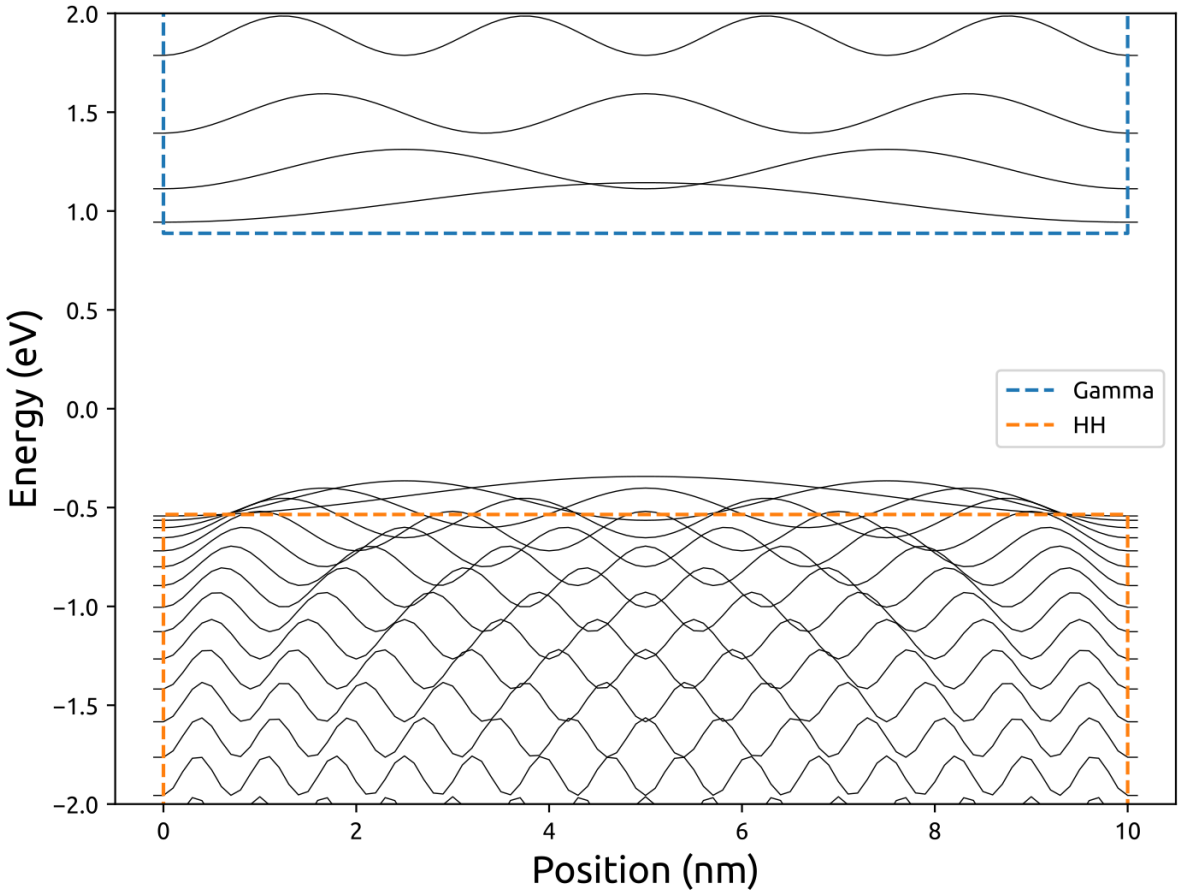


Figure 6.4.12.93: Computed eigenstates in the GaAs infinite quantum well with effective mass Hamiltonian in conduction and valence bands. The colored dashed line are band edges, the solid lines are eigenstates.

In the figure below, the computed absorption in the quantum well is shown (Figure 6.4.12.94). 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 6.4.12.95

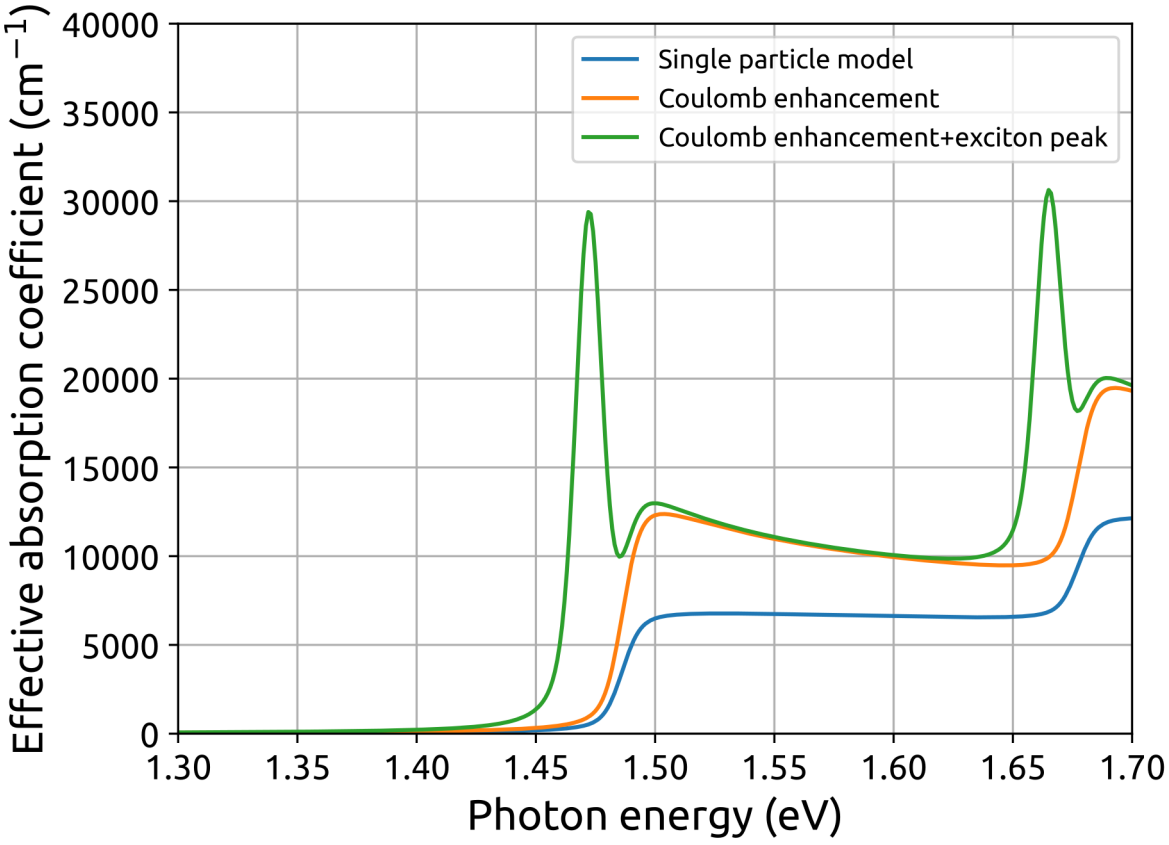


Figure 6.4.12.94: Absorption in infinite quantum well computed with effective mass Hamiltonians. The figure shows absorption with and without exciton correction.

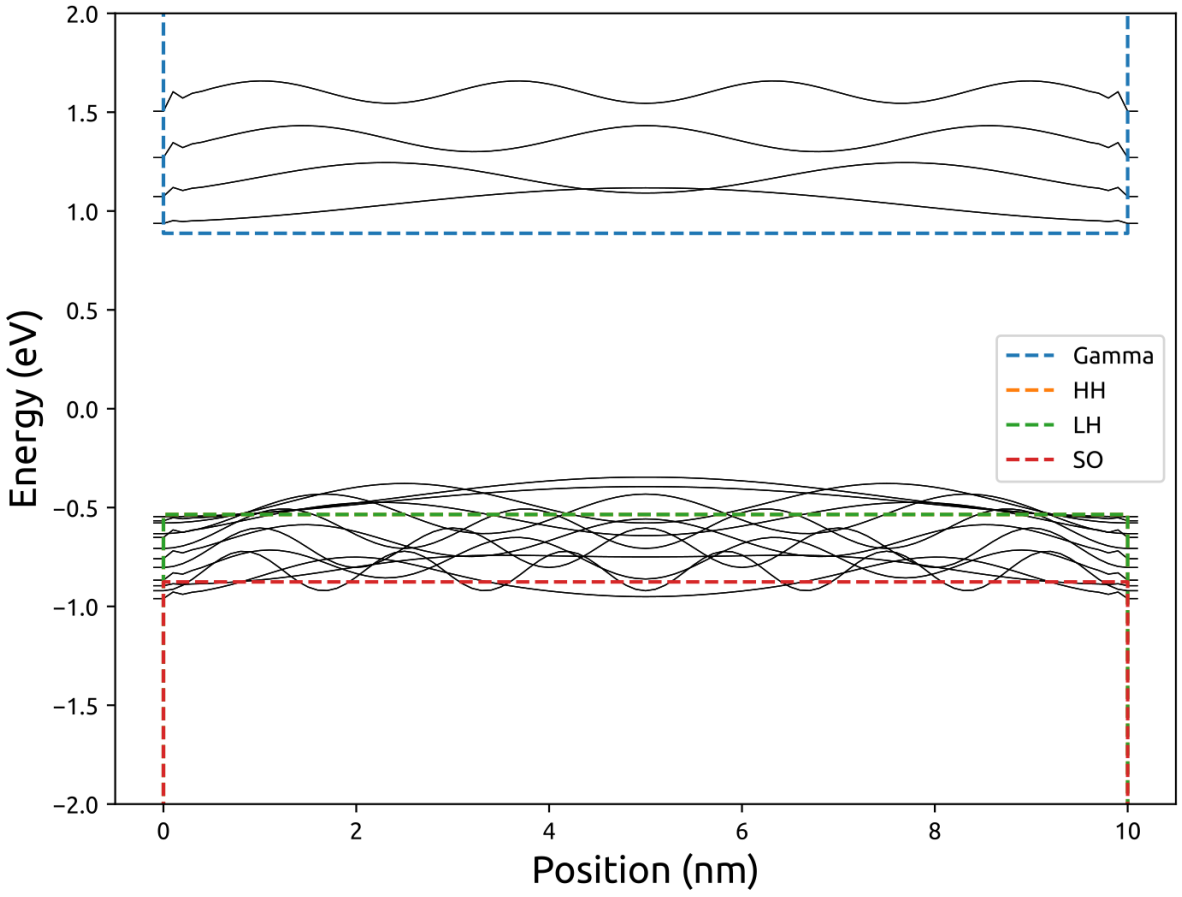


Figure 6.4.12.95: Eigenstates in the GaAs infinite quantum well computed with 8-band kp Hamiltonian. THe colored dashed line are band edges, the solid lines are eigenstates.

In the figure below, the computed absorption in the quantum well is shown (Figure 6.4.12.96). Similarly to the Simulation 1, the figure shows the absorption with and without exciton correction.

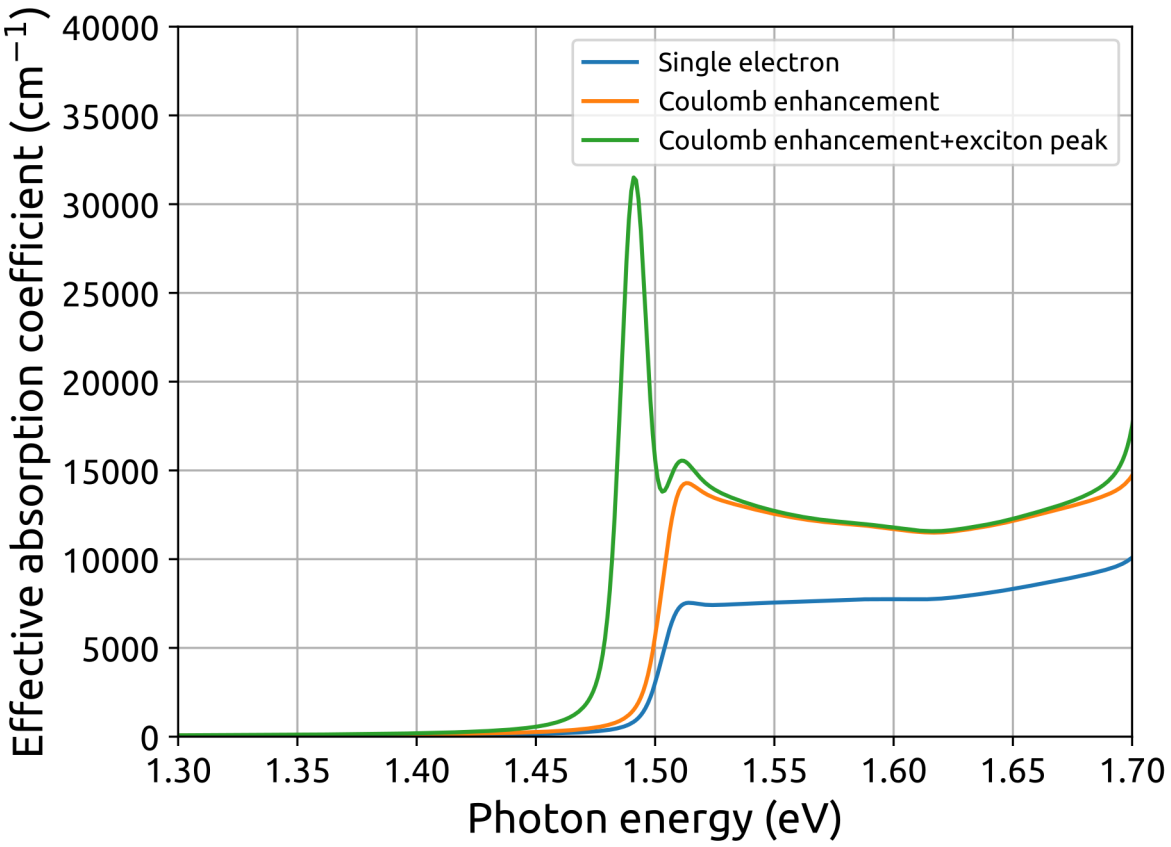


Figure 6.4.12.96: Absorption in infinite quantum well computed with 8-band kp Hamiltonian. The figure shows absorption with and without exciton correction.

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.

Acknowledgment

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 grant agreement No 101017194.

Last update: nn/nn/nnnn

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 6.4.12.97 shows the wave functions in conduction and valence bands.

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"*. The absorption spectra at different biases is shows in the figure Figure 6.4.12.98.

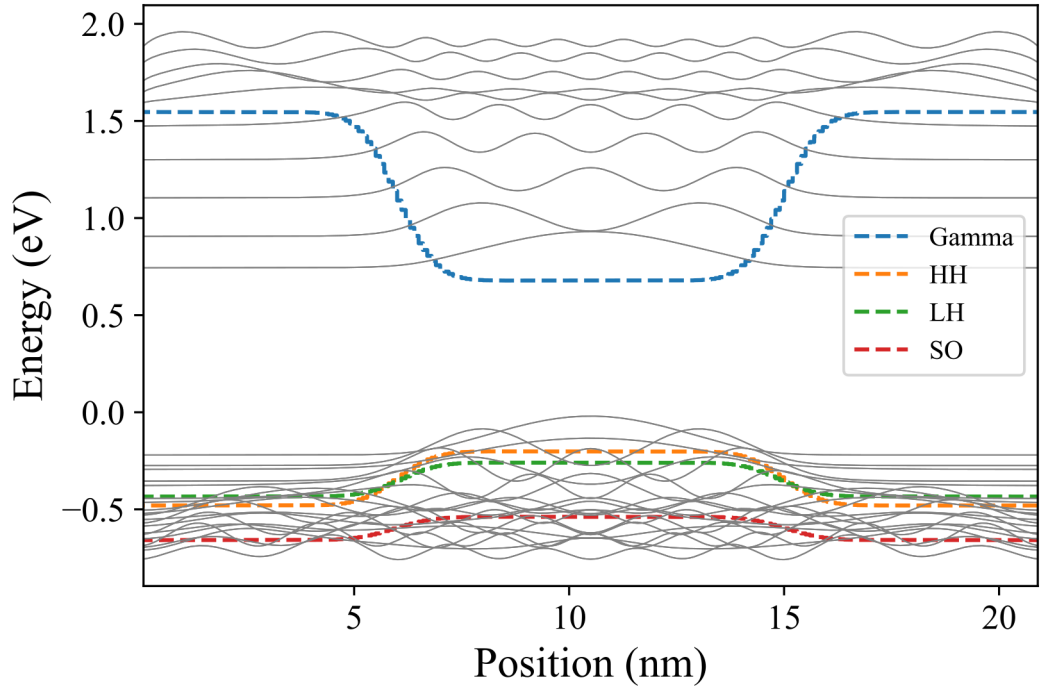


Figure 6.4.12.97: The band edges (colored) and the wave function probabilities (gray) in the quantum well under 0 bias.

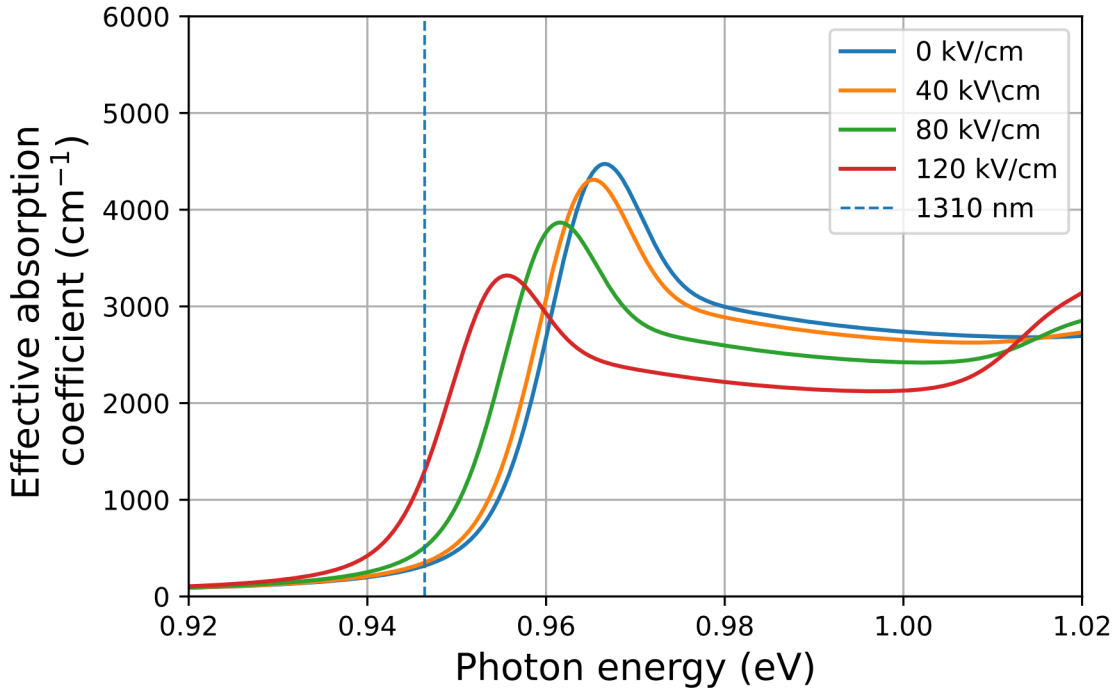


Figure 6.4.12.98: Excitonic absorption spectra in the device. Labels indicate electric field in the middle of the quantum well.

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.

Acknowledgment

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 grant agreement No 101017194.

Last update: nn/nn/nnnn

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 6.4.12.99

When additional bias applied, the electric filed in the quantum well is increased (figure Figure 6.4.12.99)

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.

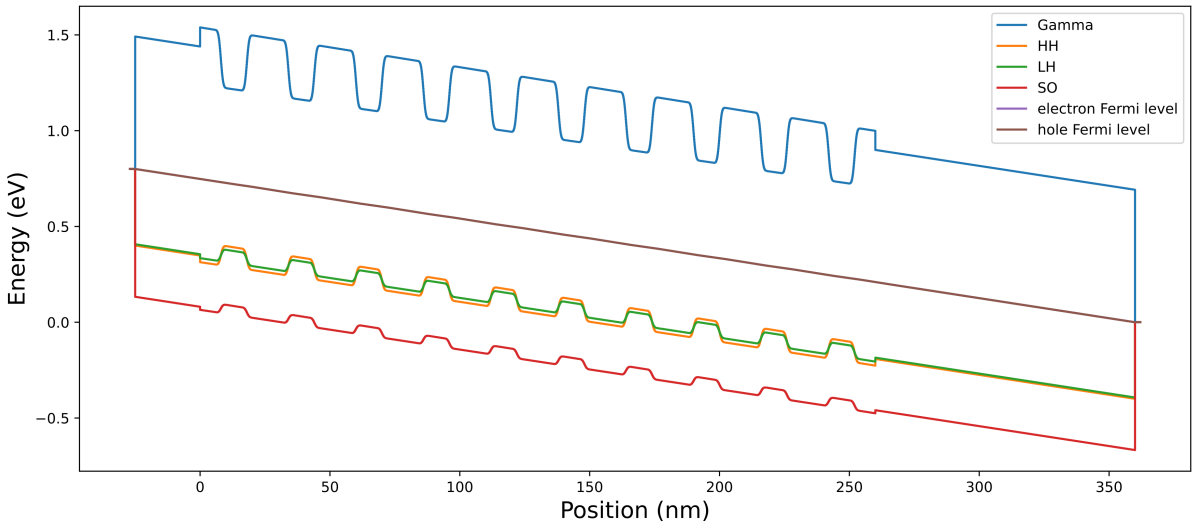


Figure 6.4.12.99: Valence and conduction band edges at zero bias. The electric field is induced by 0.8V built-in potential, included in the simulation

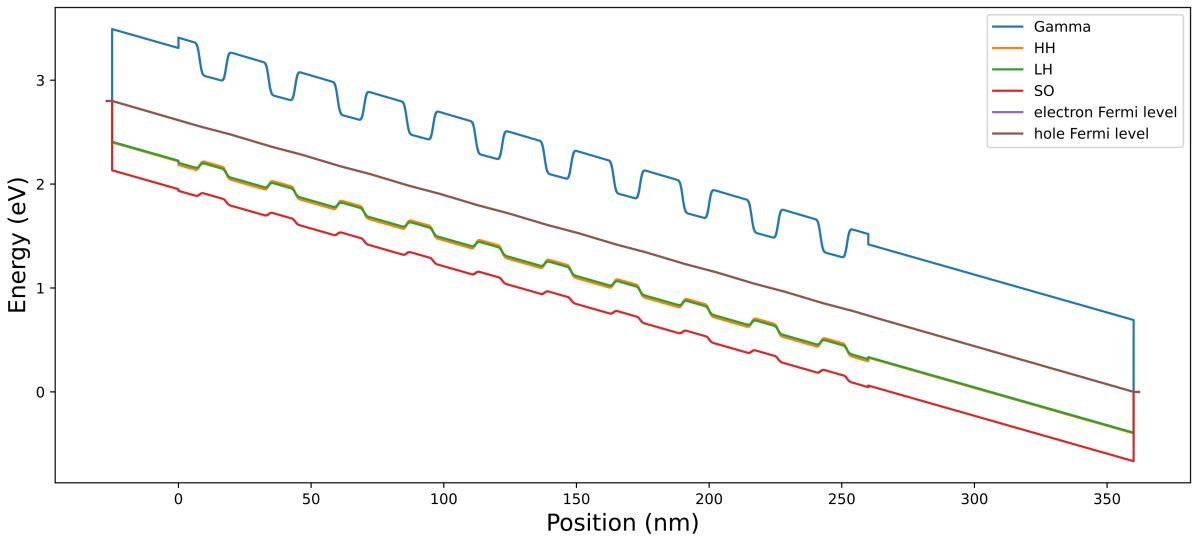


Figure 6.4.12.100: Valence and conduction band edges at 2V external bias.

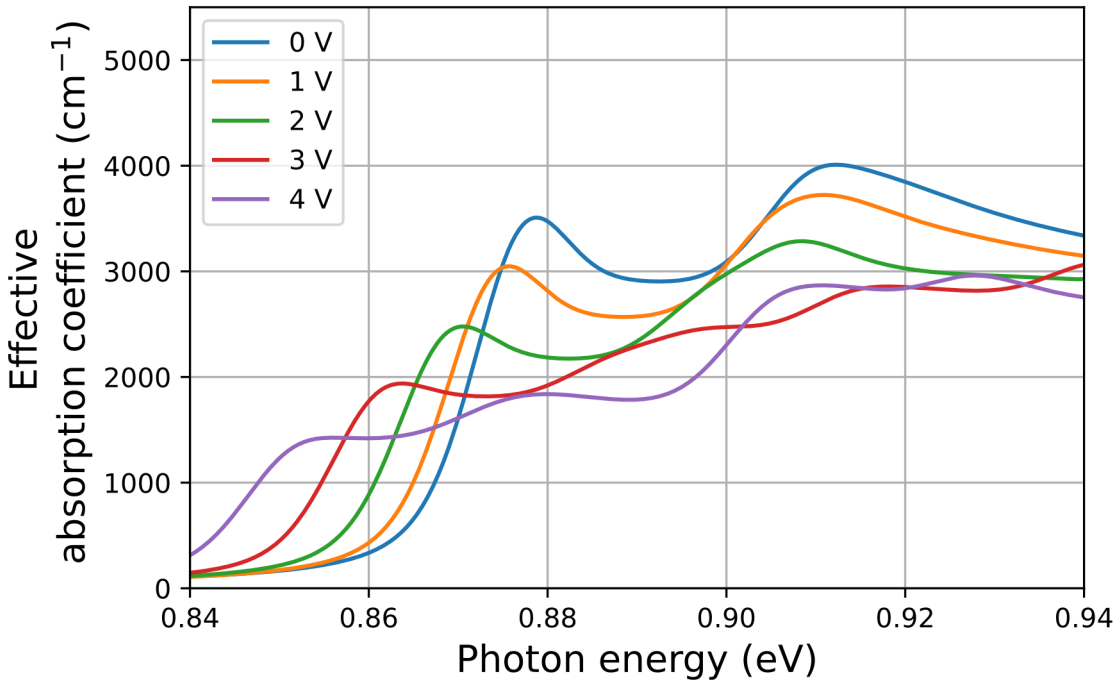


Figure 6.4.12.101: Absorption spectra inside the MQW region at different external bias

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 6.4.12.102.

The band edges at zero bias is shown in the figure Figure 6.4.12.103

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 6.4.12.104).

The absorption spectra computed in this example are shown in the Figure 6.4.12.105.

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.

Acknowledgment

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

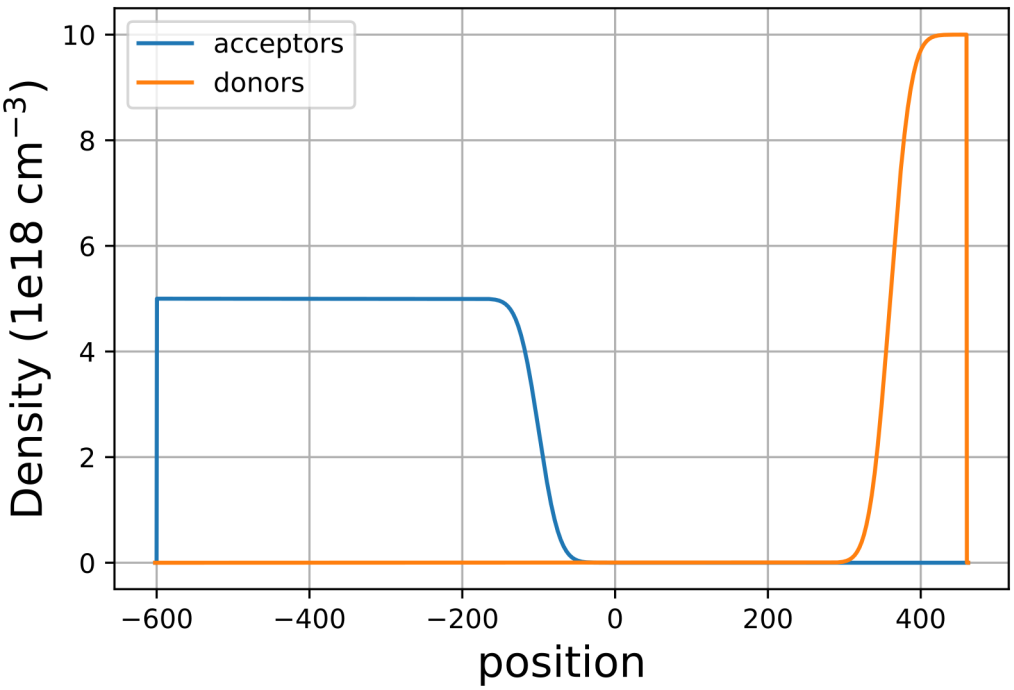


Figure 6.4.12.102: The doping profile in the device

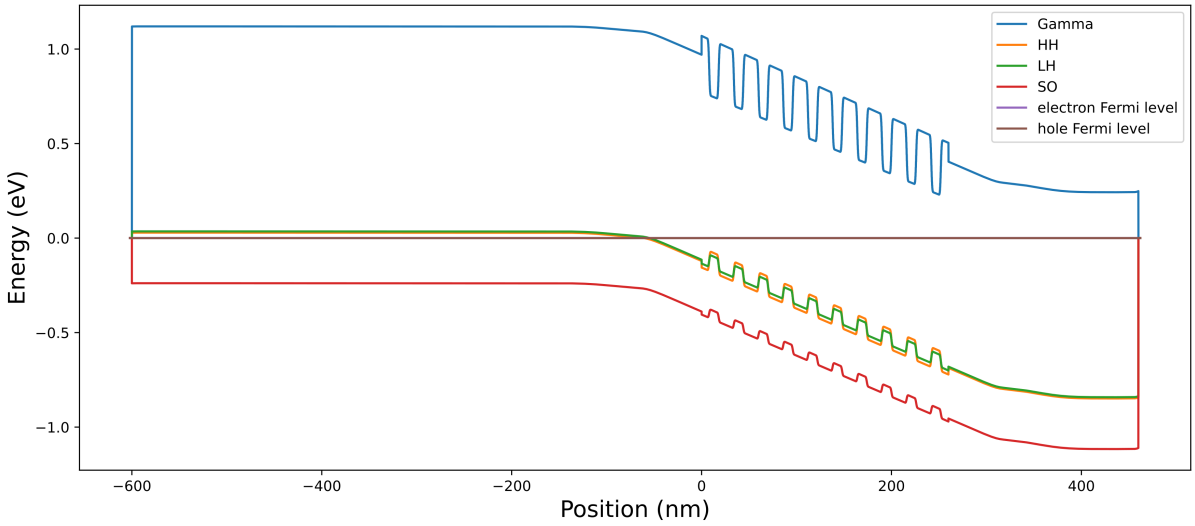


Figure 6.4.12.103: Valence and conduction band edges at zero bias

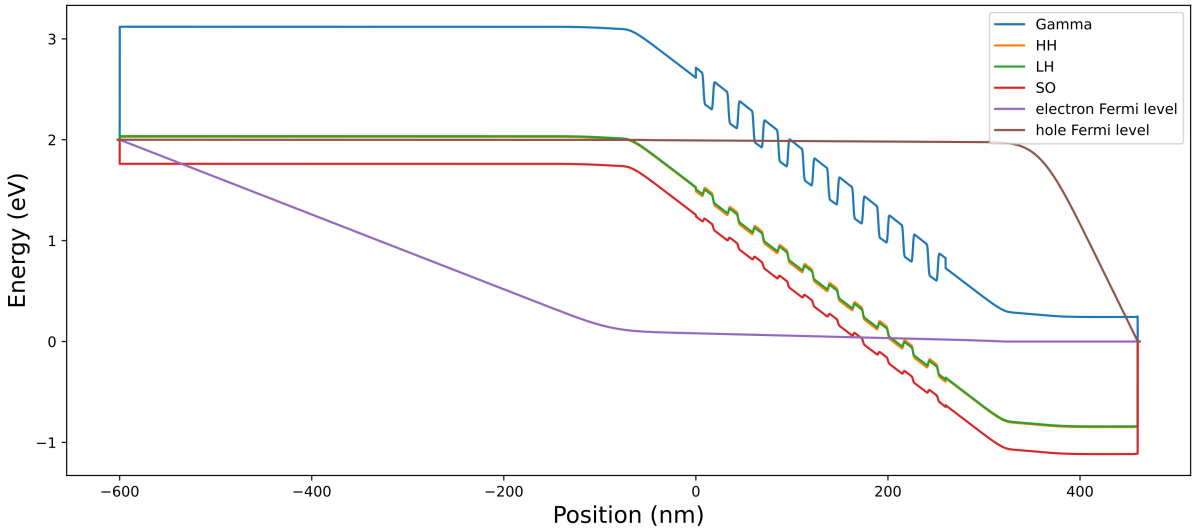


Figure 6.4.12.104: Valence and conduction band edges at zero bias

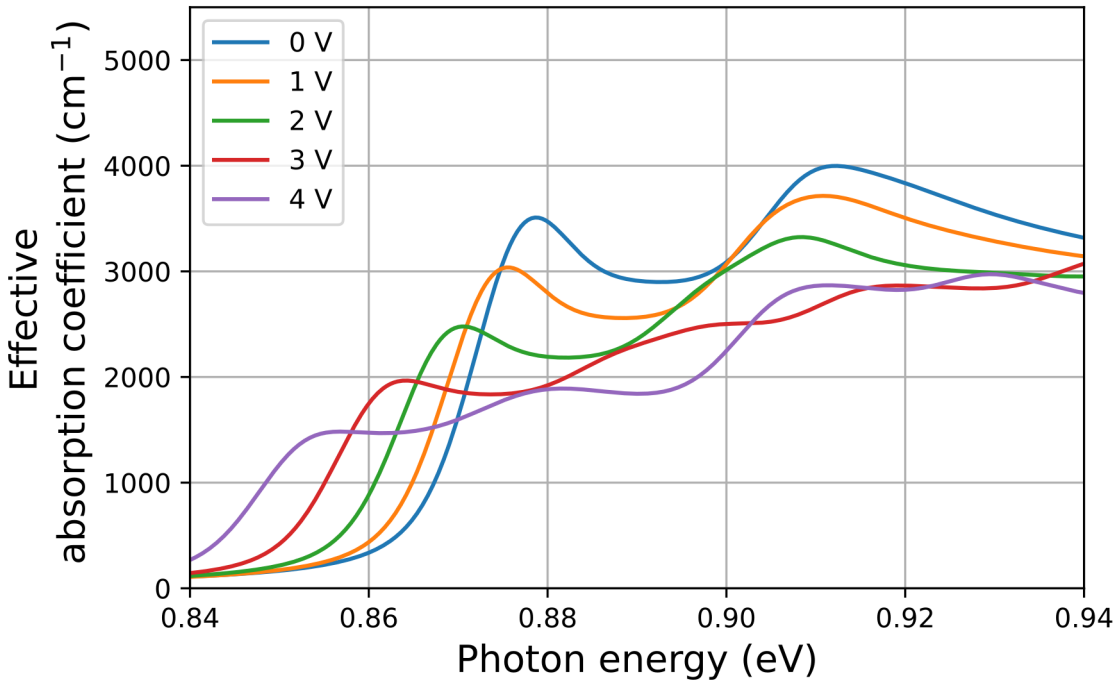


Figure 6.4.12.105: Absorption spectra inside the MQW region at different external bias for Simulation 2

program under grant agreement No 101017194.

Last update: nn/nn/nnnn

6.4.13 2-Dimensional Electron Gases (2DEGs)

— 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:

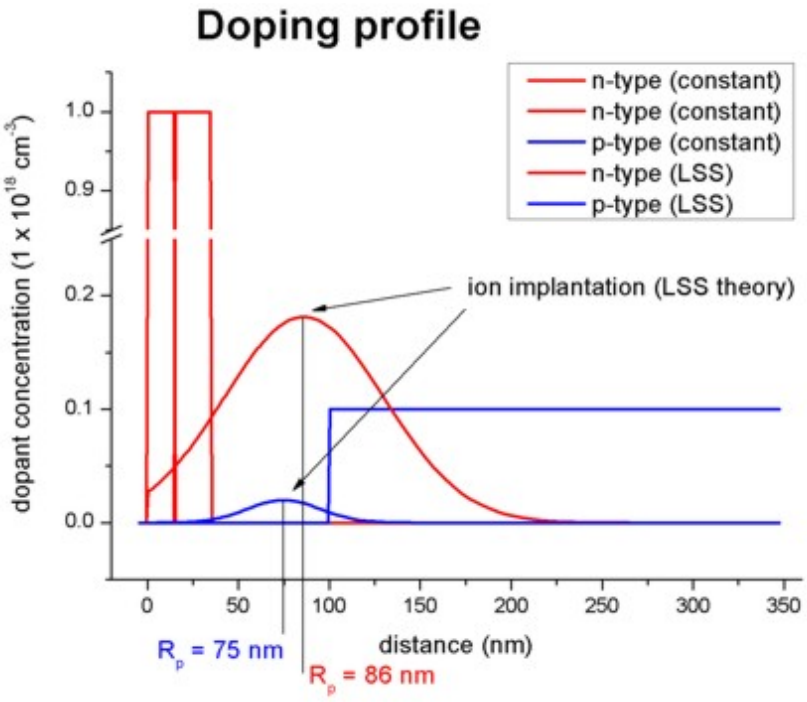


Figure 6.4.13.1: Doping profiles separated by each region.

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 *structure{ region{ 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.

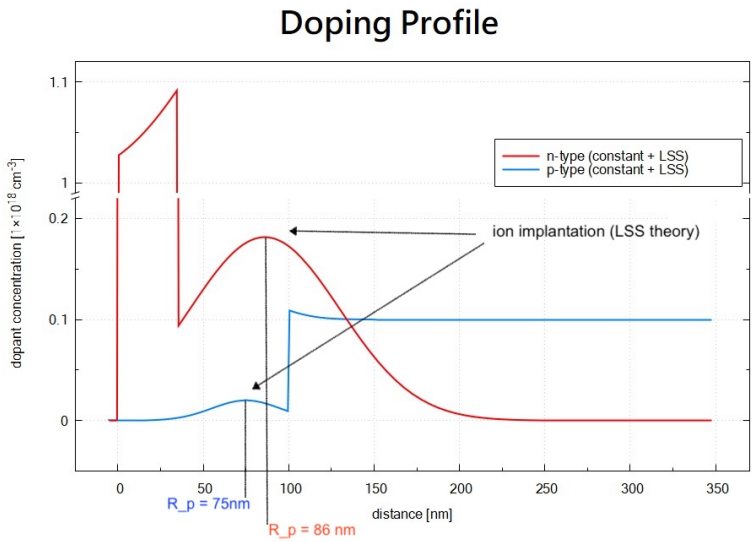
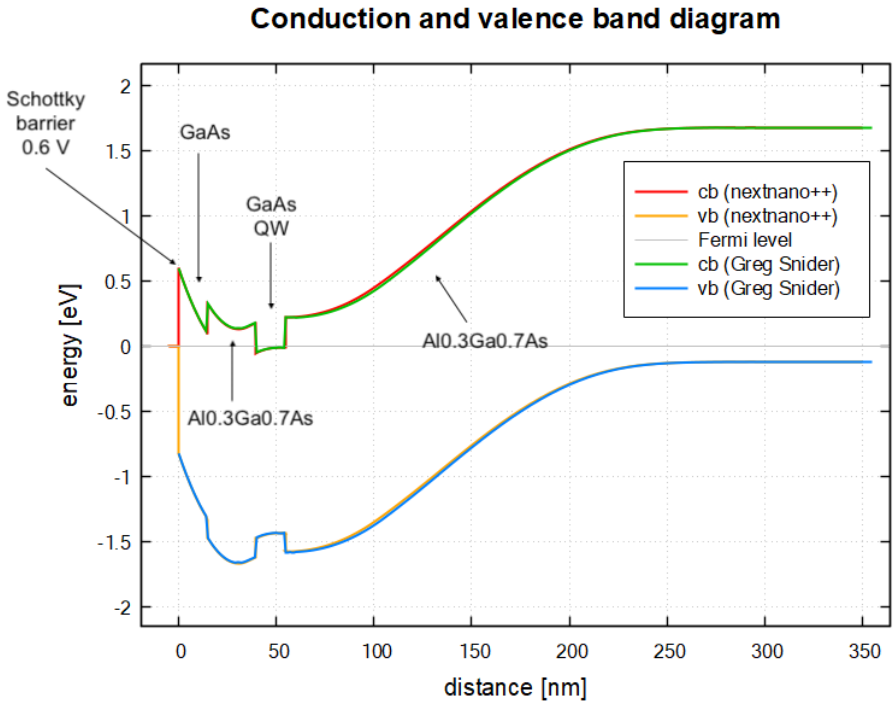
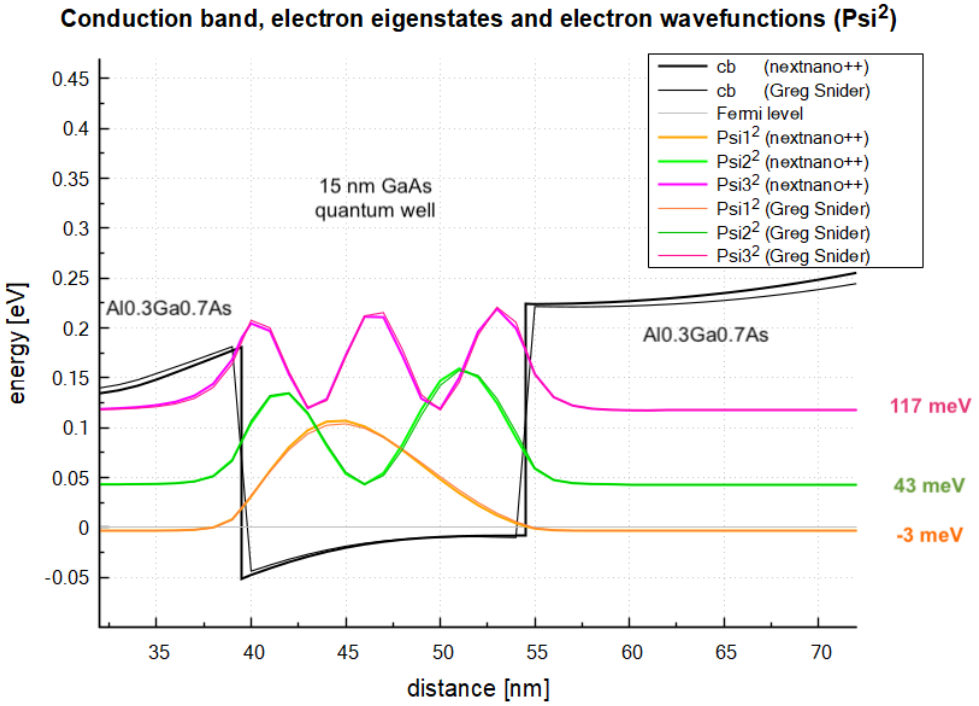


Figure 6.4.13.2: Resulting doping profiles.



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.

Electron states	*nextnano++*	*nextnano*^{3}	Greg Snider's "1D Poisson" code
E_1 [meV]	-3.1	-3.0	-1.3
E_2 [meV]	43.4	43.5	44.0
E_3 [meV]	117.4	117.5	117.8

Electron and hole densities

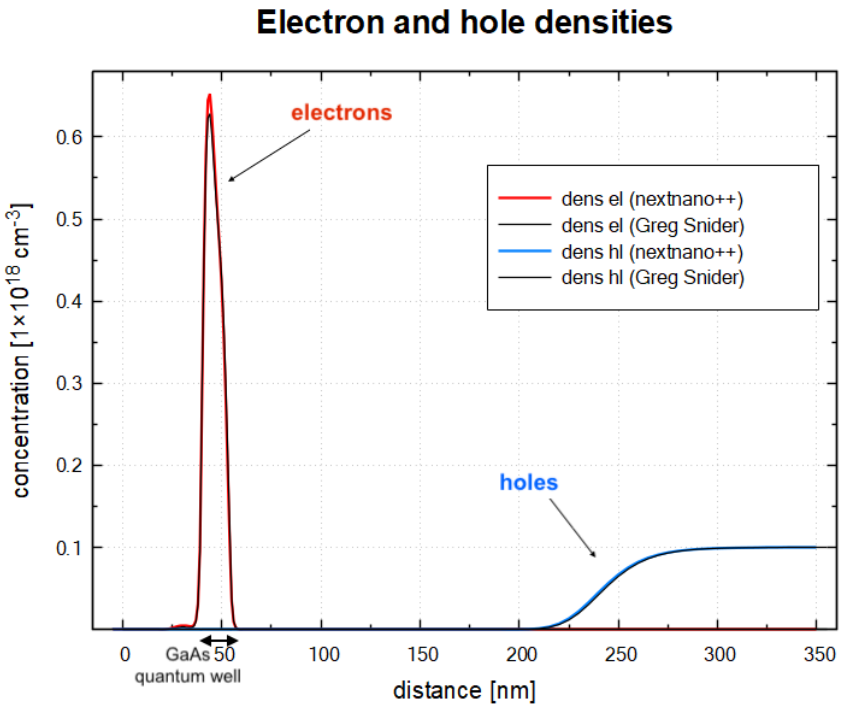
The electron and hole densities are depicted in this figure, there is also nice agreement between the two codes.

- 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 $f(E) := \frac{1}{e^{(E-E_f)/k_B T}+1}$.

$$n^{el}(\mathbf{r}) := \int dE \rho^{el}(\mathbf{r}, E) f(E)$$
$$n^{hole}(\mathbf{r}) := \int dE \rho^{hole}(\mathbf{r}, E) f(E)$$

3. Solve Poisson equation and obtain the potential distribution $\phi(\mathbf{r})$ caused by the distributed electrons, holes, and ions.

$$\nabla \cdot \big(\epsilon_s(\mathbf{r})\nabla\big)\phi(\mathbf{r}) = \frac{-e[n^{hole}(\mathbf{r}) - n^{el}(\mathbf{r}) + N_D(\mathbf{r}) - N_A(\mathbf{r})]}{\epsilon}$$

   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 $\longrightarrow$ End
   - No $\longrightarrow$ 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.

Last update: nn/nn/nnnn

— DEV — Shubnikov-de Haas effect and subband occupation of 2DEG

Attention: The tutorial is under development

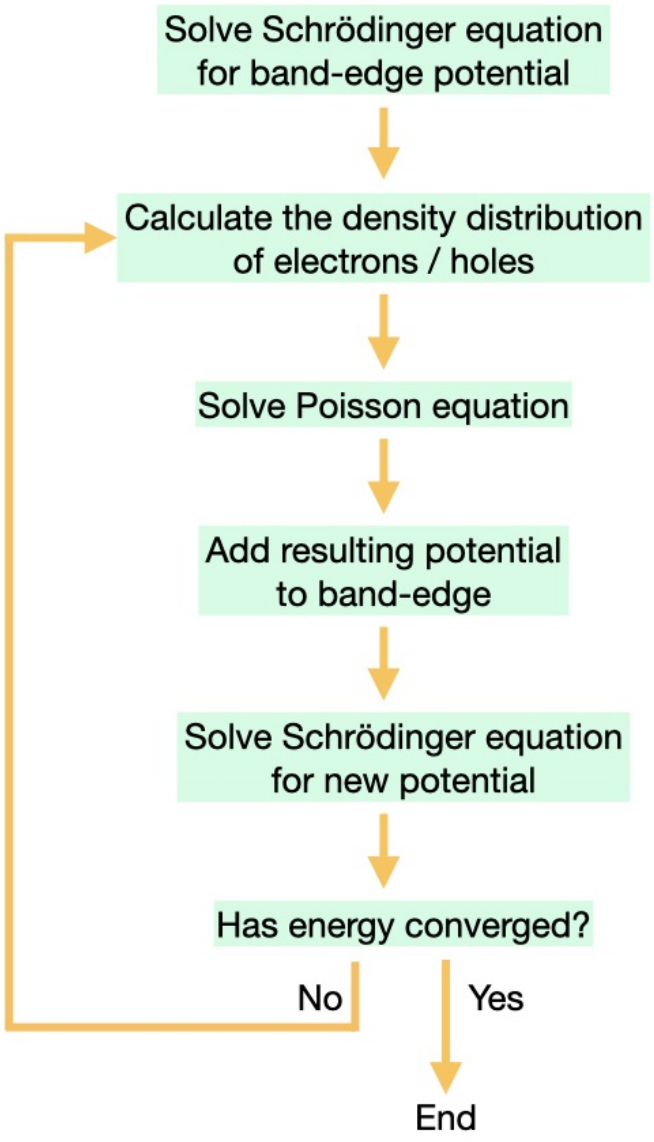
Depletion of electrons in a two-dimensional electron gas (2DEG)

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 6.4.13.3 and Figure 6.4.13.4 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.

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.



Schematics of the self-consistent iteration

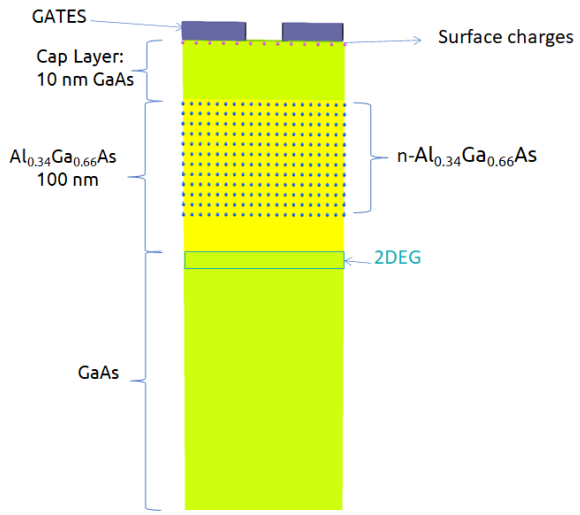


Figure 6.4.13.3: Schematics of a side view of the simulated device

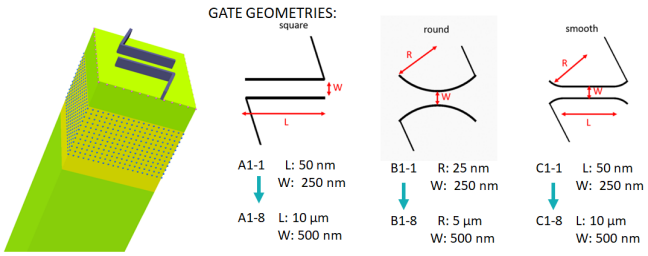


Figure 6.4.13.4: 3-dimensional schematics of the simulated structure and typical shapes of gates

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 6.4.13.5) 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.

A slice of the conduction band at the mid-distance between the gates across the growth direction is displayed in Figure 6.4.13.6. 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.

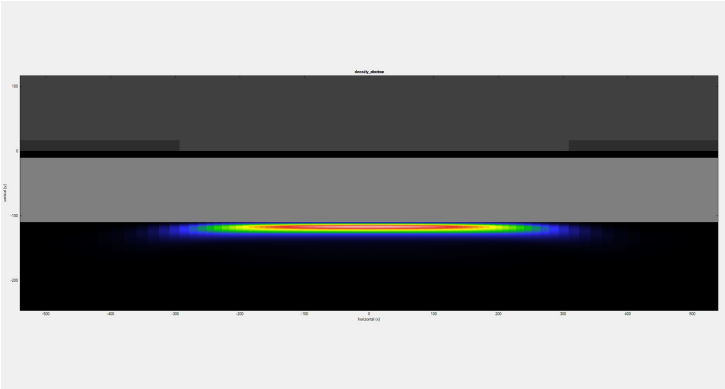


Figure 6.4.13.5: Density of electrons resulting from a 2D simulation as a negative bias is applied to the gates

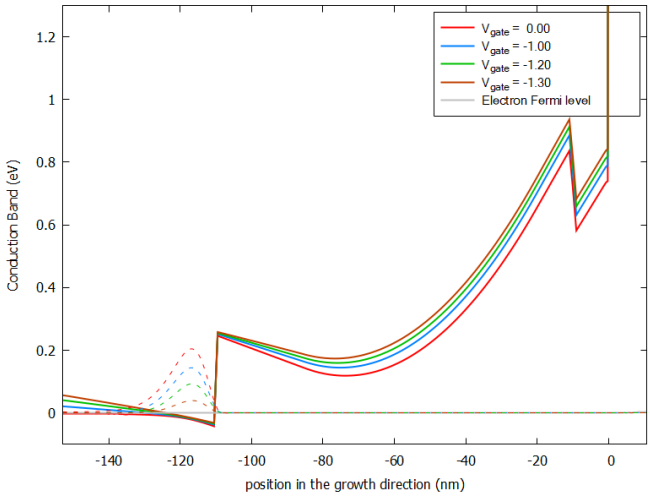


Figure 6.4.13.6: Conduction band (solid lines) and density of electrons (dashed lines) resulting for a 2D simulation as function of the applied bias to the gate. This plot corresponds to the results at the mid-distance of the gates across the growth direction.

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 6.4.13.7 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 6.4.13.8 and Figure 6.4.13.9 show that in this case, the pinch-off voltage V_{gate} is around -1.20 and 1.30 V.

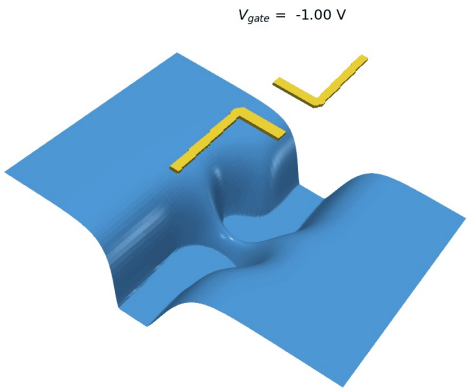


Figure 6.4.13.7: Density of electrons resulting from a 2D simulation as a negative bias is applied to the gates

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.

Acknowledgment

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.

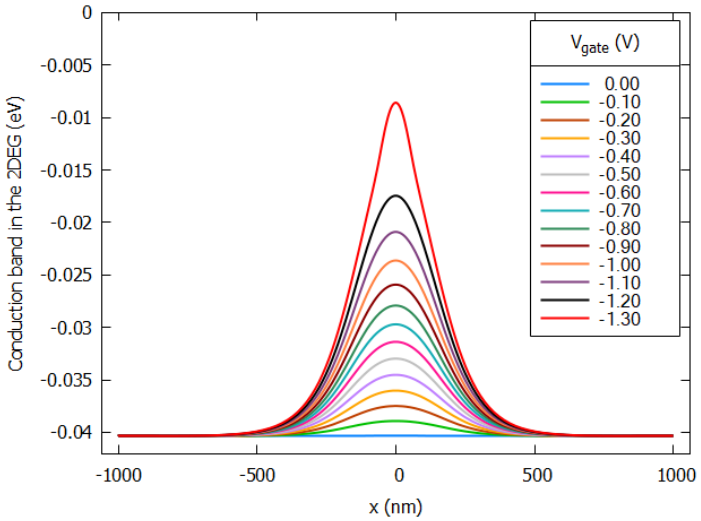


Figure 6.4.13.8: Slice of the computed conduction band in the 2DEG region at 111 nm under the surface as function of the applied bias to the gate. From the image the pinch-off voltage occurs around -1.30 and -1.20 V.

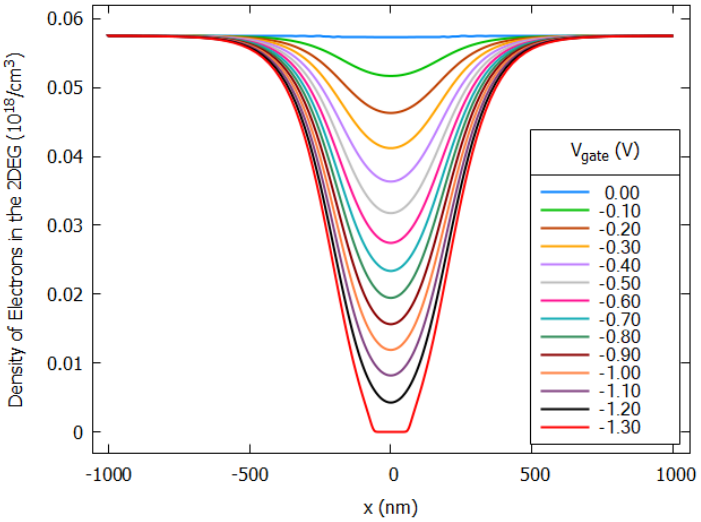
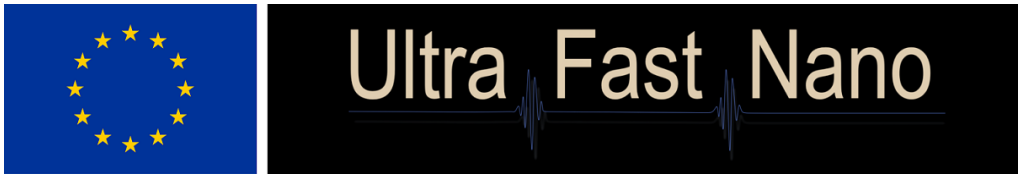


Figure 6.4.13.9: Slice of the computed density of electrons in the 2DEG region at 111 nm under the surface as function of the applied bias to the gate. From the image the pinch-off voltage occurs around -1.30 and -1.20 V.



Last update: nn/nn/nnnn

6.4.14 Transmission and Conductance (CBR method)

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 $E_{barrier} = 0.3\ \mathrm{eV}$.

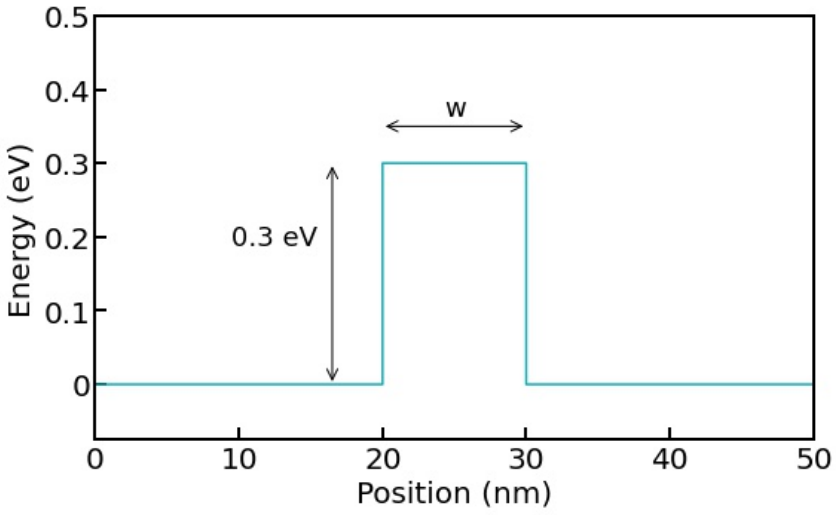
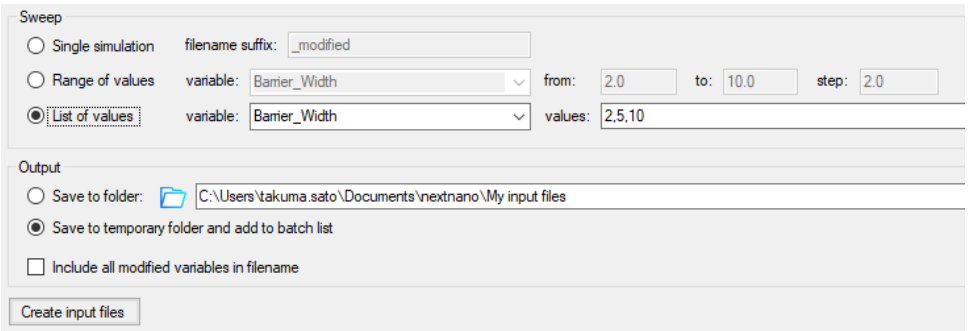


Figure 6.4.14.1: The conduction band edge profile (*bandedge_Gamma.dat*).

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.



The result is written in *transmission_cbr_Gammma.dat*. The barrier width w affects the transmission coefficient as shown in Figure 6.4.14.2.

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 ($w = 5\ \mathrm{nm}$ in this example). Quantum mechanics also predicts a oscillatory behavior above $E_{barrier}$.

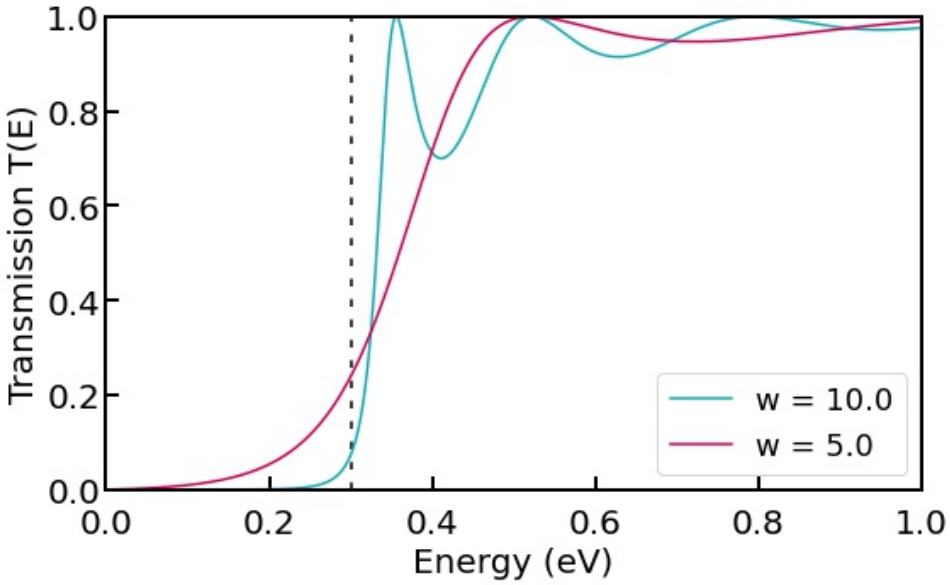


Figure 6.4.14.2: The transmission coefficient as a function of energy for different barrier width w (nm). The dashed line marks $E_{barrier}$.

Step potential

For a step potential structure (*transmission-step_1D_nnp.in*) as shown in Figure 6.4.14.3 (a), the transmission of electrons with energy below $E_{barrier}$ is prohibited because the barrier is infinitely thick.

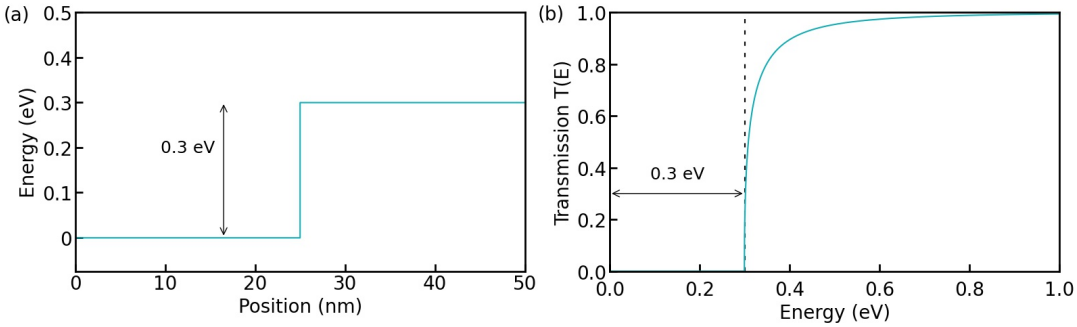


Figure 6.4.14.3: The conduction band edge profile is shown in (a). The transmission spectrum for a step potential is shown in (b). Transmission is only allowed above the step.

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.

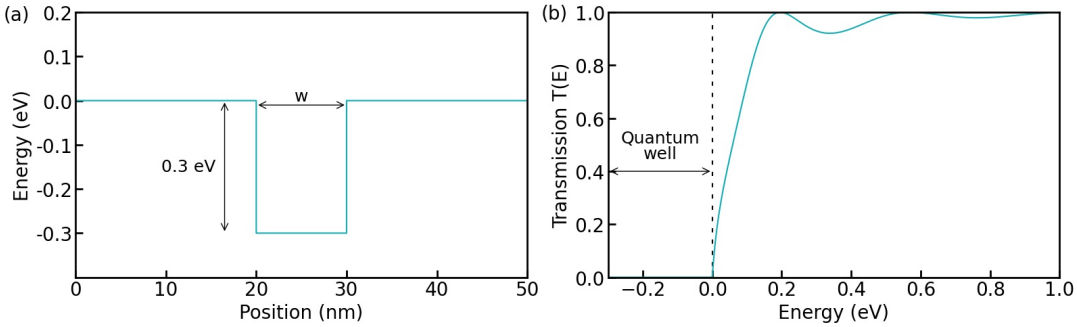


Figure 6.4.14.4: The conduction band edge profile is shown in (a). The transmission spectrum for a quantum well is shown in (b). The dashed line marks the top of the barrier.

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.

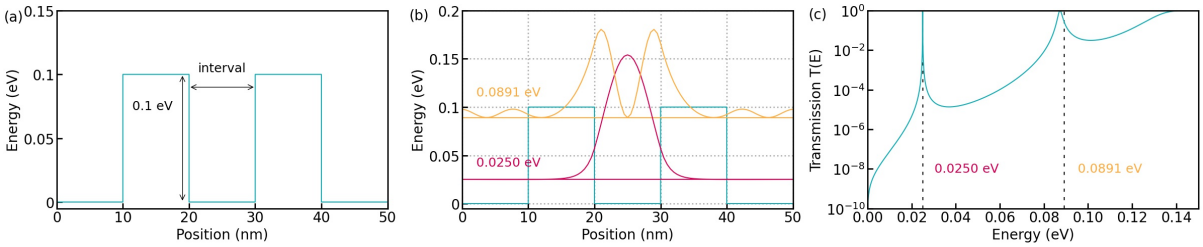


Figure 6.4.14.5: The conduction band edge profile is shown in (a). The probability distribution $|\psi(x)|^2$ of the two resonant modes. The transmission coefficient of the double barrier structure is shown in (c). The spectrum has two sharp peaks below the barrier height 0.1 eV, which corresponds to 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.

See also for *3D case* for another CBR efficiency assessment.

Last update: nn/nn/nnnn

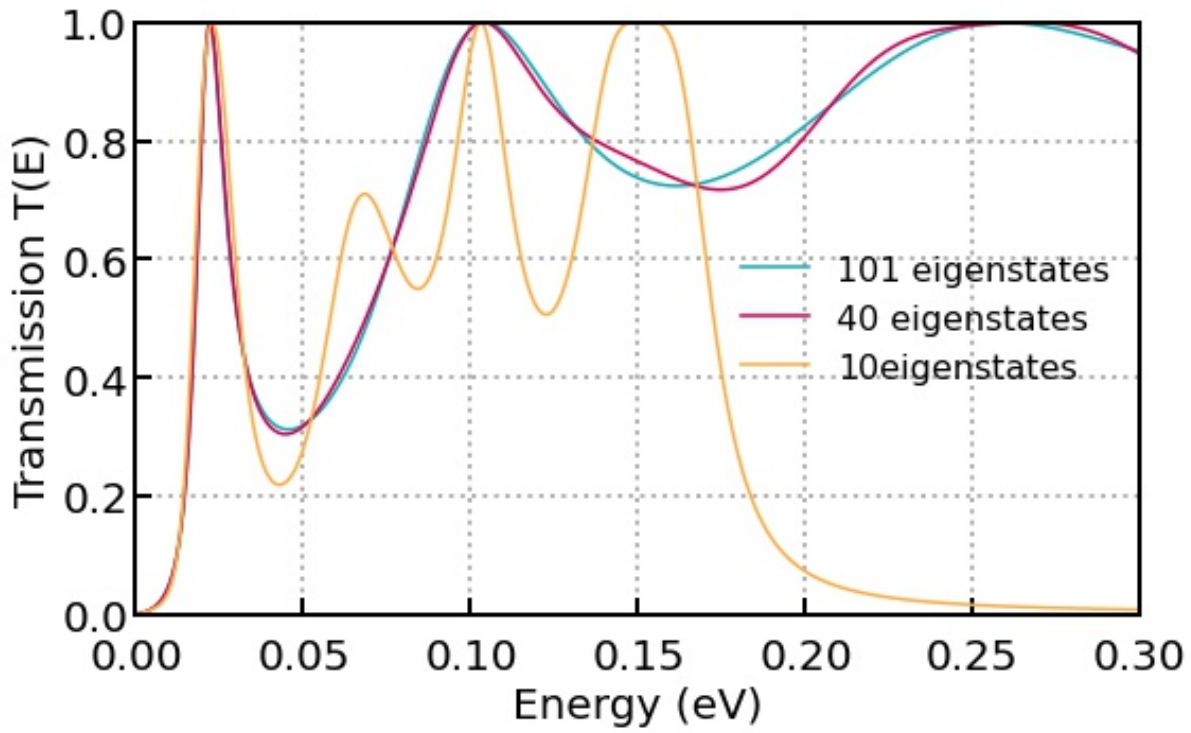


Figure 6.4.14.6: Transmission coefficient for three different CBR parameters. The blue curve is the result considering complete set of eigenstates, whereas violet and orange curves take into account only 40 % and 10 % of them, respectively.

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++*
- *1D_GaAs_conductance_nn3.in* - simulations of 1D quantum wire in *nextnano*^{3}
- *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 6.4.14.7. 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.

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.

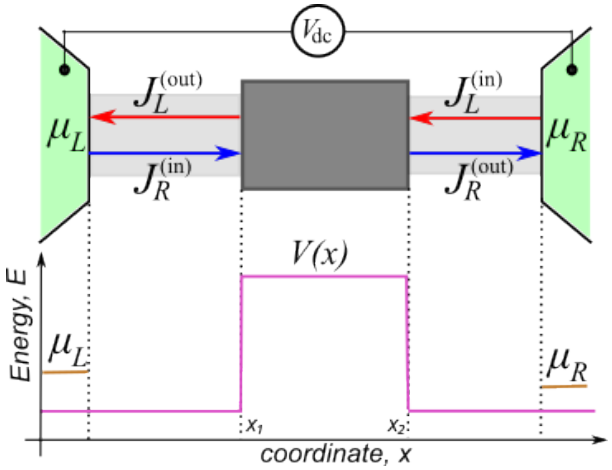


Figure 6.4.14.7: Upper panel: Landauer setup. Left and right leads (green regions) are connected to a semiconductor device (dark gray square) via connecting wires (light gray regions). Lower panel: chemical potentials of the leads (orange lines) and the energy of the potential barrier (magenta lines).

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, $\mu_R - \mu_L = eV_{\rm dc}$.

> **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 $J_{\rm R,L}^{(\rm out)}$ 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 6.4.14.7. 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:

$$\mathcal{T}(0 < E < V_0) = 1/\Big(1 + [\kappa \sinh\Big(\tilde{k}a\Big)]^2\Big), \tag{6.4.14.1}$$

$$\mathcal{T}(E > V_0) = 1/\Big(1 + [\kappa \sin\Big(\tilde{k}a\Big)]^2\Big); \tag{6.4.14.2}$$

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: $J = J_R^{(\rm in)} - J_L^{(\rm out)} = J_R^{(\rm out)} - J_L^{(in)}$. 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:

$$J = 2e \int \frac{dk}{2\pi} v(k)\mathcal{T}(k)\Big(f_L(k) - f_R(k)\Big) = \frac{2e}{h} \int dE\, \mathcal{T}(E)\Big(f_L(E) - f_R(E)\Big); \tag{6.4.14.3}$$

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 (6.4.14.3) 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 $J = G_0 V_{\rm dc}$ 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 6.4.14.8. The numerical simulations presented in this tutorial were done by using *Contact Block Reduction method [CBR]*, see also *tutorial on the CBR method*.

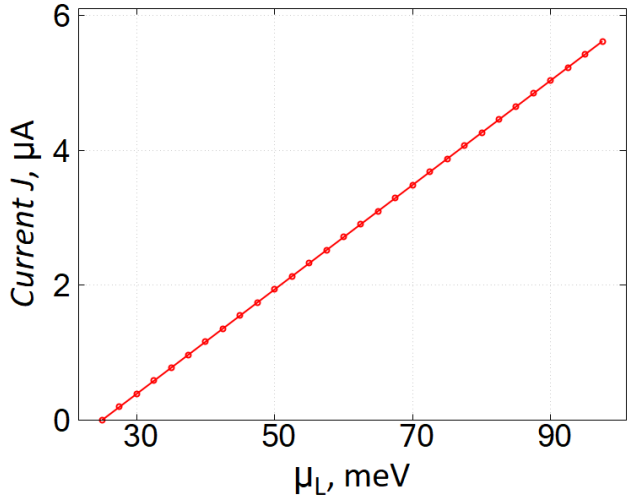


Figure 6.4.14.8: Numerically calculated IV-characteristics of a ballistic 1D conductor, $V_0 = 0$. We have chosen GaAs as the material of the conductor with the total length 32 nm at $\mu_R = 25\text{meV}$; the temperature was set to $T = 50\text{mK}$. Note that these parameters has no influence on the 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 6.4.14.9 and Figure 6.4.14.10 shows the transmission and the IV characteristics of the device which contains the square scattering potential of width 30 nm with $V_0 = 100\text{meV}$.

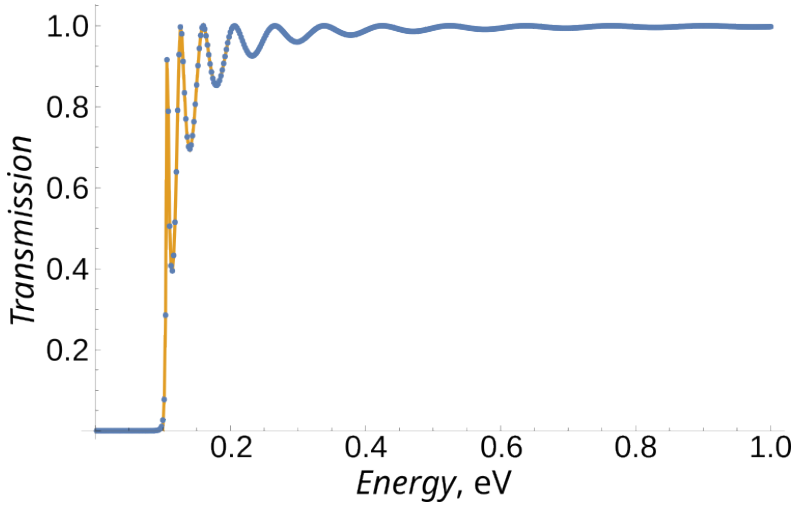


Figure 6.4.14.9: Transmission of a 1D conductor with $V_0 = 100\text{meV}$ and width 30nm. Orange line and blue dots shows the exact analytical answer, Eqs. (6.4.14.1) and (6.4.14.2), and CBR calculations, respectively.

Since transmission of the device is exponentially small at energies below 0.1 eV, the current become nonzero only at $\mu_L > 0.1\text{eV}$ and, after some transient, the IV characteristics becomes again linear with the slope being close to

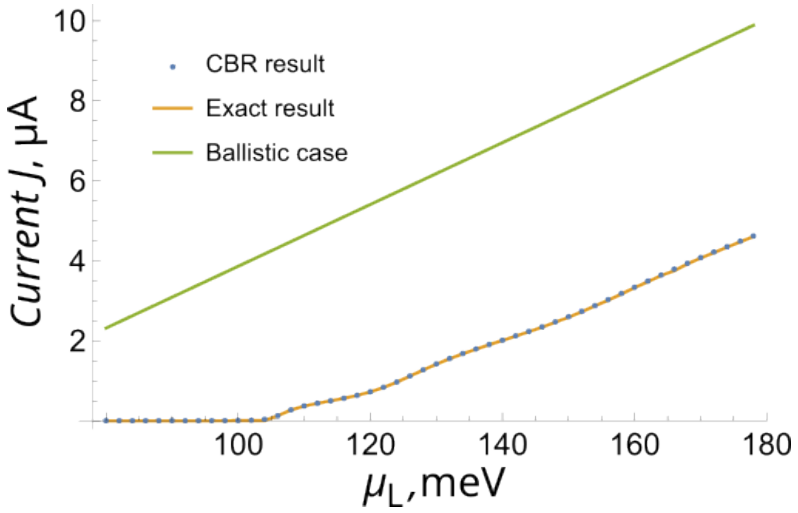


Figure 6.4.14.10: IV-characteristics of a 1D conductor with $V_0 = 100\text{meV}$ and width 30 nm at $\mu_R = 50\text{meV}$. Other parameters are the same as in Figure 6.4.14.8. Orange line and blue dots shows the exact analytical answer [obtained by using Eqs. (6.4.14.1) and (6.4.14.2)], and CBR calculations. Green line exemlifies the ballistic law $J = G_0 V_{\text{dc}}$.

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 6.4.14.11.

The energy profile can be imported into the nextnano GmbH 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 6.4.14.12. The right CBR-lead is attached at $x = 400\text{nm}$.

Numerically calculated energy dependence of the QPC transmission is shown in Figure 6.4.14.13. 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.

The example of the gate-induced QPC is 2D and requires 2D simulations. However, the second equation in (6.4.14.3) still can be used. It suggests that, if temperature and V_{dc} are extremely small, then linear conductance is proportional to transmission: $G_{\text{QPC}} = G_0 \mathcal{T}(\mu)$. Negative values of the chemical potential, μ, of the gated

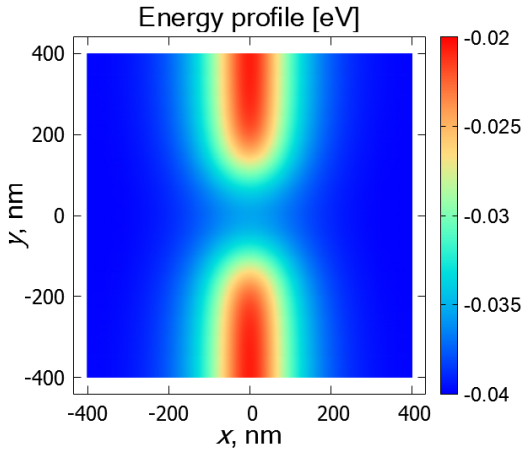


Figure 6.4.14.11: An example of the numerically obtained energy profile for a QPC in the plane of the 2D electron gas. The simulations were done for the 2D electron gas in GaAs at temperature $100\mathrm{mK}$.

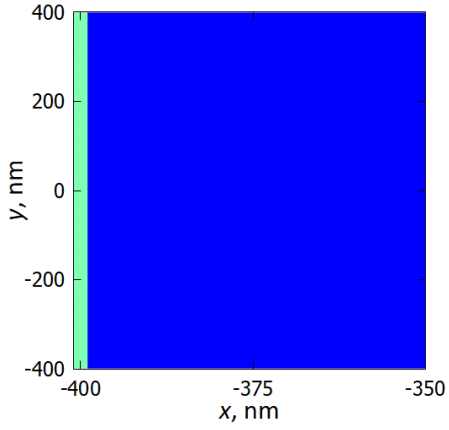


Figure 6.4.14.12: Illustration of how the left CBR-lead (light green region) is attached to the device (blue region). The width of the lead along x-axis is equal to the step of the space discretization. The width of the lead along y-axis has been chosen to be equal to the width of the device.

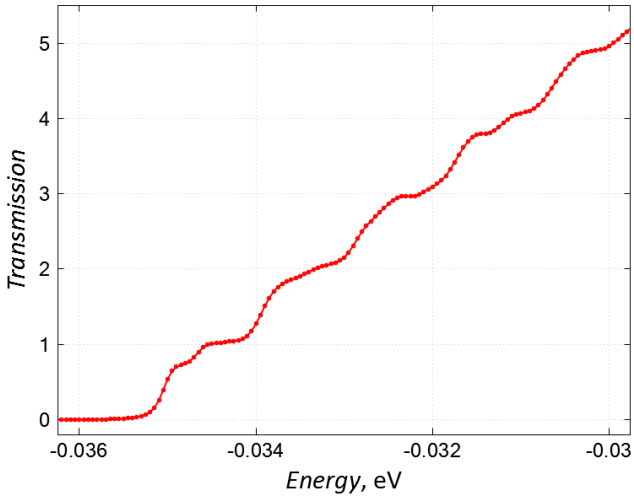


Figure 6.4.14.13: Numerically calculated energy dependence of the transmission via the QPC which is presented in Figure 6.4.14.11. The bottom of the conduction band, E_0, of the gated 2DEG is located at $\simeq -40\mathrm{meV}$. Hence, E_0 is the origin of the energy for this example.

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 GmbH 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 $\leftrightarrow$ 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.

Last update: 17/07/2024

Electron Flying Qubit

Input Files:

- *EPJQT2022_2D_TCW_nnp.in*
- *EPJQT2022_2D_ABI_nnp.in*
- *EPJQT2022_1D_slice_TCW_nnp.in*

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 6.4.14.14;
- Aharonov-Bohm interferometer, ABI - the electronic counterpart of the optical interferometer, see the central region of Figure 6.4.14.15;
- Circuits containing these elements connected in a series, see Figure 6.4.14.15.

Left rectangular regions in Figure 6.4.14.14 and Figure 6.4.14.15 (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 6.4.14.14 and Figure 6.4.14.15 reflect the strength of the electrostatic potential in different parts of the device, ranging from 0 eV (dark blue color) up to $\gg 1$ 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.

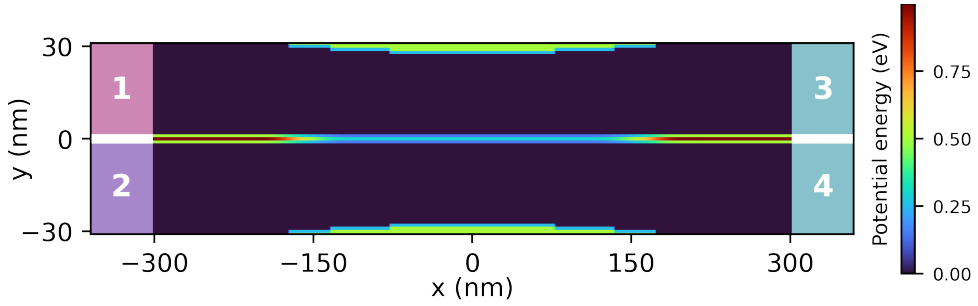


Figure 6.4.14.14: Geometry and potential landscape of TCW connected to four terminals (external leads marked by white numbers). Red and light blue separation regions denote impenetrable (very high with the height V_∞ = 10 eV) and penetrable (tunneling with the height V_T) potential barriers, respectively. Green regions mark those parts of the device where the gate voltages 0.5 eV and V_g are applied.

Let us first discuss transport in TCW. The horizontal line in Figure 6.4.14.14 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 6.4.14.15 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

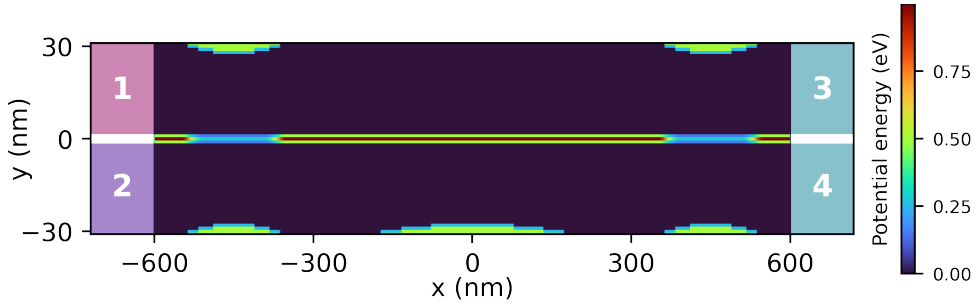


Figure 6.4.14.15: Geometry and potential of a circuit containing two TCWs and one ABI, also connected to four terminals. The additional barrier around x=0 in the lower path yields the electrostatic Aharonov-Bohm effect.

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 6.4.14.16, and the transmission from the lead 1 to the leads 3 and 4, Figure 6.4.14.17 and Figure 6.4.14.18, 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 GmbH simulations of the pLDoS and of the transmission are discussed in detail our recent review (see SpringerOpen or ArXiv) 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.

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.

Acknowledgment

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: nn/nn/nnnn

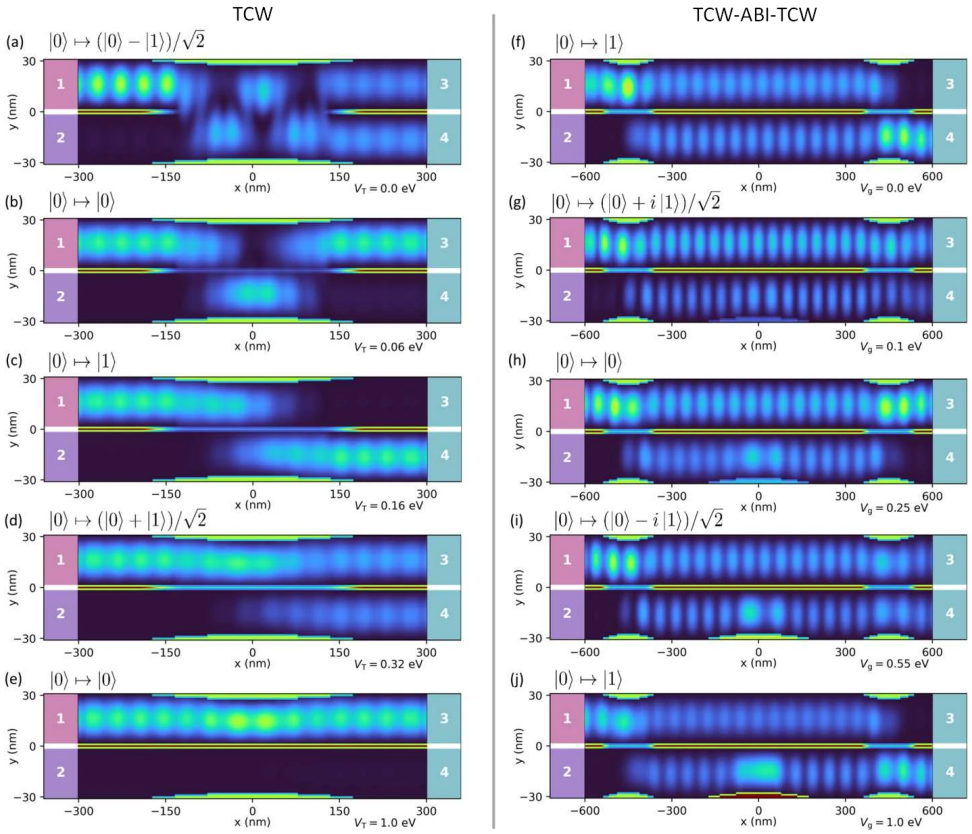


Figure 6.4.14.16: nextnano GmbH simulations of the electron partial local density of states in TCW [panels (a-e)] and the TCW - ABI - TCW [panels (f-j)] nanodevices. Both devices are connected to four terminals (marked by white numbers). The background shows the potential landscape defined by the voltage 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).

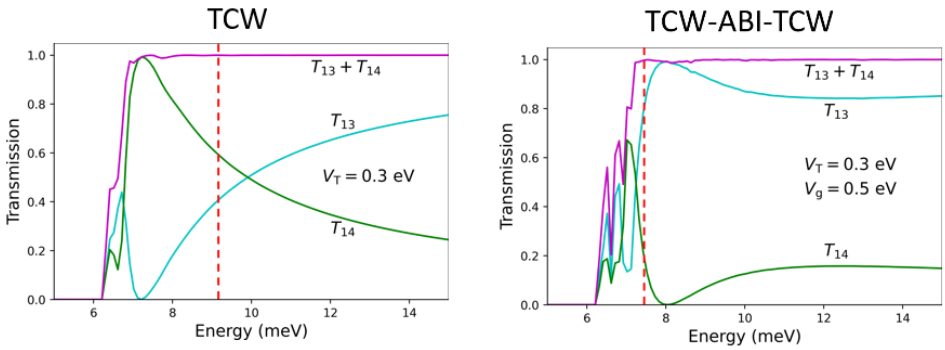


Figure 6.4.14.17: Energy-dependent transmission of the electron from the lead no. 1 into the leads no. 3 (T_{13}) and no. 4 (T_{14}). Red dashed lines mark some electron energies where the reflection is almost absent, $T_{13} + T_{14} \simeq 1$ (E = 9.2 meV for TCW and 7.5 meV for TCW-ABI-TCW).

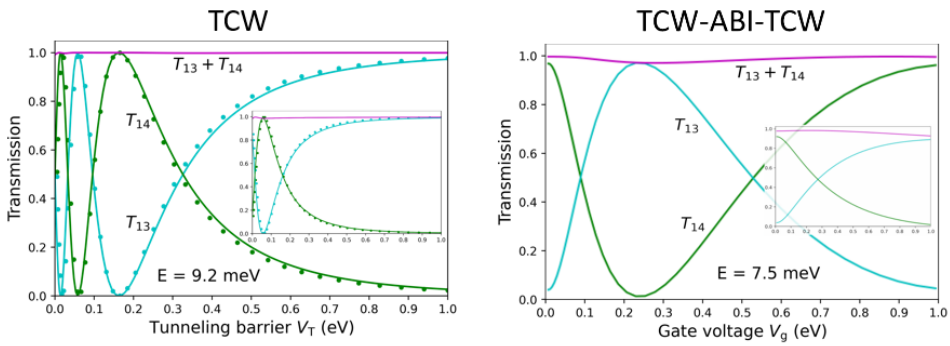


Figure 6.4.14.18: Almost reflectionless transmission of the electron with fixed energy as a function of V_T (TCW) and V_g (TCW-ABI-TCW). Dots in the left panel correspond to the semi-phenomenological theory supplied by the 1D simulation of the spectrum at the center of the device, x = 0. Insets: The same dependence as in the main figures but for devices with half-length, where the accessible number of quantum oscillations is much smaller.

— 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 6.4.14.19 shows the calculated conduction band edge of the device.

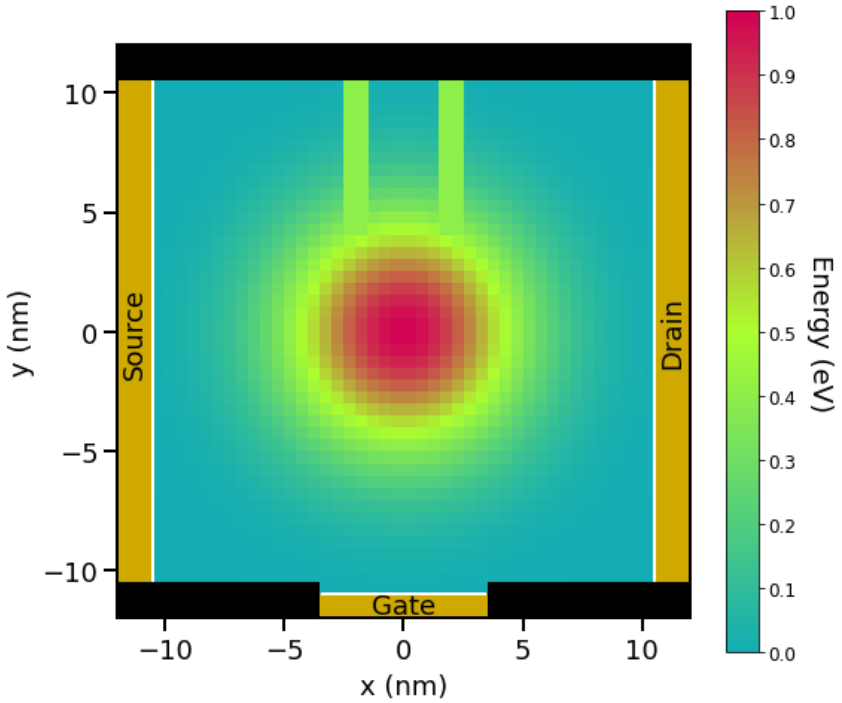


Figure 6.4.14.19: The calculated conduction band edge. The center of the device $((x, y) = (0, 0)\ (\mathrm{nm}))$ is AlAs and the energy is $1.0\ (\mathrm{eV})$. The vicinity of the edges of the device is GaAs and the energy is $0\ (\mathrm{eV})$. The double potential barrier is set so that the energy is equivalent to $0.4\ (\mathrm{eV})$. 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 $2.0\ (\mathrm{eV})$ 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 6.4.14.19

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 $E_{c,0} = 1.0\ (\mathrm{eV})$ so that the maximum height of the Gaussian barrier becomes $1.0\ (\mathrm{eV})$ at the center of the device. In this tutorial, $a = 5\ (\mathrm{nm})$.

In the upper part of the device, a thin tunneling double barrier is present and the height is $0.4\ (\mathrm{eV})$.

These conduction band profiles are achieved by adjusting the `database{ }` as below.

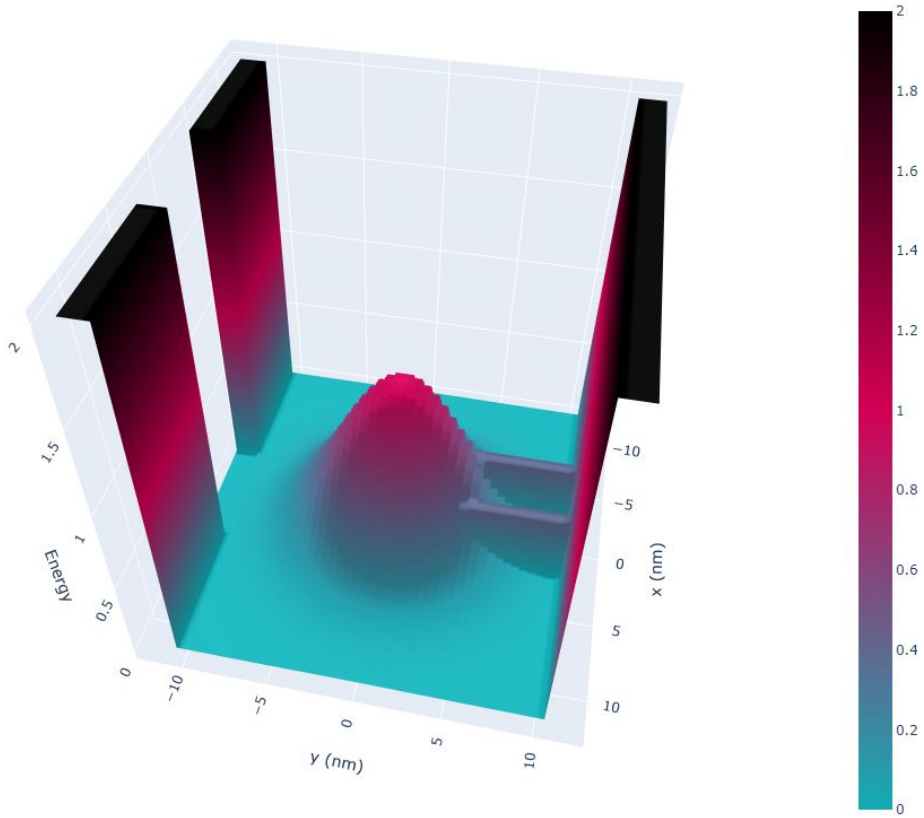


Figure 6.4.14.20: Potential landscape

```
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
            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␣
```

(continues on next page)

(continued from previous page)

```
↪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
        }
    }

    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 6.4.14.19.

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{
```

(continues on next page)

(continued from previous page)

```
        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 }
    }

    # 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++* and *nextnano*^{3}. 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 6.4.14.21 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.

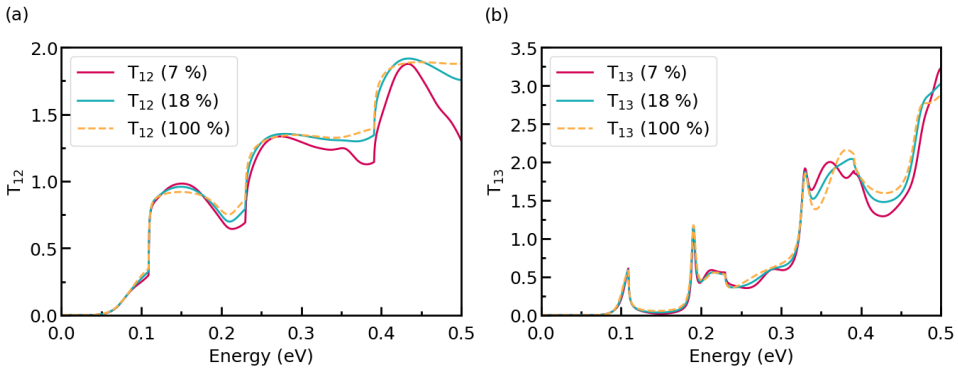


Figure 6.4.14.21: The transmission coefficient $T(E)$ of a 2D sample with 3 leads. T_{12} in (a), whereas T_{13} in (b). *bias_00000\CBR\transmission_device_Gamma.dat*

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:

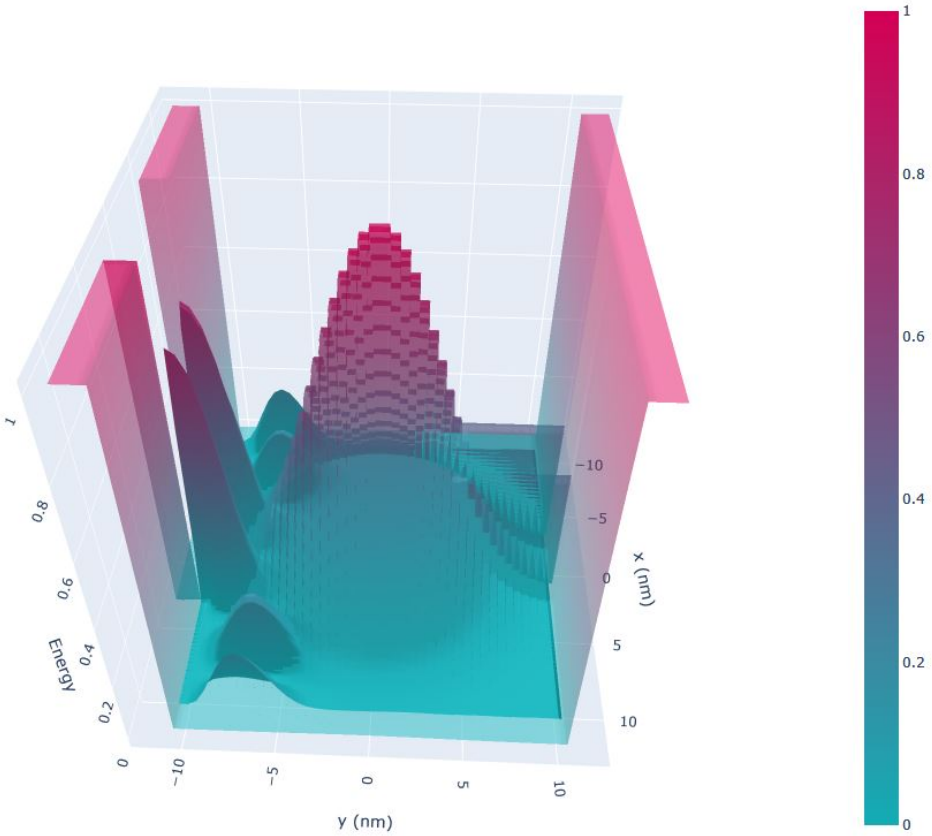


Figure 6.4.14.22: The 16th eigenstate

  - the 26th eigenstate: $0.177\ (\mathrm{eV})$ (delocalized)
  - the 29th eigenstate: $0.193\ (\mathrm{eV})$ (more localized)
- 2nd resonance:
  - the 56th eigenstate: $0.311\ (\mathrm{eV})$ (delocalized)
  - the 59th eigenstate: $0.328\ (\mathrm{eV})$ (more localized)
  - the 61th eigenstate: $0.336\ (\mathrm{eV})$ (delocalized)
  - the 63th eigenstate: $0.347\ (\mathrm{eV})$ (delocalized)
  - the 64th eigenstate: $0.352\ (\mathrm{eV})$ (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 $190\ (\mathrm{meV})$.

Note that the square of the wave function is rescaled so that you can see the shape clearly.

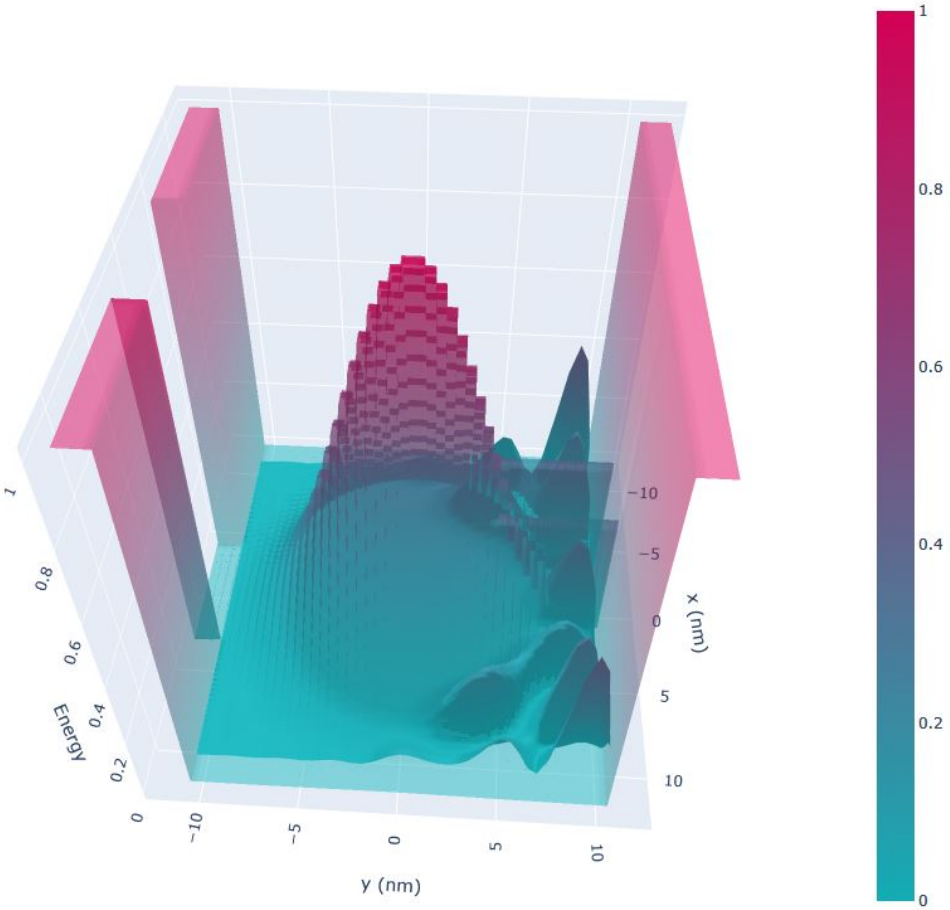


Figure 6.4.14.23: The 26th eigenstate

Lead modes

Figure 6.4.14.24 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').

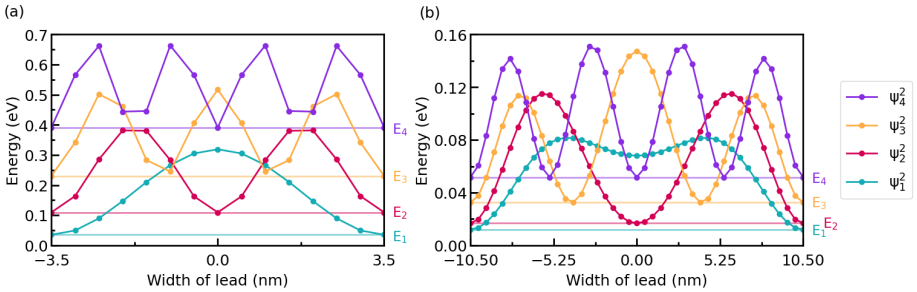


Figure 6.4.14.24: The lead modes of lead 2 ('gate') are shown in (a), whereas the lead modes of lead 1, 3 ('source', 'drain') are shown in (b). *bias_00000\Quantum\probabilities_shift_lead_X_Gamma.dat*

Notes

Attention: Here the difference in the way of setting up boundary conditions in *nextnano++* and *nextnano*^{3} will be discussed.

Last update: nn/nn/nnnn

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. The corresponding tutorial for *nextnano*^{3} is *here*.

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 (Note: this is slightly different in *nextnano*[3] input). 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␣
```

(continues on next page)

(continued from previous page)

```
↪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 6.4.14.25 shows the calculated transmission coefficient as a function of energy. The result of *nextnano*[3] 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 GmbH 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)

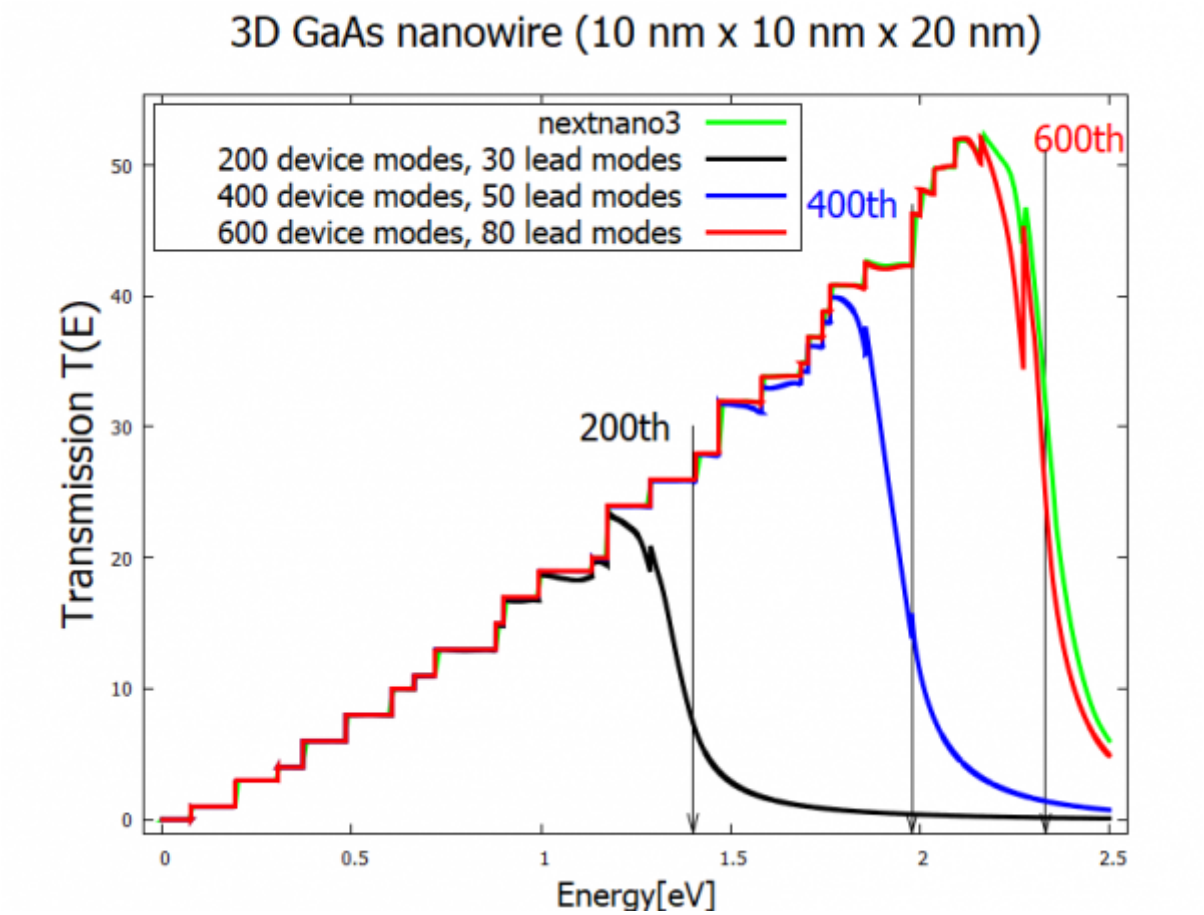


Figure 6.4.14.25: Transmission coefficient of a GaAs 3D nanowire simulated with three different CBR parameters. The *nextnano*[3] result is shown for reference. Arrows indicate the cutoff energies, namely the eigenenergy of the highest device eigenmode considered in each simulation.

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.

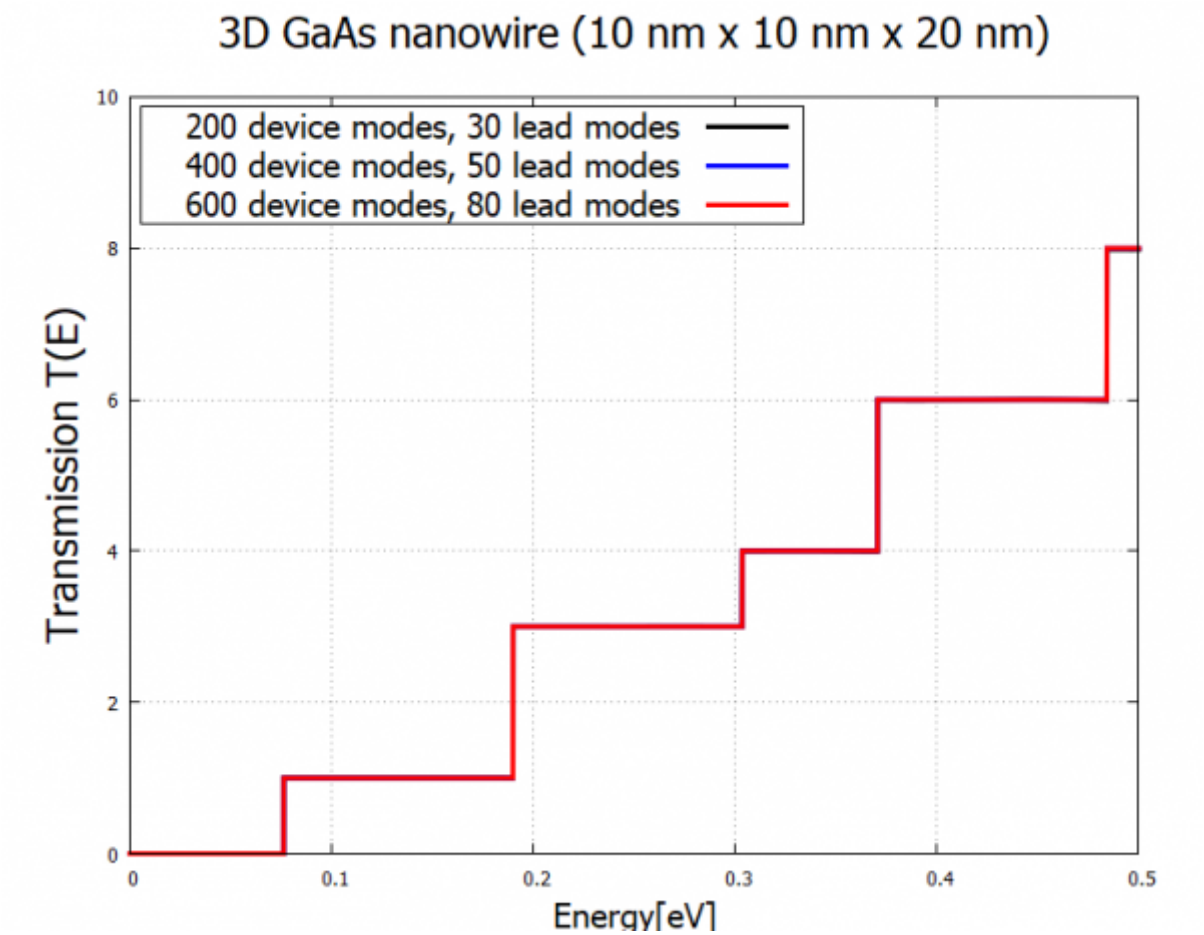


Figure 6.4.14.26: Zoom into the first few steps of T(E). The transmission increases by integer at the eigenenergies of the lead.

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.

Last update: nn/nn/nnnn

Input | Template | Template (Beta) | Simulation | Output

wf_energy_spectrum_lead_1_Gamma_0000.dat

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

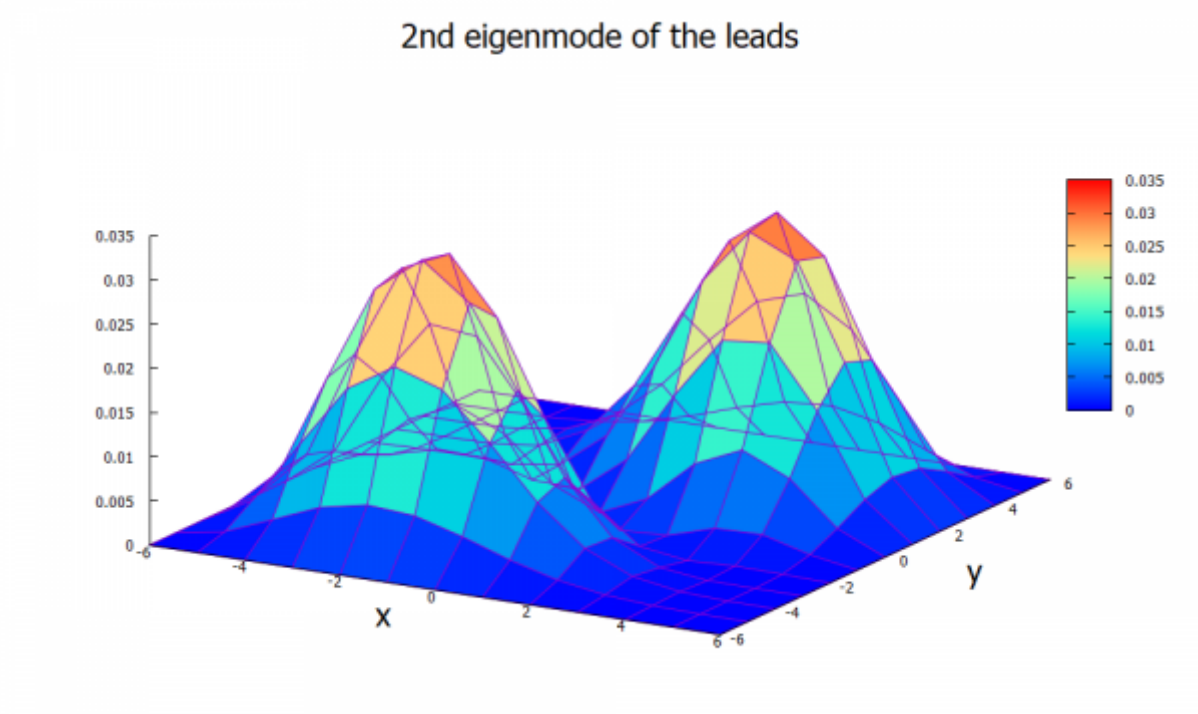


Figure 6.4.14.27: The probability distribution $|\psi(x,y)|^2$ of the 2nd lead mode.

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 6.4.14.28 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 6.4.14.29 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]*

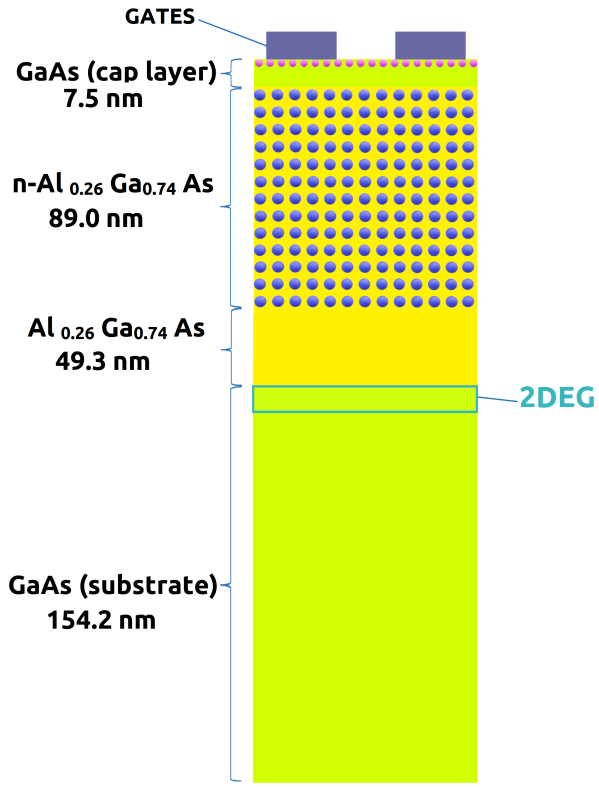


Figure 6.4.14.28: Schematics of a side view of the simulated device

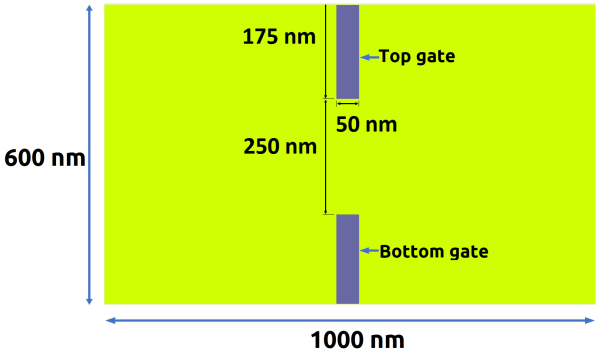


Figure 6.4.14.29: Top view of the gates deposited on the top of the simulations

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.

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 6.4.14.30.

At this point it is convenient to verify the band edges of both leads, one of them plotted in the Figure 6.4.14.30. 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 $2e^2/h$

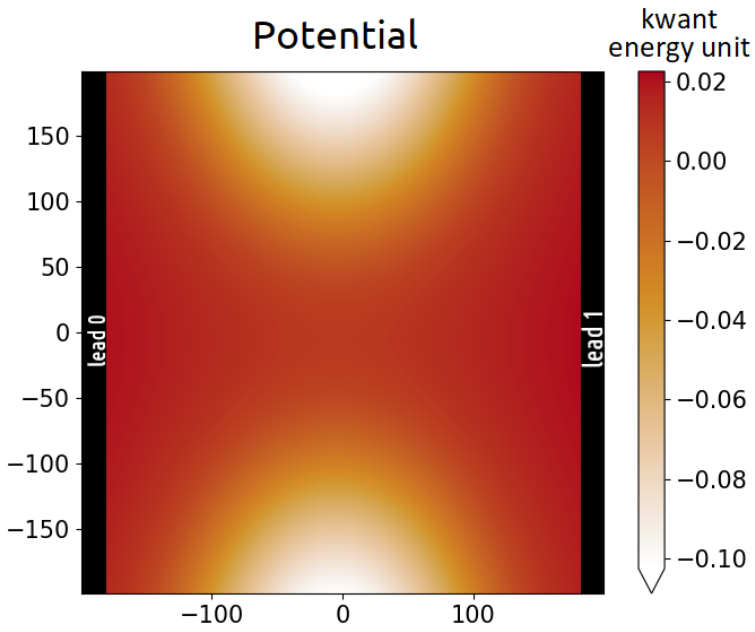


Figure 6.4.14.30: Imported conduction band when a bias of -1.11V is applied to the top gate.

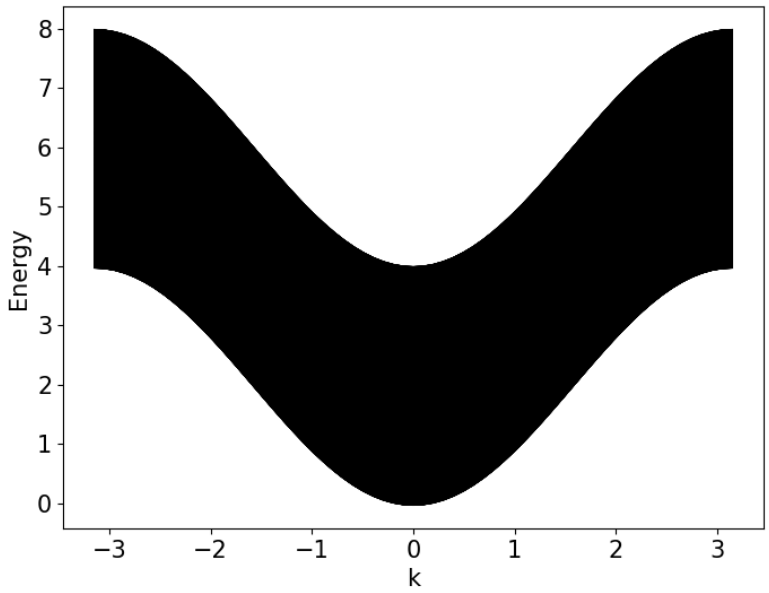


Figure 6.4.14.31: Band structure of the lead 0 for top gate voltage equal to -1.11 V.

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 GmbH 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.

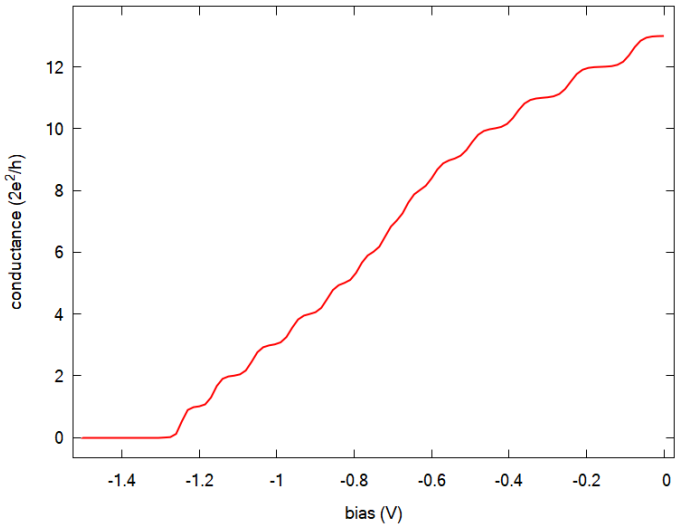


Figure 6.4.14.32: Conductance between lead 0 to lead 1 as function of the bias applied to the top gate

The Figure 6.4.14.32 presents the channel conductance computed for each value of V_{gate}. The steps in the curve show the expected quantization for this device.

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 6.4.14.33 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

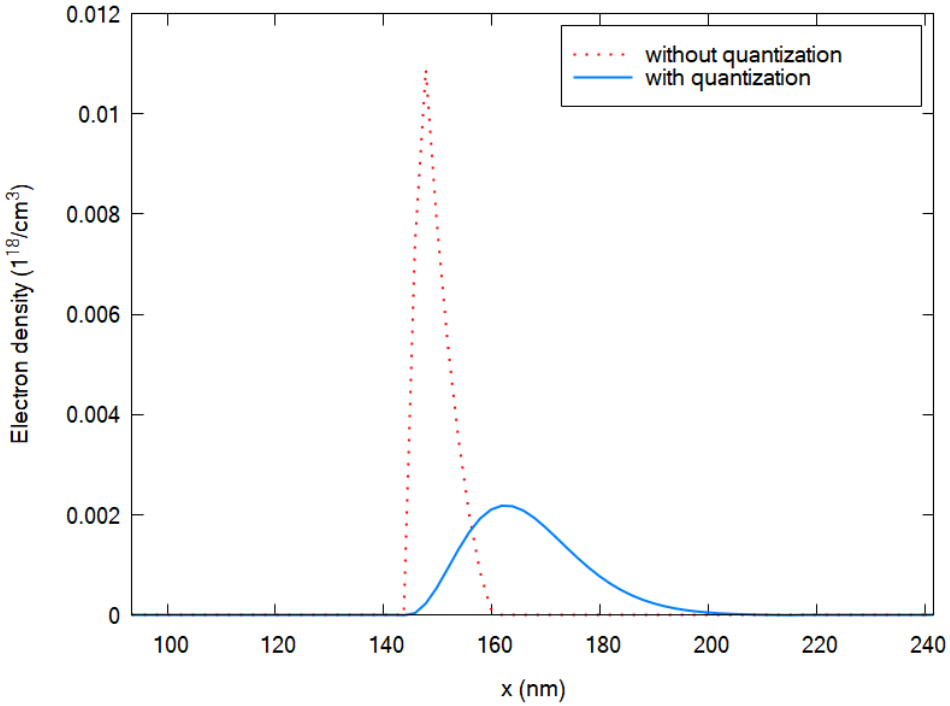


Figure 6.4.14.33: Density of electrons in the growth direction at the center of the device ($y = 0$ nm and $z = 0$) for $V_{gate} = -1.17V$ for semiclassical computation (without quantization) and for self-consistent Schrödinger-Poisson calculations (with quantization)

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 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.

Acknowledgment

This tutorial is a result 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. The tutorial contains part of results from a collaboration of Institut Néel (CNRS), CEA-IRIG and nextnano GmbH Lab in France, and nextnano GmbH in Germany.

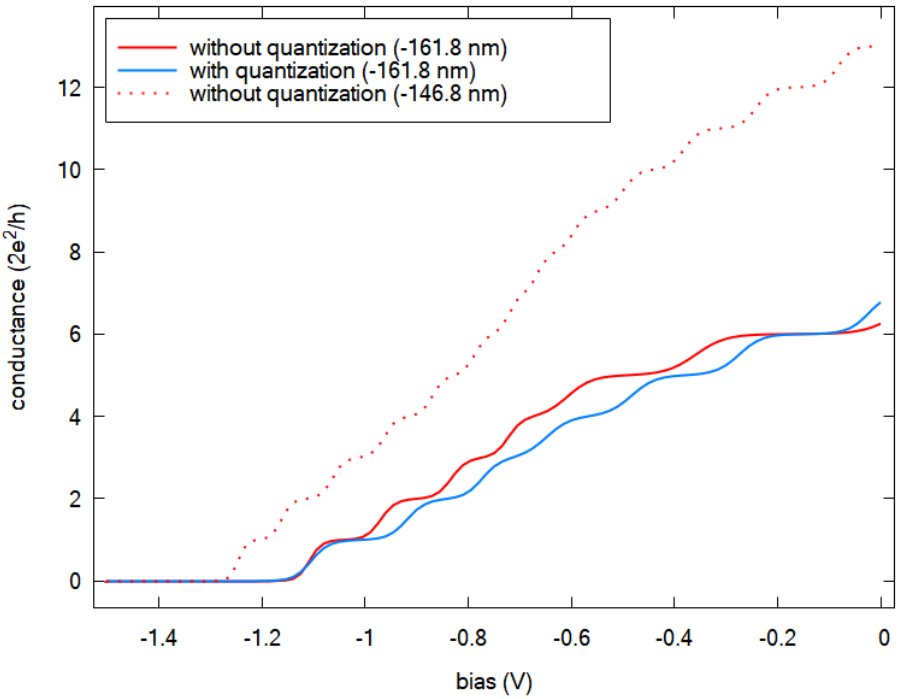


Figure 6.4.14.34: Conductance between lead 0 to lead 1 as function of the bias applied to the top gate at the plane z = 161.8 nm in the 2DEG region with and without quantization along the growth direction (in solid lines). In dotted lines the conductance without quantization is shown at the depth where the electron density is higher in the 2DEG (146.8 nm below the surface, as shown in Figure 6.4.14.32)

Last update: nn/nn/nnnn

6.4.15 Transistors

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 6.4.15.1.

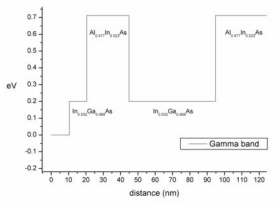


Figure 6.4.15.1: Calculated conduction band edge profile.

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})

We obtain two eigenstates and their corresponding wave functions inside the HEMT channel which leads to a two-dimensional electron gas (2DEG), see Figure 6.4.15.2. The electron density is plotted in blue.

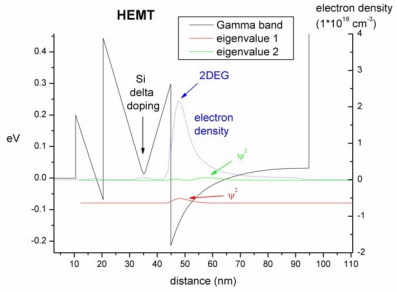


Figure 6.4.15.2: Calculated conduction band edge profile and probability densities.

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: nn/nn/nnnn

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 Schottky barrier tutorial.

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 6.4.15.3 shows the results of the input file *1DJogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig4Fig1Fig7_nnp.in*.

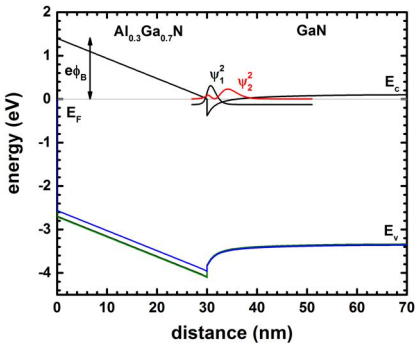


Figure 6.4.15.3: Calculated conduction and three valence band edges with the probability densities of the two lowest subbands of a 30 nm $Al_{0.3}Ga_{0.7}N$ / 40 nm GaN heterostructure.

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).

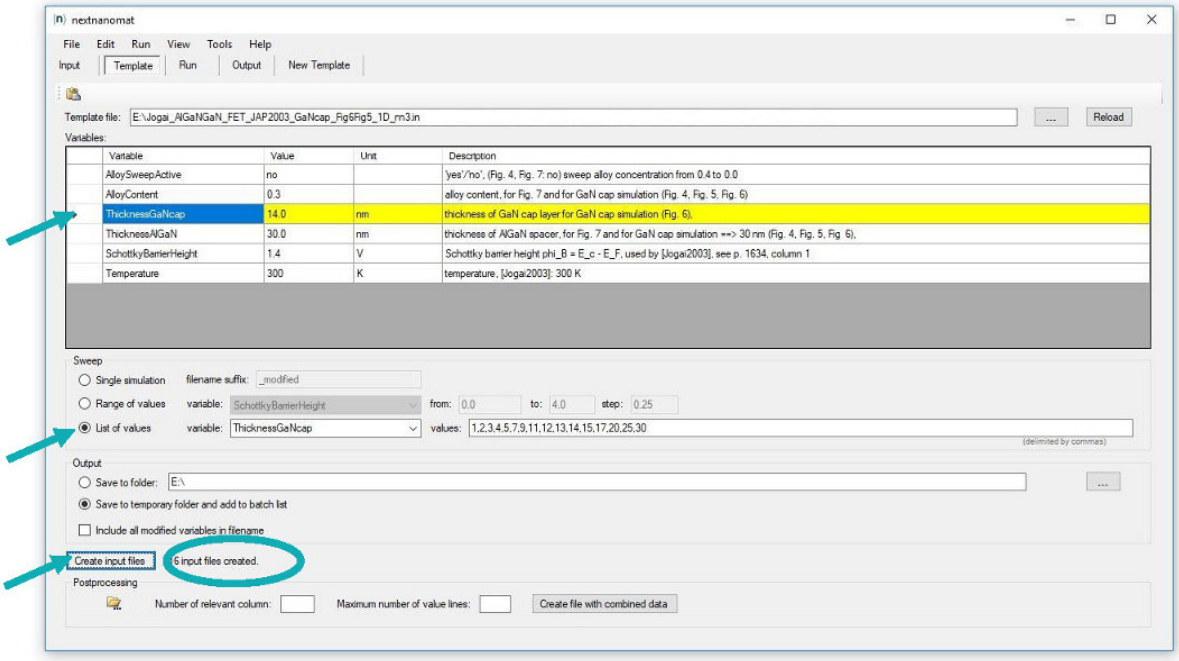
Within the *nextnano++* input file, we can perform a sweep over the alloy concentration very conveniently:

```
$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.

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*.

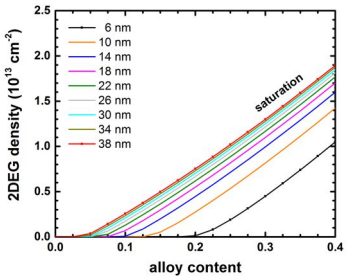


Figure 6.4.15.4: Calculated 2DEG density for different layer widths of $Al_xGa_{1-x}N$ as a function of alloy content x.

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 6.4.15.5 where we show exactly the same data.

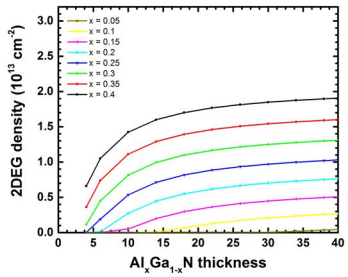


Figure 6.4.15.5: Calculated 2DEG density for different alloy contents x of $Al_xGa_{1-x}N$ as a function of layer widths.

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])

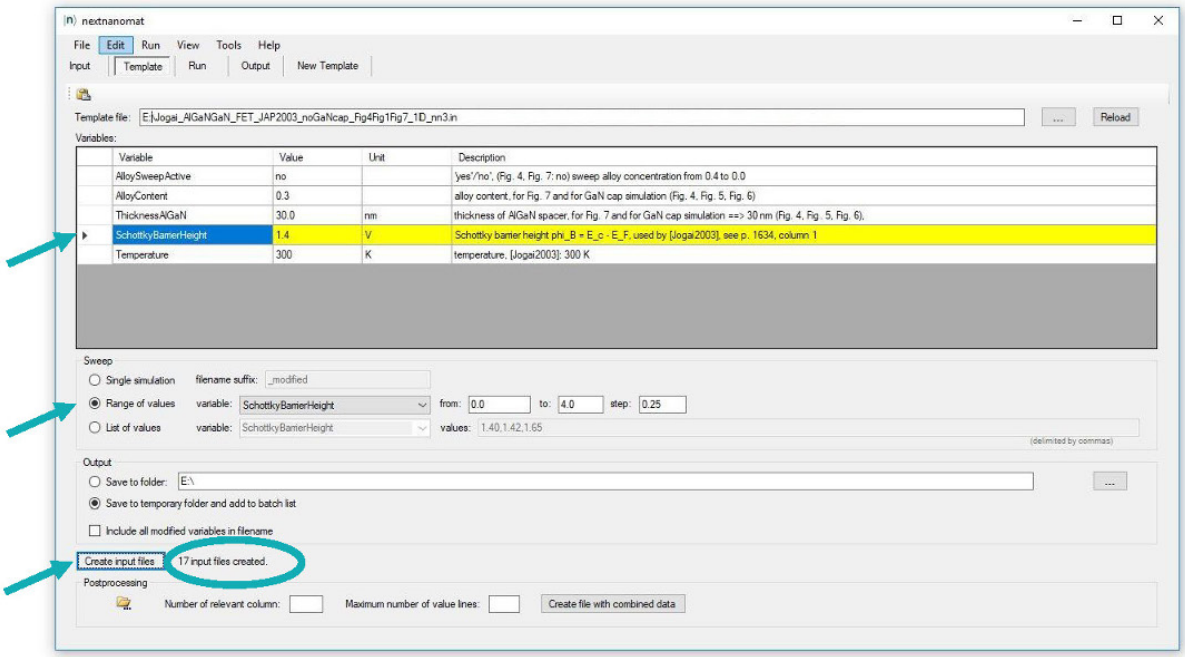
Using te input file *Jogai_AlGaNGaN_FET_JAP2003_noGaNcap_Fig4Fig1Fig7_1D_nnp.in*, we vary the Schottky barrier height phi_B and calculate for each value the 2DEG density:

```
$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 6.4.15.6 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]*.



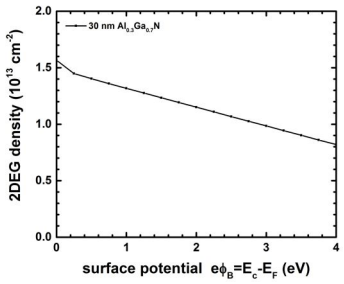


Figure 6.4.15.6: Calculated 2DEG density as a function of surface potential $e\phi_B$.

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 6.4.15.7 compares the band edges of capped and uncapped HEMT structure.

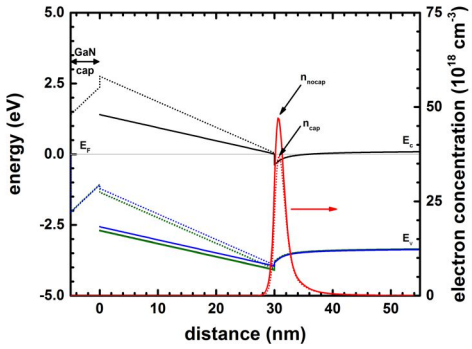


Figure 6.4.15.7: Calculated conduction and valence band edges of a $Al_{0.3}Ga_{0.7}N$/ GaN FET **with** (solid lines) and **without** (dotted lines) a 5 nm GaN cap layer.

Figure 6.4.15.8 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.

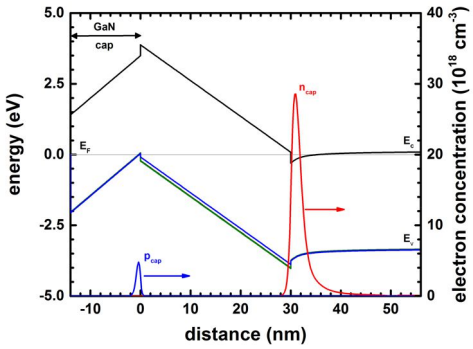


Figure 6.4.15.8: Calculated conduction and valence band edges of a $Al_{0.3}Ga_{0.7}N$/ GaN FET with 14 nm GaN cap.

- 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.

```
$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)
```

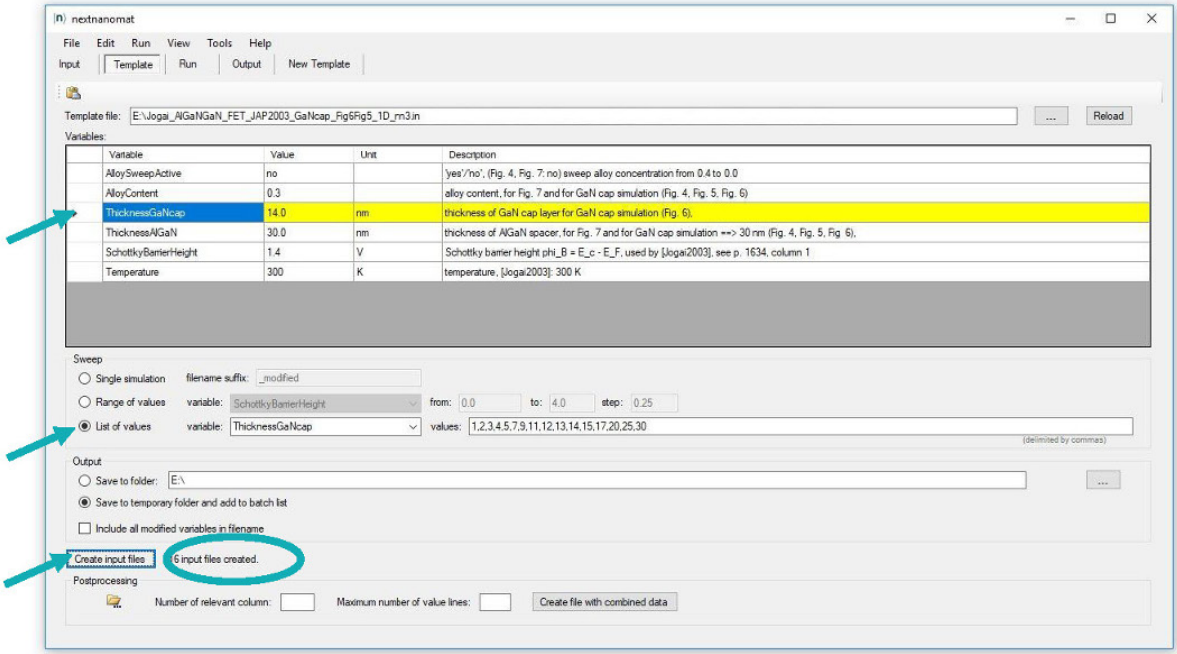


Figure 6.4.15.9 shows the 2DEG density vs. GaN cap layer thickness for a 30 nm $Al_{0.3}Ga_{0.7}N$ barrier. Beyond a GaN cap layer thickness of ~13 nm (12 nm *[Jogai2003]*) the 2DEG density saturates.

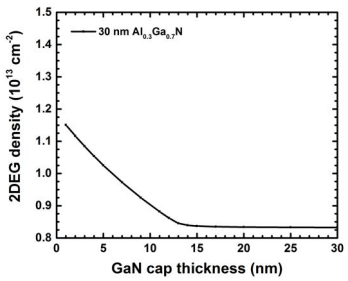


Figure 6.4.15.9: Calculated 2DEG density as a function of GaN cap thickness.

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: nn/nn/nnnn

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 6.4.15.10

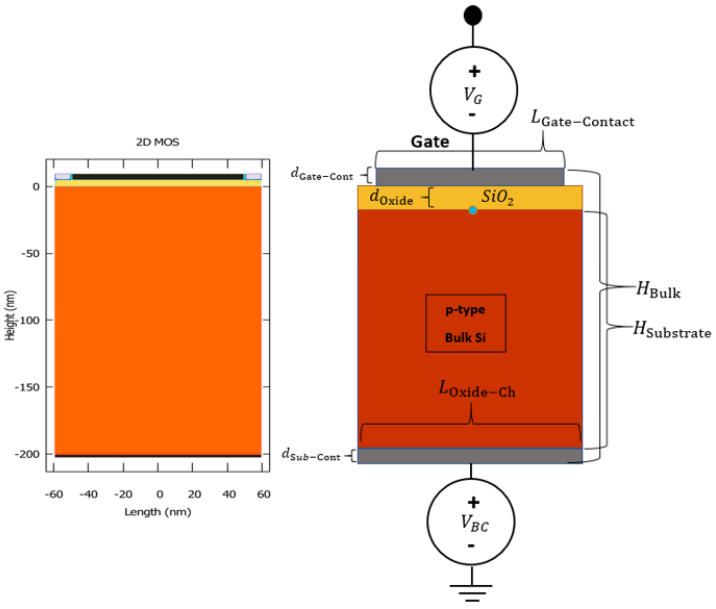


Figure 6.4.15.10: The geometry of the 2D MOS design, and its equivalent geometry from the output file *regions.vtr* (colored differently in post-processing). The blue circle indicates the position of the origin of our (x, y) coordinate system.

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 $1 \times 10^{16}\text{cm}^{-3}$, and the oxide layer which consists of SiO:sub:*2* has a thickness of $d_{\text{ox}} = 5\text{nm}$. The length of the channel is $L_{\text{G}} = 100\text{nm}$, the substrate has a height of $H_{\text{Substrate}} = 200\text{nm}$. 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,

$$C_{\mathrm{tot}} = \frac{C_{\mathrm{ox}} C_{\mathrm{D}}}{C_{\mathrm{ox}} + C_{\mathrm{D}}}. \tag{6.4.15.1}$$

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 (6.4.15.2),

$$Q_{\mathrm{D}} = qW_{\mathrm{D}}N_{\mathrm{Sub}}, W_{\mathrm{D}} = \sqrt{\frac{\varepsilon_{\mathrm{s}}^2}{C_{\mathrm{ox}}^2} + \frac{2\varepsilon_{\mathrm{s}}V}{qN_{\mathrm{Sub}}}} - \frac{\varepsilon_{\mathrm{s}}}{C_{\mathrm{ox}}}, \tag{6.4.15.2}$$

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 (6.4.15.3) , i.e. the capacitance at the voltage, which creates the flat-band condition in the MOS band structure,

$$C_{\mathrm{FB}}(\psi_{\mathrm{s}} = 0) = \frac{\varepsilon_{\mathrm{s}}\varepsilon_{\mathrm{ox}}}{\varepsilon_{\mathrm{s}}d_{\mathrm{ox}} + \varepsilon_{\mathrm{ox}}L_{\mathrm{D}}}, L_{\mathrm{D}} = \sqrt{\frac{k_{\mathrm{B}}T\varepsilon_{\mathrm{s}}}{q^2N_{\mathrm{Sub}}}}, \tag{6.4.15.3}$$

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 6.4.15.12. Note that the output of the simulations, however, is only the total charge (per cm in 2D), as shown in figure Figure 6.4.15.11, which needs to be (first multiplied with the elementary charge $|q|$, and then) derived with respect to the bias voltage:

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 (6.4.15.3).

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 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 6.4.15.13:

The bias voltage that results in a band structure in the figure Figure 6.4.15.13, is called the flat-band voltage V_{FB}. This voltage is related to, and is a part of the definition of the threshold voltage,

$$V_{\mathrm{Th}} = V_{\mathrm{FB}} + 2\psi_{\mathrm{B}} + \frac{\sqrt{4\varepsilon_{\mathrm{Si}}qN_{\mathrm{Sub}}\psi_{\mathrm{B}}}}{C_{\mathrm{ox}}}. \tag{6.4.15.4}$$

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

$$\psi_{\mathrm{s}}(\mathrm{stronginversion}) \approx \frac{2k_{\mathrm{B}}T}{q}\ln\left(\frac{N_{\mathrm{Sub}}}{n_i}\right) \tag{6.4.15.5}$$

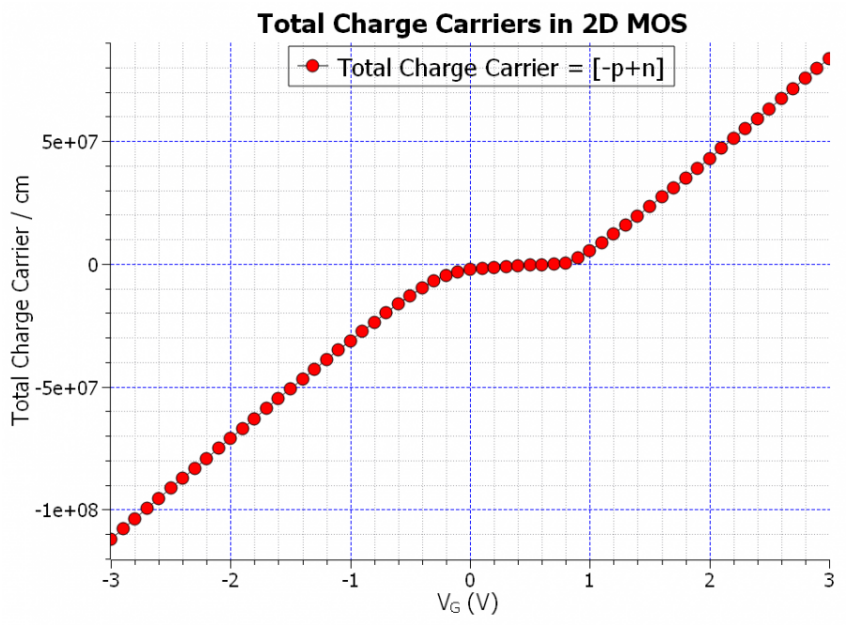


Figure 6.4.15.11: The total charge carriers per cm of the MOS, integrated in the substrate, vs. the applied gate bias.

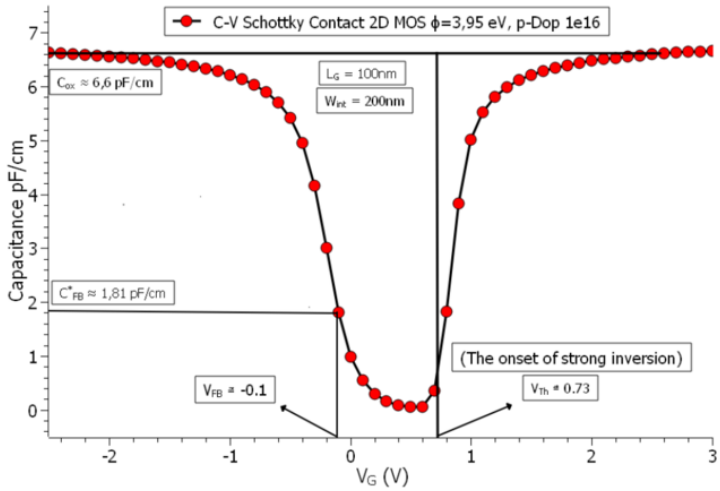


Figure 6.4.15.12: The C-V characteristics of the 2D MOS with $N_{\mathrm{Sub}} = 10^{16}\mathrm{cm}^{-3}$ doping concentration in the p-doped silicon substrate, channel length of 100 nm, a Schottky barrier of $\phi_{\mathrm{B}} = 3.95\mathrm{eV}$, and a charge integration region equal to the entire substrate. (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 $-0.1\mathrm{V}$).

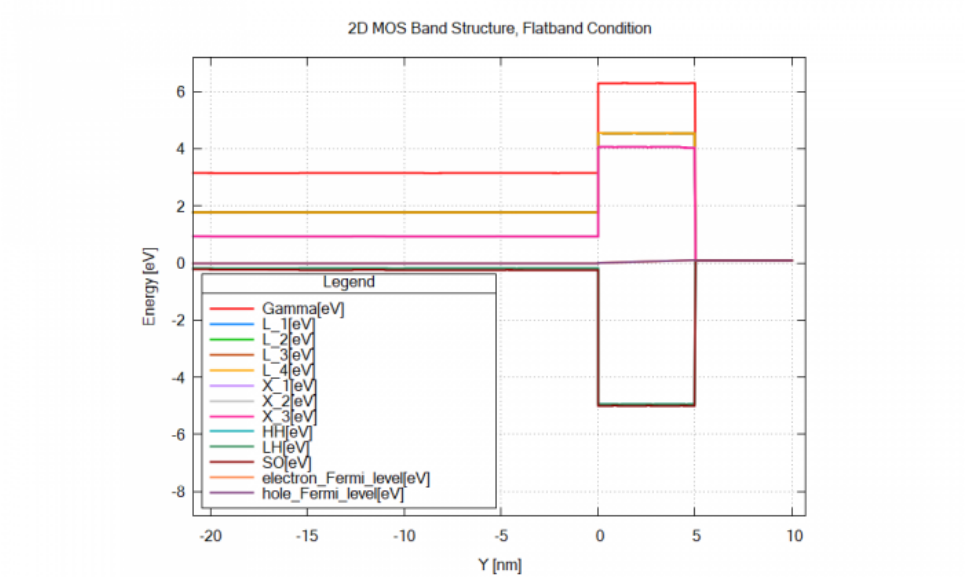


Figure 6.4.15.13: The alignment of conduction and valence band edges with respect to the Fermi levels of the 2D MOS under the flat-band condition along a one-dimensional slice along the y direction. (The lowest conduction band edge is labeled with X.

Calculating this expression for our system, the surface potential amounts to $\approx 0.713\mathrm{V}$, while the expression $\sqrt{4\varepsilon_{\mathrm{Si}} q N_{\mathrm{Sub}} \psi_{\mathrm{B}}}/C_{\mathrm{ox}} \approx 0.073\mathrm{V}$, 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 6.4.15.14 illustrates:

As it could be seen in the above figure Figure 6.4.15.14, 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'_{min}, which are the bottom limits of the C-V curve (C'_{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 6.4.15.15, illustrates the effect of changing the charge integration region on the flatband capacitance C_{FB}:

And figure Figure 6.4.15.16 shows the C-V curve of the MOS capacitor for a charge integration region of $W_{\mathrm{int}} = 300\mathrm{nm}$:

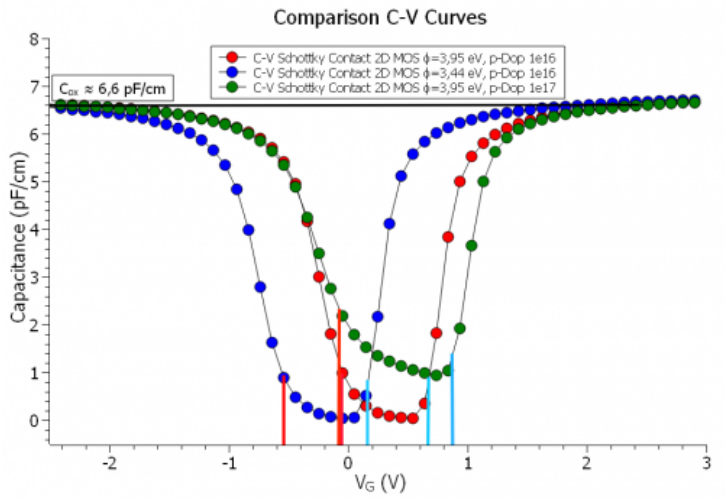


Figure 6.4.15.14: The comparison of the C-V characteristics of the 2D MOS for varying Schottky barrier and the substrate doping concentration, and their effects on the threshold voltage (vertical blue lines), and the flatband voltage (vertical red lines)

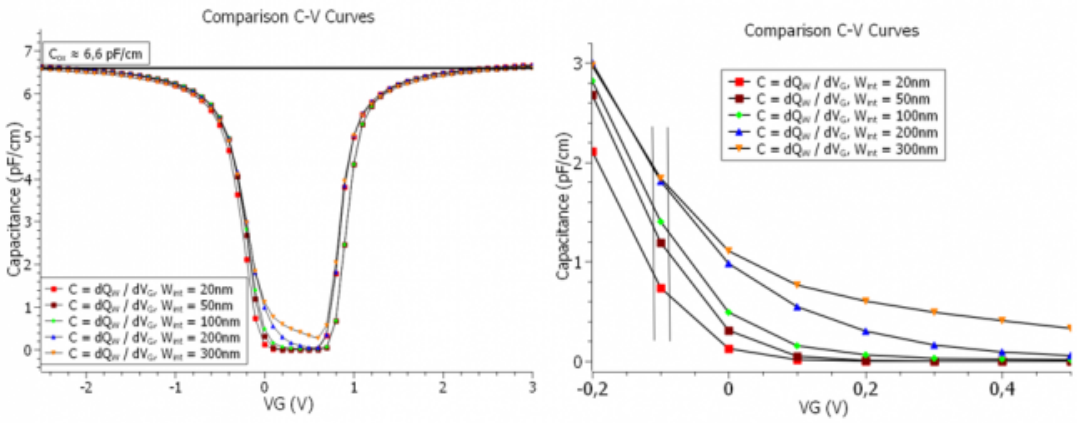


Figure 6.4.15.15: The comparison of the C-V characteristics of the 2D MOS for varying the width of the charge integration region.

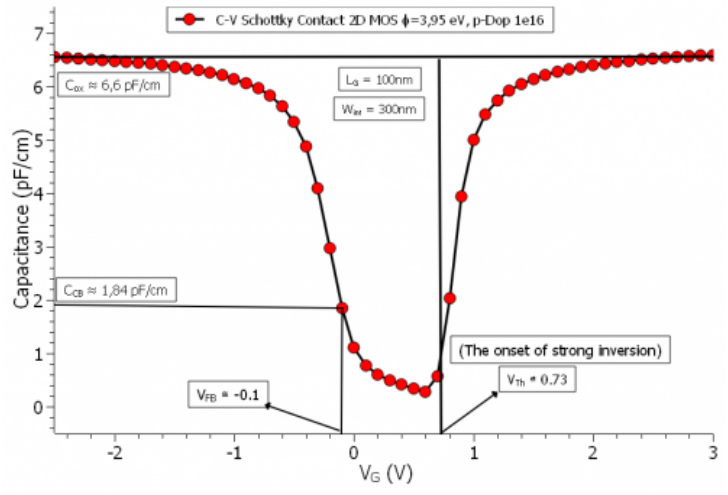


Figure 6.4.15.16: The comparison of the C-V characteristics of the 2D MOS for varying the width of the charge integration region.

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 (6.4.15.2), 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:

$$W_{\mathrm{D},max} \approx \sqrt{\frac{2\epsilon_s \psi_s(\text{strong inv.})}{qN_\mathrm{A}}} \approx \sqrt{\frac{2\epsilon_s k_\mathrm{B} T \ln(N_\mathrm{A}/n_i)}{q^2 N_\mathrm{A}}} \tag{6.4.15.6}$$

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 *<nn++_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 6.4.15.17 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 6.4.15.18

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
```

(continues on next page)

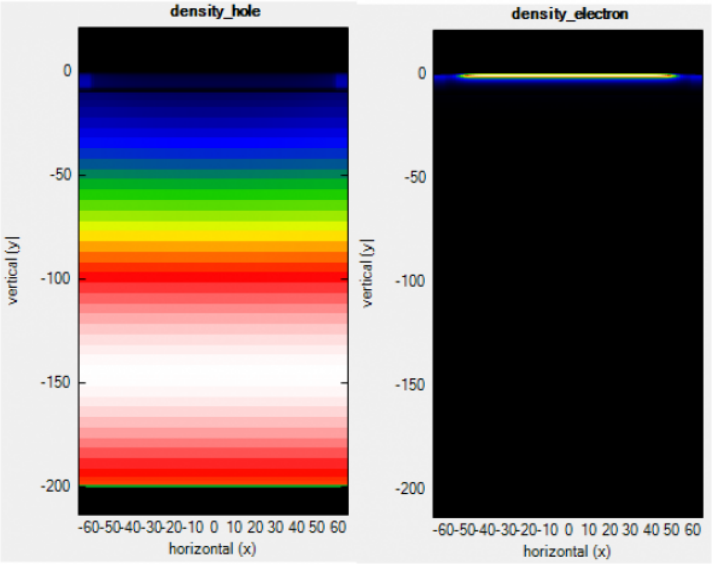


Figure 6.4.15.17: The spatial distribution of charge carriers (electrons) in the inversion layer during inversion, compared to the ones (holes) in the depletion region during depletion.

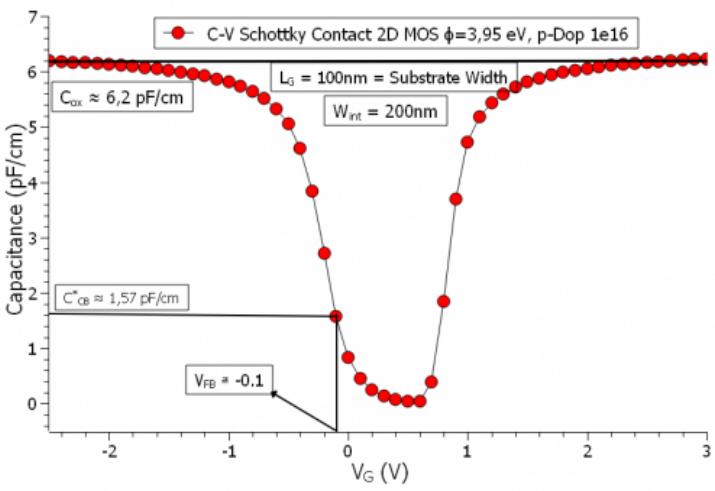


Figure 6.4.15.18: The C-V curve of the quasi 1-D Simulations of the MOS (this is when we set the length of the oxide and the channel-length equal in a 2D simulation and set the boundary condition in x-direction as periodic).

(continued from previous page)

```
  }
}
```

Using the post-processing in the template, one can then construct a curve, which should look like the one shown in figure Figure 6.4.15.19

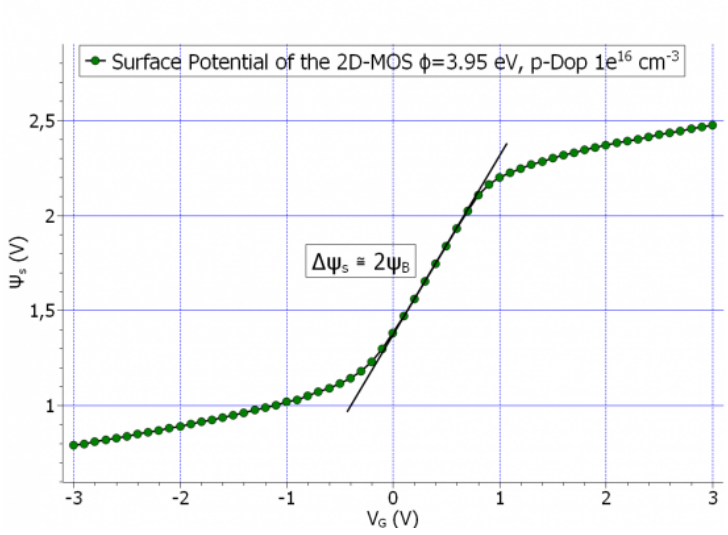


Figure 6.4.15.19: The surface potential, at the semiconductor-oxide interface ψ_{s}, as a function of the gate.voltage V_{G}

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:

$d_{\text{oxide}} = t_{\text{ox}} = 5\text{nm}$, $L_{\text{Ch}} = 100\text{nm}$, $N^{+} = 10^{20}\text{cm}^{-3}$, $P = 10^{16}\text{cm}^{-3}$.

The overall geometry of the simulated N-Ch MOSFET in this tutorial is illustrated in the following figure Figure 6.4.15.20:

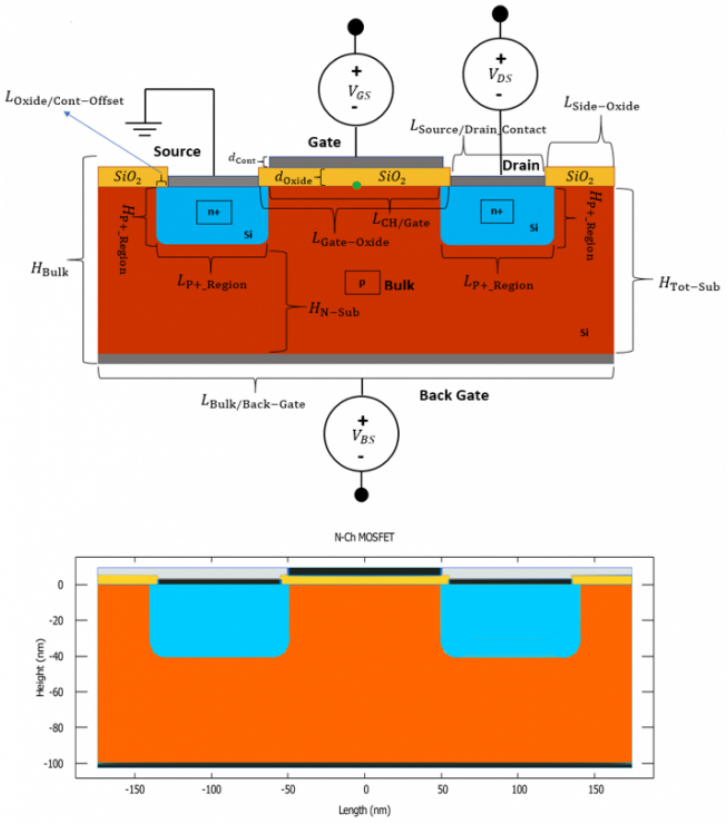


Figure 6.4.15.20: The geometry of the N-Ch MOSFET design, and its corresponding geometry from the output file user_index.vtr. The individual regions can also be found in the output file regions.vtr.

The drain-source current of the MOSFET is given by equation (6.4.15.7)

$$I_{\text{DS}} = \frac{W}{L}\mu_n^{\text{eff}} C_{\text{ox}} \left\{ (V_{\text{GS}} - V_{\text{Th}})V_{\text{DS}} - \left[\frac{1}{2} + \frac{\sqrt{4\epsilon_{\text{Si}} q N_{\text{Sub}} \psi_{\text{B}}}}{C_{\text{ox}}} \right] V_{\text{DS}}^2 \right\} \tag{6.4.15.7}$$

where the threshold voltage V_{Th} is the same threshold voltage for the MOS as defined in equation (6.4.15.4). For the limit of $V_{\text{DS}} \ll (V_{\text{GS}} - V_{\text{Th}})$ equation (6.4.15.7) reduces to:

$$I_{\text{DS}} = \frac{W}{L}\mu_{\text{n}}^{\text{eff}} C_{\text{ox}} (V_{\text{GS}} - V_{\text{Th}} - \frac{V_{\text{DS}}}{2}) V_{\text{DS}} \tag{6.4.15.8}$$

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 6.4.15.21 on a linear scale,

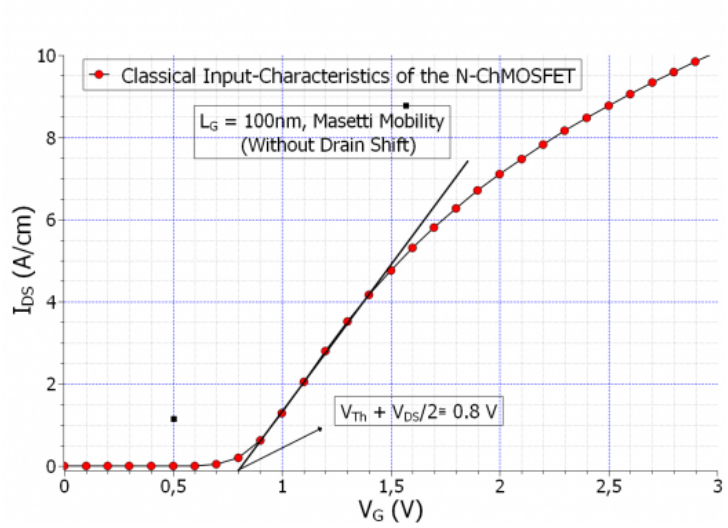


Figure 6.4.15.21: The input characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the position of the threshold voltage V_{Th}.

and in figure Figure 6.4.15.22, on a logarithmic scale:

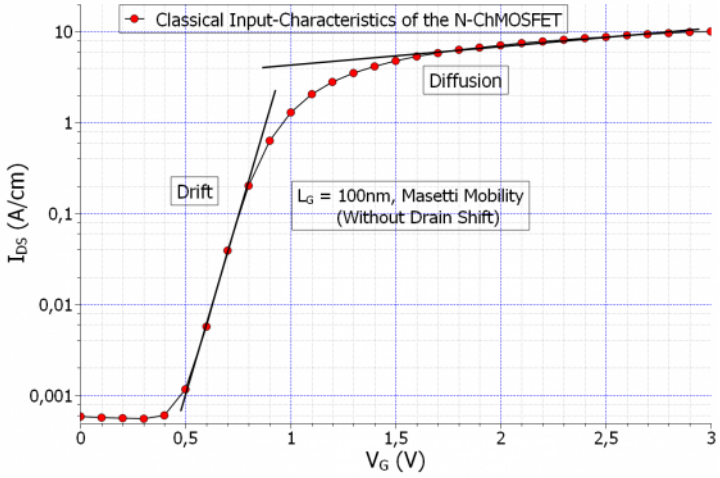


Figure 6.4.15.22: The input characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the drift and diffusion current regions on the logarithmic scale.

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 $V_{\mathrm{Th}} + \frac{V_{\mathrm{D}}}{2}$. Having set the V_{DS}, to 0.2V, for the calculation of the input characteristics, the value is very well expected to be $\approx 0.8\mathrm{V}$, since the threshold voltage V_{Th} was calculated to be $\approx 0.7\mathrm{V}$. However we also used a small backgate bias $V_{\mathrm{BS}} = -0.1\mathrm{V}$ in the above 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 6.4.15.23

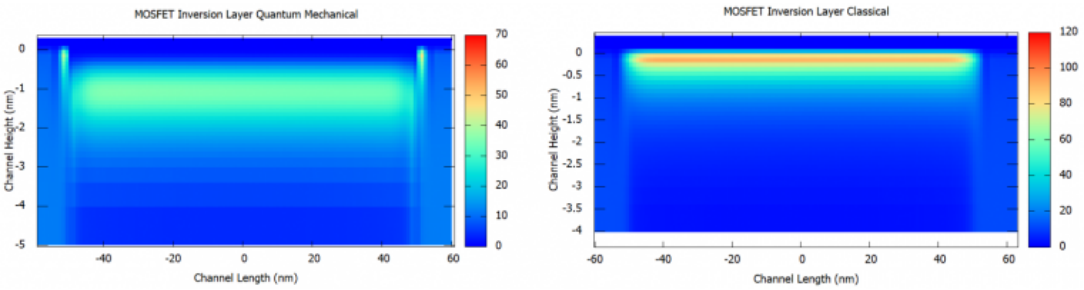


Figure 6.4.15.23: The comparison of the charge inversion layer of the N-Ch MOSFET calculated classically (right), and quantum mechanically (left) at $V_{\mathrm{GS}} > V_{\mathrm{Th}}$ and $V_{\mathrm{DS}} = 0.2\mathrm{V}$.

The following set of curves in figure Figure 6.4.15.24 are the comparison of the input characteristics calculated classically and quantum mechanically, with and without *quantum decomposition* method:

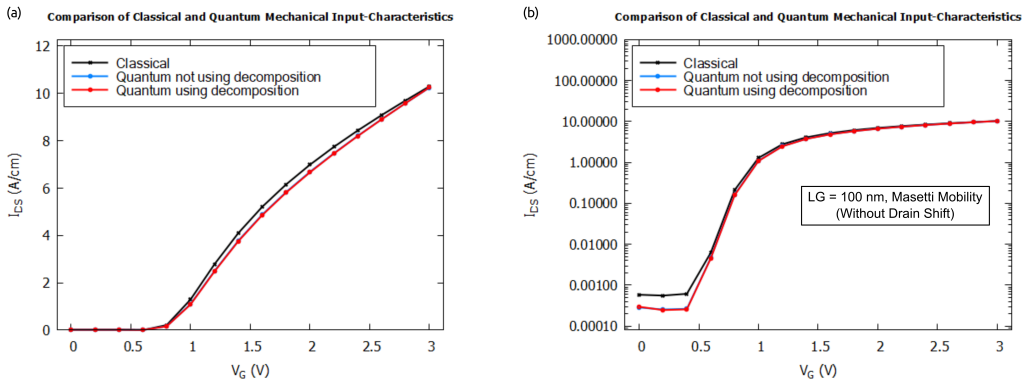


Figure 6.4.15.24: The comparison of the input characteristics of the MOSFET calculated classically and quantum mechanically wit (a) linear and (b) logarithmic scales.

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 6.4.15.25:

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 6.4.15.26:

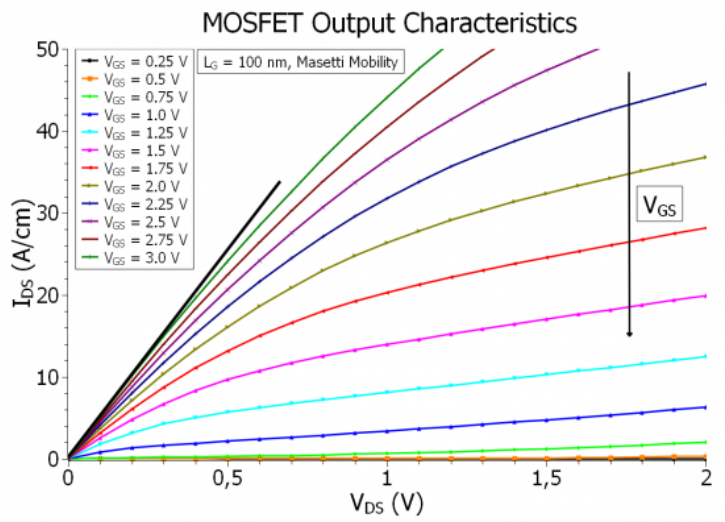


Figure 6.4.15.25: The output characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the linear and the saturation regions of the output characteristics.

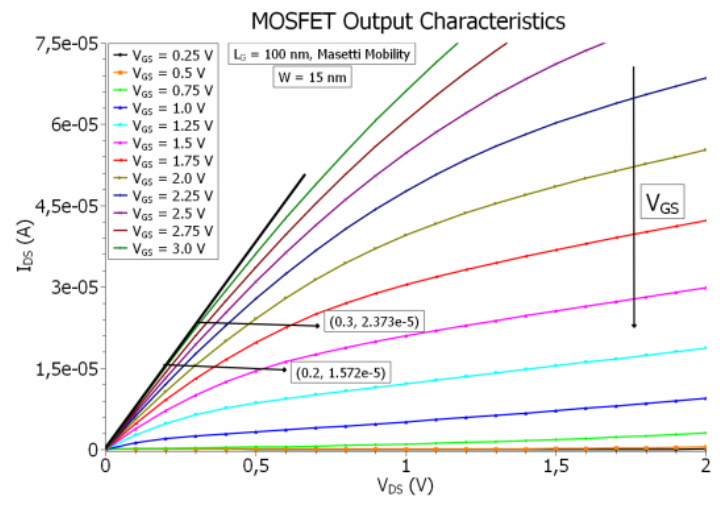


Figure 6.4.15.26: The output characteristics of the N-Ch MOSFET calculated classically with Masetti mobility, showing the linear and the saturation regions of the output characteristics for a width of $15\mathrm{nm}$.

From the readings on the curve we can estimate the specific channel resistivity,

$$\frac{1}{R_{\text{specific}}} = \frac{L}{W}\frac{I_{\text{DS}}}{V_{\text{DS}}} \rightarrow R_{\text{specific}} = 1.8\text{k}\Omega.$$

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 (6.4.15.9):

$$V_{\text{DS,sat}} = \frac{V_{\text{GS}} - V_{\text{Th}}}{M}, M = 1 + \frac{K}{2\sqrt{\psi_{\text{B}}}}, K = \sqrt{\varepsilon_s q N_{\text{A}}}/C_{\text{ox}} \tag{6.4.15.9}$$

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 $V_{\text{DS,sat}}$, for each V_{GS} as defined in equation (6.4.15.10):

$$I_{\text{DS,sat}} = \frac{W}{2ML}\mu_n^{\text{eff}} C_{\text{ox}}(V_{\text{GS}} - V_{\text{Th}})^2 = \frac{WM}{2L}\mu_n^{\text{eff}} C_{\text{ox}} V_{\text{DS,sat}}^2 \tag{6.4.15.10}$$

and plotting this current over the output characteristics, the curve crosses each I_{DS}, exactly at the corresponding $V_{\text{DS,sat}}$ for that output current, as shown in figure Figure 6.4.15.27

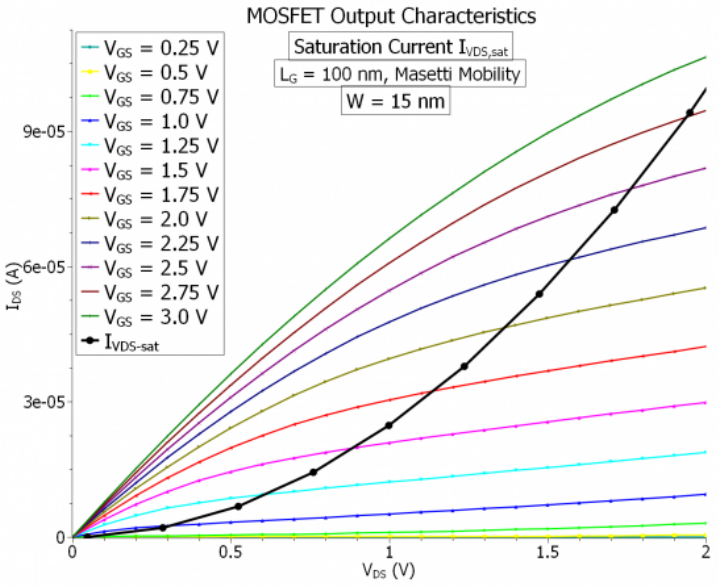


Figure 6.4.15.27: The output characteristics of the N-Ch MOSFET calculated classically with Masetti mobility for a width of 15nm, and the saturation current $I_{\text{DS,sat}}$ plot.

If we take the effective mobility to be *field-independent* (which is the case in our simulations), the above $I_{\text{DS,sat}}$ curve could be fitted with $I_{\text{DS,sat}} = A \cdot V_{\text{DS,sat}}^2$ 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 $V_{\text{DS,sat}}^2$ voltages), because, as we can see, the output charateristic curves do not really saturate after drain source voltage reaches $V_{\text{DS,sat}}$. 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 $V_{\text{DS,sat}}$.

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 $C_{\text{ox}} \approx 6.6\text{mF}/\text{m}^2$, the effective mobility of the electrons is then estimated to be

$$\mu_n^{\text{eff}} \approx 525 \frac{\text{cm}^2}{\text{V} \cdot \text{s}},$$

The calculated bulk mobility from the simulations is given to be $\approx 933\text{cm}^2/\text{Vs}$ in the p-doped substrate, and $\approx 567\text{cm}^2/\text{Vs}$ 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 6.4.15.28 shows the tranconductance curve and its maximum value:

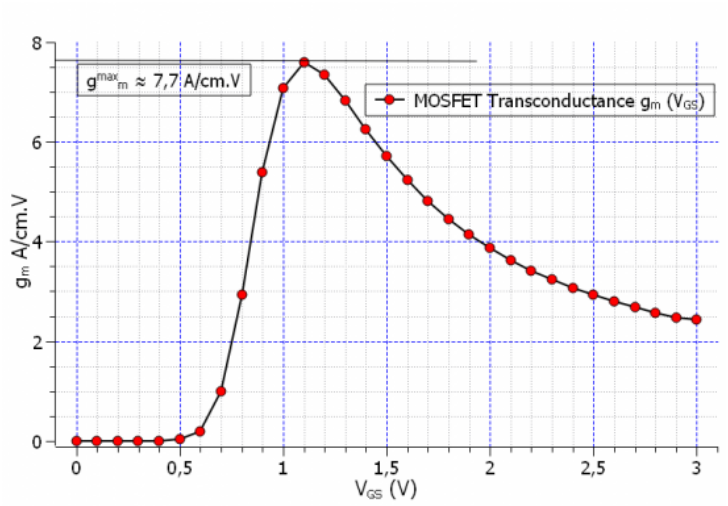


Figure 6.4.15.28: The transconductance of the MOSFET as a a derivative of the source-drain current I_{DS} with respect to the gate voltage V_{GS}.

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 (6.4.15.11), since we now know the value of the effective mobility:

$$g_m = \left.\frac{\partial I_{\mathrm{DS}}}{\partial V_{\mathrm{GS}}}\right|_{V_{\mathrm{DS}}=\mathrm{const.}} = \frac{W}{L}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}} V_{\mathrm{DS}} \tag{6.4.15.11}$$

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 (6.4.15.12):

$$g_{\mathrm{D}} = \left.\frac{\partial I_{\mathrm{DS}}}{\partial V_{\mathrm{DS}}}\right|_{V_{\mathrm{GS}}=\mathrm{const.}} = \frac{W}{L}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}}(V_{\mathrm{GS}} - V_{\mathrm{Th}}) \tag{6.4.15.12}$$

which is in turn a function of the gate voltage V_{GS}. Figure Figure 6.4.15.29 illustrates this conductance for a set of gate voltages:

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 (6.4.15.13) with respect to V_{GS}:

$$g_m = \left.\frac{\partial I_{\mathrm{DS}}}{\partial V_{\mathrm{GS}}}\right|_{V_{\mathrm{DS}} \geq V_{\mathrm{DS,sat}}} = \frac{W}{ML}\mu_n^{\mathrm{eff}} C_{\mathrm{ox}}(V_{\mathrm{GS}} - V_{\mathrm{Th}}) \tag{6.4.15.13}$$

which would be straight line with respect to V_{DS}, and V_{GS}.

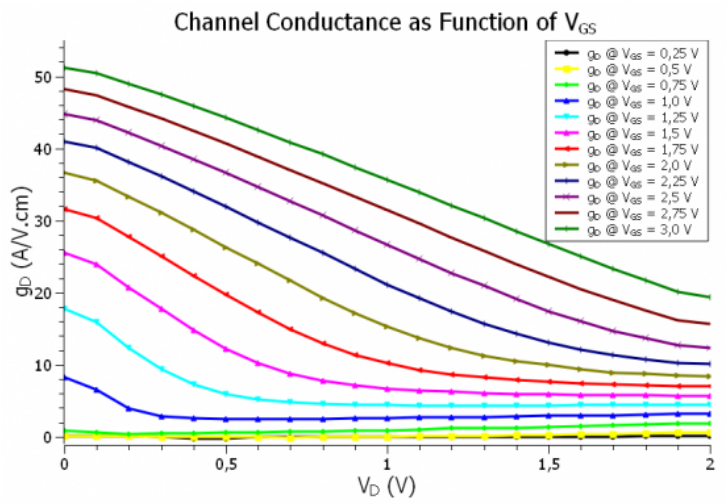


Figure 6.4.15.29: The channel conductance of the MOSFET as a derivative of the source-drain current I_{DS} with respect to the source-drain voltage V_{DS}.

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 6.4.15.30 illustrates the effect of different mobility models on the input characteristics of the MOSFET:

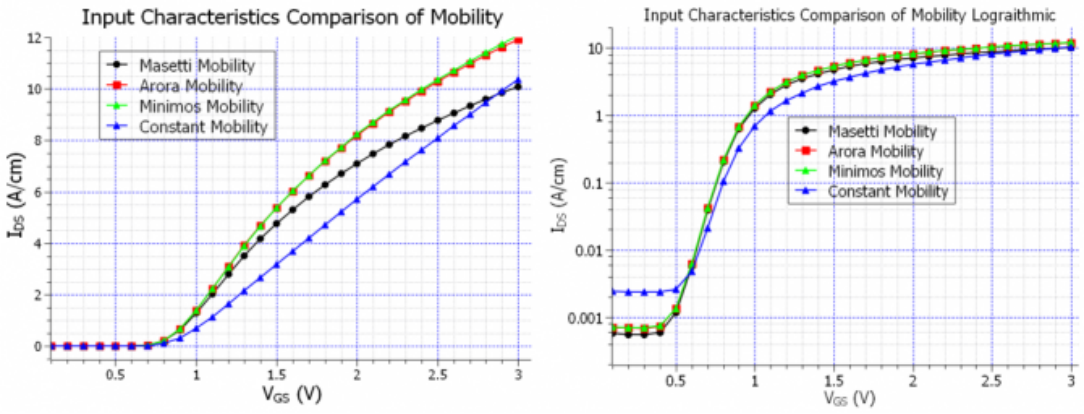


Figure 6.4.15.30: The input characteristics of the MOSFET calculated classically with different mobility models, in normal and logarithmic scales.

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 $\approx$ 12, while it is $\approx$ 10 for the Minimos and Arora models. Obviously, this ratio is 1 for the constant mobility model.

The following figure Figure 6.4.15.31 shows the output characteristics calculated with the constant mobility model set at $\mu_0 \approx 460\mathrm{cm}^2/\mathrm{Vs}$:

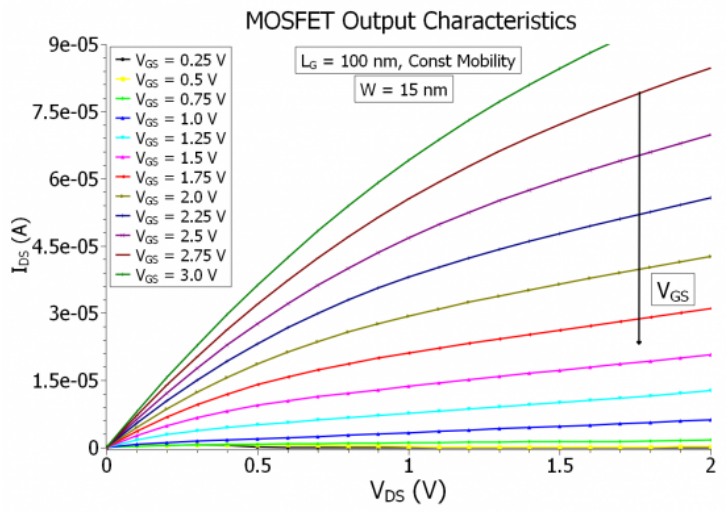


Figure 6.4.15.31: The output characteristics of the MOSFET calculated classically with the constant mobility model, taking the width W to be $15\mathrm{nm}$.

We can now compare this to the Masetti mobility as the example of doping dependent models. Figure Figure 6.4.15.32 shows the comparison for a selection of the V_{GS} values:

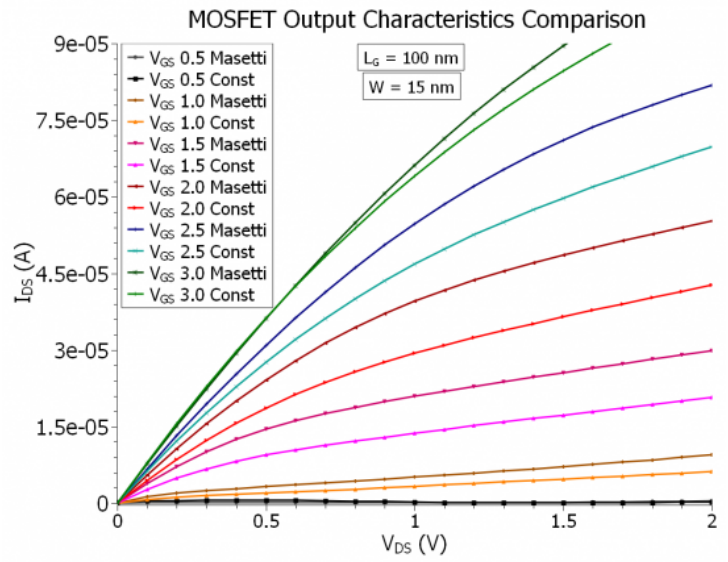


Figure 6.4.15.32: The comparison of the output characteristics of the MOSFET calculated classically with constant mobility and Masetti models, for a selection of gate voltages, and the width $W = 15\mathrm{nm}$.

As the curves suggest, the difference is negligible for very high and very low gate voltages. The difference becomes significant only for $1.5 \leq V_{\mathrm{GS}} \leq 2.5\mathrm{V}$.

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 6.4.15.33 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 $V_{\mathrm{GS}} = 2.0\mathrm{V}$:

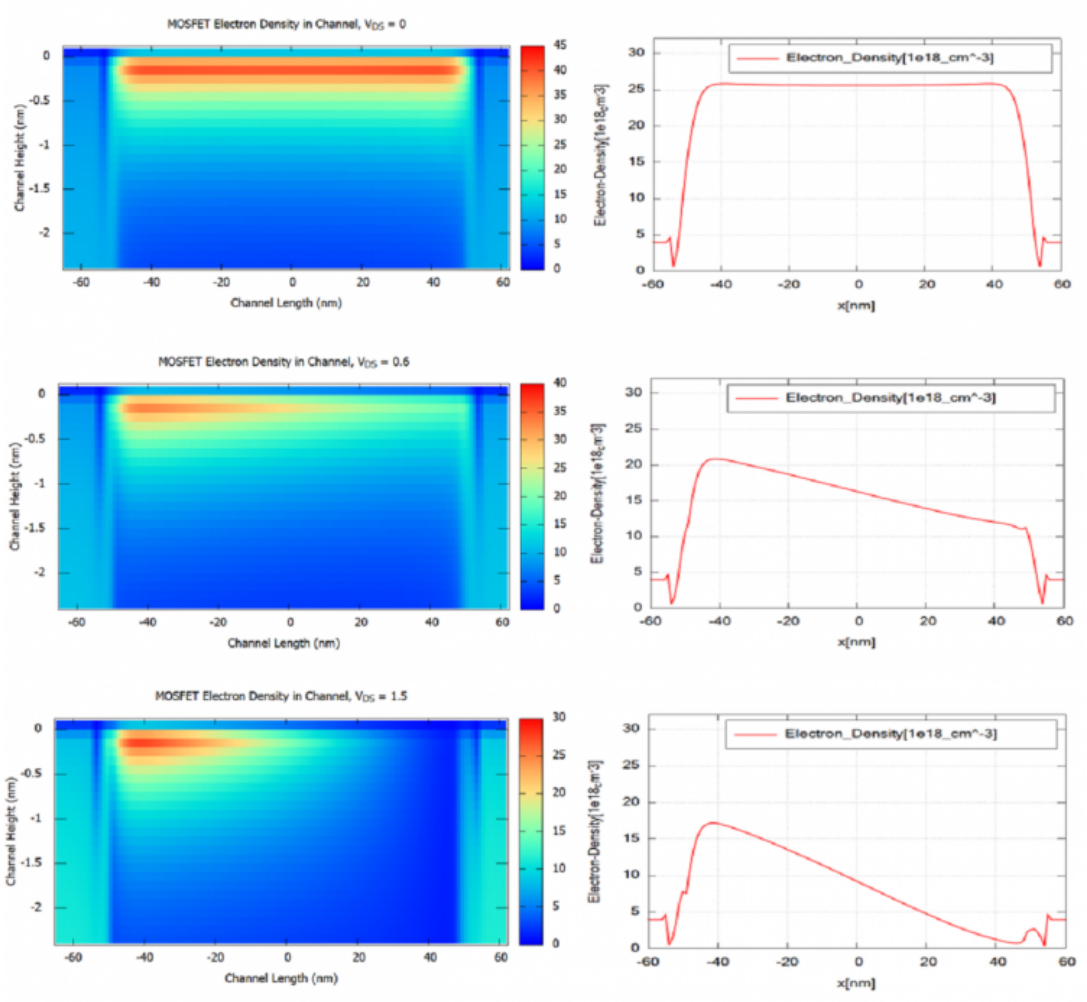


Figure 6.4.15.33: The comparison of the electron density distribution in the channel for $V_{\mathrm{DS}} = [0.0, 0.6, 1.5]\mathrm{V}$ at the gate voltage of $V_{\mathrm{GS}} = 2.0\mathrm{V}$, showing the pinch-off effect and the effective channel shortening. The 3 pictures of the left show the electron 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 6.4.15.34 illustrates these depletion widths:

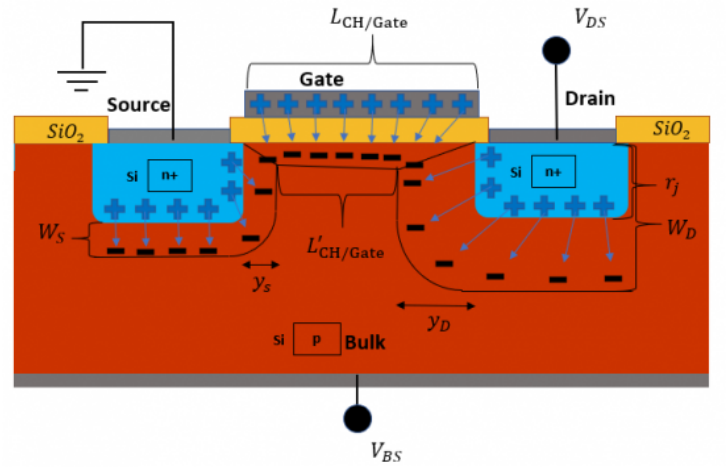


Figure 6.4.15.34: The illustration of the shortening of the effective channel length due to the expansion of the drain and source depletion widths.

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 (6.4.15.14),

$$y_{\mathrm{S}} \approx \sqrt{\frac{2\varepsilon_{\mathrm{s}}}{qN_{\mathrm{A}}}(\psi_{\mathrm{bi}} - \psi_{\mathrm{s}} - V_{\mathrm{BS}})}, y_{\mathrm{D}} \approx \sqrt{\frac{2\varepsilon_{\mathrm{s}}}{qN_{\mathrm{A}}}(\psi_{\mathrm{bi}} + V_{\mathrm{D}} - \psi_{\mathrm{s}} - V_{\mathrm{BS}})}. \tag{6.4.15.14}$$

From which then results the effective channel length (as also illustrated in figure Figure 6.4.15.34), 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 (6.4.15.15),

$$L_{\mathrm{min}} = C\left[r_j d_{\mathrm{ox}}(W_{\mathrm{S}} + W_{\mathrm{D}})^2\right]^{1/3}, \tag{6.4.15.15}$$

where C is a constant, and W_{S} and W_{D} are the depletion widths of source and drain,

$$W_S = \sqrt{\frac{2\epsilon_s}{qN_A}(\psi_{\mathrm{bi}} - V_{\mathrm{BS}})}, W_{\mathrm{D}} = \sqrt{\frac{2\varepsilon_{\mathrm{s}}}{qN_{\mathrm{A}}}(\psi_{\mathrm{bi}} + V_{\mathrm{D}} - V_{\mathrm{BS}})}. \tag{6.4.15.16}$$

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 $L_{\mathrm{min}} \geq y_{\mathrm{S}} + y_{\mathrm{D}}$. 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 $L_{\mathrm{min}} = 390\mathrm{nm}$. 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 6.4.15.35 illustrates the pinch-off effect and channel length modulation in the same MOSFET model with a $L_{\mathrm{G}} = 2\mu\mathrm{m}$:

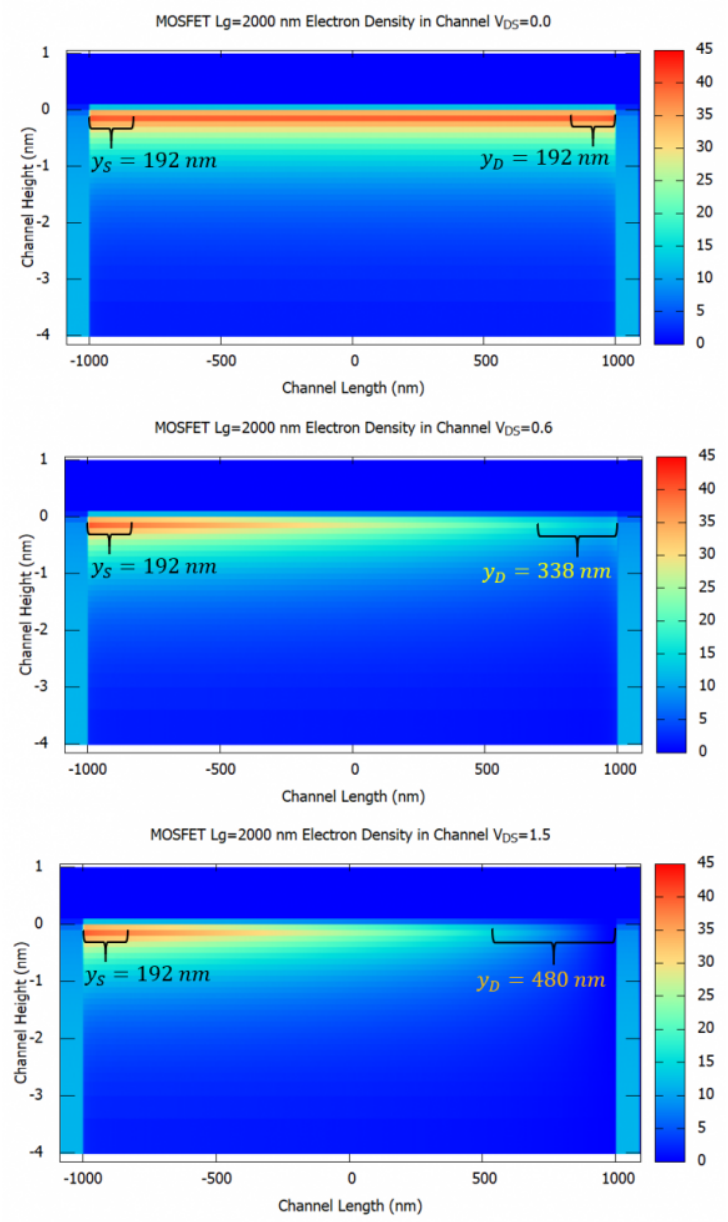


Figure 6.4.15.35: The illustration of the pinch-off effect, and the channel length modulation, in the N-Ch MOSFET with a channel length of $L_{\mathrm{G}} = 2\mu m$, calculated classically.The depletion widths at the source and drain, y_{S} and y_{D}, estimated from the analytical formulas given above, are indicated.

So therefore, according to the calculations in figure Figure 6.4.15.35, 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 100nm 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 $I_{\text{DS,sat}}(V_{\text{DS,sat}})$. This phenomenon is known as the punch-trough effect and is present in our output characteristics in figures Figure 6.4.15.25 and Figure 6.4.15.26 of the output characteristics section. The DIBL effect is shown in figure Figure 6.4.15.36, comparing two channel lengths:

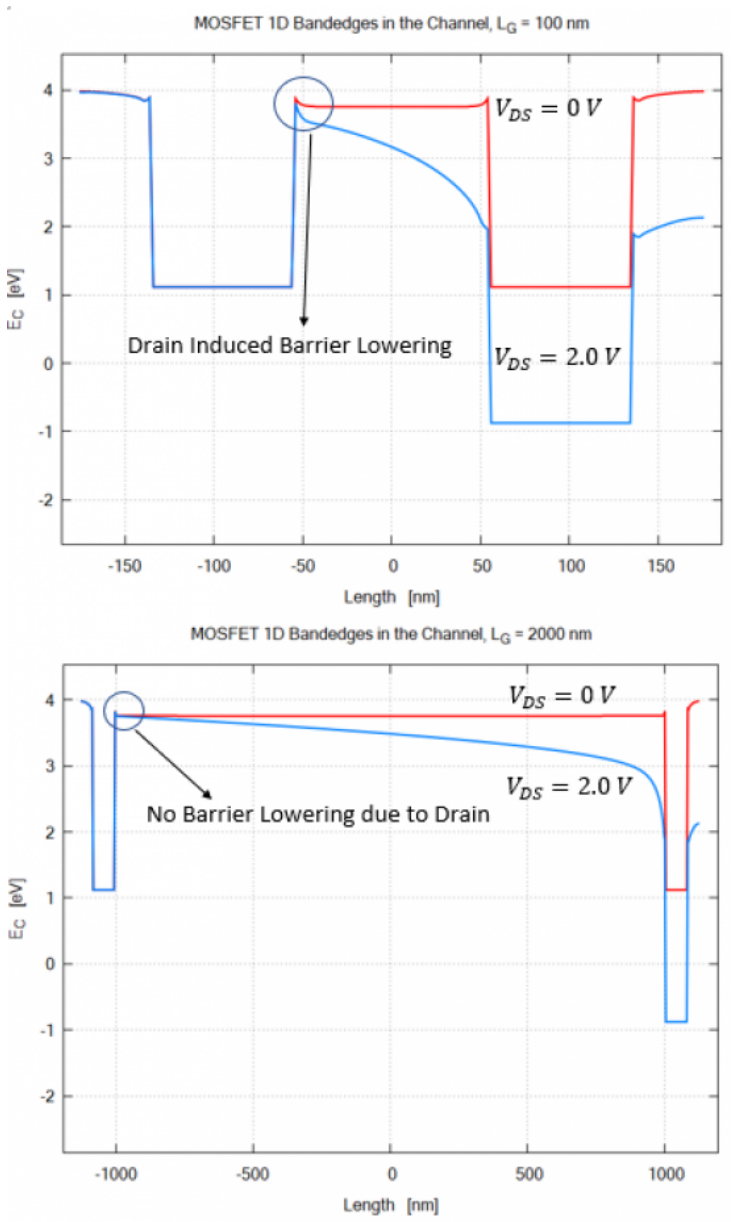


Figure 6.4.15.36: The illustration of the drain induced barrier lowering (DIBL) in 100nm gate-length MOSFET, compared to the 2000nm gate-length variant (where there are no barrier lowering).

In order to recognize the punch-through effect, the sweep of the gate-length should be performed at high drain-source voltages (for example $V_{\text{DS}} = 2.0\text{V}$) 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 6.4.15.37 shows this effect:

As it could be seen in Figure 6.4.15.37, the MOSFET with gate-length of $L_G \leq 400\text{nm}$ would definitely suffer from the punch-through effect. However, one could be safe with a channel length of 500nm or 600nm. Let us now

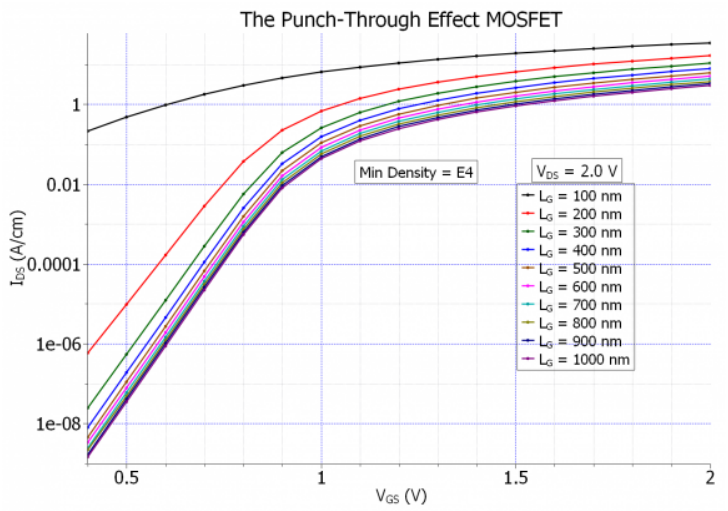


Figure 6.4.15.37: The punch-through effect for a set of channel lengths in MOSFET apparent in the input characteristics (calculated with minimum density of $10e4$).

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 6.4.15.38:

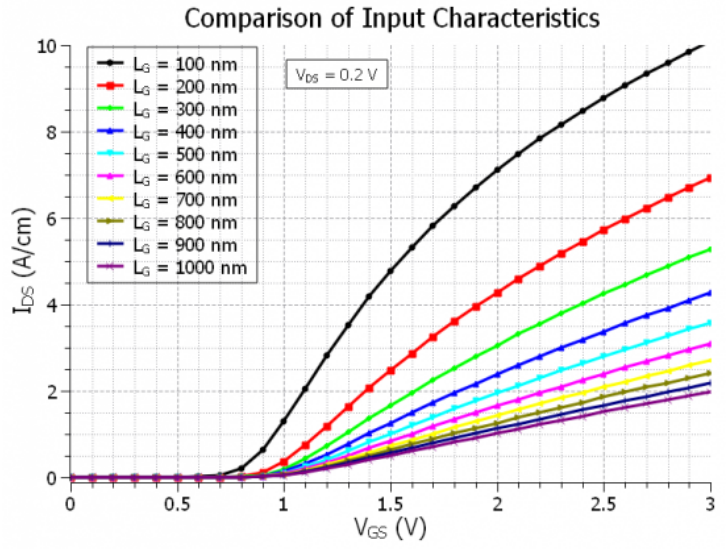


Figure 6.4.15.38: The effect of increasing the channel length on the input characteristics at $V_{\mathrm{DS}} = 0.2\mathrm{V}$.

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 6.4.15.39 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 6.4.15.40:

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 taking into consideration the new channel length $L_G = 600\mathrm{nm}$, we get for the effective mobility:

$$\mu_n^{\mathrm{eff}} \approx 788 \frac{\mathrm{cm}^2}{\mathrm{V} \cdot \mathrm{s}}$$

The calculated mobility from the simulation is once again $933\mathrm{cm}^2/\mathrm{Vs}$ in the substrate, however it is $576\mathrm{cm}^2/\mathrm{Vs}$

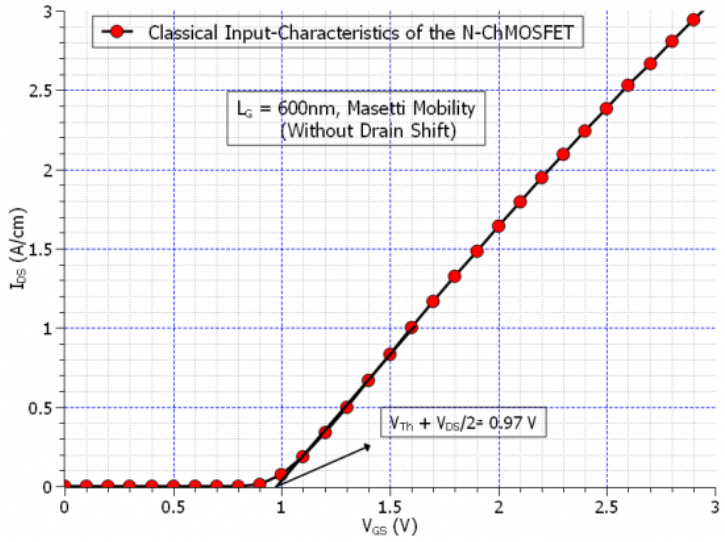


Figure 6.4.15.39: The input characteristics of the long-channel $L_{\mathrm{G}} = 600\mathrm{nm}$ MOSFET, calculated with the Masetti mobility, showing the value of the threshold voltage V_{Th}.

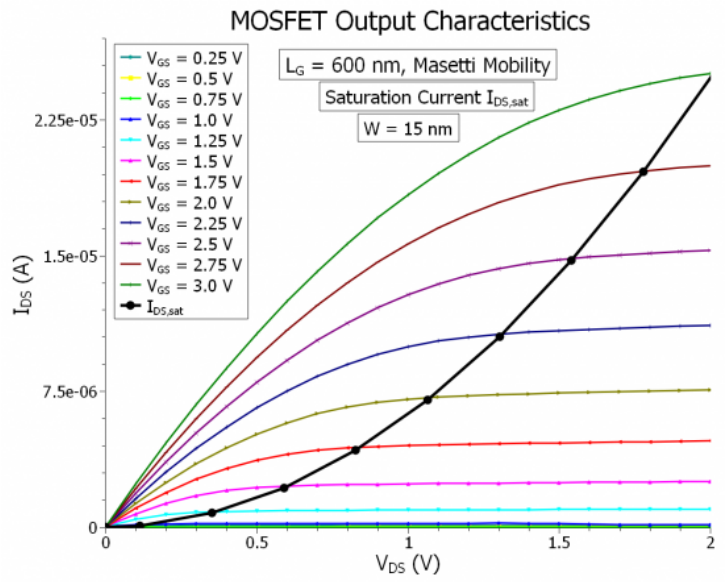


Figure 6.4.15.40: The output characteristics of the long-channel $L_{\mathrm{G}} = 600\mathrm{nm}$ MOSFET, showing the diminishing of DIBL effect.

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 $V_{\mathrm{DS}} = 2\mathrm{V}$, that the MOSFET device with a gate length of smaller than $L \leq 400\mathrm{nm}$, would suffer from the punch-through effect. However, if we further shorten our gate length below $100\mathrm{nm}$, the situation would even be worse. Namely the leakage current would be so high, that even at very low source-drain voltages $V_{\mathrm{DS}} = 0.2\mathrm{V}$, the MOSFET would conduct, even at gate-voltages below the threshold voltage $V_{\mathrm{GS}} < V_{\mathrm{Th}}$, and therefore the switching capability of the MOSFET would be diminished and eliminated. Figure Figure 6.4.15.41 illustrates this phenomenon:

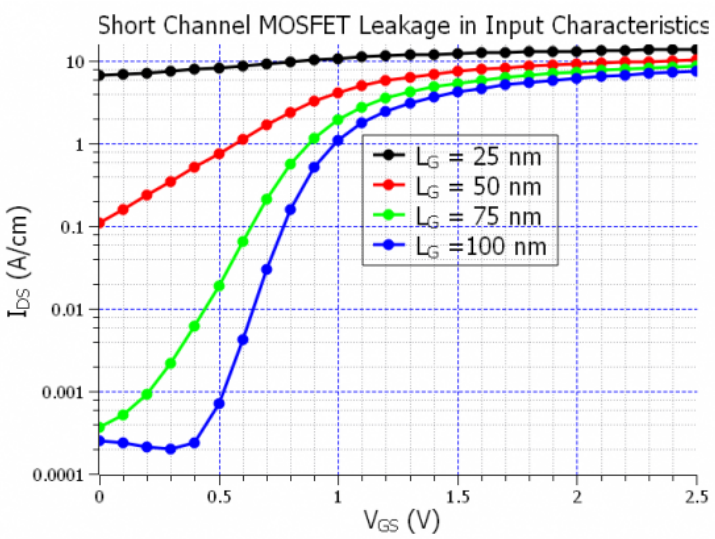


Figure 6.4.15.41: The comparison of input characteristics of the N-Ch MOSFET calculated quantum mechanically with the Masetti mobility, showing the leakage current in the input characteristics.

As the above input characteristics curves show, for gate-length below $100\mathrm{nm}$ there is basically no valid switching function possible, as the drift current has already started at $V_{\mathrm{GS}} = 0\mathrm{V}$ for $L_G = 75\mathrm{nm}$. 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 6.4.15.42 shows:

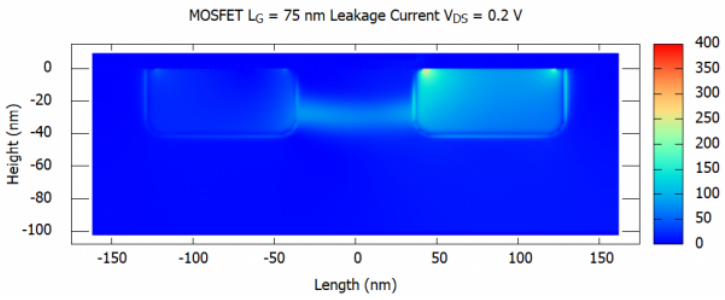


Figure 6.4.15.42: The norm of the leakage current in $L_G = 75\mathrm{nm}$ MOSFET, at zero gate-voltage $V_{GS} = 0$, flowing within the bulk.

If we even consider the $L_G = 25\mathrm{nm}$ 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_quantum_regions = yes
   }

   energy_resolved_density{
       min =  -0.5
       max =  1.0
       energy_resolution = 0.001
       only_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_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 6.4.15.43:

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 6.4.15.45, 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: nn/nn/nnnn

— NEW — Two-dimensional electron gas in a Si MOSFET

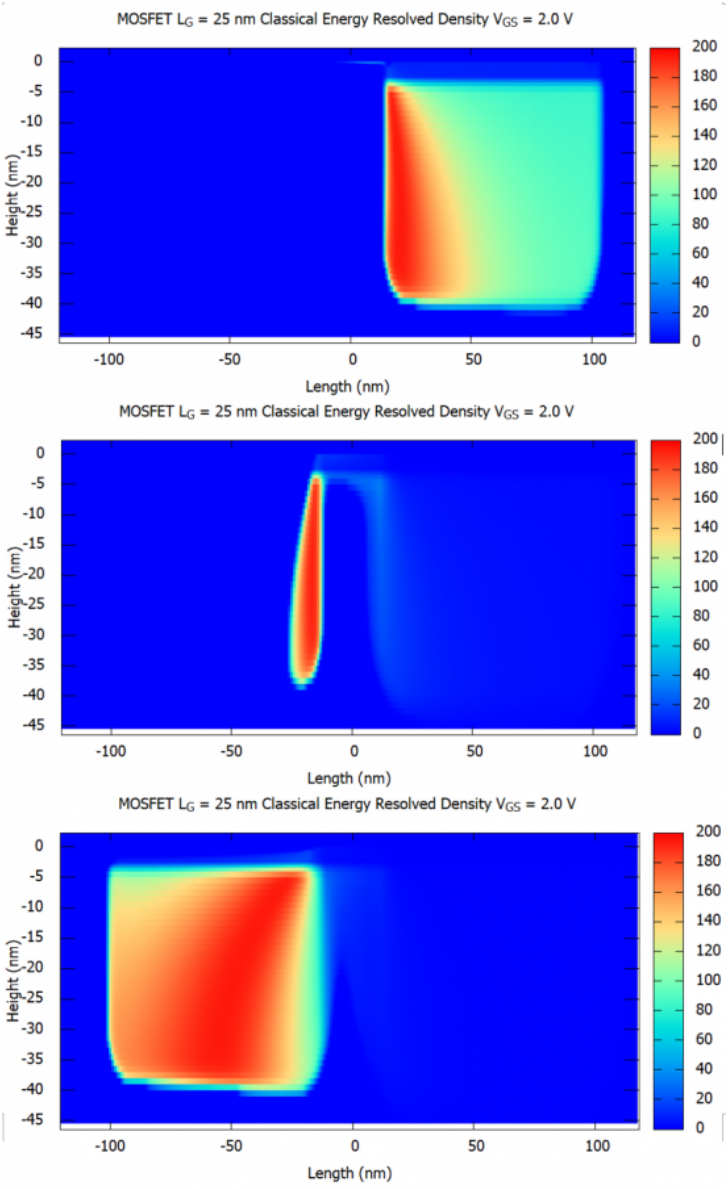


Figure 6.4.15.43: The classical energy resolved density in the $L_{\mathrm{G}} = 25\mathrm{nm}$ MOSFET at three different energy levels.

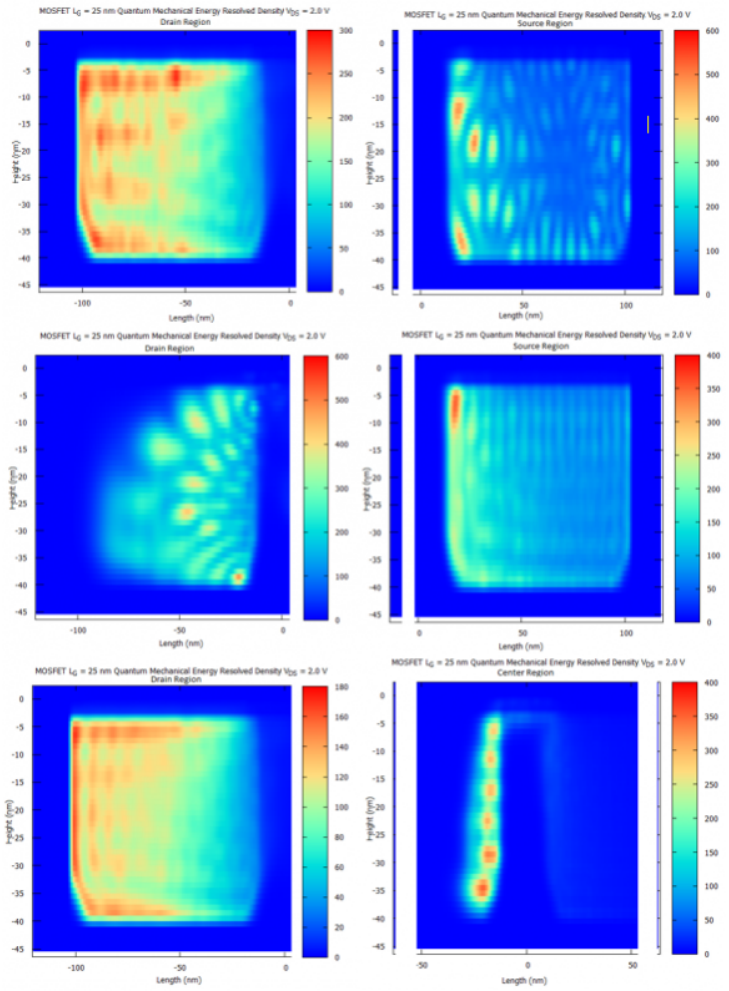


Figure 6.4.15.44: The quantum mechanical energy resolved density in the MOSFET source and drain regions, showing spacial quantum confinement at discrete energy levels.

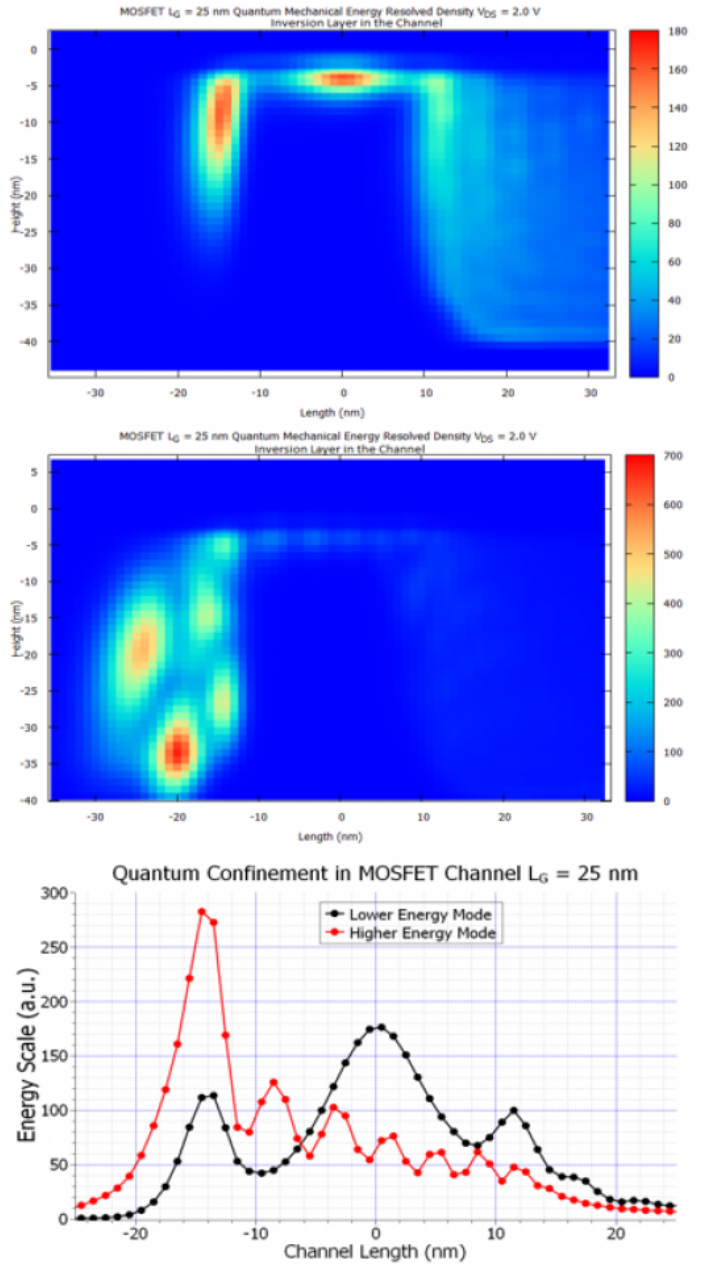


Figure 6.4.15.45: The quantum mechanical energy resolved density in the inversion layer of the MOSFET-channel, at two different energy levels, showing the standing wave pattern, which indicates quantum confinement.

- *Electron sheet density in the inversion channel as a function of applied gate voltage*

Header

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

- *2DEG_Si_MOSFET_1D_nnp.in*

Main adjustable parameters in the input file:

- parameter `$maximum_density`
- parameter `$minimum_density`

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 6.4.15.46: **0.7 V** (The electron ground state is above the electron Fermi level $E_{F,n}$)

Figure 6.4.15.47: **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)

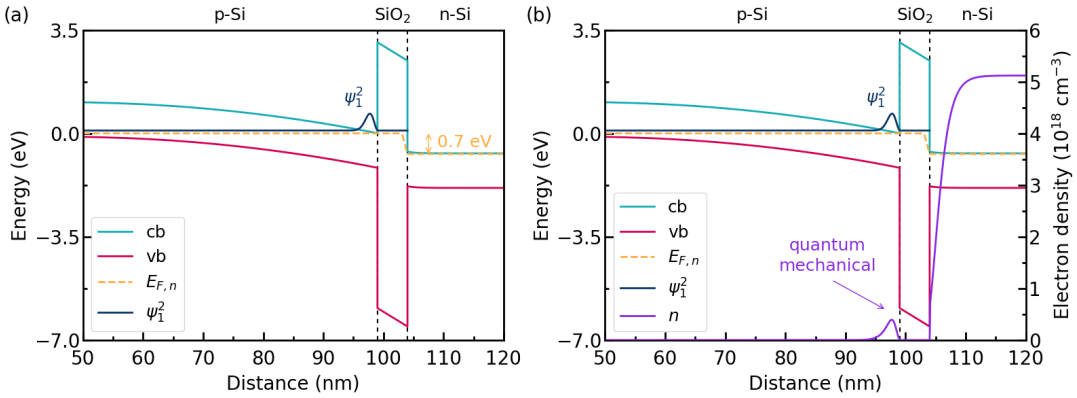


Figure 6.4.15.46: The calculated band edges are shown in (a). The quasi Fermi level of electrons $E_{F,n}$ drops 0.7 eV from p-Si to n-Si due to the gate bias. The calculated electron density n is overlaid on the band diagram in (b).

The amplitude of the ground state ψ_1^2 is **above** $E_{F,n}$ as you can see.

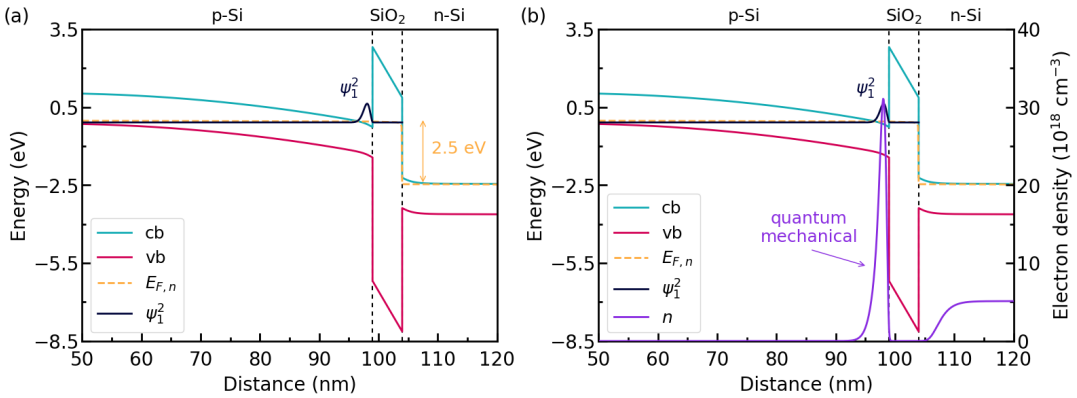


Figure 6.4.15.47: The calculated band edges are shown in (a). The quasi Fermi level of electrons $E_{F,n}$ drops 2.5 eV from p-Si to n-Si due to the gate bias. The calculated electron density n is overlaid on the band diagram in (b).

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 6.4.15.48 shows the electron sheet density of the p-Si inversion layer.

To obtain the capacitance-voltage characteristics, you have to calculate the derivative of the sheet density.

Last update: 17/07/2024

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.in / *_nnp.in*
- *2DSi_TGMOS_2Dcut_atGate_qm.in / *_nnp.in*
- *2DSi_TGMOS_2Dcut_atGate_qm_iso.in / *_nnp.in*

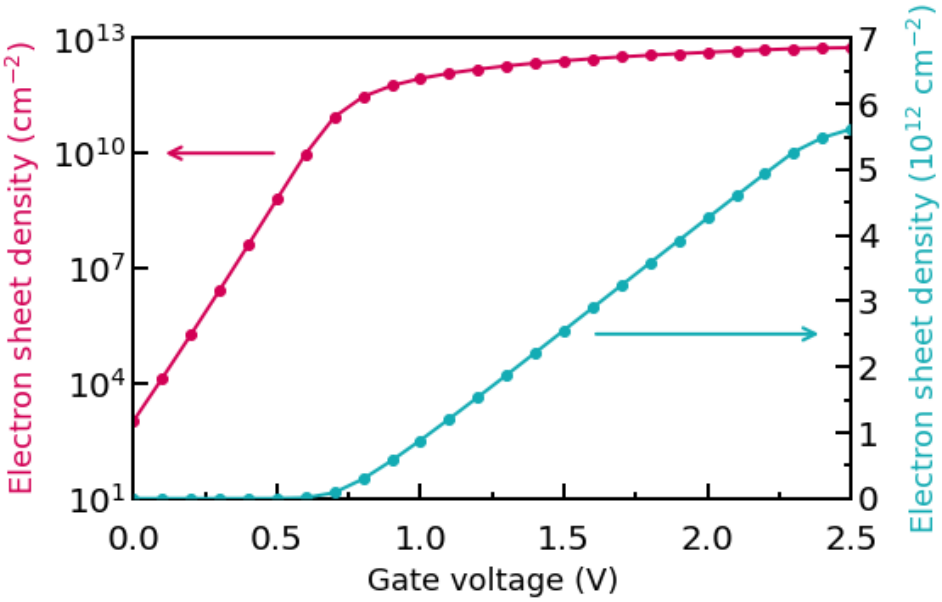


Figure 6.4.15.48: The electron sheet density of the p-Si inversion layer is shown.

- *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.

The values and graphs described in this tutorial are the result of *nextnano*[3].

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.

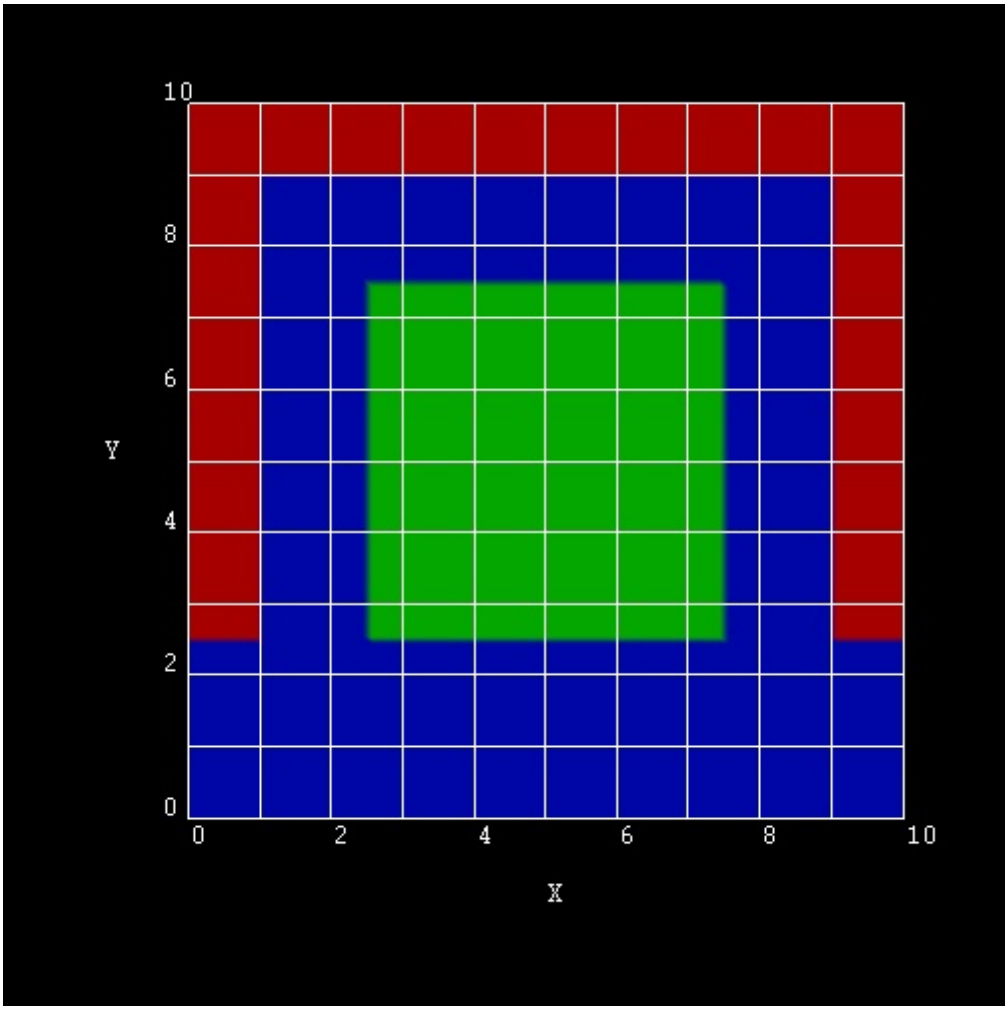


Figure 6.4.15.49: 2D slice of a 3D Triple Gate MOSFET

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).

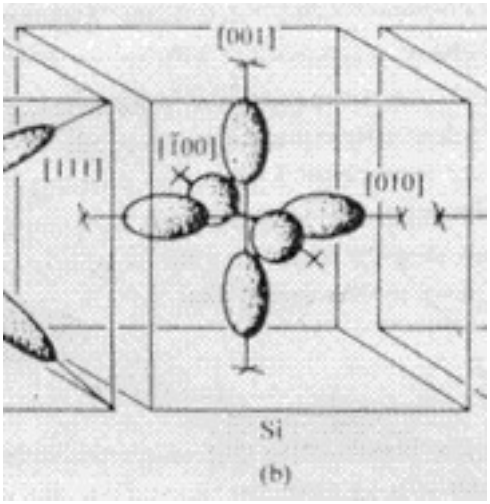


Figure 6.4.15.50: constant energy surface of Si conduction band

Therefore, we need to solve **three** 2D Schrödinger equations with different effective mass tensor orientations.

Our Schrödinger equations are numbered `X1`, `X2`, `X3` in *nextnano++* or `deg1`, `deg2`, `deg3` in *nextnano*[3].

- `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.

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
        }
    }
}
```

```
! nextnano3
$output-1-band-schroedinger
 ...
 effective-mass-tensor = 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) $m_{xx} = m_l = 0.916 m_0$, $m_{yy} = m_t = 0.190 m_0$

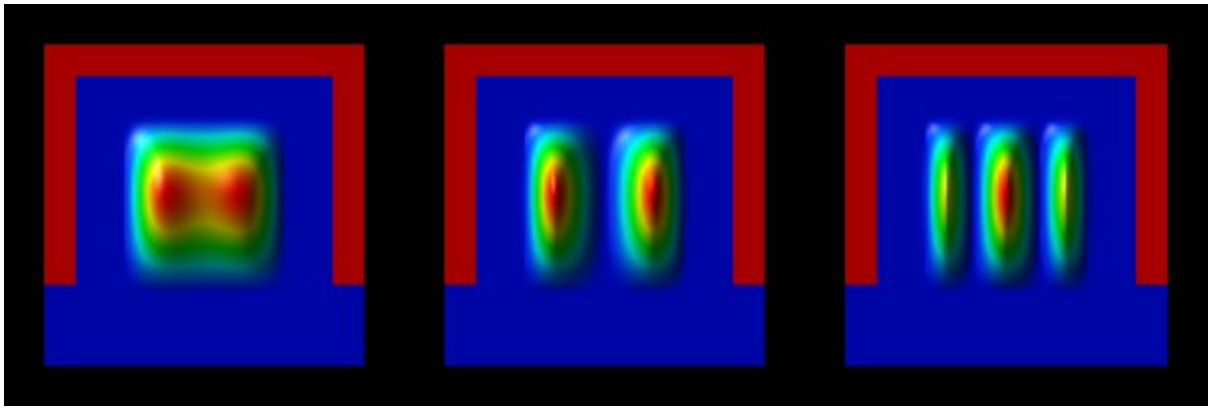


Figure 6.4.15.51: $E_{1,X1} = -26$ meV, $E_{2,X1} = -1$ meV, $E_{3,X1} = 77$ meV

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) $m_{xx} = m_t = 0.190 m_0$, $m_{yy} = m_l = 0.916 m_0$

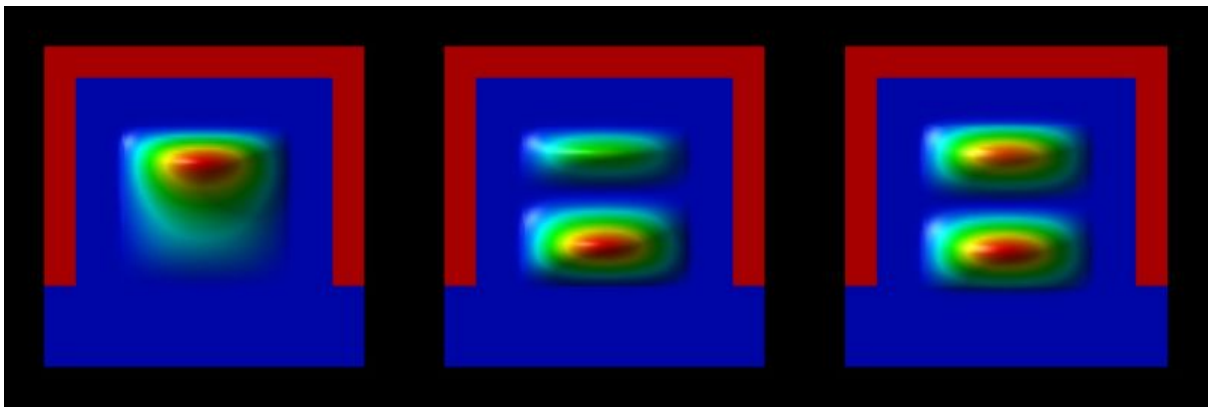


Figure 6.4.15.52: $E_{1,X2} = -28$ meV, $E_{2,X2} = 6$ meV, $E_{3,X2} = 82$ meV

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) $m_{xx} = m_{yy} = m_t = 0.190 m_0$

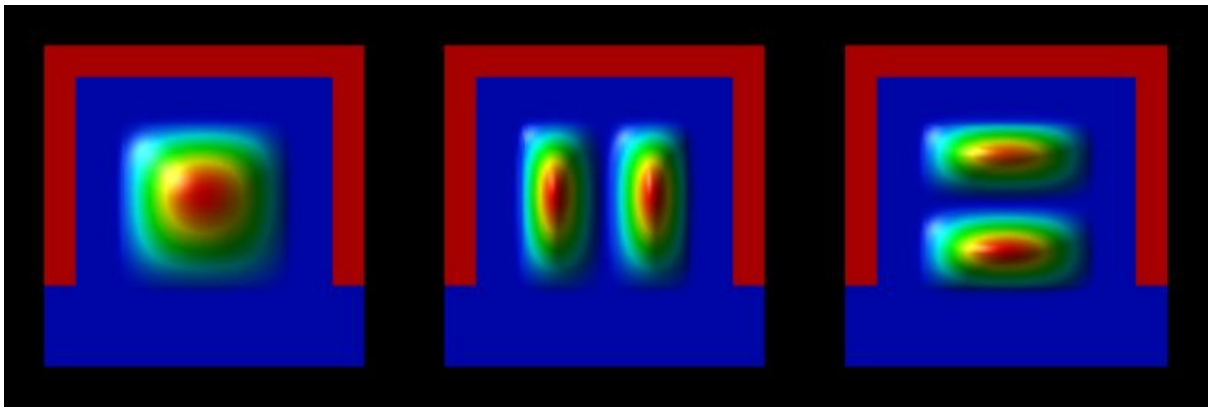


Figure 6.4.15.53: $E_{1,X3} = 16$ meV, $E_{2,X3} = 167$ meV, $E_{3,X3} = 173$ meV

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*

(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 6.4.15.54:

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 6.4.15.55 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 6.4.15.56.

- *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.

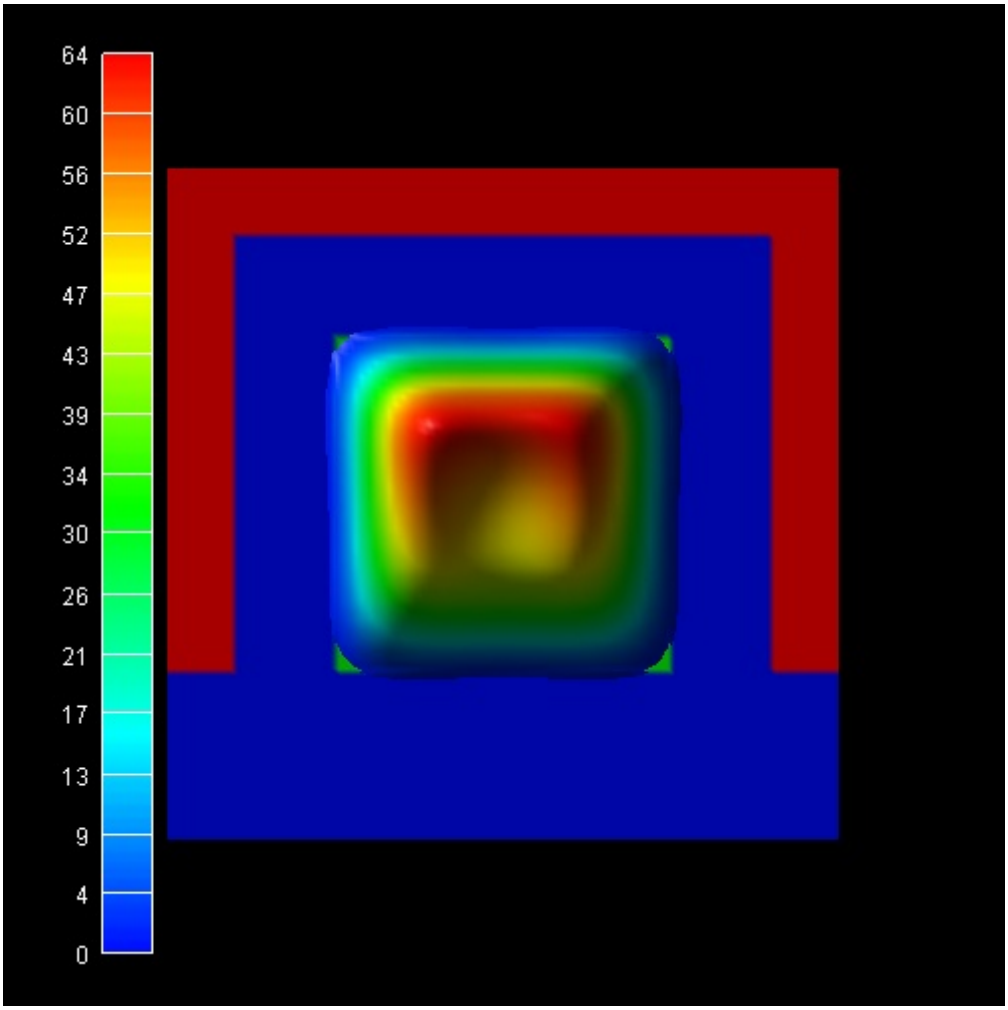


Figure 6.4.15.54: electron density

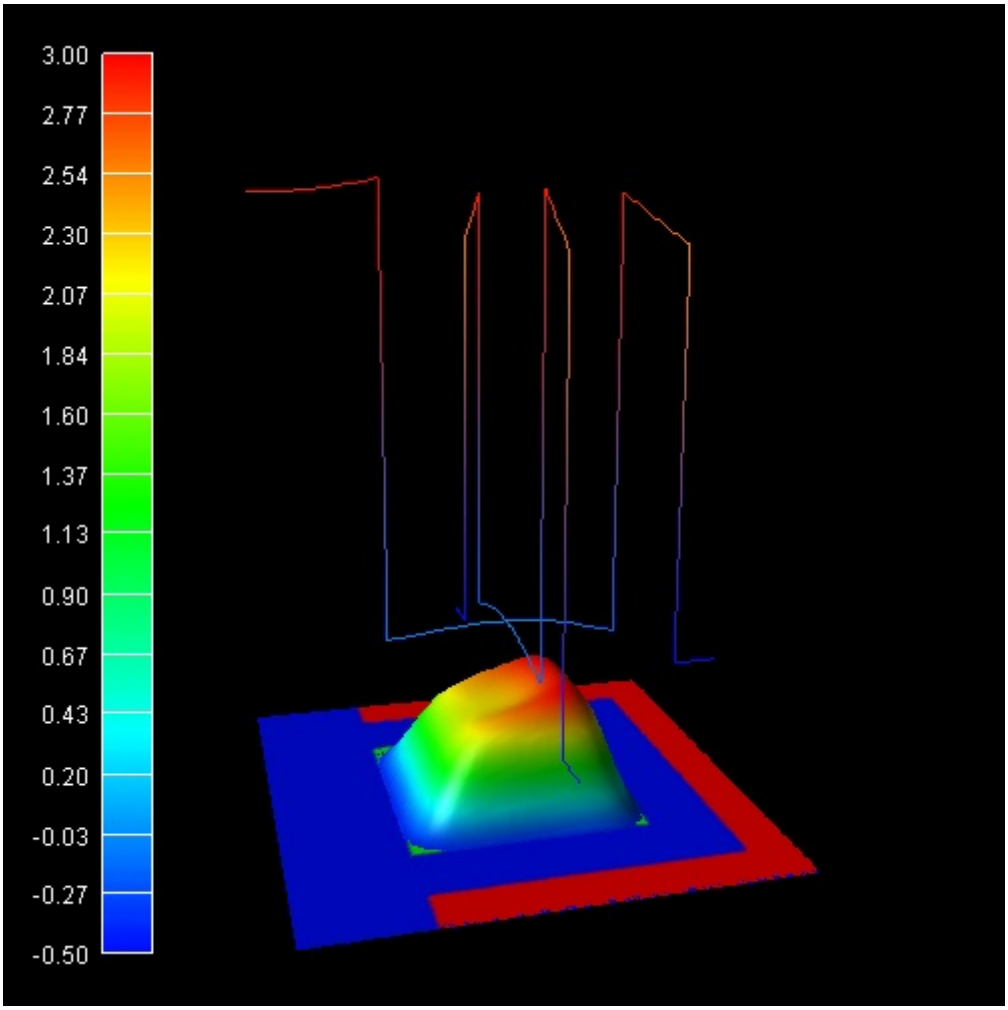


Figure 6.4.15.55: electron density and slices through the conduction band edges

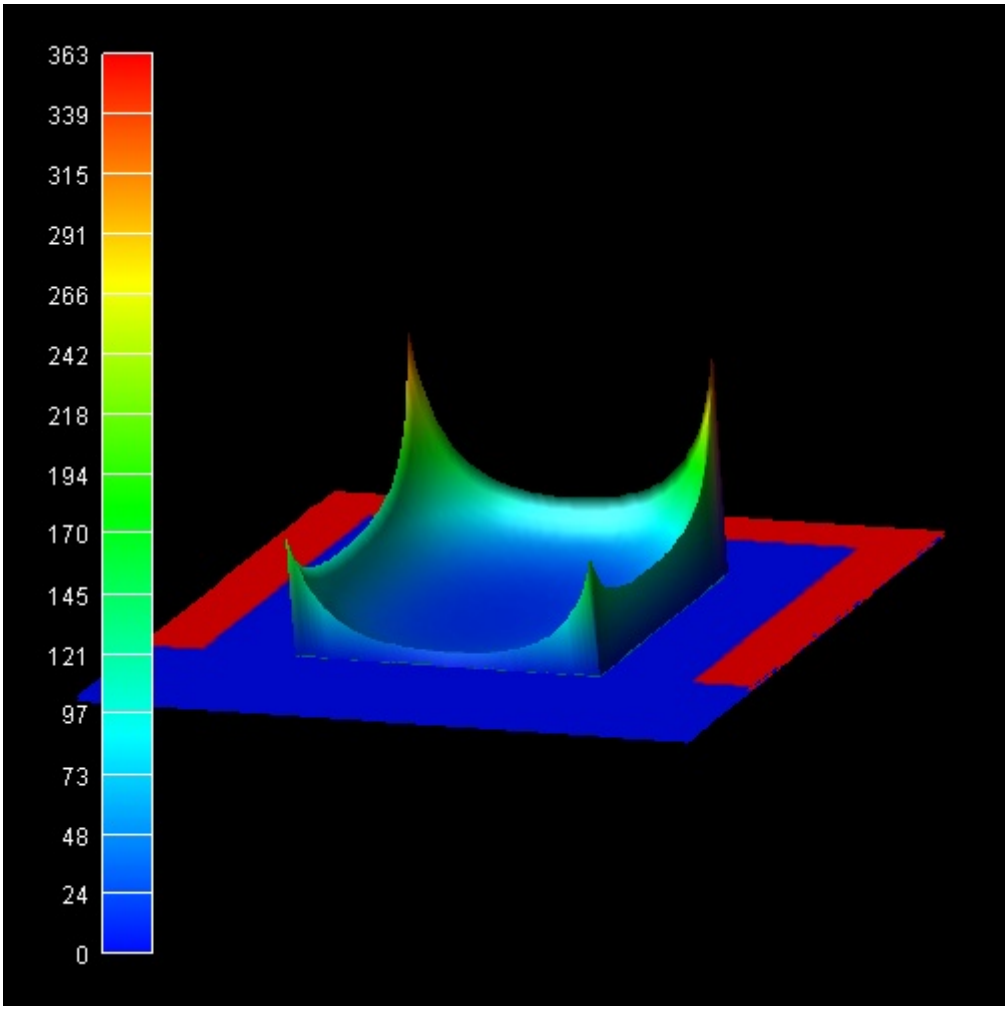


Figure 6.4.15.56: classical electron density calculated by *2DSi_TGMOS_2Dcut_atGate_cl.in*

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:

- $m_{xx} = m_{yy} = m_{DOS} = 0.321 m_0$

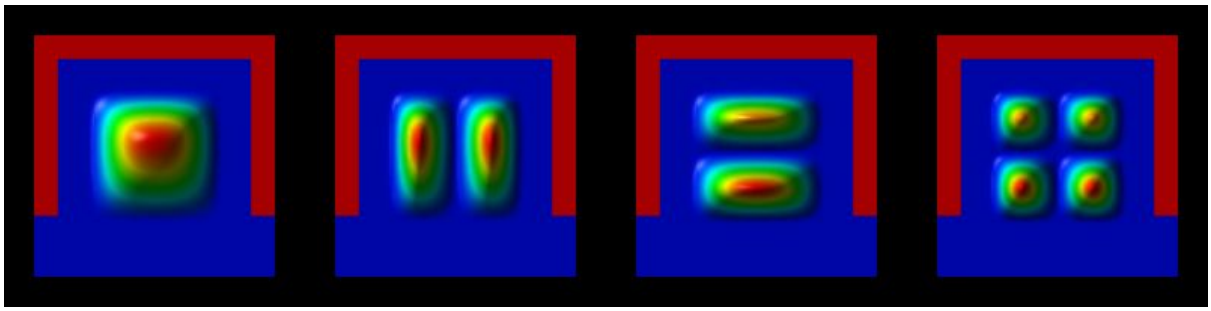


Figure 6.4.15.57: $E_1 = -21$ meV, $E_2 = 69$ meV, $E_3 = 75$ meV, $E_3 = 166$ meV

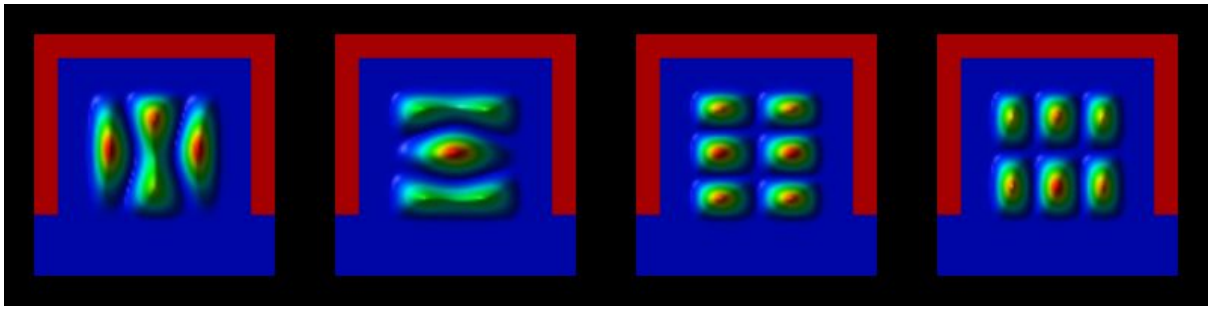


Figure 6.4.15.58: $E_5 = 262$ meV, $E_6 = 270$ meV, $E_7 = 360$ meV, $E_8 = 360$ meV

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/ev21*

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 6.4.15.60 and Figure 6.4.15.61 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.

Last update: nn/nn/nnnn

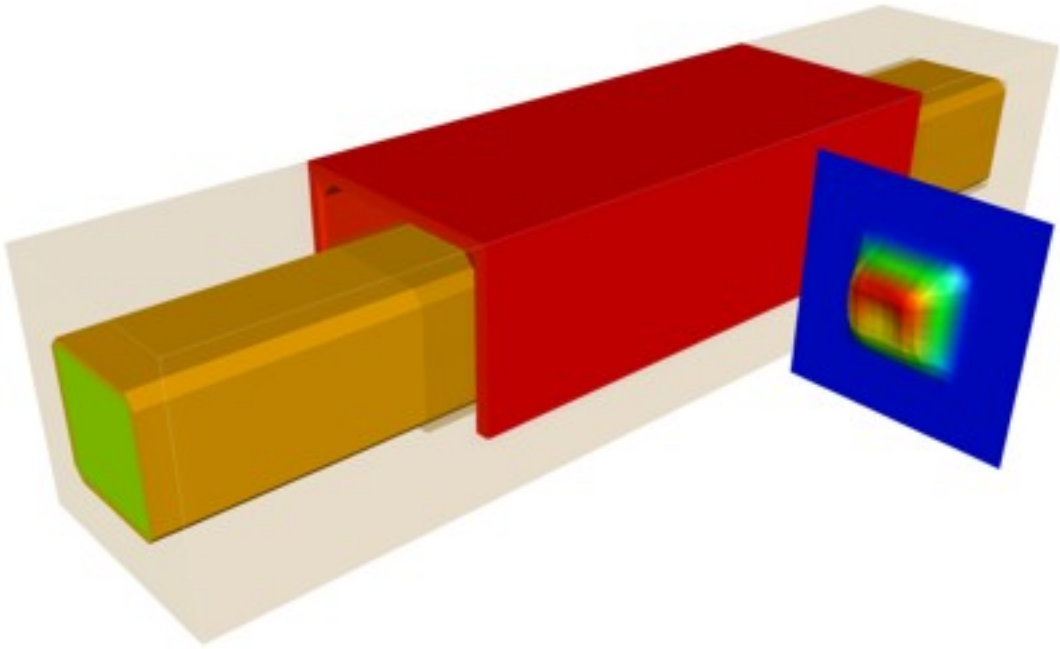


Figure 6.4.15.59: The whole 3D structure of this triple gate MOSFET and electron density through a 2D slice

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 6.4.15.62 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.

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*

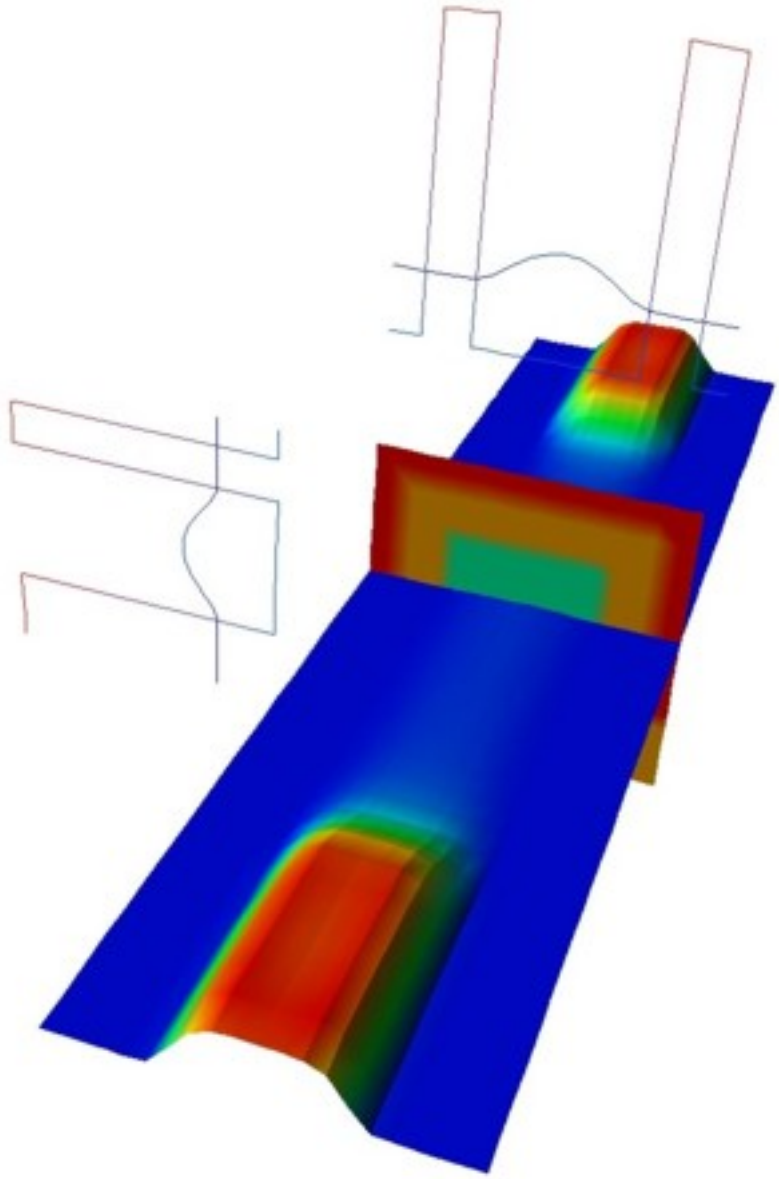


Figure 6.4.15.60: closed channel, $V_{SD} = 0.0$V, $V_{SG} = 0.0$V

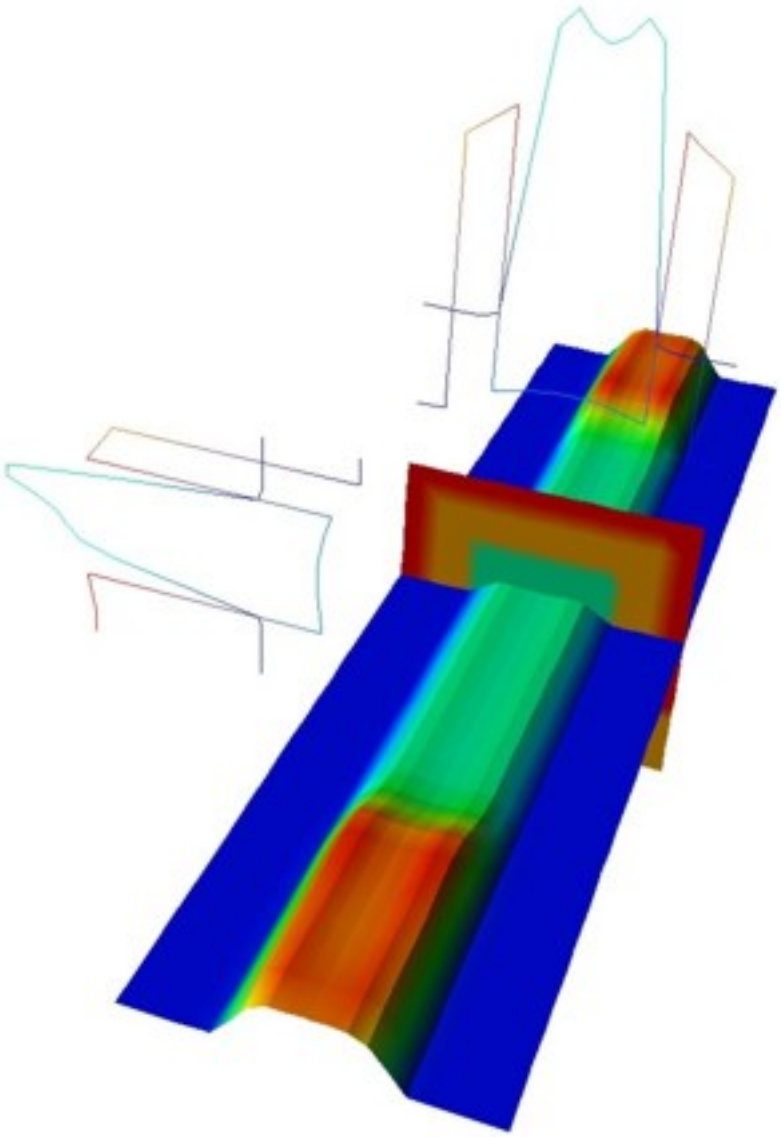


Figure 6.4.15.61: open channel, $V_{SD} = 0.0$V, $V_{SG} = 0.5$V

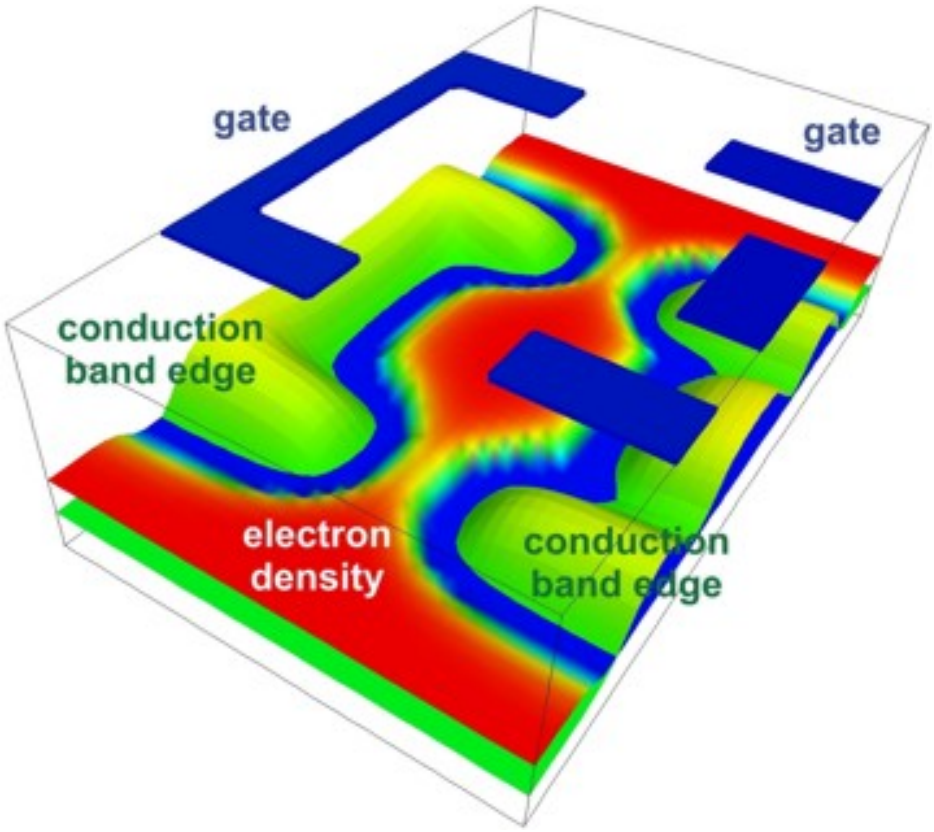


Figure 6.4.15.62: Conduction band edges (green), electron density (red) and geometry of the top gates (blue).

Part 1: 1D simulation (self-consistent Schrödinger-Poisson)

Figure 6.4.15.63 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 6.4.15.64 shows the 3D structure that we are going to simulate.

Figure 6.4.15.65 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: nn/nn/nnnn

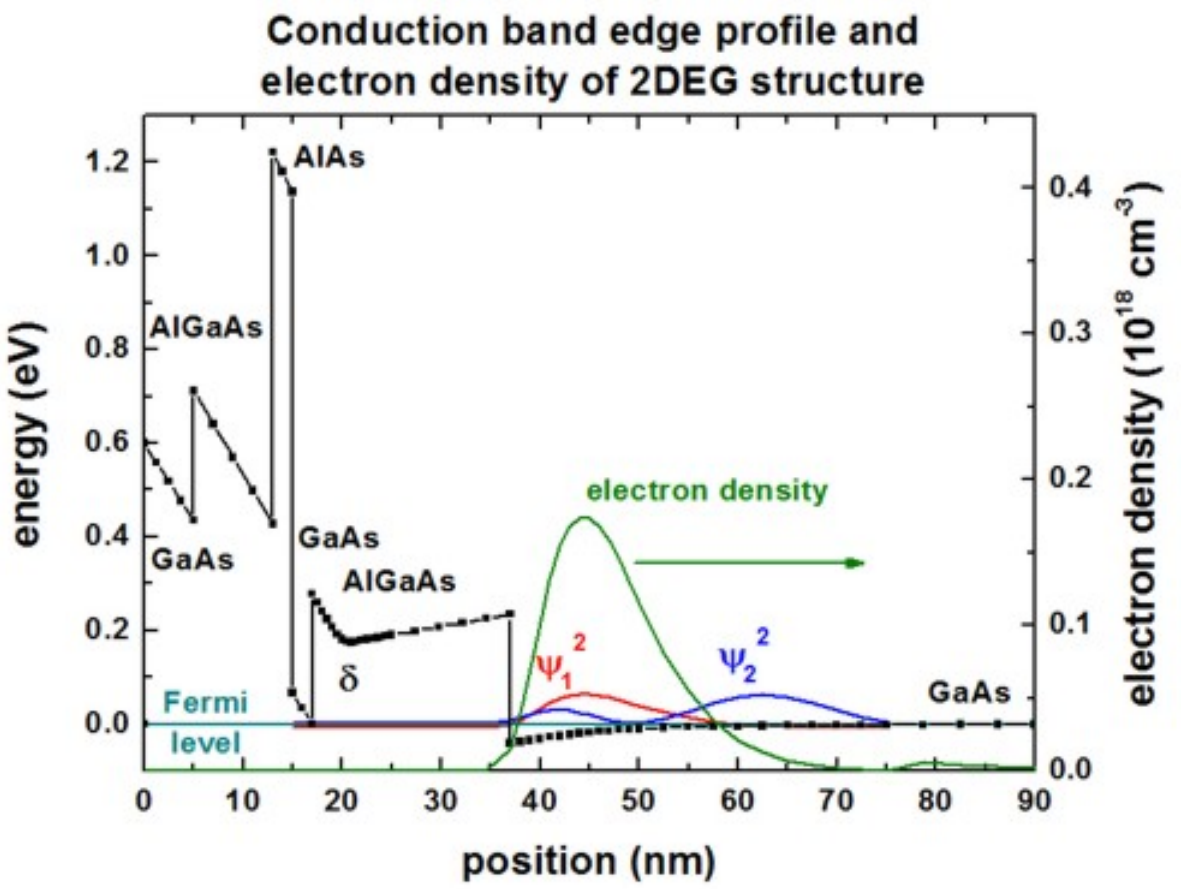


Figure 6.4.15.63: Conduction band edge profile (black), electron density (green) and Fermi levels (cyan) of 1D simulation.

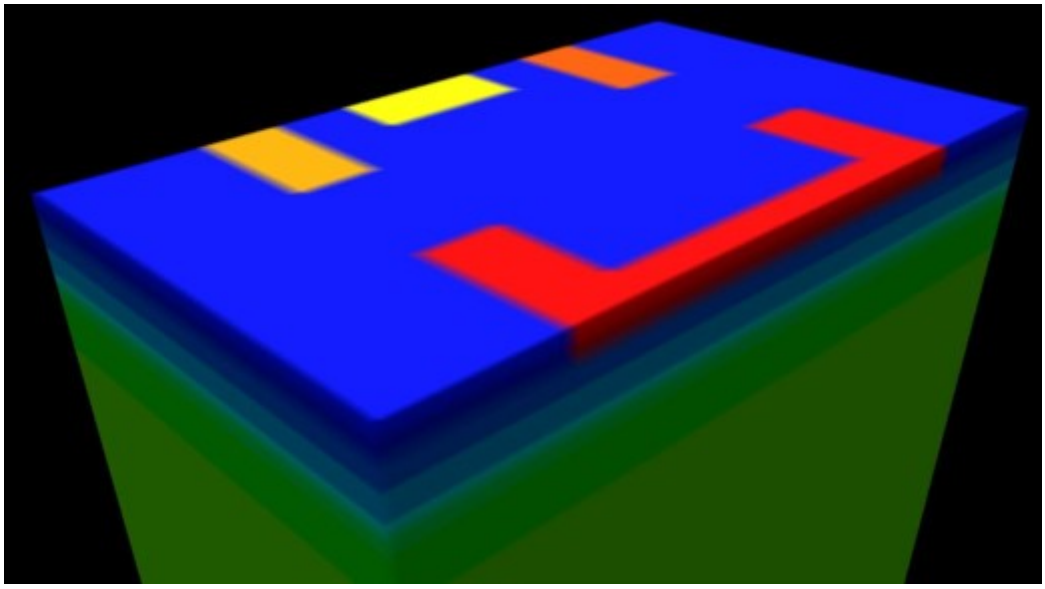


Figure 6.4.15.64: Device structure (3D)

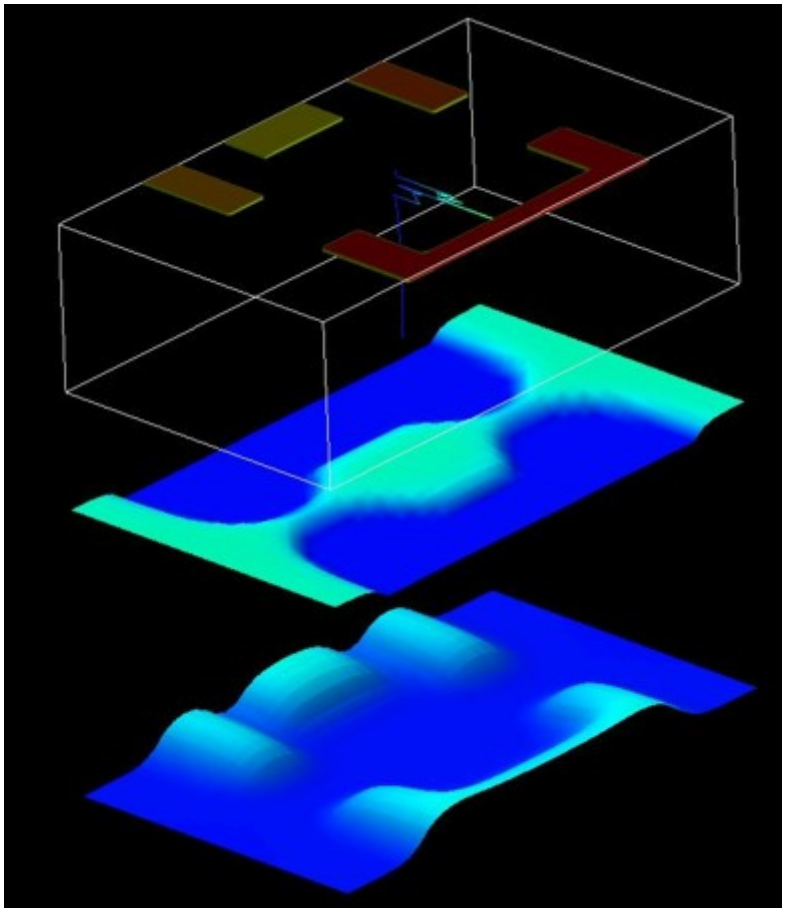


Figure 6.4.15.65: In the middle, the electron density is shown. The electron density has been calculated classically. At the bottom, the conduction band edge is shown.

— DEV — I–V characteristics of ultrathin-body Double Gate Metal Oxide Semiconductor Field Effect Transistor (DG MOSFET)

Attention: This tutorial is under construction

Input files:

- *IV_DG-MOSFET_Si_2D_classical_nnp.in*
- *IV_DG-MOSFET_Si_2D_quantum_nnp.in*
- *IV_DG-MOSFET_Si_3D_classical_nnp.in*

Scope:

This tutorial aims to simulate the I-V characteristics of a Double Gate metal oxide semiconductor field effect transistor (MOSFET).

Output files:

- *bias_xxxxx\density_electron.avs.fld*
- *IV_characteristics.dat*

Double Gate MOSFET

The main idea of a Double Gate 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 6.4.15.66. 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.

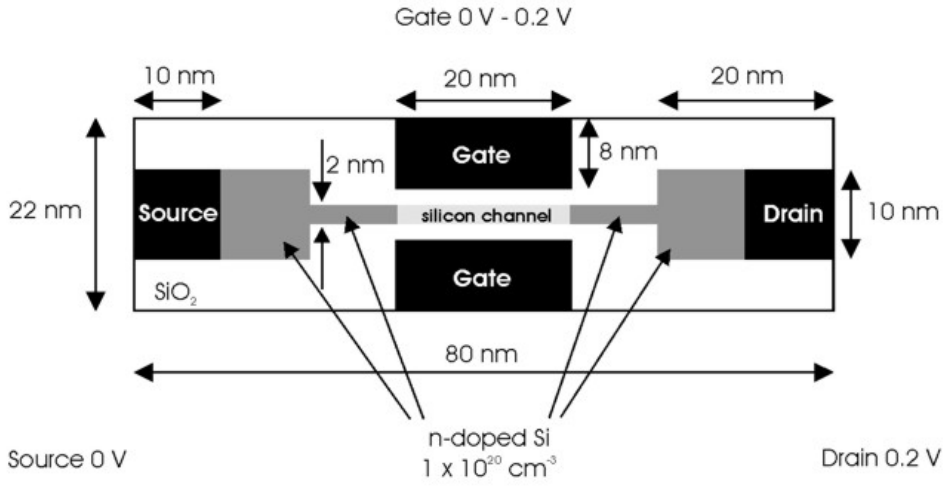


Figure 6.4.15.66: Geometry of the simulated Double Gate MOSFET.

The material regions defined in the input file for the *nextnano++* simulations are shown in Figure 6.4.15.67. 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 $0\,\mathrm{V}$ to $1\,\mathrm{V}$.

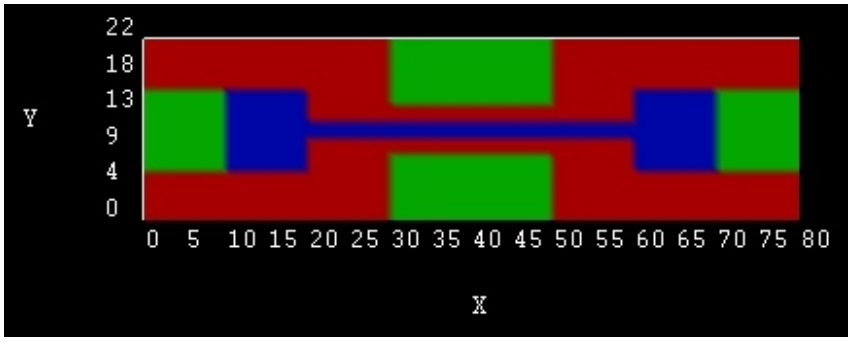


Figure 6.4.15.67: Schematic top view of the material regions defined in the *nextnano++* simulations.

The numerical grid employed in the simulations is shown in Figure 6.4.15.68.

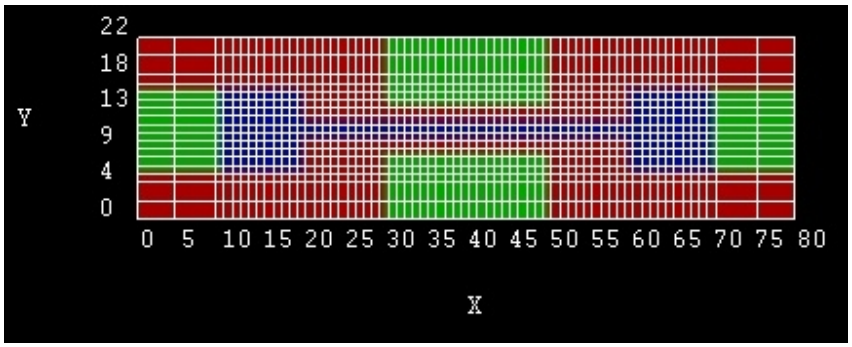


Figure 6.4.15.68: Grid lines of the Double Gate MOSFET

Input file

For the simulations, the following parameters, which are specified in the corresponding input file *IV_DG-MOSFET_Si_2D_classical_nnp.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 6.4.15.69 the electron density inside the MOSFET structure at $0\,\mathrm{V}$ is shown. The corresponding data is contained in the file *bias_00000\density_electron.avs.fld*.

In Figure 6.4.15.70 the electron density inside the MOSFET structure at $0.2\,\mathrm{V}$ is shown. The corresponding data 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.

For comparison, Figure 6.4.15.70 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 *IV_DG-MOSFET_Si_2D_quantum_nnp.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.

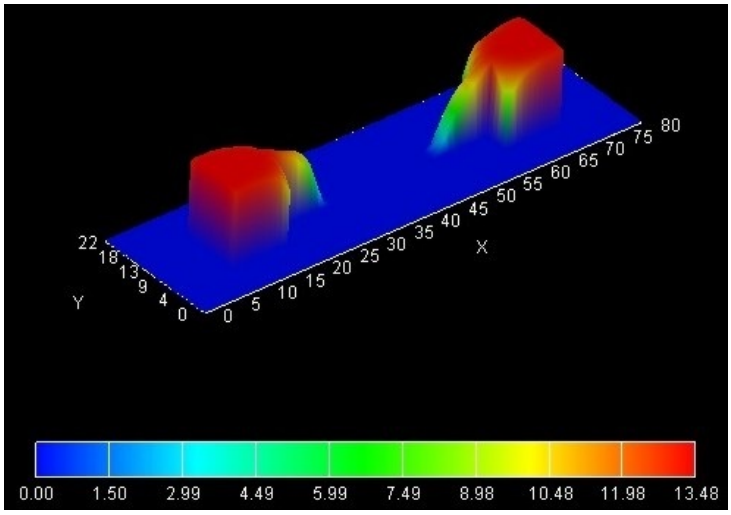


Figure 6.4.15.69: Electron density in units of $1 \cdot 10^{18}\mathrm{cm}^{-3}$ at $0\,\mathrm{V}$ gate voltage.

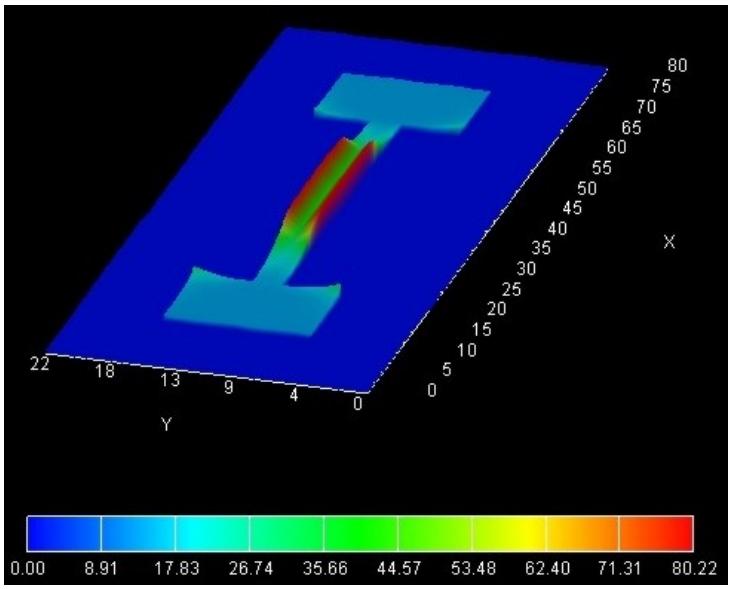


Figure 6.4.15.70: Electron density in units of $1 \cdot 10^{18}\mathrm{cm}^{-3}$ at $0.2\,\mathrm{V}$ gate voltage.

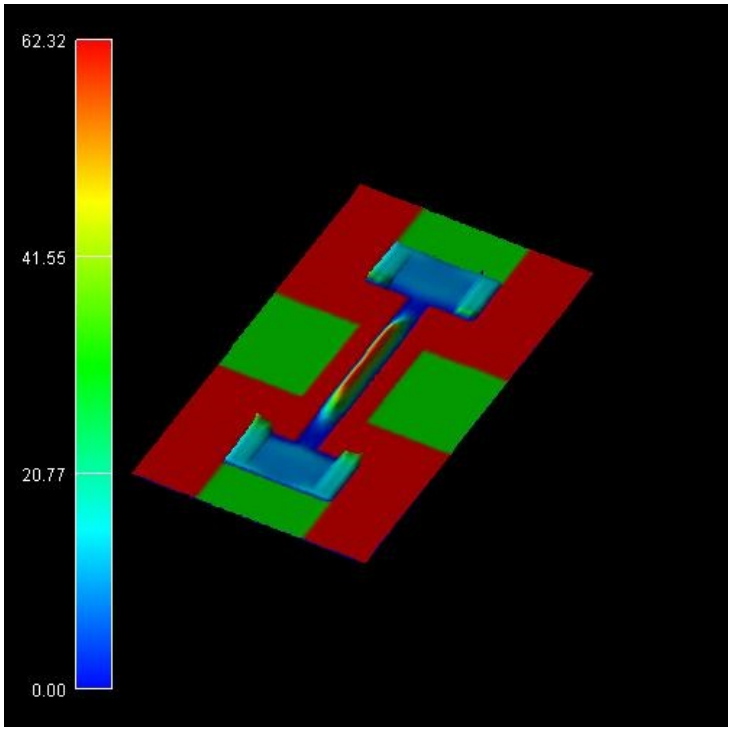


Figure 6.4.15.71: Quantum mechanical electron density in units of $1 \cdot 10^{18}\mathrm{cm}^{-3}$ at $0.2\,\mathrm{V}$ gate voltage.

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 6.4.15.72 confirms that the 3D results are in agreement with the 2D results.

Last update: 17/07/2024

— DEV — Ultrathin-body Double Gate Metal Oxide Semiconductor Field Effect Transistor (DG MOSFET) (5 nm)

Attention: This tutorial is under construction

Input files:

- *DG-MOSFET-5nm_Si_Birner_APPA_2006_2D_cl_nnp.in*
- *DG-MOSFET-5nm_Si_Birner_APPA_2006_2D_qm_nnp.in*

Scope:

This tutorial compares classical and quantum mechanical simulations of a Double Gate MOSFET (DG MOSFET). This tutorial is related to the following publication: *[BirnerAPhys2006]*.

Output files:

- *bias_xxxxx\density_electron.dat*
- *bias_xxxxx\bandeges.dat*

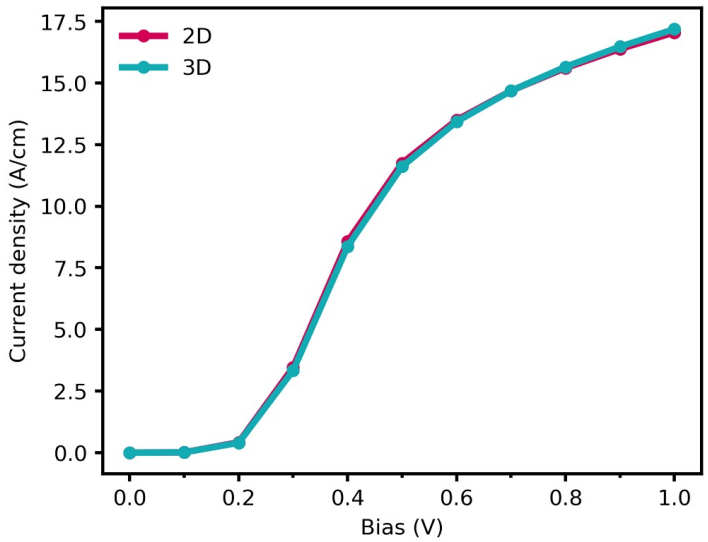


Figure 6.4.15.72: Comparison of the current-voltage characteristics between 2D and 3D simulations.

- *IV_characteristics.dat*

Structure

The main idea of a Double Gate 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 6.4.15.73. 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 $\epsilon = 3.9$.

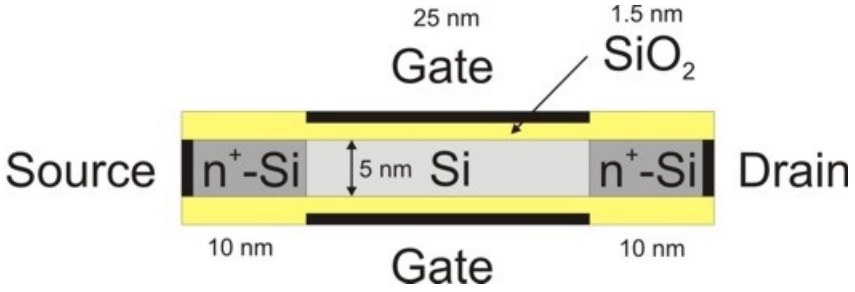


Figure 6.4.15.73: Geometry of the DG MOSFET, which consists of source contact, n-type doped source region (Si), Si channel (undoped), n-type doped drain region (Si), drain contact, SiO_2 insulator, top gate and bottom gate.

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.

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 6.4.15.74 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:

- 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-5nm_Si_Birner_APPA_2006_2D_cl_nnp.in* should be used.
- 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-5nm_Si_Birner_APPA_2006_2D_qm_nnp.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 $\mathrm{Si}/\mathrm{SiO}_2$ interface, because $E_{\mathrm{F,n}} - E_{\mathrm{C}}$ has its maximum there. The quantum mechanical density is practically zero at the $\mathrm{Si}/\mathrm{SiO}_2$ 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 $\mathrm{Si}/\mathrm{SiO}_2$ interfaces.

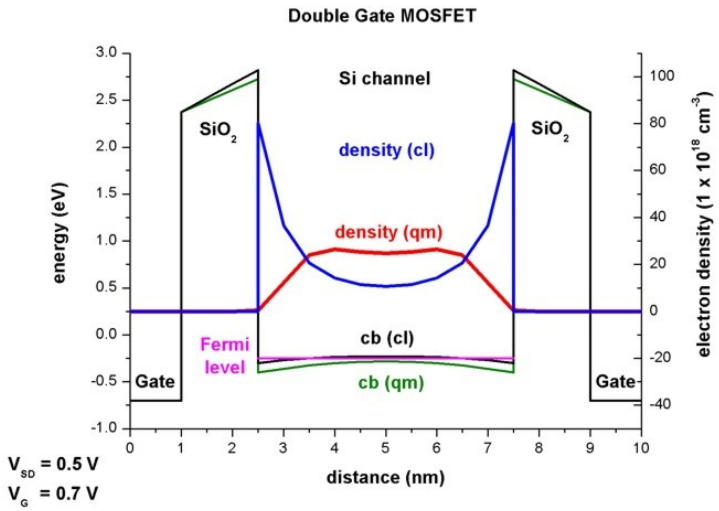


Figure 6.4.15.74: Conduction band profile, electron density and Fermi energy across the DB MOSFET structure at gate voltage $V_{\mathrm{G}} = 0.7\,\mathrm{V}$.

Figure 6.4.15.75 and Figure 6.4.15.76 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 $H(x, y)$:

$$
\begin{aligned}
&1. \quad m_{xx} = m_{\mathrm{longitudinal}} \quad \text{and} \quad m_{yy} = m_{zz} = m_{\mathrm{transversal}},\\
&2. \quad m_{yy} = m_{\mathrm{longitudinal}} \quad \text{and} \quad m_{xx} = m_{zz} = m_{\mathrm{transversal}},\\
&3. \quad m_{zz} = m_{\mathrm{longitudinal}} \quad \text{and} \quad m_{xx} = m_{yy} = m_{\mathrm{transversal}},
\end{aligned}
$$

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 ($k_{\parallel}$ dispersion). The quantum mechanical

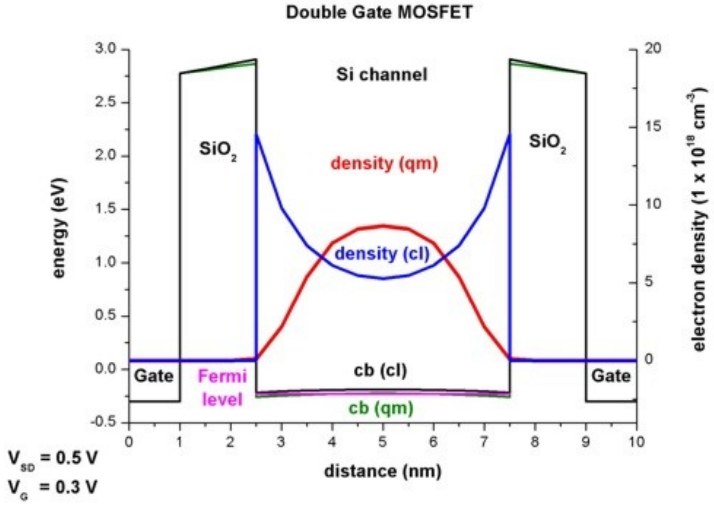


Figure 6.4.15.75: Conduction band profile, electron density and Fermi energy across the DB MOSFET structure at gate voltage $V_{\mathrm{G}} = 0.3\,\mathrm{V}$.

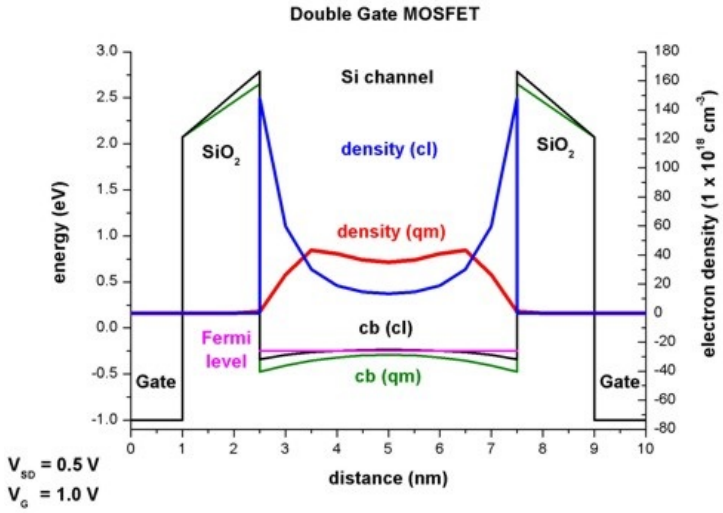


Figure 6.4.15.76: Conduction band profile, electron density and Fermi energy across the DB MOSFET structure at gate voltage $V_{\mathrm{G}} = 1\,\mathrm{V}$.

density for such a two-dimensional simulation is proportional to the square root of $m_{zz}(x,y)$. More precisely, the 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 6.4.15.77 and Figure 6.4.15.78, 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: (Note: They are sorted by energies, but their distance is not equivalent to their energy spacing.)

- deg1: 35^{th} state with $E_{35} = -0.215\,\mathrm{eV}$ (ψ^2 is zero at the 1D slice which can be seen in Figure 6.4.15.78)
- deg1: 32^{nd} state with $E_{32} = -0.224\,\mathrm{eV}$ (25 meV above Fermi level)
- deg3: 13^{th} state with $E_{13} = -0.226\,\mathrm{eV}$ (23 meV above Fermi level)
- deg2: 32^{nd} state with $E_{32} = -0.250\,\mathrm{eV}$ (below Fermi level, corresponding to 2^{nd} subband)
- deg2: 25^{th} state with $E_{25} = -0.277\,\mathrm{eV}$ (below Fermi level, corresponding to 1^{st} subband)

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).

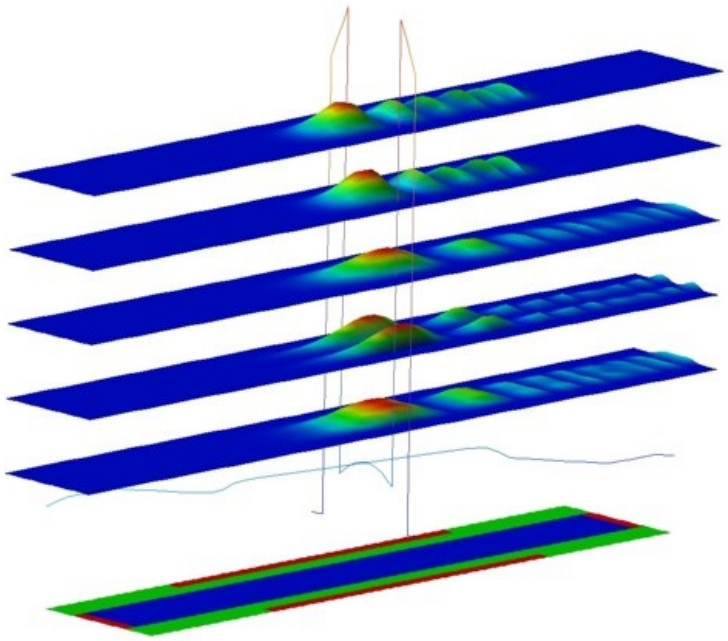


Figure 6.4.15.77: Wavefunctions located inside the Si-channel for $V_{\mathrm{G}} = 0.7\,\mathrm{V}$ and $V_{\mathrm{SD}} = 0.5\,\mathrm{V}$.

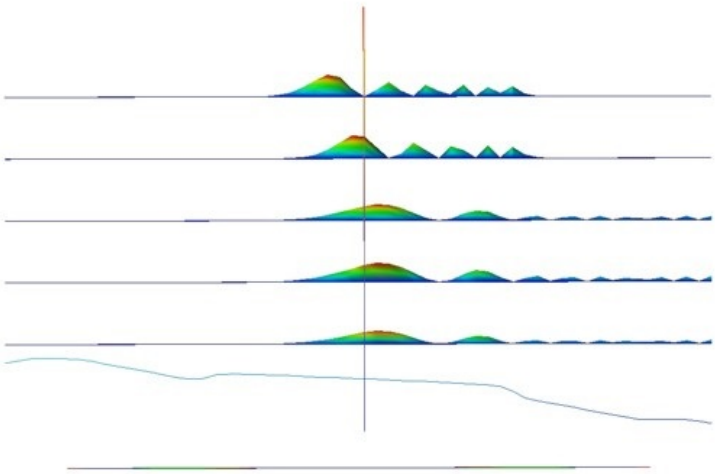


Figure 6.4.15.78: Wavefunctions located inside the Si-channel for $V_{\mathrm{G}} = 0.7\,\mathrm{V}$ and $V_{\mathrm{SD}} = 0.5\,\mathrm{V}$ (side-view).

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 6.4.15.79. Due to the influence of quantum mechanics the current densities obtained from the quantum mechanical calculations are lower than from the classical calculations.

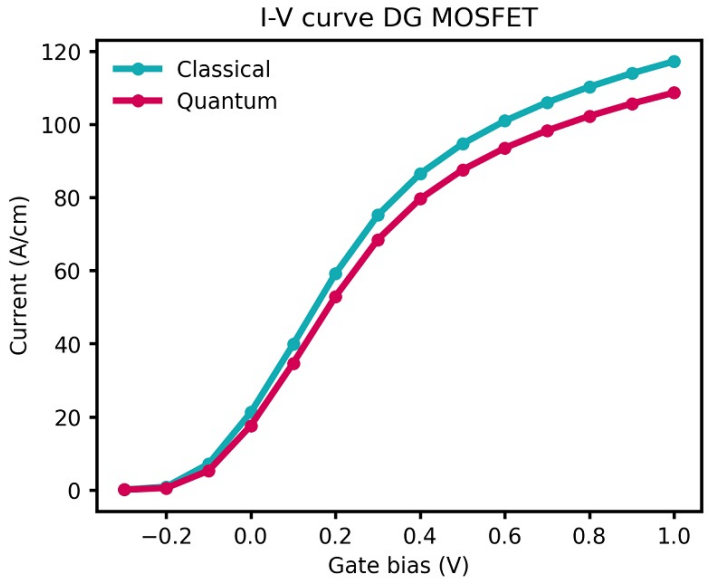


Figure 6.4.15.79: Comparison between classical and quantum mechanical calculation of the I-V characteristics.

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: 17/07/2024

6.4.16 Magnetic Effects

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.in/*_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*

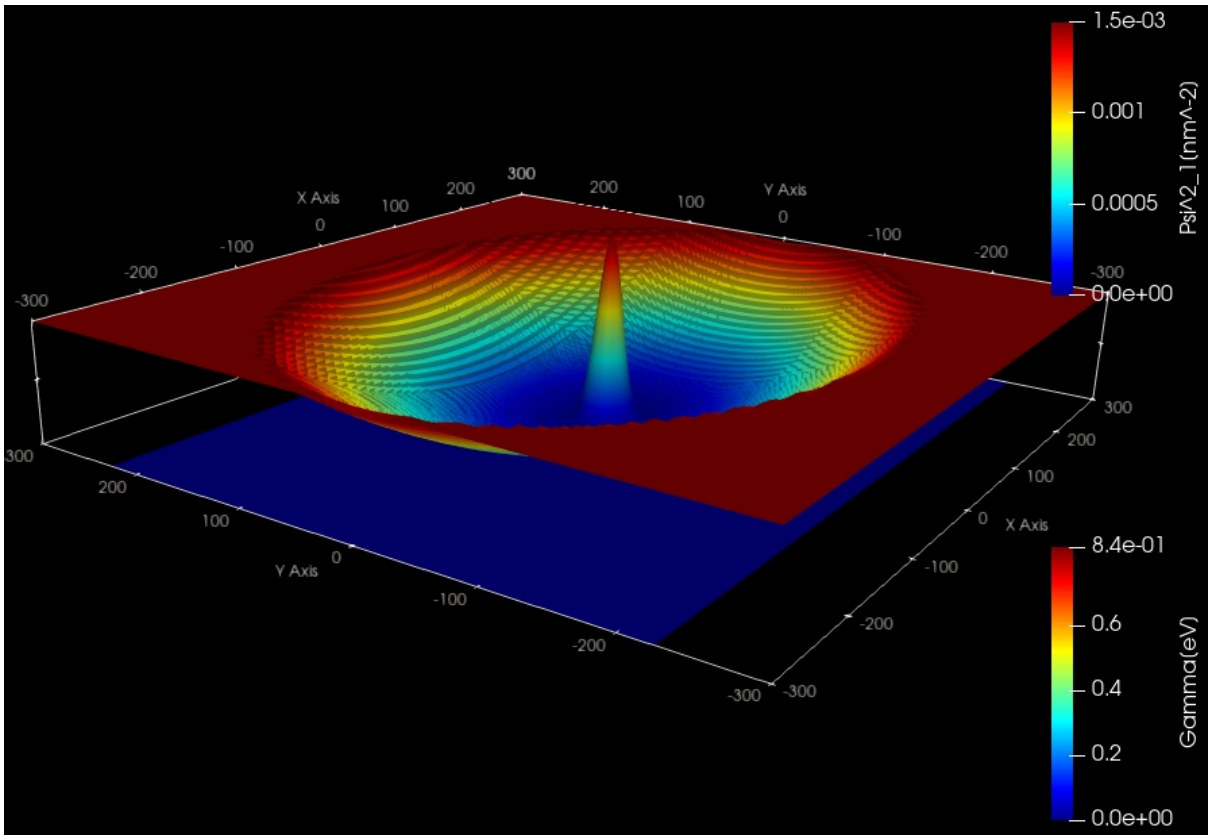
The electron effective mass in GaAs is $m_e^* = 0.067 m_0$. We assume this value for the effective mass in the whole region (i.e. also inside the AlGaAs alloy).

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:

$$E_{n_x,n_y} = (n_x + \frac{1}{2})\hbar\omega_x + (n_y + \omega\frac{1}{2})\hbar\omega_y n_x = n + \frac{1}{2}|l| - \frac{1}{2}|l| n_y = n + \frac{1}{2}|l| + \frac{1}{2}|l|$$
$$for\ n = 0, 1, 2, 3, \ldots\ \ l = 0, \pm1, \pm2, \ldots$$

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

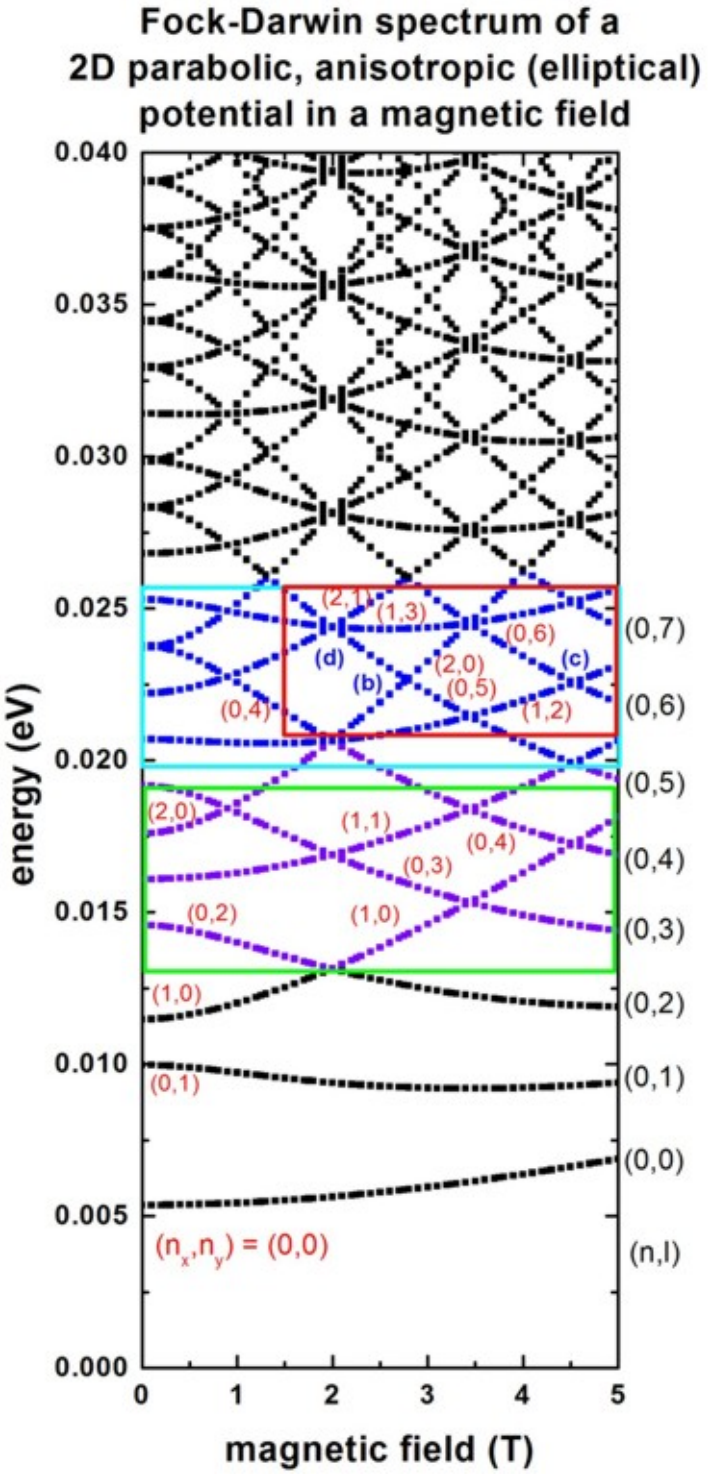
Probing by transport the single-particle energy spectrum up to high energy of one quantum dot with the ground state of an adjacent weakly coupled quantum dot
D.G. Austing, G. Yu, C. Payette, J.A. Gupta, M. Korkusinski, G.C. Aers
physica status solidi (a), 508 (2007)

(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.)

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.

Last update: nn/nn/nnnn

Fock-Darwin states of parabolic, isotropic potential in a magnetic field


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)
(c)
(2,0)
(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)

Header

Input files

- *2DGaAs_BiParabolicQW_4meV_GovernalePRB1998.in/*_nnp.in*
- *2DGaAs_BiParabolicQW_3meV_FockDarwin.in/*_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. On the other hand, the effect of spin is not considered in *nextnano³*. Thus the number of eigenstates calculated in this *nextnano++* input file is two times of the one in the *nextnano³* input file. 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++* doesn't 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
    }
}
```

In *nextnano*[3], the magnetic field sweeping can be specified in the input file:

```
$magnetic-field
 magnetic-field-on                    = yes
 magnetic-field-strength              = 0.0d0   ! 1 Tesla = 1 Vs/m2
 magnetic-field-direction             = 0 0 1   ! [001] direction
 magnetic-field-sweep-active          = yes     !
 magnetic-field-sweep-step-size       = 0.5d0   ! 0.5 Tesla = 0.5 Vs/m2
 magnetic-field-sweep-number-of-steps = 40      ! 40 magnetic field sweep steps
$end_magnetic-field
```

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 $m_e^* = 0.067 m_0$. 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:

$$\omega_c = \frac{|e|B}{m_e^*}$$

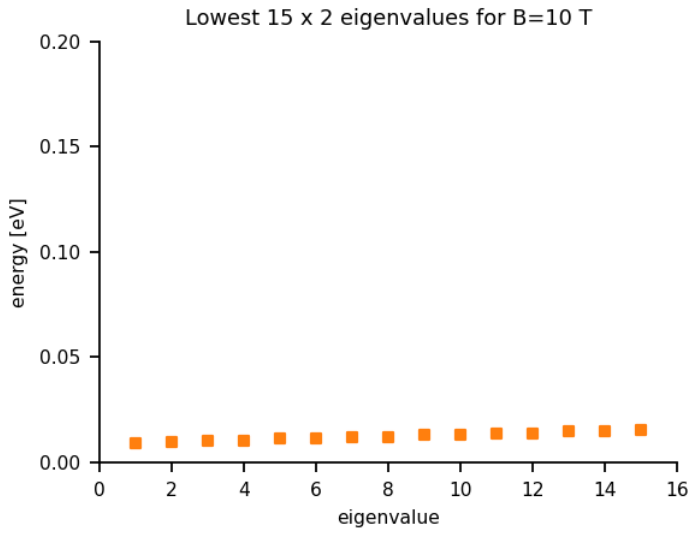
Thus for the electrons in GaAs, where $m_e^* = 0.067 m_0$, it holds for the different magnetic field strengths:

- 1 T: $\hbar\omega_c = 1.7279$ meV
- 2 T: $\hbar\omega_c = 3.4558$ meV
- 3 T: $\hbar\omega_c = 5.1836$ meV
- ...
- 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) in *nextnano++*. The corresponding eigenvalue calculated in *nextnano*[3] is $E_1 = 9.44$ meV.

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.

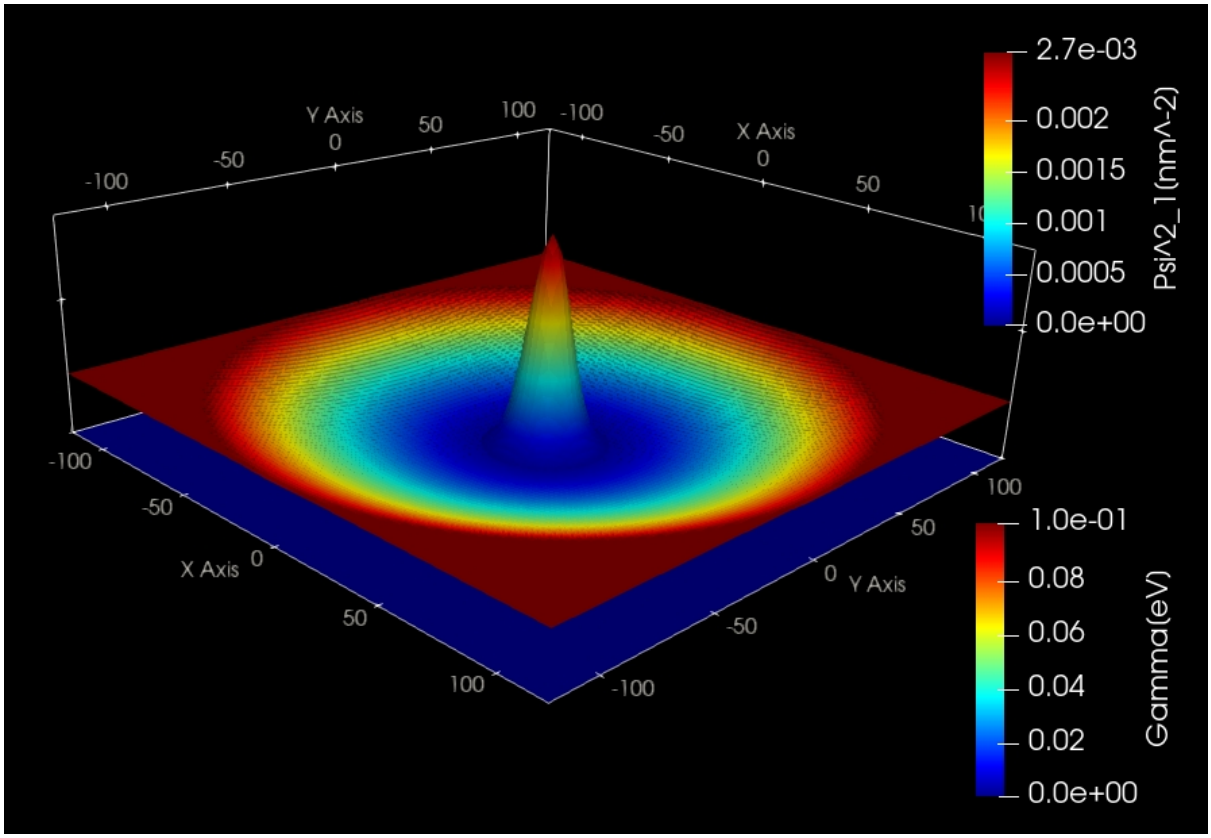
The following figure shows the probability density (ψ^2) of the 14th excited state (up-spin) (i.e. $E_{15,\uparrow}$) for a magnetic field magnitude of $B = 10$ T. It is in perfect agreement with Fig. 3(a) of *[GovernalePRB1998]*. 14th excited states have the energy $E_{15,\uparrow} = 21.71$ and $E_{15,\downarrow} = 21.88$ meV (at $B = 10$ T). The corresponding eigenvalue calculated in *nextnano*[3] is $E_{15} = 21.72$ meV. The left, vertical axis shows ψ^2 in units of nm^{-2} (the peak value is 0.000283 nm^{-2}).

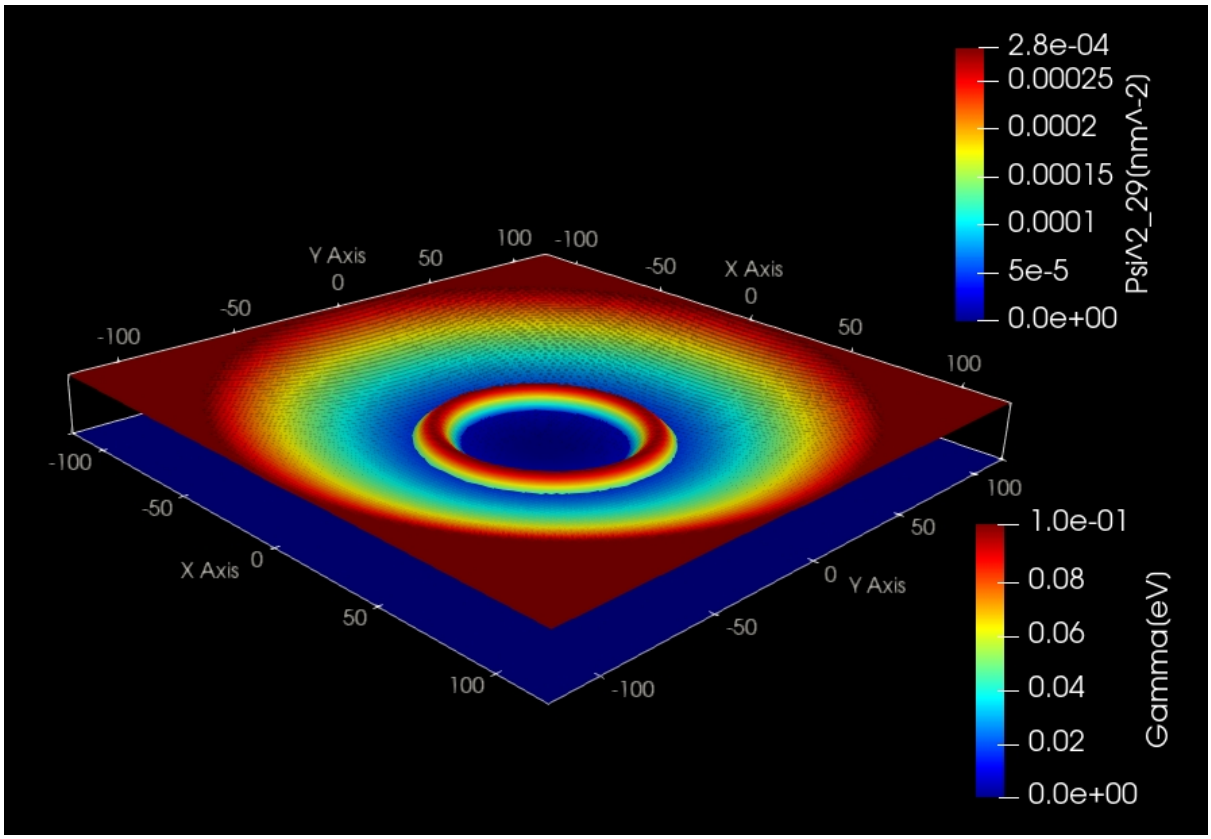
In the same figure, parabolic conduction band edge confinement potential is also shown. The above axis shows the colormap of the conduction band edge values. In the middle of the sample the conduction band edge is 0 eV, and at the boundary region, the conduction band edge has the value 0.1014 eV.

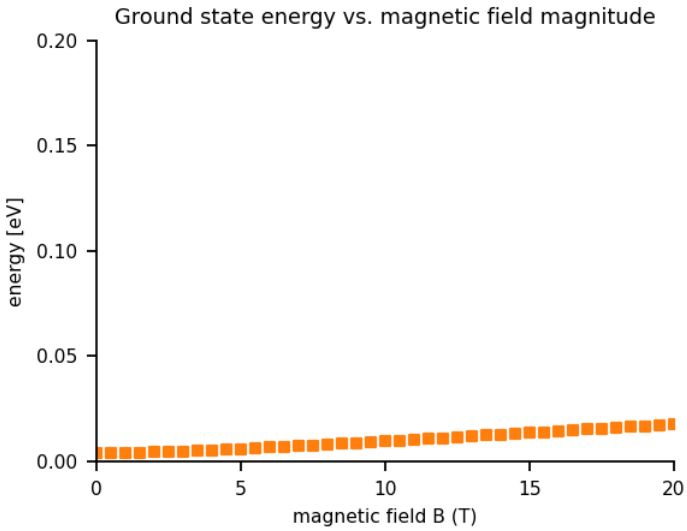
Ground state energy vs. magnetic field magnitude

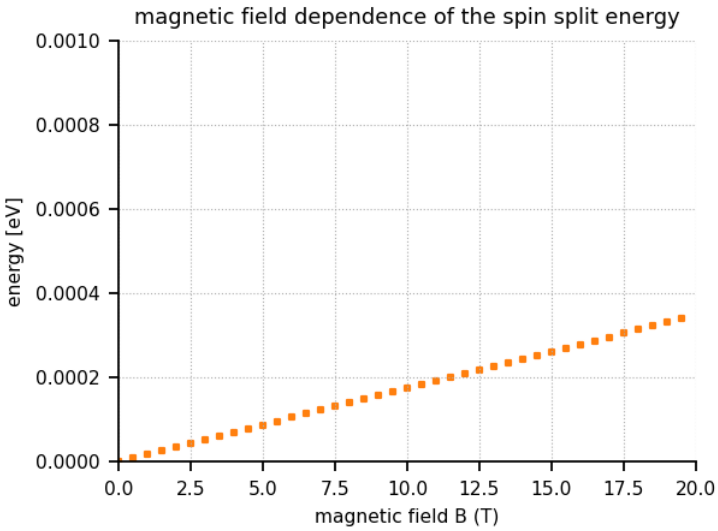
The following figure shows the ground state energy as a function of magnetic field magnitude. It is in perfect agreement with Fig.4 of *[GovernalePRB1998]*. The ground state has the energy $E_1 = 4.04$ meV (spin-degenerated) in *nextnano++* and $E_1 = 4.01$ meV in *nextnano*[3] at $B = 0$ T.

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].


2.7e-03
0.002
0.0015
0.001
0.0005
0.0e+00
Psi^2_1(nm^-2)
Y Axis
X Axis
1.0e-01
0.08
0.06
0.04
0.02
0.0e+00
Gamma(eV)


2.8e-04
0.00025
0.0002
0.00015
0.0001
5e-5
0.0e+00
Psi^2_29(nm^-2)
Y Axis
X Axis
1.0e-01
0.08
0.06
0.04
0.02
0.0e+00
Gamma(eV)


Ground state energy vs. magnetic field magnitude
0.20
0.15
0.10
0.05
0.00
energy [eV]
0
5
10
15
20
magnetic field B (T)


magnetic field dependence of the spin split energy
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
energy [eV]
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
magnetic field B (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 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 $Al_xGa_{1-x}As$ alloy that has a parabolic alloy profile in the (x,y) plane. This is chosen so that the eclectron ground state has the energy: $E_1 = \hbar\omega_0 = 3$ 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]".

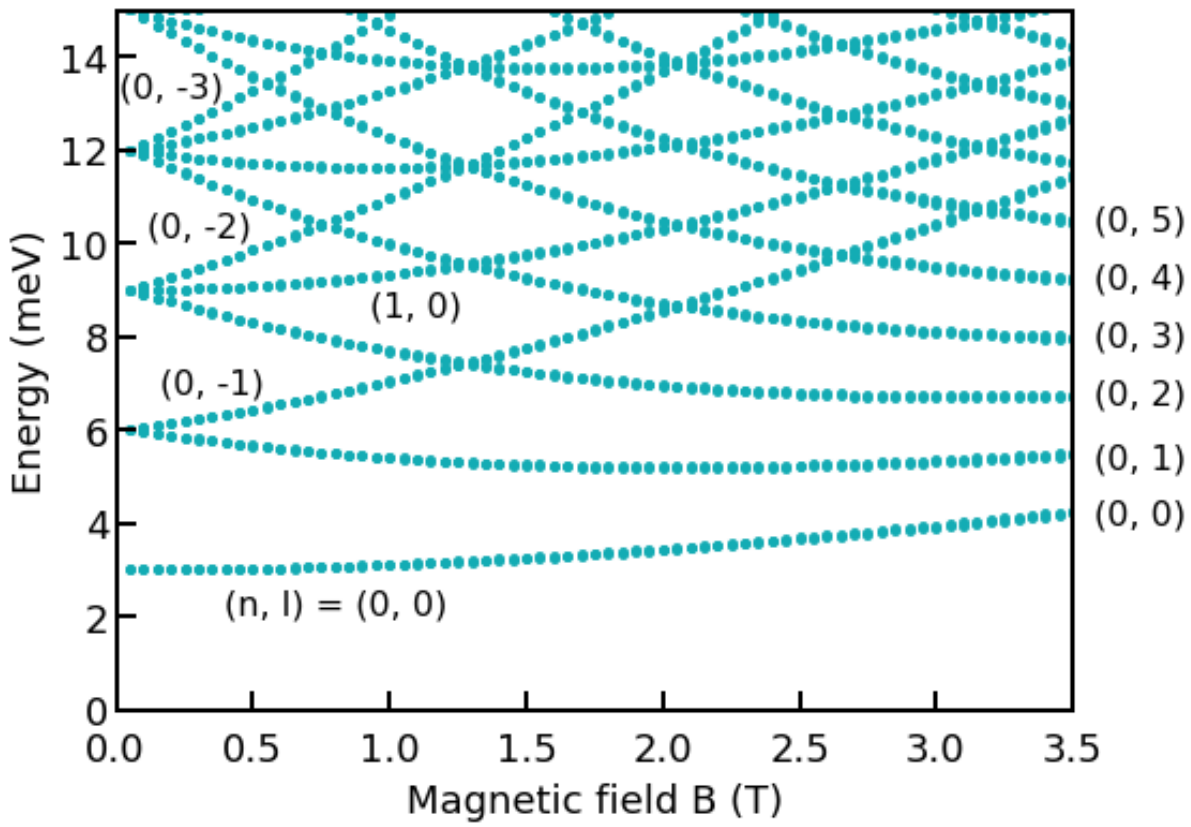
The figures are in excellent agreement with Fig. 6(a) of *[KouwenhovenRPP2001]*.

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).



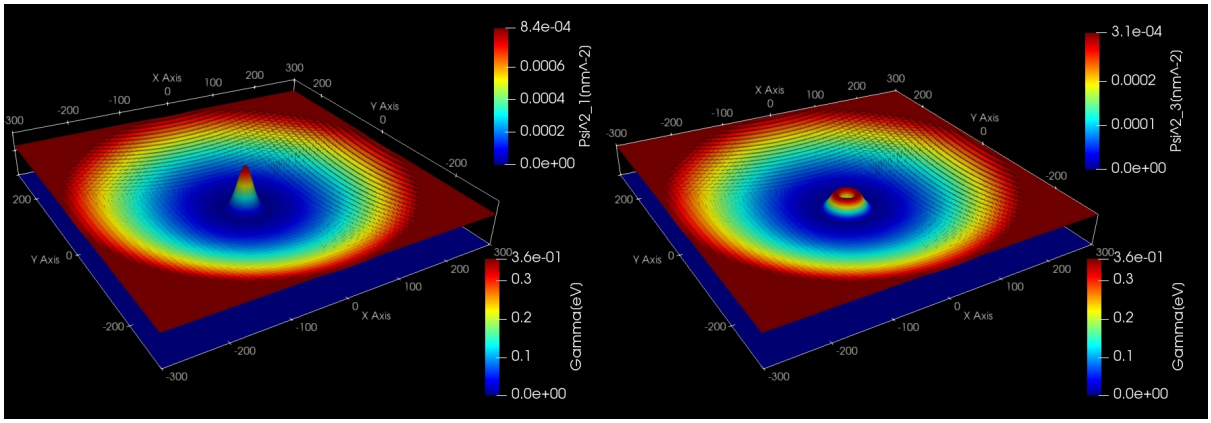


Figure 6.4.16.1: left: $(n, l) = (0, 0)$ (1st), right: $(n, l) = (0, 1)$ (2nd)

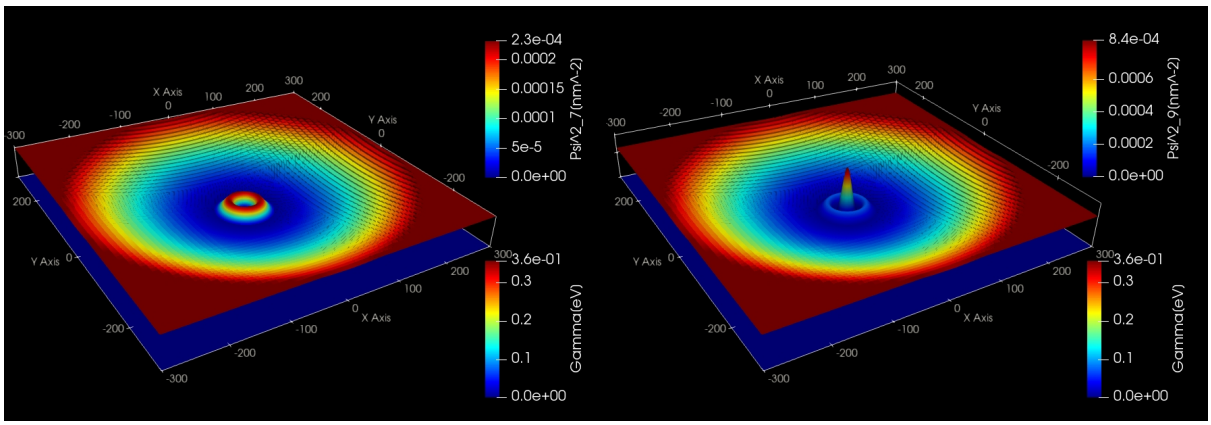


Figure 6.4.16.2: left: $(n, l) = (0, 2)$ (4th), right: $(n, l) = (1, 0)$ (5th)

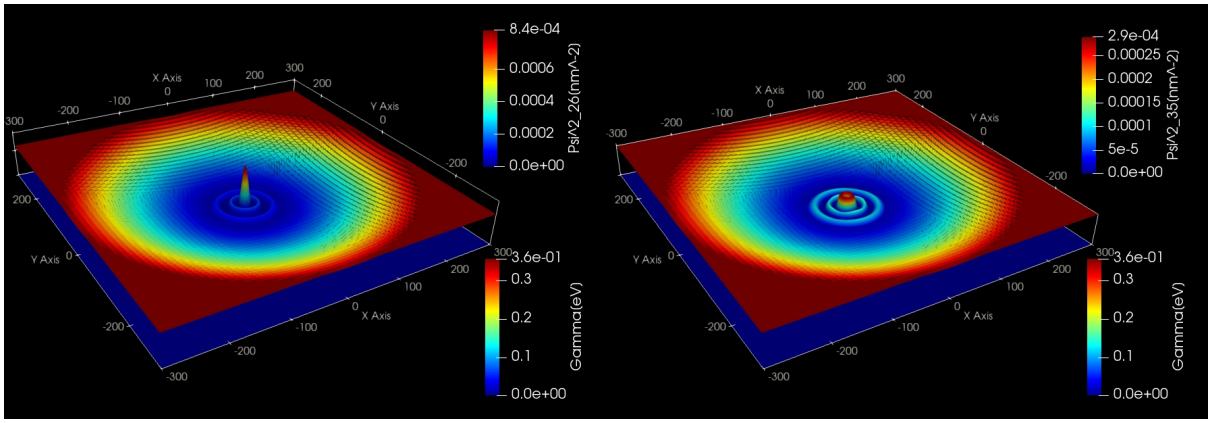
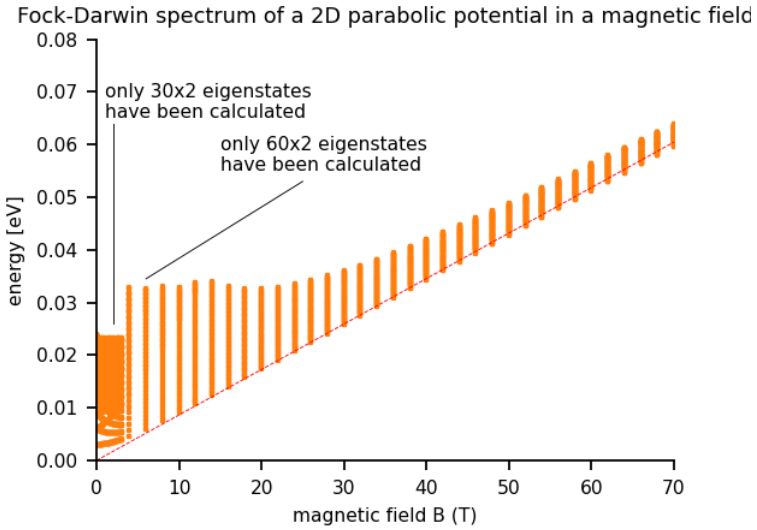


Figure 6.4.16.3: left: $(n, l) = (2, 0)$ (13th), right: $(n, l) = (2, 2)$ (18th)



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).

Last update: nn/nn/nnnn

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 $\times$ 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-field

 magnetic-field-on                    = yes
 magnetic-field-strength              = 0.0     ! 0 Tesla = 0 Vs/m2
 magnetic-field-direction             = 0 0 1   ! [001] direction

 magnetic-field-sweep-active          = yes     !
 magnetic-field-sweep-step-size       = 1.0     ! 1 Tesla = 1 Vs/m2
 magnetic-field-sweep-number-of-steps = 3       ! 3 magnetic field sweep steps

 output-magnetic-vector-potential     = yes     ! output of A(x,y,z)
$end_magnetic-field
```

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:

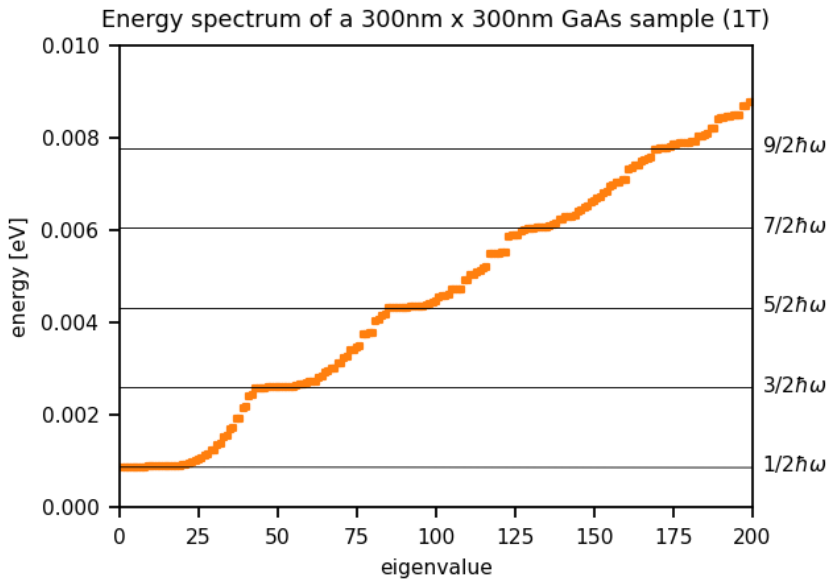
$$\omega_c = \frac{|e|B}{m_e^*}$$

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:

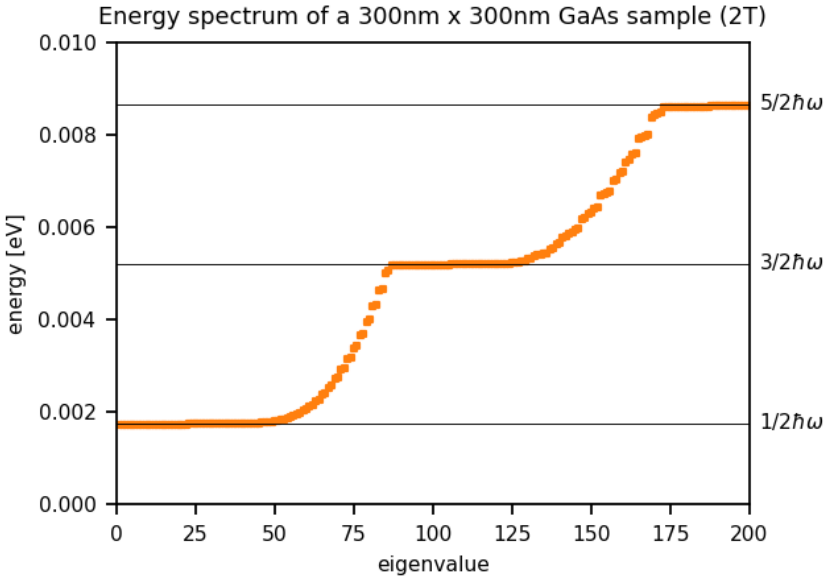


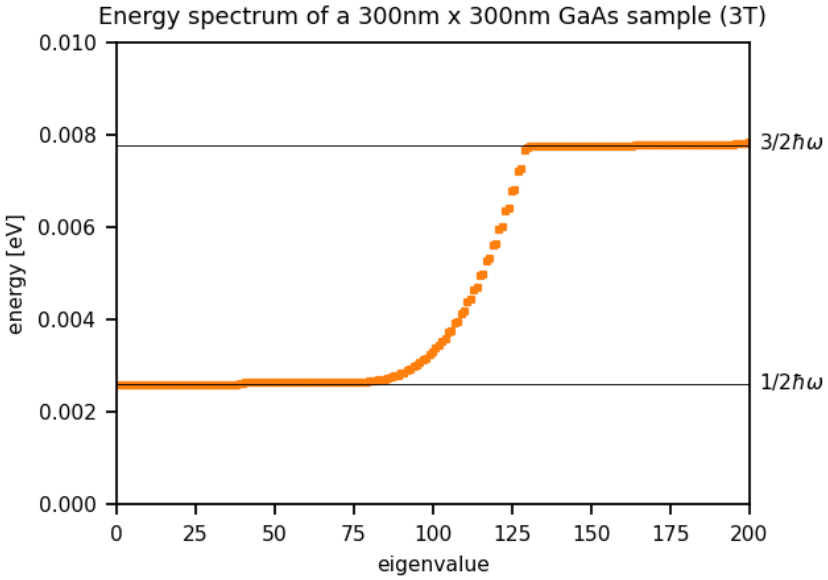
Landau levels

The Landau levels are analytically given by

$$E_n = \left(n - \frac{1}{2}\right)\hbar\omega_c$$

where $n = 1, 2, 3, ...$


Energy spectrum of a 300nm x 300nm GaAs sample (2T)
energy [eV]
eigenvalue
0.010
0.008
0.006
0.004
0.002
0.000
0
25
50
75
100
125
150
175
200
5/2ħω
3/2ħω
1/2ħω


Energy spectrum of a 300nm x 300nm GaAs sample (3T)
energy [eV]
eigenvalue
0.010
0.008
0.006
0.004
0.002
0.000
0
25
50
75
100
125
150
175
200
3/2ħω
1/2ħω

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).

Last update: nn/nn/nnnn

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 6.4.16.4. 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.

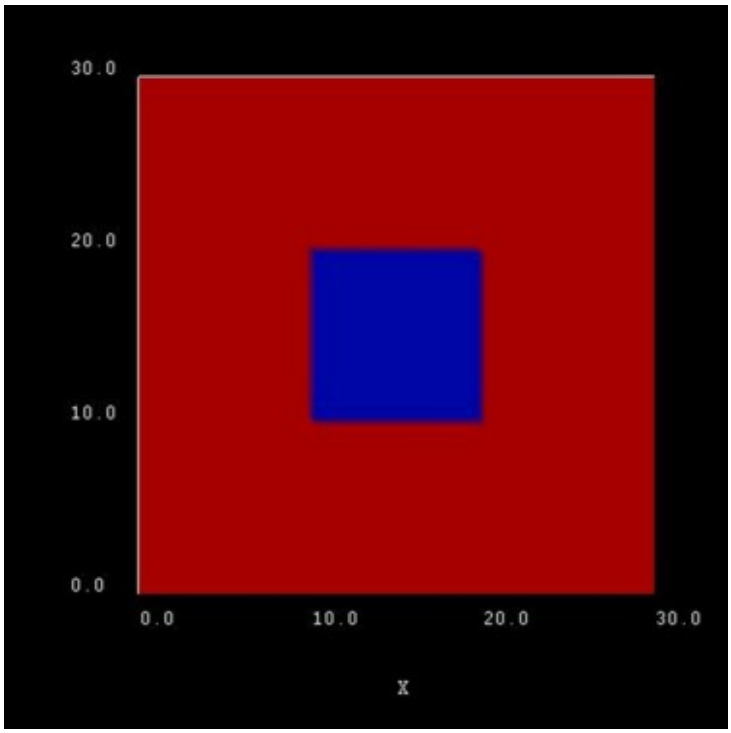


Figure 6.4.16.4: Simulated quantum wire (blue region) consisting of InAs surrounded by GaAs (red).

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 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):

$$E_{n_1,n_2} = \frac{\hbar^2 \pi^2}{2 m^*_{\mathrm{hh}}} \left(\frac{n_1^2}{L_x^2} + \frac{n_2^2}{L_y^2} \right) = -9.17\,\mathrm{meV} \cdot \left(n_1^2 + n_2^2 \right),$$

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

$$E_{\mathrm{hh}} = E_{n_1,n_2} + \frac{\hbar^2 k_z^2}{2 m^*_{hh}},$$

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.

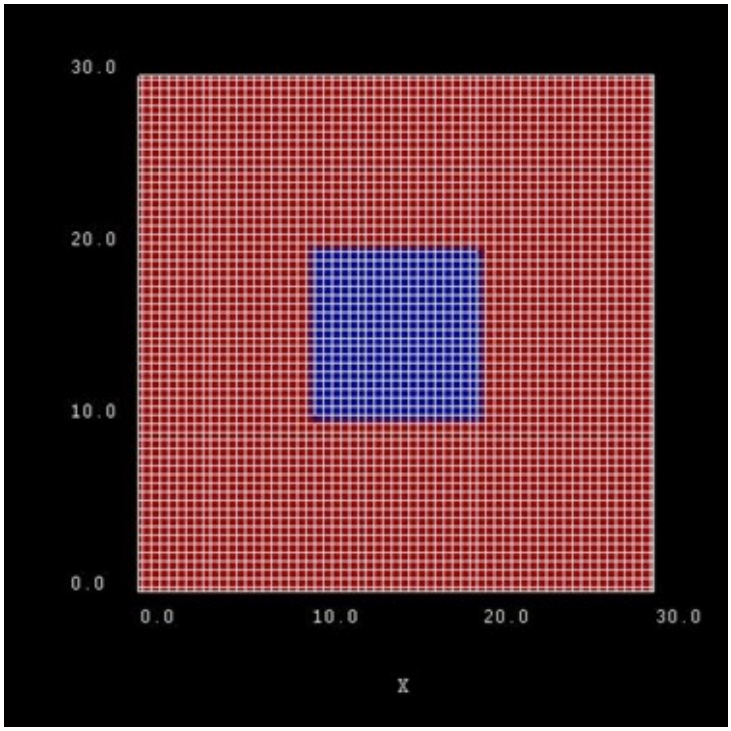


Figure 6.4.16.5: Possible configuration of rectangular grid lines. Here, the grid spacing is 0.5 nm, thus the quantum wire (blue area) consists of 21 x 21 = 400 grid points.

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:

$$
\begin{aligned}
E_{11} &= -0.018343\,\mathrm{eV} \quad (\mathrm{groundstate}) \\
E_{12} = E_{21} &= -0.045857\,\mathrm{eV} \\
E_{12} = E_{21} &= -0.045857\,\mathrm{eV} \\
E_{22} &= -0.073372\,\mathrm{eV} \\
E_{13} = E_{31} &= -0.091715\,\mathrm{eV} \\
E_{23} = E_{32} &= -0.119229\,\mathrm{eV} \\
E_{14} = E_{41} &= -0.155915\,\mathrm{eV} \\
&\dots \\
E_{18} = E_{81} = E_{47} = E_{74} &= -0.596145\,\mathrm{eV} \quad (\mathrm{Here,\ the\ degeneracy\ is\ a\ coincidence.})
\end{aligned}
$$

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:

$$
\begin{aligned}
E_{11} &= 0.018341\,\text{eV} \\
E_{12} &= -0.045845\,\text{eV} \quad (\text{two}-\text{fold degenerate}) \\
E_{21} &= -0.045845\,\text{eV} \quad (\text{two}-\text{fold degenerate}) \\
E_{22} &= -0.073348\,\text{eV} \\
E_{13} &= -0.091653\,\text{eV} \quad (\text{two}-\text{fold degenerate}) \\
E_{31} &= -0.091653\,\text{eV} \quad (\text{two}-\text{fold degenerate}) \\
E_{23} &= -0.119156\,\text{eV} \quad (\text{two}-\text{fold degenerate}) \\
E_{32} &= -0.119156\,\text{eV} \quad (\text{two}-\text{fold degenerate}) \\
E_{14} &= -0.155721\,\text{eV} \quad (\text{two}-\text{fold degenerate}) \\
E_{41} &= -0.155721\,\text{eV} \quad (\text{two}-\text{fold degenerate})
\end{aligned}
$$

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.

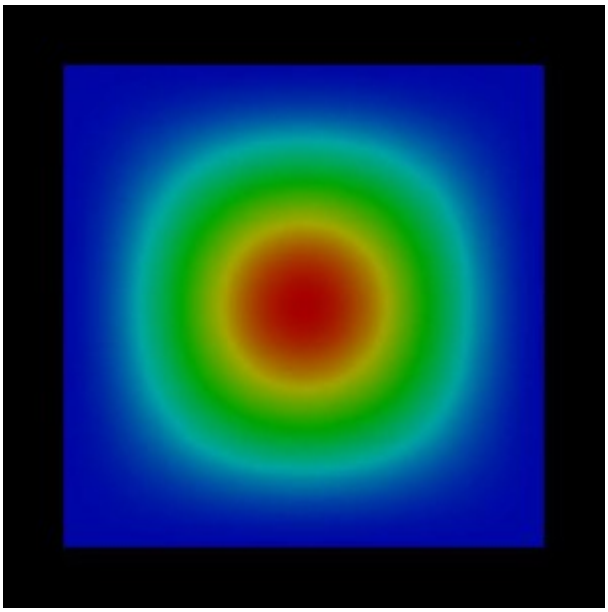


Figure 6.4.16.6: Probability density $\psi_{11}(x,y)^2$ of the 1^{st} heavy hole state.

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.

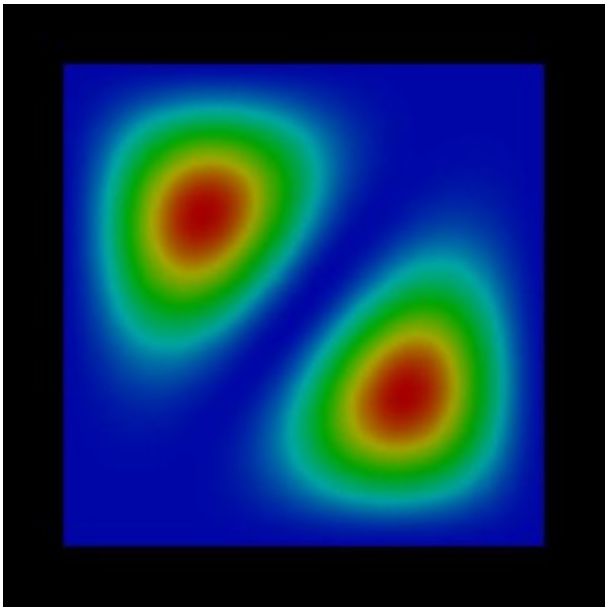


Figure 6.4.16.7: Probability density $\psi_{12}(x,y)^2$ of the 2^{nd} heavy hole state.

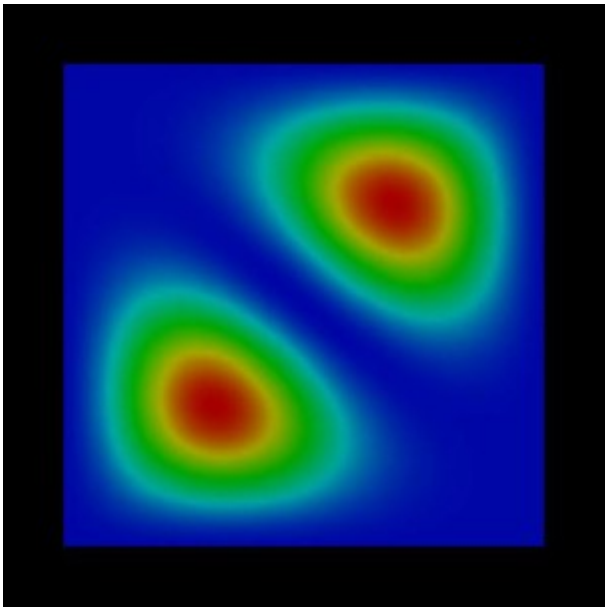


Figure 6.4.16.8: Probability density $\psi_{21}(x,y)^2$ of the 3^{rd} heavy hole state.

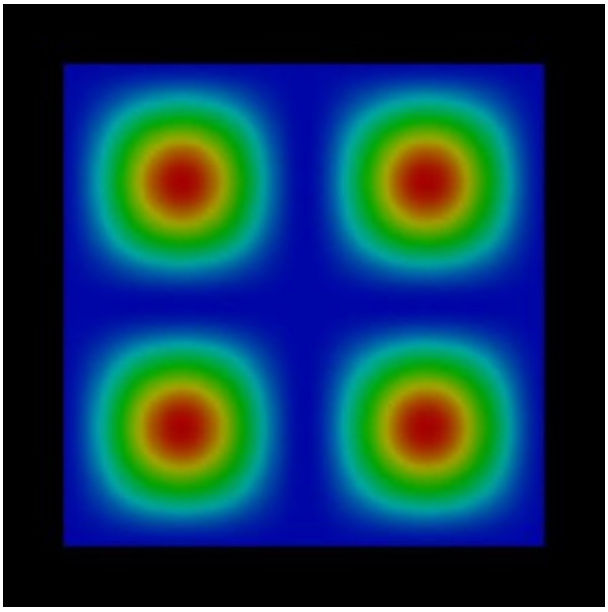


Figure 6.4.16.9: Probability density $\psi_{22}(x,y)^2$ of the 4^{th} heavy hole state.

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]
```

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.

In Figure 6.4.16.14, the probability density of the 2^{nd} eigenstate is plotted from a different perspective.

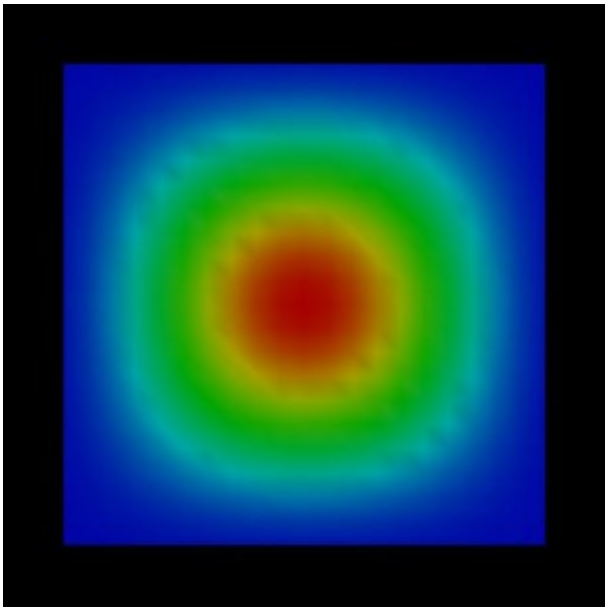


Figure 6.4.16.10: Probability density $\psi_{11}(x,y)^2$ of the 1^{st} heavy hole state with magnetic field applied.

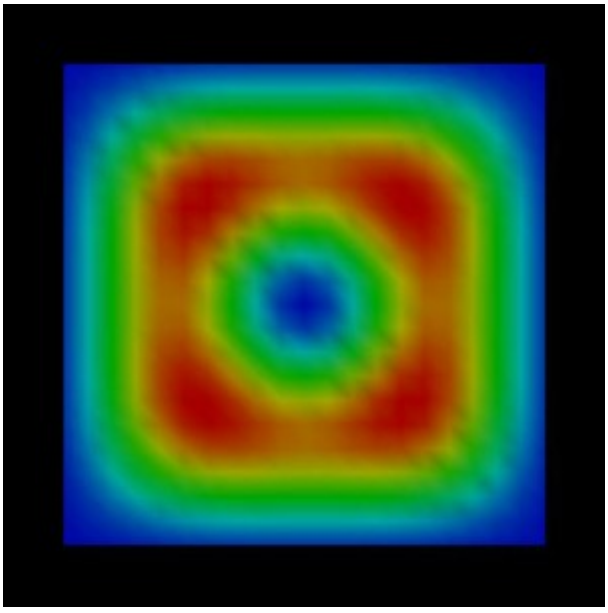


Figure 6.4.16.11: Probability density $\psi_{12}(x,y)^2$ of the 2^{nd} heavy hole state with magnetic field applied.

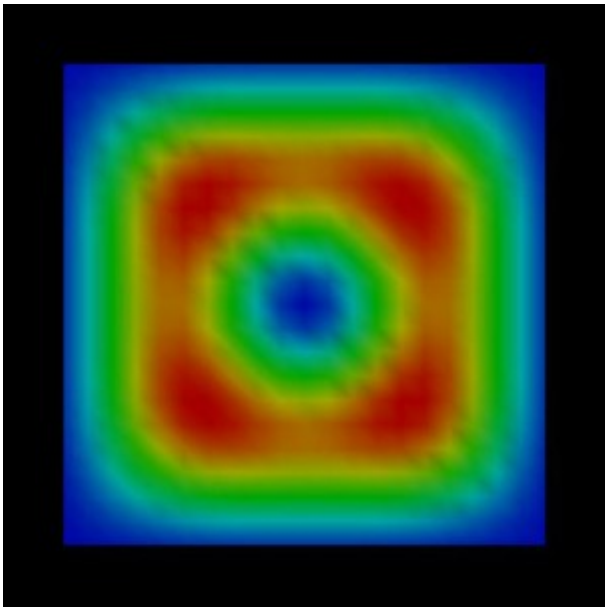


Figure 6.4.16.12: Probability density $\psi_{21}(x,y)^2$ of the 3^{rd} heavy hole state with magnetic field applied.

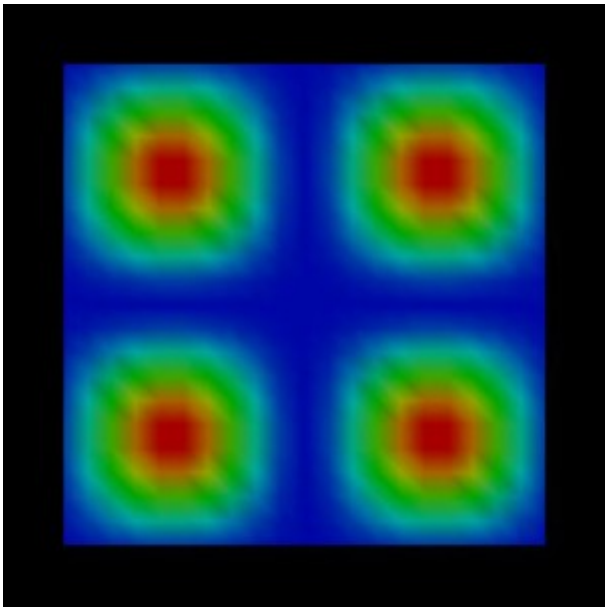


Figure 6.4.16.13: Probability density $\psi_{22}(x,y)^2$ of the 4^{th} heavy hole state with magnetic field applied.

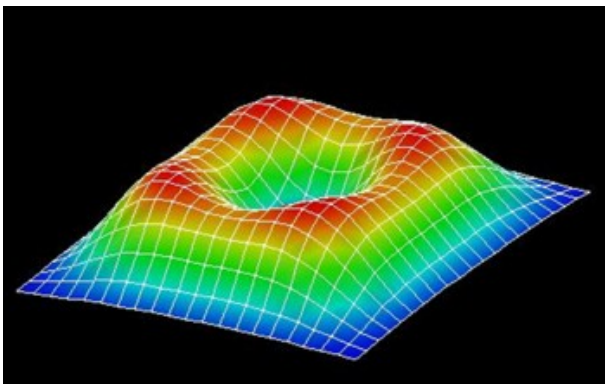


Figure 6.4.16.14: Probability density $\psi_{12}(x,y)^2$ of the 2^{nd} heavy hole state with magnetic field applied (viewed from a different perspective).

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.

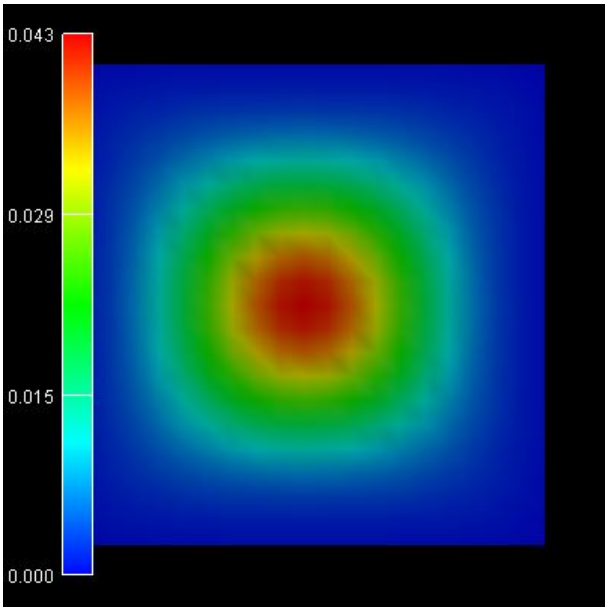


Figure 6.4.16.15: Probability density of the $1^{\mathrm{st}}/2^{\mathrm{nd}}$ heavy hole state with energy eigenvalue -0.0171 eV.

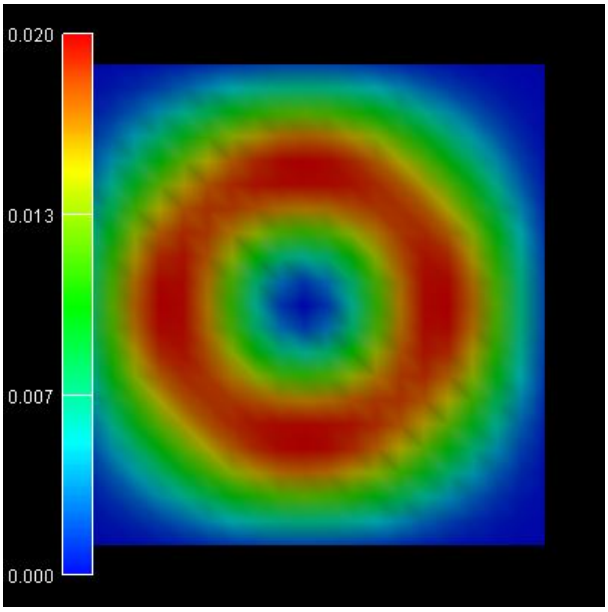


Figure 6.4.16.16: Probability density of the $3^{\mathrm{rd}}/4^{\mathrm{th}}$ heavy hole state with energy eigenvalue -0.0282 eV.

Last update: nn/nn/nnnn

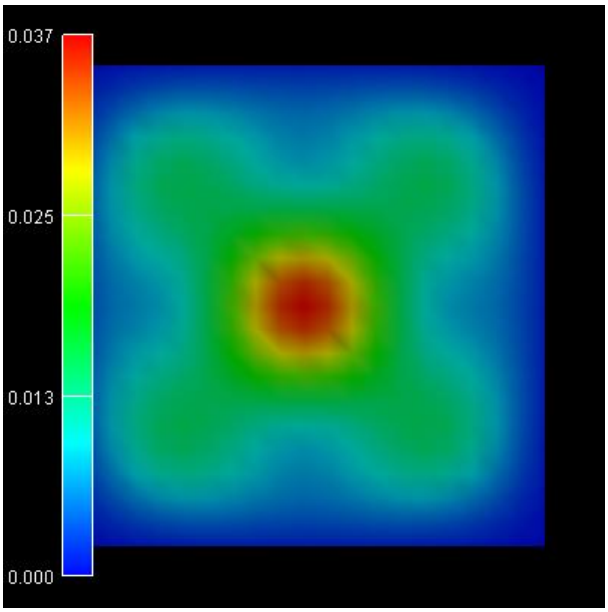


Figure 6.4.16.17: Probability density of the 5^{th}/6^{th} heavy hole state with energy eigenvalue -0.0294 eV.

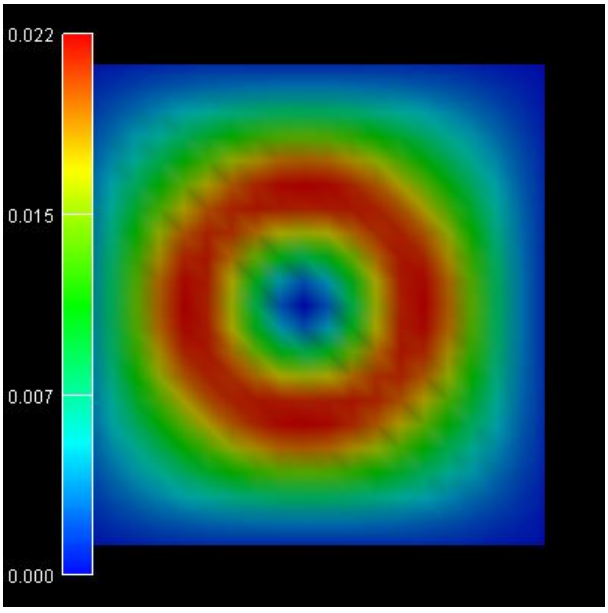


Figure 6.4.16.18: Probability density of the 7^{th}/8^{th} heavy hole state with energy eigenvalue -0.0367 eV.

— 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 ($\mathrm{Al_{0.32}Ga_{0.68}As}$) 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 $\mathrm{Al_{0.32}Ga_{0.68}As}$ tunnel barrier.

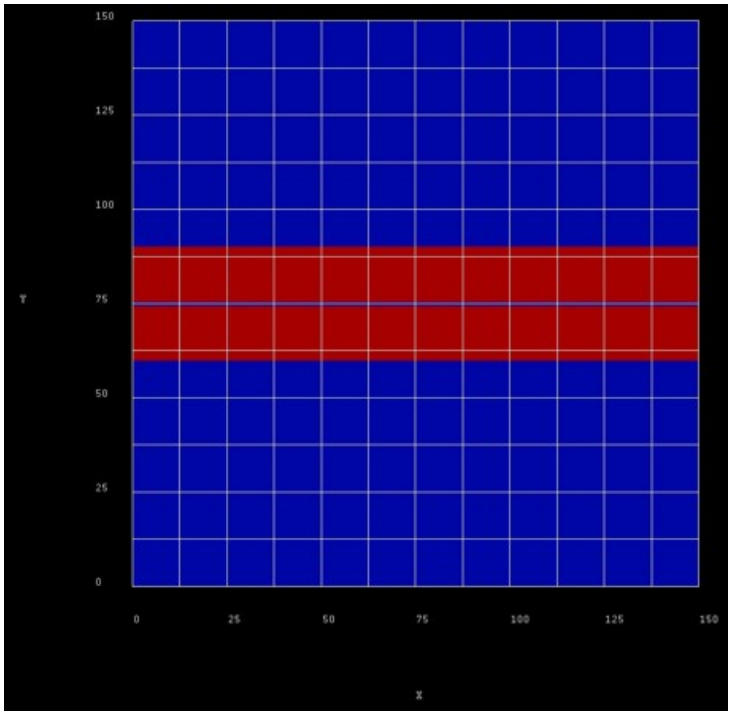


Figure 6.4.16.19: Quantum wire structure.

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]*

$$\Delta E_y = \hbar\omega_0 = \hbar\sqrt{\frac{2C}{m^*}}.$$

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 $\mathrm{Al_{0.32}Ga_{0.68}As}$: 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 ($0\,\mathrm{T}$, $0.5\,\mathrm{T}$, $1.0\,\mathrm{T}$, …, $16\,\mathrm{T}$).

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

6.4.17 Numerics

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 6.4.17.1 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.

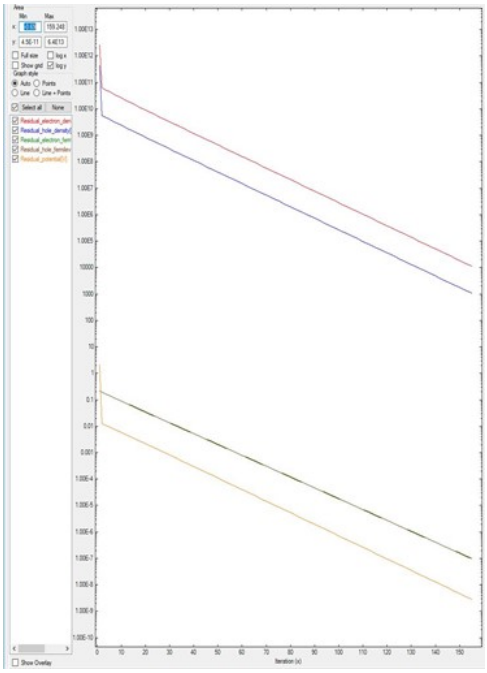


Figure 6.4.17.1: Residual evolution for a `system A` exhibiting quick convergence.

In contrast, Figure 6.4.17.2 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.

Sometimes the number of iterations is not enough to reach the convergence. Figure 6.4.17.3 and Figure 6.4.17.2 plot the results of the same `system B` but differ in their number of iterations. Figure 6.4.17.3 is simulated with only 150 iterations. As it was shown in Figure 6.4.17.2, only after 180 iterations the residuals start to decrease. Hence Figure 6.4.17.3 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 6.4.17.2 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

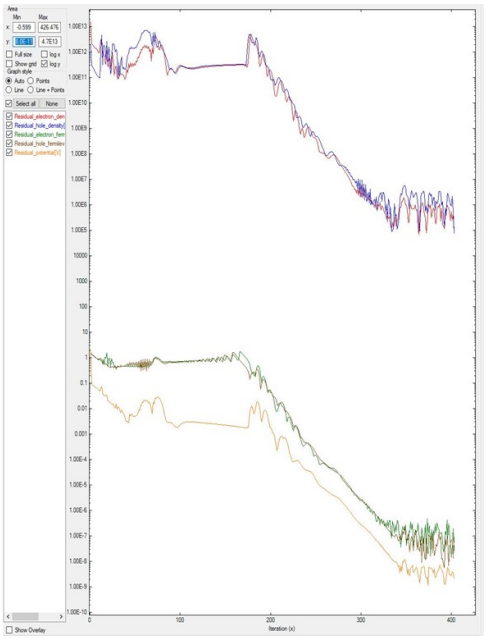


Figure 6.4.17.2: Residual evolution for a `system B` with slow convergence. In the input file were specified `residual_fermi = 10^-7 eV`, `residual (density) = 10^5 /cm^3`, and `iterations = 2000`. The `alpha_Fermi` parameter was set to 0.01.

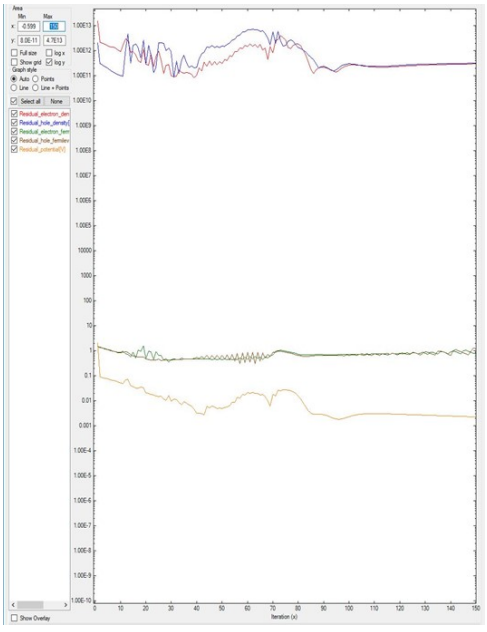


Figure 6.4.17.3: Residual evolution of `system B` with 150 iterations, exhibiting a pseudo-non-convergence behavior. Specifications in the input file: `residual_fermi = 10^-7 eV`, `residual` (density) of 10^5 /cm^3, and `iterations = 150`. The value of `alpha_Fermi` is 0.01.

Figure 6.4.17.4. In this example, `alpha_Fermi` was reduced from 0.01 (value used for Figure 6.4.17.2 and Figure 6.4.17.3) to 0.0001, and after 2000 iterations the system does not converge. Here we used the `system B` of the previous two images.

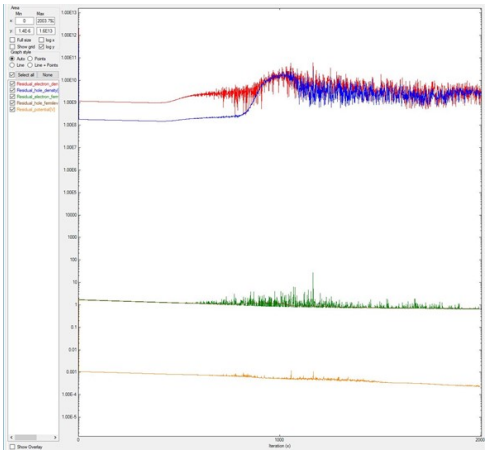


Figure 6.4.17.4: Residual evolution for a system exhibiting pseudo-non-convergence. Specifications in the input file: `residual_fermi =10^-7 eV`, `residual (density) = 10^5 /cm^3`, and `iterations = 2000`. The `alpha_Fermi` parameter was reduced to 0.0001.

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.

... 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 GmbH 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.

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 6.4.17.5 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.

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

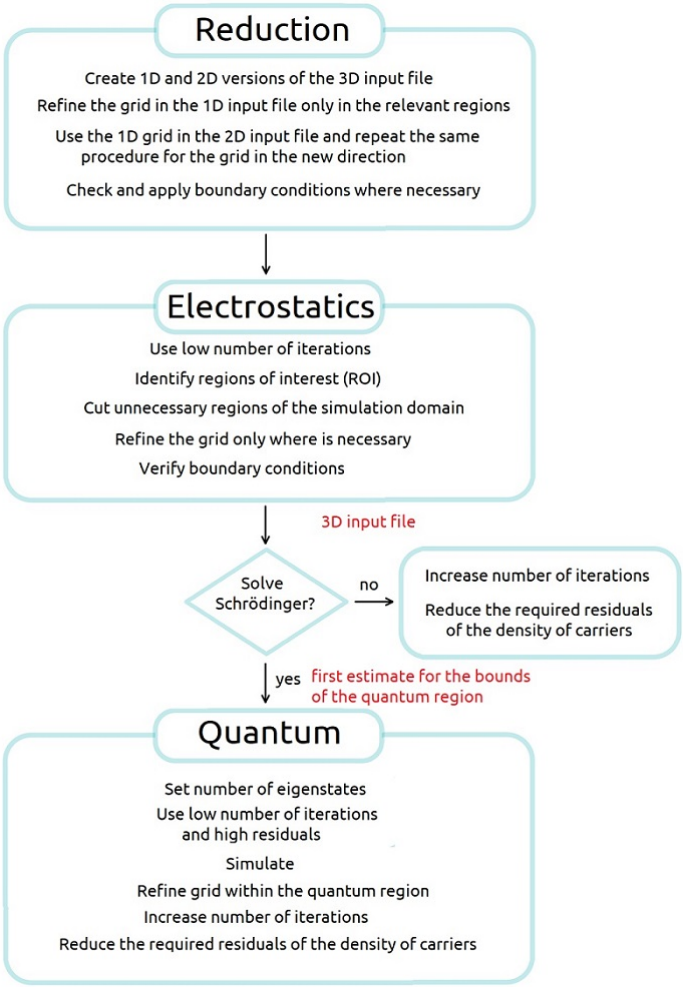


Figure 6.4.17.5: Methodology for 3D-simulation of large devices.

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:

— SOON — Reducing dimensionality of large 3D designs

— SOON — Optimizing electrostatics simulation for large 3D designs

— SOON — 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

— SOON — 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 6.4.17.6 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.

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 6.4.17.7

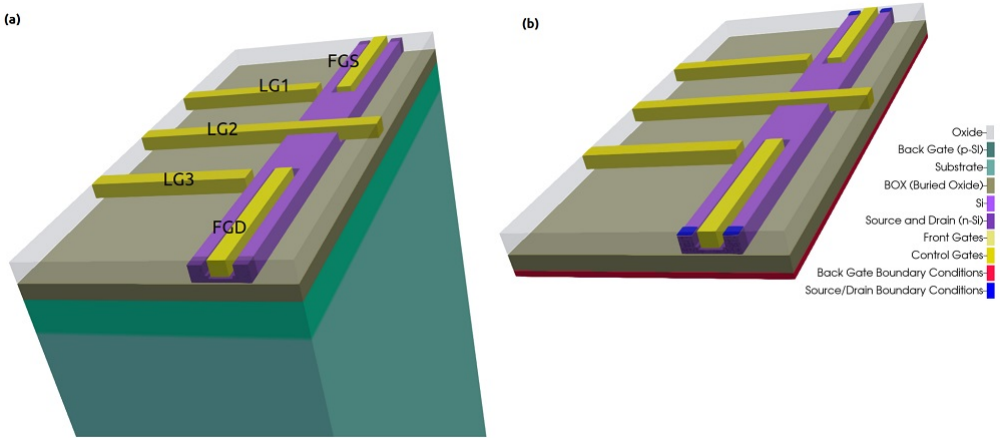


Figure 6.4.17.6: Device to be simulated. The Si-channel is buried in the oxide. FGS, FGD, LG1, LG2, and LG3 are used to shape the electrostatic potential.The back gate is used to allow or to interrupt the transport of electrons through the channel. The source and drain are the reservoirs of carriers.

presents the most important coordinates in the device coordinate system, used in all versions of the input files of our example.

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

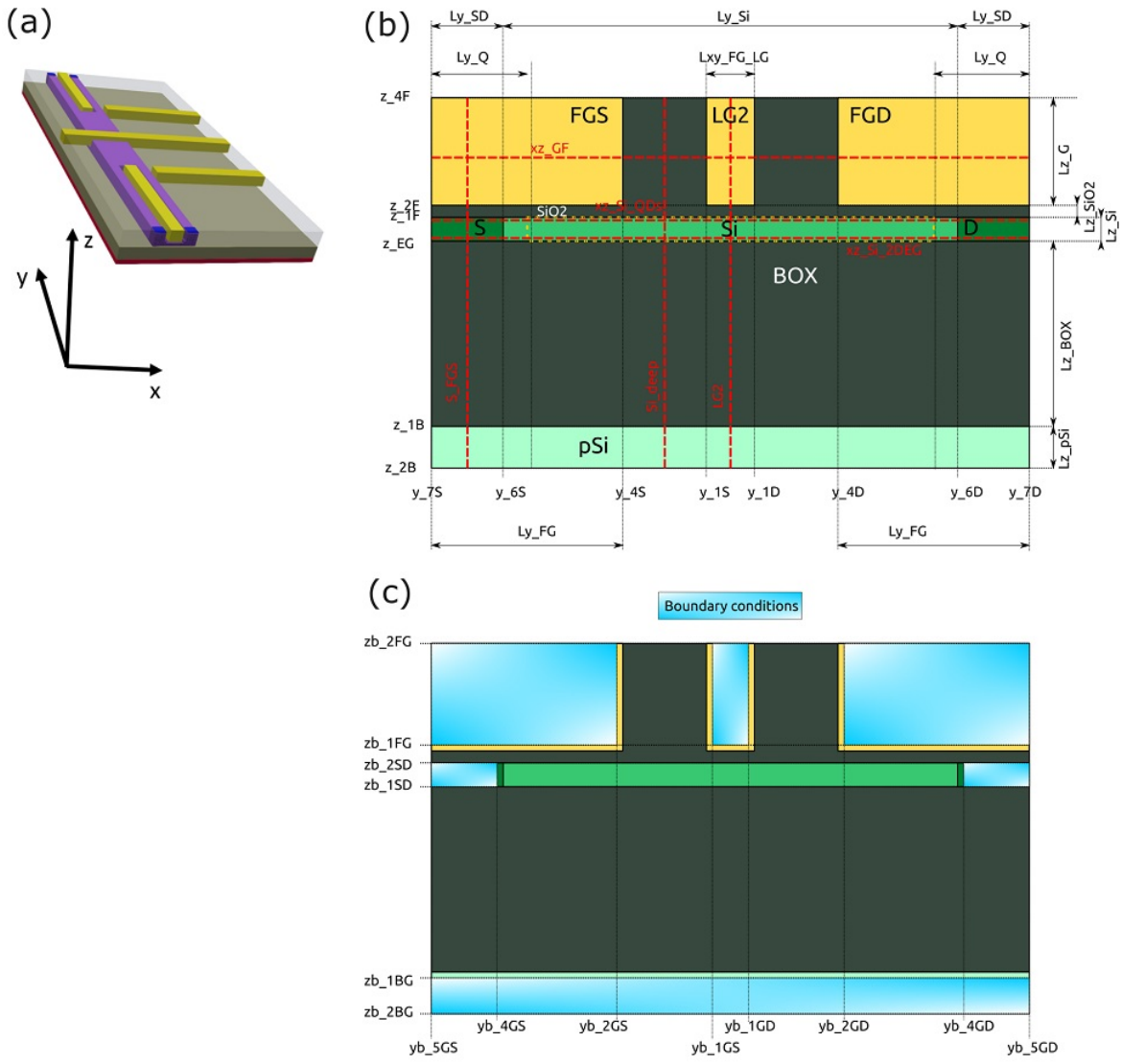


Figure 6.4.17.7: Device reference system and most important coordinates used for 1D and 2D simulations: (a) the 3D representation, (b) structure definition, and (c) structure after applying boundary conditions to the contacts and gates. Dotted lines (in red) represent sections defined in the input files.

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 6.4.17.8 illustrates the final device to be simulated where the substrate and the back gate (green in Figure 6.4.17.6) 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.

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 6.4.17.9 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 6.4.17.10.

These plots were obtained by running this input file for different homogeneous grid line spacings in the growth 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

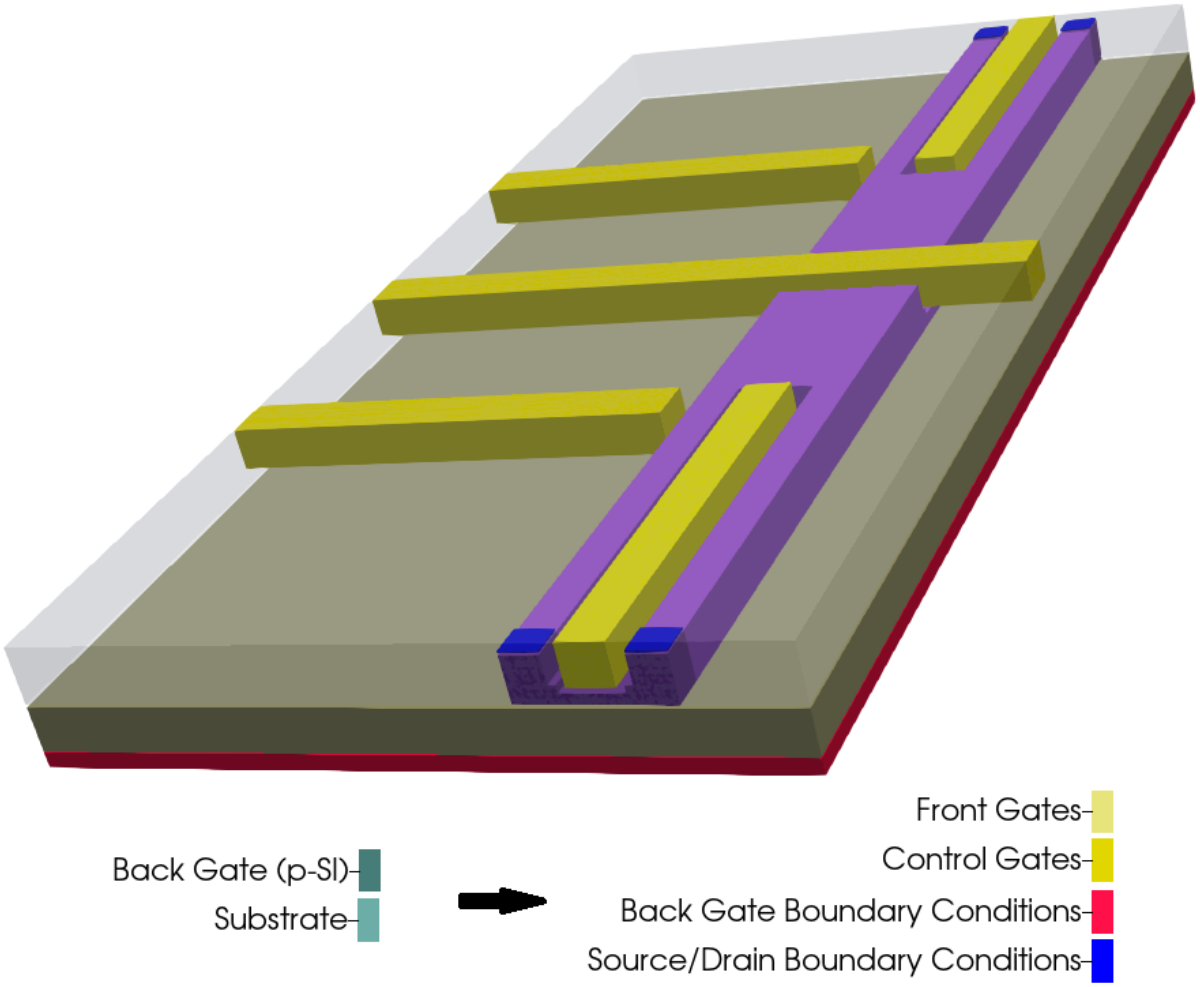


Figure 6.4.17.8: Regions substituted by adequate boundary conditions and final device representation

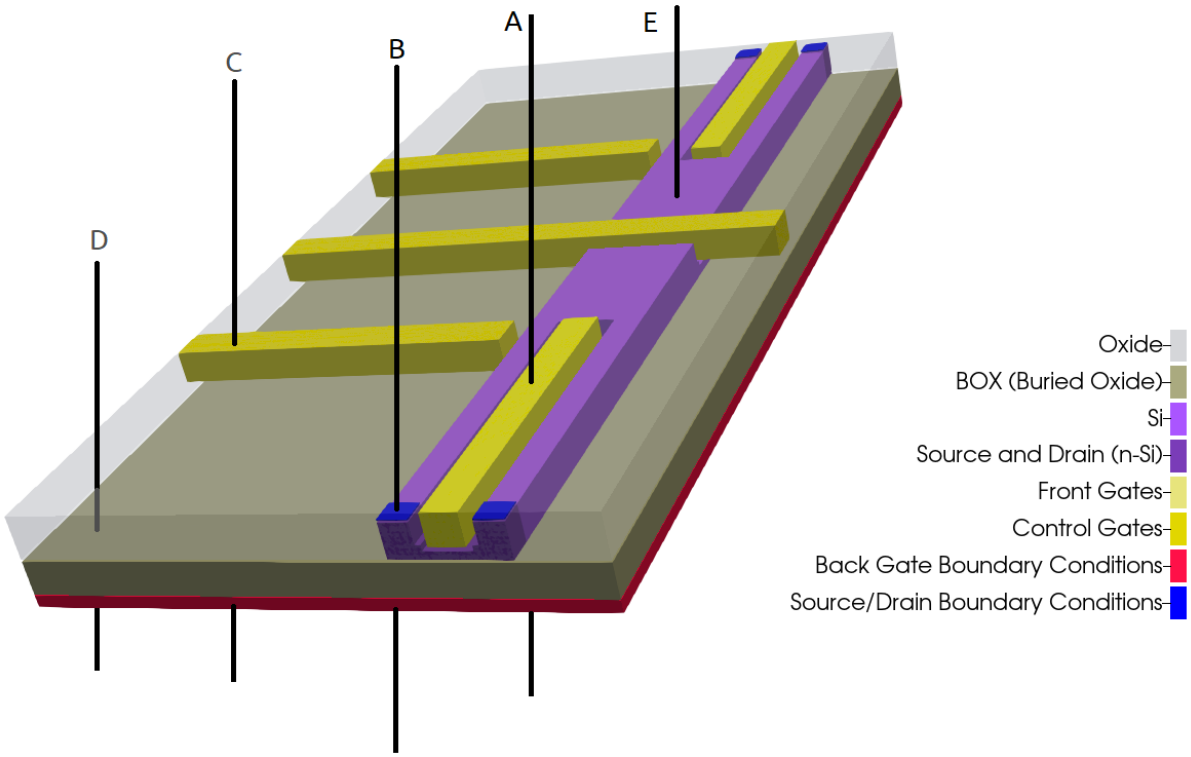


Figure 6.4.17.9: Representation of possible regions of study for 1D simulation in the growth direction.

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 6.4.17.10).

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 6.4.17.11.

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 6.4.17.12. 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.

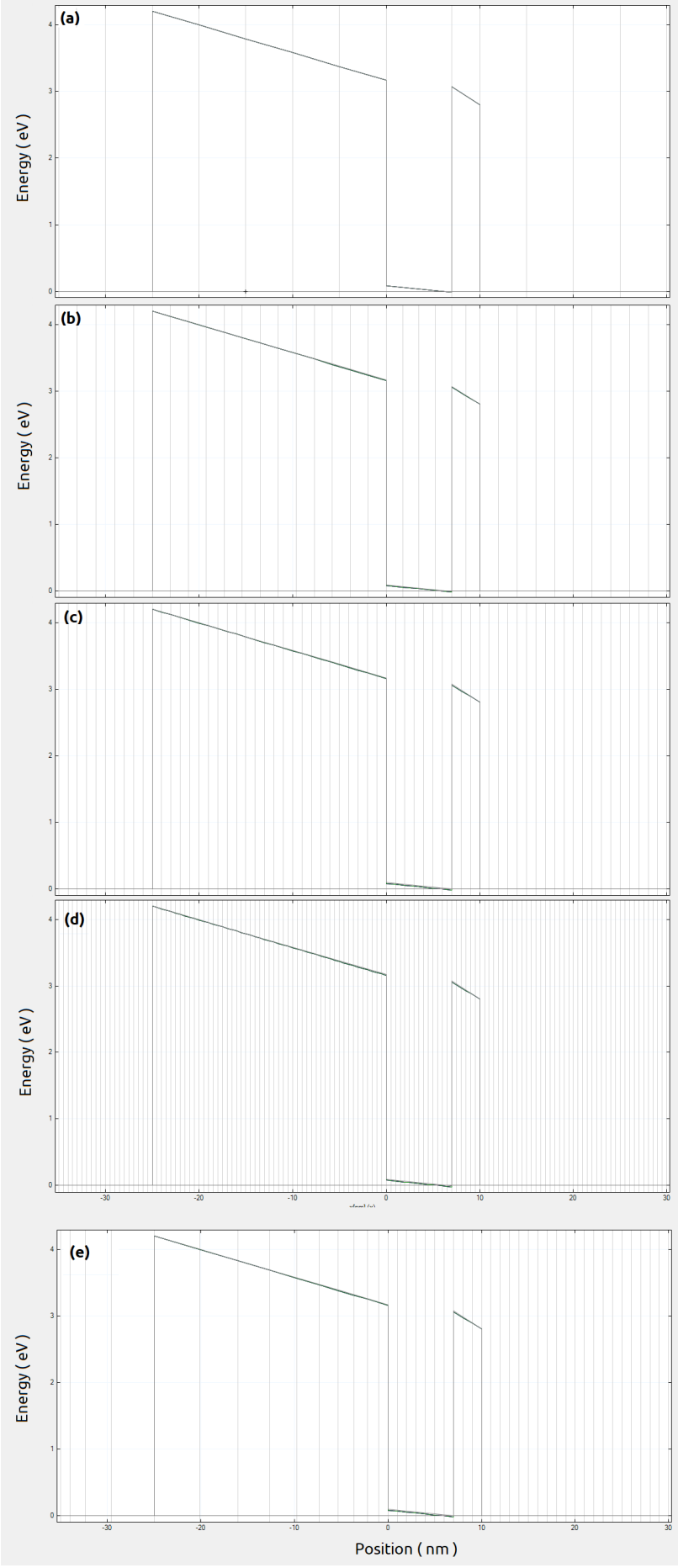


Figure 6.4.17.10: Conduction band resulting from 1D simulations in the growth direction along the line A for homogeneous grid resolution: (a) 5 nm, (b) 2 nm, (c) 1 nm and (d) 0.5 nm. The gray vertical lines represent the grid lines used in this simulation. (e) corresponds to a grid resolution of 1 nm inside the active region and 5 nm in the remaining parts of the structure (in the growth direction).

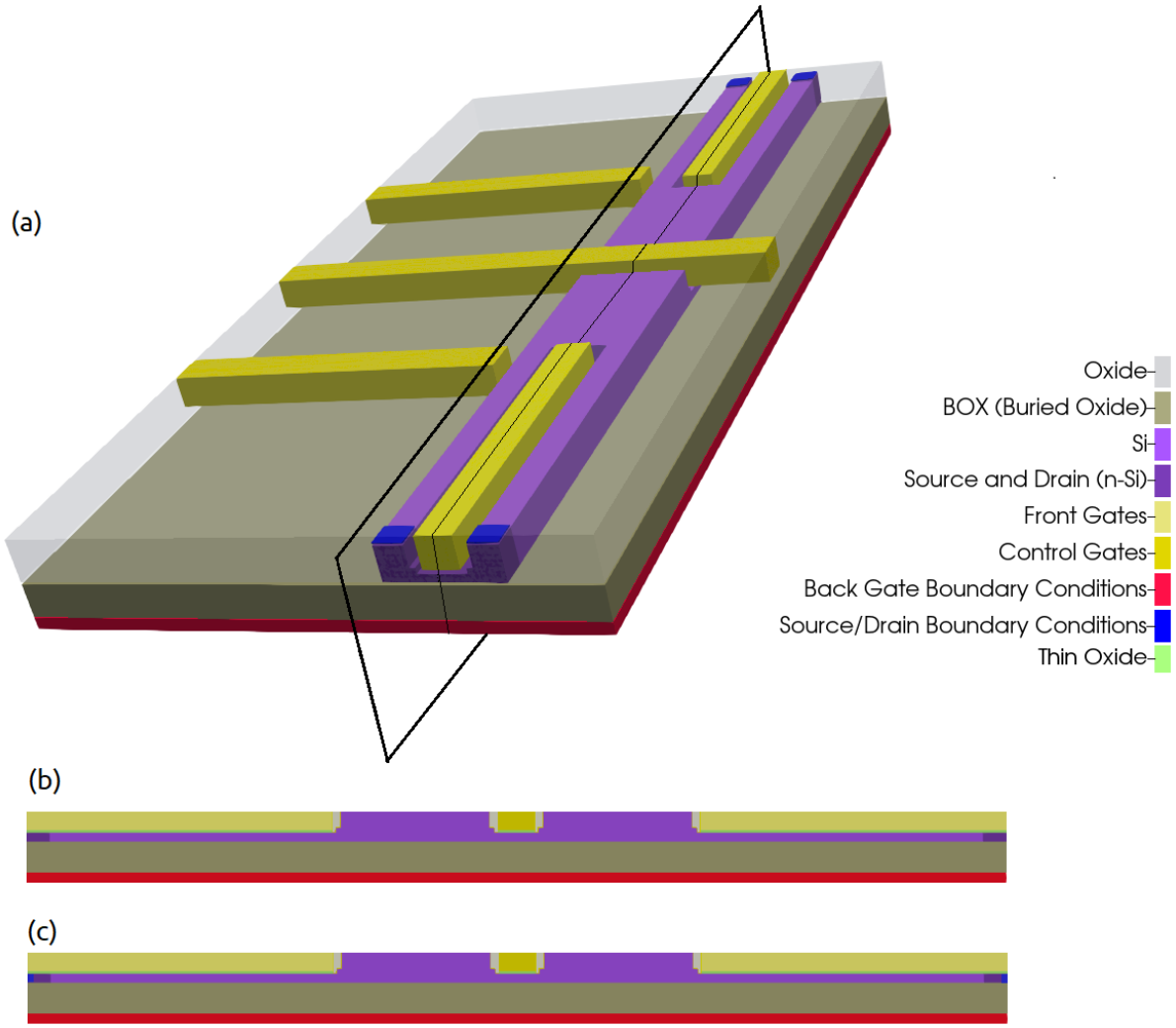


Figure 6.4.17.11: Slice simulated in our example.

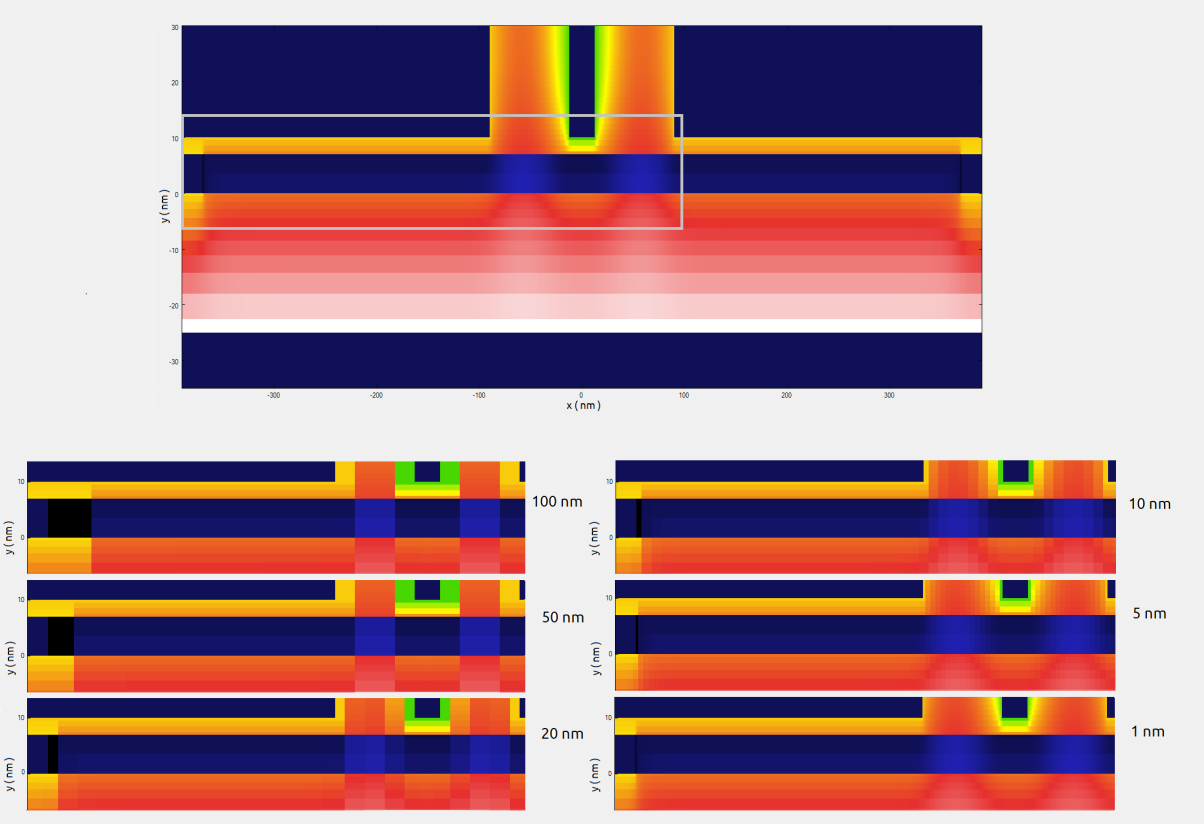


Figure 6.4.17.12: Conduction band along a plane containing the growth direction and the center of the front gates. This result was obtaining grounding all gates and contacts, except the front gates that were biased at 0.8 V. The upper image corresponds to the full simulation domain simulated. The region 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 6.4.17.13 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 6.4.17.7) 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.

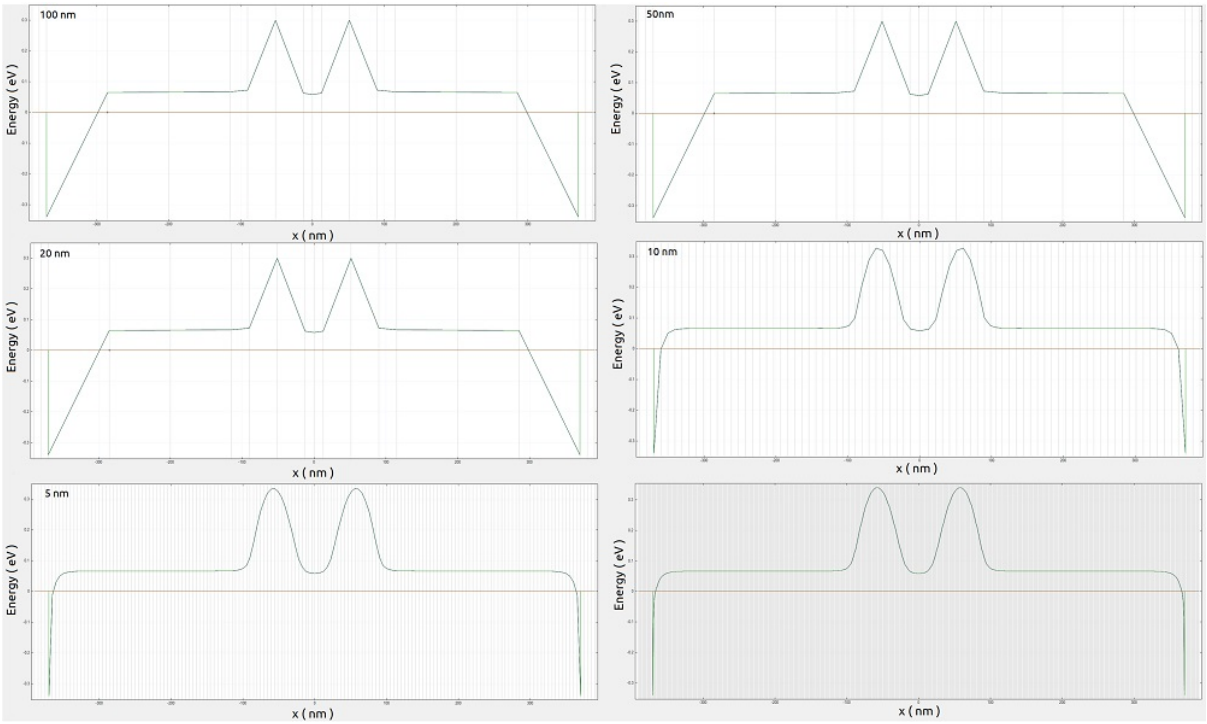


Figure 6.4.17.13: Conduction band at 1 nm above the interface between the buried oxide and the Si-channel (section `xz_Si_2DEG` of Figure 6.4.17.7) from 2D simulations with the different grid spacing. The gray lines correspond to the grid lines.

In Figure 6.4.17.14 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 6.4.17.15 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 6.4.17.15 for your system. All information required for this exercise (number of nodes and runtime) you can find in the file *large-3D-systems-reduction_2D_nnp.log* in the output folder of each simulation.

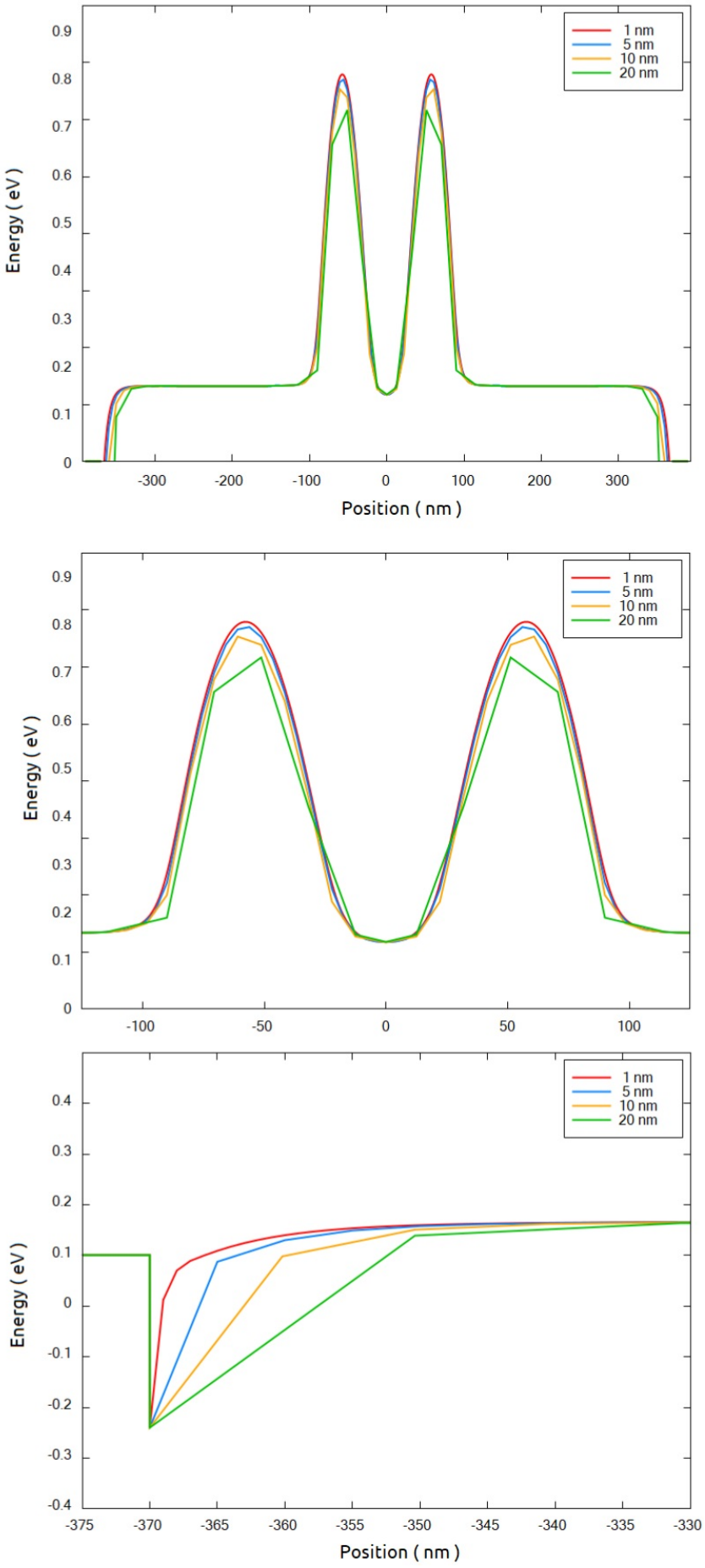


Figure 6.4.17.14: Comparison of the conduction band at a 1 nm above the interface between the buried oxide and the Si-channel (section `xz_Si_2DEG` of Figure 6.4.17.7) from 2D simulations. The central region and the source contact regions are also shown with more details.

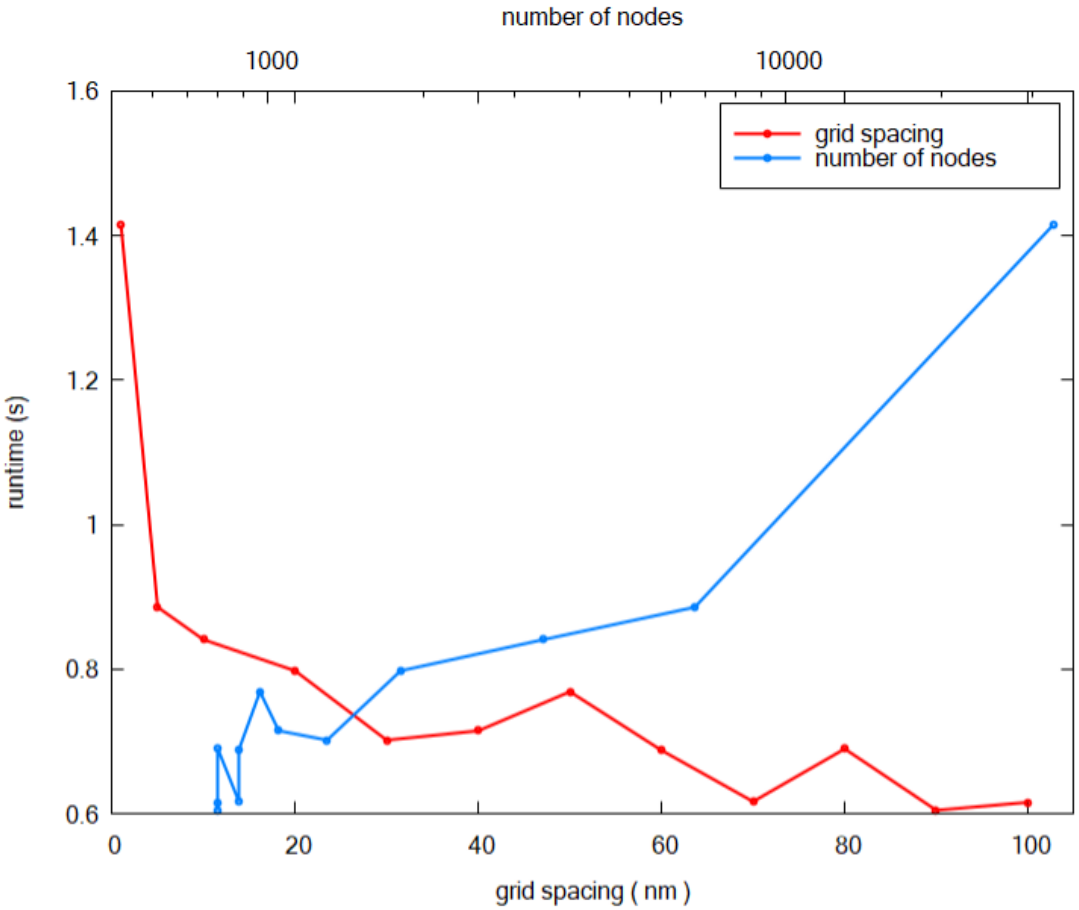


Figure 6.4.17.15: Runtime for 2D simulations as function of the number of nodes in the grid and the grid spacing.

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:

— SOON — Optimizing electrostatics simulation for large 3D designs

— SOON — 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

— SOON — 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 *— SOON — 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 6.4.17.16 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 6.4.17.16 (shown in (b)).

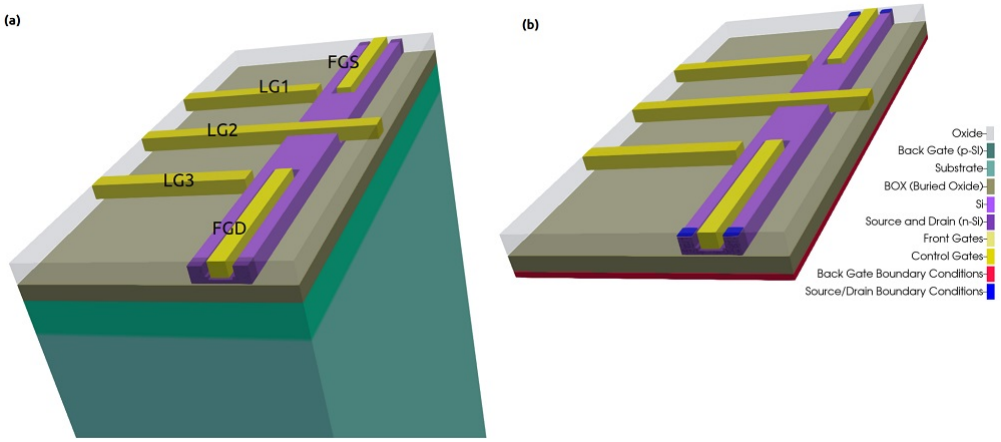


Figure 6.4.17.16: Device to be simulated. The Si-channel is buried in the oxide. FTS, FTD, LG1, LG2, and LG3 at the top of the structure and the back-gate, between the thick oxide layer under of Si layer (BOX) and the substrate, are gates used to shape the electrostatic potential. Source and drain act 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 (*— SOON — 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 6.4.17.17 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.

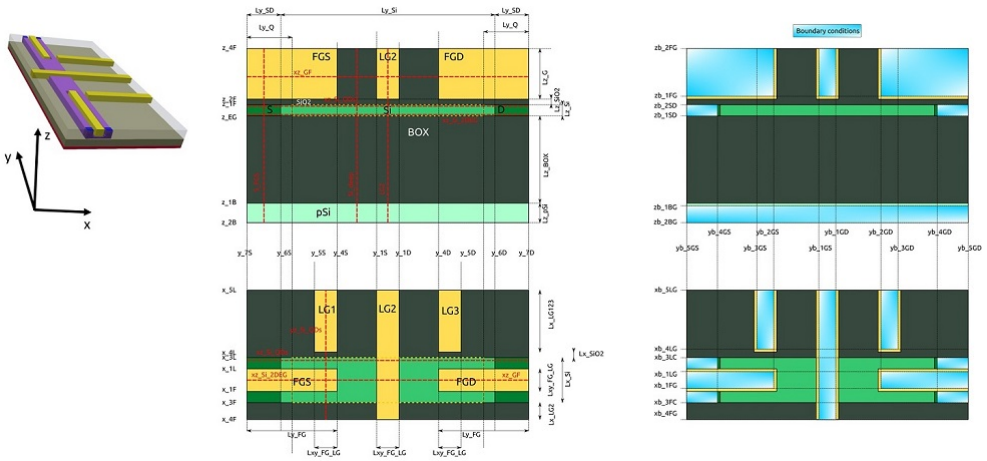


Figure 6.4.17.17: Device reference system and most important coordinates: (a) the 3D representation, (b) structure definition, and (c) structure after applying boundary conditions to the contacts and gates. Dotted lines (in red) represent sections defined in the input files.

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 6.4.17.18 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 6.4.17.16 using dotted lines (in red).

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 6.4.17.19 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.

Another natural candidate to be eliminated is the region from the start of the simulation system (`$x_min = $x_4F`). Figure 6.4.17.20 shows the results of the conduction band for different values of `$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`).

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 6.4.17.21.

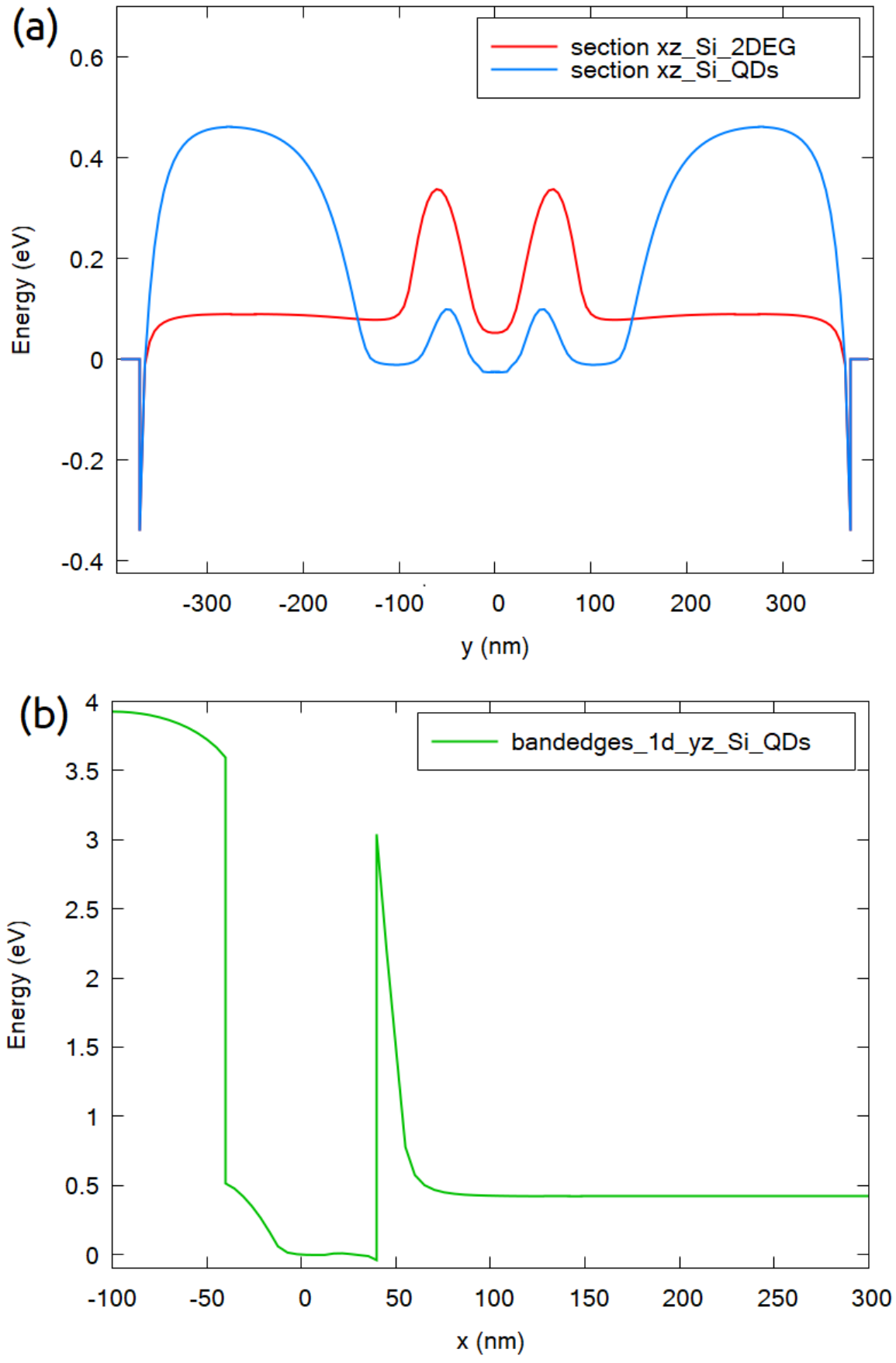


Figure 6.4.17.18: Slices of the conduction band in the two most relevant regions of interest for a particular combination of bias applied to the gates (see text): (a) `xz_Si_QDs`, at 1 nm above the interface between the Si-channel and the BOX, close to the lateral gate LG1 (x = 35 nm), and `xz_Si_2DEG`, 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.

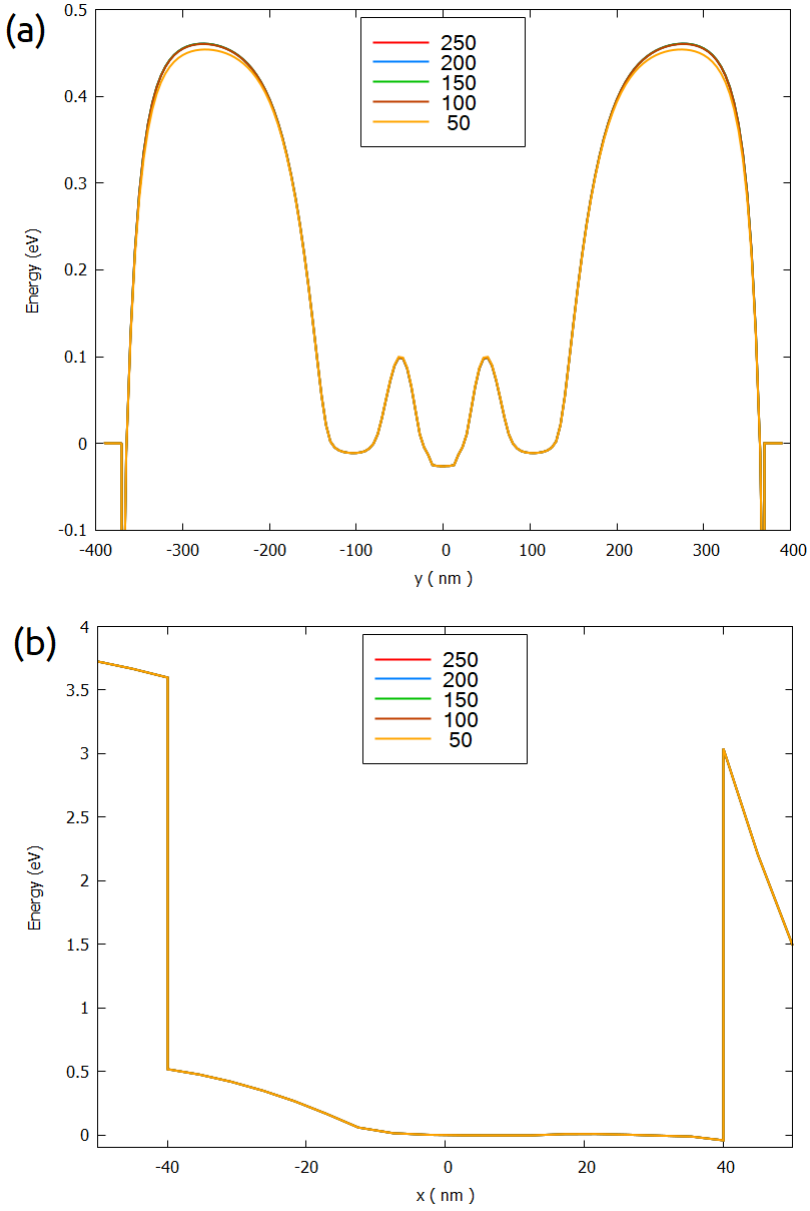


Figure 6.4.17.19: Conduction band in the quantum dot region as function of the length of the gates LG1 and LG3 in the simulation: (a) slice `xz_Si_QDs`, and (b) slice `yz_Si_QDs`

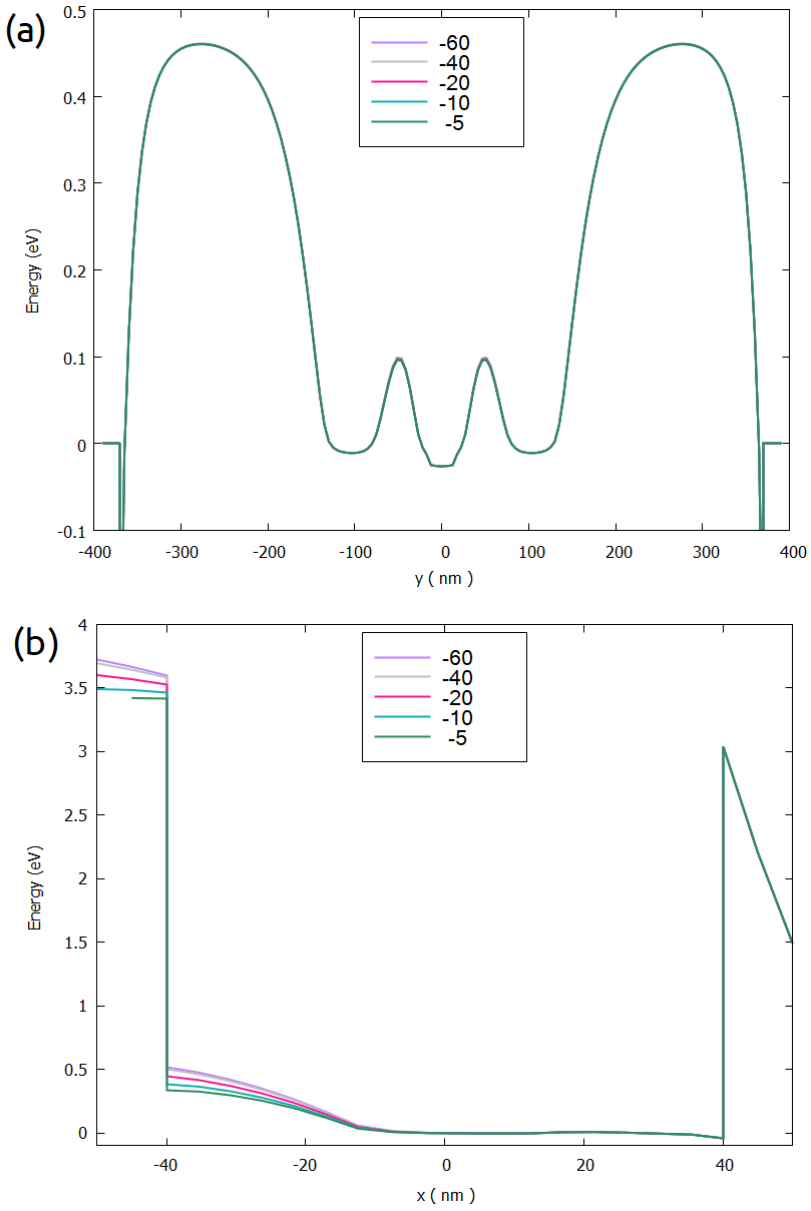


Figure 6.4.17.20: Conduction band in the quantum dot region as function of the value of `$xmin`: (a) slice `xz_Si_QDs`, and (b) slice `yz_Si_QDs`.

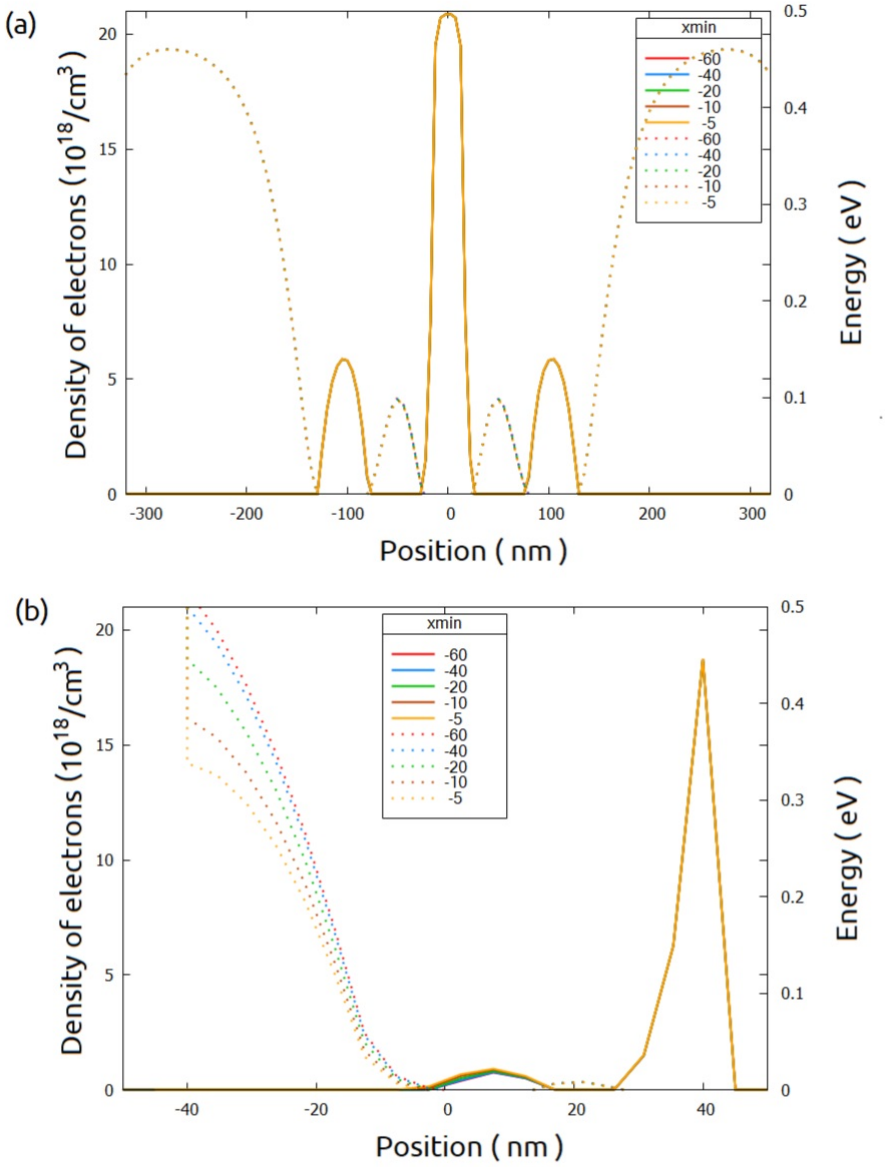


Figure 6.4.17.21: Conduction band (dotted lines) and density of electrons (solid lines) in the `xz_Si_QDs` and `yz_Si_QDs`.

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.

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 6.4.17.21 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 6.4.17.22 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:

> *— SOON — Optimizing electrostatics simulation for large 3D designs*
>
> *— SOON — 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 calculations (the second one).

Last update: 15/07/2024

— SOON — Optimizing Schrödinger-Poisson self-consistent solver for electrostatic quantum dots

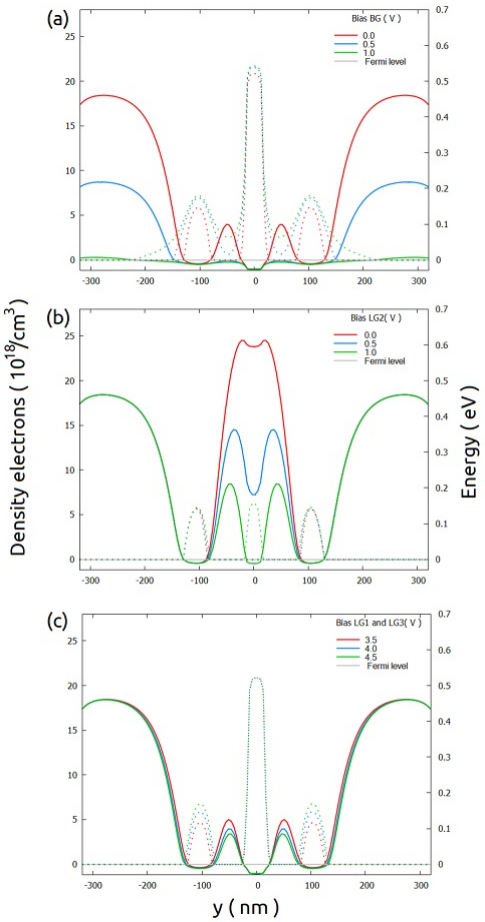


Figure 6.4.17.22: Conduction band (solid lines) and density of electrons (dotted lines) in the `xz_Si_QDs` sections changing only one of the bias of the combination discussed above: (a) the back gate, (b) the central lateral gate (LG2), and (c) simultaneously to the lateral gates LG1 and LG3. Here the full length of the lateral gate LG2 (250 nm) was used.

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 6.4.17.23 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 6.4.17.23 (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).

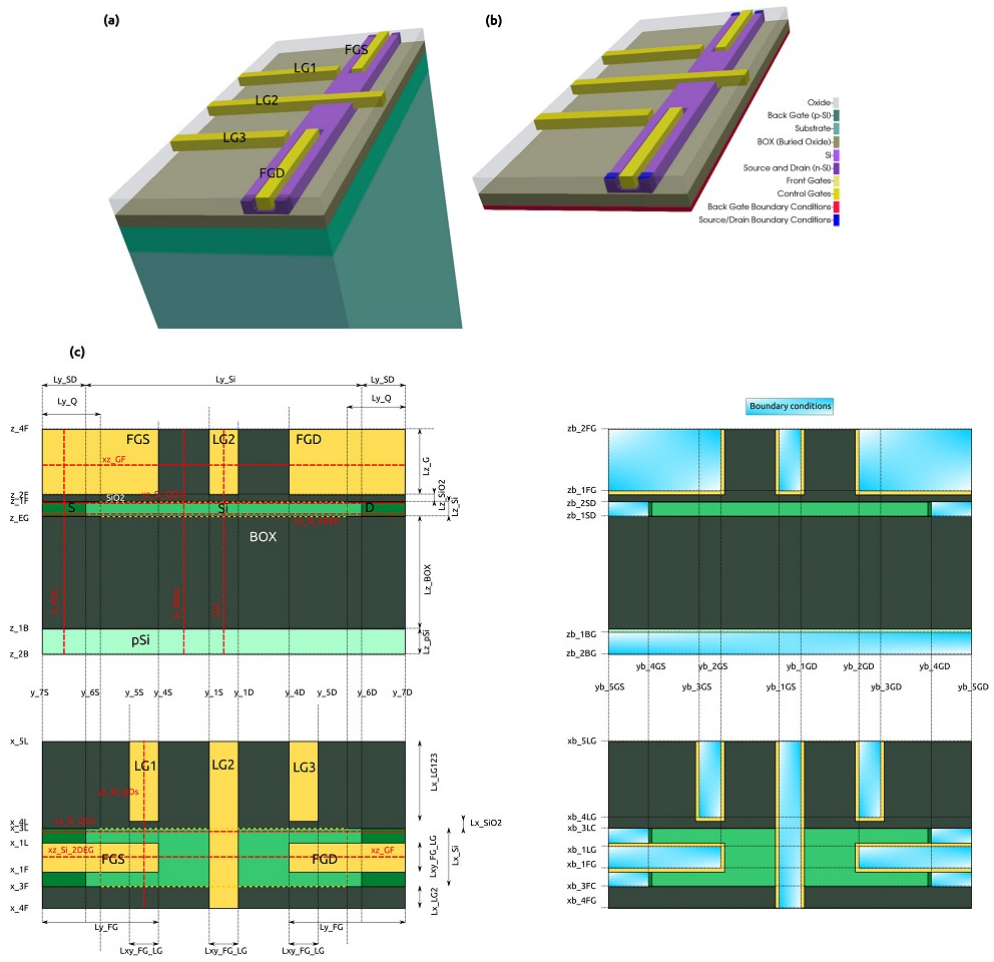


Figure 6.4.17.23: Device to be simulated. The Si-channel is buried in the oxide. The electrostatic potential is shaped by applying bias to the gates (FTS, FTD, LG1, LG2,LG3 and the back-gate). Source and drain act as reservoirs of carriers propagating through the channel. Device (a) before and (b) after 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 *— SOON — Optimizing electrostatics simulation for large 3D designs* concerning some considerations that may be taken into account. Figure 6.4.17.24 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 6.4.17.24, 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.

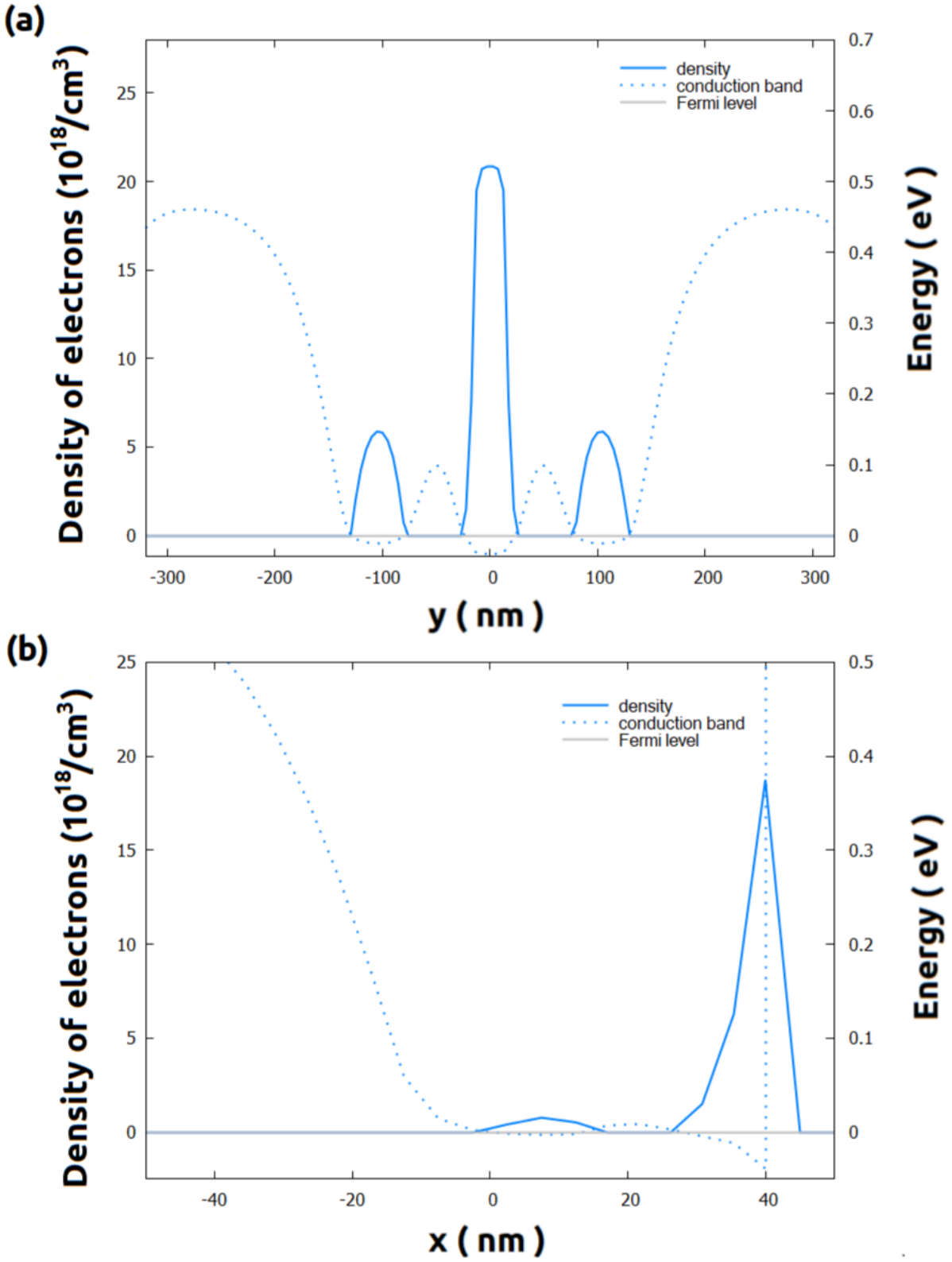


Figure 6.4.17.24: Conduction band (dotted lines) and semi-classical density of electrons (solid lines) in the slices `xz_Si_QDs` and `yz_Si_QDs` (see red dotted lines in Figure 6.4.17.24), when 0.8 V is applied to both front gates, 4 V to LG1 and LG3 and 1.7 V to the central gate LG2. The remaining gates and contacts are kept grounded.

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.

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 *quantum{ 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 6.4.17.25 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.

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 6.4.17.26 for the same sections of Figure 6.4.17.25. 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.

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

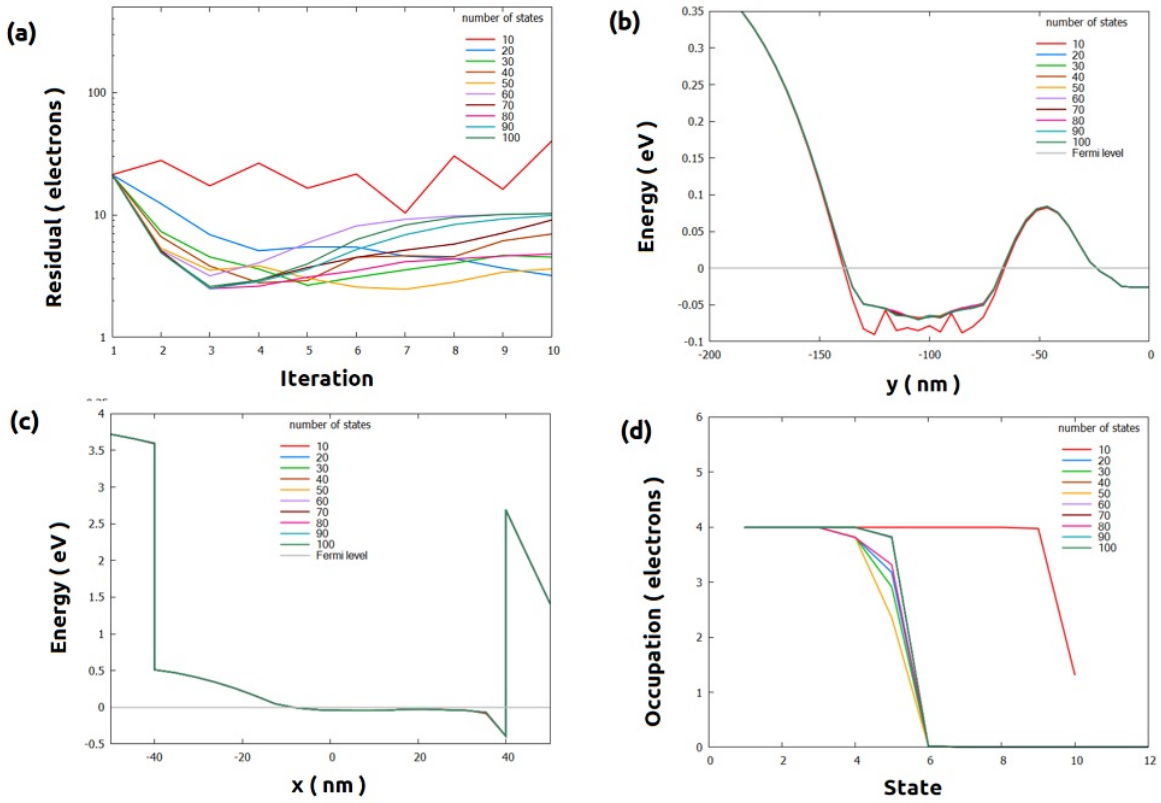


Figure 6.4.17.25: Results of the self-consistent Schrödinger-Poisson simulations, in the reduced QROI as function of the number of eigenstates computed, after 10 iterations. (a) The evolution of the residuals, (b) and (c) the conduction band for the sections `xz_Si_QDs` and `yz_Si_QDs`, respectively, and (d) the occupation number after only 10 iterations. These are intermediate results: the convergence process was still not completed.

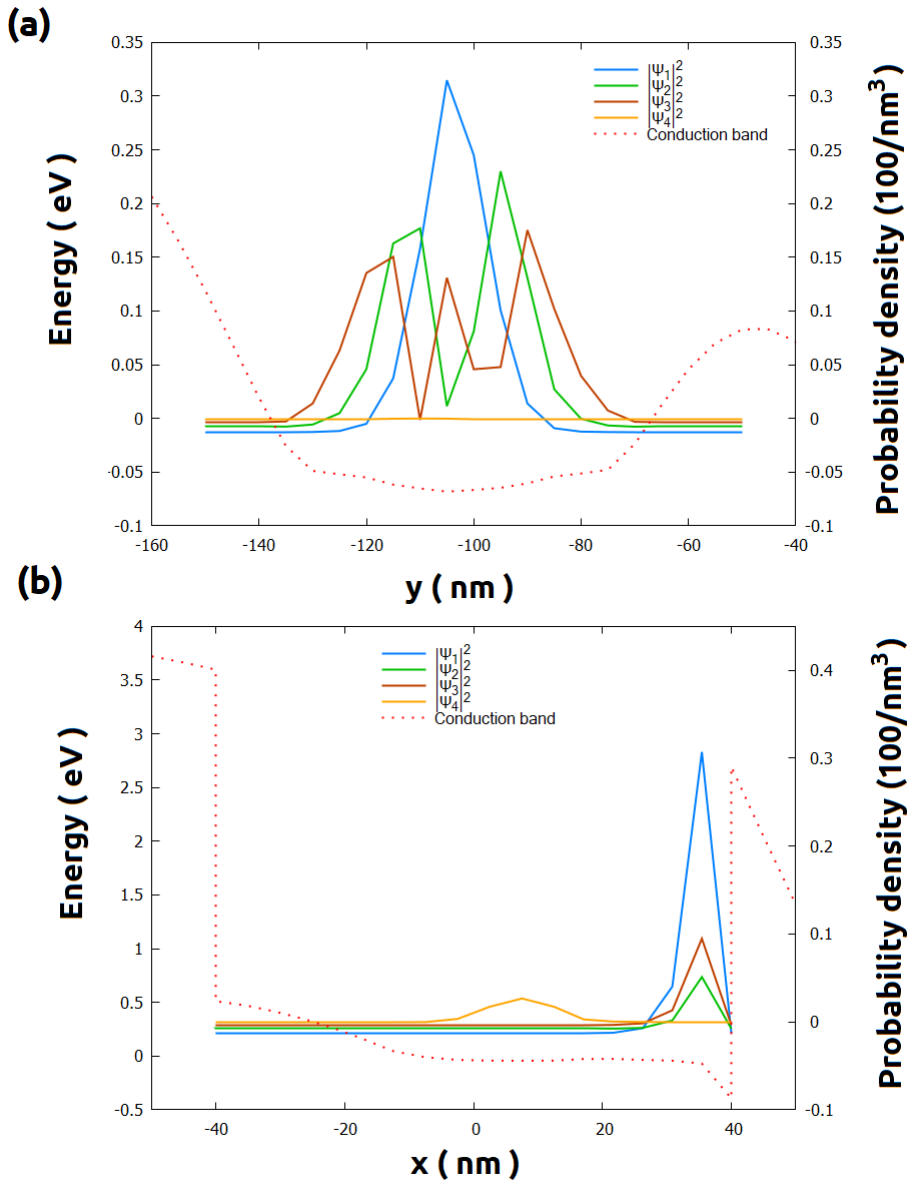


Figure 6.4.17.26: Wavefunctions overlapped to the conduction band from self-consistent quantum-Poisson simulations for the sections (a) `xz_Si_QDs` and (b) `yz_Si_QDs`. The "quantum" density of electrons were computed considering 20 states and a grid resolution of 5 nm in the y-direction. These are intermediate results: the convergence process was stopped after 10 iterations.

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 }
        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 }
```

(continues on next page)

(continued from previous page)

```
        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 6.4.17.27 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.

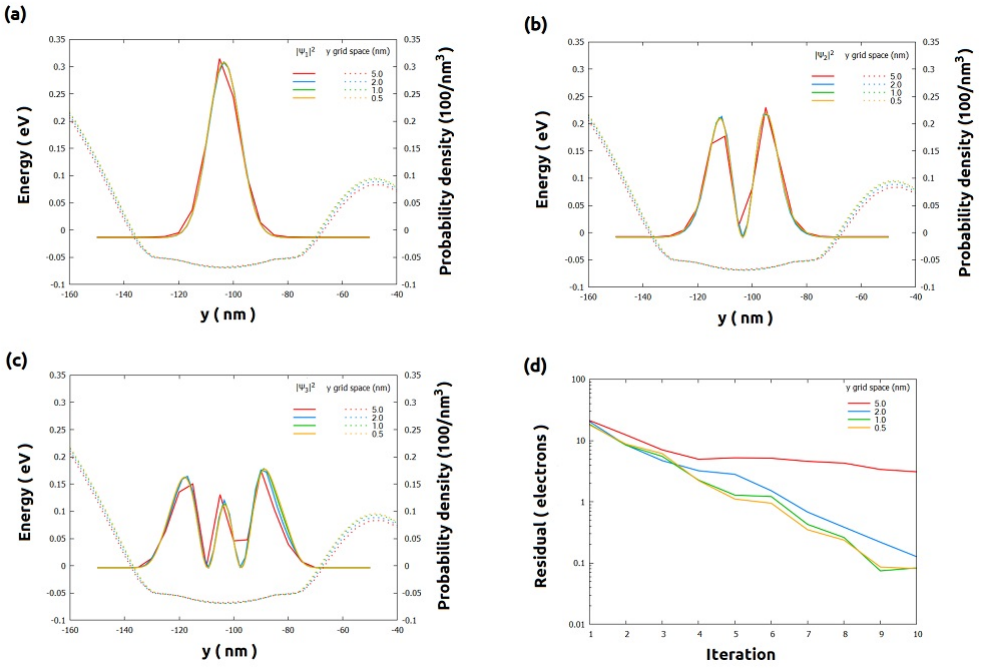


Figure 6.4.17.27: Results of the self-consistent Schrödinger-Poisson simulations using different grid spacing in the y-direction only within the reduced QROI (section `xz_Si_QDs`). (a) -(c) wavefunctions (solid lines) for the three lowest states overlapped with the conduction band (dotted lines) (d) 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 (`$space_x_QR`) we obtain the wavefunctions of the Figure 6.4.17.28. In the y-direction the grid resolution in this region was considered equal to 1 nm (`$space_y_QR = 1` in our input file).

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.

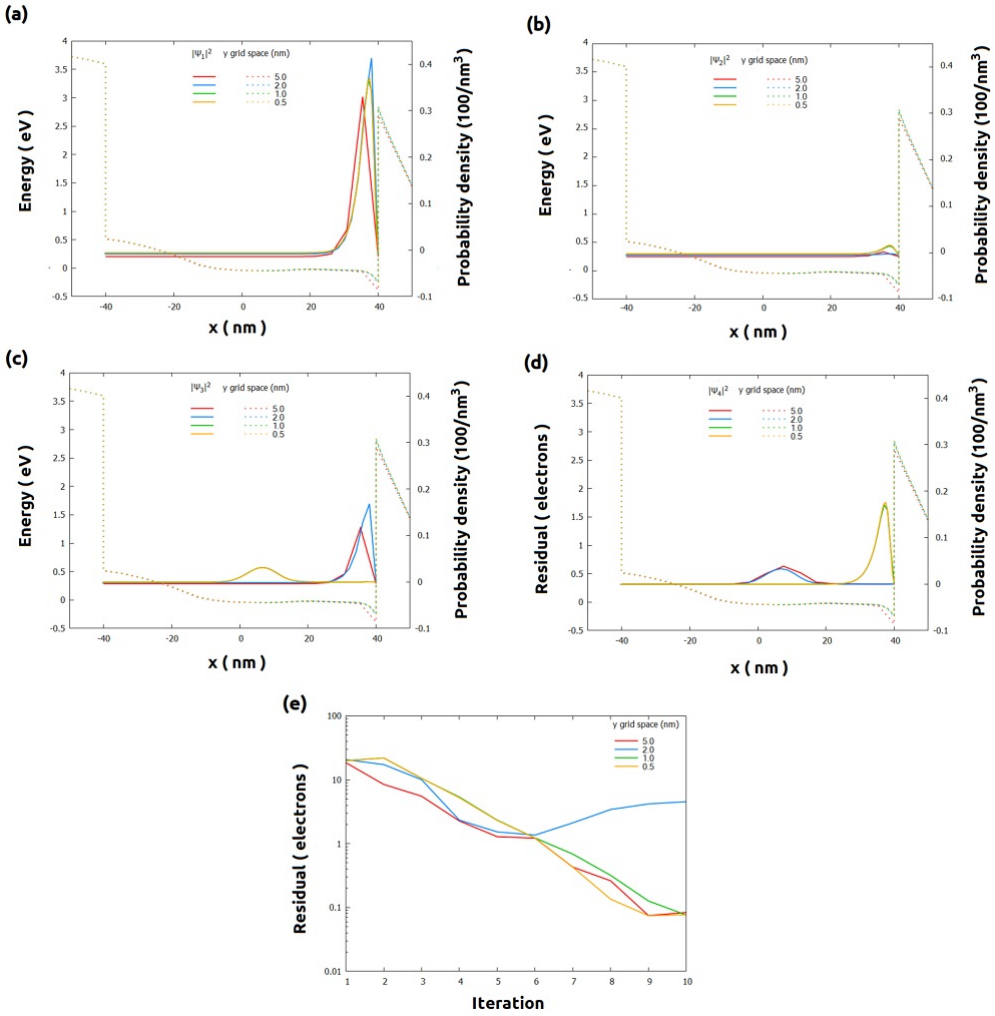


Figure 6.4.17.28: Results of the self-consistent Schrödinger-Poisson simulations for section `yz_Si_QDs` using different grid spacing in the x-direction within the QROI: (a)-(d) wavefunctions (solid lines) for the four lowest states overlapped with the conduction band (dotted lines), and (e) residual evolution. These are intermediate results after only 10 iterations: the convergence process was still not completed. `$space_y_QR` was kept 5 nm.

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 6.4.17.24 (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 6.4.17.29 illustrates a sequence of simulations where the grid resolution and the number of states were iteratively changed.

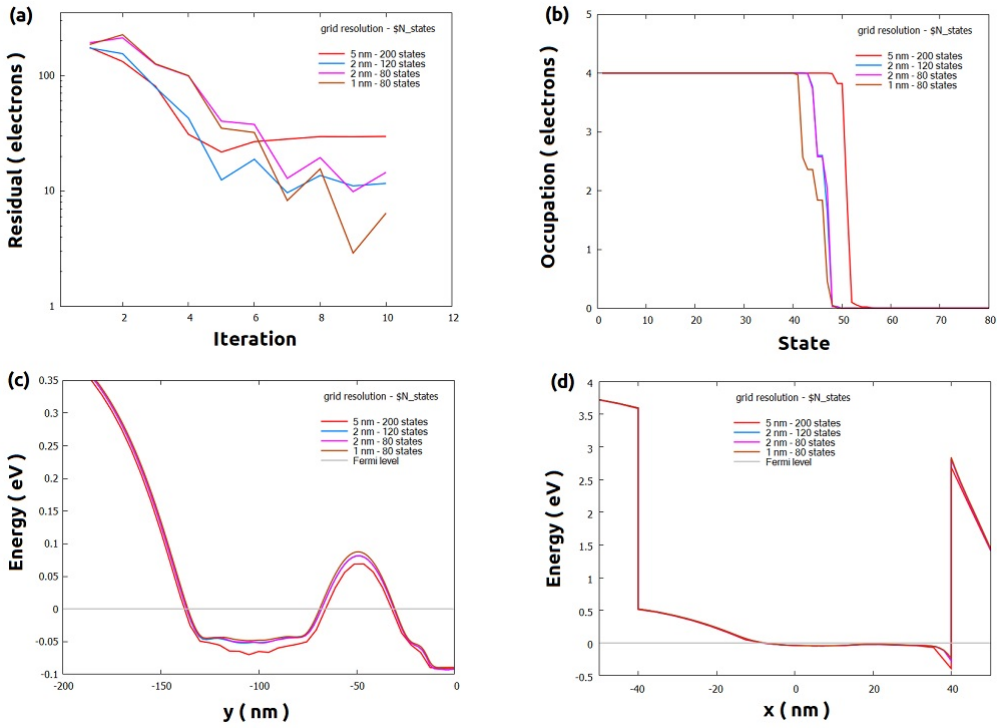


Figure 6.4.17.29: Sequence of simulations for defining a suitable value for `$Nstates`. (a) residual evolution, (b) occupation number of the most populated band, (c) and (d) conduction band for the sections `xz_Si_QDs` and `yz_Si_QDs`, respectively. These are intermediate results (only 10 iterations 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 6.4.17.30 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.

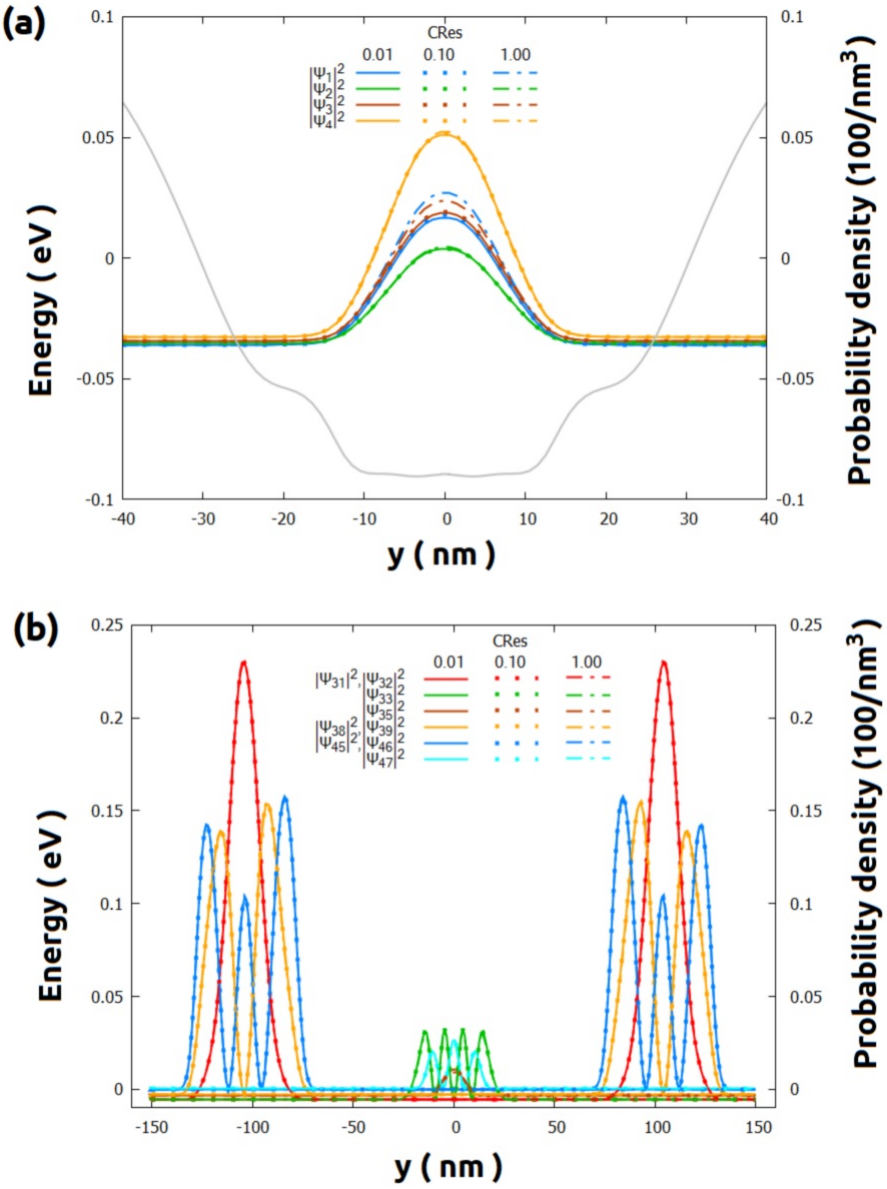


Figure 6.4.17.30: Some wavefunctions in the section `xz_Si_QDs` as function of the residual used in the convergence process `$CRes`. (a) in the central, and (b) in the whole quantum region. All solutions converged before reaching the maximum number of iterations (100).

The solutions for the section `yz_Si_QDs` are even more robust than the previous one (see Figure 6.4.17.31): the wavefunctions do not present relevant variation even when the value of `$Cres` is higher.

Hint: 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`.

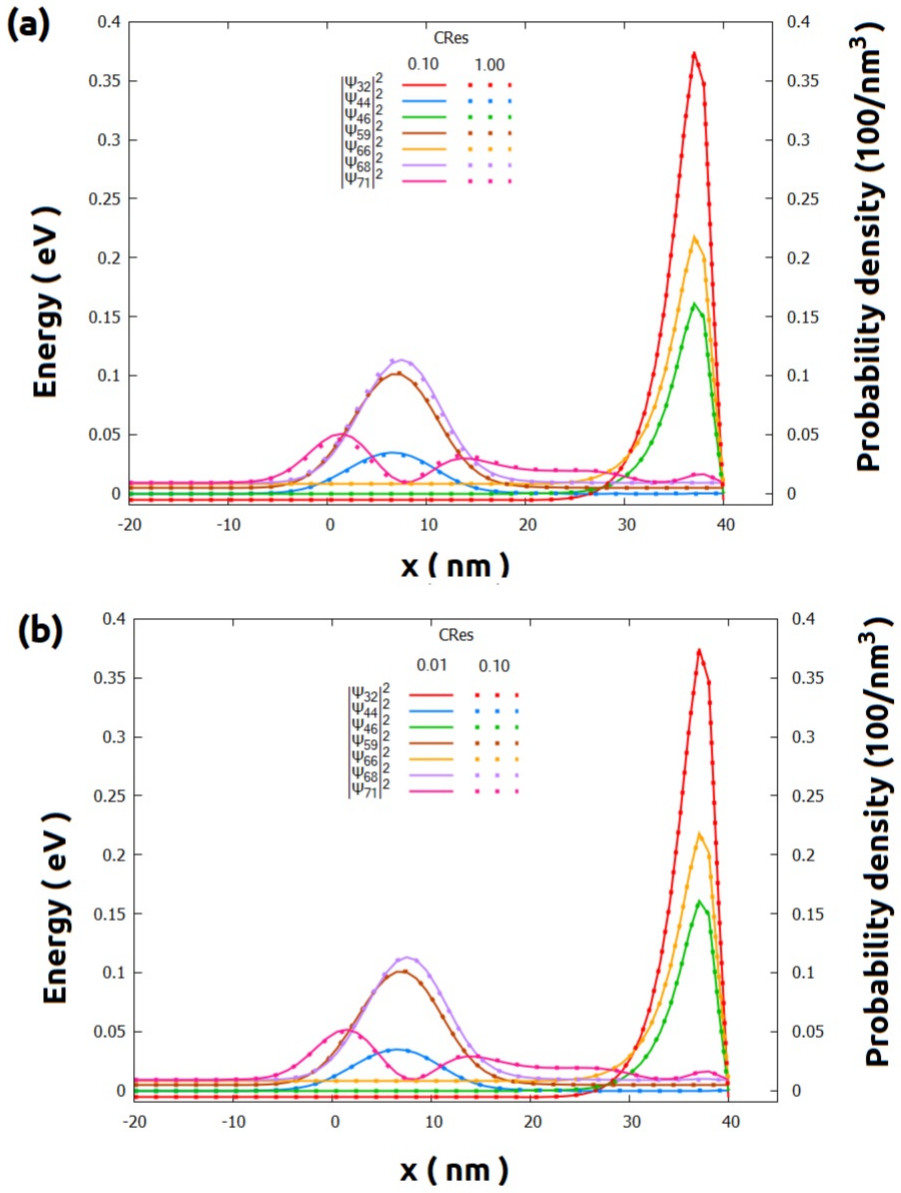


Figure 6.4.17.31: Comparison of some wavefunctions in the section `yz_Si_QDs` for convergence residual (`$Cres`) (a) from 1.00 to 0.10, and (b) from 0.10 to 0.01. All solutions converged before reaching the maximum number of iterations (100).

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... { } } }* 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:

> *— SOON — Reducing dimensionality of large 3D designs*
>
> *— SOON — Optimizing electrostatics simulation for large 3D designs*

Last update: 15/07/2024

6.4.18 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.

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.

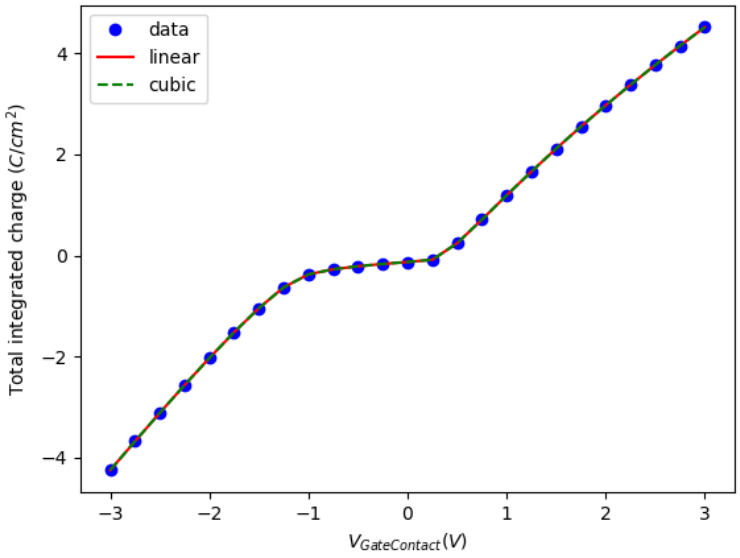


Figure 6.4.18.1: Q-V characteristics obtained by post-processing the result of *MIS_CV_1nmSiO2_1D_nnp.in* by *CapacitanceBySplines_2021_Nov.py*. Linear and cubic interpolation are done to the output data.

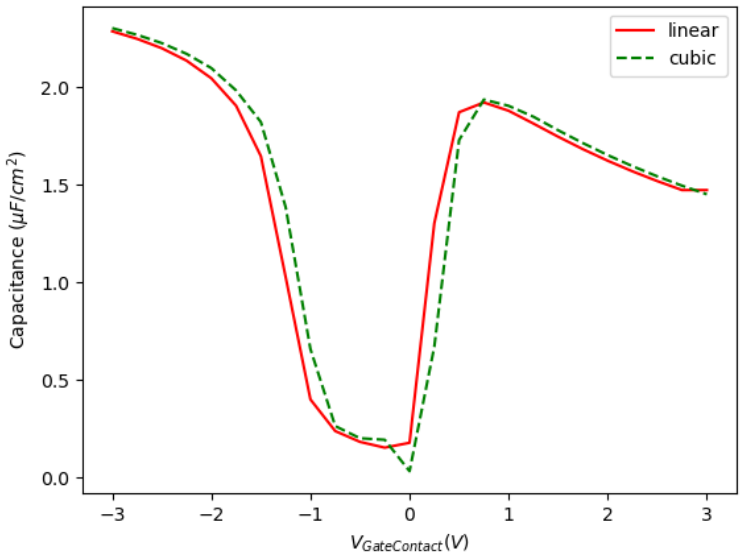


Figure 6.4.18.2: C-V characteristics obtained by post-processing the result of *MIS_CV_1nmSiO2_1D_nnp.in* by *CapacitanceBySplines_2021_Nov.py*.

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
```

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 capacities (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

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

$$I_{ij} = \frac{2\pi q}{\hbar} \sum_{\sigma} |M_{ij}^{\sigma}|^2 \cdot \frac{m_r}{2\pi\hbar^2} = \frac{q m_r}{\hbar^3} \sum_{\sigma} |M_{ij}^{\sigma}|^2$$

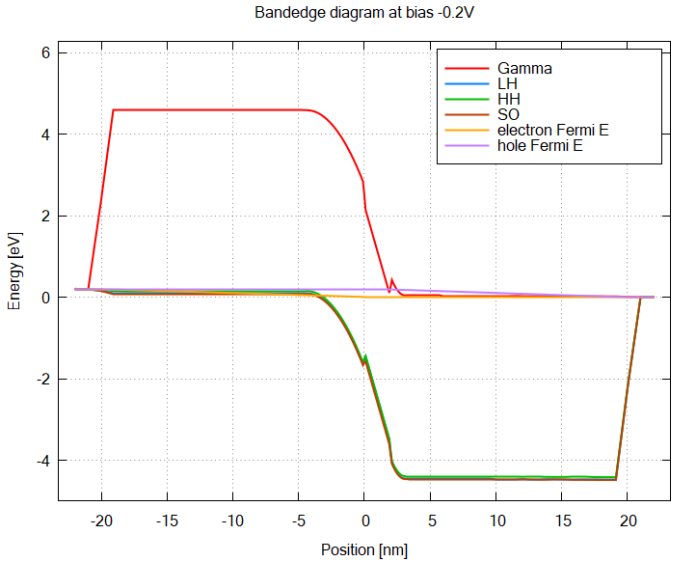
using the *nextnano++* outputs of the effective masses.

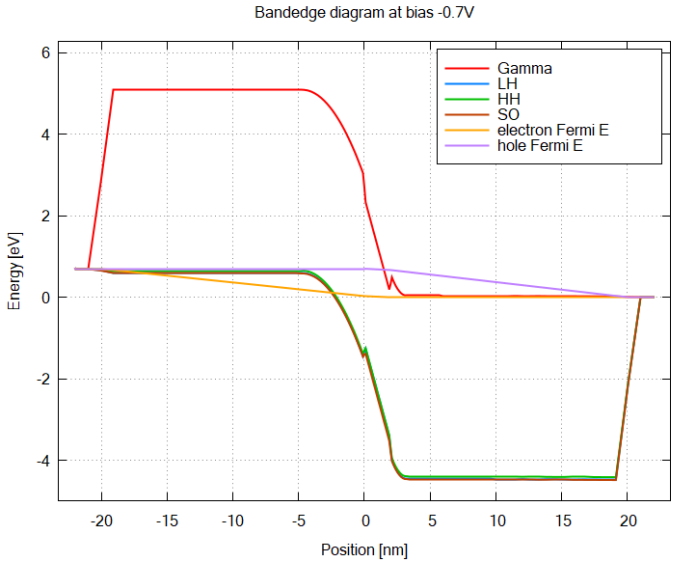
When `CalculateReducedMass_fromOutput = False` *(the default)*, then the assumption as in *[Duboz2019]* is used.

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.

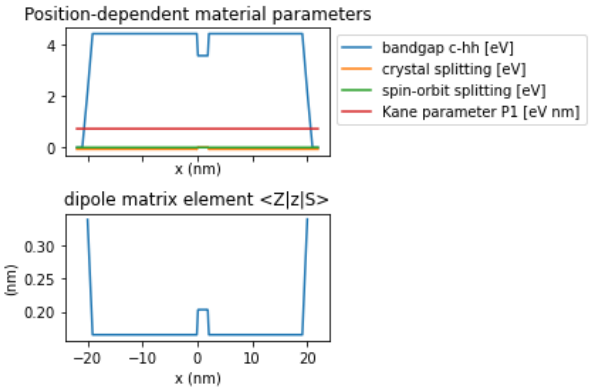




The Python script calculates dipole matrix elements from the simulation results:

from which we obtain the tunnel current as a function of bias:

Last update: 17/07/2024



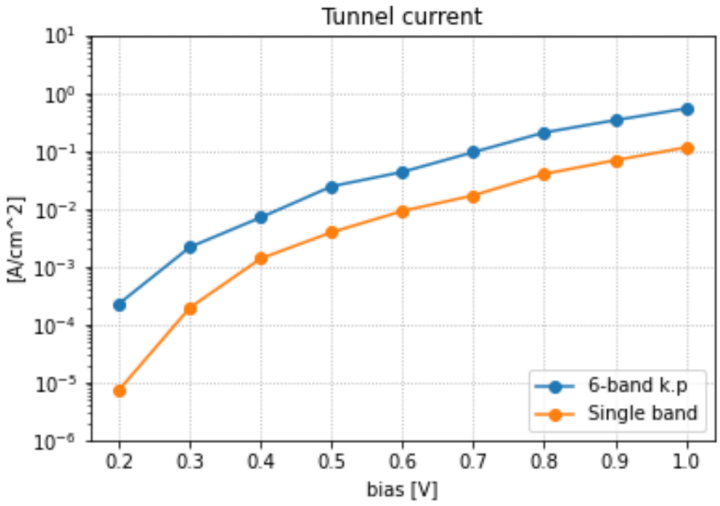


Figure 6.4.18.3: Interband tunneling current in a nitride p-i-n junction. Following the paper, backward bias is taken to be positive in this plot.

Optical generation in InGaAs/GaAs QW

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 6.4.18.4 is employed.

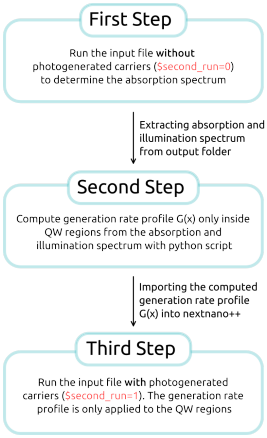


Figure 6.4.18.4: Visualization of the Simulation Procedure

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
                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 6.4.18.5 (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 6.4.18.5 (right).

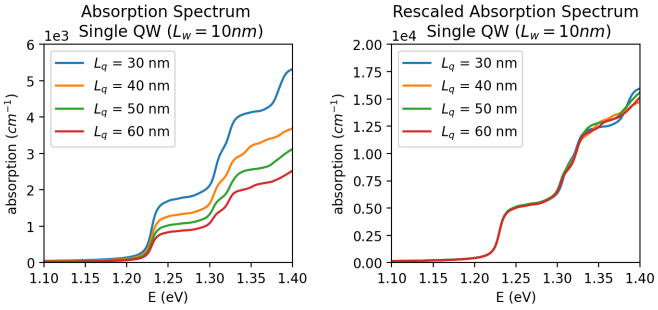


Figure 6.4.18.5: Computed absorption spectrum of a single InGaAs/GaAs quantum well for different quantum region widths L_q, unscaled (left) and rescaled by the factor L_w/L_q (right)

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␣
↪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␣
```

(continues on next page)

(continued from previous page)

```
↪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 6.4.18.6 shows the generation rate profile calculated according to the above described methodology.

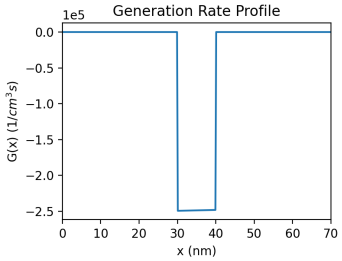


Figure 6.4.18.6: Computed generation rate profile G(x) for single QW structure

Bandedges and Fermi Levels

Figure 6.4.18.7 compares the band edges and Fermi levels without photogeneration (left) and with photogeneration based on the imported generation rate profile (right).

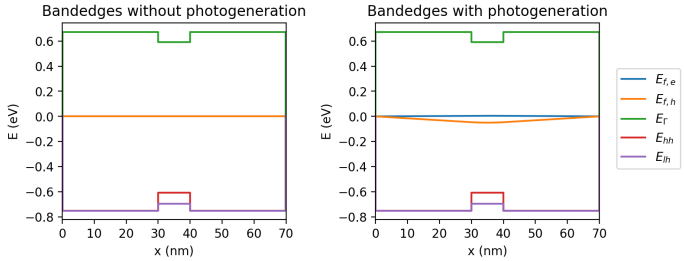


Figure 6.4.18.7: Band edge profile of 1D QW (L_w = 10 nm) structure without photogeneration (left) and with photogeneration (right)

Last update: 17/07/2024

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 6.4.18.8, the recombination process happens mainly in the quantum well.

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

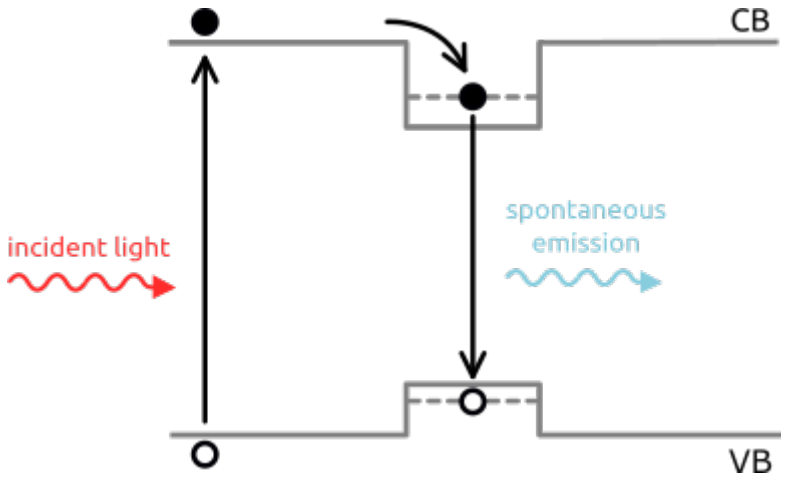


Figure 6.4.18.8: Visualization of involved processes: 1) light absorption and generation of electron - hole pairs, 2) trapping of carriers inside the quantum well, 3) recombination and spontaneous emission of light

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 6.4.18.9 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.

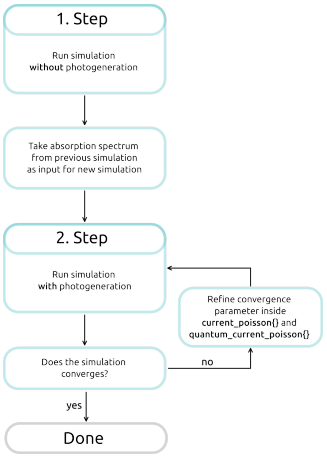


Figure 6.4.18.9: Iterative procedure calculating absorption spectrum until convergences is reached

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
            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

        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
        }
```

(continues on next page)

(continued from previous page)

```
        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
        max = 5.0
        energy_resolution = 0.001
        only_quantum_regions = no
    }

    energy_distribution{
        # settings for energy resolved density
        min = -4.0
        max = 4.0
        energy_resolution = 0.001
        only_quantum_regions = no
    }
```

(continues on next page)

(continued from previous page)

```
}
```

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 6.4.18.10 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.

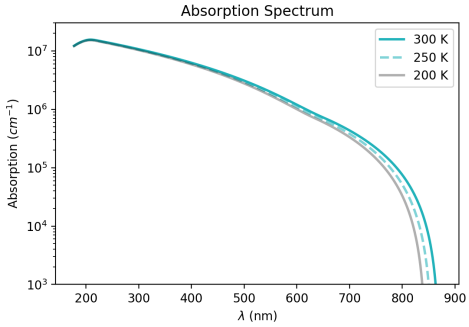


Figure 6.4.18.10: Calculated absorption spectrum

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 6.4.18.11 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.

2) **Electron/ hole density**

Figure 6.4.18.12 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

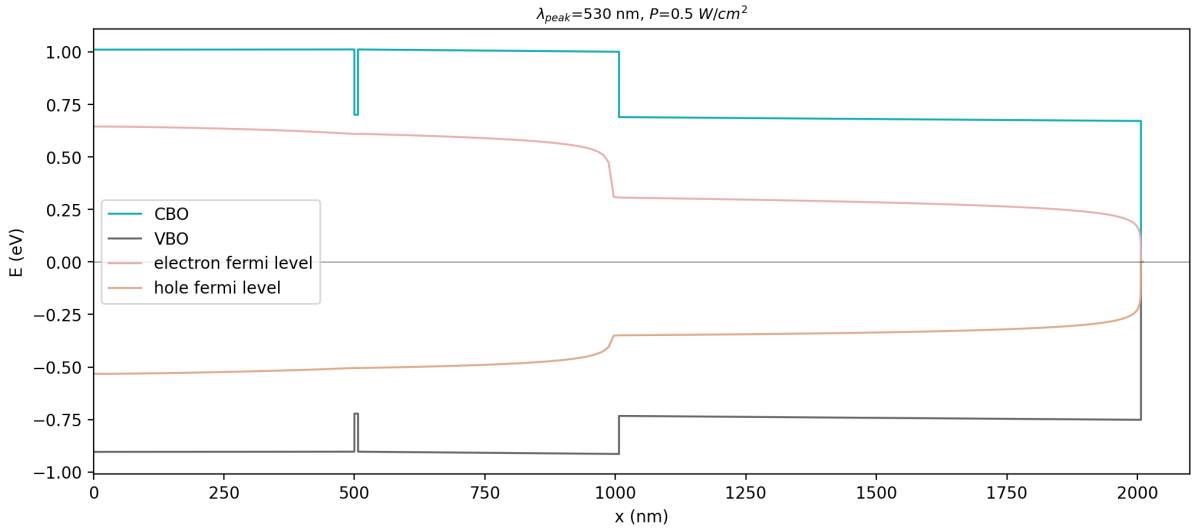


Figure 6.4.18.11: Energy profiles of conduction band (CBO) and valence band (VBO), with electron- and hole-Fermi levels across the structure

(*recombination*) between electron states in the conduction band and hole states in the valence band. From Figure 6.4.18.12 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.

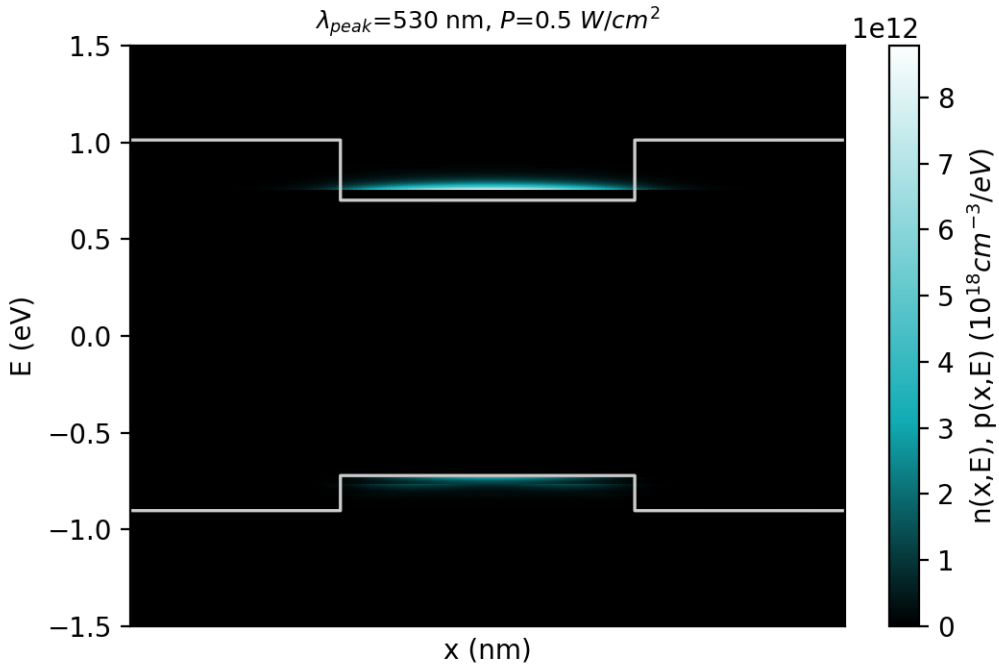


Figure 6.4.18.12: Electron density n(x,E) and hole density p(x,E) with conduction and valence band edges at 300K.

3) **Photogeneration**

Figure 6.4.18.13 depicts the spatial and energy resolved generation rate inside the structure for the case: $P_{illumination}$ $0.5 \cdot 10^4$ W/cm^2 at 300 K.

4) **Spontaneous emission spectrum**

Figure 6.4.18.14 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 shrink-

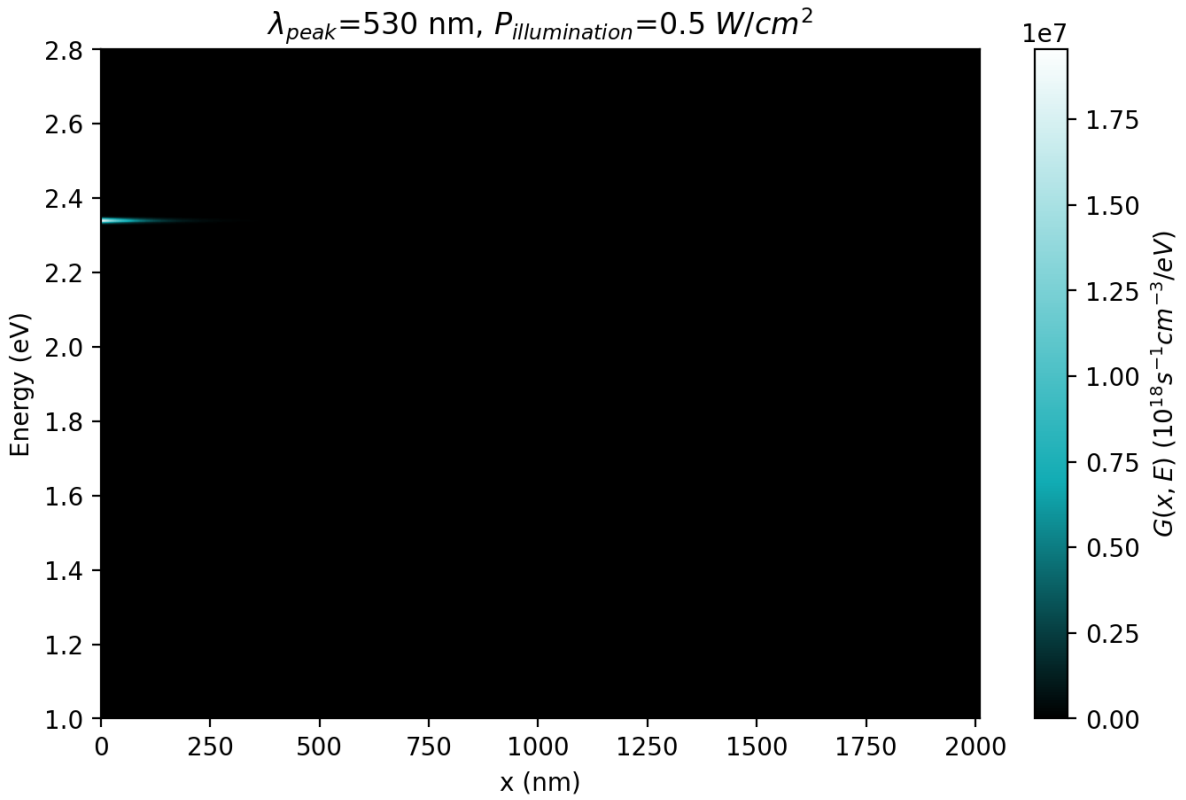


Figure 6.4.18.13: Photogeneration rate $G(x, E)$ at $T = 300K$ and $P = 0.5 \cdot 10^{4}$ W/cm^2

ing 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

From GDSII to Transmission Workflow

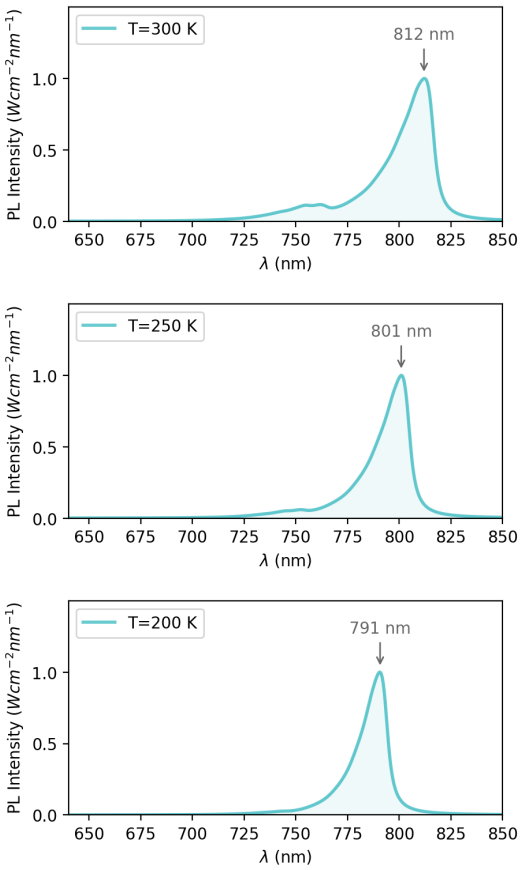


Figure 6.4.18.14: Normalized luminescence spectra with highlighted peak wavelength at each temperature (200 K, 250 K and 300 K), when illuminated by P = $0.5 \cdot 10^4$ W/cm^2.

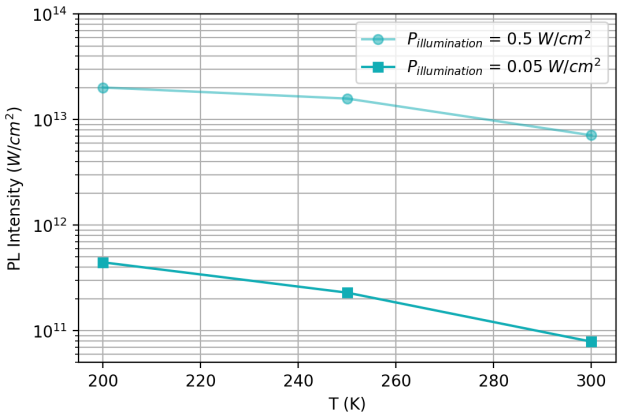


Figure 6.4.18.15: Total emitted intensity as a function of temperature.

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 6.4.18.16.

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

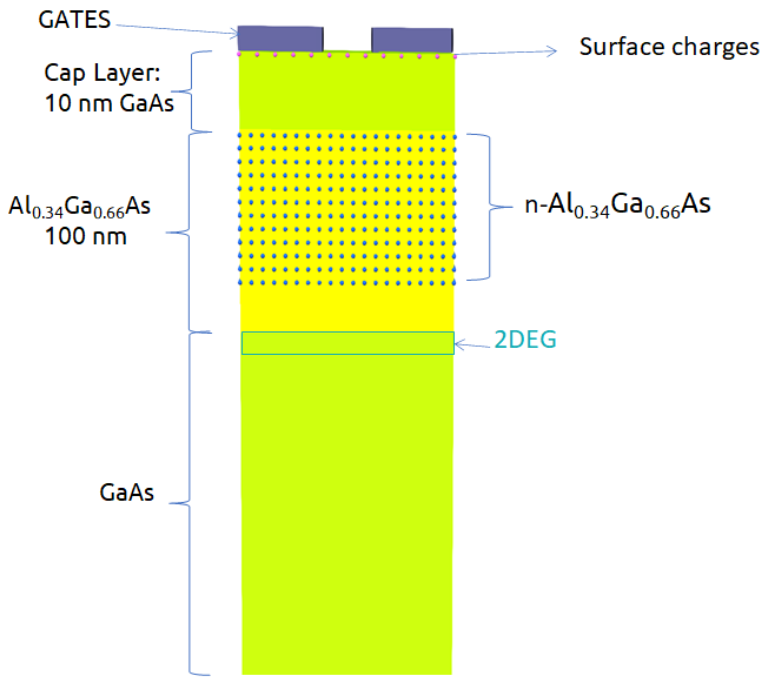


Figure 6.4.18.16: Device under simulation

gates with different geometries are directly deposited on the top of surface. Figure 6.4.18.17 presents several geometries used for testing this methodology.

Work flow

Figure 6.4.18.18 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 of the sample, a simple 1D simulation can be used. Nevertheless 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

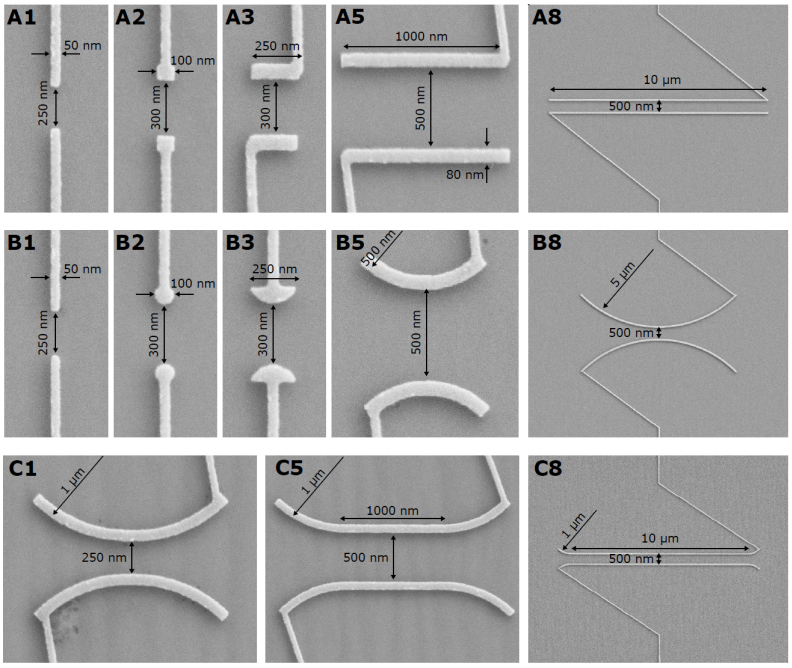


Figure 6.4.18.17: Geometry of the gates used to test this methodology. Source: *[Chatzikyriakou_PhysRevResearch_2022]*

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 6.4.18.16 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:

```
# --- BEGINNING OF THE GATE REGION --- #
```

and

```
# --- END OF THE GATE REGION --- #
```

as in this snapshot of the template file:

```
201 # --- BEGINNING OF THE GATE REGION --- #
202
203
204 # ANY LINE BETWEEN THESE TWO IDENTIFIERS WILL BE REPLACED BY THE
```

(continues on next page)

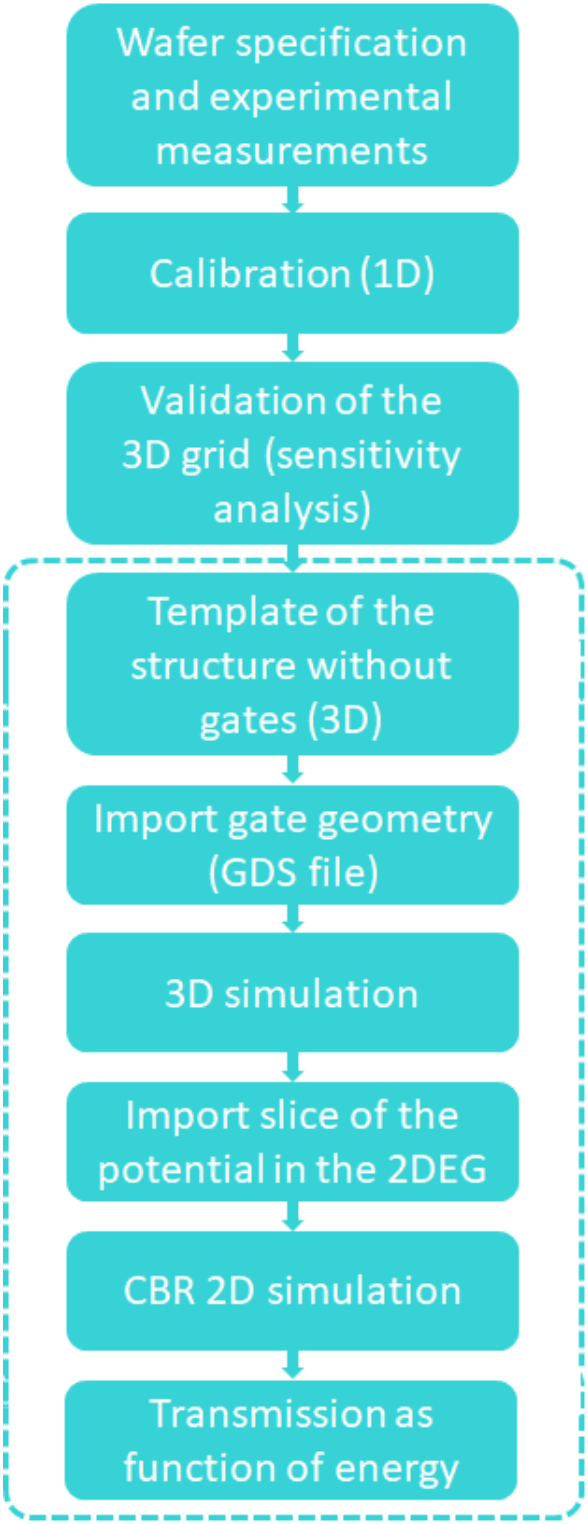


Figure 6.4.18.18: Workflow proposed by this methodology

(continued from previous page)

```
# 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 6.4.18.19 presents the imported slice of the potential. The dashed lines represent the leads of the structure.

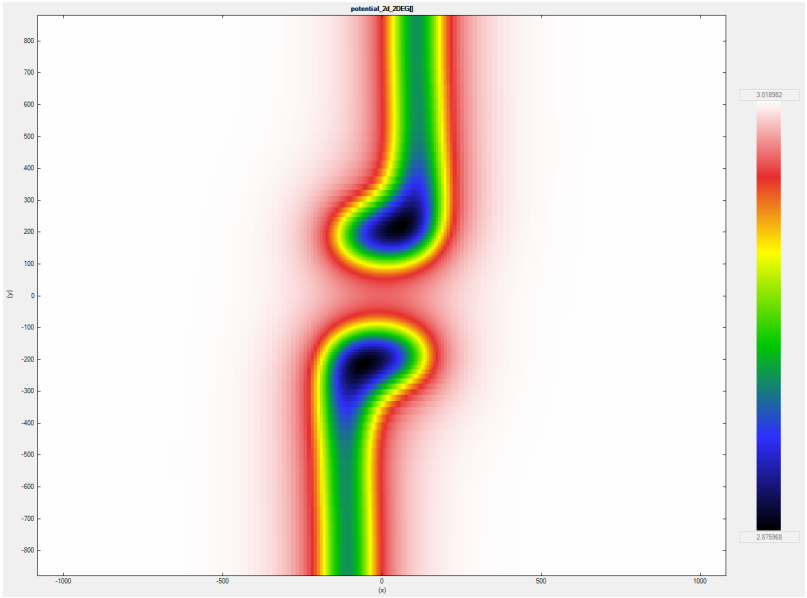


Figure 6.4.18.19: One slice of the potential in the 2DEG region

5. Plotting the transmission through the channel

Figure 6.4.18.19 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.

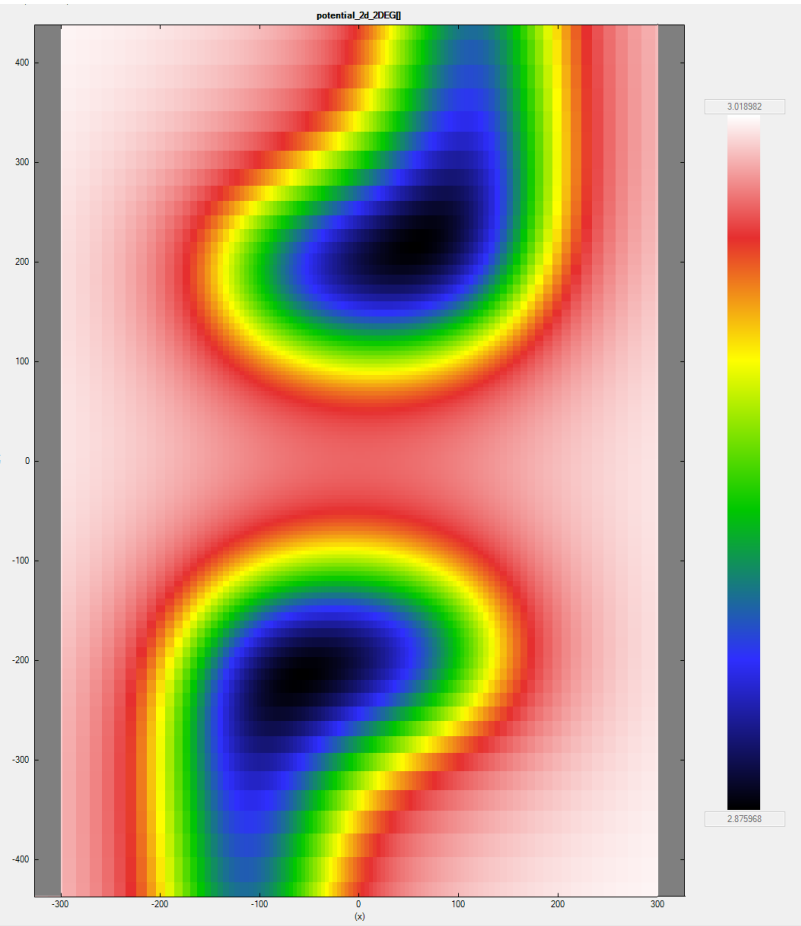


Figure 6.4.18.20: Portion of the slice of the imported potential shown in Figure 6.4.18.19 that will actually be used in the computation of the conductance in the channel in the 2DEG.

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 6.4.18.21.

The stepwise behavior of the transmission is consequence of the fact that the conductance is quantized.

Acknowledgment

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: 17/07/2024

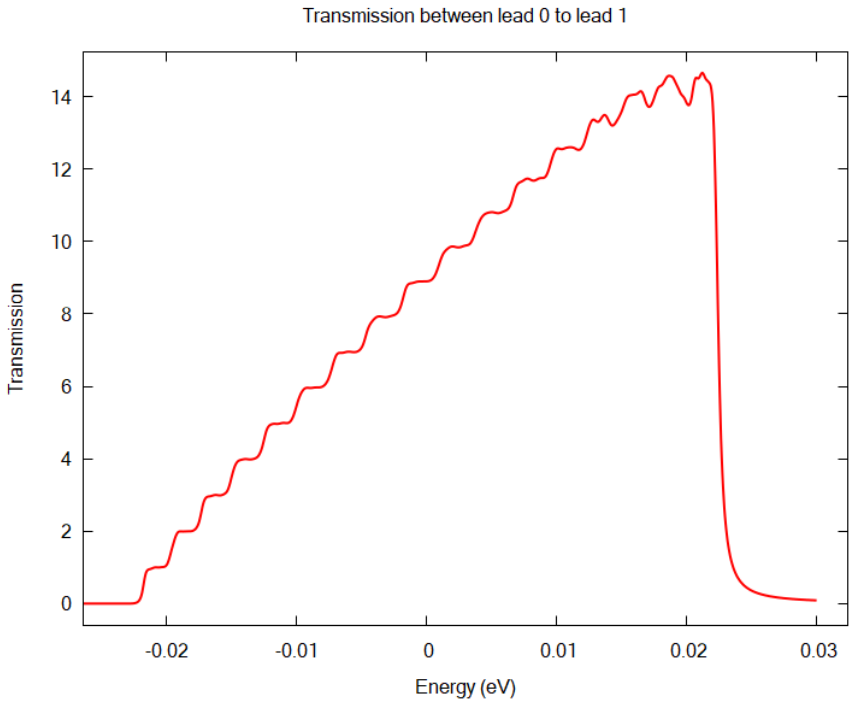


Figure 6.4.18.21: Transmission in the 2DEG region between two leads for $V_{gate} = -1.03\ V$.

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 *— SOON/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's consider a substrate made of any material having zincblende or diamond structure with the surface (111) being prepared for the epitaxial growth. Let's 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 6.4.18.22), 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.

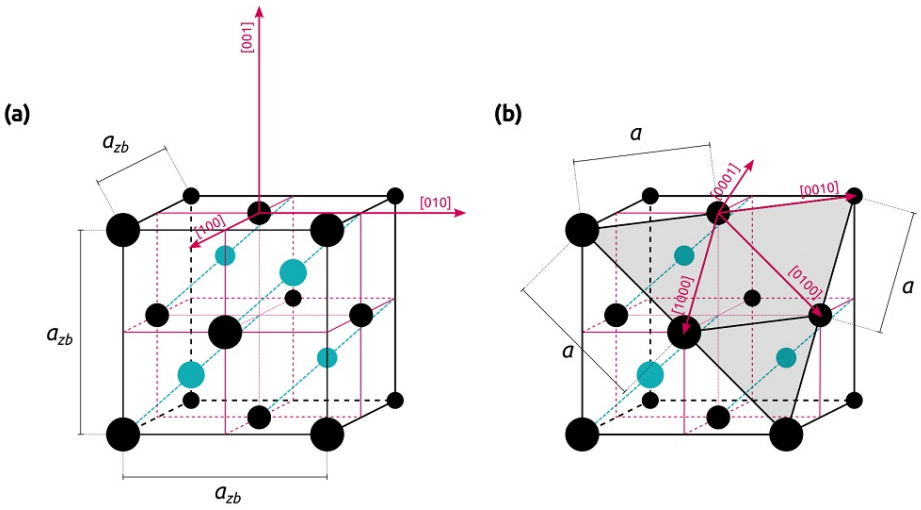


Figure 6.4.18.22: Conventional unit cell of a zincblende crystal or diamond. (a) Crystallographic directions of cubic crystals are plotted. Lattice constant of the crystal is denoted as a_{zb} (b) The unit cell is sliced through one of (111) planes. Crystallographic directions of wurtzite crystals which 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

$$a = \frac{a_{zb}}{\sqrt{2}} \tag{6.4.18.1}$$

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

$$a = \frac{a_{Si}}{\sqrt{2}} = \frac{5.4304 \text{ Å}}{\sqrt{2}} = 3.8399 \text{ Å} \tag{6.4.18.2}$$

at room temperature and the expansion coefficient

$$a_{exp} = \frac{a_{exp,Si}}{\sqrt{2}} = \frac{1.8138 \times 10^{-5} \text{ Å/K}}{\sqrt{2}} = 1.2826 \times 10^{-5} \text{ Å/K}. \tag{6.4.18.3}$$

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

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

6.5 Keywords

6.5.1 postprocessor{ }

Calling sequence

`postprocessor{ }`

Properties

- using: optional within the scope
- 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.

Examples

```
postprocessor{
    datafile = "query.bat"
    call = "query.bat"
    goto_output = yes
}

!DATA
# some list of commands here
```

Nested keywords

- *datafile*
- *goto_output*
- *call*

datafile

Calling sequence

`postprocessor{ datafile }`

Properties

- using: optional within the scope
- 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
```

---

goto_output

Calling sequence

`postprocessor{ goto_output }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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
}
```

---

call

Calling sequence

```
postprocessor{ call }
```

Properties

- using: optional within the scope
- 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.

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
}
```

6.5.2 import{ }

Calling sequence

```
import{ }
```

Properties

- using: optional within the scope
- 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{...}
```

(continues on next page)

(continued from previous page)

```
    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{ }*

---

directory

Calling sequence

`import{ directory }`

Properties

- using: optional within the scope
- 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\\"
```

(continues on next page)

(continued from previous page)

```
    file{...}
}
```

---

file{ }

Calling sequence
`import{ file{ } }`

Properties

- using: optional within the scope
- items: no constraints

Functionality
—

Example

```
import{
    file{...}
}
```

---

file{ name }

Calling sequence
`import{ file{ name } }`

Properties

- using: required within the scope
- 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"
        ...
    }
}
```

---

file{ filename }

Calling sequence

```
import{ file{ filename } }
```

Properties

- using: required within the scope
- 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"
        ...
    }
}
```

file{ format }

Calling sequence

`import{ file{ format } }`

Properties

- using: required within the scope
- type: choice
- choices: `AVS`; `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
    }
}
```

---

file{ scale }

Calling sequence

`import{ file{ scale } }`

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- default: `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
    }
}
```

---

file{ number_of_dimensions }

Calling sequence

`import{ file{ number_of_dimensions } }`

Properties

- using: optional within the scope
- type: integer
- values: `{1, 2, 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
    }
}
```

---

analytic_function{ }

Calling sequence

`import{ analytic_function{ } }`

Properties

- using: optional within the scope
- 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 = ...
    }
}
```

---

analytic_function{ name }

Calling sequence

```
import{ analytic_function{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

Name for referencing the imported function in the input file.

Example

```
import{
    analytic_function{
        name = "Distribution_FD"
        function = ...
    }
}
```

---

analytic_function{ function }

Calling sequence

```
import{ analytic_function{ function } }
```

Properties

- using: optional within the scope
- 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)
    }
}
```

---

analytic_function{ label }

Calling sequence

`import{ analytic_function{ label } }`

Properties

- using: optional within the scope
- 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"
    }
}
```

---

analytic_function{ component{ } }

Calling sequence

```
import{ analytic_function{ component{ } } }
```

Properties

- using: optional within the scope
- 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{...}
    }
}
```

---

analytic_function{ component{ function_i } }

Calling sequence

```
import{ analytic_function{ component{ function_i } } }
```

Properties

- using: optional within the scope
- 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)
        }
    }
}
```

---

analytic_function{ component{ label } }

Calling sequence

```
import{ analytic_function{ component{ label } } }
```

Properties

- using: optional within the scope
- 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"
        }
    }
}
```

---

output_imports{ }

Calling sequence

```
import{ output_imports{ } }
```

Properties

- using: optional within the scope
- 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 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 $\vert\,\vert$
`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

6.5.3 output{ }

Calling sequence

`output{ }`

Properties

- using: optional within the scope
- 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 } }*

---

directory

Calling sequence

```
output{ directory }
```

Properties

- using: optional within the scope
- type: character string

Functionality

Defines alternative output directory. Using this path is controlled by *mandatory_path*

Example

```
output{
    directory = "../output/the_best_simulation"
}
```

---

mandatory_path

Calling sequence

```
output{ mandatory_path }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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.

---

set_origin{ }

Calling sequence

```
output{ set_origin{ } }
```

Properties

- using: optional within the scope
- 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.

---

set_origin{ x }

Calling sequence

```
output{ set_origin{ x } }
```

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- unit: nm
- default: `0`

Functionality

Defines x-coordinate of the origin of the output coordinate system r_{ori} within the coordinate system of the simulation.

---

set_origin{ y }

Calling sequence

```
output{ set_origin{ y } }
```

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- unit: nm
- default: `0`

Functionality

Defines y-coordinate of the origin of the output coordinate system r_{ori} within the coordinate system of the simulation.

---

set_origin{ z }

Calling sequence

```
output{ set_origin{ z } }
```

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- unit: nm
- default: `0`

Functionality

Defines z-coordinate of the origin of the output coordinate system r_{ori} within the coordinate system of the simulation.

---

format2D

Calling sequence

```
output{ format2D }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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 6.5.3.1: 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_file` ...	`VTKAscii` + `AvsAscii_one_file`
`VTKAscii_AvsBinary` ...	`VTKAscii` + `AvsBinary`
`VTKAscii_AvsBinary_one_file` ...	`VTKAscii` + `AvsBinary_one_file`
`Origin`	Origin file format (Origin steering files `*.plt`, data files `*.dat`)

---

format3D

Calling sequence

```
output{ format3D }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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 6.5.3.2: 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_file` ...	`VTKAscii` + `AvsAscii_one_file`
`VTKAscii_AvsBinary` ...	`VTKAscii` + `AvsBinary`
`VTKAscii_AvsBinary_one_file` ...	`VTKAscii` + `AvsBinary_one_file`
`Origin`	Origin file format (Origin steering files `*.plt`, data files `*.dat`)

---

silent

Calling sequence

```
output{ silent }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

Functionality

If set to `no` then prints additional warnings concerning output.

---

write_avs_v

Calling sequence

```
output{ write_avs_v }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Functionality

Outputs AVS steering file `.v`.

---

write_origin_plt

Calling sequence
`output{ write_origin_plt }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Functionality
Outputs Origin steering file `.plt`.

---

write_gnuplot_plt

Calling sequence
`output{ write_gnuplot_plt }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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.

---

use_gnuplot_one_file

Calling sequence
`output{ use_gnuplot_one_file }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Functionality
If `yes` then all information (metadata and data) necessary for the gnuplot figure is contained in one file.

---

only_sections

Calling sequence

```
output{ only_sections }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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.

---

section{ }

Calling sequence

```
output{ section{ } }
```

Properties

- using: optional within the scope
- 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
    }
}
```

---

section{ name }

Calling sequence

`output{ section{ name } }`

Properties

- using: required within the scope
- type: character string

Functionality

Defines a suffix to a name of the generated output file.

---

section{ range_x }

Calling sequence

`output{ section{ range_x } }`

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: no constraints
- default: `[0.0, 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.

---

section{ range_y }

Calling sequence

```
output{ section{ range_y } }
```

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: no constraints
- default: `[0.0, 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.

---

section{ range_z }

Calling sequence

```
output{ section{ range_z } }
```

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: no constraints
- default: `[0.0, 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.

---

section1D{ }

Calling sequence

```
output{ section1D{ } }
```

Properties

- using: optional within the scope
- 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
```

(continues on next page)

(continued from previous page)

```
    }
}
```

section1D{ name }

Calling sequence

```
output{ section1D{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

Defines a suffix to a name of the generated output file.

section1D{ x }

Calling sequence

```
output{ section1D{ x } }
```

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- default: `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.

section1D{ y }

Calling sequence

```
output{ section1D{ y } }
```

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- default: `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.

section1D{ z }

Calling sequence

```
output{ section1D{ z } }
```

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- default: `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.

---

section1D{ range_x }

Calling sequence

```
output{ section1D{ range_x } }
```

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: no constraints
- default: `[0.0, 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.

---

section1D{ range_y }

Calling sequence

```
output{ section1D{ range_y } }
```

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: no constraints
- default: `[0.0, 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.

---

section1D{ range_z }

Calling sequence

```
output{ section1D{ range_z } }
```

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: no constraints
- default: `[0.0, 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.

---

section2D{ }

Calling sequence

```
output{ section2D{ } }
```

Properties

- using: optional within the scope
- 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
    }
}
```

material_parameters{ }

Calling sequence

`output{ material_parameters{ } }`

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Defines additional outputs.

material_parameters{ kp_parameters{ } }

Calling sequence

`output{ material_parameters{ kp_parameters{ } } }`

Properties

- using: optional within the scope
- 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]*.

---

material_parameters{ kp_parameters{ boxes } }

Calling sequence

```
output{ material_parameters{ kp_parameters{ boxes } } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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)

---

material_parameters{ spin_orbit_coupling_energies{ } }

Calling sequence

```
output{ material_parameters{ spin_orbit_coupling_energies{ } } }
```

Properties

- using: optional within the scope
- 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]`.

---

material_parameters{ spin_orbit_coupling_energies{ boxes } }

Calling sequence

```
output{ material_parameters{ spin_orbit_coupling_energies{ boxes } } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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)

---

material_parameters{ charge_carrier_masses{ } }

Calling sequence

```
output{ material_parameters{ charge_carrier_masses{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs effective masses of all energy bands used in the simulations in `[m0]`.

---

material_parameters{ charge_carrier_masses{ boxes } }

Calling sequence

```
output{ material_parameters{ charge_carrier_masses{ boxes } } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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)

---

material_parameters{ static_dielectric_constants{ } }

Calling sequence

```
output{ material_parameters{ static_dielectric_constants{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs static relative dielectric constants for zinc blende (1 parameter) and wurtzite (3 parameters).

---

material_parameters{ static_dielectric_constants{ boxes } }

Calling sequence

```
output{ material_parameters{ static_dielectric_constants{ boxes } } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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)

---

material_parameters{ deformation_potentials{ } }

Calling sequence

```
output{ material_parameters{ deformation_potentials{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Output the deformation potentials for zinc blende and wurtzite in `[eV]`.

---

material_parameters{ deformation_potentials{ boxes } }

Calling sequence

```
output{ material_parameters{ deformation_potentials{ boxes } } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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)

6.5.4 run{ }

Calling sequence

`run{ }`

Properties

- using: required within the scope
- 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{}
```

(continues on next page)

(continued from previous page)

```
    quantum_optics{}
}
```

```
run{
    strain{}
    current_poisson{}
    quantum_current_poisson{}
    quantum_optics{}
}
```

Nested keywords

structure_only{ }

Calling sequence

```
run{ structure_only{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, 4, ...}`
- default: `all regions`
- unit: –

Functionality

Defines the highest number of region printed in to the output file.

Example

```
run{
    structure_only{
        last_region = 5
    }
}
```

strain{ }

Calling sequence

`run{ strain{ } }`

Properties

- using: optional within the scope
- 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{}
```

poisson{ }

Calling sequence

```
run{ poisson{ } }
```

Properties

- using: optional within the scope
- 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{}
```

current_poisson{ }

Calling sequence

```
run{ current_poisson{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Dependencies

- The *poisson{ }* must be defined.
- The *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*
- *output_log*
- *output_local_residuals*

---

fermi_limit

Calling sequence

```
run{ current_poisson{ fermi_limit } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, 10.0]`
- default: `2.0`
- unit: eV

Example

```
run{
    current_poisson{
        fermi_limit = 0.5
    }
}

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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

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

- using: optional within the scope
- type: integer
- values: {1, 2, 3, 4, ...}
- default: 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

- using: optional within the scope
- type: integer
- values: {1, 2, 3, 4, ...}
- default: 1
- unit: –

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
    }
```

(continues on next page)

(continued from previous page)

```
}

poisson{}
currents{}
```

---

limit_repetitions

Calling sequence

```
run{ current_poisson{ limit_repetitions } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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{
    current_poisson{
        current_repetitions = yes
    }
}

poisson{}
currents{}
```

---

residual

Calling sequence

```
run{ current_poisson{ residual } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e5` for 1D; `1e3` for 2D; `1e-3` for 3D
- unit: cm^{-2} (1D) / cm^{-1} (2D) / none (3D)

Functionality

Residual occupation changes.

Example

```
run{
    current_poisson{
        residual = 1e4
    }
}

poisson{}
currents{}
```

---

residual_fermi

Calling sequence

`run{ current_poisson{ residual_fermi } }`

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `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
    }
}

poisson{}
currents{}
```

---

alpha_fermi

Calling sequence

`run{ current_poisson{ alpha_fermi } }`

Properties

- using: optional within the scope
- type: real number
- values: `[1e-5, 1.0]`
- default: `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

- using: optional within the scope
- type: integer
- values: {1, 2, 3, 4, ...}
- default: 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

- using: optional within the scope
- type: real number
- values: `[0.1, 1.0]`
- default: `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{}
```

---

output_log

Calling sequence

```
run{ current_poisson{ output_log } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
```

quantum{ }

Calling sequence

`run{ quantum{ } }`

Properties

- using: optional within the scope
- 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{}
```

quantum_density{ }

Calling sequence

```
run{ quantum_density{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Dependencies

- The *quantum{ }* must be defined.
- The *quantum{ 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

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e5` for 1D; `1e3` for 2D; `1e-3` for 3D
- unit: cm^{-2} (1D) / cm^{-1} (2D) / none (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

- using: optional within the scope
- type: integer
- values: `{0, 1, 2, 3, ...}`
- default: `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}

quantum{
    exchange_correlation{}
}
```

---

subspace_iterations

Calling sequence

```
run{ quantum_density{ subspace_iterations } }
```

Properties

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, ..., 1000}`
- default: `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

- using: optional within the scope
- type: real number
- values: `[2.0, ...)`
- default: `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: yes; 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{}
}
```

quantum_poisson{ }

Calling sequence
`run{ quantum_poisson{ } }`

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e5` for 1D; `1e3` for 2D; `1e-3` for 3D
- unit: cm^{-2} (1D) / cm^{-1} (2D) / none (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

- using: optional within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- default: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}

poisson{}
quantum{}
```

---

subspace_iterations

Calling sequence

`run{ quantum_poisson{ subspace_iterations } }`

Properties

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, ..., 1000}`
- default: `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

- using: optional within the scope
- type: real number
- values: `[2.0, ...)`
- default: `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

- using: optional within the scope
- type: real number
- values: `[1e-3, 1.0]`
- default: `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
    }
}
```

(continues on next page)

(continued from previous page)

```
poisson{}
quantum{}
```

---

output_log

Calling sequence

```
run{ quantum_poisson{ output_log } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
```

quantum_current_poisson{ }

Calling sequence

`run{ quantum_current_poisson{ } }`

Properties

- using: optional within the scope
- 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*
- *output_log*
- *output_local_residuals*

---

residual

Calling sequence

```
run{ quantum_current_poisson{ residual } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e5` for 1D; `1e3` for 2D; `1e-3` for 3D
- unit: cm^{-2} (1D) / cm^{-1} (2D) / none (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

- using: optional within the scope
- type: integer
- values: `{0, 1, 2, 3, ...}`
- default: `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: integer
- values: {1, 2, 3, ..., 1000}
- default: 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

- using: optional within the scope
- type: real number
- values: [2.0, ...)
- default: 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

- using: optional within the scope
- type: real number
- values: `[0.0, 10.0]`
- default: `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

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, 4, ...}`
- default: `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`

- default: `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

- using: optional within the scope
- type: real number
- values: `[1e-5, 1.0]`
- default: `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
    }
}

poisson{}
currents{}
quantum{}
```

---

alpha_iterations

Calling sequence

```
run{ quantum_current_poisson{ alpha_iterations } }
```

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, 4, ...}
- default: 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

- using: optional within the scope
- type: real number
- values: [0.1, 1.0]
- default: 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
```

(continues on next page)

(continued from previous page)

```
    }
}

poisson{}
currents{}
quantum{}
```

alpha_potential

Calling sequence

```
run{ quantum_current_poisson{ alpha_potential } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[1e-3, 1.0]`
- default: `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{}
```

output_log

Calling sequence

```
run{ quantum_current_poisson{ output_log } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
```

quantum_optics{ }

Calling sequence

`run{ quantum_optics{ } }`

Properties

- using: optional within the scope
- 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

6.5.5 global{ }

Calling sequence
`global{ }`

Properties

- using: required within the scope
- 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 }*

simulate1D{ }

Calling sequence

```
global{ simulate1D{ } }
```

Properties

- using: 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{}
```

(continues on next page)

(continued from previous page)

```
    ...
}
```

---

simulate2D{ }

Calling sequence

```
global{ simulate2D{ } }
```

Properties

- using: 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{}
    ...
}
```

---

simulate3D{ }

Calling sequence

```
global{ simulate3D{ } }
```

Properties

- using: 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{}
    ...
}
```

---

crystal_zb{ }

Calling sequence

```
global{ crystal_zb{ } }
```

Properties

- using: 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{...}
    ...
}
```

---

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

- using: conditional
- type: vector of 3 integers
- 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]
    }
    ...
}
```

---

crystal_zb{ y_hkl }

Calling sequence

```
global{ crystal_zb{ y_hkl } }
```

Properties

- using: conditional
- type: vector of 3 integers
- 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]
    }
    ...
}
```

---

crystal_zb{ z_hkl }

Calling sequence

```
global{ crystal_zb{ z_hkl } }
```

Properties

- using: conditional
- type: vector of 3 integers
- 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]
    }
    ...
}
```

---

crystal_wz{ }

Calling sequence

```
global{ crystal_wz{ } }
```

Properties

- using: 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{...}
    ...
}
```

---

crystal_wz{ x_hkl }

Calling sequence

```
global{ crystal_wz{ x_hkl } }
```

Properties

- using: conditional
- type: vector of 3 integers
- 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]
    }
    ...
}
```

---

crystal_wz{ y_hkl }

Calling sequence

`global{ crystal_wz{ y_hkl } }`

Properties

- using: conditional
- type: vector of 3 integers
- 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]
    }
    ...
}
```

---

crystal_wz{ z_hkl }

Calling sequence

`global{ crystal_wz{ z_hkl } }`

Properties

- using: conditional
- type: vector of 3 integers
- 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]
    }
    ...
}
```

---

crystal_wz{ rotation_c_a_ratio_use_substrate }

Calling sequence

`global{ crystal_wz{ rotation_c_a_ratio_use_substrate } }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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
        ...
    }
    ...
}
```

---

crystal_wz{ rotation_c_a_ratio }

Calling sequence

```
global{ crystal_wz{ rotation_c_a_ratio } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.1, 1.0]`
- default: `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
        ...
    }
    ...
}
```

---

substrate{ }

Calling sequence

```
global{ substrate{ } }
```

Properties

- using: required within the scope
- 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{...}
    ...
}
```

---

substrate{ name }

Calling sequence

```
global{ substrate{ name } }
```

Properties

- using: required within the scope
- 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"
    }
    ...
}
```

---

substrate{ alloy_x }

Calling sequence

```
global{ substrate{ alloy_x } }
```

Properties

- using: conditional
- type: real number
- values: [0.0, 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
    }
    ...
}
```

---

substrate{ alloy_y }

Calling sequence

```
global{ substrate{ alloy_y } }
```

Properties

- using: conditional
- type: real number
- values: `[0.0, 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
    }
    ...
}
```

---

substrate{ alloy_z }

Calling sequence

```
global{ substrate{ alloy_z } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, 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
    }
    ...
}
```

---

temperature

Calling sequence
`global{ temperature }`

Properties

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: K

Functionality
Specifies temperature of the crystal lattice and electrons.

Example

```
global{
    temperature = 300
    ...
}
```

---

temperature_dependent_bandgap

Calling sequence
`global{ temperature_dependent_bandgap }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    ...
}
```

---

temperature_dependent_lattice

Calling sequence

`global{ temperature_dependent_lattice }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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.

$$a\,(T) = a(300\ \mathrm{K}) + a_{\mathrm{exp}}\,(T - 300)\,,$$

where a_{exp} is the expansion coefficient defined in the database and properly interpolated for alloys.

Example

```
global{
    temperature_dependent_lattice = no
    ...
}
```

---

magnetic_field{ }

Calling sequence

`global{ magnetic_field{ } }`

Properties

- using: 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{...}
    ...
}
```

---

magnetic_field{ direction }

Calling sequence

```
global{ magnetic_field{ direction } }
```

Properties

- using: conditional
- type: vector of 3 real numbers
- values: no constraints
- default 1D: `[1.0, 0.0, 0.0]`
- default 2D: `[0.0, 0.0, 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]
```

(continues on next page)

(continued from previous page)

```
        strength  = 5.3
    }
    ...
}
```

---

magnetic_field{ strength }

Calling sequence

`global{ magnetic_field{ strength } }`

Properties

- using: required within the scope
- type: real number
- values: no constraints
- unit: $\mathrm{T} = \mathrm{Vs/m^2}$

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
    }
    ...
}
```

---

periodic{ }

Calling sequence

`global{ periodic{ } }`

Properties

- using: 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{...}
    ...
}
```

---

periodic{ x }

Calling sequence

```
global{ periodic{ x } }
```

Properties

- using: required within the scope
- type: choice
- choices: `yes`; `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
    }
    ...
}
```

---

periodic{ y }

Calling sequence

```
global{ periodic{ y } }
```

Properties

- using: conditional
- type: choice
- choices: `yes`; `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
    }
    ...
}
```

---

periodic{ z }

Calling sequence

```
global{ periodic{ z } }
```

Properties

- using: conditional
- type: choice
- choices: yes; 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
    }
    ...
}
```

6.5.6 impurities{ }

Calling sequence
`impurities{ }`

Properties

- using: optional within the scope
- 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 }*
- *acceptor{ }*
- *acceptor{ name }*
- *acceptor{ degeneracy }*
- *acceptor{ energy }*
- *charge{ }*
- *charge{ name }*
- *charge{ type }*

---

donor{ }

Calling sequence
`impurities{ donor{ } }`

Properties

- using: optional within the scope
- items: no constraints

Functionality
Defines properties of donors.

Example

```
impurities{
    donor{...}
    donor{...}
}
```

---

donor{ name }

Calling sequence
`impurities{ donor{ name } }`

Properties

- using: required within the scope
- type: character string

Functionality
Name of the impurity for referencing during definition of the structure

Example

```
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

donor{ degeneracy }

Calling sequence
`impurities{ donor{ degeneracy } }`

Properties

- using: required within the scope
- type: integer
- values: `{1, 2, 3, ..., 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
        ...
    }
}
```

---

donor{ energy }

Calling sequence

`impurities{ donor{ energy } }`

Properties

- using: required within the scope
- type: real number
- values: no constraints
- unit: eV

Functionality

Ionization (activation) energy of the impurity. 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
    }
}
```

---

acceptor{ }

Calling sequence

`impurities{ acceptor{ } }`

Properties

- using: optional within the scope
- items: no constraints

Functionality

Defines properties of acceptors.

Example

```
impurities{
    acceptor{...}
    acceptor{...}
}
```

---

acceptor{ name }

Calling sequence

```
impurities{ acceptor{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

Name of the impurity for referencing during definition of the structure

Example

```
impurities{
    acceptor{
        name = "p-C"
        ...
    }
}
```

---

acceptor{ degeneracy }

Calling sequence

```
impurities{ acceptor{ degeneracy } }
```

Properties

- using: required within the scope
- type: integer
- values: {1, 2, 3, ..., 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
        ...
    }
}
```

---

acceptor{ energy }

Calling sequence

```
impurities{ acceptor{ energy } }
```

Properties

- using: required within the scope
- 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
    }
}
```

---

charge{ }

Calling sequence

```
impurities{ charge{ } }
```

Properties

- using: optional within the scope
- 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{...}
}
```

---

charge{ name }

Calling sequence

`impurities{ charge{ name } }`

Properties

- using: required within the scope
- type: character string

Functionality

A reference name.

Example

```
impurities{
    charge{
        name = "positive_charges"
        ...
    }
}
```

---

charge{ type }

Calling sequence

`impurities{ charge{ type } }`

Properties

- using: required within the scope
- type: choice
- choices: `positive`; `negative`

Functionality

Defines sign of the charge.

Example

```
impurities{
    charge{
        name = "positive_charges"
        type = positive
    }
}
```

6.5.7 contacts{ }

Calling sequence

```
contacts{ }
```

Properties

- using: required within the scope
- 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 $\phi(x)$ at the boundaries of the entire simulation. These are either, default, **Neumann** boundary conditions $\frac{\mathrm{d}}{\mathrm{d}x}\phi(x) = 0$ 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.

Examples

```
contacts{
    fermi{...}
}
```

```
contacts{
    schottky{...}
    ohmic{...}
}
```

Nested keywords

- *vacuum_level*
- *schottky{ }*
- *schottky{ name }*
- *schottky{ bias }*
- *schottky{ steps }*
- *schottky{ barrier }*
- *schottky{ work_function }*

- *ohmic{ }*
- *ohmic{ name }*
- *ohmic{ bias }*
- *ohmic{ steps }*
- *ohmic{ shift }*
- *fermi{ }*
- *fermi{ name }*
- *fermi{ bias }*
- *fermi{ steps }*
- *fermi_electron{ }*
- *fermi_electron{ name }*
- *fermi_electron{ bias }*
- *fermi_electron{ steps }*
- *fermi_hole{ }*
- *fermi_hole{ name }*
- *fermi_hole{ bias }*
- *fermi_hole{ steps }*
- *charge_neutral{ }*
- *charge_neutral{ name }*
- *charge_neutral{ bias }*
- *charge_neutral{ steps }*
- *zero_field{ }*
- *zero_field{ name }*
- *zero_field{ bias }*
- *zero_field{ steps }*
- *long_directory_names*

vacuum_level

Calling sequence

```
contacts{ vacuum_level }
```

Properties

- using: optional within the scope
- type: real number
- values: `[-1e2, 1e2]`
- default: `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.

Example

```
contacts{
    vacuum_level = 6.2
    schottky{...}
}
```

---

schottky{ }

Calling sequence

```
contacts{ schottky{ } }
```

Properties

- using: 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.

Example

```
contacts{
    schottky{...}
}
```

---

schottky{ name }

Calling sequence

```
contacts{ schottky{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

A name of the contact for referencing it in *structure{ region{ contact{} } }*

Example

```
contacts{
    schottky{
        name = "schottky_contact"
        ...
    }
}
```

---

schottky{ bias }

Calling sequence

```
contacts{ schottky{ bias } }
```

Properties

- using: required within the scope
- type: vector of up to 100 real numbers
- values: no constraints
- unit: V

Functionality

Explicitly defined bias U

Example

```
contacts{
    schottky{
        name = "schottky_1"
        bias = 0.0
        ...
    }
    schottky{
        name = "schottky_2"
        bias = [ 0.0, 0.5 ]
        ...
    }
    schottky{
        name = "schottky_3"
        bias = [ 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 ]
        ...
    }
}
```

---

schottky{ steps }

Calling sequence

```
contacts{ schottky{ steps } }
```

Properties

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, ..., 999}`
- default: `1`
- unit: –

Functionality

Number of biases between consecutive bias points

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.

Example

```
contacts{
    schottky{
        name = "schottky_1"
        bias = [ 0.0, 0.5 ]
        steps = 10
        ...
    }
    schottky{
        name = "schottky_2"
        bias = [ 0.0, 0.1, 1.0]
        steps = 10
        ...
    }
}
```

---

schottky{ barrier }

Calling sequence

```
contacts{ schottky{ barrier } }
```

Properties

- using: conditional
- type: real number
- values: `[-1e2, 1e2]`
- unit: eV

Functionality

A Schottky barrier B - a difference between conduction band minimum and the Fermi level

Example

```
contacts{
    schottky{
        name = "schottky"
        bias = 0.0
        barrier = 0.2
    }
}
```

---

schottky{ work_function }

Calling sequence

`contacts{ schottky{ work_function } }`

Properties

- using: conditional
- type: real number
- values: `[-1e2, 1e2]`
- 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.

Example

```
contacts{
    schottky{
        name = "schottky"
        bias = 0.0
        work_function = 0.2
    }
}
```

---

ohmic{ }

Calling sequence

```
contacts{ ohmic{ } }
```

Properties

- using: 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.

Example

```
contacts{
    ohmic{...}
}
```

---

ohmic{ name }

Calling sequence

```
contacts{ ohmic{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

A name of the contact for referencing it in *structure{ region{ contact{} } }*.

Example

```
contacts{
    ohmic{
        name = "ohmic_contact"
        ...
    }
}
```

---

ohmic{ bias }

Calling sequence

`contacts{ ohmic{ bias } }`

Properties

- using: required within the scope
- type: vector of up to 100 real numbers
- values: no constraints
- unit: V

Functionality

Explicitly defined bias U.

Example

```
contacts{
    ohmic{
        name = "ohmic_1"
        bias = 0.0
    }
    ohmic{
        name = "ohmic_2"
        bias = [ 0.0, 0.5 ]
    }
    ohmic{
        name = "ohmic_3"
        bias = [ 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 ]
    }
}
```

---

ohmic{ steps }

Calling sequence

`contacts{ ohmic{ steps } }`

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, ..., 999}
- default: 1
- unit: –

Functionality

Number of biases between consecutive bias points.

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.

Example

```
contacts{
    ohmic{
        name = "ohmic_1"
        bias = [ 0.0, 0.5 ]
        steps = 10
    }
    ohmic{
        name = "ohmic_2"
        bias = [ 0.0, 0.1, 1.0]
        steps = 10
    }
}
```

---

ohmic{ shift }

Calling sequence

`contacts{ ohmic{ shift } }`

Properties

- using: optional within the scope
- type: real number
- values: no constraints
- default: `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.

Example

```
contacts{
    ohmic{
        name = "ohmic_contact"
        bias = [ 0.0, 0.5 ]
        shift = 0.05
    }
}
```

---

fermi{ }

Calling sequence

```
contacts{ fermi{ } }
```

Properties

- using: 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 $\phi(x)$.

Example

```
contacts{
    fermi{...}
}
```

---

fermi{ name }

Calling sequence

```
contacts{ fermi{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

A name of the contact for referencing it in *structure{ region{ contact{} } }*

> **Attention:** When triggered, both Poisson and Schrödinger equations are solved in the regions with these boundary conditions.

Example

```
contacts{
    fermi{
        name = "fermi_contact"
        ...
    }
}
```

---

fermi{ bias }

Calling sequence

```
contacts{ fermi{ bias } }
```

Properties

- using: required within the scope
- type: vector of up to 100 real numbers
- values: no constraints
- unit: V

Functionality

Explicitly defined bias U.

Example

```
contacts{
    fermi{
        name = "fermi_1"
        bias = 0.0
    }
    fermi{
        name = "fermi_2"
        bias = [ 0.0, 0.5 ]
    }
    fermi{
        name = "fermi_3"
        bias = [ 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 ]
    }
}
```

---

fermi{ steps }

Calling sequence

```
contacts{ fermi{ steps } }
```

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, ..., 999}
- default: 1
- unit: –

Functionality

Number of biases between consecutive bias points

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.

Example

```
contacts{
    fermi{
        name = "fermi_1"
        bias = [ 0.0, 0.5 ]
        steps = 10
    }
    fermi{
        name = "fermi_2"
        bias = [ 0.0, 0.1, 1.0]
        steps = 10
    }
}
```

---

fermi_electron{ }

Calling sequence

`contacts{ fermi_electron{ } }`

Properties

- using: 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.

Example

```
contacts{
    fermi_electron{...}
}
```

---

fermi_electron{ name }

Calling sequence

```
contacts{ fermi_electron{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

A name of the contact for referencing it in *structure{ region{ contact{} } }*

Example

```
contacts{
    fermi_electron{
        name = "fermi_el"
        ...
    }
}
```

---

fermi_electron{ bias }

Calling sequence

```
contacts{ fermi_electron{ bias } }
```

Properties

- using: required within the scope
- type: vector of up to 100 real numbers
- values: no constraints
- unit: V

Functionality

Explicitly defined bias U.

Example

```
contacts{
    fermi_electron{
        name = "fermi_el_1"
        bias = 0.0
    }
    fermi_electron{
        name = "fermi_el_2"
        bias = [ 0.0, 0.5 ]
    }
    fermi_electron{
        name = "fermi_el_3"
        bias = [ 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 ]
    }
}
```

---

fermi_electron{ steps }

Calling sequence

```
contacts{ fermi_electron{ steps } }
```

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, ..., 999}
- default: 1
- unit: –

Functionality

Number of biases between consecutive bias points.

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.

Examples

```
contacts{
    fermi_electron{
        name = "fermi_el_1"
        bias = [ 0.0, 0.5 ]
        steps = 10
    }
    fermi_electron{
        name = "fermi_el_2"
        bias = [ 0.0, 0.1, 1.0]
        steps = 10
    }
}
```

---

fermi_hole{ }

Calling sequence

```
contacts{ fermi_hole{ } }
```

Properties

- using: 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 $E_{F,e}(x)$.

Example

```
contacts{
    fermi_hole{...}
}
```

---

fermi_hole{ name }

Calling sequence

```
contacts{ fermi_hole{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

A name of the contact for referencing it in *structure{ region{ 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.

Example

```
contacts{
    fermi_hole{
        name = "fermi_hl"
        ...
    }
}
```

---

fermi_hole{ bias }

Calling sequence

```
contacts{ fermi_hole{ bias } }
```

Properties

- using: required within the scope
- type: vector of up to 100 real numbers
- values: no constraints
- unit: V

Functionality

Explicitly defined bias U.

Example

```
contacts{
    fermi_hole{
        name = "fermi_hl_1"
        bias = 0.0
    }
    fermi_hole{
        name = "fermi_hl_2"
        bias = [ 0.0, 0.5 ]
    }
    fermi_hole{
        name = "fermi_hl_3"
        bias = [ 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 ]
    }
}
```

fermi_hole{ steps }

Calling sequence

`contacts{ fermi_hole{ steps } }`

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, ..., 999}
- default: 1
- unit: –

Functionality

Number of biases between consecutive bias points

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.

Examples

```
contacts{
    fermi_hole{
        name = "fermi_hl_1"
        bias = [ 0.0, 0.5 ]
        steps = 10
    }
    fermi_hole{
        name = "fermi_hl_2"
        bias = [ 0.0, 0.1, 1.0]
        steps = 10
    }
}
```

charge_neutral{ }

Calling sequence

```
contacts{ charge_neutral{ } }
```

Properties

- using: 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.

Example

```
contacts{
    charge_neutral{...}
}
```

charge_neutral{ name }

Calling sequence

```
contacts{ charge_neutral{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

A name of the contact for referencing it in *structure{ region{ contact{} } }*.

Example

```
contacts{
    charge_neutral{
        name = "charge_neutral_contact"
        ...
    }
}
```

---

charge_neutral{ bias }

Calling sequence

`contacts{ charge_neutral{ bias } }`

Properties

- using: required within the scope
- type: vector of up to 100 real numbers
- values: no constraints
- unit: V

Functionality

Explicitly defined bias U.

Example

```
contacts{
    charge_neutral{
        name = "charge_neutral_1"
        bias = 0.0
    }
    charge_neutral{
        name = "charge_neutral_2"
        bias = [ 0.0, 0.5 ]
    }
    charge_neutral{
        name = "charge_neutral_3"
        bias = [ 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 ]
    }
}
```

---

charge_neutral{ steps }

Calling sequence

`contacts{ charge_neutral{ steps } }`

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, ..., 999}
- default: 1
- unit: –

Functionality

Number of biases between consecutive bias points.

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.

Example

```
contacts{
    charge_neutral{
        name = "charge_neutral_1"
        bias = [ 0.0, 0.5 ]
        steps = 10
    }
    charge_neutral{
        name = "charge_neutral_2"
        bias = [ 0.0, 0.1, 1.0]
        steps = 10
    }
}
```

zero_field{ }

Calling sequence

```
contacts{ zero_field{ } }
```

Properties

- using: 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.

Example

```
contacts{
    zero_field{...}
}
```

zero_field{ name }

Calling sequence

```
contacts{ zero_field{ name } }
```

Properties

- using: required within the scope
- type: character string

Functionality

A name of the contact for referencing it in *structure{ region{ 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.

Example

```
contacts{
    zero_field{
        name = "zero_field_contact"
        ...
    }
}
```

---

zero_field{ bias }

Calling sequence

```
contacts{ zero_field{ bias } }
```

Properties

- using: required within the scope
- type: vector of up to 100 real numbers
- values: no constraints
- unit: V

Functionality

Explicitly defined bias U

Example

```
contacts{
    zero_field{
        name = "zero_field_1"
        bias = 0.0
    }
    zero_field{
        name = "zero_field_2"
        bias = [ 0.0, 0.5 ]
    }
    zero_field{
        name = "zero_field_3"
```

(continues on next page)

(continued from previous page)

```
        bias = [ 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 ]
    }
}
```

---

zero_field{ steps }

Calling sequence

```
contacts{ zero_field{ steps } }
```

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, ..., 999}
- default: 1
- unit: –

Functionality

Number of biases between consecutive bias points

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.

Example

```
contacts{
    zero_field{
        name = "zero_field_1"
        bias = [ 0.0, 0.5 ]
        steps = 10
    }
    zero_field{
        name = "zero_field_2"
        bias = [ 0.0, 0.1, 1.0]
        steps = 10
    }
}
```

---

long_directory_names

Calling sequence

```
contacts{ long_directory_names }
```

Properties

- using: optional within the scope
- type: choice
- choices: yes; 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.

Example

```
contacts{
    long_directory_names = yes
    fermi{...}
}
```

6.5.8 structure{ }

Calling sequence

`structure{ }`

Properties

- using: required within the scope
- items: exactly 1

Functionality

definition of device structure (including doping{})

Example

```
structure{
    region{...}
}
```

Nested keywords

region{ }

Calling sequence

`structure{ region{ } }`

Properties

- using: required within the scope
- 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

- using: optional within the scope
- type: integer
- unit: –
- values: {0, 1, 2, 3, ...}

Functionality

Additional arbitrary index assigned to a region.

Example

```
structure{
    region{
        user_index = 1
        ...
    }
}
```

array_x{ }

Calling sequence

```
structure{ region{ array_x } }
```

Properties

- using: 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

- using: required within the scope
- 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

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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

- using: optional within the scope
- type: integer
- values: {..., -3, -2, -1, 0}
- unit: –
- default: 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

- using: 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

- using: required within the scope
- 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

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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

- using: optional within the scope
- type: integer
- values: {..., -3, -2, -1, 0}
- unit: –
- default: 0

Functionality

Number of regions added in the negative direction of the y-axis.

Example

```
structure{
    region{
        array_y{
            shift = 11.0
            max = 3
            min = 2
        }
        ...
    }
}
```

array_z{ }

Calling sequence

`structure{ region{ array_z } }`

Properties

- using: 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

- using: required within the scope
- 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

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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

- using: optional within the scope
- type: integer
- values: {..., -3, -2, -1, 0}
- unit: –
- default: 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
```

(continues on next page)

(continued from previous page)

```
        }
        ...
    }
}
```

array2_x{ }

Calling sequence

```
structure{ region{ array2_x } }
```

Properties

- using: 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

- using: required within the scope
- 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

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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

- using: optional within the scope
- type: integer
- values: {..., -3, -2, -1, 0}
- unit: –
- default: 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

- using: 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

- using: required within the scope
- 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

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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

- using: optional within the scope
- type: integer
- values: {..., -3, -2, -1, 0}
- unit: –
- default: 0

Functionality

Number of copies added in the negative direction of the y-axis.

Example

```
structure{
    region{
        array2_y{
            shift = 11.0
            max = 3
            min = 2
        }
        array_y{...}
        ...
    }
}
```

array2_z{ }

Calling sequence

```
structure{ region{ array2_z } }
```

Properties

- using: 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

- using: required within the scope
- 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

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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

- using: optional within the scope
- type: integer
- values: `{..., -3, -2, -1, 0}`
- unit: –
- default: `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

- using: 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

- using: 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

- using: 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"
        }
        ...
    }
}
contacts{
    schottky{
        name = "my_boundary_conditions"
        ...
    }
}
```

---

remove

Calling sequence

```
structure{ region{ contact{ remove{ } } } }
```

Properties

- using: 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

- using: 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 6.5.8.1 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.

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{
```

(continues on next page)

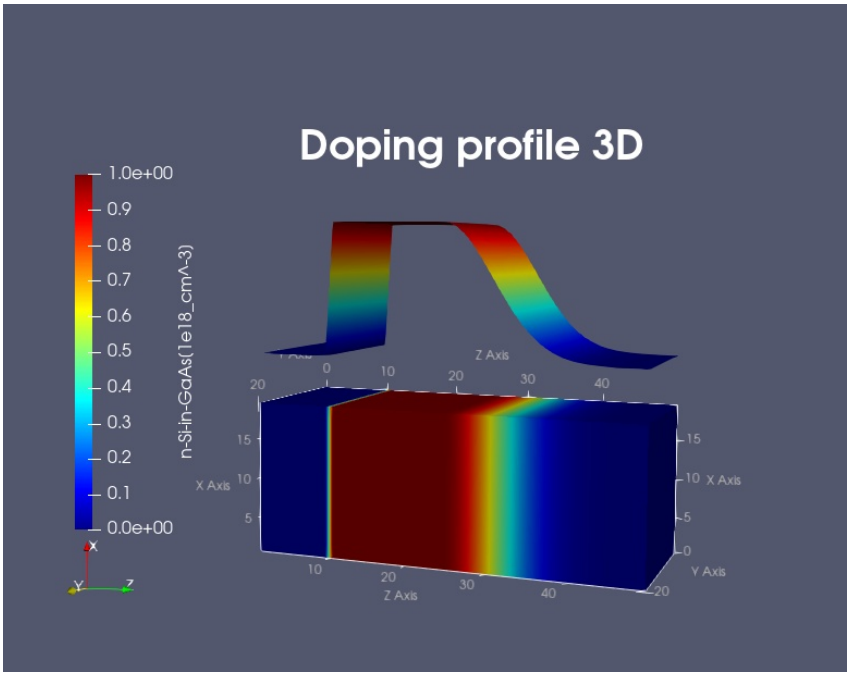


Figure 6.5.8.1: Three-dimensional doping profile (image generated by *Exporting to ParaView*).

(continued from previous page)

```
        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{}
```

(continues on next page)

(continued from previous page)

```
}
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

- using: optional within the scope
- items: no constraints

Functionality

Removes all dopants from a specific region.

Example

```
structure{
    region{
        doping{
            remove{}
        }
        ...
    }
}
```

(continues on next page)

(continued from previous page)

```
impurities{
    ...
}
```

---

constant{ }

Calling sequence

`structure{ region{ doping{ constant{ } } } }`

Properties

- using: optional within the scope
- 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

- using: required within the scope
- type: character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            constant{
                name = "n-Si"
                ...
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

constant{ conc }

Calling sequence

`structure{ region{ doping{ constant{ conc } } } }`

Properties

- using: required within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- 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

- using: required within the scope
- 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

- using: required within the scope
- type: vector of 2 real numbers
- 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 ]
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

linear{ x }

Calling sequence

```
structure{ region{ doping{ linear{ x } } } }
```

Properties

- using: conditional
- type: vector of 2 real numbers
- 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

- using: conditional
- type: vector of 2 real numbers
- 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{
```

(continues on next page)

(continued from previous page)

```
                x = [ 50.0, 100.0 ]
                y = [ 50.0, 100.0 ]
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

linear{ z }

Calling sequence

`structure{ region{ doping{ linear{ z } } } }`

Properties

- using: conditional
- type: vector of 2 real numbers
- 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 ]
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

linear{ add }

Calling sequence

`structure{ region{ doping{ linear{ add } } } }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- 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{
```

(continues on next page)

(continued from previous page)

```
                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
    }
}
global{
    simulate2D{ }
}
```

gaussian1D{ name }

Calling sequence

```
structure{ region{ doping{ gaussian1D{ name } } } }
```

Properties

- using: required within the scope
- 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

- using: conditional
- type: real number
- values: `[0.0, ...)`
- unit: cm^{-3}

Functionality

Defines maximum of dopant concentration.

Example

```
structure{
    region{
        doping{
            gaussian1D{
                conc = 1.0e18
                ...
```

(continues on next page)

(continued from previous page)

```
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ dose }

Calling sequence

`structure{ region{ doping{ gaussian1D{ dose } } } }`

Properties

- using: 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
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ x }

Calling sequence

```
structure{ region{ doping{ gaussian1D{ x } } } }
```

Properties

- using: 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

- using: 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
                ...
```

(continues on next page)

(continued from previous page)

```
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian1D{ z }

Calling sequence

`structure{ region{ doping{ gaussian1D{ z } } } }`

Properties

- using: 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

- using: 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

- using: 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

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{
```

(continues on next page)

(continued from previous page)

```
                add = no
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ }

Calling sequence

`structure{ region{ doping{ gaussian2D{ } } } }`

Properties

- using: optional within the scope
- items: no constraints

Functionality

Defines Gaussian distribution function in two directions, constant in remaining perpendicular direction.

Examples

```
structure{
    region{
        doping{
            gaussian2D{
                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

- using: required within the scope
- type: character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            gaussian2D{
                name = "n-Si"
                ...
            }
        }
        ...
    }
}
impurities{
    donor{
```

(continues on next page)

(continued from previous page)

```
        name = "n-Si"
        ...
    }
}
```

---

gaussian2D{ conc }

Calling sequence

`structure{ region{ doping{ gaussian2D{ conc } } } }`

Properties

- using: 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

- using: 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

- using: 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

- using: 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

- using: 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
```

(continues on next page)

(continued from previous page)

```
                x = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian2D{ sigma_x }

Calling sequence

`structure{ region{ doping{ gaussian2D{ sigma_x } } } }`

Properties

- using: 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

- using: 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

- using: 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{
```

(continues on next page)

(continued from previous page)

```
            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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- 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

- using: required within the scope
- type: character string

Functionality

Refers to a dopant definition in *impurities{ }*.

Example

```
structure{
    region{
        doping{
            gaussian3D{
                name = "n-Si"
                ...
            }
        }
        ...
    }
}
impurities{
    donor{
        name = "n-Si"
        ...
    }
}
```

---

gaussian3D{ conc }

Calling sequence

`structure{ region{ doping{ gaussian3D{ conc } } } }`

Properties

- using: 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

- using: 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

- using: 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
```

(continues on next page)

(continued from previous page)

```
                z = 50.0
                ...
            }
        }
        ...
    }
}
impurities{
    ...
}
```

---

gaussian3D{ y }

Calling sequence

`structure{ region{ doping{ gaussian3D{ y } } } }`

Properties

- using: 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

- using: 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

- using: 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{
```

(continues on next page)

(continued from previous page)

```
                x = 50.0
                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

- using: 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

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- 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"
        ...
    }
}
import{
    file{
        name = "importing_dopant_profile"
        filename = "precious_dopant_profile.dat"
        ...
    }
}
```

---

import{ name }

Calling sequence

```
structure{ region{ doping{ import{ name } } } }
```

Properties

- using: required within the scope
- 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

- using: required within the scope
- type: character string

Functionality

Reference to imported data in *import{ }*.

Example

```
structure{
    region{
        doping{
            import{
                import_from = "importing_dopant_profile"
                ...
```

(continues on next page)

(continued from previous page)

```
            }
        }
        ...
    }
}
impurities{
    ...
}
import{
    file{
        name = "importing_dopant_profile"
        filename = "precious_dopant_profile.dat"
        ...
    }
}
```

output_region_index{ }

Calling sequence

`structure{ output_region_index{ } }`

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}
```

output_material_index{ }

Calling sequence

```
structure{ output_material_index{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

output_user_index{ }

Calling sequence
`structure{ output_user_index{ } }`

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}
```

output_contact_index{ }

Calling sequence

```
structure{ output_contact_index{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

output_alloy_composition{ }

Calling sequence

```
structure{ output_alloy_composition{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

output_impurities{ }

Calling sequence

```
structure{ output_impurities{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

output_generation{ }

Calling sequence

```
structure{ output_generation{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs generation rate for each grid point in units of $[10^{18}/(cm^3 s)]$.

Examples

```
structure{
    output_generation{ }
}
```

Nested keywords

- *boxes*

---

boxes

Calling sequence

```
structure{ output_generation{ boxes } }
```

Properties

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

output_injection{ }

Calling sequence

```
structure{ output_injection{ } }
```

Properties

- using: optional within the scope
- 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*

---

boxes

Calling sequence

```
structure{ output_injection{ boxes } }
```

Properties

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

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/cm^3s] (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]
```

(continues on next page)

(continued from previous page)

```
    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.
                                  # 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]
```

(continues on next page)

(continued from previous page)

```
    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␣
↪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 6.5.8.2 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.

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.

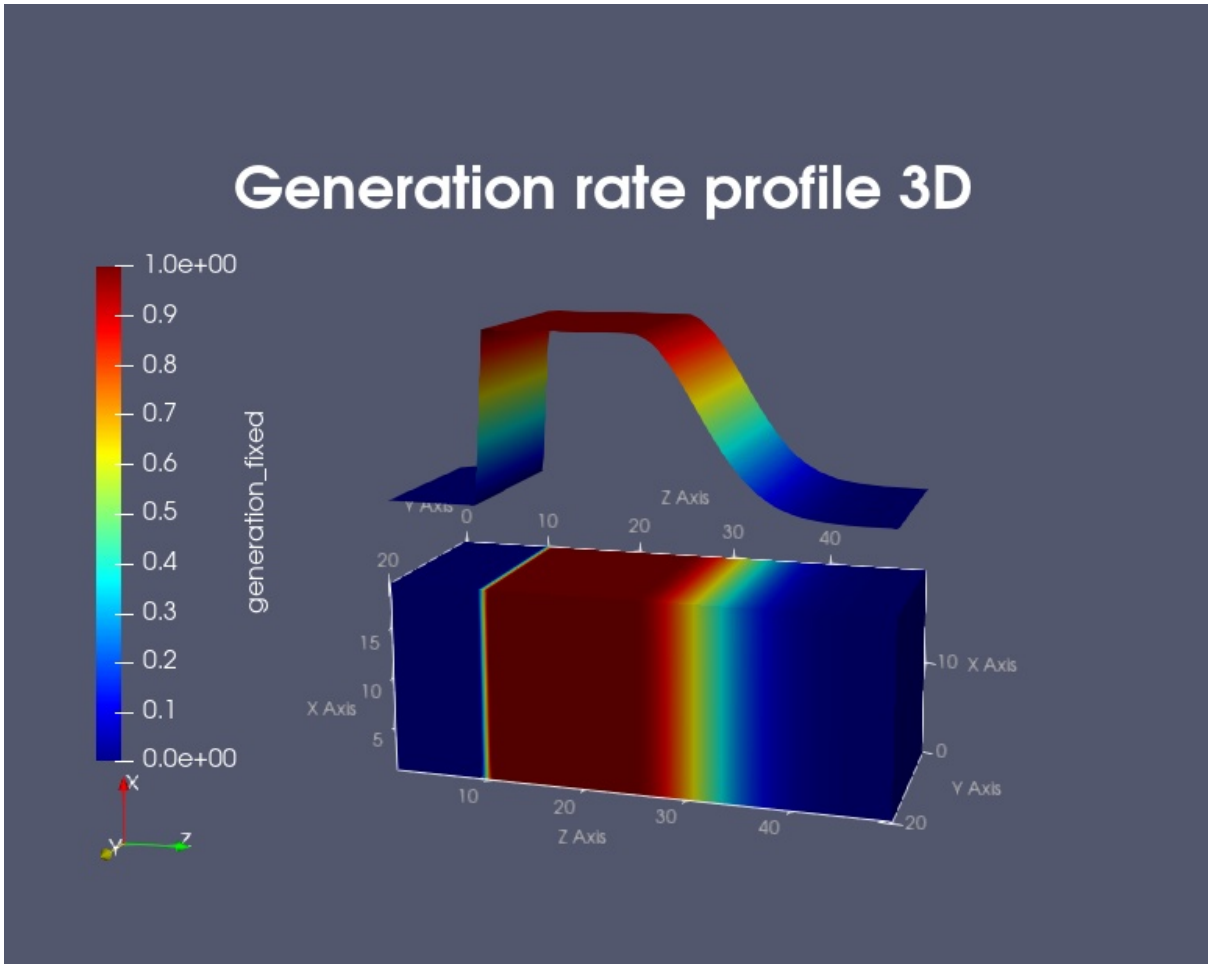


Figure 6.5.8.2: Three-dimensional generation rate profile. (Image generated by *ParaView*.)

```
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]
            }
        }
    }
}<>
```

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.

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)
}
```

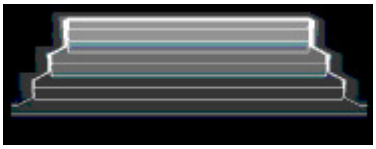
ternary_linear{}

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 =␣
```

(continues on next page)

(continued from previous page)

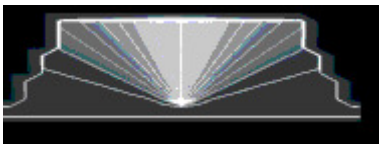
```
↪0.28 (In0.28Ga0.72As) (see figure below)
        x       = [20.0, 0]         # x coordinate of apex and x component of axis␣
↪direction [nm]
        y       = [20.0, 0]         # y coordinate of apex and y component of axis␣
↪direction [nm]
        z       = [11.0, 1]         # z coordinate of apex 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)
```

(continues on next page)

(continued from previous page)

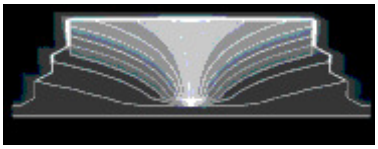
```
                                    # direction of center axis (0,0,1), i.e. along z␣
↪axis
                                    # The profile is symmetric with respect to the␣
↪inverse of the direction of the center axis,
                                    # i.e. (0,0,1) will lead to the same trumpet␣
↪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␣
↪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{}

structure{ region{} } - shape objects

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
}
```

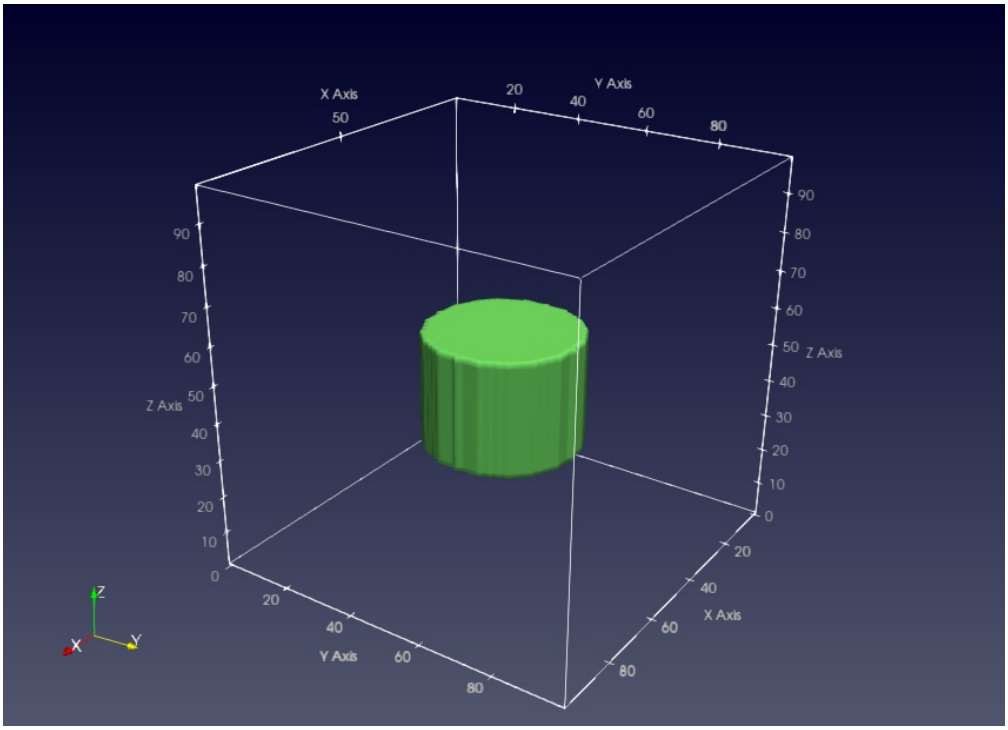
obelisk{}

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.
```

(continues on next page)



(continued from previous page)

```
        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
}
```

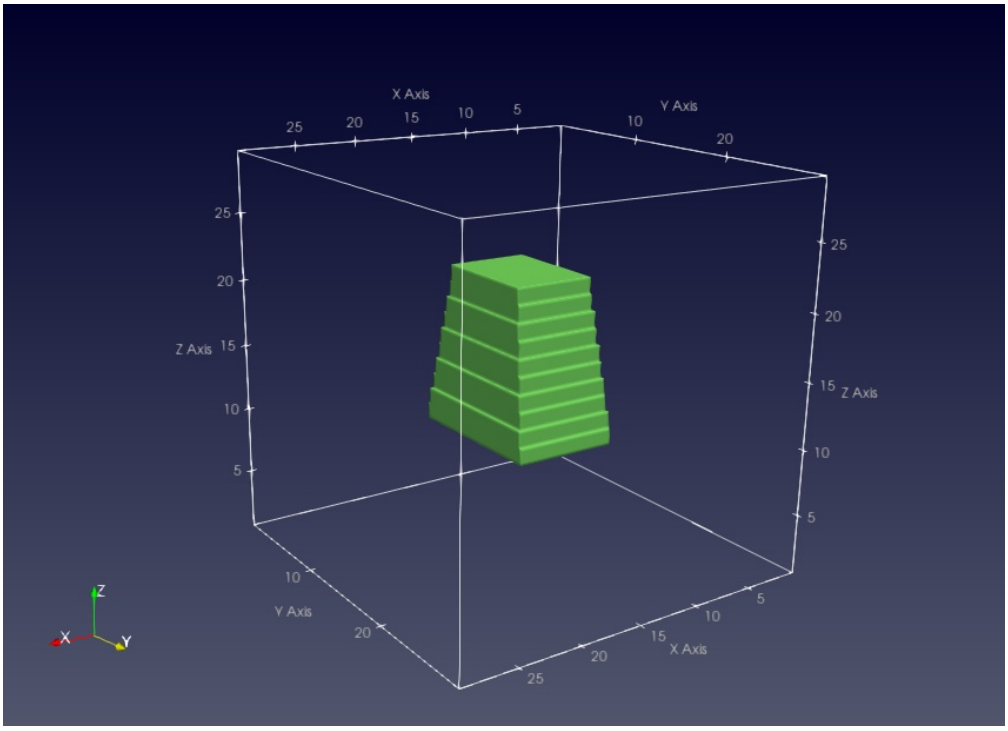
Note: Exactly one of the elements `base_x`, `base_y` and `base_z` has to be set by two equal numbers to define the base plane. The same holds for `top_x`, `top_y` and `top_z` to define the top line.

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
        ... (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.

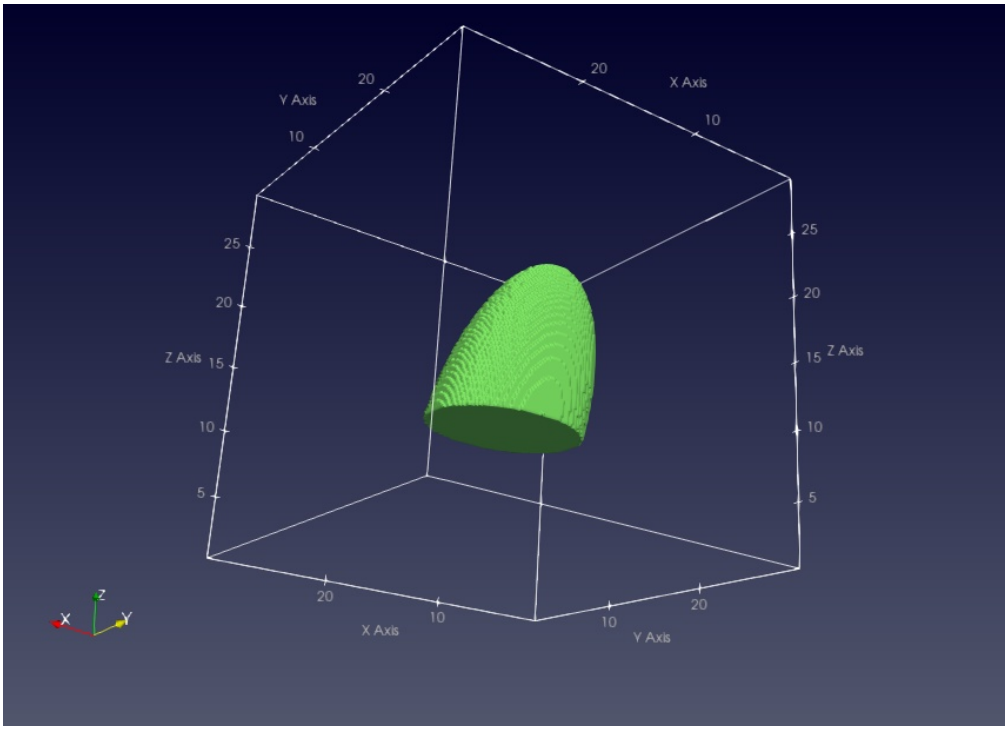
cone{}

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
```

(continues on next page)



(continued from previous page)

```
        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␣
```

(continues on next page)

(continued from previous page)

```
↪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␣
↪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{}

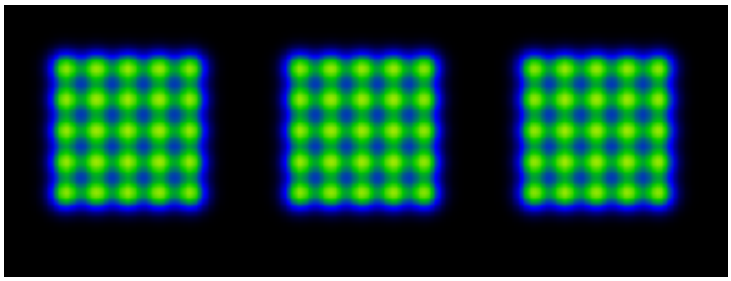
3D object, e.g. a pyramid with 4 freely defined corner points

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



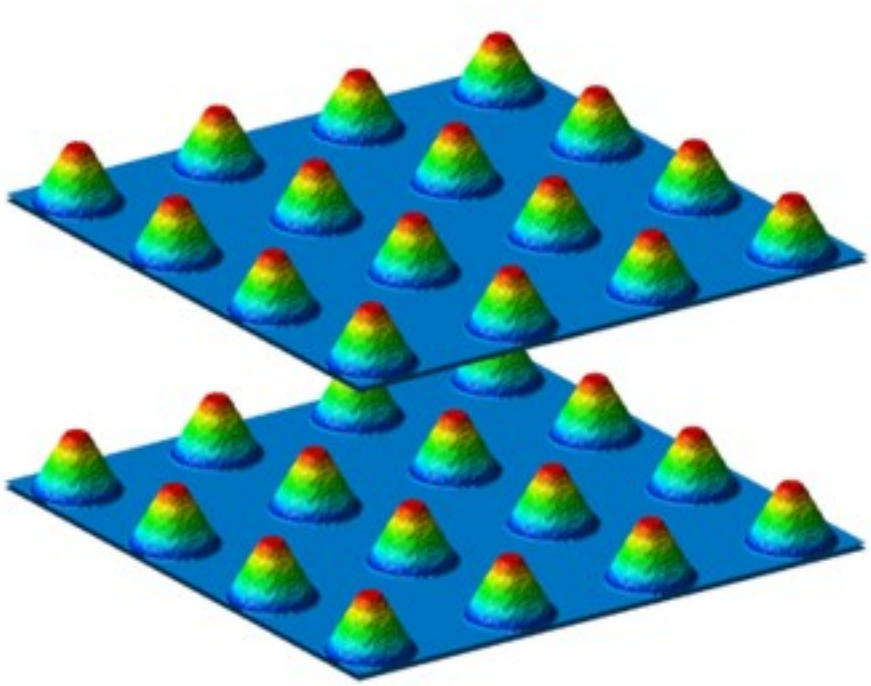
The pattern above can be produced by

```
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
            }
        }
    }
}
```

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
```

(continues on next page)



(continued from previous page)

```
        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
    }
    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.

6.5.9 grid{ }

Calling sequence

`grid{ }`

Properties

- using: required within the scope
- items: exactly 1

Functionality

Specifications of the non-uniform rectangular grid lines.

Example

```
grid{
    xgrid{}
}

global{
    simulate1D{}
}
```

```
grid{
    xgrid{}
    ygrid{}
}

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 }*

---

xgrid{ }

Calling sequence

```
grid{ xgrid{ } }
```

Properties

- using: required within the scope
- items: exactly 1

Functionality

This group is used to define simulation space grid along the x-axis.

Example

```
grid{
    xgrid{}
}
```

---

xgrid{ min_pos }

Calling sequence

```
grid{ xgrid{ min_pos } }
```

Properties

- using: optional within the scope
- 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
    }
}
```

---

xgrid{ max_pos }

Calling sequence

```
grid{ xgrid{ max_pos } }
```

Properties

- using: optional within the scope
- 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
    }
}
```

---

xgrid{ allow_spacing_jumps }

Calling sequence

```
grid{ xgrid{ allow_spacing_jumps } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}
```

---

xgrid{ line{ } }

Calling sequence

```
grid{ xgrid{ line{ } } }
```

Properties

- using: required within the scope
- 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{ }
    }
}
```

---

xgrid{ line{ pos } }

Calling sequence

```
grid{ xgrid{ line{ pos } } }
```

Properties

- using: required within the scope
- 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 }
    }
}
```

---

xgrid{ line{ spacing } }

Calling sequence

```
grid{ xgrid{ line{ spacing } } }
```

Properties

- using: required within the scope
- 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 }
    }
}
```

---

xgrid{ line{ array{ } } }

Calling sequence

```
grid{ xgrid{ line{ array{ } } } }
```

Properties

- using: 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{...}
        }
    }
}
```

---

xgrid{ line{ array{ shift } } }

Calling sequence

```
grid{ xgrid{ line{ array{ shift } } } }
```

Properties

- using: required within the scope
- 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 }
        }
    }
}
```

xgrid{ line{ array{ min } } }

Calling sequence

```
grid{ xgrid{ line{ array{ min } } } }
```

Properties

- using: optional within the scope
- type: integer
- values: {..., -3, -2, -1, 0}
- unit: –
- default: 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 }
        }
    }
}
```

xgrid{ line{ array{ max } } }

Calling sequence

```
grid{ xgrid{ line{ array{ max } } } }
```

Properties

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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 }
        }
    }
}
```

xgrid{ line{ array2{ } } }

Calling sequence

```
grid{ xgrid{ line{ array2{ } } } }
```

Properties

- using: 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{...}
        }
    }
}
```

---

xgrid{ line{ array2{ shift } } }

Calling sequence

```
grid{ xgrid{ line{ array2{ shift } } } }
```

Properties

- using: required within the scope
- 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 }
        }
    }
}
```

---

xgrid{ line{ array2{ min } } }

Calling sequence

```
grid{ xgrid{ line{ array2{ min } } } }
```

Properties

- using: optional within the scope
- type: integer
- values: {..., -3, -2, -1, 0}
- unit: –
- default: 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 }
        }
    }
}
```

---

xgrid{ line{ array2{ max } } }

Calling sequence

```
grid{ xgrid{ line{ array2{ max } } } }
```

Properties

- using: required within the scope
- type: integer
- values: {0, 1, 2, 3, ...}
- 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 }
        }
    }
}
```

---

ygrid{ }

Calling sequence

```
grid{ ygrid{ } }
```

Properties

- using: 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 }
        }
    }
}
```

---

zgrid{ }

Calling sequence

```
grid{ zgrid{ } }
```

Properties

- using: 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 }
        }
    }
}
```

energy_grid{ }

Calling sequence

```
grid{ energy_grid{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Specifying the discretization of energy.

Example

```
grid{
    energy_grid{...}
}
```

energy_grid{ min_energy }

Calling sequence

```
grid{ energy_grid{ min_energy } }
```

Properties

- using: required within the scope
- 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
    }
}
```

energy_grid{ max_energy }

Calling sequence

```
grid{ energy_grid{ max_energy } }
```

Properties

- using: required within the scope
- 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
    }
}
```

---

energy_grid{ energy_resolution }

Calling sequence

```
grid{ energy_grid{ energy_resolution } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: eV
- default: `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
    }
}
```

6.5.10 classical{ }

Calling sequence

```
classical{ }
```

Properties

- using: required within the scope
- 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

Gamma{ }

Calling sequence

```
classical{ Gamma{ } }
```

Properties

- using: 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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
}
```

HH{ }

Calling sequence

```
classical{ HH{ } }
```

Properties

- using: 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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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{}
}
```

LH{ }

Calling sequence

```
classical{ LH{ } }
```

Properties

- using: 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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
}
```

SO{ }

Calling sequence

```
classical{ SO{ } }
```

Properties

- using: 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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
        }
```

(continues on next page)

(continued from previous page)

```
    }
    Gamma{}
}
```

X{ }

Calling sequence

```
classical{ X{ } }
```

Properties

- using: 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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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{
```

(continues on next page)

(continued from previous page)

```
    ...
    crystal_zb{...}
}
```

Delta{ }

Calling sequence

```
classical{ Delta{ } }
```

Properties

- using: 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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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{
```

(continues on next page)

(continued from previous page)

```
    ...
    crystal_zb{...}
}
```

L{ }

Calling sequence

```
classical{ L{ } }
```

Properties

- using: 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*
  - *output_bandedge{ }*
  - *output_bandedge{ averaged }*

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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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{...}
}
```

carrier_statistics

Calling sequence

```
classical{ carrier_statistics }
```

Properties

- using: optional within the scope
- type: choice
- values: `maxwell_boltzmann`; `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{}
```

(continues on next page)

(continued from previous page)

```
    HH{}
}
```

energy_distribution{ }

Calling sequence

```
classical{ energy_distribution{ } }
```

Properties

- using: optional within the scope
- 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_quantum_regions*

min_energy

Calling sequence

```
classical{ energy_distribution{ min_energy } }
```

Properties

- using: required within the scope
- 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

- using: required within the scope
- 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

- using: optional within the scope
- type: real number
- values: no constraints
- unit: eV
- default: `0.1`

Functionality

energy spacing

Example

```
classical{
    energy_distribution{
        min_energy = -0.5
        max_energy = 1.8
        energy_resolution = 0.01
    }

    Gamma{}
    HH{}
}
```

---

only_quantum_regions

Calling sequence

```
classical{ energy_distribution{ only_quantum_regions } }
```

Properties

- using: conditional
- type: choice
- choices: `yes`; `no`
- default: `no`

Dependencies

- *quantum{ region{ } }* must be defined in the input file to use *only_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_quantum_regions = yes

        min_energy = -0.5
        max_energy = 1.8
    }

    Gamma{}
    HH{}
    quantum{
        region{...}
    }
}
```

energy_resolved_density{ }

Calling sequence

`classical{ energy_resolved_density{ } }`

Properties

- using: 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_quantum_regions*
- *output_energy_resolved_densities{ }*
- *output_LDOS{ }*

only_quantum_regions

Calling sequence

```
classical{ energy_resolved_density{ only_quantum_regions } }
```

Properties

- using: conditional
- type: choice
- choices: yes; no
- default: no

Dependencies

- *quantum{ region{ } }* must be defined in the input file to use *only_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_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

- using: 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

- using: 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{
```

(continues on next page)

(continued from previous page)

```
    simulate2D{...}
}
```

bulk_dispersion{ }

Calling sequence

`classical{ bulk_dispersion{ } }`

Properties

- using: 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"
            ...
        }
        path{
            name = "name_2"
            ...
        }
        full{
            name = "name_3"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
   ...
   crystal_zb{...}
}
```

Nested keywords

- *Gamma{ }*
- *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 }*
- *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

- using: 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{}
```

(continues on next page)

(continued from previous page)

```
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

HH{ }

Calling sequence

```
classical{ bulk_dispersion{ HH{ } } }
```

Properties

- using: 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

- using: 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

- using: 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{}
    }
    Gamma{}
    HH{}
}
```

KP6{ }

Calling sequence

```
classical{ bulk_dispersion{ KP6{ } } }
```

Properties

- using: 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

- using: conditional
- type: choice
- choices: yes; 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

- using: conditional
- type: choice
- choices: yes; 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

- using: 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{}
```

(continues on next page)

(continued from previous page)

```
        path{
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

KP8{ use_Luttinger_parameters }

Calling sequence

```
classical{ bulk_dispersion{ KP8{ use_Luttinger_parameters } } }
```

Properties

- using: conditional
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; 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:

- using: optional within the scope
- type: real number
- values: no constraints
- default: `0.0`
- unit: –

Properties for Wurtzite:

- using: optional within the scope
- type: vector of 2 real numbers
- values: no constraints
- default: `[0.0, 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"
            ...
        }
        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

- using: conditional
- type: choice
- choices: yes; 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{...}
}
```

KP14{ }

Calling sequence

```
classical{ bulk_dispersion{ KP14{ } } }
```

Properties

- using: 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"
            ...
```

(continues on next page)

(continued from previous page)

```
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
   ...
   crystal_zb{...}
}
```

KP14{ use_Luttinger_parameters }

Calling sequence

`classical{ bulk_dispersion{ KP14{ use_Luttinger_parameters } } }`

Properties

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; 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"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
   ...
   crystal_zb{...}
}
```

KP30{ }

Calling sequence

```
classical{ bulk_dispersion{ KP30{ } } }
```

Properties

- using: 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{}
}
```

(continues on next page)

(continued from previous page)

```
global{
    ...
    crystal_zb{...}
}
```

---

full{ }

Calling sequence

```
classical{ bulk_dispersion{ full{ } } }
```

Properties

- using: 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

- using: required within the scope
- 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

- using: required within the scope
- 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

- using: required within the scope
- type: real number
- values: no constraints
- default: `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

- using: conditional
- type: real number
- values: no constraints
- default: `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

- using: conditional
- type: real number
- values: no constraints
- default: `0.0`
- unit: nm

using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
            name = "name"
            position{...}
            ...
        }
        output_bulk_dispersions{}
    }
```

(continues on next page)

(continued from previous page)

```
    Gamma{}
    HH{}
}
```

---

full{ kxgrid{ } }

Calling sequence

```
classical{ bulk_dispersion{ full{ kxgrid{ } } } }
```

Properties

- using: required within the scope
- 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

- using: required within the scope
- items: minimum 2

Functionality

Setting options defining the grid in k-space.

Example

```
classical{
    bulk_dispersion{
        KP8{}
```

(continues on next page)

(continued from previous page)

```
        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

- using: required within the scope
- 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 }
                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

- using: required within the scope
- 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

- using: required within the scope
- 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{}
```

(continues on next page)

(continued from previous page)

```
        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

- using: required within the scope
- 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{
            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{}
}
```

(continues on next page)

(continued from previous page)

```
global{
    ...
    simulate3D{}
}
```

---

path{ }

Calling sequence

`classical{ bulk_dispersion{ path{ } } }`

Properties

- using: 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
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ name }

Calling sequence

```
classical{ bulk_dispersion{ path{ name } } }
```

Properties

- using: required within the scope
- 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

- using: required within the scope
- 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

- using: required within the scope
- type: real number
- values: no constraints
- default: `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{
   ...
   simulate1D{}
}
```

path{ position{ y } }

Calling sequence

```
classical{ bulk_dispersion{ path{ position{ y } } } }
```

Properties

- using: conditional
- type: real number
- values: no constraints
- default: `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

- using: conditional
- type: real number
- values: no constraints
- default: `0.0`
- unit: nm

using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
```

(continues on next page)

(continued from previous page)

```
    }
    Gamma{}
    HH{}
}
```

---

path{ point{ } }

Calling sequence

```
classical{ bulk_dispersion{ path{ point{ } } } }
```

Properties

- using: required within the scope
- 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{...}
            point{...}
            point{...}
            point{...}
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

path{ point{ k } }

Calling sequence

```
classical{ bulk_dispersion{ path{ point{ k } } } }
```

Properties

- using: required within the scope
- type: vector of 3 real numbers
- values: no constraints
- default: `[0.0, 0.0, 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

- using: 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{}
```

(continues on next page)

(continued from previous page)

```
    }
    Gamma{}
    HH{}
}
```

---

path{ num_points }

Calling sequence

```
classical{ bulk_dispersion{ path{ num_points } } }
```

Properties

- using: conditional
- type: integer
- values: {2, 3, 4, 5, ...}
- unit: nm^{-1}

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

- using: 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"
            ...
        }
        lines{
            name = "name_2"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ name }

Calling sequence

```
classical{ bulk_dispersion{ lines{ name } } }
```

Properties

- using: required within the scope
- type: character string

Dependencies

Nname of the dispersions which also defines the names of the output files.

Example

```
classical{
    bulk_dispersion{
        KP8{}

        lines{
```

(continues on next page)

(continued from previous page)

```
            name = "name"
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}
```

---

lines{ position{ } }

Calling sequence

```
classical{ bulk_dispersion{ lines{ position{ } } } }
```

Properties

- using: required within the scope
- 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

- using: required within the scope
- type: real number
- values: no constraints
- default: `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{}
    HH{}
}

global{
   ...
   simulate1D{}
}
```

---

lines{ position{ y } }

Calling sequence

`classical{ bulk_dispersion{ lines{ position{ y } } } }`

Properties

- using: conditional
- type: real number
- values: no constraints
- default: `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{
```

(continues on next page)

(continued from previous page)

```
            name = "name"
            position{
                x = 10.5
                y = 35.0
            }
            ...
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
   ...
   simulate2D{}
}
```

lines{ position{ z } }

Calling sequence

```
classical{ bulk_dispersion{ lines{ position{ z } } } }
```

Properties

- using: conditional
- type: real number
- values: no constraints
- default: `0.0`
- unit: nm

using: 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
            }
            ...
```

(continues on next page)

(continued from previous page)

```
        }
        output_bulk_dispersions{}
    }
    Gamma{}
    HH{}
}

global{
   ...
   simulate3D{}
}
```

---

lines{ shift_holes_to_zero }

Calling sequence

`classical{ bulk_dispersion{ lines{ shift_holes_to_zero } } }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: required within the scope
- 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

- using: required within the scope
- 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"
```

(continues on next page)

(continued from previous page)

```
            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

- using: 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

- using: 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{}
        path{...}
    }
    Gamma{}
    HH{}
}
```

---

output_inverse_masses{ }

Calling sequence

```
classical{ bulk_dispersion{ output_inverse_masses{ } } }
```

Properties

- using: 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

- using: optional within the scope
- items: maximum 1

Functionality

Outputs k-vectors for which the dispersions are computed.

Example

```
classical{
    bulk_dispersion{
        output_k_vectors{}

        KP8{}
        path{...}
    }
    Gamma{}
    HH{}
}
```

output_bandgap{ }

Calling sequence

```
classical{ output_bandgap{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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{}
}
```

output_bandedges{ }

Calling sequence

```
classical{ output_bandedges{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
}
```

output_carrier_densities{ }

Calling sequence

```
classical{ output_carrier_densities{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Output electron and hole densities to a file *total_charges.txt* 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{}
}
```

output_band_densities{ }

Calling sequence

```
classical{ output_band_densities{ } }
```

Properties

- using: optional within the scope
- 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{}
}
```

output_ionized_dopant_densities{ }

Calling sequence

```
classical{ output_ionized_dopant_densities{ } }
```

Properties

- using: optional within the scope
- 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{}
}
```

output_intrinsic_density{ }

Calling sequence

```
classical{ output_intrinsic_density{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: yes; 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{}
}
```

6.5.11 strain{ }

Calling sequence

```
strain{ }
```

Properties

- using: optional within the scope
- 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

debuglevel

Calling sequence

```
strain{ debuglevel }
```

Properties

- using: optional within the scope
- type: integer
- values: {-1, 0, 1, 2, ...}
- unit: –
- default: 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{}
}
```

no_strain{ }

Calling sequence

```
strain{ no_strain{ } }
```

Properties

- using: 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{}
}
```

pseudomorphic_strain{ }

Calling sequence

```
strain{ pseudomorphic_strain{ } }
```

Properties

- using: 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{}
}
```

minimized_strain{ }

Calling sequence

```
strain{ minimized_strain{ } }
```

Properties

- using: 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{}
}
```

growth_direction

Calling sequence

`strain{ growth_direction }`

Properties

- using: conditional
- type: vector of 3 integers
- values: no constraints
- default: `[1.0, 0.0, 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{}
    ...
}
```

residual_strain

Calling sequence

`strain{ residual_strain }`

Properties

- using: conditional
- type: real number
- values: `[-1.0, 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{}
}
```

linear_solver{ }

Calling sequence

```
strain{ linear_solver{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, 4, ...}`
- default: `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

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e-8`
- unit: GP (1D) / GP nm (2D) / GP nm^2 (3D)

Functionality

—

Example

```
strain{
    minimized_strain{}
    linear_solver{
        abs_accuracy = 1e-9
    }
}
```

---

rel_accuracy

Calling sequence

```
strain{ linear_solver{ rel_accuracy } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, 1e-2]`
- default: `1e-12`
- unit: –

Functionality

—

Example

```
strain{
    minimized_strain{}
    linear_solver{
        rel_accuracy = 1e-10
    }
}
```

---

use_cscg

Calling sequence

```
strain{ linear_solver{ use_cscg } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    minimized_strain{}
    linear_solver{
        force_diagonal_preconditioner = yes
    }
}
```

import_strain{ }

Calling sequence

```
strain{ import_strain{ } }
```

Properties

- using: 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

- using: required within the scope
- 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: $\varepsilon_{xx}\ \varepsilon_{yy}\ \varepsilon_{zz}\ \varepsilon_{xy}\ \varepsilon_{xz}\ \varepsilon_{yz}$

Example

```
strain{
    import_strain{
        import_from = "strain_import"
    }
}

import{
    file{
        name = "strain_import"
        ...
    }
}
```

---

coordinate_system

Calling sequence

```
strain{ import_strain{ coordinate_system } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `crystal`; `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"
        ...
    }
}
```

piezo_density

Calling sequence

`strain{ piezo_density }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
}
```

second_order_piezo

Calling sequence

`strain{ second_order_piezo }`

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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{}
}
```

pyro_density

Calling sequence

```
strain{ pyro_density }
```

Properties

- using: conditional
- type: choice
- choices: `yes`; `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{...}
    ...
}
```

output_hydrostatic_strain{ }

Calling sequence

```
strain{ output_hydrostatic_strain{ } }
```

Properties

- using: 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]

Note: The hydrostatic strain output is in percent (This is different compared to *nextnano*[3].)

Example

```
strain{
    output_hydrostatic_strain{}
    pseudomorphic_strain{}
}
```

Nested keywords

- *boxes*

boxes

Calling sequence

```
strain{ output_hydrostatic_strain{ boxes } }
```

Properties

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_hydrostatic_strain{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_strain_tensor{ }

Calling sequence

```
strain{ output_strain_tensor{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: yes; 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{}
}
```

output_distortion_tensor{ }

Calling sequence

```
strain{ output_distortion_tensor{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_distortion_tensor{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_stress_tensor{ }

Calling sequence

```
strain{ output_stress_tensor{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_stress_tensor{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_displacement{ }

Calling sequence

```
strain{ output_displacement{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

Functionality

output displacement vector in simulation coordinate system

Example

```
strain{
    output_displacement{
        simulation_system = no
    }
    minimized_strain{}
}
```

output_force_density{ }

Calling sequence

```
strain{ output_force_density{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

Functionality

output force density vector field in simulation coordinate system

Example

```
strain{
    output_force_density{
        simulation_system = no
    }
    pseudomorphic_strain{}
}
```

output_elastic_energy_density{ }

Calling sequence

```
strain{ output_elastic_energy_density{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_elastic_energy_density{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_polarization_charge{ }

Calling sequence

```
strain{ output_polarization_charge{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs sum of piezoelectric and pyroelectric polarization charge densities (ρ_{pz} and ρ_{py}) expressed in 10^{18}/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{}
}
```

output_polarization_charge_components{ }

Calling sequence

```
strain{ output_polarization_charge_components{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs piezoelectric and pyroelectric charge densities (ρ_{pz} and ρ_{py}) expressed in 10^{18}/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{}
}
```

output_polarization_vector{ }

Calling sequence

```
strain{ output_polarization_vector{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
    pseudomorphic_strain{}
}
```

---

simulation_system

Calling sequence

```
strain{ output_polarization_vector{ simulation_system } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Functionality

—

Example

```
strain{
    output_polarization_vector{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_polarization_vector_components{ }

Calling sequence

```
strain{ output_polarization_vector_components{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_polarization_vector_components{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_lattice_constants{ }

Calling sequence

```
strain{ output_lattice_constants{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_lattice_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_elastic_constants{ }

Calling sequence
`strain{ output_elastic_constants{ } }`

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality
—

Example

```
strain{
    output_elastic_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_piezo_constants{ }

Calling sequence

```
strain{ output_piezo_constants{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_piezo_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_second_order_piezo_constants{ }

Calling sequence

```
strain{ output_second_order_piezo_constants{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_second_order_piezo_constants{
        boxes = yes
    }
    pseudomorphic_strain{}
}
```

output_pyro_constants{ }

Calling sequence

```
strain{ output_pyro_constants{ } }
```

Properties

- using: 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

- using: optional within the scope
- type: choice
- choices: yes; no
- default: no

Functionality

—

Example

```
strain{
    output_pyro_constants{
        boxes = yes
    }
```

(continues on next page)

(continued from previous page)

```
    pseudomorphic_strain{}
}
```

6.5.12 poisson{ }

Calling sequence

```
poisson{ }
```

Properties

- using: optional within the scope
- items: maximum 1

Dependencies

- Exactly one of *import_potential{ }*, *electric_field{ }*, *between_fermi_levels{ }*, and *charge_neutral{ }* 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{ }*
- *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 }*

---

debuglevel

Calling sequence

```
poisson{ debuglevel }
```

Properties

- using: optional within the scope
- type: integer
- values: `{-1, 0, 1, 2, ...}`
- default: 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{}
}
```

---

import_potential{ }

Calling sequence

```
poisson{ import_potential{ } }
```

Properties

- using: optional within the scope
- 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{...}
```

---

import_potential{ import_from }

Calling sequence

```
poisson{ import_potential{ import_from } }
```

Properties

- using: required within the scope
- 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"
    }
}
```

(continues on next page)

(continued from previous page)

```
import{
    file{
        name = "qpc_landscape"
        ...
    }
}
```

---

import_potential{ component_number }

Calling sequence

```
poisson{ import_potential{ component_number } }
```

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, 4, ...}
- default: 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"
        ...
    }
}
```

---

electric_field{ }

Calling sequence

```
poisson{ electric_field{ } }
```

Properties

- using: optional within the scope
- 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 = ...
        ...
    }
}

global{
    simulate2D{ }
}
```

---

electric_field{ direction }

Calling sequence

`poisson{ electric_field{ direction } }`

Properties

- using: optional within the scope
- type: vector of 3 real numbers
- values: no constraints
- default: `[1.0, 0.0, 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 ]
        ...
    }
```

(continues on next page)

(continued from previous page)

```
}

global{
    simulate3D{ }
}
```

---

electric_field{ strength }

Calling sequence

```
poisson{ electric_field{ strength } }
```

Properties

- using: required within the scope
- 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
    }
}
```

---

electric_field{ reference_potential }

Calling sequence

```
poisson{ electric_field{ reference_potential } }
```

Properties

- using: optional within the scope
- 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
    }
}
```

---

between_fermi_levels{ }

Calling sequence

```
poisson{ between_fermi_levels{ } }
```

Properties

- using: optional within the scope
- 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{}
}
```

---

charge_neutral{ }

Calling sequence

```
poisson{ between_fermi_levels{ } }
```

Properties

- using: optional within the scope
- 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{}
}
```

---

newton_solver{ }

Calling sequence

```
poisson{ newton_solver{ } }
```

Properties

- using: optional within the scope
- 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:

---

newton_solver{ iterations }

Calling sequence

```
poisson{ newton_solver{ iterations } }
```

Properties

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, 4, ...}`
- default: `30`
- unit: –

Functionality

Number of iterations for Newton solver

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{}
}
```

---

newton_solver{ search_steps }

Calling sequence

```
poisson{ newton_solver{ search_steps } }
```

Properties

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, ..., 50}`
- default: `30`

- unit: –

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{
        search_steps = 40
    }
}
```

---

newton_solver{ residual }

Calling sequence

`poisson{ newton_solver{ residual } }`

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e3` for 1D; `1e1` for 2D; `1e-4` for 3D
- unit: cm^{-2} (1D) / cm^{-1} (2D) / none (3D)

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{
        residual = 1e2
    }
}
```

---

newton_solver{ gradient_shift }

Calling sequence

`poisson{ newton_solver{ gradient_shift } }`

Properties

- using: optional within the scope
- type: real number
- values: `[-1e-6, 1e-6]`
- default: `1e-11`
- unit: –

Functionality

Slightly nudges the gradient in case it is effectively zero

Example

```
poisson{
    between_fermi_levels{}
    newton_solver{
        residual = -1e-8
    }
}
```

---

linear_solver{ }

Calling sequence

```
poisson{ linear_solver{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Parameters for linear equation solver in Newton algorithm.

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{}
}
```

---

linear_solver{ iterations }

Calling sequence

```
poisson{ linear_solver{ iterations } }
```

Properties

- using: optional within the scope
- type: integer
- values: {1, 2, 3, 4, ...}
- default: 1000
- unit: –

Functionality

number of iterations for linear equation solver

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        iterations = 2000
    }
}
```

---

linear_solver{ abs_accuracy }

Calling sequence

```
poisson{ linear_solver{ abs_accuracy } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e1` for 1D; `1e-3` for 2D; `1e-8` for 3D
- unit: cm^{-2} (1D) / cm^{-1} (2D) / none (3D)

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        abs_accuracy = 1e-2
    }
}
```

---

linear_solver{ rel_accuracy }

Calling sequence

```
poisson{ linear_solver{ rel_accuracy } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, 1e-2]`
- default: `1e-13`
- unit: –

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    linear_solver{
        rel_accuracy = 1e-15
    }
}
```

---

linear_solver{ dkr_value }

Calling sequence

```
poisson{ linear_solver{ dkr_value } }
```

Properties

- using: optional within the scope
- type: real number
- values: `(..., 0.5]`
- default: `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
    }
}
```

---

linear_solver{ use_cscg }

Calling sequence

```
poisson{ linear_solver{ use_cscg } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}
```

---

linear_solver{ force_diagonal_preconditioner }

Calling sequence

```
poisson{ linear_solver{ force_diagonal_preconditioner } }
```

Properties

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

---

linear_solver{ force_iteration }

Calling sequence

```
poisson{ linear_solver{ force_iteration } }
```

Properties

- using: optional within the scope
- type: choice
- choices: yes; 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
    }
}
```

---

bisection{ }

Calling sequence

```
poisson{ bisection{ } }
```

Properties

- using: optional within the scope
- 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{}
}
```

---

bisection{ delta }

Calling sequence

```
poisson{ bisection{ delta } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `10.0`
- unit: eV

Functionality

Range of bisection search.

Example

```
poisson{
    between_fermi_levels{}
    bisection{
        delta = 7.0
    }
}
```

---

bisection{ residual }

Calling sequence

```
poisson{ bisection{ residual } }
```

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1e3`
- unit: cm^{-3}

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    bisection{
        residual = 1e1
    }
}
```

---

bisection{ iterations }

Calling sequence

```
poisson{ bisection{ iterations } }
```

Properties

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, ..., 100}`
- default: `40`
- unit: –

Functionality

—

Example

```
poisson{
    between_fermi_levels{}
    bisection{
        iterations = 60
    }
}
```

---

bisection{ robust }

Calling sequence

```
poisson{ bisection{ robust } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}
```

---

output_potential{ }

Calling sequence

```
poisson{ output_potential{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Prints out the electrostatic potential in `[eV]`.

Example

```
poisson{
    between_fermi_levels{}
    output_potential{}
}
```

---

output_electric_field{ }

Calling sequence

```
poisson{ output_electric_field{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Prints out the electric field in kv/cm.

Example

```
poisson{
    between_fermi_levels{}
    output_electric_field{}
}
```

---

output_electric_displacement{ }

Calling sequence

```
poisson{ output_electric_displacement{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Prints out the output electric displacement

Example

```
poisson{
    between_fermi_levels{}
    output_electric_displacement{}
}
```

---

output_electric_polarization{ }

Calling sequence

```
poisson{ output_electric_polarization{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Prints out the output electric polarization

Example

```
poisson{
    between_fermi_levels{}
    output_electric_polarization{}
}
```

---

output_dielectric_tensor{ }

Calling sequence

```
poisson{ output_dielectric_tensor{ } }
```

Properties

- using: optional within the scope
- 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{}
}
```

---

output_dielectric_tensor{ boxes }

Calling sequence

```
poisson{ output_dielectric_tensor{ boxes } }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}
```

6.5.13 currents{ }

Calling sequence

```
currents{ }
```

Properties

- using: optional within the scope
- items: maximum 1

Dependencies

- *insulator_bandgap* is not allowed if any of *import_electron_fermi_level{ }* or *import_hole_fermi_level{ }* is already defined.
- *minimum_density* and *maximum_density* are required if *minimum_density_factor* and *maximum_density_factor* are already defined, respectively.

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 = 1
    minimum_density_factor = [ 1e-5, 1e-7 ]
}
```

Nested keywords

debuglevel

Calling sequence

```
currents{ debuglevel }
```

Properties

- using: optional within the scope
- type: integer
- unit: –
- values: {-1, 0, 1, 2, ...}
- default: 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
}
```

import_electron_fermi_level{ }

Calling sequence

```
currents{ import_electron_fermi_level{ } }
```

Properties

- using: optional within the scope
- 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

- using: required within the scope
- 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"
```

(continues on next page)

(continued from previous page)

```
        ...
    }
}
```

component_number

Calling sequence

```
currents{ import_electron_fermi_level{ component_number } }
```

Properties

- using: optional within the scope
- type: integer
- unit: –
- values: {1, 2, 3, 4, ...}
- default: 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"
        ...
    }
}
```

import_hole_fermi_level{ }

Calling sequence

```
currents{ import_hole_fermi_level{ } }
```

Properties

- using: optional within the scope
- 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

- using: required within the scope
- 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_hole_fermi_level{ component_number } }
```

Properties

- using: optional within the scope
- type: integer
- unit: –
- values: {1, 2, 3, 4, ...}
- default: 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"
        ...
    }
}
```

insulator_bandgap

Calling sequence

```
currents{ insulator_bandgap }
```

Properties

- using: optional within the scope
- type: real number
- unit: eV
- values: `[1e-6, ...)`
- default: `1.0`

Functionality

This keyword, I_{gap}, initializes the quasi-Fermi levels following the formula:

$$\operatorname{div} \exp\left(E_{\text{gap}}/I_{\text{gap}}\right) \nabla E_{\text{F}} = 0,$$

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
}
```

electron_mobility{ }

Calling sequence

```
currents{ electron_mobility{ } }
```

Properties

- using: optional within the scope
- 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

- using: required within the scope
- type: choice
- choices: `constant`; `masetti`; `arora`; `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

- using: optional within the scope
- type: choice
- choices: `none`; `haensch`; `canali`; `transferred`; `eastman`; `eastman4`
- default: `none`

Functionality

Selects high-field mobility model for electrons.

choice	model
`none`	High-field models are not used
`haensch`	*Hänsch model*
`canali`	*Extended Canali model*
`transferred`	*Hänsch model*
`eastman`	*Eastman-Tiwari-Shur model with standard parametrization*
`eastman4`	*Eastman-Tiwari-Shur model with observable parametrization*

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

hole_mobility{ }

Calling sequence

```
currents{ hole_mobility{ } }
```

Properties

- using: optional within the scope
- 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

- using: required within the scope
- type: choice
- choices: `constant`; `masetti`; `arora`; `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

- using: optional within the scope
- type: choice
- choices: `none`; `haensch`; `canali`; `transferred`; `eastman`; `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

recombination_model{ }

Calling sequence

```
currents{ recombination_model{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
```

(continues on next page)

(continued from previous page)

```
        enable_generation = yes
    }
}
```

linear_solver{ }

Calling sequence

```
currents{ linear_solver{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: integer
- unit: –
- values: `{1, 2, 3, 4, ...}`
- default: `10000`

Functionality

Maximum number of iterations

Example

```
currents{
    recombination_model{}
    linear_solver{
        iterations = 50000
    }
}
```

---

abs_accuracy

Calling sequence

```
currents{ linear_solver{ abs_accuracy } }
```

Properties

- using: optional within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`
- default: `1e-30`

Functionality

—

Example

```
currents{
    recombination_model{}
    linear_solver{
        abs_accuracy = 1e-32
    }
}
```

---

rel_accuracy

Calling sequence

```
currents{ linear_solver{ rel_accuracy } }
```

Properties

- using: optional within the scope
- type: real number
- unit: –
- values: `[0.0, 1e-2]`
- default: `1e-13`

Functionality

—

Example

```
currents{
    recombination_model{}
    linear_solver{
        rel_accuracy = 1e-15
    }
}
```

---

dkr_value

Calling sequence

```
currents{ linear_solver{ dkr_value } }
```

Properties

- using: optional within the scope
- type: real number
- unit: –
- values: `[0.0, 0.5]`
- default: `-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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: yes; 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: integer
- unit: –
- values: `{0, 1}`
- default: `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
    }
}
```

minimum_density

Calling sequence

```
currents{ minimum_density }
```

Properties

- using: optional within the scope
- type: real number
- unit: cm^{-3}
- values: `[0.0, 1e20]`
- default: `1e10`

Functionality

A keyword allowing to improve the condition number of the matrix representing the current equation.

Minimum carrier density, ρ_{min}, is defined for both types of carriers at once (electrons and 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, $\rho_{\mathrm{sim}}\left(x\right)$, 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, every carrier 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 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 is never smaller than $10^{-10}\ \mathrm{cm}^{-3}$ in the algorithm. This value corresponds to a conductivity 10 orders of magnitude lower than of the best insulators. The syntax allows selecting smaller values, including zero, for convenience.

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. Recombination processes can be affected by setting *minimal_recombination* to `yes`.

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.

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 = 1.0  # cm^-3
    minimum_density_factor = [ 1e10 , 1e8 ]
}
```

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 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 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.

minimum_density_factor

Calling sequence

```
currents{ minimum_density_factor }
```

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: `[0.0, ...)` for every dimension
- default: `[1.0, 1.0]`
- unit: $-$

Functionality

The two numbers of `minimum_density_factor` are scaling factors by which the minimum density ρ_{min} (see *minimum_density*) is multiplied for electron and for hole density distributions. The first number is scaling the ρ_{min} for electrons, while the second number is scaling the ρ_{min} for holes.

Examples

A minimum density of 10^{10} cm^{-3} for electrons and 10^8 cm^{-3} for holes can be defined as

```
currents{
    recombination_model{}
    minimum_density = 1e10 # cm^-3
    minimum_density_factor = [ 1.0 , 0.01 ]
}
```

or

```
currents{
    recombination_model{}
    minimum_density = 1e9 # cm^-3
    minimum_density_factor = [ 10 , 0.1 ]
}
```

or explicitly as

```
currents{
    recombination_model{}
    minimum_density = 1.0  # cm^-3
    minimum_density_factor = [ 1e10 , 1e8 ]
}
```

Irrespective to the method of definition, the values used for the minimum densities are the same and output close to the beginning of the log file.

maximum_density

Calling sequence

`currents{ maximum_density }`

Properties

- using: optional within the scope
- type: real number
- values: `[0.0, 1e30]`
- default: `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 both types of carriers at once (electrons and 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 a given carrier type, $\rho_{\mathrm{sim}}(x)$, 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, $\rho_{\mathrm{current}}(x)$, is given by

$$\rho_{\mathrm{current}}(x) = \min\left[\rho_{\mathrm{sim}}(x), \rho_{\max}\right].$$

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 ($\nabla\ \mu\ \rho_{\mathrm{current}}\ \nabla$) and the corresponding current for each type of carriers. Thus it has no direct influence on computed densities, Poisson equation, etc.

Hint:

- The $\rho_{\max}$ 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 = 1.0  # cm^-3
    maximum_density_factor = [ 1e10 , 1e8 ]
}
```

maximum_density_factor

Calling sequence

`currents{ maximum_density_factor }`

Properties

- using: optional within the scope
- type: vector of 2 real numbers
- values: `[0.0, ...)` for every dimension
- default: `[1.0, 1.0]`
- unit: –

Functionality

The two numbers of `maximum_density_factor` are scaling factors by which the maximum density $\rho_{\max}$ (see *maximum_density*) is multiplied for electron and for hole density distributions. The first number is scaling the $\rho_{\max}$ for electrons, while the second number is scaling the $\rho_{\max}$ for holes.

Examples

A maximum density of 10^{10} cm^{-3} for electrons and 10^{8} cm^{-3} for holes can be defined as

```
currents{
    recombination_model{}
    maximum_density = 1e10 # cm^-3
    maximum_density_factor = [ 1.0 , 0.01 ]
}
```

or

```
currents{
    recombination_model{}
    maximum_density = 1e9 # cm^-3
    maximum_density_factor = [ 10 , 0.1 ]
}
```

or explicitly as

```
currents{
    recombination_model{}
    maximum_density = 1.0  # cm^-3
    maximum_density_factor = [ 1e10 , 1e8 ]
}
```

Irrespective to the method of definition, the values used for the maximum densities are the same and output close to the beginning of the log file.

minimal_recombination

Calling sequence

```
currents{ minimal_recombination }
```

Properties

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Functionality

If set to `yes`, then the minimum densities applies to the recombination and generation terms of the current equation.

Example

```
currents{
    recombination_model{}
    minimal_recombination = yes
}
```

electron_contact

Calling sequence

```
currents{ electron_contact }
```

Properties

- using: optional within the scope
- 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"
        ...
    }
}
```

hole_contact

Calling sequence

```
currents{ hole_contact }
```

Properties

- using: optional within the scope
- 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"
        ...
    }
}
```

output_fermi_levels{ }

Calling sequence

```
currents{ output_fermi_levels{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs quasi-Fermi levels for electrons and holes in eV.

Example

```
currents{
    recombination_model{}
    output_fermi_levels{}
}
```

output_fermi_level_difference{ }

Calling sequence

```
currents{ output_fermi_level_difference{ } }
```

Properties

- using: optional within the scope
- 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{}
}
```

output_velocities{ }

Calling sequence

```
currents{ output_velocities{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs electron and hole drift velocities in $\mathrm{cm/s}$.

Example

```
currents{
    recombination_model{}
    output_velocities{}
}
```

output_forces{ }

Calling sequence

```
currents{ output_forces{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs driving forces of electrons and holes $\mathrm{eV/nm}$

Example

```
currents{
    recombination_model{}
    output_forces{}
}
```

output_currents{ }

Calling sequence

```
currents{ output_currents{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs the electron and hole current densities expressed in 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{}
}
```

output_power_density{ }

Calling sequence

```
currents{ output_power_density{ } }
```

Properties

- using: optional within the scope
- items: maximum 1

Functionality

Outputs power density of Joule heating expressed in W/cm^3.

Example

```
currents{
    recombination_model{}
    output_power_density{}
}
```

output_mobilities{ }

Calling sequence

```
currents{ output_mobilities{ } }
```

Properties

- using: optional within the scope
- 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

- using: optional within the scope
- type: choice
- choices: `yes`; `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
    }
}
```

output_recombination{ }

Calling sequence
`currents{ output_recombination{ } }`

Properties

- using: optional within the scope
- 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{}
}
```

output_injection{ }

Calling sequence
`currents{ output_injection{ } }`

Properties

- using: optional within the scope
- 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{}
}
```

6.5.14 quantum{ }

A group of keywords specifying quantum models, i.e. how the Schrödinger equation should be solved.

Groups located directly under `quantum{ }` are:

quantum{ region{ } }

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.

Top-level attributes in quantum{ region{} }

name

value
"string"

Provides the name of the quantum region.

no_density

value
`yes` or `no`

default
`no`

Tells if to not calculate quantum mechanical charge density.

x

value
2D float vector

Provides the extension of quantum region in x direction in nanometers (nm)

y

value
2D float vector

Provides the extension of quantum region in y direction in nanometers (nm). To be used for 2D or 3D calculations only.

z

value
2D float vector

Provides the extension of quantum region in x direction in nanometers (nm). To be used for 3D calculation only.

One-band models in quantum{ region{} }

quantum{ region{ Gamma{} } }

Solves single-band effective mass Schrödinger equation for the Gamma conduction band.

num_ev

value
integer >= 0

default
0

Provides the number of eigenvalues to be calculated.

force_complex_solver

value
`yes` or `no`

default
`no`

Should be set flag to `yes`, when the results are ment to be used in the *optics{ }* group.

force_pauli_solver

value
`yes` or `no`

default
`no`

When se to `yes`, the a Pauli equation is solved even in the absence of magnetic field. It may be needed when the results are passed to *optics{ }* group.

lapack{}
LAPACK eigensolver is used to solve dense matrix problem (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{}
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.

accuracy

value
any float > 0

default
1e-10 for LAPACK 1e-7 for ARPACK

accuracy of eigenvalue

iterations

value
any integer > 1

default
500

number of iterations for eigenvalue solver

preconditioner

value
0 or polynomial, 1 or chebyshev, 2 or legendre

default
1 or chebyshev

Polynomial preconditioner is the slowest but does not require to specify cutoff energy whereas chebyshev or legendre preconditioner requires you to specifiy cutoff energy.

order_polynomial

value
any integer > 1

default
15

order of the polynomial used for polynomial preconditioning

order_chebychev

value
any integer > 1

default
20

order of the polynomial used for Chebyshev preconditioning

order_legendre

value
any integer > 1

default
20

order of the polynomial used for Legendre preconditioning

cutoff

value
any float >= 0.001

default
`0.3 # [eV]`

abs_cutoff

value
any float >= 0.001

default
`0.0 # [eV]`

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.

---

dispersion{}
calculate the $\mathbf{k}_{||}$ and $\mathbf{k}_{\mathbf{superlattice}}$ (if applicable) dispersion. 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{}`. For more details, see *quantum{ region{ ... { dispersion{ } } } }*.

quantum{ region{ L{} } }

solves single-band Schrödinger equation for the **L** conduction band. The options are the same as *Gamma{}*.

quantum{ region{ X{} } }

solves single-band Schrödinger equation for the **X** conduction band. The options are the same as *Gamma{}*.

quantum{ region{ Delta{} } }

solves single-band Schrödinger equation for the **Delta** conduction band. The options are the same as *Gamma{}*.

quantum{ region{ HH{} } }

solves single-band Schrödinger equation for the **heavy hole** valence band. The options are the same as *Gamma{}*.

quantum{ region{ LH{} } }

solves single-band Schrödinger equation for the **light hole** valence band. The options are the same as *Gamma{}*.

quantum{ region{ SO{} } }

solves single-band Schrödinger equation for the **split-off hole** valence band. The options are the same as *Gamma{}*.

Multi-band models in quantum{ region{ } }

- *quantum{ region{ kp_6band{ } } }*
  - *kp_parameters{}*
  - *lapack{}*
  - *arpack{}*
  - *k_integration{}*
- *quantum{ region{ kp_8band{ } } }*
  - *num_electrons*
  - *num_holes*
  - *accuracy*
  - *iterations*
  - *kp_parameters{}*
  - *interface{}*
  - *k_integration{}*
  - *lapack{}*
  - *arpack_inv{}*
  - *davidson{}*
  - *shift_window*
  - *shift*
  - *abs_shift*
  - *linear_solver{}*

- *shift_min_CB*
- *shift_max_VB*
- *tunneling*
- *classify_kspace*
- *threshold_classification*
- *full_band_density*
- *spurious_handling*

quantum{ region{ kp_6band{ } } }

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

kp_parameters{}

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.

use_Luttinger_parameters
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`

approximate_kappa
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`

lapack{}

LAPACK eigensolver: solves dense matrix problem (for 1D and small 2D systems only)

arpack{}

ARPACK eigensolver (default) ARPACK should be faster for large matrices (N > 1000) where only a few eigenvalues are sought (~5-30).

k_integration{}

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.

relative_size
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

num_points
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

num_subpoints
number of points between two $\mathbf{k}_{||}$ points, where wave functions and eigenvalues will be interpolated.

value
integer >= 1

default
5

max_symmetry
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

force_k0_subspace
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`

quantum{ region{ kp_8band{ } } }

It solves 8-band $\mathbf{k} \cdot \mathbf{p}$ Schrödinger equation for the Gamma conduction band and the heavy, light and split-off hole valence bands.

num_electrons

value
integer >= 0

default
0

number of electron eigenvalues

num_holes

value
integer >= 0

default
0

number of hole eigenvalues

accuracy

value
any float > 0

default
1e-7

accuracy of eigenvalue

iterations

value
any integer > 1

default
500

number of iterations for eigenvalue solver

kp_parameters{}

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.

use_Luttinger_parameters
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`

from_6band_parameters
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`

approximate_kappa
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`

evaluate_S
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`

rescale_S_to
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

interface{}

Note: Better description will be available soon.

Attention: The feature is already available, yet we are still testing it.

Optional group to add interface effects to the Hamiltonian [*LivnehPRB2012*], [*Livneh-PRB2014*]. It can be used multiple times.

position (required)
A real number defining position of the interface

kp_parameters (required)
The group storing all parameters for the interface Hamiltonian.

D_s (required)
a real number

D_x (required)
a real number

D_z (required)
a real number

alpha (required)
a real number

beta (required)
a real number

reverse (optional)
choice (yes/no)

k_integration{}

Provides options for integration over $\mathbf{k}_{||}$ space for $\mathbf{k} \cdot \mathbf{p}$ density calculations (for 1D and 2D only) same as *kp_6band{ k_integration{}}*

lapack{}

LAPACK eigensolver: solves dense matrix problem (for 1D and small 2D systems only)

arpack_inv{}

ARPACK shift invert eigensolver. ARPACK should be faster for large matrices (N > 1000) where only a few eigenvalues are sought (~5-30).

davidson{}

When called, the Davidson solver is used to solve Schrödinger equation.

Hint: The Davidson solver for 8-band k.p offers both better speed as well as increased stability compared to ARPACK inverse in 2D and 3D.

Warning: This routine 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.

shift_window

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.

shift

value
float >=0

default
`0.1 # [eV]`

energy shift relative to band edges in `arpack_inv`.

abs_shift

value
float >=0

default
`0.0 # [eV]`

energy shift on an absolute energy scale in `arpack_inv`.

linear_solver{}

Provides parameters for linear equation solver in `arpack_inv` shift invert preconditioner

iterations

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.

abs_accuracy

value
float between 0.0 and 0.01

default
1e-8

absolute accuracy in `arpack_inv`.

rel_accuracy

value
float between 0.0 and 0.01

default
1e-8

relative accuracy in `arpack_inv`.

use_cscg

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.

force_diagonal_preconditioner

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.

shift_min_CB

value
float

default
0.0

(relevant only if `classify_kspace = 0`) Shifts the minimum of the conduction band to manipulate cutoff energy and thereby the quantum density classification.

shift_max_VB

value
float

default
0.0

(relevant only if `classify_kspace = 0`) Shifts the maximum of the valence band to manipulate cutoff energy and thereby the quantum density classification.

tunneling

value
`yes` or `no`

default
`yes`

(relevant only if `classify_kspace = 0`) Choice of the (position-dependent) cutoff energy. `yes` defines the cutoff energy at max((minimum of the conduction band in the structure), (position-dependent valence band edge)), while `no` sets it to min((maximum of the valence band in the structure), (position-dependent conduction band edge)).

classify_kspace

value
0, 1, 2, or 3

default
0

Choice of the classification method in the 8-band k.p quantum density calculation.

- `classify_kspace = 0`: Eigenstates are classified by comparing the zone-center eigenvalues with the (possibly position-dependent) cutoff energies. For the definition of cutoff energies, see `shift_min_CB`, `shift_max_VB`, and `tunneling`.
- `classify_kspace = 1`: Eigenstates are classified by comparing the zone-center spinor composition with `threshold_classification`.
- `classify_kspace = 2`: Eigenstates are classified at each in-plane k vector (1D simulation) and at each k value (2D simulation) using spinor composition averaged with the neighbouring k points.

- `classify_kspace = 3`: Eigenstates are classified at each in-plane k vector (1D simulation) and at each k value (2D simulation) using spinor composition averaged with the neighbouring k points, but skipping the average if any of the neighbouring k points has the opposite sign of charge. The resulting quantum density will be different from the case `classify_kspace = 2` if electron-hole hybridization occurs (e.g. type-II broken-gap superlattices).

threshold_classification

value
0.0 <= float <= 1.0

default
0.5

(relevant only if `classify_kspace >= 1`) Classify states to electrons if the electron spinor composition is greater than this threshold and otherwise to holes.

full_band_density

value
`yes` or `no`

default
`no`

Calculate density by filling all states above Fermi level with holes and subtracting a negative background charge (`lapack` only). This ignores `classify_kspace`.

spurious_handling

value
six dimensional double vector

default
[0.0, 1.0, -1.0, 1.0, 0.0, 0.0]

- `first component:` If value > 0, forward-/backward differences are used for the first derivative discretization of the P material parameter (Kane parameter) in the 8-band k.p Hamiltonian. Default is 0 (= FALSE), i.e. centered differences are used instead. This parameter might affect spurious solutions of the wave functions. See eq. (1.50) and eq. (1.51) of *PhD thesis T. Andlauer*.
- `second component:` far-band contribution to electrons = value - 1.0 (conduction band g factor, should be a material parameter but it is not) (default is: 1.0) S = 1 + farband contribution, by default farband contribution = 0. This corresponds to setting S=1. It can be useful to set this value to 0.0 (farband contribution = -1). Then it corresponds to setting S=0. Otherwise, the default is rescaling to that S=1.
- `third component:` correction for electron g factor [eV] (default is: -1.0)
- `fourth component:` If value > 0, rescale everywhere (default is: 1 = TRUE)
- `fifth component:` If value > 0, upwinding is TRUE (default is: 0 = FALSE) ==> It seems that upwinding is not used at all.
- `sixth component:` If value > 0, avoid spurious solutions. (default is: 0 = FALSE)

To avoid spurious solutions, an example configuration could be given by `spurious_handling = [0.0, 1.0, 0.0, 1.0, 0.0, 1.0]`.

Matrix elements in quantum{ region{} }

- *interband_matrix_elements{}* (optional)
- *intraband_matrix_elements{}* (optional)
- *dipole_moment_matrix_elements{}* (optional)

interband_matrix_elements{} *(optional)*

Provides the option to calculate interband matrix elements between wave functions of two different bands.

output_matrix_elements *(optional)*
If `output_matrix_elements = yes` then matrix elements are saved in output file.

type
choice

values
`yes` or `no`

default
`yes`

output_transition_energies *(optional)*
If `output_transition_energies = yes` then transition energies are saved in output file.

type
choice

values
`yes` or `no`

default
`no`

KP6_Gamma{} *(optional)*
$\sum_k \langle kp6_{k,i} | \Gamma_j \rangle$, 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{} *(optional)*
Matrix element of the transition between the heavy hole valence band and the gamma conduction band $\langle HH_i | \Gamma_j \rangle$

LH_Gamma{} *(optional)*
Matrix element of the transition between the light hole valence band and the gamma conduction band $\langle LH_i | \Gamma_j \rangle$

SO_Gamma{} *(optional)*
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{} *(optional)*
Matrix element of the transition between the heavy hole valence band and the Delta conduction band $\langle LH_i | \Delta_j \rangle$

LH_Delta{} *(optional)*
Matrix element of the transition between the light hole valence band and the Delta conduction band $\langle LH_i | \Delta_j \rangle$

SO_Delta{} *(optional)*
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{} *(optional)*
Matrix element of the transition between the heavy hole valence band and the X conduction band $\langle HH_i|X_j\rangle$

LH_X{} *(optional)*
Matrix element of the transition between the light hole valence band and the X conduction band $\langle LH_i|X_j\rangle$

SO_X{} *(optional)*
Matrix element of the transition between the split-off valence band and the X conduction band $\langle SO_i|X_j\rangle$

HH_L{} *(optional)*
Matrix element of the transition between the heavy hole valence band and the L conduction band $\langle HH_i|L_j\rangle$

LH_L{} *(optional)*
Matrix element of the transition between the light hole valence band and the L conduction band $\langle LH_i|L_j\rangle$

SO_L{} *(optional)*
Matrix element of the transition between the split-off valence band and the L conduction band $\langle SO_i|L_j\rangle$

intraband_matrix_elements{} *(optional)*

Calculate intraband matrix elements $\langle i|\epsilon\cdot\hat{\mathbf{p}}|j\rangle$ for wave functions within one band. The light polarization direction ϵ is automatically normalized in the program. $\hat{\mathbf{p}}=i\hbar\nabla$ is the momentum vector.

For further reading: J. H. Davies, *The Physics of Low-Dimensional Semiconductors. An Introduction*, 2006, Chapters 10 and 8.

name *(optional)*
defines suffix for related output files

type
string

direction *(optional)*
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.

value
3D real vector

default
[1 , 0 , 0]

output_matrix_elements *(optional)*
If `output_matrix_elements = yes` then matrix elements are saved in output file.

type
choice

values
`yes` or `no`

default
`yes`

output_transition_energies *(optional)*
If `output_transition_energies = yes` then transition energies are saved in output file.

type
choice

values
`yes` or `no`

default
`no`

output_oscillator_strengths *(optional)*
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.

type
choice

values
`yes` or `no`

default
`no`

Gamma{} *(optional)*
Calculates the matrix element $\langle \Gamma_i | \epsilon \cdot \hat{\mathbf{p}} | \Gamma_j \rangle$.

X{} *(optional)*
Calculates the matrix element $\langle X_i | \epsilon \cdot \hat{\mathbf{p}} | X_j \rangle$.

Delta{} *(optional)*
Calculates the matrix element $\langle \Delta_i | \epsilon \cdot \hat{\mathbf{p}} | \Delta_j \rangle$.

L{} *(optional)*
Calculates the matrix element $\langle L_i | \epsilon \cdot \hat{\mathbf{p}} | L_j \rangle$.

HH{} *(optional)*
Calculates the matrix element $\langle HH_i | \epsilon \cdot \hat{\mathbf{p}} | HH_j \rangle$.

LH{} *(optional)*
Calculates the matrix element $\langle LH_i | \epsilon \cdot \hat{\mathbf{p}} | LH_j \rangle$.

SO{} *(optional)*
Calculates the matrix element $\langle SO_i | \epsilon \cdot \hat{\mathbf{p}} | SO_j \rangle$.

KP6{} *(optional)*
Calculates the matrix element $\sum_k \langle kp6_{k,i} | \epsilon \cdot \hat{\mathbf{p}} | kp6_{k,j} \rangle$, k = 1,...,6.

KP8{} *(optional)*
Calculates the matrix element $\sum_k \langle kp8_{k,i} | \epsilon \cdot \hat{\mathbf{p}} | kp8_{k,j} \rangle$, k = 1,...,8.

dipole_moment_matrix_elements{} *(optional)*

Calculate dipole moment matrix elements $\langle i|\epsilon \cdot \hat{\mathbf{d}}|j\rangle$ for wave functions within one band. The light polarization direction ϵ is automatically normalized in the program. $\hat{\mathbf{d}} = e\hat{\mathbf{r}}$ is the dipole moment vector.

For further reading: J. H. Davies, *The Physics of Low-Dimensional Semiconductors. An Introduction*, 2006, Chapters 10 and 8.

name ***(optional)***
defines suffix for related output files

type
string

direction ***(optional)***
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.

value
3D real vector

default
[1 , 0 , 0]

output_matrix_elements ***(optional)***
If `output_matrix_elements = yes` then matrix elements are saved in output file.

type
choice

values
`yes` or `no`

default
`yes`

output_transition_energies ***(optional)***
If `output_transition_energies = yes` then transition energies are saved in output file.

type
choice

values
`yes` or `no`

default
`no`

output_oscillator_strengths ***(optional)***
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.

type
choice

values
`yes` or `no`

default
`no`

Gamma{} *(optional)*
Calculates the matrix element $\langle \Gamma_i | \epsilon \cdot \hat{\mathbf{d}} | \Gamma_j \rangle$.

X{} *(optional)*
Calculates the matrix element $\langle X_i | \epsilon \cdot \hat{\mathbf{d}} | X_j \rangle$.

Delta{} *(optional)*
Calculates the matrix element $\langle \Delta_i | \epsilon \cdot \hat{\mathbf{d}} | \Delta_j \rangle$.

L{} *(optional)*
Calculates the matrix element $\langle L_i | \epsilon \cdot \hat{\mathbf{d}} | L_j \rangle$.

HH{} *(optional)*
Calculates the matrix element $\langle HH_i | \epsilon \cdot \hat{\mathbf{d}} | HH_j \rangle$.

LH{} *(optional)*
Calculates the matrix element $\langle LH_i | \epsilon \cdot \hat{\mathbf{d}} | LH_j \rangle$.

SO{} *(optional)*
Calculates the matrix element $\langle SO_i | \epsilon \cdot \hat{\mathbf{d}} | SO_j \rangle$.

KP6{} *(optional)*
Calculates the matrix element $\sum_k \langle kp6_{k,i} | \epsilon \cdot \hat{\mathbf{d}} | kp6_{k,j} \rangle$, k = 1,...,6.

KP8{} *(optional)*
Calculates the matrix element $\sum_k \langle kp8_{k,i} | \epsilon \cdot \hat{\mathbf{d}} | kp8_{k,j} \rangle$, k = 1,...,8.

Output groups in quantum{ region{} }

quantum{ region{ output_wavefunctions{} } }

Provides options for output of wave function data

max_num

value
any integer between 1 and 9999

default
1.0

all_k_points

value
`yes` or `no`

default
false

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

value
yes or no

default
no

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.

amplitudes

value
string

default
" no "

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

value
string

default
yes

Prints out the wave functions $|\psi|^2$ in units of 1D: nm^{-1}, 2D: nm^{-2}, 3D: nm^{-3}.

options
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)

If multiple choices are required type them together inside a string like

```
probabilities = "
                yes
                CB_HH_LH_SO
                "
```

scale

value
float

default
1.0

scale factor for output of amplitudes and probabilities

in_one_file

value
`yes` or `no`

default
`yes`

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.

energy_shift

value
string

default
both

options
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

value
`yes` or `no`

default
`yes`

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*.

---

quantum{ region{ output_subband_densities{} } }

Provides options for output of subband densities.

max_num

value
any integer between 0 and 9999

default
1

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

value
`yes` or `no`

default
`yes`

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.

quantum{ region{ output_rotated_inverse_mass_tensor{} } }

Outputs components of tensor of the inverse mass in simulation coordinate system

boxes ***(optional)***

value
`yes` or `no`

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

value
`yes` or `no`

default
`no`

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.

quantum{ region{ quantize...{ } } }

quantize_x{}
In 2D or 3D simulation, the Schrödinger equation is solved within the 1D slices parallel to the x direction. 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.

Note: Quantum decomposition regions cannot be used for CBR or *run{ quantum_optics{ } }* and `shifted_neumann` boundary conditions are not supported for the direction of the decomposition (here x-direction).

Warning: This keyword should be used only for structures in which quantization in other than x-direction is not expected to be relevant. Otherwise, certain relevant quantum properties of the simulated structure may get lost while using this group.

quantize_y{}
The same as `quantize_x{}`, but the slices are in y-direction.

quantize_z{}
The same as `quantize_x{}`, but the slices are in z-direction and only in 3D simulation.

quantum{ region{ boundary{} } }

Specifies the boundary condition for Schrödinger equation along various axis dimensions. In general, **Dirichlet** boundary conditions correspond to $f = \text{constant}$ and **Neumann** boundary conditions correspond to $df/dx = \text{constant}$. 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.

quantum{ region{ boundary{ x } } }
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.

type
choice

values
dirichlet / neumann / shifted_neumann

default
neumann

quantum{ region{ boundary{ y } } }
Specifies boundary conditions at the borders of respective `quantum{ region{} }` in the **y** 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.

type
choice

values
dirichlet / neumann / shifted_neumann

default
neumann

quantum{ region{ boundary{ z } } }
Specifies boundary conditions at the borders of respective `quantum{ region{} }` in the **z** 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.

type
choice

values
dirichlet / neumann / shifted_neumann

default
neumann

quantum{ region{ boundary{ classical_boundary_x } } }

type
choice

value
`yes` or `no`

default
no

quantum{ region{ boundary{ classical_boundary_y } } }

type
choice

value
`yes` or `no`

default
no

quantum{ region{ boundary{ classical_boundary_z } } }

type
choice

value
yes or no

default
no

quantum{ region{ boundary{ num_classical_x } } }

value
2D integer vector

default
[1 , 1]

quantum{ region{ boundary{ num_classical_y } } }

value
2D integer vector

default
[1 , 1]

quantum{ region{ boundary{ num_classical_z } } }

value
2D integer vector

default
[1 , 1]

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).

---

quantum{ region{ transition_energies{} } }

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`.

- 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{}

quantum{ region{ lifetimes{} } }

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.

phonon_energy
LO phonon energy

value
any float > 0.0

default
0.01

Gamma{}, X{}, Delta{}, L{}, HH{}, LH{}, SO{}
One-band models for computing the lifetimes. At lease one has to be chosen.

quantum{ region{ excitons{} } }

An **optional** group triggering computation of binding energies of excitons.

Note: This feature is under development.

---

electron_mass
Effective mass of electron involved in the exciton

type
real

values
1e-3 < x < 10

default
volume average of values from database

hole_mass
Effective mass of hole involved in the exciton

type
real

values
1e-3 < x < 10

default
volume average of values from database

density_averaged_masses
Effective masses of hole and electron are averaged with weights taken from probability densities of related states

type
choice

values
`yes` or `no`

default
`no`

dielectric_const
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

type
real

values
1 < x < 1e3

default
volume average of values from database

energy_cutoff
Maximum energy difference of electron and hole states involved in forming exciton

type
real

values
1e-3 < x

accuracy ***(optional)***
Accuracy used in minimisation procedure to compute the exciton binding energy

type
real

values
1e-10 < x < 0.1

quantum{ region{ ... { dispersion{ } } } }

- *path{ }*
  - *name*
  - *point{ }*
  - *spacing*
  - *num_points*
- *lines{ }*

- *name*
- *spacing*
- *k_max*

- *full{ }*
  - *name*
  - *kxgrid{ }*
  - *kygrid{ }*
  - *kzgrid{ }*
- *superlattice{ }*
  - *name*
  - *num_points_x*
  - *num_points_y*
  - *num_points_z*
  - *num_points*
- *output_dispersions{ }*
  - *max_num*
- *output_masses{ }*
  - *max_num*

This section reffers to the groups:

- `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{ } } } },`
- `quantum{ region{ kp_8band{ dispersion{ } } } }`, and
- `quantum{ region{ kp_6band{ dispersion{ } } } }`.

These groups provide keywords to define a path for computation of electronic band structures - $\mathbf{k}_{||}$ 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{ }`.

path{ }

`quantum{ region{ ...{ dispersion{ path{ } } } } }` calculates dispersion along custom path in k-space. Multiple instances are allowed.

name

`quantum{ region{ ...{ dispersion{ path{ name } } } } }` is a name of the dispersions which also defines the names of the output files.

value
string

point{ }

`quantum{ region{ ...{ dispersion{ path{ point{ } } } } } }` 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.

k

value
3D float vector

`quantum{ region{ ...{ dispersion{ path{ point{ k } } } } } }` 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.

spacing

value
float

`quantum{ region{ ...{ dispersion{ path{ spacing } } } } }` specifies approximate spacing for intermediate points in the path segments in nm^{-1}. Excludes `num_points`.

num_points

value
integer > 1

`quantum{ region{ ...{ dispersion{ path{ num_points } } } } }` specifies number of points (intermediate + two corner points) for each single path segment. Excludes `spacing`.

lines{ }

`quantum{ region{ ...{ dispersion{ lines{ } } } } }` calculate dispersions along some predefined paths of high symmetry in k-space, e.g. [100], [110], [111] and their equivalents (in total maximally 13).

name

value
string

`quantum{ region{ ...{ dispersion{ lines{ name } } } } }` is a name of the dispersions which also defines the names of the output files.

spacing

value
float

`quantum{ region{ ...{ dispersion{ lines{ spacing } } } } }` specifies approximate spacing for intermediate points in the path segments in nm^{-1}.

k_max

value
float

`quantum{ region{ ...{ dispersion{ lines{ k_max } } } } }` specifies a maximum absolute value (radius) for the k-vector in nm^{-1}.

full{ }

`quantum{ region{ ...{ dispersion{ full{ } } } } }` calculates dispersion in 1D/2D/3D k-space depending on simulation dimensionality and pereodic boundary conditions.

name

value
string

`quantum{ region{ ...{ dispersion{ full{ name } } } } }` is a name of the dispersion which also defines the name of the output file.

kxgrid{ }

`quantum{ region{ ...{ dispersion{ full{ kxgrid{ } } } } } }` 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{ }*

kygrid{ }

`quantum{ region{ ...{ dispersion{ full{ kygrid{ } } } } } }` is analogous to `kxgrid{ }`.

kzgrid{ }

`quantum{ region{ ...{ dispersion{ full{ kzgrid{ } } } } } }` is analogous to `kxgrid{ }`.

superlattice{ }

`quantum{ region{ ...{ dispersion{ superlattice{ } } } } }` 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 $i = x, y, z$.

name

value
string

`quantum{ region{ ...{ dispersion{ superlattice{ name } } } } }` is a name of the dispersion which also defines the name of the output file.

num_points_x

value
any integer > 1

`quantum{ region{ ...{ dispersion{ superlattice{ num_points_x } } } } }` 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.

num_points_y

value
any integer > 1

`quantum{ region{ ...{ dispersion{ superlattice{ num_points_y } } } } }` 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.

num_points_z

value
any integer > 1

`quantum{ region{ ...{ dispersion{ superlattice{ num_points_z } } } } }` 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.

num_points

value
any integer > 1

`quantum{ region{ ...{ dispersion{ superlattice{ num_points } } } } }` is a convenience keyword to specifies number of points along all appropriate directions in k space.

output_dispersions{ }

`quantum{ region{ ...{ dispersion{ output_dispersions{ } } } } }` outputs all defined dispersions.

max_num

value
any integer between 1 and 9999

`quantum{ region{ ...{ dispersion{ output_dispersions{ max_num } } } } }` is a number of bands to print out

output_masses{ }

`quantum{ region{ ...{ dispersion{ output_masses{ } } } } }` 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.

max_num

value
any integer between 1 and 9999

`quantum{ region{ ...{ dispersion{ output_masses{ max_num } } } } }` output effective masses calculated from the dispersions.

quantum{ exchange_correlation{ } }

Attention: The feature is currently available only for 1-band models. It is ignored for multi-band $\mathbf{k} \cdot \mathbf{p}$ models.

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.

type

value
string

options
lda : Include exchange-correlation effects in the LDA approximation (Local Density Approximation)

lsda : Include exchange-correlation effects in the LSDA approximation (Local Spin Density Approximation)

initial_spin_pol

value
float between 0.0 and 1.0

default
0.0

Breaks spin up/down symmetry if no magnetic field is present.

output_spin_polarization{}
output spin polarization [dimensionless]

output_exchange_correlation{}
output exchange correlation potentials in `[eV]`.

quantum{ cbr{} } *(optional)*

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]*

At a glance: CBR current calculation

- full 1D, 2D and 3D calculation of quantum mechanical ballistic transmission probabilities for open systems with scattering boundary conditions
- **C**ontact **B**lock **R**eduction 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
}
```

Attention: Following conditions has to be satisfied to use the `quantum{ cbr{} }` group:

- 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

Basic Definition

quantum{ cbr{ name } } ***(required)***
refers to quantum region to which CBR method will be applied (d-dimensional)

type
string

quantum{ cbr{ lead{} } } ***(required)***
Defining a lead. The lead region has dimension $d - 1$.

quantum{ cbr{ lead{ name } } } ***(required)***
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.

type
string

quantum{ cbr{ lead{ x } } } ***(optional)***

type
real number

unit
nm

default
0.0

constraints
only for 1D simulations

quantum{ cbr{ lead{ kinetic_coupling } } } ***(optional)***

type
real number

unit
eV

default
disabled

constraints
> 0.0 and `rel_kinetic_coupling` is not defined

quantum{ cbr{ lead{ rel_kinetic_coupling } } } ***(optional)***

type
real number

default
1.0

constraints
> 0.0 and `kinetic_coupling` is not defined

Energy & Transmission

quantum{ cbr{ min_energy } } ***(optional)***
Lower boundary for transmission energy interval on an absolute energy scale

value
real number

unit
eV

default
-1e100

constraints
`rel_min_energy` is not defined

quantum{ cbr{ max_energy } } ***(optional)***
Upper boundary for transmission energy interval on an absolute energy scale

value
real number

unit
eV

default
1e100

constraints
`rel_max_energy` is not defined

quantum{ cbr{ rel_min_energy } ***(optional)***
Lower boundary for transmission energy interval relative to the lowest eigenvalue

value
real number

default
-1e100

constraints
`min_energy` is not defined

quantum{ cbr{ rel_max_energy } ***(optional)***
Upper boundary for transmission energy interval relative to the highest eigenvalue

value
real number

default
1e100

constraints
`max_energy` is not defined

quantum{ cbr{ energy_resolution } } ***(optional)***
This value determines the resolution of the transmission curve $T(E)$.

value
real number

unit
eV

default
1e-4

quantum{ cbr{ transmission_threshold } } ***(optional)***
This value determines the resolution of the transmission curve $T(E)$.

type
real number

default
0.0

constraints
≥ 0.0

Densities of States

quantum{ cbr{ ildos } } ***(optional)***
Outputs **i**ntegrated **l**ocal **d**ensity of **s**tates.

type
choice

values
`yes` or `no`

default
`no`

quantum{ cbr{ ldos } } ***(optional)***
Outputs **l**ocal **d**ensity of **s**tates.

type
choice

values
`yes` or `no`

default
`no`

quantum{ cbr{ output_ldos_single_file } } ***(optional)***

type
choice

values
`yes` or `no`

default
`yes`

> **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

quantum{ cbr{ two_particle_options } } *(optional)*
11 values for two-particle model `[number of states, relative permittivity, x1, y1, z1, x2, y2, z2, splitting, tunneling]`

type
array of 3 real numbers

units
[–, –, nm, nm, nm, nm, nm, nm, eV, eV]

default
not initialized

constraints
number of states = 2

```
numStates2_   =  (int)two_particle_options_[0];
const double epsRel = two_particle_options_[1];
const DVector3 r1(two_particle_options_[2]*uNanometer,two_particle_
↪options_[3]*uNanometer,two_particle_options_[4]*uNanometer);
const DVector3 r2(two_particle_options_[5]*uNanometer,two_particle_
↪options_[6]*uNanometer,two_particle_options_[7]*uNanometer);
const double delta = two_particle_options_[8]*uEVolt; // splitting
const double z     = two_particle_options_[9]*uEVolt; // tunneling

// [prefactor] = Q^2/[cEps0], [cEps0] = Q/L*V => [prefactor] = Q L␣
↪V = eV L

const double prefactor = two_particle_options_[10] * sqr(cEcharge)/
↪(4*Pi*epsRel*cEps0);
```

Example

Figure 6.5.14.1 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: The quantum region must be a surface in a 3D simulation, a line in a 2D simulation, and a point in a 1D simulation.

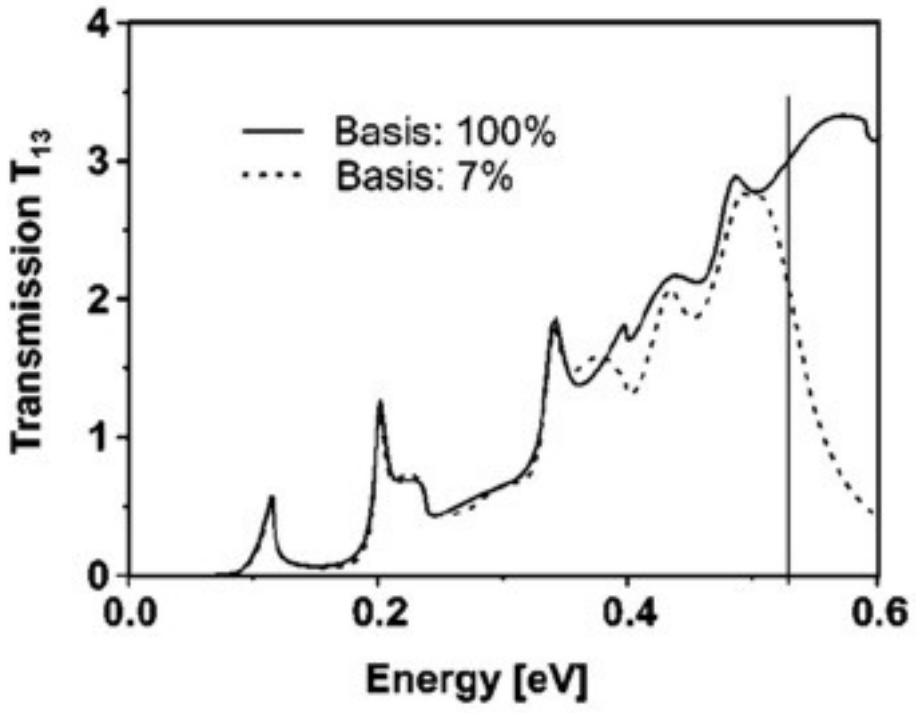


Figure 6.5.14.1: The transmission calculated with the CBR method using all eigenstates and only 7% of the eigenstates. In the latter case, the transmission is still very accurate for the lower energies.

quantum{ debuglevel }

value
any integer between -1 and 3

default
1

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{ }`).

quantum{ allow_overlapping_regions }

value
`yes` or `no`

default
`no`

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.

Code Example

```
quantum{
    debuglevel      = 1
    allow_overlapping_regions = no
```

(continues on next page)

(continued from previous page)

```
#----------------
# Quantum regions
#----------------
region{
    name = "qr1"

    quantize_x{}
    quantize_y{}
    quantize_z{}

    no_density = yes
    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
    }
```

(continues on next page)

(continued from previous page)

```
    # 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
        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
```

(continues on next page)

(continued from previous page)

```
            num_points_z   = 20
            num_points     = 20
        }
    }
}

L{
    ... (same as Gamma)
}

X{
    ... (same as Gamma)
}

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
```

(continues on next page)

(continued from previous page)

```
        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)
        }

        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

        full_band_density = no
    }

    #Matrix elements definitions
    #---------------------------
    interband_matrix_elements{
        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 #
```

(continues on next page)

(continued from previous page)

```
        }

        intraband_matrix_elements{
            direction = [1,1,0]
            Gamma{}
            Delta{}
            X{}
            L{}
            HH{}
            LH{}
            SO{}
            KP6{}
            KP8{}

            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{}
```

(continues on next page)

(continued from previous page)

```
            SO{}
            KP6{}
            KP8{}
        }


        lifetimes{
            phonon_energy = 0.036
            Gamma{}
            HH{}
            LH{}
        }

    } # end: region{}

    #Many body effects
    #-----------------
    exchange_correlation{
        type             = lda
        initial_spin_pol = 1.0
        output_spin_polarization{}
        output_exchange_correlation{}
    }

}
```

6.5.15 optics{ }

- using: optional within the scope
- items: maximum 1

This group defines models to calculate optical spectra.

The following keywords are available within this group.

optics{ debuglevel }

- using: optional within the scope
- type: integer
- values: `{-1, 0, 1, ..., 4}`
- unit: –
- default: `0`

Parameter controlling diagnostic output in the *.log file. The larger the value is, the more details are included.

optics{ global_illumination{ } }

- using: optional within the scope
- 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 }*

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

- using: optional within the scope
- type: integer
- values: `{-1, +1}`
- unit: $-$

Sets ascending $+1$ or descending -1 direction of illuminating radiation along the x-axis of simulation.

---

direction_y

- using: optional within the scope
- type: integer
- values: `{-1, +1}`
- unit: $-$

Sets ascending $+1$ or descending -1 direction of illuminating radiation along the y-axis of simulation.

---

direction_z

- using: optional within the scope
- type: integer
- values: `{-1, +1}`
- unit: $-$

Sets ascending $+1$ or descending -1 direction of illuminating radiation along the z-axis of simulation.

---

database_spectrum{ }

- using: optional within the scope
- 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 }

- using: required within the scope
- type: character string

Name of the illumination spectrum contained in the database to be used.

---

database_spectrum{ concentration }

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- default: `1.0`
- unit: $-$

Scaling factor multiplying the values of the spectrum.

---

import_spectrum{ }

- using: optional within the scope
- 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 }

- using: required within the scope
- type: character string

Reference name used in the `import{ }` group to label the imported spectrum.

---

import_spectrum{ cutoff }

- using: required within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: required within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- unit: –
- default: `1.0`

Scaling factor multiplying the values of the spectrum.

---

constant_spectrum{ }

- using: optional within the scope
- items: maximum 1

Define illumination source with a constant radiation spectrum of the form

$$I(E) = \frac{I_0}{E_{\max} - E_{\min}}$$

---

constant_spectrum{ irradiance }

- using: required within the scope
- 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{ }

- using: optional within the scope
- items: no constraints

Define illumination source with a black-body radiation spectrum

$$I(E,T) = \frac{I_0}{\sigma T^4} \frac{2\pi E^3}{c^2 h^3} \frac{1}{\exp\left\{\left(\frac{E}{k_B T}\right)\right\} - 1},$$

where σ is the Stefan–Boltzmann constant.

---

planck_spectrum{ irradiance }

- using: required within the scope
- 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 }

- using: required within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: K

Temperature T entering the spectrum model

---

lorentzian_spectrum{ }

- using: optional within the scope
- 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 }

- using: required within the scope
- 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 }

- using: optional within the scope
- type: real number
- values: `[10.0, ...)`
- unit: nm

Central wavelength λ_0 of the spectrum

---

lorentzian_spectrum{ energy }

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: eV

Central energy E_0 of the spectrum

---

lorentzian_spectrum{ width }

- using: optional within the scope
- type: real number
- values: `[1e-3, ...)`
- unit: nm

Define the width of the spectrum in nm

---

lorentzian_spectrum{ gamma }

- using: optional within the scope
- 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

- using: optional within the scope
- 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 }

- using: required within the scope
- 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 }

- using: optional within the scope
- type: real number
- values: `[10.0, ...)`
- unit: nm

Central wavelength λ_0 of the spectrum

---

gaussian_spectrum{ energy }

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: eV

Central energy E_0 of the spectrum

---

gaussian_spectrum{ width }

- using: optional within the scope
- type: real number
- values: [1e-3, ...)
- unit: nm

Define the width of the spectrum in nm

---

gaussian_spectrum{ gamma }

- using: optional within the scope
- 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=?)
```

(continues on next page)

(continued from previous page)

```
        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|
}
```

optics{ global_reflectivity{ } }

- using: optional within the scope
- 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*
  - *database_spectrum{ }*
  - *database_spectrum{ name }*

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{ }

- using: optional within the scope
- items: maximum 1

Importing the spectrum from the database or external files.

database_spectrum{ name }

- using: required within the scope
- type: character string

Name of the spectrum contained in the database.

import_spectrum{ }

- using: optional within the scope
- 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 }

- using: required within the scope
- type: character string

Path to a spectrum for importing

---

import_spectrum{ cutoff }

- using: required within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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{ }

- using: optional within the scope
- items: maximum 1

Specify a constant reflectance spectrum $R(\lambda) = \text{const}$

---

constant_spectrum{ reflectivity }

- using: required within the scope
- type: real number
- values: `(0.0, 1.0]`
- unit: $-$
- default: `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)
}
```

optics{ global_absorption_coeff{ } }

- using: optional within the scope
- 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{ }

- using: optional within the scope
- items: maximum 1

Importing absorption spectra from the database or external files.

---

database_spectrum{ name }

- using: required within the scope
- type: character string

Name of the spectrum contained in the database.

---

import_spectrum{ }

- using: optional within the scope
- 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 }

- using: required within the scope
- type: character string

Path to a spectrum for importing

---

import_spectrum{ cutoff }

- using: required within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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{ }

- using: optional within the scope
- 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 }

- using: optional within the scope
- 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 }

- using: optional within the scope
- 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
    }
}
```

optics{ global_refractive_index{ } }

- using: optional within the scope
- items: maximum 1

This group is used to specify the effective refractive index $n_{\text{eff}}(\lambda)$ 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{ }

- using: optional within the scope
- items: maximum 1

Importing the spectrum from the database or external files.

---

database_spectrum{ name }

- using: required within the scope
- type: character string

Name of the spectrum contained in the database.

---

import_spectrum{ }

- using: optional within the scope
- 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 }

- using: required within the scope
- type: character string

Path to a spectrum of the real part of the refractive index for importing

---

import_spectrum{ import_k_from }

- using: optional within the scope
- type: character string

Path to a spectrum of the imaginary part of the refractive index for importing

---

import_spectrum{ cutoff }

- using: required within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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{ }

- using: optional within the scope
- items: maximum 1

Specify a constant refractive index spectrum

---

constant_spectrum{ n }

- using: required within the scope
- type: real number
- values: `(0.0, ...)`
- unit: –
- default: `1.0`

Constant value of the real part of the refractive index.

---

constant_spectrum{ k }

- using: optional within the scope
- type: real number
- values: no constraints
- unit: –
- default: `0.0`

Constant value of the imaginary part of the refractive index.

---

compute_absorption_coeff{ }

- using: required within the scope
- type: choice
- choices: `yes`; `no`

Calculate absorption coefficient from imaginary part of the reflective index.

optics{ light_propagation{ } }

- using: optional within the scope
- 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_absorption{ }* and *use_computed_absorption{ }* can be defined within this group.

- *Maintained Keywords*
  - *min_wavelength*
  - *max_wavelength*
  - *min_energy*
  - *max_energy*
  - *energy_resolution*
  - *use_local_absorption{ }*
  - *use_computed_absorption{ }*
  - *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_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

- using: optional within the scope
- type: real number
- values: `[10.0, 1e6]`
- unit: nm

---

max_wavelength

- using: optional within the scope
- type: real number
- values: `[10.0, 1e6]`
- unit: nm

---

min_energy

- using: optional within the scope
- type: real number
- values: `[1e-6, 100.0]`
- unit: eV

Low-energy boundary of the energy grid for propagating photons.

---

max_energy

- using: optional within the scope
- type: real number
- values: `[1e-6, 100.0]`
- unit: eV

High-energy boundary of the energy grid for propagating photons.

---

energy_resolution

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: eV
- default: `1e-2`

Spacing between subsequent energy grid points.

---

use_local_absorption{ }

- using: optional within the scope
- items: maximum 1

If this group is present, the global absorption spectrum is used within local absorption framework in non-contact regions. Zero absorption coefficient, perfect optical transparency, is assigned in contact regions which impose boundary conditions on the Poisson equation (see *contacts{ }*).

Note: In the future, this feature is planned to use imported position-dependent optical absorption spectra.

---

use_computed_absorption{ }

- using: optional within the scope
- items: maximum 1

If this group is present, then the spatially-resolved absorption coefficient from `semiclassical_spectra{ }` is used for calculating the optical field.

Dependencies

- *energy_grid{ }* must be defined.
- *optics{ semiclassical_spectra{ } }* must be defined.
- *output_local_spectra{ }* must be defined.
- *energy_resolution* is not specified in the input file.
- *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{ }

- using: required within the scope
- items: exactly 1

This group is used to output optical spectra which entered the calculation of the light propagation through the device.

---

output_global_spectra{ reflectivity }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then the reflectivity spectrum is outputted.

---

output_global_spectra{ absorption_coeff }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then the absorption spectrum is outputted.

---

output_global_spectra{ decadic_absorption_coeff }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then the absorption spectrum in decadic units is outputted.

---

output_global_spectra{ refractive_index }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then the refractive index spectrum is outputted.

---

output_global_spectra{ spectra_over_energy }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes` then the selected spectra are outputted over photon energy.

---

output_global_spectra{ spectra_over_frequency }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes` then the selected spectra are outputted over photon frequency.

---

output_global_spectra{ spectra_over_wavenumber }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes` then the selected spectra are outputted over photon wavenumber.

---

output_global_spectra{ spectra_over_wavelength }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes` then the selected spectra are outputted over photon wavelength.

---

output_light{ }

- using: required within the scope
- items: exactly 1

---

output_light{ illumination }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes`, then the illumination spectrum is outputted.

---

output_light{ total_absorption }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes` then the selected spectra are outputted over photon energy.

---

output_light{ spectra_over_frequency }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes` then the selected spectra are outputted over photon frequency.

---

output_light{ spectra_over_wavenumber }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes` then the selected spectra are outputted over photon wavenumber.

---

output_light{ spectra_over_wavelength }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes` then the selected spectra are outputted over photon wavelength.

---

output_light{ photon_spectra }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then photon power spectrum is outputted with units W/cm^2.

optics{ photogeneration{ } }

- using: optional within the scope
- 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*
  - *output*
  - *output_integrated*
  - *output_energy_resolved*

- *Examples*

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

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes`, then the generation rate as function of position $G(x)$ is outputted.

output_integrated

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then the generation rate as function of energy $G(E)$ is outputted.

output_energy_resolved

- using: optional within the scope
- type: choice
- choices: `yes`; `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)
}
```

optics{ semiclassical_spectra{ } }

- using: optional within the scope
- 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 $R_{\mathrm{radiative}}(x, E) = C(x) \int dE_h \int dE_e \; n(x, E_e) p(x, E_h) \delta(E_e - E_h - E)$, where $C(x)$ $[\mathrm{cm}^3/\mathrm{s}]$ is the (material-dependent) radiative recombination parameter. "spectra" and "density" in the following refer to the integrals of $R_{\mathrm{radiative}}$ 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*
  - *output_spectra{ }*
  - *output_spectra{ emission_photons }*
  - *output_spectra{ emission_power }*
  - *output_spectra{ gain }*
  - *output_spectra{ decadic_gain }*
  - *output_spectra{ absorption_coeff }*
  - *output_spectra{ decadic_absorption_coeff }*
  - *output_spectra{ im_epsilon }*
  - *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{ emission_photons }*
  - *output_local_spectra{ emission_power }*
  - *output_local_spectra{ gain }*
  - *output_local_spectra{ decadic_gain }*

- *output_local_spectra{ absorption_coeff }*
- *output_local_spectra{ decadic_absorption_coeff }*
- *output_local_spectra{ im_epsilon }*
- *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

- using: optional within the scope
- 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

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: eV

---

energy_broadening_lorentzian

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: eV

---

output_spectra{ }

- using: required within the scope
- items: exactly 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{ emission_photons }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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{ gain }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

Gain spectra are outputted, only the positive part. The upper 30% of the spectra are cut off.

---

output_spectra{ decadic_gain }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Decadic gain spectra are outputted, only the positive part. The upper 30% of the spectra are cut off.

---

output_spectra{ absorption_coeff }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Decadic absorption spectra are outputted, both positive and negative parts. The upper 30% of the spectra are cut off.

---

output_spectra{ im_epsilon }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

The upper 30% of the spectra are cut off.

---

output_spectra{ spectra_over_energy }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

selected spectra are outputted over energy

---

output_spectra{ spectra_over_frequency }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

selected spectra are outputted over frequency

---

output_spectra{ spectra_over_wavenumber }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

selected spectra are outputted over wavenumber

---

output_spectra{ spectra_over_wavelength }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

selected spectra are outputted over wavelength

---

output_local_spectra{ }

- using: optional within the scope
- 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{ emission_photons }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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{ gain }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Decadic gain spectra are outputted, only the positive part. The upper 30% of the spectra are cut off.

---

output_local_spectra{ absorption_coeff }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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{ im_epsilon }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

The upper 30% of the spectra are cut off.

---

output_local_spectra{ spectra_over_energy }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

selected spectra are outputted over energy

---

output_local_spectra{ spectra_over_frequency }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

selected spectra are outputted over frequency

---

output_local_spectra{ spectra_over_wavenumber }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

selected spectra are outputted over wavenumber

---

output_local_spectra{ spectra_over_wavelegth }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

selected spectra are outputted over wavelegth

---

output_photon_density

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output emitted photon density in $\mathrm{cm}^{-3}\mathrm{s}^{-1}$ to *emitted_photon_density.dat*

---

output_power_density

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output emitted power density in $\mathrm{W}/\mathrm{cm}^{3}$ to *emitted_power_density.dat*

optics{ quantum_spectra{ } }

- using: optional within the scope
- 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., *quantum{ region{ Gamma{} } }* and *quantum{ region{ HH{} } }*) or between a 1-band model and 6-band model (e.g., *quantum{ region{ Gamma{} } }* and *quantum{ region{ kp_6band{ } } }*). Computations within single models (e.g., only within *quantum{ region{ kp_8band{ } } }*, only within *quantum{ region{ 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*
  - *energy_threshold*
  - *transition_threshold*
  - *occupation_threshold*
  - *occupation_ignore*
  - *occupation_zero_fermilevel*
  - *occupation_interpolate_invfermi*
  - *classify_states*
  - *classification_threshold*
  - *excitons{ }*
  - *excitons{ num_exciton_levels }*
  - *excitons{ coulomb_enhancement }*
  - *spontaneous_emission*
  - *output_energies*

- *output_occupations*
- *output_transitions*
- *output_spinor_components*
- *output_spectra{ }*
- *output_spectra{ output_components }*
- *output_spectra{ absorption_coeff }*
- *output_spectra{ decadic_absorption_coeff }*
- *output_spectra{ gain }*
- *output_spectra{ decadic_gain }*
- *output_spectra{ emission }*
- *output_spectra{ spectra_over_energy }*
- *output_spectra{ spectra_over_frequency }*
- *output_spectra{ spectra_over_wavelength }*
- *output_spectra{ spectra_over_wavenumber }*
- *output_spectra{ photon_spectra }*
- *output_spectra{ power_spectra }*
- *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{ use_for_absorption }*
- *kramers_kronig{ use_for_emission }*
- *k_integration{ }*
- *k_integration{ relative_size }*
- *k_integration{ num_points }*
- *k_integration{ num_integrationpoints }*

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

- using: required within the scope
- type: character string

The name of already defined region in *quantum{ region{ } }* for which optical generation should be calculated. Multiple numerical parameters are inherited after the definitions in the *quantum{ region{ } }* referred to.

spin_align

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

Compute optical transitions dominating in interband transitions, typically conduction band to valence band transitions.

intraband

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

Compute optical transitions dominating in intraband transitions, typically conduction band to conduction band transitions.

---

interband_approximation

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If `yes` then transitions within valence bands are included according to applied classification.

---

enable_electron_hole

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If `yes` then transitions between conduction and valence bands are included according to applied classification.

---

enable_electron_electron

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If `yes` then transitions within conduction bands are included according to applied classification.

---

use_kp8_EP

- using: optional within the scope
- type: choice
- choices: `yes`; `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.

---

energy_threshold

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- unit: eV
- default: `1e-6`

Only transitions between states with at least this energy difference are regarded when computing optical spectra.

---

transition_threshold

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- unit: eV
- default: `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

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- unit: –
- default: `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: choice
- choices: `yes`; `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 $k_{\parallel}$ space integration.

Warning: This feature is under development.

---

classify_states

- using: optional within the scope
- type: choice
- choices: `yes`; `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

- using: optional within the scope
- type: real number
- values: no constraints
- unit: eV
- default: `0.0`

A parameter shifting the reference energy for the classification of the states.

---

excitons{ }

- using: optional within the scope
- items: maximum 1

Include excitonic effects.

Attention: Excitons are implemented only for 1D simulations.

---

excitons{ num_exciton_levels }

- using: optional within the scope
- type: integer
- values: `{1, ..., 10}`
- unit: –
- default: `1`

Number of exciton levels included in the model.

---

excitons{ coulomb_enhancement }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes`, then the Coulomb enhancement factor, also known as the Sommerfeld factor, is taken into account.

---

spontaneous_emission

- using: optional within the scope
- type: choice
- choices: `yes`; `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.)

---

output_energies

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output energy dispersion for every transition.

---

output_occupations

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output occupation dispersion for every transition.

---

output_transitions

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output transition strength for every transition.

---

output_spinor_components

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output the spinor components for each state at each $k_{\parallel}$ 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{ }

- using: required within the scope
- items: exactly 1

Control of optical spectra output

---

output_spectra{ output_components }

- using: optional within the scope
- type: integer
- values: `{0, 1, 2}`
- unit: –
- default: `0`

If its value is set different from zero, this attribute generates an output of state-to-state spectral components of any type of spectra triggered by the keywords *output_spectra{ absorption_coeff }*, *output_spectra{ decadic_absorption_coeff }*, *output_spectra{ gain }*, *output_spectra{ decadic_gain }*, and *output_spectra{ emission }*. If set to 1 then components with vanishing or nearly vanishing values are omitted in the output. If set to 2 then all components are outputted.

Warning: Setthing this attribute to 2 may lead to a big number of files being written.

---

output_spectra{ absorption_coeff }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes`, then the optical absorption coefficient expressed in cm^{-1} is outputted.

---

output_spectra{ decadic_absorption_coeff }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes`, then the optical gain coefficient expressed in cm^{-1} is outputted.

---

output_spectra{ decadic_gain }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then the optical gain coefficient expressed in $\mathrm{dB}/\mu\mathrm{m}$ is outputted.

---

output_spectra{ emission }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

If set to `yes`, then emission spectrum is outputted.

---

output_spectra{ spectra_over_energy }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `yes`

Output spectra with respect to the energy.

---

output_spectra{ spectra_over_frequency }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output spectra with respect to the frequency.

---

output_spectra{ spectra_over_wavelength }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output spectra with respect to the wavelength.

---

output_spectra{ spectra_over_wavenumber }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

Output spectra with respect to the wave number.

---

output_spectra{ photon_spectra }

- using: optional within the scope
- type: choice
- choices: `yes`; `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{ power_spectra }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then photon power spectrum is outputted with units W/cm^2.

---

polarization{ }

- using: required within the scope
- 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 }

- using: required within the scope
- type: character string

name attached to output files with computed spectra for the defined polarization

---

polarization{ re }

- using: optional within the scope
- type: vector of 3 real numbers
- values: no constraints
- unit: –
- default: `[0.0, 0.0, 0.0]`

real part of the polarization vector

---

polarization{ im }

- using: optional within the scope
- type: vector of 3 real numbers
- values: no constraints
- unit: –
- default: `[0.0, 0.0, 0.0]`

imaginary part of the polarization vector

---

refractive_index

- using: optional within the scope
- type: real number
- values: `(0.0, ...)`
- unit: –
- default: `substrate`

Specify constant refractive index for the simulation of the optical spectra.

---

normalization_volume

- using: optional within the scope
- type: real number
- values: `(0.0, ...)`
- unit: $\mathrm{nm}^{\mathrm{dimension}}$
- default: `related quantum region`

Specifies normalization volume for the optical spectra.

---

min_energy

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- unit: eV
- default: `0.0`

lower energy bound for optical spectra

---

max_energy

- using: optional within the scope
- type: real number
- values: `[1e-3, ...)`
- unit: eV
- default: `2.0`

upper energy bound for optical spectra

---

energy_resolution

- using: optional within the scope
- type: real number
- values: `[1e-6, ...)`
- unit: eV
- default: `1e-3`

Spacing between subsequent energy grid points.

---

energy_broadening_gaussian

- using: optional within the scope
- 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

- using: optional within the scope
- 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{ }

- using: optional within the scope
- 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 }

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- unit: eV
- default: `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 }

- using: optional within the scope
- type: real number
- values: `(0.0, ...)`
- unit: –
- default: `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 }

- using: optional within the scope
- type: real number
- values: `[0.0, ...)`
- unit: –
- default: `0.0`

If this attribute is set to non-zero value then delta-function multiplied by the value is added to ε_i at energy *kramers_kronig{ delta_position }* to be used in Kramers-Kronig transformation.

---

kramers_kronig{ delta_position }

- using: optional within the scope
- type: real number
- values: `(0.0, ...)`
- unit: eV

This parameter is defining position of the delta function added to ε_i.

---

kramers_kronig{ use_for_absorption }

- using: optional within the scope
- type: choice
- choices: `yes`; `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `no`
- default: `no`

If set to `yes`, then the computed refractive index is used to calculate emission. Otherwise, constant value is used.

---

k_integration{ }

- using: optional within the scope
- 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 }

- using: optional within the scope
- type: real number
- values: `[1e-3, 1.0]`
- unit: –
- default: `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 }

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, ..., 100}`
- unit: –
- default: `4`

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 }

- using: optional within the scope
- type: integer
- values: `{1, 2, 3, 4, ...}`
- unit: –
- default: `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 }

- using: optional within the scope
- type: choice
- choices: `yes`; `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.

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]                }
```

6.5.16 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.

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{}`, `mobility_simba{}`, `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{}`, `mobility_simba{}`, `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";
}


optical_absorption{
    name = "Ge";
    ...
} : {
    name = "Germanium";
}


optical_absorption{
    name = "GaAs";
    ...
}


optical_absorption{
    name = "InP";
    ...
}


optical_absorption{
    name = "GaN";
    ...
}
```

(continues on next page)

(continued from previous page)

```
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"


# 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)
```

(continues on next page)

(continued from previous page)

```
#
#    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␣
↪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)
#
```

(continues on next page)

(continued from previous page)

```
#    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)
#
#    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"
```

(continues on next page)

(continued from previous page)

```
# 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"



# 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"
```

(continues on next page)

(continued from previous page)

```
# 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"



# 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"
```

(continues on next page)

(continued from previous page)

```
# 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␣
↪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)
#
```

(continues on next page)

(continued from previous page)

```
#     name = "Lamp-Xenon"
```

Nested groups in database{ ... _zb{} } and database{ ... _wz{} }

Bands 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 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.

$$\mathrm{a_{c,a}} = \mathrm{a_2} = \mathrm{a_{2,Vurgaftman}} + \mathrm{D2}$$

$$\mathrm{a_{c,c}} = \mathrm{a_1} = \mathrm{a_{1,Vurgaftman}} + \mathrm{D1}$$

---

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, "$\mathbf{k} \cdot \mathbf{p}$ method for strained wurtzite semiconductors", Phys. Rev. B 54 (**4**), 2491 (1996)):

$$\mathrm{E_{v,av}} = (\mathrm{E_{hh}} + \mathrm{E_{lh}} + \mathrm{E_{ch}})/3 - 2/3 \cdot \mathrm{Delta_{cr}}$$

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:

$$\mathrm{VBO_{v,av}} = \mathrm{bandoffset_{material1}} - \mathrm{bandoffset_{material2}}$$

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 $\text{Delta}_{cr} = \text{Delta}_1$, 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 $\text{Delta}_2 = \text{Delta}_3 = 1/3\ \text{Delta}_{so}$.

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 $S_1 = S_{parallel}$ for 8-band $\mathbf{k} \cdot \mathbf{p}$

value
double

S2
electron effective mass parameter $S_2 = S_{perpendicular}$ for 8-band $\mathbf{k} \cdot \mathbf{p}$

value
double

E_P1
Kane's momentum matrix elements $E_{p1} = E_{p,\ parallel}$

value
double

E_P2
Kane's momentum matrix elements $E_{p2} = E_{p,perpendicular}$

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:

$$\mathrm{a(T)} = \mathrm{a_{300K}} + \mathrm{a\_expansion} \cdot (\mathrm{T} - 300\mathrm{K})$$

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^{nd} order piezoelectric constant

type
double

unit
C/m^2

B124 (optional)
2^{nd} order piezoelectric constant

type
double

unit
C/m^2

B156 (optional)
2^{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^{nd} order piezoelectric constant

type
double

unit
C/m^2

B312 (optional)
2^{nd} order piezoelectric constant

type
double

unit
C/m^2

B313 (optional)
2^{nd} order piezoelectric constant

type
double

unit
C/m^2

B333 (optional)
2^{nd} order piezoelectric constant

type
double

unit
C/m^2

B115 (optional)
2^{nd} order piezoelectric constant

type
double

unit
C/m^2

B125 (optional)
2^{nd} order piezoelectric constant

type
double

unit
C/m^2

B135 (optional)
2^{nd} order piezoelectric constant

type
double

unit
C/m^2

B344 (optional)
2^{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_simba{} } }*

Attention: database{ ... { mobility_simba{} } } is implemented only in *nextnano*[3]

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^n_1)

type
double

unit
cm^2 V^{-1} s^{-1}

pc ***(optional)***
reference doping concentration parameter (P^n_c)

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_{min1}^p)

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_{min}^n)

type
double

unit
$cm^2\ V^{-1}\ s^{-1}$

alm ***(Optional)***
reference mobility exponent (α_m^n)

type
double

unit
None

mud ***(Optional)***
reference mobility parameter (μ_d^n)

type
double

unit
$cm^2\ V^{-1}\ s^{-1}$

ald ***(Optional)***
reference mobility exponent (α_d^n)

type
double

unit
None

n0 ***(Optional)***
reference impurity parameter (N_0^n)

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 (6.2.6.5) and (6.2.6.6) 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 (6.2.6.5) and (6.2.6.6) 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

database{ . . . { mobility_simba{} } }

See *Low-field mobility models* for details on this model.

> **Attention:** this group is implemented only in *nextnano³*

electrons{} ***(optional)***

alphaN ***(optional)***
exponent for Caughey/Thomas model (α^n)

type
double

unit
None

Nref ***(optional)***
reference doping density (N^n_{ref})

type
double

unit
cm^{-3}

mumin ***(optional)***
minimum mobility (μ^n_{min})

type
double

unit
$cm^2\ V^{-1}\ s^{-1}$

muN ***(optional)***
reference mobility

type
double

unit
$cm^2\ V^{-1}\ s^{-1}$

gammaT ***(optional)***
exponent for temperature dependence (γ^n)

type
double

unit
None

Esat0 ***(optional)***
peak electric field (E^n_0); models: 1, 3, 5

type
double

unit
$V\ cm^{-1}$

Esatd ***(optional)***
temperature dependence parameter of peak electric field (d^n_E); models: 1, 3, 5

type
double

unit
V K^{-1} cm^{-1}

alphaE ***(optional)***
alpha parameter for electric field dependence (α^n); models: 1, 3, 5

type
double

unit
None

betaE ***(optional)***
beta parameter for electric field dependence (β^n); models: 1, 3, 5

type
double

unit
None

vsat0 ***(optional)***
saturation velocity (v_0^n); models: 2, 3, 4, 5

type
double

unit
cm s^{-1}

vsatd ***(optional)***
temperature dependence parameter of saturation velocity (d_v^n); models: 2, 3, 4, 5

type
double

unit
cm K^{-1} s^{-1}

kappa ***(optional)***
exponent kappa for electric field dependence (κ^n); models: 2, 3, 4, 5

type
double

unit
None

ETnormal ***(optional)***
perpendicular electric field (E_T^n)

type
double

unit
V cm^{-1}

holes{} ***(optional)***

alphaN ***(optional)***
exponent for Caughey/Thomas model (α^p)

type
double

unit
None

Nref ***(optional)***
reference doping density (N_{ref}^p)

type
double

unit
cm^{-3}

mumin ***(optional)***
minimum mobility (μ^p_{min})

type
double

unit
$cm^2\ V^{-1}\ s^{-1}$

muN ***(optional)***
reference mobility

type
double

unit
$cm^2\ V^{-1}\ s^{-1}$

gammaT ***(optional)***
exponent for temperature dependence (γ^p)

type
double

unit
None

Esat0 ***(optional)***
peak electric field (E^p_0); models: 1, 3, 5

type
double

unit
$V\ cm^{-1}$

Esatd ***(optional)***
temperature dependence parameter of peak electric field (d^p_E); models: 1, 3, 5

type
double

unit
$V\ K^{-1}\ cm^{-1}$

alphaE ***(optional)***
alpha parameter for electric field dependence (α^p); models: 1, 3, 5

type
double

unit
None

betaE ***(optional)***
beta parameter for electric field dependence (β^p); models: 1, 3, 5

type
double

unit
None

vsat0 ***(optional)***
saturation velocity (v_0^p); models: 2, 3, 4, 5

type
double

unit
cm s^{-1}

vsatd ***(optional)***
temperature dependence parameter of saturation velocity (d_v^p); models: 2, 3, 4, 5

type
double

unit
cm K^{-1} s^{-1}

kappa ***(optional)***
exponent kappa for electric field dependence (κ^p); models: 2, 3, 4, 5

type
double

unit
None

ETnormal ***(optional)***
perpendicular electric field (E_T^p)

type
double

unit
V cm^{-1}

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:

Parameters

- *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{ }

Structure

```
database{ *_zb{ mobility_haensch{ } } }
database{ *_wz{ mobility_haensch{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Description

Stores parameters for the *Hänsh model*.

---

mobility_haensch{ electrons{ } }

Structure

```
database{ *_zb{ mobility_haensch{ electrons{ } } }
database{ *_wz{ mobility_haensch{ electrons{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description

Stores parameters for the *Hänsh model* for electrons.

---

mobility_haensch{ electrons{ vsat } }

Structure

```
database{ *_zb{ mobility_haensch{ electrons{ vsat } } }
database{ *_wz{ mobility_haensch{ electrons{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *Hänsh model* for electrons.

---

mobility_haensch{ holes{ } }

Structure

```
database{ *_zb{ mobility_haensch{ holes{ } } }
database{ *_wz{ mobility_haensch{ holes{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description

Stores parameters for the *Hänsh model* for holes.

---

mobility_haensch{ holes{ vsat } }

Structure

```
database{ *_zb{ mobility_haensch{ holes{ vsat } } }
database{ *_wz{ mobility_haensch{ holes{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *Hänsh model* for holes.

---

mobility_canali{ }

Structure

```
database{ *_zb{ mobility_canali{ } } }
database{ *_wz{ mobility_canali{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Description

Stores parameters for the *extended Canali model*.

---

mobility_canali{ electrons{ } }

Structure

```
database{ *_zb{ mobility_canali{ electrons{ } } }
database{ *_wz{ mobility_canali{ electrons{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description
Stores parameters for the *extended Canali model* for electrons.

---

mobility_canali{ electrons{ vsat } }

Structure

```
database{ *_zb{ mobility_canali{ electrons{ vsat } } }
database{ *_wz{ mobility_canali{ electrons{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description
Saturation velocity v_{sat} for the *extended Canali model* for electrons.

---

mobility_canali{ electrons{ alpha } }

Structure

```
database{ *_zb{ mobility_canali{ electrons{ alpha } } }
database{ *_wz{ mobility_canali{ electrons{ alpha } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`

Description
Parameter α for the *extended Canali model* for electrons.

---

mobility_canali{ electrons{ beta } }

Structure

```
database{ *_zb{ mobility_canali{ electrons{ beta } } }
database{ *_wz{ mobility_canali{ electrons{ beta } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[1e-3, ...)`

Description

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{ } }

Structure

```
database{ *_zb{ mobility_canali{ holes{ } } }
database{ *_wz{ mobility_canali{ holes{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description

Stores parameters for the *extended Canali model* for holes.

---

mobility_canali{ holes{ vsat } }

Structure

```
database{ *_zb{ mobility_canali{ holes{ vsat } } }
database{ *_wz{ mobility_canali{ holes{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *extended Canali model* for holes.

---

mobility_canali{ holes{ alpha } }

Structure

```
database{ *_zb{ mobility_canali{ holes{ alpha } } }
database{ *_wz{ mobility_canali{ holes{ alpha } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`

Description

Parameter α for the *extended Canali model* for holes.

---

mobility_canali{ holes{ beta } }

Structure

```
database{ *_zb{ mobility_canali{ holes{ beta } } }
database{ *_wz{ mobility_canali{ holes{ beta } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[1e-3, ...)`

Description

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{ }

Structure

```
database{ *_zb{ mobility_transferred{ } } }
database{ *_wz{ mobility_transferred{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Description

Stores parameters for the *transferred electron model*.

---

mobility_transferred{ electrons{ } }

Structure

```
database{ *_zb{ mobility_transferred{ electrons{ } } }
database{ *_wz{ mobility_transferred{ electrons{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description

Stores parameters for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ vsat } }

Structure

```
database{ *_zb{ mobility_transferred{ electrons{ vsat } } }
database{ *_wz{ mobility_transferred{ electrons{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ alpha } }

Structure

```
database{ *_zb{ mobility_transferred{ electrons{ alpha } } }
database{ *_wz{ mobility_transferred{ electrons{ alpha } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[1e-3, ...)`
- default: `1.0`

Description

Parameter α for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ beta } }

Structure

```
database{ *_zb{ mobility_transferred{ electrons{ beta } } }
database{ *_wz{ mobility_transferred{ electrons{ beta } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[1.001, ...)`

Description

Parameter β for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ gamma } }

Structure

```
database{ *_zb{ mobility_transferred{ electrons{ gamma } } }
database{ *_wz{ mobility_transferred{ electrons{ gamma } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`
- default: `0.0`

Description
Parameter γ for the *transferred electron model* for electrons.

---

mobility_transferred{ electrons{ E0 } }

Structure

```
database{ *_zb{ mobility_transferred{ electrons{ E0 } } }
database{ *_wz{ mobility_transferred{ electrons{ E0 } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`
- default: `0.0`

Description
Parameter E_0 for the *transferred electron model* for electrons.

---

mobility_transferred{ holes{ } }

Structure

```
database{ *_zb{ mobility_transferred{ holes{ } } }
database{ *_wz{ mobility_transferred{ holes{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description
Stores parameters for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ vsat } }

Structure

```
database{ *_zb{ mobility_transferred{ holes{ vsat } } }
database{ *_wz{ mobility_transferred{ holes{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s

- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ alpha } }

Structure

```
database{ *_zb{ mobility_transferred{ holes{ alpha } } }
database{ *_wz{ mobility_transferred{ holes{ alpha } } }
```

Properties

- using: required within the scope
- type: real number
- unit: $-$
- values: `[1e-3, ...)`
- default: `1.0`

Description

Parameter α for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ beta } }

Structure

```
database{ *_zb{ mobility_transferred{ holes{ beta } } }
database{ *_wz{ mobility_transferred{ holes{ beta } } }
```

Properties

- using: required within the scope
- type: real number
- unit: $-$
- values: `[1.001, ...)`

Description

Parameter β for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ gamma } }

Structure

```
database{ *_zb{ mobility_transferred{ holes{ gamma } } }
database{ *_wz{ mobility_transferred{ holes{ gamma } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`
- default: `0.0`

Description

Parameter γ for the *transferred electron model* for holes.

---

mobility_transferred{ holes{ E0 } }

Structure

```
database{ *_zb{ mobility_transferred{ holes{ E0 } } }
database{ *_wz{ mobility_transferred{ holes{ E0 } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`
- default: `0.0`

Description

Parameter E_0 for the *transferred electron model* for holes.

---

mobility_eastman{ }

Structure

```
database{ *_zb{ mobility_eastman{ } } }
database{ *_wz{ mobility_eastman{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Description

Stores parameters for the *Eastman-Tiwari-Shur*.

---

mobility_eastman{ electrons{ } }

Structure

```
database{ *_zb{ mobility_eastman{ electrons{ } } }
database{ *_wz{ mobility_eastman{ electrons{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description
Stores parameters for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ electrons{ vsat } }

Structure

```
database{ *_zb{ mobility_eastman{ electrons{ vsat } } }
database{ *_wz{ mobility_eastman{ electrons{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description
Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ electrons{ alpha } }

Structure

```
database{ *_zb{ mobility_eastman{ electrons{ alpha } } }
database{ *_wz{ mobility_eastman{ electrons{ alpha } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`

Description
Parameter α for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ electrons{ beta } }

Structure

```
database{ *_zb{ mobility_eastman{ electrons{ beta } } }
database{ *_wz{ mobility_eastman{ electrons{ beta } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[1.0, ...)`

Description

Parameter β for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman{ holes{ } }

Structure

```
database{ *_zb{ mobility_eastman{ holes{ } } }
database{ *_wz{ mobility_eastman{ holes{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description

Stores parameters for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman{ holes{ vsat } }

Structure

```
database{ *_zb{ mobility_eastman{ holes{ vsat } } }
database{ *_wz{ mobility_eastman{ holes{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman{ holes{ alpha } }

Structure

```
database{ *_zb{ mobility_eastman{ holes{ alpha } } }
database{ *_wz{ mobility_eastman{ holes{ alpha } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[0.0, ...)`

Description

Parameter α for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman{ holes{ beta } }

Structure

```
database{ *_zb{ mobility_eastman{ holes{ beta } } }
database{ *_wz{ mobility_eastman{ holes{ beta } } }
```

Properties

- using: required within the scope
- type: real number
- unit: –
- values: `[1.0, ...)`

Description

Parameter β for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman4{ }

Structure

```
database{ *_zb{ mobility_eastman4{ } } }
database{ *_wz{ mobility_eastman4{ } } }
```

Properties

- using: optional within the scope
- items: maximum 1

Description

Stores alternative, observable, parameters for the *Eastman-Tiwari-Shur*.

---

mobility_eastman4{ electrons{ } }

Structure

```
database{ *_zb{ mobility_eastman4{ electrons{ } } }
database{ *_wz{ mobility_eastman4{ electrons{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description

Stores alternative, observable, parameters for the *Eastman-Tiwari-Shur* for electrons.

---

mobility_eastman4{ electrons{ vsat } }

Structure

```
database{ *_zb{ mobility_eastman4{ electrons{ vsat } } }
database{ *_wz{ mobility_eastman4{ electrons{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ v_mid } }

Structure

```
database{ *_zb{ mobility_eastman4{ electrons{ v_mid } } }
database{ *_wz{ mobility_eastman4{ electrons{ v_mid } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Velocity v_{mid} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ v_peak } }

Structure

```
database{ *_zb{ mobility_eastman4{ electrons{ v_peak } } }
database{ *_wz{ mobility_eastman4{ electrons{ v_peak } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Velocity v_{peak} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ E_mid } }

Structure

```
database{ *_zb{ mobility_eastman4{ electrons{ E_mid } } }
database{ *_wz{ mobility_eastman4{ electrons{ E_mid } } }
```

Properties

- using: required within the scope
- type: real number
- unit: V/cm
- values: `[1.0, ...)`

Description

Driving force E_{mid} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ electrons{ E_peak } }

Structure

```
database{ *_zb{ mobility_eastman4{ electrons{ E_peak } } }
database{ *_wz{ mobility_eastman4{ electrons{ E_peak } } }
```

Properties

- using: required within the scope
- type: real number
- unit: V/cm
- values: `[1.0, ...)`

Description

Driving force E_{peak} for the *Eastman-Tiwari-Shur* for electrons within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ } }

Structure

```
database{ *_zb{ mobility_eastman4{ holes{ } } }
database{ *_wz{ mobility_eastman4{ holes{ } } }
```

Properties

- using: required within the scope
- items: maximum 1

Description

Stores alternative, observable, parameters for the *Eastman-Tiwari-Shur* for holes.

---

mobility_eastman4{ holes{ vsat } }

Structure

```
database{ *_zb{ mobility_eastman4{ holes{ vsat } } }
database{ *_wz{ mobility_eastman4{ holes{ vsat } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s
- values: `[1.0, ...)`

Description

Saturation velocity v_{sat} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ v_mid } }

Structure

```
database{ *_zb{ mobility_eastman4{ holes{ v_mid } } }
database{ *_wz{ mobility_eastman4{ holes{ v_mid } } }
```

Properties

- using: required within the scope
- type: real number
- unit: cm/s

- values: `[1.0, ...)`

Description

Velocity v_{mid} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ v_peak } }

Structure

```
database{ *_zb{ mobility_eastman4{ holes{ v_peak } } }
database{ *_wz{ mobility_eastman4{ holes{ v_peak } } }
```

Properties

- using: required within the scope
- type: real number
- unit: $\mathrm{cm/s}$
- values: `[1.0, ...)`

Description

Velocity v_{peak} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ E_mid } }

Structure

```
database{ *_zb{ mobility_eastman4{ holes{ E_mid } } }
database{ *_wz{ mobility_eastman4{ holes{ E_mid } } }
```

Properties

- using: required within the scope
- type: real number
- unit: $\mathrm{V/cm}$
- values: `[1.0, ...)`

Description

Driving force E_{mid} for the *Eastman-Tiwari-Shur* for holes within the alternative, observable, set of parameters.

---

mobility_eastman4{ holes{ E_peak } }

Structure

```
database{ *_zb{ mobility_eastman4{ holes{ E_peak } } }
database{ *_wz{ mobility_eastman4{ holes{ E_peak } } }
```

Properties

- using: required within the scope
- type: real number
- unit: V/cm
- values: `[1.0, ...)`

Description

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
        }

                                        # 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)
```

(continues on next page)

(continued from previous page)

```
    }

}
```

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:

$$R_{SRH} = \frac{p \cdot n - n_i^2}{\tau_p(n + n_i) + \tau_n(p + p_i)},$$
$$\tau_{p/n} = \frac{\tau_{p0/n0}}{1 + \frac{N_D + N_A}{N_{n/p,ref}}},$$

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).

6.6 Input Syntax

Note: this site is under reconstruction

Syntax if input files for *nextnano++*, *nextnano.MSB* (incl. in *nextnano++*), and *nextnano.NEGF* are 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: *nextnano++*, *nextnano.MSB*, and *nextnano.NEGF*.

6.6.1 General

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.

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.

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.

6.6.2 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`.

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.

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.

6.6.3 Comments

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.
```

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.

6.6.4 Conditional Statements

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.

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.

6.6.5 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.

6.6.6 Operators and functions

Tables for number variables

The following fucntions 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 $\vert\,\vert$
`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	`||`	

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.

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.

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

6.6.7 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
$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)

6.6.8 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.

6.6.9 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 <...>.

6.6.10 Additional Examples and Remarks

E.g. you can define:

```
$pi = 4 * atan(1)
```

This will give `3.14159265358979323846264338327950 29`. You can also specify:

```
$pi = 3.14159265358979323846264338327950 29
```

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.

6.7 Simulation Output

Here, we will add soon more information on the content of the output file names.

6.7.1 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.

6.7.2 Error handling with log file

ERROR: erroneous format of file

If the following error occurs in your log file

```
00:00:00 Start importing files.
00:00:00 ERROR: erroneous format of file 'C:\...\imported_file.dat'
00:00:00 (nodes number of coordinate 1) !=
00:00:00 (lines number in file)
00:00:00 1001 != 1003
00:00:00 Terminating program !!
```

It means that you have defined values at some grid points twice inside the imported file. It this case The message says that there are 1001 nodes defined in the file but there is 1003 lines. Most likely, it means that two points are defined twice, or one point is defined three times.

Solution: Review your imported file for duplicated definitions. Each coordinate should appear only once.

6.7.3 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)

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)

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␣
```

(continues on next page)

(continued from previous page)

```
↪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
```

(continues on next page)

(continued from previous page)

```
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.

6.8 Command Line

Command line usage:

A general form is `nextnano++_Intel_64bit.exe [runmode] [options] filename1 [filename2 ...]`, where `filename1` is the input file you want to simulate.

An example:

```
"nextnano++_Intel_64bit.exe" --license "C:\My Documents\nextnano\
License\License_nnp.lic" --database "C:\Program Files\nextnano\
2023_02_19\nextnano++\Syntax\database_nnp.in" --threads 4
--outputdirectory "C:\My Documents\nextnano\Output\HEMT_1D_nnp"
--noautooutdir "C:\Program Files\nextnano\2023_02_19\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.

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 GmbH 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.
               = "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
```

(continues on next page)

(continued from previous page)

```
    }
}
```

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
}
```

6.9 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 cores and 16 threads:

1. If many small (e.g. 1D) runs are performed at a time:
   - max. number of simulations = number of CPU cores (not CPU threads!), i.e. 8 in the present case.
   - max. number of threads per simulation = 1
   - process priority: on a dedicate machine, or if you are not bothered by CPU load: use “normal” instead of “below normal”

1. If only individual runs are performed at a time, which optionally may be large (e.g. 3D):
   - max. number of simulations = 1
   - max. number of threads per simulation = number of CPU cores (not CPU threads!), i.e. 8 in the present case
   - process priority: on a dedicate machine, or if you are not bothered by CPU load: use “normal” instead of “below normal”

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.

6.10 Release Notes

6.10.1 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 } }`

6.10.2 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

6.10.3 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

6.10.4 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

6.10.5 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{ }`

6.10.6 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

6.10.7 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

---

6.10.8 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{}`

6.10.9 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`

6.10.10 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{ }`

6.10.11 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{ }`

6.10.12 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`

6.10.13 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.

CHAPTER

SEVEN

NEXTNANO[3]

The *nextnano*[3] tool is the predecessor of *nextnano++*.

7.1 Overview

Features

- includes group IV materials (Si, Ge, SiGe), all III-V materials, II-VI materials and its ternaries as well as lattice-matched quaternaries. The nitride and II-VI materials are available in the zinc blende and wurtzite crystal structure.
- flexible structures and geometries (1D, 2D and 3D)
- quantum mechanics based on single-band and multi-band $\mathbf{k} \cdot \mathbf{p}$ model within a finite differences grid
- includes 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), ballistic transport
- magnetic field

Not all keywords have been documented in the new documentation.

The old documentation for *nextnano*[3] is available here: https://www.nextnano.com/nextnano3/

7.2 Input Syntax

7.2.1 Macro features

The Macro feature of *nextnano*[3] is also described here: https://github.com/nextnano/FortranInputParser. The Fortran Input Parser source code is open-source.

Comment signs

Everything in the *same* line following after such a symbol is treated as a *comment*.

- `#`

  Note: This is the same comment sign as used in *nextnano++*.

- `!`
- `//`
- `/* This is a comment. */`

  The end tag `*/` is optional.

  Note: No line breaks are allowed for the comments `/* ... */`.

- < This is a comment. >

Note: However, the following are exceptions and are treated differently.

- #IF
- !IF
- #WHEN
- !WHEN

These are special statements for conditional lines (see below).

Note: XML tags can be used in the input file, .e.g. to provide additional meta information. They are ignored by *nextnano*[3] but could be used in e.g. *nextnanomat*. Everything that is between these brackets < ... > is replaced with blanks. This can be used to define XML tags such as:

```
<variables>
# comment
 %TEMPERATURE = 300.0                          # (DisplayUnit:K)
 %WIDTH       =  10.0                          # (DisplayUnit:nm)
 %HEIGHT      =  20.0  < unit = "nm" >         # (DisplayUnit:nm)
</variables>
<variables> %TEMPERATURE = 300.0 </variables> # (DisplayUnit:K)
```

Macro 1: Variables

The % symbol is used to define a variable such as %VARIABLE = that should be replaced with another string or value. Variables are case-sensitive.

A variable is only defined and initialized if the first character in a line starts with the % symbol.

This macro distinguishes between *strings* (default), i.e. simple string replacements where 1 and 1.0 are treated as a *string*) and *functions* (i.e. arithmetic operations if %FunctionParser = yes) where both 1 or 1.0 are treated as a *real* number. Both can be combined but it is important to understand the differences to avoid errors. For instance, for some *specifiers*, the input parser expects an *integer* number and not a *real* number. So real numbers have to be converted back to integers in some cases. This macro is supported by *nextnanomat*.

String feature

Find & Replace of variables

Example: %WIDTH = 10.0

The *nextnano*[3] tool supports a simple *Find & Replace* feature where *strings* in the input file can be replaced with another *string*.

How to use it

Just put something like the following lines anywhere into your input file. We recommend that you put it at the beginning of the input file for better readability.

```
#-------------------------------------------------------------
# By default, we do not evaluate functions.
# Everything is just a simple "Find & Replace String" operation.
#-------------------------------------------------------------
 %width            = 10.0            # quantum well width␣
```

(continues on next page)

(continued from previous page)

```
↪(DisplayUnit:nm)
 %length           = 10              # quantum well length␣
↪(DisplayUnit:nm)
 %size             = 10.0            # quantum well size (DisplayUnit:nm)
 %z_coordinates    = -21.0  101.0    # from z = -21 to z = 101␣
↪(DisplayUnit:nm)
 %well_material    = GaAs            # quantum well material: GaAs
 %barrier_material = "AlAs"          # quantum barrier material: AlAs
 %zmin             = -10.0           # zmin = - 10 nm (DisplayUnit:nm)
 %zmax             =  10.0           # zmax =   10 nm (DisplayUnit:nm)

# derived variable: quantum_region = zmin zmax = -10.0 10.0
 %quantum_region   = %zmin  %zmax    # (DisplayUnit:nm)

 %hkl_z_direction  = 0 1 1           # [0 1 1] growth direction

# The whole string including blanks after the first '=' sign is used.
 %Region1          = region-number = 1   base-geometry = line   region-
↪priority = 1
```

The `%` sign indicates that you define a *variable*. When the input file is processed, all occurences of `%width`, `%z-coordinates`, `%well_material`, ... are replaced by the according strings.

A *comment* sign (e.g. `#`, `!`) is allowed in this line. Blanks are allowed within the string which is useful for reading in arrays of numbers. Almost everything is allowed.

Restrictions

- The variables and their definitions are *case-sensitive*.
- A variable name must not contain a `-` sign, e.g.

```
%cb-mass = 0.067  # (DisplayUnit:m0)
```

  is not allowed. Use underscores `_` instead.

- In one line, only *one* variable can be initialized.

Function feature

Find, Evaluate & Replace

Example: `%WIDTH = %LENGTH / 2.0`

The statement `%FunctionParser = yes` switches the function parser on (default: off).

The following operators

round arithmetic brackets	`( )`
power (exponentiation)	`^`
arithmetic multiplication, division	`* /`
arithmetic plus and minus	`+ -`

and functions are supported:

`sqrt()`	square root $\sqrt{}$
`exp()`	exponential function $\exp(\,)$
`log()`	natural logarithm $\log$
`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 $\tanh(\,)$
`abs()`	absolute value $|\,|$

Example

```
#-------------------------------------------------------------------------
# If '%FunctionParser = yes', then from now on we evaluate all functions.
# If '%FunctionParser = no' (or not present), no functions are evaluated.
# "Find & Evaluate & Replace String" operation.
#-------------------------------------------------------------------------

 %FunctionParser = yes # This is an important line!

 %a         = 4.0               # a = 4.0
 %b         = 3                 # b = 3
 %function1 = SQRT(%a) - %b / 2 # Evaluate: sqrt(a) - b / 2 = 2.0 - 1.5 = 0.
↪5
 ...
 %xalloy = %function1           # Insert evaluated value for %function1.

# Strings containing blanks are also possible.
# They must be in quotation marks "" or apostrophes ''.
 %string1 = "GaAs"
 %string2 = "GaAs"
 %string3 = "zero-potential = yes"

# Logical variables that are .TRUE. or .FALSE. can be defined
# and used in conditional #IF statements.
 %Logical1 = .TRUE.
 %Logical2 = .FALSE.
```

We have to be careful about the value of a variable. Should it be regarded as a *string* (default), or as a *function* (`%FunctionParser = yes`)?

```
#-------------------------------------------------------------------------
↪-
# Treat variables as strings (default):
# -------------------------------------
#   e.g. %variable1 = %zmin %zmax # '%zmin %zmax' is treated as a string
#        %variable2 = 2.5e0       #       '2.5e0' is treated as a string
#        %variable3 = 2.5         #       '2.5' is treated as a string␣
↪without exponent 'e0'
#-------------------------------------------------------------------------
↪-
```

(continues on next page)

(continued from previous page)

```
#%radius       = 5e0              # quantum dot radius = 5 nm␣
↪(DisplayUnit:nm)
 %grid_spacing = 0.5              # grid spacing             ␣
↪(DisplayUnit:nm)

#-------------------------------------------------------------------------
↪-
# From now on, evaluate all functions:
# ------------------------------------
#   e.g. %variable1 = %zmin %zmax # '%zmin %zmax' is treated as a formula
#                                 # (which will produce an error in this␣
↪case)
#        %variable2 = 2.5         #       '2.5' is converted to a real␣
↪number including exponent 'e0'
#-------------------------------------------------------------------------
↪-
 %FunctionParser = yes            # use function parser
 %radius         = 5              # quantum dot radius = 5 nm    ␣
↪(DisplayUnit:nm)
 %boundary       = %radius + 1    # device_width/2 = 5 nm + 1 nm␣
↪(DisplayUnit:nm)
```

Special treatment of integers

The parser distinguishes between integers and real numbers. Thus we have to be careful when using functions. This means that we have to convert the real values back to integers. Integers will be rounded down up >= .5 or rounded down if < .5, i.e. they will be rounded to the nearest integer.

- Option a) (recommended)

```
#---------------------------------------------
# If you need an integer, then use INT(...).
# %nodes contains an integer value.
#---------------------------------------------
%nodes = INT( %width / %grid_spacing - 1 )
```

- Option b) (deprecated)

```
#---------------------------------------------
# If you need an integer, then simply call the
# variable '%INT(<name>)'
# %INT(nodes) contains an integer value.
#---------------------------------------------
%INT(nodes) = %width / %grid_spacing - 1
```

- Option c) (deprecated)

```
#---------------------------------------------
# If you need an integer, then simply call the
# variable '%INT(<name>)'
# %nodes is real number.
# %INT(nodes) contains an integer value.
#---------------------------------------------
%nodes = %width / %grid_spacing - 1
%INT(nodes) = %nodes
```

Integer numbers can be defined using quotation marks "``´ ´``" or INT():

```
%NumberOfEigenvalues = "10"     # 10 eigenvalues
%NumberOfEigenvalues = INT(10)  # 10 eigenvalues
```

Integer arrays can be defined using quotation marks "`` ´ ``":

```
%hkl_z_direction = "0 1 1"   # [0 1 1] growth direction
```

conditional lines

`#IF`, `!IF`, `#WHEN` or `!WHEN` statements can also be used to define conditional lines, i.e. variables can be used to define conditional lines.

`#IF`, `!IF`, `#WHEN` or `!WHEN` are not *case-sensitive*, i.e.

`#if`, `!if`, `#when` or `!when` also work.

```
#IF %TemperatureDependentBandGap  varshni-parameters-on = yes
```

Such a statement has the following meaning:

- If the value of `%TemperatureDependentBandGap` is either `.TRUE.` or `1` or any other *nonzero value*, then the statement is *included* in the input file.
- If the value of `%TemperatureDependentBandGap` is either `.FALSE.` or `0` or *undefined*, then the statement is simply *ignored* as if it were a comment.

In this example, the text is always commented out, unless `%IncludeHoles` is defined with a value `%IncludeHoles /= 0` or `/= .FALSE.`.

```
    %IncludeHoles = .TRUE.   # or %IncludeHoles = 1

!IF %IncludeHoles $quantum-model-holes
!IF %IncludeHoles  model-number                   = 1
!IF %IncludeHoles  model-name                     = effective-mass
!IF %IncludeHoles  cluster-numbers                = 1
!IF %IncludeHoles  valence-band-numbers           = 1 2 3
!IF %IncludeHoles  number-of-eigenvalues-per-band = 10
!IF %IncludeHoles $end_quantum-model-holes
```

In the following, the evaluated value of `%IncludeElectrons` is `0`, therefore the three lines starting with `!IF %IncludeElectrons` are treated as comments.

```
    %IncludeHoles     = 1
    %IncludeElectrons = 1 - %IncludeHoles   # evaluated to: 1 - 1 = 0
!IF %IncludeElectrons $quantum-model-electrons  # treated as comment
!IF %IncludeElectrons  model-number = 1         # treated as comment
!IF %IncludeElectrons  ...                      # treated as comment
```

If the variable definition

```
%IncludeHoles = .TRUE.
```

appears after the line

```
%FunctionParser = yes
```

quotation marks " or apostrophes ' must be used for the logical strings `.TRUE./.FALSE.` to indicate that they have to be treated as strings:

```
%IncludeHoles = ".TRUE."  # or
%IncludeHoles = '.TRUE.'
```

Variable replacement

- Once the input file has been processed, the input file (`*.in`) is copied to the output folder. Additionally, this output folder contains a file called `*.in.mo_macro` which contains the input file with all variables having been replaced with their values.
- All variables and their values that are used in a simulation are written to the output folder into a file called `variables_input.txt`.

Macro 2: Namelist (deprecated)

Alternatively, namelists are supported for macros. The input syntax for this macro is implemented as a `NAMELIST` which is a standard Fortran feature.

The *nextnano*[3] tool supports a simple *Find & Replace* feature where strings in the input file can be replaced with another string. The first lines of the input file can contain the following additional entries.

```
&macro filename = 'macro.in' /
```

Here, `macro.in` is an ASCII file containing the following content:

```
&macro  Find_and_Replace_1 = '$WIDTH' ,'0d0 10.0d0'
        Find_and_Replace_2 = 'x-nodes = 9','x-nodes = 29'
        Find_and_Replace_3 = 'GaAs' ,'InP'
/
```

These three lines have the following meaning:

- Find `$WIDTH` and replace it with `0d0 10.0d0`.
- Find `x-nodes = 9` and replace it with `x-nodes = 29`.
- Find `GaAs` and replace it with `InP`.

Dowload example file: macro.in

Example 1

```
!
↪*****************************************************************************!
↪
! Macro: 'Find & Replace'
!
↪*****************************************************************************!
↪
!
! A macro can be used to simply search for a specific string and to replace
! it with another string.
!
! Example: Replace the material 'GaAs' with 'InAs'.
!       material-name = GaAs
!   ==> material-name = InAs
!
! Replace the quantum well width '$WIDTH' with '10d0':
!       x-coordinates = -$WIDTH $WIDTH
!   ==> x-coordinates = -10d0 10d0
!
! Replace the string '2*$WIDTH' with '20d0':     ! (Mathematical operations␣
```

(continues on next page)

(continued from previous page)

```
↪are not supported yet. They are treated as strings.)
!       x-coordinates = 0d0 2*$WIDTH
!   ==> x-coordinates = 0d0 20d0
!
! Replace the string '$WIDTH+$SIZE' with '30d0': ! (Mathematical operations␣
↪are not supported yet. They are treated as strings.)
!       x-coordinates = 0d0 $WIDTH+$SIZE
!   ==> x-coordinates = 0d0 30d0
!
! Replace the quantum well width '$WIDTH' with '0d0 10d0':
!       x-coordinates = $WIDTH
!   ==> x-coordinates = 0d0 10d0
!
! Simulate along x, y or z direction:
!       Find_and_Replace_1 = '$DIRECTION' ,'x'      ! simulation along x␣
↪direction
!       Find_and_Replace_2 = '$DOMAINTYPE','1 0 0'  ! simulation along x␣
↪direction
!
!       Find_and_Replace_1 = '$DIRECTION' ,'y'      ! simulation along y␣
↪direction
!       Find_and_Replace_2 = '$DOMAINTYPE','0 1 0'  ! simulation along y␣
↪direction
!
! a) The macro can be defined in the input file: filename = '' (or)
! b) The macro can be read in from another file: filename = 'macro.in'
!
!-------------------------------------------------------------------------
↪!
&macro  filename            = 'macro.in'                    ! (optional)␣
↪macro file to be read in, use '' for no macro input file
        Find_and_Replace_1  = '$WIDTH' ,'0d0 10.0d0'        ! is ignored if␣
↪macro file is read in
        Find_and_Replace_2  = 'x-nodes = 9','x-nodes = 29'  ! is ignored if␣
↪macro file is read in
        Find_and_Replace_3  = 'GaAs' ,'InP'                 ! is ignored if␣
↪macro file is read in
/
```

Example 2

```
&macro  filename            = ''                            ! (optional)␣
↪macro file to be read in, use '' for no macro input file
        Find_and_Replace_1  = '$WIDTH' ,'0d0 10.0d0'        ! find string,␣
↪replace string
        Find_and_Replace_2  = 'x-nodes = 9','x-nodes = 29'  ! find string,␣
↪replace string
        Find_and_Replace_3  = 'GaAs' ,'InP'                 ! find string,␣
↪replace string
/
```

Example 3

```
&macro  Find_and_Replace_1  = '$WIDTH' ,'0d0 10.0d0'        ! find string,␣
↪replace string
        Find_and_Replace_2  = 'x-nodes = 9','x-nodes = 29'  ! find string,␣
↪replace string
```

(continues on next page)

(continued from previous page)

```
        Find_and_Replace_3  = 'GaAs' ,'InP'                  ! find string,␣
↪replace string
        ...
        Find_and_Replace_40 = 'GaAs' ,'InP'                  ! find string,␣
↪replace string
/
```

Currently up to 40 *Find & Replace* definitions can be specified.

7.2.2 The input file keywords

Here you can find detailed descriptions about the keywords and specifiers for the input file. The general definition file is `keywords.val` where valid keywords are defined.

Keywords for setting up the device geometry, material and general settings

- $input_filename
- $global-parameters (temperature)
- $simulation-dimension
- $simulation-flow-control
- $strain-minimization-model
- $electric-field
- $magnetic-field
- $numeric-control

Keywords for setting up the device geometry

- $domain-coordinates
- $regions
- $region-cluster
- $grid-specification

Keywords for setting up material properties

- $material
- $binary-zb-default
- $binary-wz-default
- $ternary-zb-default
- $ternary-wz-default
- $alloy-function
- $import-data-on-material-grid

Keywords for doping

- $doping-function
- $impurity-parameters
- $material-interfaces
- $interface-states

Keywords for output

- $global-settings (output directory)
- $output-bandstructure
- $output-densities
- $output-strain
- $output-1-band-schroedinger
- $output-kp-data
- $output-current-data
- $output-raw-data
- $output-grid
- $output-geometry
- $output-material
- $output-file-format
- $output-section

Keywords for quantum calculations

- $quantum-regions
- $quantum-cluster
- $quantum-model-electrons
- $quantum-model-holes
- $quantum-bound-states
- $quantum-dot-layer-density
- $optical-absorption
- $tighten

Keywords for current calculations

- $poisson-boundary-conditions
- $voltage-sweep
- $current-regions
- $current-cluster
- $current-models
- $simple-drift-models
- $Monte-Carlo (not available yet)

Keywords for recombination

- $Auger-recombination
- $direct-recombination (radiative)
- $SRH-recombination (Shockley-Read-Hall)
- $quantumstate-recombination-rates

Keywords for mobility models

- $mobility-model-constant
- $mobility-model-arora
- $mobility-model-dar (Darwish)

- $mobility-model-lom (Lombardi)
- $mobility-model-masetti
- $mobility-model-minimos
- $mobility-model-simba

Keywords for electrolytes

- $electrolyte
- $electrolyte-ion-content
- $buffer-solutions
- $buffer-constant-A(T)

Keywords for NEGF

- $CBR-current (ballistic)
- $global-parameters-NEGF
- $scattering-mechanisms
- $contact-type
- $damping-parameters
- $roughness-profile
- $left-contact-potential-profile
- $right-contact-potential-profile
- $potential-profile (deprecated)
- $mass-profile (deprecated)
- $alloy-profile (deprecated)
- $nonparabolicity-profile (deprecated)
- $dielectric-profile (deprecated)
- $doping-function-NEGF (deprecated)
- $NEGF-spintronics (for NEGF spintransport code)

Other keywords

- $warnings

The Fortran Input Parser source code of *nextnano*[3] is open-source: https://github.com/nextnano/FortranInputParser

7.2.3 Keywords

$simulation-dimension

Simulation dimension and orientation

Simulation coordinate system

In general, the underlying coordinate system for the simulation is a three-dimensional cartesian coordinate system (x,y,z). Dependent on the type of simulation, i.e. whether a one-, two- or three-dimensional simulation has to be performed, the corresponding number of axes has to be selected out of this triplet of coordinate directions. For these specifications, the keyword `$simulation-dimension` together with its specifiers `dimension` and `orientation` have to be used in the input file.

```
$simulation-dimension                       required
 dimension                 integer          required
 orientation               integer_array    required
$end_simulation-dimension                   required
```

Specification of the simulation domain dimension and its orientation

dimension

type
integer

presence
required

options
1 (1D)

2 (2D)

3 (3D)

Specify here if a 1D, 2D or 3D simulation is performed.

orientation

type
integer array of dimension 3

presence
required

options
for 1D simulation: `1 0 0`, `0 1 0`, `0 0 1`

for 2D simulation: `1 1 0`, `1 0 1`, `0 1 1`

for 3D simulation: `1 1 1`

The specifier `orientation` defines the orientation of the simulation domain relative to the (x,y,z) coordinate system. A three-dimensional array containing `0` and/or `1` is expected. The relevant axes in this array are selected using `1`.

Specify here if the simulation is performed along a direction, in a plane or in a volume.

- simulation direction (1D: x, y or z axis)
- simulation plane (2D: (x,y), (x,z) or (y,z) plane)
- simulation volume (3D)

Example 1 (1D)

```
!----------------------------!
$simulation-dimension        !
 dimension   = 1             ! 1D simulation
 orientation = 0 0 1         ! z axis
$end_simulation-dimension    !
!----------------------------!
```

Example 2 (2D)

```
!----------------------------!
$simulation-dimension        !
 dimension   = 2             ! 2D simulation
 orientation = 1 1 0         ! (x,y) plane
```

(continues on next page)

(continued from previous page)

```
$end_simulation-dimension    !
!----------------------------!
```

Example 3 (3D)

```
!----------------------------!
$simulation-dimension        !
 dimension   = 3             ! 3D simulation
 orientation = 1 1 1         !
$end_simulation-dimension    !
!----------------------------!
```

$regions

General specifiers

To built up a geometry, there are - **dependent on the dimension** of the simulation to be performed - various basic geometry elements available. These geometry elements are specified within the keyword `$regions` and can be clustered to a bigger object later on.

```
$regions                                    optional
    region-number          integer          required
    region-priority        integer          required

    base-geometry          character        required

    x-coordinates          double_array     optional
    y-coordinates          double_array     optional
    z-coordinates          double_array     optional

    center                 double_array     optional
    radius                 double           optional

    base-coordinates       double_array     optional
    top-coordinates        double_array     optional
    corner-coordinates     double_array     optional

    semi-ellipse-base      double_array     optional
    semi-ellipse-top       double_array     optional

$end_regions                                optional
```

region-number

type
integer (≥ 1)

presence
required

example
`4`

An integer number to refer to geometry element. Numbering must be unique. All region numbers together must form a dense set `1,2,3,....,maxnumber`.

region-priority

type
integer (≥ 1)

presence
required

example
`2`

A positive integer to set overwriting priority. In case of overlapping regions, the region with higher priority (= higher numerical value) overwrites the region with lower priority.

base-geometry

type
character

presence
required

example
`rectangle`

type of geometry object.

- 3D: `cuboid`, `shpere`, ...
- 2D: `rectangle`, `circle`, ...
- 1D: `line`

These are specified in detail later.

x-coordinates

type
double_array

presence
optional

unit
[nm]

example
`-10.0 10.0`

This is for `cuboid`, `rectangle`, `line`. The values represent `xmin`, `xmax` from left.

y-coordinate

type
double_array

presence
optional

unit
[nm]

example
`-10.0 10.0`

This is for `cuboid`, `rectangle`, `line`. The values represent `ymin`, `ymax` from left.

z-coordinates

type
double_array

presence
optional

unit
[nm]

example
`-10.0 10.0`

This is for `cuboid`, `rectangle`, `line`. The values represent `zmin`, `zmax` from left.

center

type
double_array

presence
optional

unit
[nm]

example
`2.0`

This is for `circle` (2D), `sphere` (3D).

radius

type
double

unit
[nm]

presence
optional

example
`3.0`

This is for `circle` (2D), `sphere` (3D).

base-coordinates

type
double_array

presence
optional

unit
[nm]

example
`10.0 20.0 10.0 20.0 15.0 15.0`

This is for `obelisk`, `cone`, `truncated-cone`, `trapezoid`. The values represent `xmin xmax ymin ymax zmin zmax` of obelisk (cone) base plane. (one pair must be equal)

top-coordinates

type
double_array

presence
optional

unit
[nm]

example
`10.0 20.0 10.0 20.0 30.0 30.0`

This is for `obelisk`, `cone`, `truncated-cone`, `trapezoid`, `semiellipsoid`: The values represents `xmin xmax ymin ymax zmin zmax` of obelisk (cone) base plane (one pair must be equal) or `x y z` for semiellipsoid.

corner-coordinates

type
double_array

presence
optional

unit
[nm]

example
`10.0 10.0 10.0 10.0 30.0 10.0 20.0 20.0 10.0`

Triangle corner coordinates, interpreted via orientation and similar for polygon. For triangular prism: `x1 y1 z1 x2 y2 z2 x3 y3 z3`

semi-ellipse-base

type
double_array

presence
optional

unit
[nm]

example
`60.0 120.0 40.0 40.0`

For semiellipse: base line ellipse. These are two pairs of `xmin`, `xmax` `ymin`, `ymax` `zmin`, `zmax` - one pair equal numbers

semi-ellipse-top

type
double_array

presence
optional

unit
[nm]

example
`100.0 20.0`

For semiellipse: top coordinate pair e.g. for coordinate orientation (101) -> (x,z)

Details of specification

- *1-dimensional objects (only possible in 1D simulations)*
  - *line*
- *2-dimensional objects*
  - *rectangle*
  - *circle*
  - *triangle*
  - *trapezoid*
  - *polygon*
  - regular-polygon (not implemented yet but could be added -> use *polygon* instead)
  - hexagon (not implemented yet but could be added -> use *polygon* instead)
  - *semiellipse*
- ***3-dimensional objects***
  - *cuboid*
  - *sphere*
  - cylinder (not implemented yet but could be added -> use *truncated-cone* instead)
  - *obelisk*
  - *hexagonal-obelisk*
  - *cone*
  - *truncated-cone*
  - *semiellipsoid*
  - regular-prism (not implemented yet but could be added)
  - hexagonal-prism (not implemented yet but could be added -> use *hexagonal-obelisk* instead)
  - *triangular-prism*
  - polygonal-prism (not implemented yet but could be added)
  - pyramid (not implemented yet but could be added -> use *obelisk* instead)
  - regular_pyramid (not implemented yet but could be added -> use *obelisk* instead)
  - hexagonal_pyramid (not implemented yet but could be added -> use *hexagonal-obelisk* instead)
  - polygonal_pyramid (not implemented yet but could be added)

1-dimensional objects (only possible in 1D simulations)

line

```
$regions
    region-number      = 1
    base-geometry      = line
    region-priority    = 1
    x-coordinates      = xmin  xmax
$end_regions
```

- Chosen coordinates must be consistent with simulation orientation.

2-dimensional objects

rectangle

```
$regions
    region-number     = 1
    base-geometry     = rectangle
    region-priority   = 1
    x-coordinates     = xmin xmax
    y-coordinates     = ymin ymax
$end_regions
```

- Two pairs of delimiting coordinates are required. Whether these have to be `x-coordinates` and `y-coordinates` as in the example above, or another combination (e.g. x, z) depends on the simulation orientation which is specified already.

circle

```
$regions
    region-number     = 1
    base-geometry     = circle
    region-priority   = 1
    center            = x     y
    radius            = r
$end_regions
```

- The `circle` is defined by a center with coordinates (x,y) and a radius r.

triangle (can also be specified using polygon)

```
$regions
    region-number      = 1
    base-geometry      = triangle
    region-priority    = 1
    corner-coordinates = x1 y1    x2 y2    x3 y3
$end_regions
```

- The corner coordinates refer to the plane, specified by the simulation orientation.

polygon

```
$regions
    region-number      = 1
    base-geometry      = polygon
    region-priority    = 1
    corner-coordinates = x1 y1    x2 y2    x3 y3    x4 y4    x5 y5    ...
$end_regions
```

- The corner coordinates of a polygon refer to the plane, specified by the simulation orientation.
- The vertices may be listed clockwise or anticlockwise. The first point could be repeated, i.e. the first point and the last point may be identical but this is not required.
- The input polygon may be a compound polygon consisting of several separate subpolygons.

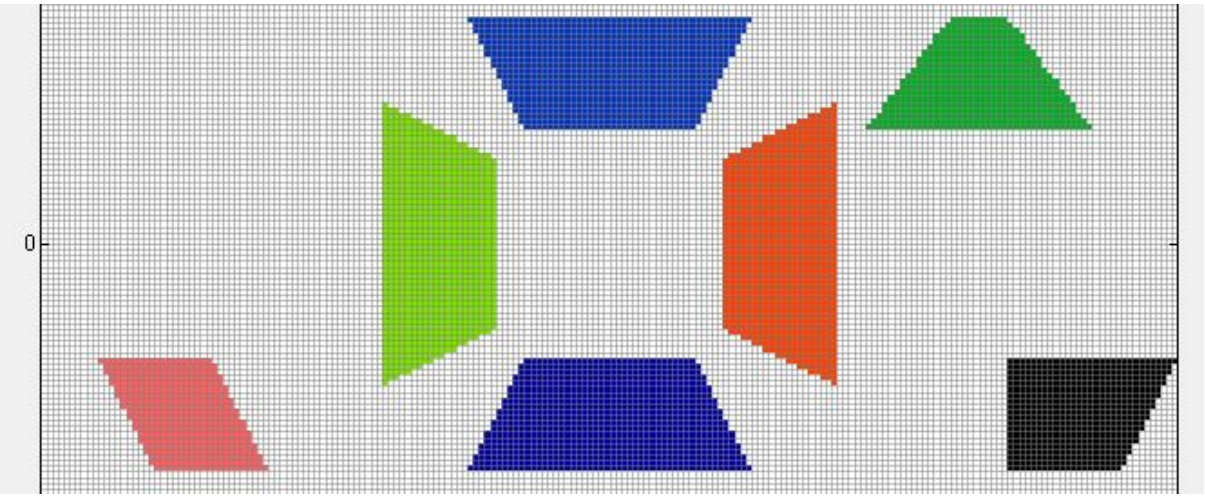
If so, the first vertex of each subpolygon must be repeated.

trapezoid (can also be specified using polygon)

```
$regions
    region-number       = 1
    base-geometry       = trapezoid
    region-priority     = 1
    top-coordinates     = -15.0 15.0    -20.0 -20.0     ! xmin xmax  ymin ymax
    base-coordinates    = -25.0 25.0    -40.0 -40.0     ! xmin xmax  ymin ymax
$end_regions
```

- For each, `base-coordinates` and `top-coordinates`, 2 values must be equal (meaning **only** 2 values can be equal), e.g. ymin=ymax=-40.0. Then one boundary of the object is a line at y=-40 nm. Base and top planes of the trapezoid must be in a coordinate plane. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. ymin ymax) has identical values (ymin = ymax). This also implies that base and top lines are parallel to either the x or the y axis.

The example given above shows the blue trapezoid at the bottom of this figure. The top line extends in x direction from -15 to 15 nm. The top line has a constant y coordinate of y = -20 nm The base line extends in x direction from -25 to 25 nm. The base line has a constant y coordinate of y = -40 nm.



semiellipse

```
$regions
    region-number       = 1
    base-geometry       = semiellipse
    region-priority     = 1
    semi-ellipse-base   = xmin xmax  ymin ymax
    semi-ellipse-top    =    x1         y1
$end_regions
```

- `semi-ellipse-base`: Here, 2 values must be equal (meaning only 2 values can be equal), e.g. ymin=ymax=40.0. Then one boundary of the object is a line at y=40 nm. Base plane of the semiellipse must be in a coordinate plane. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. ymin ymax) has identical values (ymin = ymax).
- `semi-ellipse-top`: This defines a point which determines the height of the semiellipse. In our example (ymin = ymax), the plane is in the (x,z)-coordinate plane. Top coordinates specify an arbitrary point "above" the ellipse, representing the base of the semiellipse.

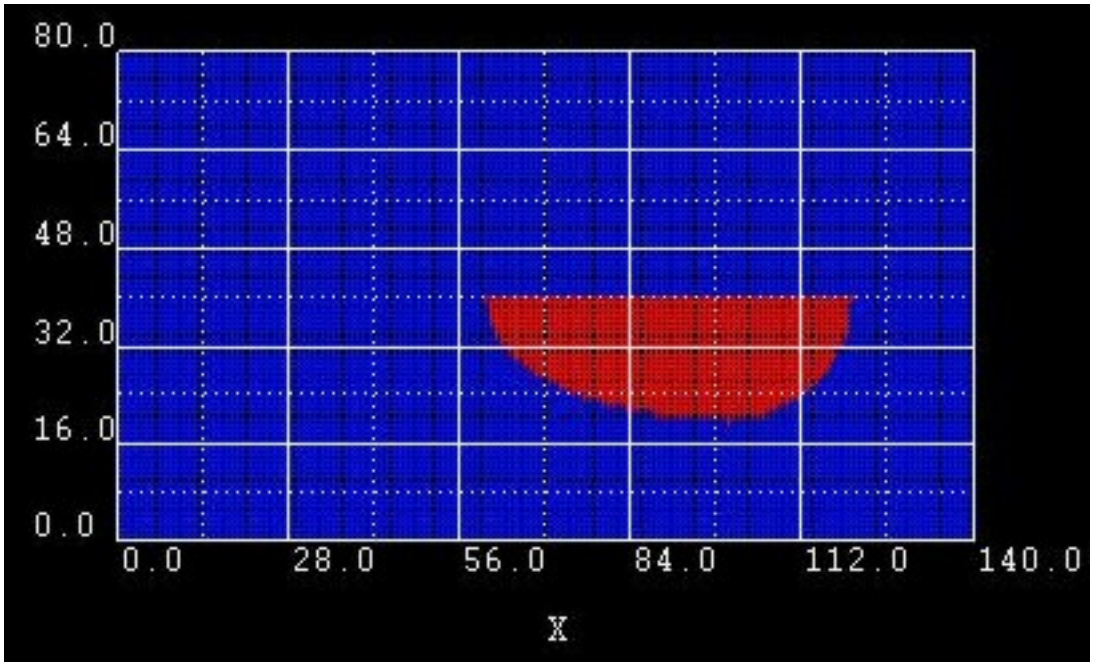
- Top coordinate must be lower than upper coordinate of base line border in direction of axis mentioned above.
- Top coordinate must be higher than lower coordinate of base line border in direction of axis mentioned above.
- e.g.) in case of ymin = ymax, xmin < x1 < xmax and (y1 < ymin=ymax or y1 > ymin=ymax)

Example1

```
semi-ellipse-base = 60.0 120.0 40.0 40.0 semi-ellipse-top = 100.0
20.0
```

From these data, the following points are extracted: (point: (x,y))

- base: (60,40) (120,40)
- top: (100,20)



Example2

```
semi-ellipse-base = 60.0 120.0 40.0 40.0 semi-ellipse-top = 100.0
60.0
```

We changed the **y1** coordinate of `semi-ellipse-top` from `20.0` to `60.0`.

Example3

```
semi-ellipse-base = 40.0 40.0 20.0 80.0  semi-ellipse-top = 100.0
60.0
```

Here we changed semi-ellipse-base: Now the two **x** coordinates have identical values.

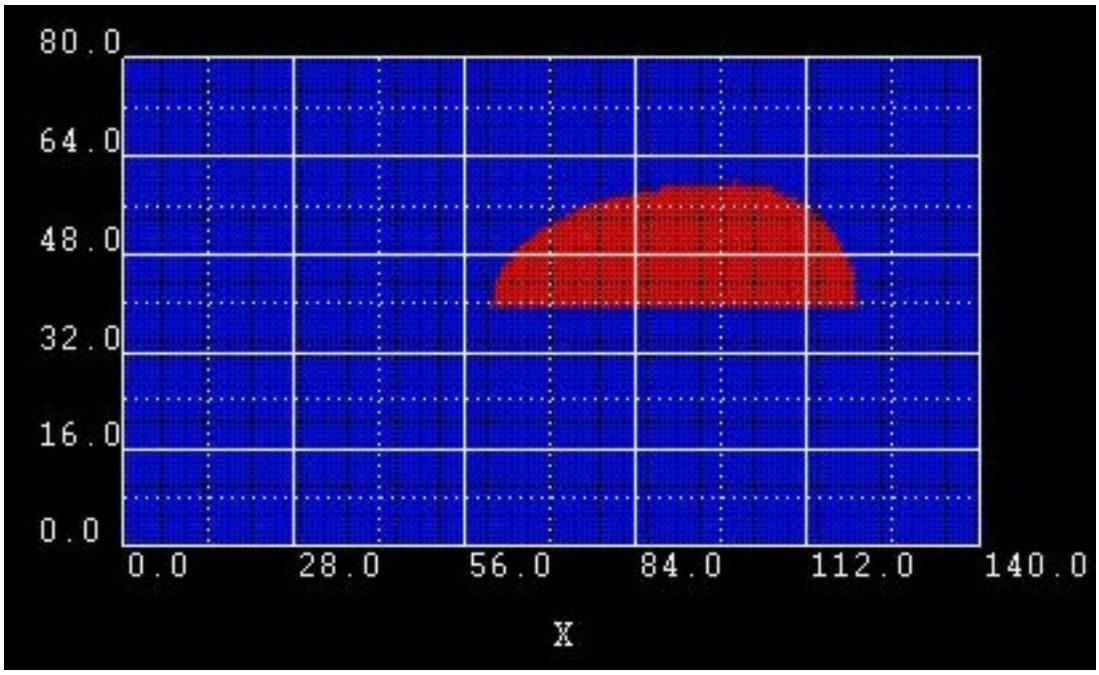
Example4

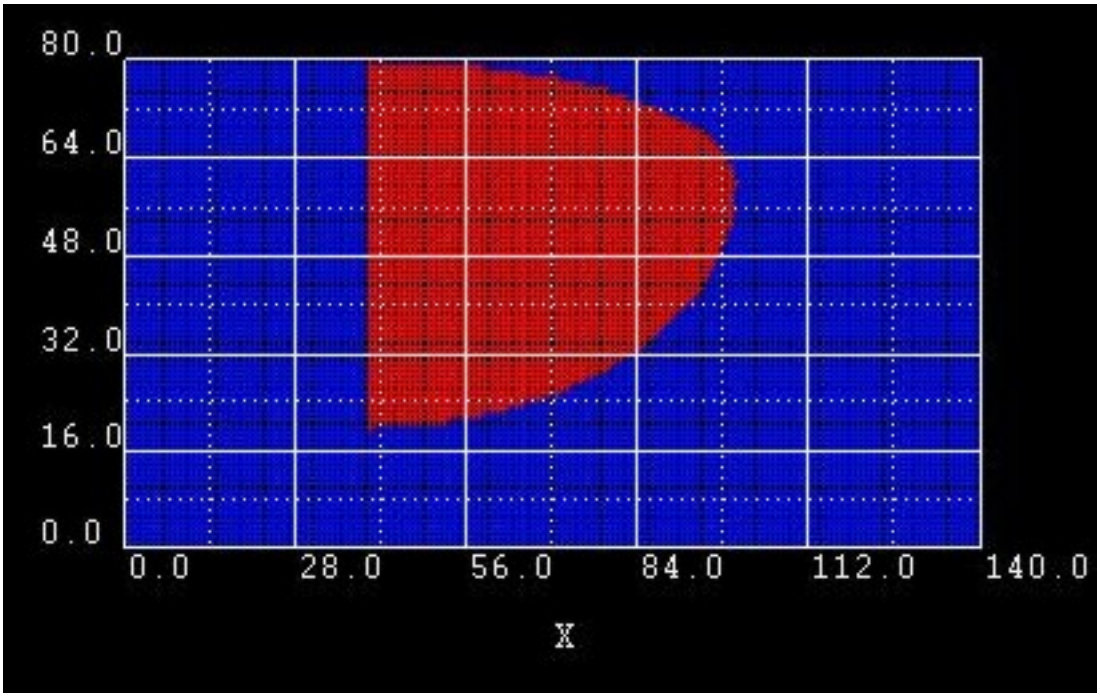
```
semi-ellipse-base = 40.0 40.0 50.0 70.0  semi-ellipse-top = 100.0
60.0
```

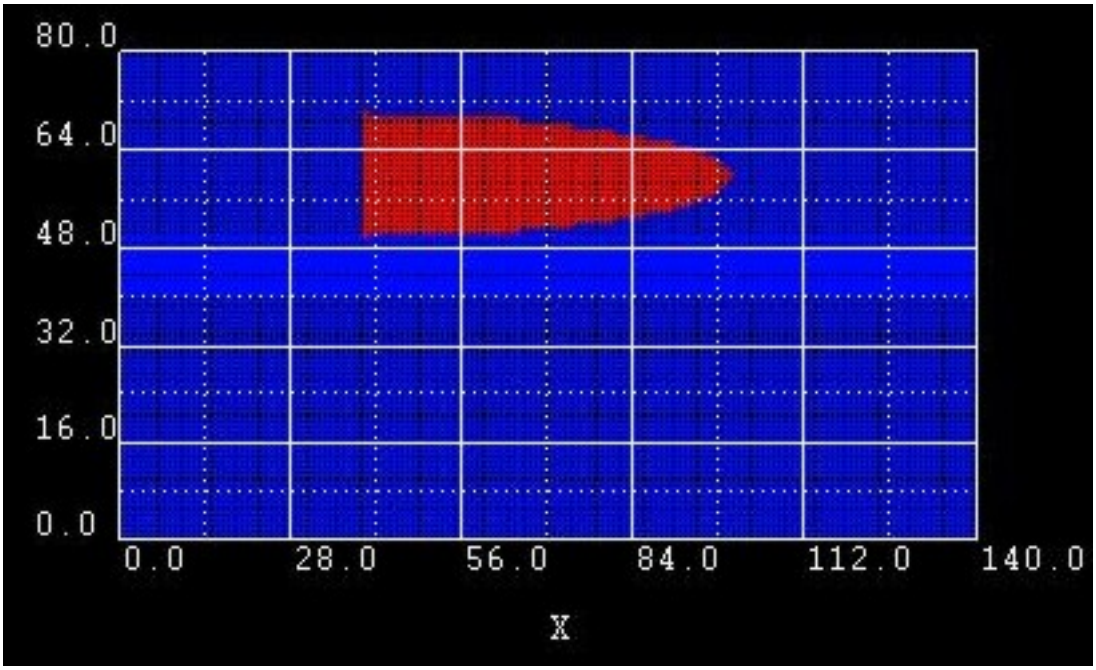
Here we changed the **ymin** and **ymax** coordinates of `semi-ellipse-base`: Now the **y** extension of the semiellipse is restricted from ymin = 50 nm to ymax = 70 nm. The baseline is at the fixed value for x = 40 nm.

Example5

```
semi-ellipse-base = 40.0 40.0 40.0 80.0   semi-ellipse-top = 60.0
60.0
```

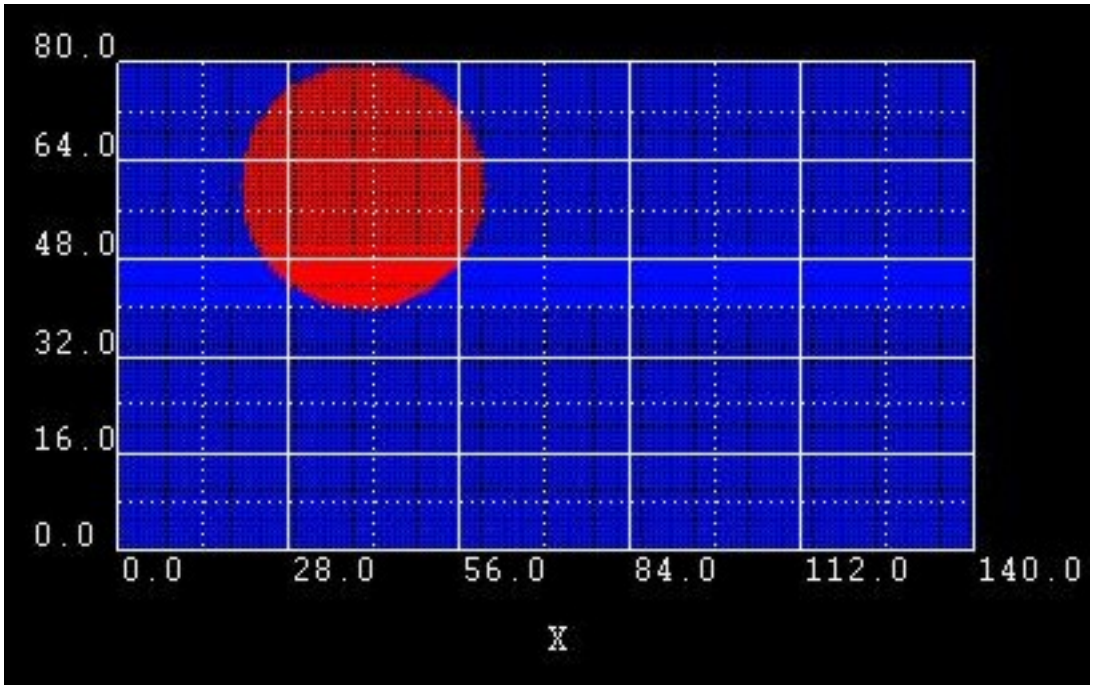

80.0
64.0
48.0
32.0
16.0
0.0
0.0
28.0
56.0
84.0
112.0
140.0
X


80.0
64.0
48.0
32.0
16.0
0.0
0.0
28.0
56.0
84.0
112.0
140.0
X




```
semi-ellipse-base = 40.0 40.0 40.0 80.0   semi-ellipse-top = 20.0
60.0
```

Here we built a circle out of 2 semi-ellipses. However, it is obviously easier to use `circle` instead.

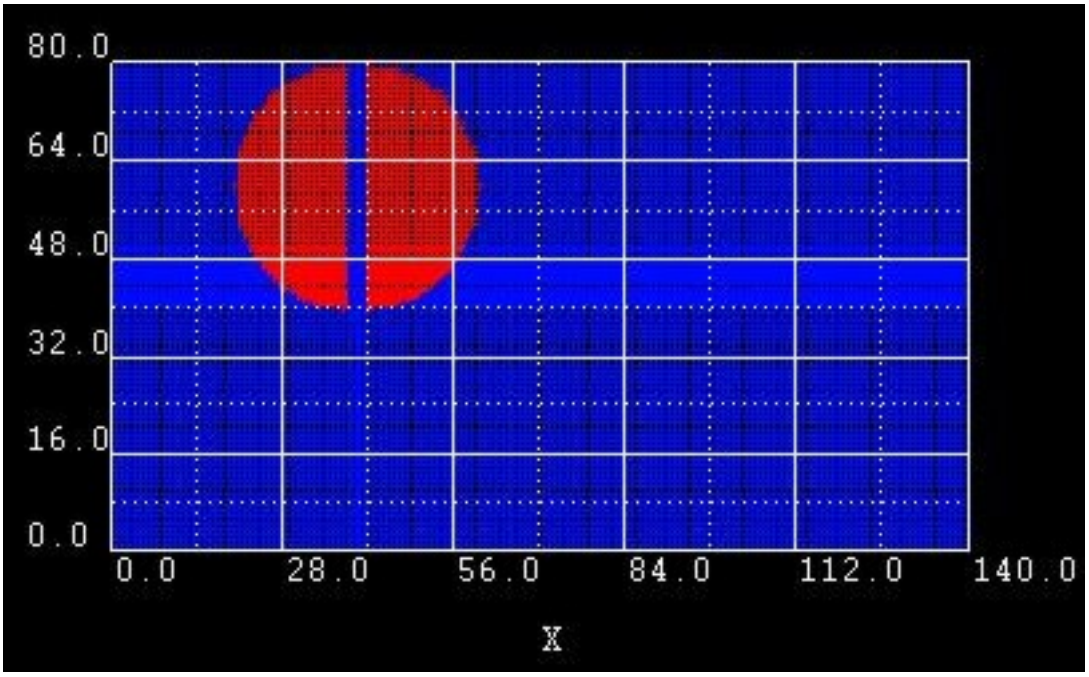


Example6

```
semi-ellipse-base = 42.0 42.0 40.0 80.0   semi-ellipse-top = 60.0
60.0
```

```
semi-ellipse-base = 38.0 38.0 40.0 80.0   semi-ellipse-top = 20.0
60.0
```

Same as example 5 but this time, we moved the baselines a little bit apart from each other to make example 5 easier to understand.



3-dimensional objects

cuboid

```
$regions
    region-number      = 1
    base-geometry      = cuboid
    region-priority    = 1
    x-coordinates      = xmin  xmax
    y-coordinates      = ymin  ymax
    z-coordinates      = zmin  zmax
$end_regions
```

- The surfaces of the `cuboid` are assumed to be in coordinate planes of the simulation coordinate system. The coordinates above specify the six coordinate planes which limit the cuboid.

sphere

```
$regions
    region-number      = 1
    base-geometry      = sphere
    region-priority    = 1
    center             = x     y     z  ! [nm]
    radius             = r              ! [nm]
$end_regions
```

- The `sphere` is defined by a center with coordinates (x,y,z) and a radius r.

obelisk

```
$regions
    region-number      = 1
    base-geometry      = obelisk
    region-priority    = 1
    base-coordinates   = xmin xmax  ymin ymax  zmin zmax
    top-coordinates    = xmin xmax  ymin ymax  zmin zmax
$end_regions
```

- Base and top plane of the obelisk have to be in parallel coordinate planes. These planes are identified by the implicit rule, that a pair of coordinate values (e.g. ymin ymax) has the same value (ymin = ymax).
- In this example, the plane is in the (x,z)-coordinate plane. The remaining four coordinates specify a rectangle in the corresponding plane.

truncated-cone and cylinder

```
$regions
    region-number      = 1
    base-geometry      = truncated-cone                    ! can be used to␣
↪specify a cylinder
    region-priority    = 1
    base-coordinates   = xmin xmax  ymin ymax  zmin zmax
    top-coordinates    = xmin xmax  ymin ymax  zmin zmax
$end_regions
```

- A cone with its apex cut off by a plane is called a truncated cone. In our implementation, the truncated cone is bounded by two ellipses of different size that are aligned parallel to each other.
- Base and top plane of the truncated cone have to be in parallel coordinate planes. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. xmin xmax) has the same value (xmin = xmax).
- In this example, the plane is in the (y,z)-coordinate plane. ymin ymax and zmin zmax specify the diameter of the truncated cone top and base in the y and z direction, respectively. This corresponds to the specification of ellipses in the base and top plane.

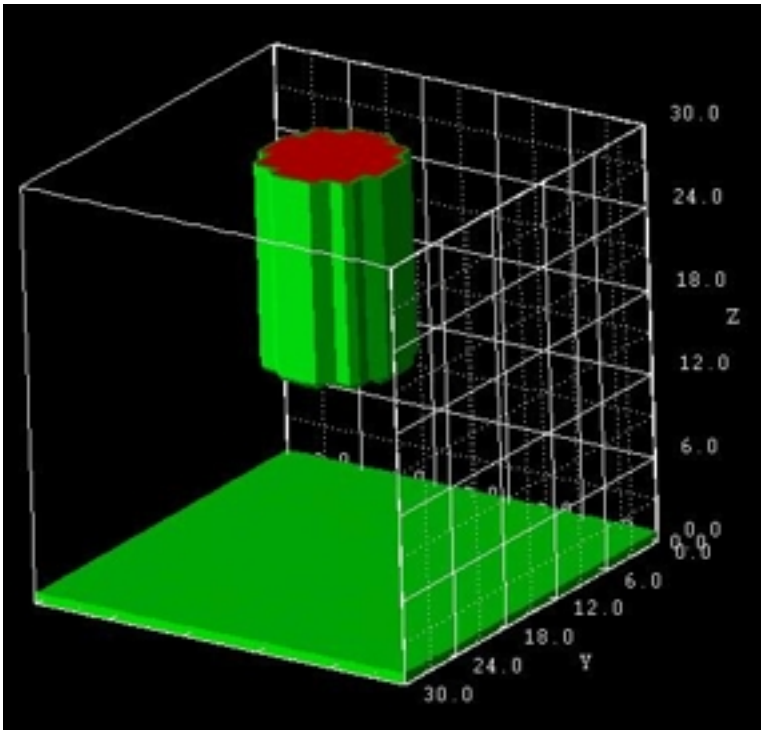
How to specify a cylinder?

A cylinder is specified as a special case of a `truncated-cone` where the two boundary planes are circles. For a truncated cone, one specifies base and top coordinates.

Let us assume we have a spherical cylinder of diameter 10 nm and height 15 nm. Then the base and top coordinates would be, for example, `base-coordinates = 10.0 20.0 10.0 20.0 15.0 15.0` (xmin,xmax,ymin,ymax,zmin,zmax) = (10,20,10,20,15,15)

`top-coordinates = 10.0 20.0 10.0 20.0 30.0 30.0` (xmin,xmax,ymin,ymax,zmin,zmax) = (10,20,10,20,30,30)

In other words, the x and y-coordinates specify the principal axes of the bottom and top ellipse (or circle) of the cylinder, respectively, and the z-coordinates specify the planes in which these two ellipses lie.



cone

```
$regions
    region-number       = 1
    base-geometry       = cone
    region-priority     = 1
    base-coordinates    = xmin xmax  ymin ymax  zmin zmax
    top-coordinates     = x y z
$end_regions
```

Base plane of the cone has to be in parallel to the coordinate system planes. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. xmin xmax) has the same value (xmin = xmax). In this example, the plane is in the (y,z)-coordinate plane. ymin ymax and zmin zmax specify the diameter of the cone base in the y and z direction, respectively. This corresponds to the specification of an ellipse in the base plane. The top of the cone, the apex, is defined by the point (x,y,z).

The cone does not have to be *right circular* (where *circular* means that the base is a circle and *right* means that the axis passes through the center of the base at right angles to its plane).

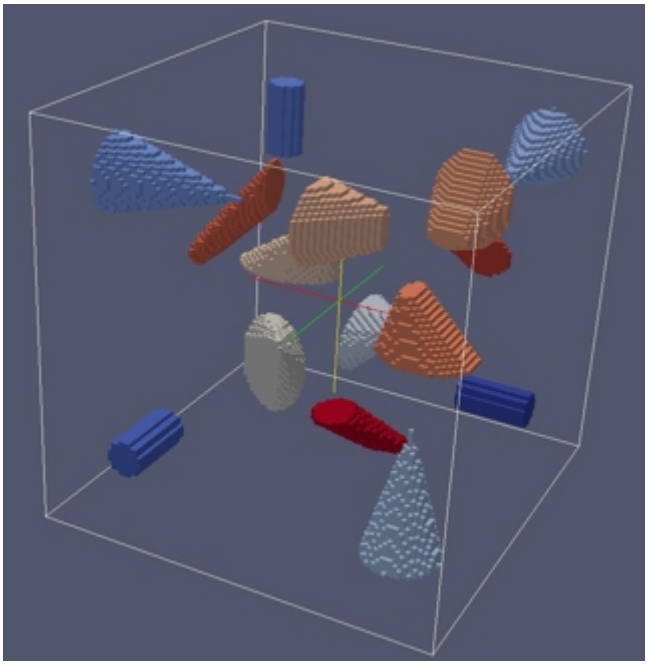
Oblique cones are allowed, in which the axis does not pass perpendicularly through the center of the base.

The following figure shows several **cones** and **truncated-cones** (including the special case of **cylinders**).

The cones are defined as follows:

```
!-------------------------------------------------------------------------
↪----
! This is a cylinder. The base and top planes are parallel to the (y,z)␣
↪plane.
!-------------------------------------------------------------------------
↪---
region-number     = 1     base-geometry = truncated-cone      region-priority␣
```

(continues on next page)



(continued from previous page)

```
↪= 1       ! cylinder
base-coordinates =  80.0 80.0     -80.0 -60.0     -80.0 -60.0                ␣
↪         ! xmin xmax ymin ymax zmin zmax
top-coordinates  =  40.0 40.0     -80.0 -60.0     -80.0 -60.0                ␣
↪         ! xmin xmax ymin ymax zmin zmax


!-------------------------------------------------------------------------
↪----
! This is a right circular cone. The base plane is parallel to the (y,z)␣
↪plane.
!-------------------------------------------------------------------------
↪----
region-number    = 2    base-geometry = cone               region-priority␣
↪= 1       ! cone (right circular)
base-coordinates = -80.0 -80.0     50.0 90.0     50.0 90.0                  ␣
↪         ! xmin xmax ymin ymax zmin zmax
top-coordinates  = -10.0 70.0 70.0                                          ␣
↪         ! x y z

!-------------------------------------------------------------------------
↪-----------------------------
! This is a cone where the projection of the apex onto the base plane is␣
↪located outside the base plane.
! The base plane is parallel to the (y,z) plane.
!-------------------------------------------------------------------------
↪-----------------------------
region-number    = 3    base-geometry = cone               region-priority␣
↪= 2       ! cone
base-coordinates = -10.0 -10.0    -60.0 -20.0    -80.0 -20.0                ␣
↪         ! xmin xmax ymin ymax zmin zmax
top-coordinates  =  40.0 -10.0 -10.0                                        ␣
↪         ! x y z
```

(continues on next page)

(continued from previous page)

```
!-------------------------------------------------------------------------
! This is a truncated cone. The base plane is parallel to the (y,z) plane.
!-------------------------------------------------------------------------
region-number    = 4   base-geometry = truncated-cone     region-priority =␣
↪3      ! truncated-cone
base-coordinates =  20.0 20.0      50.0 90.0      50.0 100.0              ␣
↪        ! xmin xmax ymin ymax zmin zmax
top-coordinates  =  50.0 50.0      60.0 80.0      60.0  70.0              ␣
↪        ! xmin xmax ymin ymax zmin zmax

!-------------------------------------------------------------------------
↪----------------------------------------
! These are three truncated cones where the projection of the top plane on␣
↪the base plane is outside the base plane.
! The base plane is parallel to the (y,z) plane.
!-------------------------------------------------------------------------
↪----------------------------------------
region-number    = 5    base-geometry = truncated-cone     region-priority␣
↪= 4      ! truncated-cone
base-coordinates = -50.0 -50.0     20.0 40.0      20.0  60.0              ␣
↪        ! xmin xmax ymin ymax zmin zmax
top-coordinates  = -20.0 -20.0     50.0 80.0      20.0  40.0              ␣
↪        ! xmin xmax ymin ymax zmin zmax
```

semiellipsoid

```
$regions
    region-number      = 1
    base-geometry      = semiellipsoid
    region-priority    = 1
    base-coordinates   = xmin xmax  ymin ymax  zmin zmax
    top-coordinates    = xtop ytop ztop
$end_regions
```

Base plane of the semiellipsoid must be in a coordinate plane. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. ymin ymax) has identical values (ymin = ymax).

In this example, the plane is in the (x,z)-coordinate plane. Top coordinates specify an arbitrary point "above" the ellipse, representing the base of the semiellipsoid.

Example

A 3D sphere can be constructed from two semiellipsoids. However, it is obviously easier to use `sphere` instead.

In this example, the bottom planes of the two half-spheres are at z = 5 nm. The upper half-sphere extends from 5 nm to 6 nm, the lower half-sphere from 5 nm to 4 nm. The extensions in x and y directions for both half-spheres are from 4 nm to 6 nm.

Consequently, the sphere has a diameter of 2 nm.

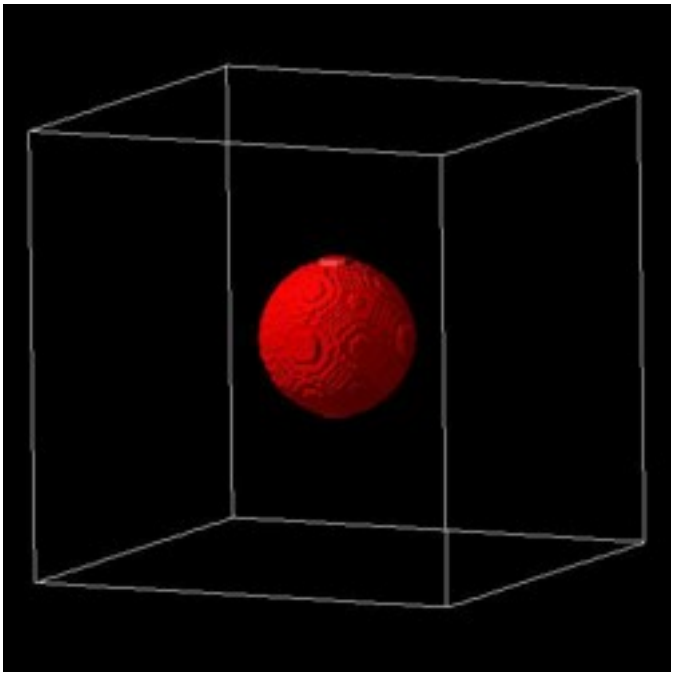
```
region-number    = 1
base-geometry    = semiellipsoid                ! (upper half-
↪sphere)
base-coordinates = 4.0 6.0   4.0 6.0   5.0 5.0   ! xmin xmax␣
↪ymin ymax zmin=zmax
top-coordinates  = 5.0 5.0 6.0                  ! xtop ytop␣
```

(continues on next page)

(continued from previous page)

```
↪ztop

region-number     = 2
base-geometry     = semiellipsoid                 ! (lower half-
↪sphere)
base-coordinates = 4.0 6.0   4.0 6.0   5.0 5.0   ! xmin xmax␣
↪ymin ymax zmin=zmax
top-coordinates   = 5.0 5.0 4.0                   ! xtop ytop␣
↪ztop
```



hexagonal-obelisk

```
$regions
    region-number       = 1
    base-geometry       = hexagonal-obelisk
    region-priority     = 1
    base-coordinates    = xmin xmax  ymin ymax  zmin zmax
    top-coordinates     = xmin xmax  ymin ymax  zmin zmax
$end_regions
```

Base and top plane of the hexagonal-obelisk have to be in parallel coordinate planes. These planes are identified by the implicit rule, that a pair of coordinate values (e.g. zmin zmax) has the same value (zmin = zmax). In this example, the plane is in the (x,y)-coordinate plane. The remaining four coordinates specify a rectangle in the corresponding plane.

This geometry element is useful for wurtzite.

Many thanks to Lu Fu-Fa (Institute of Technology (CCIT), Taiwan, R.O.C.) for useful suggestions regarding the implementation of this geometry element.

Two hexagonal-obelisk shapes are possible:

- **'hexagonal-cylinder' with 6-fold rotational symmetry axis oriented along the z direction (i.e. zmin = zmax).**
  Requirements: - xmin(base) = xmin(top) - xmax(base) = xmax(top) - ymin(base) =

ymin(top) - ymax(base) = ymax(top)

Note: This condition should be fullfilled:

– ymax(base) - ymin(base) > (xmax(base) - xmin(base)) / 0.866

where 0.866 = $cos\pi/6$

- **'hexagonal-pyramid' with 6-fold rotational symmetry axis oriented along the z direction (i.e. zmin = zmax).**
  Requirements: - xmin(top), xmax(top), ymin(top), ymax(top) arbitrary

  ---

  Note: This condition should be fullfilled:

  – ymax - ymin > (xmax - xmin) / 0.866

For both shapes it holds:

- height of pyramid/cylinder: zmax(top) - zmax(base) (zmin = zmax)
- width in x direction (distance between two parallel planes): xmax(base) - xmin(base)
- width in y direction (distance between two corners) if the condition ymax(base) - ymin(base) > (xmax(base) - xmin(base)) / 0.866 is fullfilled:
- If the above cited condition is not fullfilled, then the width is: ymax(base) - ymin(base)
- width along y > width along x if ymax(base) - ymin(base) > xmax(base) - xmin(base)
- center of hexagonal base plane on x axis: 0.5*(xmin(base) + xmax(base))
- center of hexagonal base plane on y axis: 0.5*(ymin(base) + ymax(base)): ymin(base) and ymax(base) can be used to shift the hexagon along the y axis.

Two sides of the hexagonal base plane are aligned parallel to the y axis. To rotate the hexagonal base plane by 30 degrees, the user has to specify values for xmin(top), xmax(top), ymin(top), ymax(top) so that it holds:

xmax(top) - xmin(top) > ymax(top) - ymin(top)

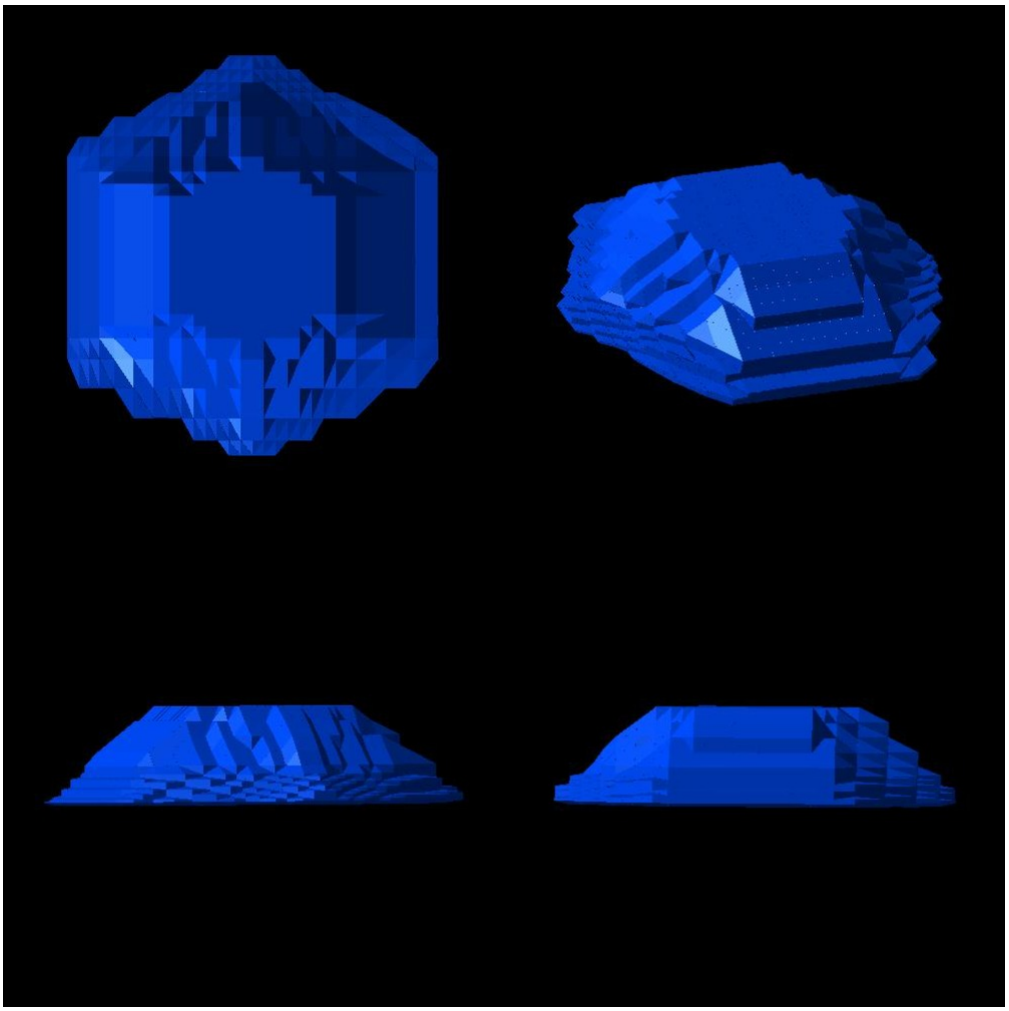
In this case it holds:

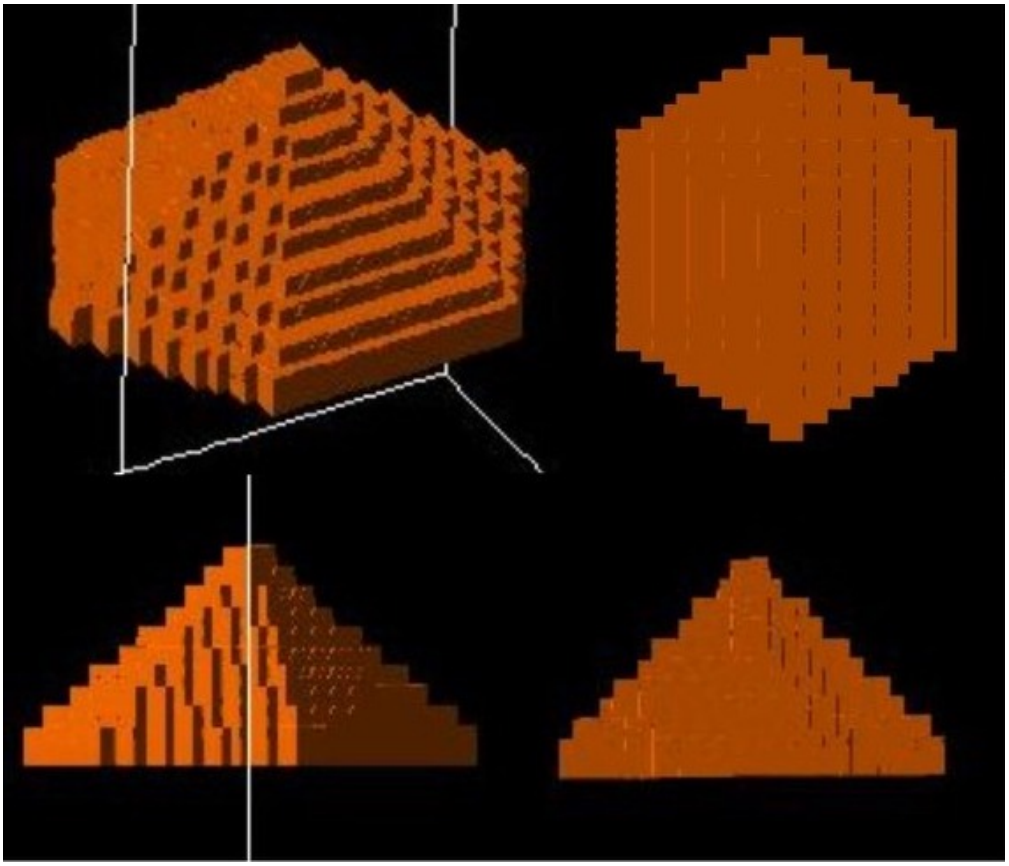
- width in x direction (distance between two corners) if the condition xmax(base) - xmin(base) > (ymax(base) - ymin(base)) / 0.866 is fullfilled:
- width in y direction (distance between two parallel planes): ymax(base) - ymin(base)
- width along y < width along x if xmax(base) - xmin(base) > ymax(base) - ymin(base)
- center of hexagonal base plane on x axis: 0.5*(xmin(base) + xmax(base)): xmin(base) and xmax(base) can be used to shift the hexagon along the x axis.

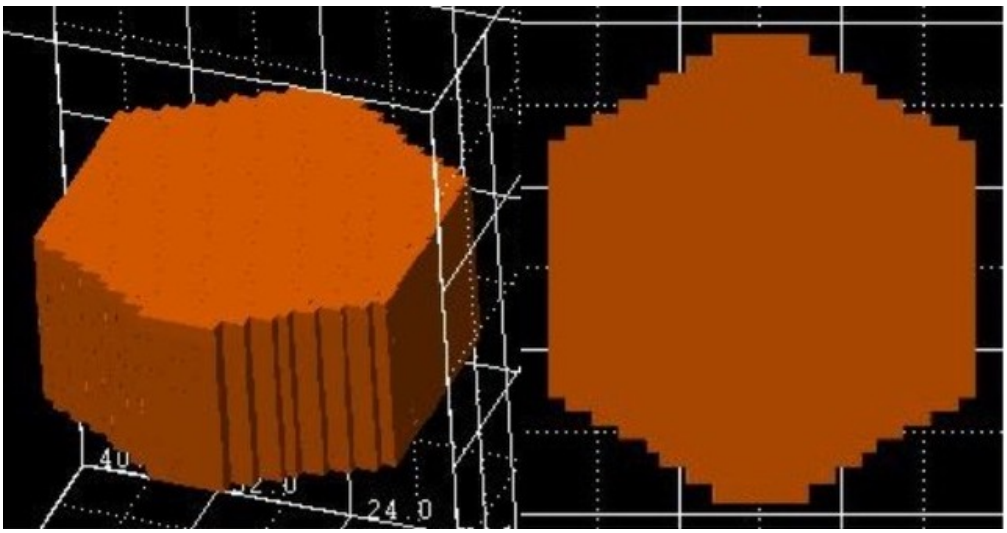
If the 6-fold rotational axis is oriented along the x (i.e. xmin = xmax) or y directions (i.e. ymin = ymax), cyclic permutations hold for the above statements.

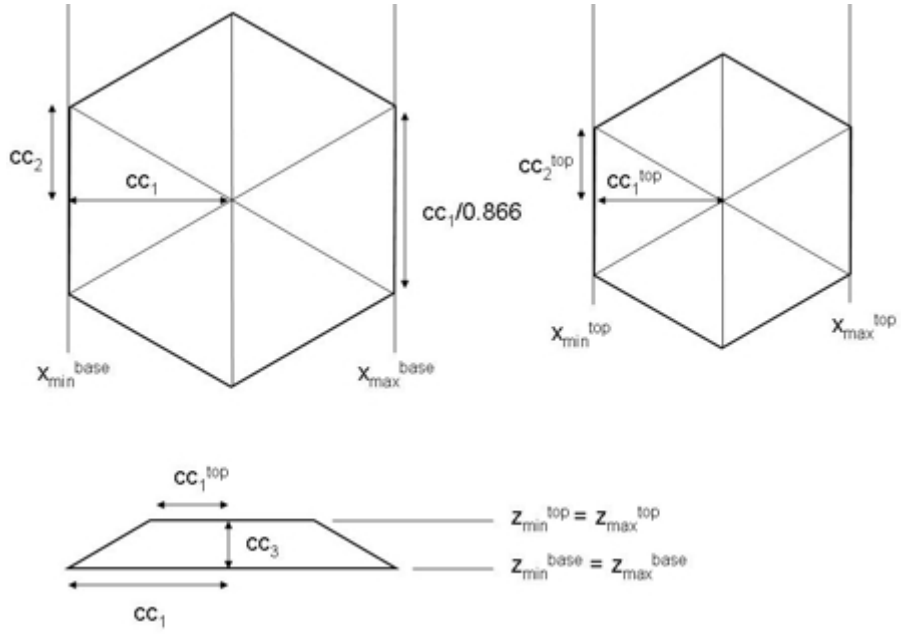
Example

- Hexagonal shaped pyramid with flat top plane:
- Hexagonal shaped pyramid:
- Hexagonal shaped "cylinder":






24.0



triangular-prism

```
$regions
    region-number      = 1
    base-geometry      = triangular-prism
    region-priority    = 1
    corner-coordinates = x1 y1 z1   x2 y2 z2    x3 y3 z3
                         x4 y4 z4   x5 y5 z5    x6 y6 z6
$end_regions
```

Restrictions: `triangular-prism` must be oriented so that the triangles are perpendicular to either the x, y or z directions.

Example: Triangles perpendicular to z direction. Then it must hold:

`corner-coordinates`

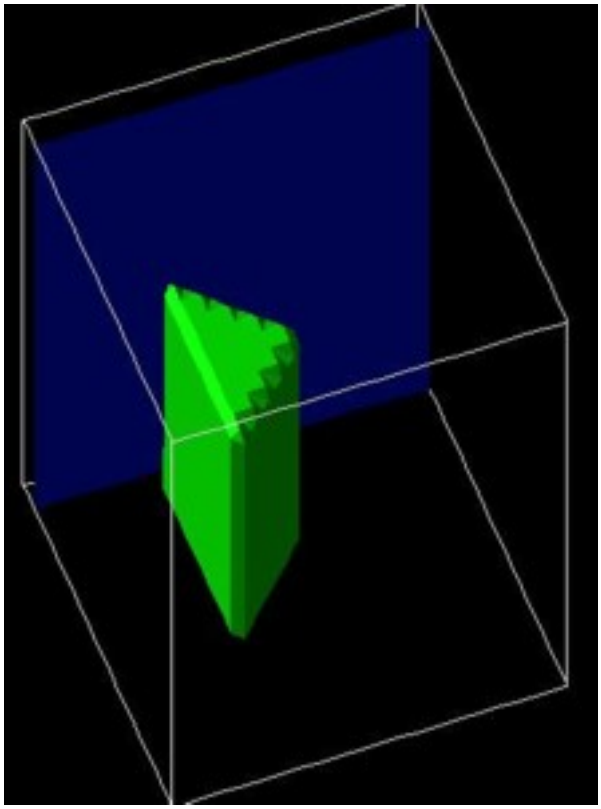
- z1 = z2 = z3
- 4 = z5 = z6

In addition it holds:

- x1 = x4, y1 = y4
- x2 = x5, y2 = y5
- x3 = x6, y3 = y6

Example

```
$regions
    region-number      = 1
    base-geometry      = triangular-prism
    region-priority    = 1
    corner-coordinates = x1 y1 z1   x2 y2 z2    x3 y3 z3
                         x4 y4 z4   x5 y5 z5    x6 y6 z6
$end_regions
```



$region-cluster

Regions can be clustered to bigger objects by the keyword `$region-cluster`. Any region must be assigned to a cluster which is labeled by its `cluster-number`.

Also the so-called *default region* must be assigned to a cluster. The default region is the rest of the simulation domain which is not filled out by defined regions. It might happen that the default region is an *empty region*. Nevertheless, it must be assigned to a cluster.

In any case it holds:

default region number = maximum region number defined + 1

The regions which have to be clustered to one cluster are specified by their numbers like this `region-number = 2 3 5`.

$region-cluster		**required**
cluster-number	**integer**	**required**
region-numbers	**integer_array**	**required**
apply-constant-el-Fermi-level	**double**	**optional**
apply-constant-hl-Fermi-level	**double**	**optional**
$end_region-cluster		**required**

Explanation of specifiers

cluster-number

type
integer >= 1

presence
required

An integer number to refer to clustered geometry element.

region-numbers

type
integer array containing integers >= 1

presence
required

Region numbers belonging to this cluster `cluster-number`.

apply-constant-el-Fermi-level

type
double

presence
optional

unit
`[eV]`

example
`0.2`

Applies a constant electron Fermi level $E_{\mathrm{F,n}}$ to this cluster. The energetic distance of the conduction band edges with respect to this Fermi level determines the electron density.

apply-constant-hl-Fermi-level

type
double

presence
optional

unit
`[eV]`

example
`-0.1`

Applies a constant hole Fermi level $E_{\mathrm{F,p}}$ to this cluster. The energetic distance of the valence band edges with respect to this Fermi level determines the hole density.

Example

We have defined 5 regions under keyword *$regions*, region 2 and 4 are clustered to form `cluster-number 2` (i.e. region 2 and 4 now will consist of the same *material* as a material kind is assigned to a *cluster* and not to a *region*). `region-number 6` is an undefined *default region* (Assumption: We have defined only 5 regions.) and must be the default rest. Now there are two possibilities:

Possibility 1:

```
!-------------------------------------------------!
$region-cluster                                   !
 cluster-number = 1      region-numbers = 1       ! e.g. GaAs
 cluster-number = 2      region-numbers = 2  4    ! e.g. AlAs
 cluster-number = 3      region-numbers = 3       ! e.g. GaAs
 cluster-number = 4      region-numbers = 5  6    ! e.g. InP, 6 = default␣
↪region
$end_region-cluster                               !
!-------------------------------------------------!
```

Here, the default region has the same material properties as `cluster-number 4`. The default region can consist of some undefined region (the *remaining part* of the simulation area which is not filled yet) or it can be *empty* if the *whole* simulation area is already filled with other objects.

Possibility 2:

```
!-------------------------------------------------!
$region-cluster                                   !
 cluster-number = 1      region-numbers = 1       ! e.g. GaAs
 cluster-number = 2      region-numbers = 2  4    ! e.g. AlAs
 cluster-number = 3      region-numbers = 3       ! e.g. GaAs
 cluster-number = 4      region-numbers = 5       ! e.g. InP
 cluster-number = 5      region-numbers = 6       ! e.g. InP, 6 = default␣
↪region
$end_region-cluster                               !
!-------------------------------------------------!
```

Here, the default region has its *own* material properties which have to be defined later under keyword *$material*. Again, the default region can consist of some undefined region (the *remaining part* of the simulation area which is not filled yet) or it can be *empty* if the *whole* simulation area is already filled with other objects.

Useful application for a default region

If one has a complicated geometry such as a quantum dot which is surrounded by a GaAs cap layer, one can define the complicated geometry as usual and for the (complicated) surrounding material one just uses the *default* option.

Note: Be careful in 3D when you specify for strain calculation `strain-minimization` (see *$strain-minimization-model*) to define your substrate as a separate cluster.

$quantum-cluster

For quantum mechanical calculations, you have to define quantum regions (*$quantum-regions*) and quantum clusters on which certain quantum models (*$quantum-model-electrons*, *$quantum-model-holes*) are applied. As for the *regions* and *current-regions*, *quantum-regions* can be *clustered*. At least cluster number 1 has to be specified if the quantum mechanical properties should be calculated at all.

The syntax of the *region-cluster*, *quantum-cluster*, and *current-cluster* is very similar. For more information on these specifiers see *$region-cluster* and *$current-cluster*.

```
$quantum-cluster                                optional
 cluster-number                 integer         required
 region-numbers                 integer_array   required
 deactivate-cluster             character       optional
 apply-constant-el-Fermi-level  double          optional
 apply-constant-hl-Fermi-level  double          optional
$end_quantum-cluster                            optional
```

Explanation of specifiers

cluster-number

type
integer >= 1

presence
required

An integer number to refer to clustered geometry element.

region-numbers

type
integer array containing integers >= 1

presence
required

Region numbers belonging to the cluster `cluster-number`.

deactivate-cluster

type
character

presence
optional

value
`yes` or `no`

default
`no`

Flag to switch off quantum cluster. Switching off quantum calculation means classical simulation. This is a very convenient way to turn the quantum calculation off and on. For a first test, it is recommended to a classical calculation as it is much faster. Then you can check whether the geometry and the doping regions are defined correctly, and if the strain has the expected influence on the conduction and valence band edges. Once you are convinced that you have set up the structure correctly, you can do a quantum mechanical simulation.

apply-constant-el-Fermi-level

type
double

presence
optional

unit
`[eV]`

example
`0.2`

Applies a constant electron Fermi level $E_{\mathrm{F,n}}$ to this quantum cluster. The energetic distance of the conduction band edges with respect to this Fermi level determines the electron density.

apply-constant-hl-Fermi-level

type
double

presence
optional

unit
`[eV]`

example
`-0.1`

Applies a constant hole Fermi level $E_{\mathrm{F,p}}$ to this quantum cluster. The energetic distance of the valence band edges with respect to this Fermi level determines the hole density.

Examples

Here, only one quantum cluster is defined.

```
!-------------------------------------------
$quantum-cluster
 cluster-number = 1     region-numbers = 1
 deactivate-cluster = no
!deactivate-cluster = yes
$end_quantum-cluster
!-------------------------------------------
```

Here, two quantum regions are combined to form a common quantum cluster.

```
!-------------------------------------------
$quantum-cluster
 cluster-number = 1     region-numbers = 1 2
$end_quantum-cluster
!-------------------------------------------
```

Here, three separate quantum clusters are defined that are treated independently, i.e. three different Schrödinger equations are solved. Typically, it is best to have only one quantum cluster. However, in some cases, using separate clusters can lead to faster calculations. But this should be used with care, as often the wave functions with higher energies are distributed over the whole device.

```
!-------------------------------------------
$quantum-cluster
 cluster-number = 1     region-numbers = 1
 cluster-number = 2     region-numbers = 2
 cluster-number = 3     region-numbers = 3
$end_quantum-cluster
!-------------------------------------------
```

$current-cluster

For current calculations, you have to define current regions (*$current-regions*) and current clusters on which certain current models (*$current-models*) are applied. As for the *regions* and *quantum-regions*, *current-regions* can be *clustered*. At least cluster number 1 has to be specified if the current has to be calculated at all.

The syntax of the *region-cluster*, *quantum-cluster*, and *current-cluster* is very similar. For more information on these specifiers see *$region-cluster* and *$quantum-cluster*.

```
$current-cluster                        optional
 cluster-number          integer        required
 region-numbers          integer_array  required
 deactivate-cluster      character      optional
$end_current-cluster                    optional
```

Explanation of specifiers

cluster-number

type
integer >= 1

presence
required

An integer number to refer to clustered geometry element.

region-numbers

type
integer array containing integers >= 1

presence
required

Region numbers belonging to cluster `cluster-number`.

deactivate-cluster

type
character

presence
optional

value
yes or no

default
no

Flag to switch off current clusters. Switching off current calculation means equilibrium simulation. This is a very convenient way to turn the current calculation off and on.

Example

```
!---------------------------------------------
$current-cluster
 cluster-number = 1     region-numbers = 1
 deactivate-cluster  = no
!deactivate-cluster  = yes
$end_current-cluster
!---------------------------------------------
```

$current-regions

In order to make it possible to combine several different modes of current calculation in one device, there is an extra set of regions and clusters for the calculation of the current. The syntax is the same as for the other regions (*$regions*, *$quantum-regions*).

If no current region is specified the whole device is numbered as current region 1, i.e. this keyword does not have to be present.

```
$current-regions                        optional
 region-number              integer     required
 base-geometry              character   required
 region-priority            integer     required

 x-coordinates              double_array optional
 y-coordinates              double_array optional
 z-coordinates              double_array optional

 base-coordinates           double_array optional
 top-coordinates            double_array optional

 corner-coordinates         double_array optional

 semi-ellipse-base          double_array optional
 semi-ellipse-top           double_array optional
$end_current-regions                    optional
```

$current-models

So far only the drift-diffusion model has been implemented but you never know what comes next.

```
$current-models                              optional
 model-number                integer         required
 transport-model-name        character       required
 cluster-numbers             integer_array   required
$end_current-models                          optional
```

Explanation of specifiers

model-number

type
integer >= 1

presence
required

value
1

An integer number to refer to current model. Currently, only one model is implemented, so you have to choose 1.

transport-model-name

type
character

presence
required

value
`simple-drift-model`

This is the only option currently.

cluster-numbers

type
integer array

presence
required

value
1

Currently, only one current cluster supported so far.

Example

```
!-------------------------------------------
$current-models
 model-number          = 1
 transport-model-name = simple-drift-model
 cluster-numbers       = 1
$end_current-models
!-------------------------------------------
```

$impurity-parameters

To specify the properties of impurities used in the simulation.

$impurity-parameters		**optional**
impurity-number	**integer**	**required**
impurity-name	**character**	**optional**
impurity-type	**character**	**required**
number-of-energy-levels	**integer**	**required**
energy-levels-relative	**double_array**	**required**
degeneracy-of-energy-levels	**integer_array**	**required**
transition-times-cb-to-levels	**double_array**	**optional**
transition-times-levels-to-vb	**double_array**	**optional**
$end_impurity-parameters		**optional**

Specify here the properties of the impurities used for doping.

impurity-number

type
integer

presence
required

example
1

1, 2, 3, ... are unique impurity numbers labeled in *$doping-function*.

Optionally, a name for the impurity can be provided.

impurity-name

type
character

presence
optional

example
`n-As-in-Si`

example
`0.010*x+0.050*(1-x)` (unit: eV)

example
`0.010*x+0.050*(1-x)-0.020*x*(1-x)` (unit: eV) Here, a bowing factor of 0.020 eV is included.

Specify an arbitrary name. The name will be part of some output files. (for later use; it is planned to read in impurity parameters from the database.)

Special feature: If `impurity-name` contains the character `x`, this string is then interpreted as an equation where `x` is the alloy content. This way, a position dependent impurity level can be simulated, e.g. for graded Al(x)Ga(1-x)N layers. In this example, the impurity level would be 10 meV for AlN (x=1) and 50 meV vor GaN (x=0). Note that this has influence on CPU time, i.e. the calculation will take longer because the result of the equation has to be evaluated during the calculation.

(For later use: It is planned to read in impurity parameters from the database.)

Several types of impurities are supported.

impurity-type

type
character

presence
required

options
`n-type`, `p-type`, `trap`

Specifies the type of an impurity. `n-type` means that the impurity is treated as a donor, `p-type` as an acceptor. Option `trap` is not supported so far.

Each impurity can have several different energy levels.

number-of-energy-levels

type
integer

presence
required

example
`1`

example
`1 2`

Number of different energy levels for this impurity. Each energy level is specified in `energy-levels-relative`.

Energy levels

energy-levels-relative

type
double array

presence
required

unit
eV

example
`0.005`

example
`0.005 0.015`

example
`1000.0`

A large negative value, e.g. -1000.0 eV implies **full ionization**.

Energy levels relative to "nearest" band edge (`n-type` -> conduction band, else valence band) in units of eV. As many energies as energy levels. These energies are meant as **ionization energies**, e.g. a donor with an energy level right below the conduction band edge would be specified by a small positive energy level.

When impurity levels are relatively deep compared to the thermal energy kBT at room temperature, incomplete ionization must be considered.

Note: 'Cheat' parameter: `energy-levels-relative = -1000.0`, for instance, that means, all electrons are fully ionized from the donors (similar for holes/acceptors). This might be useful for low temperatures like 4 K where usually the degree of ionization is very small. By using `-1000.0` one can force them to be completely ionized.

The energy levels of the donors and acceptors relative to the lowest conduction band edge and highest valence band edge can be output using `dopant-energy-levels = yes` (see *$output-densities*).

See also our tutorial on Doped semiconductors to learn more about partial ionization.

Degeneracy of the specified energy levels

degeneracy-of-energy-levels

type
integer array

presence
required

example
`2`

example
`4 4`

Shallow donors have degeneracy factor `2`: Outer s orbital is onefold occupied (neutral state). There is one possibility to get rid of one electron but there are two to incorporate one (spin up, spin down).

Shallow acceptors have degeneracy factor `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]*. Note that in nitride semiconductors crystallizing in the wurtzite structure the degeneracy factor may vary from 4 to 6 because of a small valence band splitting.

If **full ionization** is assumed, i.e. `energy-levels-relative = -1000.0`, then the degeneracy factor effectively becomes irrelevant. This can be seen from eqs. (1.4) - (1.7) in *[BirnerPhD2011]*.

transition-times-cb-to-levels

type
double array

presence
optional

unit
?

Transition times tau1, tau2, tau3, … from conduction band(s) to energy levels; required in case of `trap`: times from conduction band to discrete levels.

transition-times-levels-to-vb

type
double array

presence
optional

unit
?

Transition times tau1, tau2, tau3, … from energy levels to valence bands band(s); required in case of `trap`: times from discrete levels to valence bands.

Note: Currently no interlevel transition times implemented. Can be added provided there are also models which can handle such things.

Example

```
$impurity-parameters
```

(continues on next page)

(continued from previous page)

```
!--------------------------------
! n-type in GaAs
!--------------------------------
impurity-number              = 1
impurity-name                = n-Si-in-GaAs
impurity-type                = n-type
number-of-energy-levels      = 1
energy-levels-relative       = 0.0058
degeneracy-of-energy-levels  = 2        ! 2 for n-type

!--------------------------------
! n-type in GaAs (fully ionized)
!--------------------------------
impurity-number              = 2
impurity-name                = n-fully-ionized
impurity-type                = n-type
number-of-energy-levels      = 1
energy-levels-relative       = -1000.0  ! fully ionized
degeneracy-of-energy-levels  = 2        ! 2 for n-type

!--------------------------------
! p-type in GaAs
!--------------------------------
impurity-number              = 3
impurity-name                = p-C-in-GaAs
impurity-type                = p-type
number-of-energy-levels      = 1
energy-levels-relative       = 0.027
degeneracy-of-energy-levels  = 4         ! 4 for p-type

!--------------------------------
! p-type in GaAs (fully ionized)
!--------------------------------
impurity-number              = 2
impurity-name                = p-fully-ionized
impurity-type                = p-type
number-of-energy-levels      = 1
energy-levels-relative       = -1000.0  ! fully ionized
degeneracy-of-energy-levels  = 4        ! 4 for n-type

$end_impurity-parameters
```

Database values

```
energy-levels-relative = ...       ! energy in units of [eV]
                       = -1000.0   ! a large negative value implies full␣
↪ionization
                       = 0.054     ! n-As-in-Si
                       = 0.045     ! n-P -in-Si
                       = 0.039     ! n-Sb-in-Si
                       = 0.045     ! n-N -in-Si
                       = 0.006     ! n-Si-in-Al0.27Ga0.73As
                       = 0.0058    ! n-Si-in-GaAs
                       = 0.007     ! n-Si-in-AlAs
                       = 0.10      ! n-N -in-SiC
                       = 0.20      ! p-Al-in-SiC
```

(continues on next page)

(continued from previous page)

```
                        = 0.045     ! p-B -in-Si
                        = 0.16      ! p-In-in-Si
                        = 0.027     ! p-C -in-GaAs
```

More parameters can be found in the database file `database_nn3.in` or at this website: http://www.ioffe.ru/SVA/NSM/Semicond/

$material-interfaces

To specify additional charges at material interfaces, one has to specify

- material interfaces
- interface state properties.

See also documentation for keyword *$interface-states*.

```
$material-interfaces                                optional
 interface-number                  integer          required
 apply-between-material-numbers    integer_array    required
 state-numbers                     integer_array    required
$end_material-interfaces                            optional
```

Explanation of specifiers

interface-number

type
integer >= 1

presence
required

An integer number to refer to interface number. Dense numbering (1, 2, ...) as usual.

apply-between-material-numbers

type
integer >= 1

presence
required

Two integer numbers to refer to interface between geometry clusters, i.e. it contains the *material numbers* of the adjoining regions.

Is `3  4` equivalent to `4  3` in 1D? This becomes relevant for 2D and 3D simulations.

state-numbers

type
integer >= 1

presence
required

Identification numbers of interface states (e.g. fixed charge, trap or electrolyte) defined under keyword *$interface-states*.

Note: There can only be *one* unique `interface-number` for the the interface between `integer1` and `integer2`.

```
interface-number                = 1
apply-between-material-numbers = integer1  integer2
```

Here is an example which does not make sense (!):

```
interface-number                = 1
apply-between-material-numbers = 3 4

interface-number                = 2
apply-between-material-numbers = 3 4  ! '3 4' has already been assigned
                                      ! in 'interface-number = 1'
```

$magnetic-field

One can apply a magnetic field only for 2D and 3D simulations. For 1D simulations this is not possible.

```
$magnetic-field                                  optional
 magnetic-field-on                  character    required
 magnetic-field-strength            double       required
 magnetic-field-direction           integer_array required

 magnetic-field-sweep-active        character    optional
 magnetic-field-sweep-step-size     double       optional
 magnetic-field-sweep-number-of-steps integer    optional

 output-magnetic-vector-potential   character    optional
 exclude-region-cluster-numbers     integer_array optional
$end_magnetic-field                              optional
```

Example

```
!-------------------------------------------------!
$magnetic-field                                   !
 magnetic-field-on                = yes           ! yes/no
 magnetic-field-strength          = 4.0           ! [T]
 magnetic-field-direction         = 0 0 1         ! along z direction
                                                  !
 output-magnetic-vector-potential = yes           ! yes/no
 exclude-region-cluster-numbers   = 2 3           !
$end_magnetic-field                               !
!-------------------------------------------------!
```

Here, a magnetic field of 4 T is applied.

magnetic-field-on

type
character

presence
required

options
yes or no

default
no

Flag for switching magnetic field *on* or *off*.

magnetic-field-strength

type
double

unit
[T]

example
2.5

The magnetic field strength refers to the magnetic flux density $\mathbf{B}$ which has the SI unit [T] = [Vs/m^2]. Note that the SI unit for the magnetic field strength $\mathbf{H}$ is [A/m]. It is possible to specify a negative value which inverts the vector specified here: magnetic-field-direction

magnetic-field-direction

type
integer array of dimension 3

example
0 0 1

The 3 (integer) indices x, y, z of a vector $\mathbf{x} = (x, y, z)$ which is parallel to the magnetic field vector.

The direction of the magnetic field must be perpendicular to simulation orientation, e.g. if

```
!-------------------------------------!
$simulation-dimension                 !
 dimension                = 2         !
 orientation              = 0 1 1     ! simulation in (y,z) plane
$end_simulation-dimension             !
!-------------------------------------!
```

then the magnetic field direction must be

```
magnetic-field-direction = 1 0 0      ! along x direction
```

For a 3D simulation, the magnetic field can have any direction. The vector $\mathbf{B}$ must be specified with respect to the xyz simulation system, i.e. you cannot specify the four-digit Miller-Bravais indices as in the case for wurtzite. It is with respect to the x, y and z coordinate axes that were specified under *$domain-coordinates*: If not specified, default values specified in database_nn3.in are taken.

```
x y z  :=  hkl-x-direction-zb, hkl-y-direction-zb, hkl-z-direction-zb

or

x y z  :=  hkil-x-direction,   hkil-y-direction,   hkil-z-direction
```

(optional, only needed for magnetic field sweep)

output-magnetic-vector-potential

type
character

options
yes or no

default
no

The magnetic vector potential $\mathbf{A}(x, y, z)$ is defined with respect to the *symmetric gauge*: $\mathbf{A}(\mathbf{r}) = -1/2(\mathbf{r} \times \mathbf{B})$.

exclude-region-cluster-numbers

type
integer array

example
2 4

Here, cluster numbers 2 and 4 are exempted from magnetic field, i.e. B=0. This is useful for Aharonov–Bohm effect (*not fully implemented yet*).

Magnetic field sweep

It is possible to sweep over the magnetic field strength, i.e. to vary the strength of the magnetic field stepwise. This is similar to electric field sweeps (*$electric-field*), voltage sweeps (*$voltage-sweep*) and doping concentration sweeps (*$doping-function*).

magnetic-field-sweep-active

type
character

options
yes or no

default
no

Specify yes if you want to sweep over several values of the magnetic field.

magnetic-field-sweep-size

type
double

example
0.5

example
[T]

Here, the magnetic field increases in steps of 0.5 [T]. A negative value can be used to decrease the magnetic field.

magnetic-field-sweep-number-of-steps

type
integer

example
10

Specify here the number of steps for the magnetic field sweep.

Example

```
!-------------------------------------------------!
$magnetic-field                                   !
 magnetic-field-on                  = yes         ! yes/no
 magnetic-field-strength            = 0.0         ! [T]
 magnetic-field-direction           = 0 0 1       ! along z direction
                                                  !
 magnetic-field-sweep-active        = yes         ! yes/no
 magnetic-field-sweep-step-size     = 0.5         ! [T]
 magnetic-field-sweep-number-of-steps = 10        !
$end_magnetic-field                               !
!-------------------------------------------------!
```

Here, the magnetic field increases from 0 [T] to 5 [T] in steps of 0.5 [T], i.e. 11 simulations are performed.

The output is labeled with _ind000.dat, _ind001.dat, _ind002.dat, ... where the index refers to the number of the magnetic field sweep step.

The output for the eigenvalues as a function of applied magnetic field can be found here: magnetic_ev_vb1_qc1_sg1_deg1.dat.

In this particular example, the heavy hole valence band edge energies (vb1) have been written out as a function of magnetic field. The first column contains the strength of the magnetic field in units of [T]. The second column contains the 1st eigenvalue for the specified electric field in units of [eV], the third column contains the 2nd eigenvalue for the specified electric field in units of [eV], ...

Note: The magnetic field is only implemented for the single-band effective mass model but not for the multi-band $\mathbf{k} \cdot \mathbf{p}$ model. For the combination of $\mathbf{k} \cdot \mathbf{p}$ and magnetic field, please use *nextnano++*.

$Auger-recombination

Any values specified here in the input file overwrite the values specified in the material database.

Please refer to the database section of *$Auger-recombination* for more details on the meaning of these parameters.

```
$Auger-recombination                    optional

 material-name              character   required
 number-of-parameters       integer     required

 n-C                        double      optional  ! [cm^6/s] for electrons
 p-C                        double      optional  ! [cm^6/s] for holes

 n-bow-C                    double      optional  ! [cm^6/s] for electrons
 p-bow-C                    double      optional  ! [cm^6/s] for holes

$end_Auger-recombination                optional
```

$direct-recombination

Any values specified here in the input file overwrite the values specified in the material database.

Please refer to the database section of *$direct-recombination* for more details on the meaning of these parameters.

```
$direct-recombination                   optional

 material-name              character   required
 number-of-parameters       integer     required

 C-opt                      double      optional  ! [cm^3/s]
 bow-C-opt                  double      optional  ! [cm^3/s]

$end_direct-recombination               optional
```

$SRH-recombination

Any values specified here in the input file overwrite the values specified in the material database.

Please refer to the database section of *$SRH-recombination* for more details on the meaning of these parameters.

```
$SRH-recombination                      optional

 material-name              character   required
 number-of-parameters       integer     required

 n-N-ref                    double      optional ! [cm^-3] for electrons
```

(continues on next page)

(continued from previous page)

```
 n-tau                      double      optional ! [s]      for electrons

 p-N-ref                    double      optional ! [cm^-3] for holes
 p-tau                      double      optional ! [s]      for holes

 n-bow-N-ref                double      optional ! [cm^-3] for electrons
 n-bow-tau                  double      optional ! [s]      for electrons

 p-bow-N-ref                double      optional ! [cm^-3] for holes
 p-bow-tau                  double      optional ! [s]      for holes

$end_SRH-recombination                  optional
```

$mobility-model-constant

See also explanations under section database ==> *$mobility-model-constant*.

The only difference is that `number-of-parameters` is `optional` instead of `required`.

```
$mobility-model-constant                optional
 ...
 number-of-parameters          integer   optional
 ...
$end_mobility-model-constant            optional
```

$mobility-model-minimos

See also explanations under section database ==> *$mobility-model-minimos*.

The only difference is that `number-of-parameters` is `optional` instead of `required`.

```
$mobility-model-minimos                 optional
 ...
 number-of-parameters          integer   optional
 ...
$end_mobility-model-minimos             optional
```

$mobility-model-simba

See also explanations under section database ==> *$mobility-model-simba*.

The only difference is that `number-of-parameters` is `optional` instead of `required`.

```
$mobility-model-simba                   optional
 ...
 number-of-parameters          integer   optional
 ...
$end_mobility-model-simba               optional
```

$mobility-model-arora

See also explanations under section database ==> *$mobility-model-arora*.

The only difference is that `number-of-parameters` is `optional` instead of `required`.

```
$mobility-model-arora                    optional
 ...
 number-of-parameters          integer    optional
 ...
$end_mobility-model-arora                optional
```

$mobility-model-dar

See also explanations under section database ==> *$mobility-model-dar*.

The only difference is that `number-of-parameters` is `optional` instead of `required`.

```
$mobility-model-dar                      optional
 ...
 number-of-parameters          integer    optional
 ...
$end_mobility-model-dar                  optional
```

$mobility-model-lom

See also explanations under section database ==> *$mobility-model-lom*.

The only difference is that `number-of-parameters` is `optional` instead of `required`.

```
$mobility-model-lom                      optional
 ...
 number-of-parameters          integer    optional
 ...
$end_mobility-model-lom                  optional
```

$mobility-model-masetti

See also explanations under section database ==> *$mobility-model-masetti*.

The only difference is that `number-of-parameters` is `optional` instead of `required`.

```
$mobility-model-masetti                  optional
 ...
 number-of-parameters          integer    optional
 ...
$end_mobility-model-masetti              optional
```

$optical-absorption

This keyword allows calculating optical absorption coefficient and solar cells.

```
$optical-absorption                                optional
 destination-directory              character      required
 calculate-optics                   character      optional
 kind-of-absorption                 character      optional
 read-in-k-points                   character      optional

 num-quantum-cluster                integer        optional
 e-min-state                        double         optional
 e-max-state                        double         optional

 e-min-photon                       double         optional
 e-max-photon                       double         optional
 num-energy-steps                   integer        optional
 smoothing-of-curve                 character      optional
 smoothing-damping-parameter        double         optional
 E-P                                double         optional
 polarization-vector-1              double_array   optional
 polarization-vector-2              double_array   optional
 magnitude-relation-1-2             double         optional
 phase                              double         optional

 fermi_in_el                        double         optional
 fermi_in_hl                        double         optional
 device_thickness_in                double         optional

 k-space-symmetry                   character      optional

 !-------------------------------------------------------------
 ! The following are only relevant for solar cell simulations.
 !-------------------------------------------------------------

 incident-light-along-direction     character      optional

 import-absorption-spectrum         character      optional
 file-absorption-spectrum           character      optional

 import-reflectivity-spectrum       character      optional
 file-reflectivity-spectrum         character      optional
```

(continues on next page)

(continued from previous page)

```
 import-transmission-spectrum     character     optional
 file-transmission-spectrum       character     optional

 import-solar-spectrum            character     optional
 file-solar-spectrum              character     optional

 number-of-suns                   double        optional

 calculate-black-body-spectrum    character     optional
$end_optical-absorption                         optional
```

A tutorial is available that describes this keyword: Optical absorption of an InGaAs quantum well

Detailed description about the Physics: Absorption, Matrix elements, Inter-band transitions, Intra-band transitions (pdf).

destination-directory

type
character

presence
required

example
`optics/`

Directory for output of data files.

Optical absorption

calculate-optics

type
character

options
`yes` or `no`

default
`no`

Choose `yes` if you want to calculate the optical absorption spectrum (Step 3).

This flag can be set to `no` for Step 1 and Step 2, and `yes` for Step 3 (*see below*).

num-quantum-cluster

type
integer >= 1

default
1

Number of quantum cluster for which absorption spectrum is calculated. If this specifier is not present, the quantum cluster 1 is taken.

kind-of-absorption

type
character

options
`interband-only` `intra-vb-only` `intra-cb-only` `intra-sg-only` `inter-sg-only`

- `interband-only`

  Considers only interband transitions between holes and electrons.

  - heavy hole <==> Gamma band
  - light hole <==> Gamma band
  - split-off hole <==> Gamma band

- `intra-vb-only`

  Considers only intraband transitions within the valence bands.

  - heavy hole <==> light hole
  - heavy hole <==> split-off hole
  - light hole <==> split-off hole
  - heavy hole <==> heavy hole
  - light hole <==> light hole
  - split-off hole <==> split-off hole

- `intra-cb-only`

  Considers only intraband transitions within the conduction band (Gamma band).

  - Gamma band <==> Gamma band

- `intra-sg-only`

  Considers only intraband transitions within the same band (single-band for Gamma, L, X, heavy hole, light hole, split-off hole band)

  - Gamma band <==> Gamma band
  - L band <==> L band
  - X band <==> X band
  - heavy hole <==> heavy hole
  - light hole <==> light hole
  - split-off hole <==> split-off hole

  This is a simple algorithm taking only account the energy levels and wave functions at $k_{||} = 0$ (for single-band case). It only works for 1D and 2D simulations so far. It can also be used for the $\mathbf{k} \cdot \mathbf{p}$ wave functions as shown in this tutorial: *Intersubband transitions in InGaAs/AlInAs multiple quantum well systems*. In this case, the correct $\mathbf{k} \cdot \mathbf{p}$ density and k-dependent matrix elements and nonparabolicity and anisotrpicity of the energy dispersion $E(k_x, k_y)$ is taken into account.

  In order for this flag to work, the following must be present in the input file (*$output-1-band-schroedinger*):

```
$output-1-band-schroedinger
 ...
 intraband-matrixelements = yes  ! or any other value apart from 'no'
```

  The equation used is described here: *Intraband absorption in the single-band case*

- `inter-sg-only`

  Similar as `intra-sg-only` but for interband transitions. It currently does not work for $\mathbf{k} \cdot \mathbf{p}$ wave functions.

Specifications for energy range of absorption spectrum: lower/upper boundary for photon energy interval

e-min-photon

type
double

unit
[eV]

example
1.0

lower boundary for photon energy

e-max-photon

type
double

unit
[eV]

example
2.0

upper boundary for photon energy

num-energy-steps

type
integer

example
1000

Number of energy steps between `e-min-photon` and `e-max-photon`.

This number determines the resolution of the absorption spectrum curve $\alpha(E)$ where E is the energy in units of `[eV]`.

Note: The number of

energy grid points = `num-energy-steps` $+1$

because the first grid point (`e-min-photon`) is also included.

The energy grid spacing is

$\Delta_E =$ (`e-max-photon` $-$ `e-max-photon`)/ `num-energy-steps`.

Distinguish between calculating and reading in $k_{||}$ points.

read-in-k-points

type
character

options
`yes` or `no`

default
`no`

Flag to distinguish between Step 2 and Step 3. For Step 3, in order to avoid calculating the $k_{||}$ points again, one can simply read them in from a previous simulation, and then calculate and output the optical absorption spectrum.

Energy of lowest/highest eigenvalue considered for calculation, i.e. energies are calculated in the interval `[e-min-state,e-max-state]`. Here, the eigenvalue solver is called with these energy values. Alternatively, the eigenvalue solver can be called with a certain number of eigenvalues requested, i.e. one either has to specify

an energy interval or the number of eigenvalues. Depending on the eigenvalue solver used, different options are possible.

e-min-state

type
double

unit
`[eV]`

default
-5.0

example
-1.7

lowest eigenvalue

e-max-state

type
double

unit
`[eV]`

default
5.0

example
0.3

highest eigenvalue

Broadening of absorption curve

This is only relevant for the calculation of the absorption spectrum.

smoothing-of-curve

type
character

options
`Lorentzian`, `Gaussian`, `yes`, `no`

default
`yes`

For `Lorentzian` or `Gaussian`, we introduce an artificial broadening (smoothing) of the curve (*Lorentzian* or *Gaussian* broadening). If `yes`, both *Lorentzian* and *Gaussian* broadening will be calculated and written out. If `no`, no broadening (smoothing) of the curve is assumed.

smoothing-damping-parameter

type
double > 0.0

unit
`[eV]`

default
0.005

The artificial parameter for smoothing of absorption spectrum is `smoothing-damping-parameter`. It is usually denoted as Γ and is the *Full Width at Half Maximum* (**FWHM**).

Lorentzian lineshape

The Lorentzian function is given by

$$L(E) = \frac{1}{\pi} \frac{\Gamma/2}{(E-E_{ij})^2+(\Gamma/2)^2}$$

where $\Gamma/2$ is the scale parameter *Lorentzian half-width*, i.e. Half Width at Half Maximum (**HWHM**). It describes the shape of certain types of spectral lines (lineshape). Note that the definition of the Lorentzian function includes a factor $1/\pi$.

- E_{ij} is the transition energy between the states i and j and specifies the location of the peak in the Lorentzian function.
- Γ is specified in the input file via `smoothing-damping-parameter`. It is the *Full Width at Half Maximum* (**FWHM**).
- $\alpha = \Gamma/2$ is the *Half Width at Half Maximum* (**HWHM**).

```
!-------------------------------------------
! for k.p algorithm only:
! First, the absorption spectrum is calculated.
! Then the broadening is applied.
!-------------------------------------------

absorption_NoSmoothingV = absorptionV
absorptionV = 0.0
DO  i=1,num-energy-steps+1 ! Loop over all energy grid points E(i) and␣
↪determine absorption coefficients alpha(i)=alpha(E).
 DO j=1,num-energy-steps+1 ! This loop is essentially an integration␣
↪over energy dE.

    E_weight = (Lorentzian( E_gridV(j),   E_gridV(i), smoothing-damping-
↪parameter ) - &
                Lorentzian( E_gridV(j), - E_gridV(i), smoothing-damping-
↪parameter ) ) * DeltaEnergy

    absorptionV(i) = absorptionV(i) + absorption_NoSmoothingV(j) * E_
↪weight

 END DO
END DO
```

The following specifiers are only used for the $\mathbf{k} \cdot \mathbf{p}$ optical absorption spectrum but not for the simple single-band intersubband absorption algorithm.

E_P

type
double

example
`20.0`

unit
`[eV]`

status
currently not implemented, value from the database is used

E_P is Kane's matrix element $E_P = | < S|p|X > |^2$. It should be around 20 eV and depends on the material. The `E_P` parameter is given in the database by the specifier `8x8kp-parameters`. E_P can be converted into the P paramter by the following equation: $E_P = \frac{2m_0}{\hbar^2} P^2$. In our model the E_P parameter is only relevant for interband transitions. It enters into the matrix element prefector (*matrix_element_prefac*) which is described in Section *1.1.1 Inter-band transitions* of the documentation: Absorption spectrum, Matrix elements, Inter-band transitions, Intra-band transitions (pdf). E_P has the same value for all materials in this implementation. In principle it could have been read in from the database rather than specifying it within the keyword `$optical-absorption`.

Polarization

m is equivalent to $\tan(\theta)$ or $\tan^{-1}(\theta)$ depending whether P_1/P_2 is x or y. so instead of θ one can directly use the formula below:

$P = m \cdot P_1 + e^{\mathrm{i}\omega\pi} \cdot P_2$

This is in fact more general, as it also describes circular polarization which lead to complex coefficient.

polarization-vector-1

type
double array

example
`1.0 0.0 0.0`

x y z coordinates (in simulation system) for first in-plane vector P_1

polarization-vector-2

type
double array

example
`0.0 1.0 0.0`

x y z coordinates (in simulation system) for second in-plane vector P_2

magnitude-relation-1-2

type
double

unit
`[]`

default
`0.5`

example
`1.0`

relation of magnitudes $m = |E_1|/|E_2|$

phase

type
double

unit
`[]`

default
`0.0`

example
`0.5`

phase ω: E_2 ==> $\exp(\mathrm{i}\omega\pi)E_2$

$P = m \cdot P_1 + e^{\mathrm{i}\omega\pi} \cdot P_2$

Examples

- x-polarized light

```
polarization-vector-1  = 0.0  1.0  0.0
polarization-vector-2  = 1.0  0.0  0.0
magnitude-relation-1-2 = 0.0
```

In this case, `polarization-vector-1` is ignored as $|E_1|$ is set to be zero.

- z-polarized light

```
polarization-vector-1  = 1.0  0.0  0.0
polarization-vector-2  = 0.0  0.0  1.0
magnitude-relation-1-2 = 0.0
```

In this case, `polarization-vector-1` is ignored as $|E_1|$ is set to be zero.

- circularly polarized light in the (x,y) plane

```
polarization-vector-1  = 1.0  0.0  0.0
polarization-vector-2  = 0.0  0.0  1.0
magnitude-relation-1-2 = 1.0
```

In this case, `polarization-vector-1` is *not* ignored as $|E_1| = |E_2|$.

- quantum well, interband absorption

```
polarization-vector-1  = 1.0  0.0  0.0
polarization-vector-2  = 0.0  1.0  0.0
magnitude-relation-1-2 = 1.0
```

- quantum well, intraband absorption

```
polarization-vector-1  = 0.0  0.0  1.0
polarization-vector-2  = 0.0  0.0  1.0
magnitude-relation-1-2 = 0.0
```

Note: Intraband absorption spectrum only for z-polarized light.

Fermi levels

fermi_in_el

type
double

unit
`[eV]`

default
`0.0`

example
`0.1`

Optional input for Fermi level of electrons (default: calculated quasi-Fermi level for electrons)

fermi_in_hl

type
double

unit
`[eV]`

default
`0.0`

example
`-1.0`

Optional input for Fermi level of holes (default: calculated quasi-Fermi level for holes)

device-thickness

type
double

unit
[m]

default
thickness of device or quantum cluster? (*Check this!*)

example
1e-6

Optional input of device thickness for normalization of absorption spectrum

k-space-symmetry

type
character

options
default, none, four-fold

default
default

Symmetry of $k_{||} = (k_x, k_y)$ space to be discretized. If any symmetry is present, less $k_{||}$ points have to be calculated. By default, the appropriate symmetry is chosen taking into account any crystal rotations with respect to the simulation axes, as well as nonsymmetric strains.

Note: In order to save CPU time, we recommend the following procedure:

Instead of calculating

- $\mathbf{k} \cdot \mathbf{p}$ eigenstates and
- optical absorption spectrum

within one simulation, it is more efficient to divide this into 3 steps.

1. Step 1: Calculate eigenstates for $k_{||} = 0$.

```
calculate-optics = no
```

   This is very quick. Solve $\mathbf{k} \cdot \mathbf{p}$ to determine lowest and highest eigenvalue so that we know what to specify for e-min-state and e-max-state in Step 2.

2. Step 2: Calculate eigenstates for all $k_{||}$ vectors and save all of the wave functions to file so that they can be read in and used many times in Step 3.

```
raw-potential-in    = yes
raw-fermi-levels-in = yes
strain-calculation  = raw-strain-in

num-kp-parallel     = 1700  ! STEP 2/3   ! total number of k_|| points for␣
↪Brillouin zone discretization

calculate-optics    = yes
num-quantum-cluster = 1
read-in-k-points    = no
e-min-state         = -1.7  ! Choose a reasonable value for E_min
e-max-state         =  0.3  ! Choose a reasonable value for E_min
```

   Read in raw data (potential, quasi-Fermi levels, strain (if applicable) and all $\mathbf{k} \cdot \mathbf{p}$ wave functions) and output $k_{||}$ points. The strain calculation is very fast for a 1D simulation. Instead of reading it in, it could be directly calculated (*recommended*).

The user specifies the total number of $k_{||}$ points that are present in $k_{||}$ space. However, internally the code modifies this number according to the following algorithm:

- number of k points in positive x direction (without Gamma point): N_kx
- number of k points in positive y direction (without Gamma point): N_ky = N_kx

==> Thus the actual, total number of $k_{||}$ points is:

```
total_number_of_k|| = (2 * N_kx + 1) * (2 * N_ky + 1)
```

In this example (`num-kp-parallel = 1700`):

```
N_kx = N_ky = 20
```

```
==> total_number_of_k|| = 41 * 41 = 1681
```

3. Step 3: Calculate optical absorption spectrum.

```
calculate-optics = yes
read-in-k-points = yes
```

Read in $k_{||}$ points, calculate and output optical absorption spectrum for specific polarization of incident light.

If one wants to repeat the calculation for another polarization, one only needs to change the polarization vector and repeat Step 3. It is not necessary in this case to recalculate Step 1 or Step 2. Step 3 also outputs the energy dispersion $E(k_{||}) = E(k_x, k_y)$. (*Check: Why not Step 2?*)

Output

Results

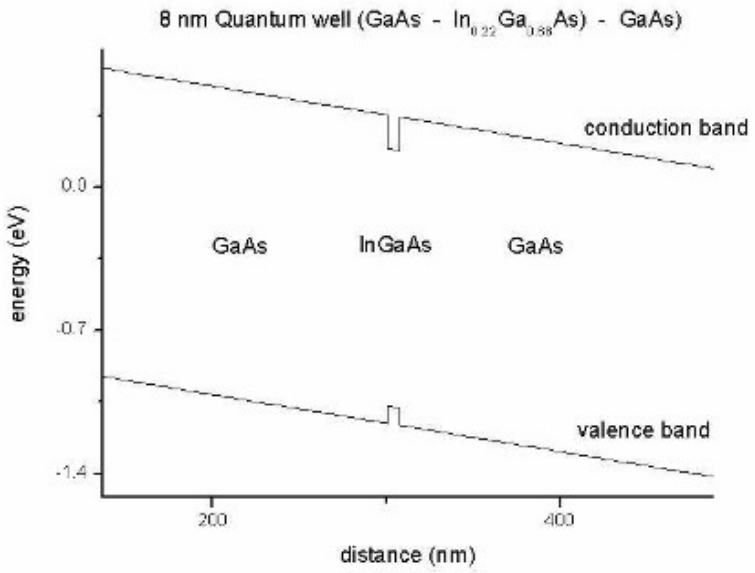


Figure 7.2.3.1: Conduction and valence band edges of the quantum well

The unit of the optical absorption coefficient is `[m^-1]` and not *arbitrary units* as indicated in the figure.

The *electric susceptibility tensor* χ is contained in the file `susceptibility_tensor.dat`:

```
chi11re  chi11im  chi22re  chi22im  chi33re  chi33im  chi12re  chi12im ␣
↪chi13re  chi13im  chi23re  chi23im
```

Note: As this tensor is complex, for each component, two values are written out.

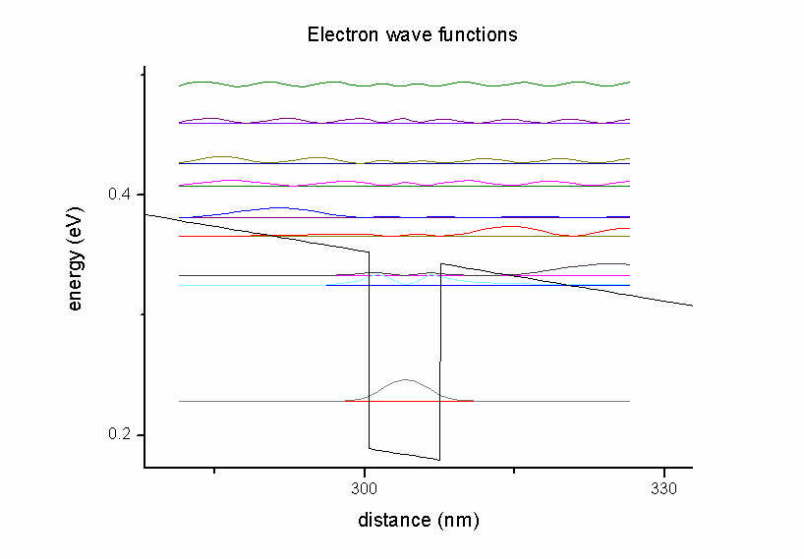


Figure 7.2.3.2: Electron states in the quantum well

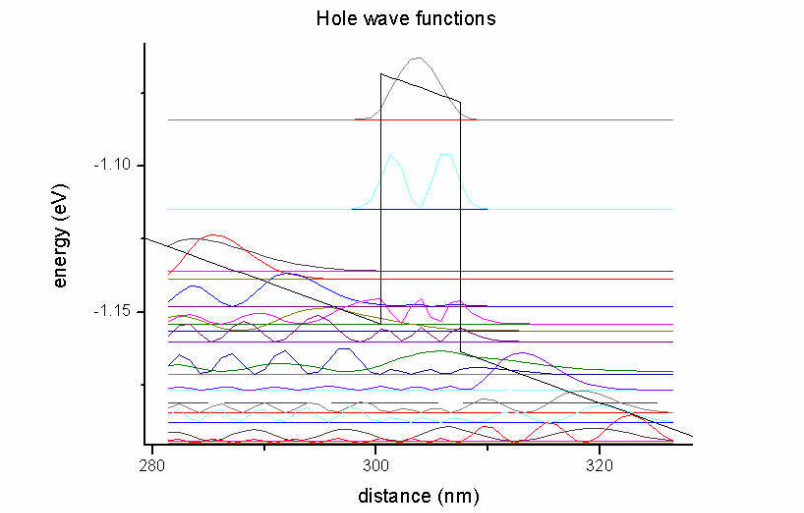


Figure 7.2.3.3: Hole states in the quantum well

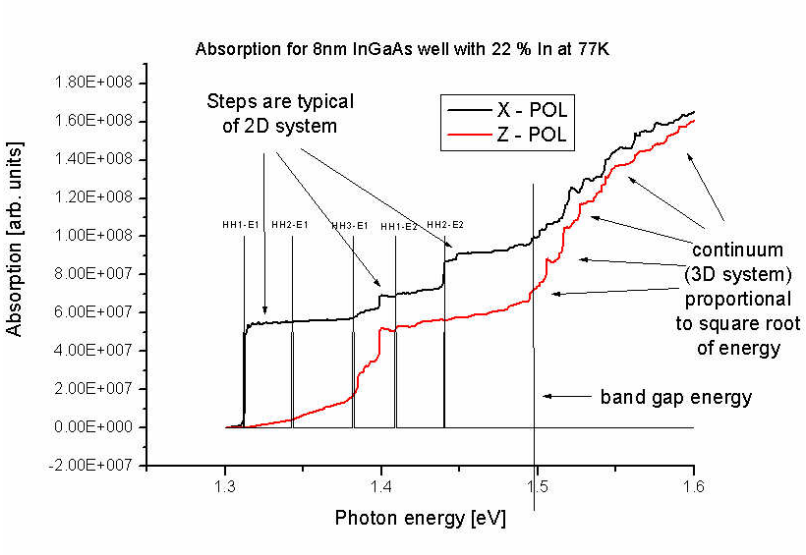


Figure 7.2.3.4: Optical absorption spectrum of the quantum well

- `re`: real part
- `im`: imaginary part

The relevant part for the absorption spectrum is only the *imaginary part*.

The units of the $k_{||}$ space grid coordinates k_x and k_y are `[Angstrom^-1]` and the energy units are `[eV]`.

The files

- `el_dispersion_100.dat`
- `el_dispersion_110.dat`
- `hl_dispersion_100.dat`
- `hl_dispersion_110.dat`

show the same data but with slices along the

- [10] (i.e. $k_{||} = (k_x, k_y = 0)$ and
- [11] (i.e. $k_{||} = (k_x = k_y)$ directions in $k_{||}$ space.

Here all electron and all hole eigenvalues are contained in one file, respectively.

Restrictions

- Only *Dirichlet* boundary conditions are supported so far.
- Step 2 and Step 3 only work if:

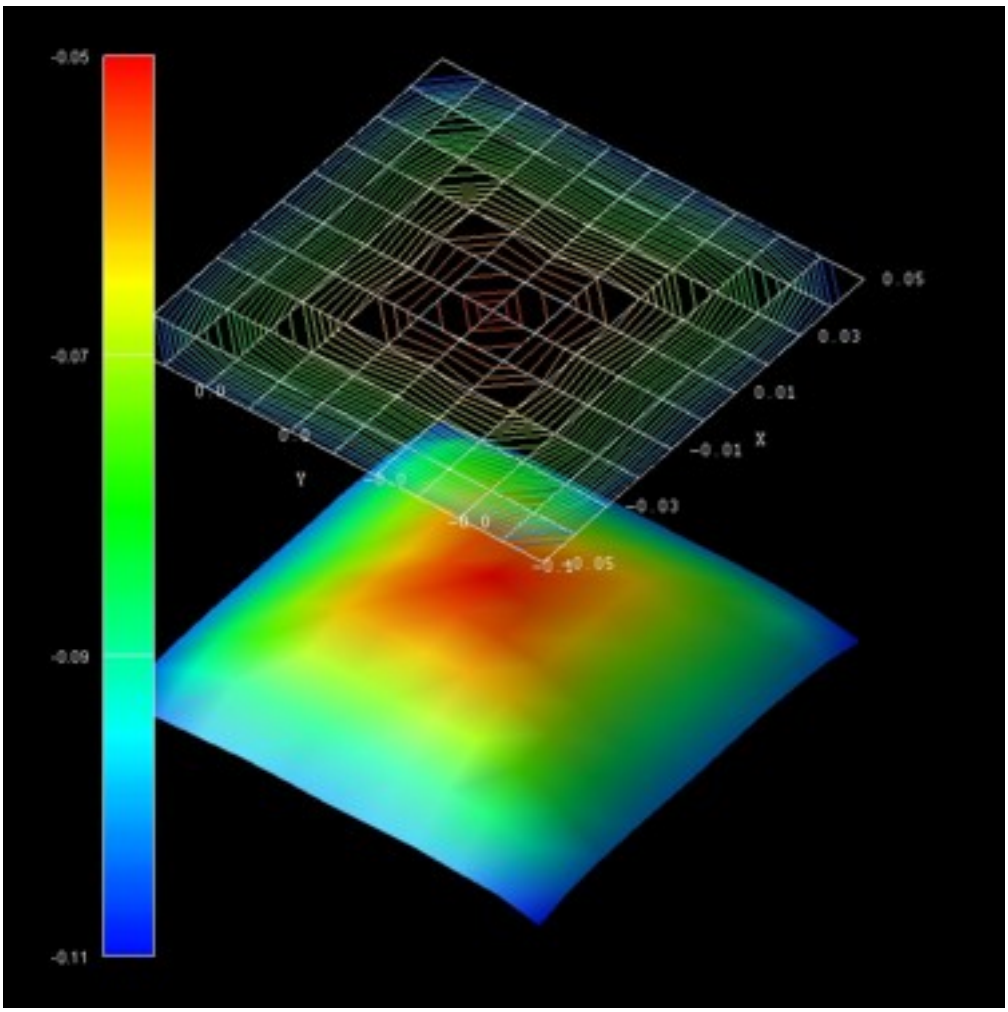


Figure 7.2.3.5: Energy dispersion $E(k_x, k_y)$ of the highest hole eigenstate (ground state).

```
raw-potential-in = yes
```

Solar cells

For solar cells, we have this tutorial: *GaAs Solar Cell*

Example files for solar spectra, absorption coefficient, transmission and reflectivity coefficients can be found in the installation folder:

- `C:\Program Files\nextnano\nextnano3\Syntax\Solar cell files\absorption\`
- `C:\Program Files\nextnano\nextnano3\Syntax\Solar cell files\reflectivity`
- `C:\Program Files\nextnano\nextnano3\Syntax\Solar cell files\solar spectra\`

The following specifiers are relevant for solar cell simulations (photovoltaics).

incident-light-along-direction

type
character

options
`x`, `y`, `z`, `-x`, `-y`, `-z`

default
along simulation direction in 1D

In a 1D simulation, this specifier is optional. For 2D and 3D, a direction must be specified.

Solar spectrum

import-solar-spectrum

type
character

options
`yes` or `no`

default
`no`

For a solar cell simulation, one has to read in a solar spectrum, e.g. AM 1.5, or AM 1.0 (AM = air mass). They can be obtained from NREL website, e.g. ASTM-E490: https://www.nrel.gov/grid/solar-resource/spectra-astm-e490.html (AMST = American Society for Testing and Materials)

file-solar-spectrum

type
character

example
`H:\solar_cells\ASTMG173_AM10.dat` AM 1.0 spectrum (extraterrestrial)

example
`H:\solar_cells\ASTMG173_AM15.dat` AM 1.5 spectrum

example
`H:\solar_cells\ASTMG173_AM15G.dat` AM 1.5G spectrum (G = global, i.e. including diffuse light)

The file must consist of two columns (wavelength and spectrum), the units are `[nm]` and `[W/m^2*nm^-1]`.

```
wavelength[nm]    AM1.5[W/m^2*nm^-1]
...               ...
```

Concentration of sun light

number-of-suns

type
integer

default
`1.0` our sun

example
`0.0` (no sun, dark)

example
`2.5` 2.5 suns

example
`300.0` 300 suns

example
`1000.0` 100 suns

The number of suns can be set to increase the power of the solar spectrum in order to model concentrator solar cells.

Absorption Spectra

import-absorption-spectrum

type
character

options
`yes` or `no`

default
`no`

file-absorption-spectrum

type
character

example
`AbsorptionCoefficient_GaAs_300K.dat`

The file must consist of two columns (wavelength and absorption coefficient), the units are `[nm]` and `[cm^-1]`.

```
wavelength[nm]    absorption[1/cm]
...               ...
```

Reflection coefficient

Fraction of incident photons that are reflected from surface for a particular wavelength.

import-reflectivity-spectrum

type
character

options
`yes` or `no`

default
`no`

file-reflectivity-spectrum

type
character

example
`ReflectionCoefficient_GaAs_300K.dat`

The file must consist of two columns (wavelength and reflection coefficient), the units are `[nm]` and `[]`.

```
wavelength[nm]    reflectivity[]
...               ...
```

Transmission coefficient

Fraction of incident photons that are transmitted through the device for a particular wavelength (relevant for very thin devices).

import-transmission-spectrum

type
character

options
`yes` or `no`

default
`no`

file-transmission-spectrum

type
character

example
`TransmissionCoefficient.dat`

The file must consist of two columns (wavelength and transmission coefficient), the units are `[nm]` and `[]`.

```
wavelength[nm]    transmission[]
...               ...
```

Solar cell output

All output is twofold:

- one is with respect to wavelength in units of `[nm]`
- one is with respect to photon energy in units of `[eV]` (indicated by `_eV*.dat`)

The files are:

- Absorption coefficient
  - `optics/Absorption_coefficient.dat` (as read in from file but now in units of `[m^-1]`)
  - `optics/Absorption_coefficient_interpolated.dat` (interpolated on wavelength grid of solar spectrum but now in units of `[m^-1]`)
- Reflectivity
  - `optics/Reflectivity.dat` (as read in from file)
  - `optics/Reflectivity_interpolated.dat` (interpolated on wavelength grid of solar spectrum)
- Transmission
  - `optics/Transmission.dat` (as read in from file)
  - `optics/Transmission_interpolated.dat` (interpolated on wavelength grid of solar spectrum)
- Solar spectrum
  - `optics/SolarSpectralIrradiance_sun0001.dat` (as read in from file)
  - `optics/PhotonFlux_sun0001.dat` (photon flux density calculated from solar spectrum)
- Total number of of photons in the solar spectrum above an energy value contributing to the maximum photocurrent for a solar cell made with a specific band gap:
  - `optics/PhotonFlux_BandGap_eV_sun0001.dat` (calculated from solar spectrum)

  - `optics/PhotoCurrent_BandGap_eV_sun0001.dat` (calculated from solar spectrum)
- Spectral response
  - `optics/SpectralResponse_sun0001.dat` external and internal spectral response
- Quantum efficiency
  - `optics/QuantumEfficiency_sun0001.dat` external and internal quantum efficiency
- Generation rate
  - `optics/GenerationRateLight_AVS_sun0001.fld` 2D plots $G(x, \lambda)$ and $G(x, E)$
  - `optics/GenerationRateLight_sun0001.dat` 1D plot $G(x)$
  - `optics/GenerationRate_eV_sun0001.dat` 1D plot $G(E)$ where E is the energy
  - `optics/GenerationRate_Wavelength_sun0001.dat` 1D plot $G(\lambda)$
- Current-voltage characteristics
  - `current/IV_characteristics_new.dat`

```
voltage[V]  current[A/m^2] ... power[W/m^2] powersolar[W/m^2] efficiency[%]
```

- The following information can be found in the `.log` file, such as
  - short-circuit current I_{sc}
  - open-circuit voltage U_{oc}
  - ideal conversion efficiency η
  - ...

```
*******************************************************************************************
Solar cell results
*******************************************************************************************
short-circuit current:       I_sc    = 281.473346 [A/m^2]                ␣
↪(photo current: It increases with smaller band gap.)
open-circuit voltage:        U_oc    =  -1.012500 [V]                     (U_oc
↪<= built-in potential ~ band gap)
current at maximum power:    I_max   = 273.089897 [A/m^2]
voltage at maximum power:    U_max   =  -0.925000 [V]
maximum power output:        P_max   = U_max * I_max = -252.608155 [W/m^2]␣
↪(condition for maximum power output: dP/dV = 0)
maximum extracted power:     P_solar = - P_max       =  252.608155 [W/m^2]
incident power:              P_in    =   0.000000 [W/m^2]
ideal conversion efficiency: eta     = P_max / P_in  = Infinity %
fill factor:                 FF      =   0.886370
In practice, a good fill factor is around 0.8.
All these results are approximations.
They are only correct if a lot of voltage steps have been used (i.e. a high␣
↪resolution).
*******************************************************************************************
```

Example for a solar cell simulation

```
!---------------------------------
$optical-absorption
 destination-directory        = optics/

 import-absorption-spectrum   = yes
 file-absorption-spectrum     = "..\Syntax\Solar cell files\absorption\
```

(continues on next page)

(continued from previous page)

```
↪AbsorptionCoefficient_GaAs_300K.dat"

 import-reflectivity-spectrum = yes
 file-reflectivity-spectrum   = "..\Syntax\Solar cell files\reflectivity\
↪Reflectivity_Al0.80Ga0.20As.dat"

 import-solar-spectrum        = yes
 file-solar-spectrum          = "..\Syntax\Solar cell files\solar spectra\
↪ASTMG173_AM15G.dat"

 number-of-suns               = 1

$end_optical-absorption
!---------------------------------------
```

Black body spectrum

calculate-black-body-spectrum

type
character

options
yes or no

default
no

Flag for calculating *black body spectrum* according to Planck's law, e.g. to compare the solar spectrum to the spectrum of a black body at T = 5778 K.

- The spectral *energy density*
- the spectral *radiance* (which is *emitted* per m^2 and per unit solid angle sr (sr = steradian)) and
- the spectral *irradiance* (which is *received* per m^2)

is calculated.

Note: spectral irradiance = spectral radiance $\cdot\pi$

spectral energy density = spectral radiance $\cdot 4\pi/c$

There are several output files, i.e. output with respect to

- *wavelength* λ in units of `[m]`,
  - `BlackBody_SpectralEnergyDensity_wavelength.dat`

    ```
    Wavelength[nm]                    SpectralEnergyDensity[kJ/m^3/m]
    ```

  - `BlackBody_SpectralRadiance_wavelength.dat`

    ```
    Wavelength[nm]                    SpectralRadiance[kW/m^2/nm/sr]
    ```

  - `BlackBody_SpectralIrradiance_wavelength.dat`

    ```
    Wavelength[nm]                    SpectralIrradiance[kW/m^2/nm]
    ```

- *angular frequency* $\omega = 2\pi\nu$ in units of `[1/s]`,

  - `BlackBody_SpectralEnergyDensity_angular_frequency.dat`

AngularFrequency_omega[10^15/s]	SpectralEnergyDensity[10^-15J/m^3/s^-1]

  - `BlackBody_SpectralRadiance_angular_frequency.dat`

AngularFrequency_omega[10^15/s]	SpectralRadiance[10^-12W/m^2/s^-1/sr]

  - `BlackBody_SpectralIrradiance_angular_frequency.dat`

AngularFrequency_omega[10^15/s]	SpectralIrradiance[10^-12W/m^2/s^-1]

- *frequency* ν in units of `[Hz]`,
  - `BlackBody_SpectralEnergyDensity_frequency.dat`

Frequency[THz]	SpectralEnergyDensity[10^-15J/m^3/Hz]

  - `BlackBody_SpectralRadiance_frequency.dat`

Frequency[THz]	SpectralRadiance[10^-12W/m^2/sr]

  - `BlackBody_SpectralIrradiance_frequency.dat`

Frequency[THz]	SpectralIrradiance[10^-12W/m^2/Hz]

- *photon energy* $E = h\nu$ in units of `[eV]`.
  - `BlackBody_SpectralEnergyDensity_energy.dat`

AngularFrequency_omega[10^15/s]	SpectralEnergyDensity[kJ/m^3/eV]

  - `BlackBody_SpectralRadiance_energy.dat`

AngularFrequency_omega[10^15/s]	SpectralRadiance[kW/m^2/eV/sr]

  - `BlackBody_SpectralIrradiance_energy.dat`

AngularFrequency_omega[10^15/s]	SpectralIrradiance[kW/m^2/eV]

The file `BlackBody_Info.txt` contains some additional information about the calculated black body spectrum.

Intraband absorption in the single-band case

In the following we assume a single band with a parabolic energy band dispersion.

Tutorials showing results are available here:

- *Intersubband absorption of an infinite quantum well*
- *Intersubband transitions in InGaAs/AlInAs multiple quantum well systems*

For a 1D heterostructure grown along the x direction, formula for the *absorption coefficient* α reads (see e.g. *[ChuangOpto1995]* or p. 53 in *[FaistQCL2013]*)

$$\alpha(\omega) = \frac{e^2\omega}{\varepsilon_0 n_r c} \sum_i \sum_j (\overline{n}_i - \overline{n}_j)\, x_{ij}^2 \frac{\Gamma/2}{(E_j - E_i - \hbar\omega)^2 + (\Gamma/2)^2}$$

or, equivalently in energy,

$$\alpha(E) = \frac{e^2 E}{\hbar\varepsilon_0 n_r c} \sum_i \sum_j (\overline{n}_i - \overline{n}_j)\, x_{ij}^2 \frac{\Gamma/2}{(E_j - E_i - \hbar\omega)^2 + (\Gamma/2)^2}$$

where

- $\omega = E/\hbar$ is the frequency in units of `[s^-1]`

- E the energy in units of `[J]`
- e is the elementary charge in units of `[As]`
- ε_0 is the vacuum permittivity in units of `[As/Vm]`
- c is speed of light in vacuum in units of `[m/s]`
- $n_\mathrm{r} = \sqrt{\varepsilon_\mathrm{r}}$ is the refractive index (`[]`) assumed to be homogeneous. So we take the average of the quantum region (*check this*).
- $\overline{n}_i = \frac{1}{L}\sigma_i = \frac{1}{L}\int n_i(x)\mathrm{d}x$ is the *averaged* electron density of subband i in units of `[m^-3]`, where L is the length of the quantum region and $\sigma_i = \int n_i(x)\mathrm{d}x$ is the subband density in units of `[m^-2]`
- $x_{ij} =< i|x|j >$ is the dipole moment between initial state i and final state j in units of `[m]`
- Γ is the energy linewidth (broadening) in units of `[J]` in terms of full-width at half maximum (FWHM).

This equation includes a *Lorentzian broadening* which includes a factor of $1/\pi$.

We can also define the *position dependent absorption coefficient*

$$\alpha(\omega, x) = \frac{e^2\omega}{\varepsilon_0 n_\mathrm{r} c} \sum_i \sum_j \left(n_i(x) - n_j(x)\right) x_{ij}^2 \frac{\Gamma/2}{(E_j - E_i - \hbar\omega)^2 + (\Gamma/2)^2}$$

where

- $n_i(x)$ is the electron density of state i at position x in units of `[m^-3]`.

The units of both $\alpha(\omega)$ and $\alpha(\omega, x)$ are `[m^-1]`. In plots, typically `[cm^-1]` is used.

If we integrate $\alpha(\omega, x)$ over position x in the whole quantum region of length L, and divide by the length of the quantum region L, we obtain $\alpha(\omega)$ as defined above,

$$\alpha(\omega) = \frac{1}{L}\int \alpha(\omega, x)\mathrm{d}x.$$

So $\alpha(\omega)$ as defined in the beginning of this section, where we averaged the density $\overline{n}_i$, is the averaged absorption coefficient in the quantum region and equivalent to the definition given here.

Finally, we note that this also works for the $\mathbf{k}\cdot\mathbf{p}$ wave functions:

$$\alpha(\omega, x, k_\parallel) = \frac{e^2\omega}{\varepsilon_0 n_\mathrm{r} c} \sum_i \sum_j \sum_{k\parallel} \left(n_i(x, k_\parallel) - n_j(x, k_\parallel)\right) \left(x_{ij}(k_\parallel)\right)^2 \frac{\Gamma/2}{\left(E_j(k_\parallel) - E_i(k_\parallel) - \hbar\omega\right)^2 + (\Gamma/2)^2}$$

where

- $n_i(x, k_\parallel)$ is the electron density of state i at position x and vector $k_\parallel = (k_x, k_y)$ (in 1D) or $k_\parallel = k_z$ (in 2D).

$buffer-solutions

Buffer solutions: To control pH values in electrolytes.

It is possible to overwrite the database entries of $buffer-solutions by this keyword and its specifiers in the input file. You can also define a completely new buffer which is not contained in the database. If you overwrite a buffer that is contained in the database, it is necessary that you include *all* specifiers, including the ones that you do not want to overwrite. The program then uses all the values given in the input file and ignores all database entries.

```
$buffer-solutions                                optional
 buffer-name                 character           required
 number-of-ions              integer             required
 ion-valency                 double_array        required
 ion-name-1                  character           required
 ion-name-2                  character           required
 ion-name-3                  character           optional  ! only necessary for some␣
↪buffers, e.g. PBS
 ion-name-4                  character           optional  ! only necessary for some␣
↪buffers, e.g. PBS
 pKa                         double_array        required  ! pKa at 25° C ( = 298.15 K)
 dpKa_dT                     double_array        required  ! d pKa / d T
```

(continues on next page)

(continued from previous page)

```
 z_acid                  double_array      required  ! charge on the conjugate acid␣
↪species
$end_buffer-solutions                     optional
```

$\mathrm{p}K_\mathrm{a}$ and $\mathrm{d}\, pK_\mathrm{a}/\mathrm{d}\, T$.

Example 1: MOPS

```
!---------------------------------------!
$buffer-solutions                       !
 buffer-name     = MOPS                 ! MOPS  (C7H15NO4S)  +  NaOH
 number-of-ions  = 2                    !
 ion-valency     = -1.0          +1.0   !
              !   (C7H14NO4S)^-   Na^+   !
 ion-name-1      = Mops^-               ! C7H15NO4S <==> (C7H14NO4S)^- + H^+
 ion-name-2      = Na^+                 !
 pKa             = 7.31                 ! pKa at 25° C ( = 298.15 K)
                                        ! Note: This pKa is thermodynamic value.
                                        ! The working pKa' is 7.20.
 dpKa_dT         = -0.011               ! d pKa   / d T
 z_acid          = 0.0                  ! charge on the conjugate acid species (0 =␣
↪C7H15NO4S)
$end_buffer-solutions                   !
!---------------------------------------!
```

Involved ions and molecules:

- buffer-name: MOPS ($\mathrm{C_7H_{15}NO_4S}$) + NaOH
- ion-valency: $(\mathrm{C_7H_{14}NO_4S})^-$ and Na^+
- ion-name-1: $\mathrm{C_7H_{15}NO_4S}$ <==> $(\mathrm{C_7H_{14}NO_4S})^-$ + H^+
- ion-name-2: Na^+
- z_acid: 0 = $\mathrm{C_7H_{15}NO_4S}$

Example 2: PBS (phosphate buffer)

The phosphate buffer is special (and thus more complicated) because it consists of three $\mathrm{p}K_\mathrm{a}$ values (and it thus has four different ions).

```
!-------------------------------------------------------------!
$buffer-solutions                                             !
 buffer-name     = PBS                                        ! PBS (phosphate buffer)
 number-of-ions  = 4                                          !
 ion-valency     = -1.0        -2.0        -3.0       1.0     !
              !   (H2PO4)^-   (HPO4)^2-   (PO4)^3-   Na+     !
 ion-name-1      = H2PO4^-                                    ! NaH2PO4   <==>␣
↪(H2PO4)^- +   Na^+
 ion-name-2      = HPO4^2-                                    ! Na2HPO4   <==> (HPO4)^
↪2- + 2 Na^+
 ion-name-3      = PO4^3-                                     !
 ion-name-4      = Na^+                                       !
 pKa             = 2.15     7.21    12.33                     ! pKa,1    pKa,2    pKa,
↪3   at 25° C ( = 298.15 K)
 dpKa_dT         = 0.0044  -0.0028  -0.026                    ! d pKa   / d T
 z_acid          = 0.0        -1.0            -2.0            ! charge on the␣
↪conjugate acid species
              !   0 = H3PO4  -1 = (H2PO4)^-   -2 = (HPO4)^2-  !
```

(continues on next page)

(continued from previous page)

```
$end_buffer-solutions                                                    !
!-------------------------------------------------------------------!
```

Involved ions and molecules:

- `ion-valency`: $(H_2PO_4)^-$, $(HPO_4)^{2-}$, $(PO_4)^{3-}$ and Na^+
- `ion-name-1`: NaH_2PO_4 <==> $(H_2PO_4)^-$ + Na^+
- `ion-name-2`: Na_2HPO_4 <==> $(HPO_4)^{2-}$ + 2 Na^+
- `ion-name-3`: $(PO_4)^{3-}$
- `ion-name-4`: Na^+
- `z_acid`: `0` = H_3PO_4, `-1` = $(H_2PO_4)^-$, `-2` = $(HPO_4)^{2-}$

For the explanation of the specifiers, please also check the description in the database of the keyword $buffer-solutions .

$buffer-constant-A(T)

Constant $A(T)$ used for buffer calculations: The pK_a value depends on the ionic strength.

```
$buffer-constant-A(T)                                optional
 constant-Centigrade-to-Kelvin    double             required
 T_A(T)                           double_array       required
$end_buffer-constant-A(T)                            optional
```

Example 1

```
!-------------------------------------------------------------------!
$buffer-constant-A(T)                                                !
 constant-Centigrade-to-Kelvin = 273.15                              ! Kelvin = Celsius +␣
↪273.15
                                                                     !
 !===================================================================!
 !                  first column: T[C]   second column: A(T)        !
 !===================================================================!
 T_A(T)                          =   0.0          0.4918            !   0° C = 273.15 K
                                    10.0          0.4989            !  10° C = 283.15 K
                                    20.0          0.5070            !  20° C = 293.15 K
                                    25.0          0.5114            !  25° C = 298.15 K
                                    30.0          0.5161            !  30° C = 303.15 K
$end_buffer-constant-A(T)                                            !
!-------------------------------------------------------------------!
```

Example 2

```
!-------------------------------------------------------------------!
$buffer-constant-A(T)                                                !
 constant-Centigrade-to-Kelvin = 273.15                              ! Kelvin = Celsius +␣
↪273.15
                                                                     !
 !===================================================================!
 !                  first column: T[C]   second column: A(T)        !
 !===================================================================!
 T_A(T)                          =  25.0          0.5114            !  25° C = 298.15 K
$end_buffer-constant-A(T)                                            !
!-------------------------------------------------------------------!
```

If the keyword `$buffer-constant-A(T)` is present in the input file, the values for the keyword *$buffer-constant-A(T)* in the database are overwritten.

It is possible to specify only one pair of "T, A(T)" values as shown in Example 2.

For more details on the physical significance of this keyword, please have a look at the database section: *$buffer-constant-A(T)*

$tighten

Tight-binding

The original *tighten* code that is used inside *nextnano*[3] has been written by Peter Vogl, Walter Schottky Institute, Technische Universität München. It is based on the $sp^3d^5s^*$ method. For details about this method, see e.g. Ref. *[JancuPRB1998]*.

Here is a general template for the tighten code block for the input file, in which the variables should be specified as given in the explanations and examples further below.

```
!-------------------------------------------------------------!
$tighten                                      optional  !
 calculate-tight-binding-tighten    character     required  !
 tighten-method                     character     optional  !
 tight-binding-model                character     optional  !
 destination-directory              character     required  !
 input-directory                    character     optional  !
 filename-material-parameters       character     optional  !
 filename-distance-parameters       character     optional  !
 filename-tighten                   character     optional  !
 filename-k-vectors                 character     optional  !
 k-vectors-sample-type              character     optional  !
 Brillouin-zone-path                character     optional  !
 Brillouin-zone-sections            character     optional  !
 number-of-k-points                 integer       optional  !
 k-direction-from-k-point           double_array  optional  !
 k-direction-to-k-point             double_array  required  !
 power-of-distance-dependence       double        optional  !
 calculate-eigenvectors             character     optional  !
 scale                              double        optional  !
 potential-energy-left              double        optional  !
 potential-energy-right             double        optional  !
 debug-level                        integer       optional  !
                                                            !
 filename-states                    character     optional  !
 calculate-only-lattice-geometry    character     optional  !
 output-Hamiltonian                 character     optional  !
 calculate-k-derivatives            character     optional  !
 tighten-option                     character     optional  !
 number-of-band-for-Fermi-energy    integer       optional  !
 number-of-electron-eigenvalues     integer       optional  !
 number-of-hole-eigenvalues         integer       optional  !
 number-of-band-for-psi             integer       optional  !
 number-of-bands-to-plot            integer       optional  !
 swap-cation-and-anion              character     optional  !

 rescale-to-unstrained-k-points     character     optional  ! for graphene only

!---------------------------
! Pseudopotential algorithm
```

(continues on next page)

(continued from previous page)

```
!--------------------------
 pseudopotential-scaling-factor     double         optional  !
 pseudopotential-form-factors       double_array   optional  !
 G-vectors-included                 integer        optional  !

$end_tighten                                       optional  !
!-----------------------------------------------------------!
```

`$atomic-layers` is only necessary for the heterostructure tight-binding code but not for the bulk tight-binding code.

```
!-----------------------------------------------------------!
$atomic-layers                                     optional  !
 layer-number                      integer         required  !
                                                            !
 ion-1                             character       required  !
 ion-1-content                     double          required  !
                                                            !
 ion-2                             character       optional  !
 ion-2-content                     double          optional  !
 $end_atomic-layers                                optional  !
!-----------------------------------------------------------!
```

Example: $atomic-layers

```
$atomic-layers

 !-------------------------------------
 ! layer 1: cation
 ! layer 2: anion
 !-------------------------------------
 layer-number = 1   ion-1-name = Ga   ion-1-content = 0.5   ion-2-name = Al   ion-2-
↪content = 0.5   ! Ga/Al cation / alloy content
 layer-number = 2   ion-1-name = As   ion-1-content = 1.0                          ␣
↪              ! As    anion  / alloy content
 layer-number = 3   ion-1-name = Ga   ion-1-content = 1.0                          ␣
↪              ! Ga    cation / alloy content
 layer-number = 4   ion-1-name = As   ion-1-content = 1.0                          ␣
↪              ! As    anion  / alloy content
```

Note:

- The total number of layers in [001] superlattice direction must be integer multiple of 4 (e.g. 4, 8, 12, ...) because a unit cell consists of 4 **atomic layers**.
- Convert number of **layers** (atomic layers) into [nm] units: position = `layer-number` * LatticeConstant / 4
- Convert number of **molecular layers** into [nm] units: position = `layer-number` * LatticeConstant / 2
- The lattice constant refers to the constant distance between unit cells in a crystal lattice. A unit cell consists of 2 molecular layers in diamond/zinc blende structure.
- It must hold: `ion-1-content` + `ion-2-content` = `1.0`.

Example: $tighten

```
!--------------------------------------------------------------------!
$tighten                                                             !
 calculate-tight-binding-tighten = yes                               !
 tighten-method                  = rashba2tighten-tighten            !
 tight-binding-model             = Scholz                            ! 'Scholz', 'Sarma'
 destination-directory           = TightBinding/                     ! directory name␣
↪for tight-binding output. This is where all output goes.
 input-directory                 = ../Syntax/Tight-binding files/ ! directory name␣
↪for tight-binding input, relative to executable path
!input-directory                 = H:\Tighten\TIGHTEN_nextnano3\  ! directory name␣
↪for tight-binding input, absolute path
 filename-material-parameters    = TB_material_parameters.in      ! The full filename␣
↪is then: input-directory/filename-material-parameters
 filename-distance-parameters    = TB_distance_parameters.in      ! The full filename␣
↪is then: input-directory/filename-distance-parameters
 filename-tighten                = tighten.in                     ! This file is␣
↪written to destination-directory/ and will be read in again by the tighten␣
↪algorithm.
                                                                  ! If tighten-method␣
↪= tighten, the full filename is then: input-directory/filename-tighten
 filename-k-vectors              = k_vectors.dat                  ! This file is␣
↪written to destination-directory/ and will be read in again by the tighten␣
↪algorithm.
                                                                  ! If tighten-method␣
↪= tighten, the full filename is then: input-directory/filename-k-vectors

 k-vectors-sample-type           = band                           !
 Brillouin-zone-path             = 0.5 0.5 0.5                    ! L
                                   0.0 0.0 0.0                    ! Gamma
                                   1.0 0.0 0.0                    ! X
 Brillouin-zone-sections         = 0.5 0.5 0.5                    ! L =>
                                   0.0 0.0 0.0                    ! Gamma

                                   0.0 0.0 0.0                    ! Gamma =>
                                   1.0 0.0 0.0                    ! X
 number-of-k-points              = 1000                           !

 k-direction-from-k-point        =-0.5 0.0 0.5                    ! k vector in units␣
↪of [2pi/a]
 k-direction-to-k-point          = 0.5 0.0 0.5                    ! k vector in units␣
↪of [2pi/a]
 power-of-distance-dependence    = 2.0                            !
 calculate-eigenvectors          = no                             ! 'yes' / 'no'
 scale                           = 1.0                            !
 potential-energy-left           = 0.0                            ! [eV]
 potential-energy-right          = 0.0                            ! [eV]
 debug-level                     = 1                              ! '1', '2', '3'

 !-----------------
 ! Now tighten....
 !-----------------
 filename-states                 = states.in                      ! The full filename␣
↪is then: input-directory/filename-states

 calculate-only-lattice-geometry = no                             ! 'yes' / 'no'
```

(continues on next page)

(continued from previous page)

```
 output-Hamiltonian              = no                                ! 'yes' / 'lower' /
↪'no'
 calculate-k-derivatives         = no                                ! 'yes' / 'first' /
↪'second' / 'no'
 tighten-option                  = eigenvalues-only                  ! 'eigenvalues-only
↪', 'eigenvectors-and-density', 'eigenvectors-and-g-factor', 'no-eigenvalues'
 number-of-band-for-psi          = 1                                 ! used with 2 above,
↪ index of 1st of 4 bands w |psi|^2
 number-of-bands-to-plot         = 4                                 !

 swap-cation-and-anion           = no                                ! 'no' / 'yes'␣
↪(default is no. For testing only, cation and anion material parameters of database␣
↪are exchanged.)


 !-------------------------------------------------------------------------------------
↪----------
 ! Pseudopotential algorithm ==> Reference: T. P. Pearsall, Quantum Photonics (2017),␣
↪Chapter 4
 !-------------------------------------------------------------------------------------
↪----------
 pseudopotential-scaling-factor = 1.0                                ! (default is 1.0.␣
↪scaling factor for potentials (0.0 <= ... <= 1.0)
                                                                     ! 0.0 = no␣
↪potential, i.e. free electron, 1.0 = full periodic potential, anything in between␣
↪is a mixture
                                                                     ! ==> Note: This␣
↪feature can be used to obtain the free-electron dispersion in a zincblende lattice.
 pseudopotential-form-factors    = 0.0 ... 0.136                     ! 10 values:   U0,s␣
↪  U3,s    U4,s    U8,s    U11,s  U0,a   U3,a    U4,a    U8,a    U11,a
                                                                     ! U0,s and U0,a can␣
↪be chosen both as 0.0. They just determine the reference energy.
                                                                     ! The subscripts s␣
↪and a refer to the symmetric and antisymmetric form factors.
                                                                     ! The lattice␣
↪constant is also needed!!!
 G-vectors-included              = 11                                ! include␣
↪reciprocal G vectors up to this subscript, e.g. G_0 (0), G_3 (3), G_4 (4), G_8 (8),␣
↪G_11 (11), G_12 (12); The subscript is the length squared |G|^2.
                                                                     ! G_0  = (0 0 0)   ␣
↪( 1 vector,  total number of G vectors:  1)
                                                                     ! G_3  = (1 1 1)   ␣
↪( 8 vectors, total number of G vectors:  9)
                                                                     ! G_4  = (2 0 0)   ␣
↪( 6 vectors, total number of G vectors: 15)
                                                                     ! G_8  = (2 2 0)   ␣
↪(12 vectors, total number of G vectors: 27)
                                                                     ! G_11 = (3 1 1)   ␣
↪(24 vectors, total number of G vectors: 51)
                                                                     ! G_12 = (2 2 2)   ␣
↪( 8 vectors, total number of G vectors: 59)
$end_tighten                                                         !
!--------------------------------------------------------------------!
```

Parametrization

`Scholz` and `Sarma` parametrizations are given in the articles *[JancuPRB1998]* and *[Sarma2002]*. The different models can be chosen using `tight-binding-model`.

Note: The standard parameter files are specified as follows

```
filename-material-parameters = TB_material_parameters.in  ! The full filename is␣
↪then: input-directory/filename-material-parameters
filename-distance-parameters = TB_distance_parameters.in  ! The full filename is␣
↪then: input-directory/filename-distance-parameters
```

and are located under

`\nextnano3\Syntax\Tight-binding files\`

`\nextnano3\Syntax\Tight-binding files\TB_material_parameters.in`

`\nextnano3\Syntax\Tight-binding files\TB_material_parameters_JancuPRB1998.in`

`\nextnano3\Syntax\Tight-binding files\TB_material_parameters_SawamuraOME2018.in`

Syntax

It is important to select the appropriate `flow-scheme` needed for tight-binding in *$simulation-flow-control*, e.g. `flow-scheme = 200`.

calculate-tight-binding-tighten
do tight-binding calculation with *tighten*

options
`yes` or `no`

default
`no` (no tight-binding calculation)

tighten-method
[*rashba2tighten* ==> *tighten*]

options
`rashba2tighten-tighten` generate input file for *tighten* and do tight-binding calculation with *tighten*

`rashba2tighten` only generate input file for *tighten*

`tighten` tight-binding calculation (*tighten* only)

tight-binding-model

options
`Scholz` (default) (only for III-V materials)

`Sarma` (for II-VI materials)

destination-directory
directory name for tight-binding output. This is where all output goes.

default
`./`

example
`TightBinding/`

input-directory
directory name for tight-binding input, relative to executable path.

default
`./`

example
`../Syntax/Tight-binding files/`

example
`"H:\My tight-binding folder\"`

filename-tighten

- Name of tight-binding input file. It will be generated by *rashba2tighten*, and read in again by *tighten*.
- This file is written to `destination-directory` and will be read in again by the *tighten* algorithm.
- If `tighten-method = tighten`, the full filename is then: `input-directory \ filename-tighten`

default
`tighten.in`

filename-k-vectors

- This file contains user input for lattice structure and tight-binding parameters.
- It will be generated by *rashba2tighten*, and is read in again by *tighten* program.
- If `tighten-method = tighten`, the full filename is then: `input-directory \ filename-k-vectors`.
- This file contains the k vectors for the tight-binding Hamiltonian that will be diagonalized, i.e. for which the eigenenergies and eigenfunctions will be calculated.
- The columns in this file are:

`REAL(loop_index_over_k) 0.5*(kx+ky) 0.5*(ky+kz) 0.5*(kx+kz) kx ky kz`

default
`k_vectors.dat`

k-vectors-sample-type
Here one specifies how the k vectors have to be sampled for the **bulk** or **superlattice** tight-binding code.

options
`band` for band structure plot along some predefined lines (the lines are different for the **bulk** and **superlattice** code, see below)

`user-defined-path` as defined in `Brillouin-zone-path = ...`

`user-defined-sections` as defined in `Brillouin-zone-sections = ...`

For the **superlattice** tight-binding code the relevant options are:

`band`

- for band structure plot along the lines (**superlattice** code): Γ ==> (along Δ to) X = 0.5 G1 ==> (along Z to) M = 0.5 (G1+G2) ==> (along Σ to) Γ ==> Z = 0.5 G3
- reciprocal primitive vectors:

  G1 = (2pi sqrt(2)/a_lateral) * (1 0 0) ==> along (1 0 0) in-plane direction

  G2 = (2pi sqrt(2)/a_lateral) * (0 1 0) ==> along (0 1 0) in-plane direction

  G3 = (2pi /a_vertical) * (0 0 1) ==> along superlattice direction

a_lateral: in-plane lattice constant

a_vertical: out-of-plane lattice constant

Some predefined paths are:

`k_parallel_100-Gamma-k_superlattice` X = -0.5 G1 ==> (along Δ to) Γ ==> Z = 0.5 G3

`k_parallel_110-Gamma-k_superlattice` M = -0.5 (G1+G2) ==> (along Σ to) Γ ==> Z = 0.5 G3

`110G` 3 values, namely (000), (kx0,0,0), (0,kx0,0), [110] direction and [1-10] direction

`100D` "100D" - "001D"

`two` 2 values, namely (000) and the one specified (kx0,ky0,kz0)

`cirG` circle Γ =(000) with radius kmod=|k0|, surrounding Γ point at distance specified.

`xyz number_of_k_points` values from (0 , 0 , 0) ==> (kx0,ky0,kz0)

`Lprojected-Gamma-Xprojected`

`L-Gamma-Xprojected`

`L-Gamma-X`

`k1-Gamma-k2`

`k1-k2`

`R-Z-Gamma-X-M-A` See p. 404, Fig. 8.37 of C. Hamaguchi, Basic Semiconductor Physis, 2nd edition and Fig. 8.29

`Gamma-X-R-Z-Gamma-M-A-Gamma` See Fig. 8 of *[SawamuraOME2018]*

Bulk tight-binding code

For the bulk tight-binding code the relevant options are:

`band` for band structure plot along the lines (bulk code); This is equivalent to `L-Gamma-X-W-L-K-Gamma` for the bulk tight-binding code.

Some predefined paths are:

`L-Gamma-X-W-K-L-W-X-K-Gamma`

- same as *[JancuPRB1998]* and Fig. 1 of *[SawamuraOME2018]*
- The points U and K have the same energies but the path to these points from Γ or X is not equivalent.
- L-Γ is along Λ axis, i.e. along [1,1,1] direction (L is at $2\pi/a$ (0.5,0.5,0.5); Γ is at $2\pi/a$ (0,0,0)). The distance between these points is $\sqrt{3}\pi/a$.
- Γ-X is along Δ axis, i.e. along [1,0,0] direction (X is at $2\pi/a$ (1,0,0))
- X-W is along V
- W-K is along
- K-L is along
- L-W is along Q
- W-X is along V
- X-K is along
- K-Γ is along Σ

`L-Gamma-X-W-L-K-Gamma` e.g. S. Sapra et al., PRB 66, 205202 (2002) (same as `band`)

`L-Gamma-X-K-Gamma` from L to Γ to X to K to from L to Γ

`L-Gamma-X-U-K-Gamma` e.g. Tom P. Pearsall, Quantum Photonics (Band structure plots for pseudopotential calculations) (*to do: eliminate the path between U and K*)

`L-Gamma-X` from L to Γ to X

`X-Gamma-Z-U'-L-Gamma-K` (for strain)

`110G` 3 values, namely (000) , (kx0,kx0,0) , (-kx0, kx0,0)

`110X` 3 values, namely (001) , (kx0,kx0,1) , (kx0,-kx0,1)

`110D` 3 values, namely (00kz0), (kx0,kx0,kz0), (-kx0, kx0,kz0)

`110L` 3 values, namely (0.5,0.5,0.5), (0.5+kx0,0.5+kx0,0.5-2kx0), (0.5+kx0,0.5-kx0,0.5)

`100G` 3 values, namely (000), (kx0,0,0),(0,ky0,0)

`two` 2 values, namely (000) and (kx0,ky0,kz0)

`cirG` circle Gamma=(000) in kz=0 plane with radius kmod=|k0|

`cirX` circle X =(001) in kz=1 plane with radius kmod=|k0|

`cirL` circle L =(111) in plane lambda*(1,1,-2) + mu*(1,-1,0) with radius kmod=|k0|

`xyz` `number_of_k_points` values from (0,0,0) ==> (kx0,ky0,kz0)

`xff` `number_of_k_points` values from (0,ky0,kz0) ==> (kx0,ky0,kz0)

`ffz` `number_of_k_points` values from (kx0,ky0,0) ==> (kx0,ky0,kz0)

Graphene

For the bulk tight-binding code of graphene the relevant options are:

`M'_K_Gamma_M_K'_M'` band structure of graphene along the path M' ==> K ==> Gamma ==> M ==> K' ==> M'

`K_Gamma_M_K'` band structure of graphene along the path K ==> Gamma ==> M ==> K'

Brillouin-zone-path

This is a path through the Brillouin zone passing through all these k points in this order. The number of entries must be a multiple of 3.

example

```
Brillouin-zone-path = 0.5  0.5  0.5  ! L
                      0.0  0.0  0.0  ! Gamma
                      1.0  0.0  0.0  ! X
                      ...  ...  ...  ! ...
                      ...  ...  ...  ! ...
```

Brillouin-zone-sections

Here one can specify segments along where to sample the Brillouin zone. The number of entries must be a multiple of 6.

example

```
Brillouin-zone-sections = 0.5  0.5  0.5  ! L     ! Section 1: This is a␣
↪path through the Brillouin zone between these two k points. L ==>␣
↪Gamma
                          0.0  0.0  0.0  ! Gamma

                          0.0  0.0  0.0  ! Gamma ! Section 2: Then a␣
↪path through the Brillouin zone between these two k points is taken.␣
↪Gamma ==> X
                          1.0  0.0  0.0  ! X

                          0.0  0.0  0.0  ! Gamma ! Section 3: Then a␣
↪path through the Brillouin zone between these two k points is taken.␣
↪Gamma ==> L
                          0.5  0.5  0.5  ! L

                          ...  ...  ...  !
                          ...  ...  ...  !
```

Brillouin zone of zincblende semiconductor

In order to understand the location of the k points in the Brillouin zone, the following website is very helpful: http://lampx.tugraz.at/~hadley/ss1/bzones/fcc.php

- Define Brillouin zone boundaries in units of 2pi/a where a is the lattice constant.
- The first Brillouin zone comprises those points in reciprocal space that are closer to the origin (i.e. to the Gamma point) than to any other point of the reciprocal lattice.
- The high symmetry points are called:
  - L [1/2 , -1/2 , 1/2] ~[1-11]
  - Γ [0 , 0 , 0] (origin, i.e. center of the first Brillouin zone)
  - X [1 , 0 , 0] ~[100]
  - U [1 , 1/4 , -1/4]
  - K [0 , 3/4 , 3/4]
  - Γ [0 , 0 , 0]
  - W [1 , $\sqrt{2}/2$, 0]
- The line from Γ to [X] along the high-symmetry direction [100] is called Delta Δ.
- The line from Γ to [L] along the high-symmetry direction [111] is called Lambda Λ.
- The line from [K] to [X] along the high-symmetry direction [...] is called Sigma Σ. (*should be checked*)
- The distance in k space from Γ to [L] is $\sqrt{3}\pi/a$.
- If one goes from Γ to [X] along the line Δ, the cubic symmetry splits the 8 bands into 2 bands.
- At the zone center Γ, the energy value is 8-fold degenerate.
- At the [X] point, there are two bands, each of which is 4-fold degenerate.
- If one goes from Γ to [L] along the line Λ, the cubic symmetry splits the 8 bands into 4 bands, the upper and lower one is 1-fold degenerate, the two intermediate ones are each 3-fold degenerate.
- The electronic wave function at the Γ point in the center of the Brillouin zone sees the maximum symmetry of the fcc lattice.
- There are 48 operations (rotations, reflections, inversion) that leave the structure invariant.

number-of-k-points

number of k vectors for which to calculate eigenvalues (only relevant for `band`, `k_parallel_100-Gamma-k_superlattice`, `k_parallel_100-Gamma-k_superlattice`, `xyz` so far)

example

`1000`

With the following flags one can specify a k vector in the Brillouin zone.

```
k-direction-to-k-point   = 0.5   0.0   0.5  ! for superlattice tight-
↪binding code: k = (kx,ky,kSL) vector in dimensionless units [2sqrt(2)pi/
↪alateral], [2pi/avertical]
k-direction-to-k-point   = 0.01  0.01  0.0  ! for bulk         tight-
↪binding code: k = (kx,ky,kz)  vector in units of 2pi/(kx0/alateral,ky0/
↪alateral,kz0/avertical)
```

The superlattice k.p dispersion can be calculated along an arbitrary line from the k point `k-direction-from-k-point` to the Gamma point and then to the k point `k-direction-to-k-point`. Either `k-direction-from-k-point` or `k-direction-to-k-point` or both can be zero. If both are zero, then only the Gamma point is calculated. `k-direction-from-k-point` can be omitted.

You can use this flag to specify a customized plot for the E(k) dispersion, e.g. along a line from [110] to the Gamma point and then to the [001] point.

```
k-direction-from-k-point = -0.5  -0.5   0.0  ! k vector in units of [2pi/a]
k-direction-to-k-point   =  0.0   0.0   0.0  ! k vector in units of [2pi/a]
```

power-of-distance-dependence

default

`2.0`

$t_i = t_0 (d_0/d_i)^\eta$

It seems that this value is not used inside the code, unscaled matrix elements are used.

calculate-eigenvectors

value

`yes` or `no`

default

`no` calculate eigenvalues only

scale

default

`1.0`

example

`5.0`

scale output of wave functions and $|\psi|^2$ to improve visualization of $|\psi|^2$ in the band edges plot

potential-energy-left

default

`0.0` [eV]

Add potential energy to band edges (value at first layer).

potential-energy-right

default

`0.0` [eV]

Add potential energy to band edges (value at last layer). The values in between are interpolated for each layer. This way an electrostatic potential or electric field can be included.

debug-level

default
1

get reduced standard output, this is appropriate to generate an input file if unit cell has many atoms. Note: Only first element of star is printed.

options

- 2 get very detailed output, excluding Hamiltonian
- 3 get very detailed output, including Hamiltonian to stdio

filename-states

default
states.in

The full filename is then: input-directory \ filename-states

This file is located in the folder: \nextnano3\Syntax\Tight-binding files\

For more information on this input file, see documentation further below.

calculate-only-lattice-geometry

default
no

option
yes Calculate only lattice geometry but not tight-binding Hamiltonian.

output-Hamiltonian

default
no Hamiltonian matrix not written to file

options

- yes whole Hamilton matrix is written column wise
- lower lower triangle is written row wise

calculate-k-derivatives

default
no do not calculate k-derivatives of H(k)

options

- yes calculate first and second
- first calculate d/dk H(k)
- second calculate d^2/dk_ik_j H(k)

Note: ham_1deriv_output_file

ham_2deriv_output_file

... files that contain nonzero matrix elements of first (and second) k-derivative of H in same format as ham_output_file,

but only SPIN_UP/SPIN_UP part since SPIN-DN/SPIN-DN is identical and SPIN-DN/SPIN-UP=0.

der1.dat: ham_1deriv_output_file = 'der1.dat'

der2.dat: ham_2deriv_output_file = 'der2.dat'

---

tighten-option

default
`eigenvalues-only` calculate only eigenvalues (eigval)

options

- `eigenvectors-and-density` calculate eigenvectors and density of tight-binding Hamiltonian (ev+dens)
- `eigenvectors-and-g-factor` calculate eigenvectors and g-factor (ev+g)
- `no-eigenvalues` no diagonalization

number-of-band-for-Fermi-energy

range
$\mathbb{N}_{\not<}$

default
`0` i.e. highest valence band number

example
`8`

zero of energy is taken at this band for Gamma point (or first k vector)

This specifier only applies to the bulk tight-binding algorithm and the pseudopotential code.

number-of-electron-eigenvalues

range
$\mathbb{N}$

default
`11`

number-of-electron-eigenvalues

range
$\mathbb{N}$

default
`15`

This specifier only applies to the superlattice tight-binding algorithm.

Include `11` electron and `15` hole energies in E(k) energy dispersion plot.

number-of-band-for-psi

default
`1`

used with 2 above, index of first of 4 bands with $|\psi|^2$ (actually 8 bands due to spin degeneracy)

Note:

- If `ieigenvalues_flag = 2`, wave functions are calculated at Gamma for every second of `number-of-bands-to-plot`/2 consecutive bands, starting with `number-of-band-for-psi`, where -1 and 0 = top of valence band, 1 and 2 = first conduction band. Choose an odd number.
- If `ieigenvalues_flag = 3`, g-factor tensor for this plus following (nondegenerate) band is calculated

---

number-of-bands-to-plot

default
`4`

This is the number of bands that are plotted in the files

- `Gamma_psi_squared.dat` $|\psi|^2, \ldots$ are not shifted
- `Gamma_psi_squared_shift.dat` $|\psi|^2, \ldots$ are shifted by their eigenenergies with respect to the energy dispersion plot
- `Gamma_psi_squared_shift_bandedges.dat` $|\psi|^2, \ldots$ are shifted by their eigenenergies with respect to the band edges plot

starting from the band indicated with `number-of-band-for-psi`.

This number does not take into account spin. E.g. if you specify `4`, the program will print out 8 bands, i.e. twice as much, i.e. spin is taken into account automatically by the program.

rescale-to-unstrained-k-points

default
`no` Do not rescale band structure to unstrained k points. It can be useful to keep the high symmetry points fixed on the graph in order to compare different strains.

option
`yes` Rescale band structure to unstrained k points. This moves the high symmetry points if strained.

This flag is only relevant for the tight-binding code of graphene.

Required input files

Material parameters

`TB_material_parameters.in`: located in folder `nextnano3\Syntax\Tight-binding files\TB_material_parameters.in`

Material parameters that also occur in the normal nextnano GmbH database:

`c11` `c12` elastic constants 10^{-2} [GPa]

`a` lattice constant [Angstrom]

`...` tight binding parameters [eV]

Tight-binding material parameters

`nc` `na` number of electrons on cation and number of electrons on anion (e.g. 3 for Ga and 5 for As in GaAs; 4 for Si)

`Esc` `Epc` `Esec` `Edct2` `Edce` orbital energies [eV], e.g. `Esc` is s-orbital energy of cation

`Esa` `Epa` `Esea` `Edat2` `Edae` orbital energies [eV], e.g. `Esa` is s-orbital energy of anion

(`Edct2` and `Edat2` are used instead of `Ed` to allow for d-orbital splitting)

`Dav` absolute deformation potential [eV]; difference in absolute valence band deformation potentials (in eV) between intrinsic Scholz values and Van de Walle/Needs values. Inclusion of this term guarantees that calculated av agrees with Van de Walle/Needs.

`sss` `scpas` `pcsas` `pps` `ppp` `seses` `secsas` `scseas` `secpas` `pcseas` `scdas` `dcsas` `pcdas` `dcpas` `pcdap` `dcpap` `secdas` `dcseas` `dds` `ddp` `ddd`

$ss\sigma$, $s_cp_a\sigma$, $s_ap_c\sigma$, $pp\sigma$, $pp\pi$, $s^*s^*\sigma$, $s_c^*s_a\sigma$, $s_a^*s_c\sigma$, $s_c^*p_a\sigma$, $s_a^*p_c\sigma$, $s_cd_a\sigma$, $s_ad_c\sigma$, $p_cd_a\sigma$, $p_ad_c\sigma$, $p_cd_a\pi$, $p_ad_c\pi$, $s_c^*d_a\sigma$, $s_a^*d_c\sigma$, $dd\sigma$, $dd\pi$, $dd\delta$

`so_p_c` `so_p_a` onsite and intersite spin-orbit couplings

`so_d_c` `so_d_a`

`so_ppca`

Gammav energy of unstrained top of valence band edge at Gamma point;

Gammac energy of unstrained conduction band edge minimum at Gamma point [eV]

Both, Gammav and Gammac, are with respect to an absolute energy scale (crude estimate only).

The last two values are not used for the tight-binding calculation. However, these are the relevant values that are contained in the output files band_edges_nm.dat and band_edges.dat. They are the values of the unstrained band edges on an absolute energy scale. They are crude estimates only. As already said, they are not input to the actual tight-binding calculation but they are similar to the results of a bulk tight-binding calculation.

The empirical tight-binding material parameters for Ge are the ones of *[JancuPRB1998]*, Table II with the following exceptions: E_s, E_p, E_d, E_{s^*} are shifted by +1.7683 eV.

Distance parameters

TB_distance_parameters.in: located in folder nextnano3\Syntax\Tight-binding files\TB_distance_parameters.in

These parameters are relevant for strained materials or for alloys.

nsss nscpas npcsas npps nppp nseses nsecsas nscseas nsecpas npcseas nscdas ndcsas npcdas ndcpas npcdap ndcpap nsecdas ndcseas ndds nddp nddd

bdeff strain dependent shift of onsite-d energies in [eV] (deformation potential). It is chosen to be identical for cation and anion.

bdeff = bd * ed_Scholz

States

states.in: located in folder nextnano3\Syntax\Tight-binding files\states.in

This noneditable *namelist* file (*namelist* is a Fortran feature) will be read in by the *tighten* program and contains the following information:

```
&state_description

 number_state_ref   = 10

 state_ref_name(1)  = 's'
 state_ref_name(2)  = 'px'
 state_ref_name(3)  = 'py'
 state_ref_name(4)  = 'pz'
 state_ref_name(5)  = 'se'
 state_ref_name(6)  = 'dxy'
 state_ref_name(7)  = 'dyz'
 state_ref_name(8)  = 'dzx'
 state_ref_name(9)  = 'dx2y2'
 state_ref_name(10) = 'dz2r2'

 number_coup_ref    = 21

 coup_ref_name(1)   = 'sss'
 coup_ref_name(2)   = 'sps'
 coup_ref_name(3)   = 'pss'
 coup_ref_name(4)   = 'pps'
 coup_ref_name(5)   = 'ppp'
 coup_ref_name(6)   = 'seses'
```

(continues on next page)

(continued from previous page)

```
 coup_ref_name(7)   = 'sess'
 coup_ref_name(8)   = 'sses'
 coup_ref_name(9)   = 'seps'
 coup_ref_name(10)  = 'pses'
 coup_ref_name(11)  = 'sds'
 coup_ref_name(12)  = 'dss'
 coup_ref_name(13)  = 'pds'
 coup_ref_name(14)  = 'dps'
 coup_ref_name(15)  = 'pdp'
 coup_ref_name(16)  = 'dpp'
 coup_ref_name(17)  = 'seds'
 coup_ref_name(18)  = 'dses'
 coup_ref_name(19)  = 'dds'
 coup_ref_name(20)  = 'ddp'
 coup_ref_name(21)  = 'ddd'

/
```

Generated output files

- `k_vectors.dat`
- `tighten.in`

These are the input files for *tighten*.

- `out_structure.txt`
- `band_edges_unstrained_nm.dat` Gamma conduction band edge and topmost valence band edge (units: position [nm], energy [eV], band gap [eV])
- `band_edges_unstrained_layers.dat` Gamma conduction band edge and topmost valence band edge (units: atomic layer , energy [eV])
- `E(k)_tighten_new_bandedges.dat` energy dispersion E(k) where the x axis is either in units of [1/Angstrom] or integer numbers indicating the number of k points (not shifted, energies correspond to tight-binding material parameters in input file)
- `E(k)_tighten_new.dat` energy dispersion E(k) where the x axis is either in units of [1/Angstrom] or integer numbers indicating the number of k points (shifted so that topmost valence band edge equals 0 eV)
- `E(k)_tighten.dat` energy dispersion E(k) (original output) (shifted so that topmost valence band edge equals 0 eV)

Output files of tighten (bulk)

`E(k)_tighten_bulk_new_noshift.dat` energy dispersion E(k) (original output) (not shifted, energies correspond to tight-binding material parameters in input file)

`E(k)_tighten_bulk_new.dat` energy dispersion E(k) (shifted so that valence band edge of first k vector equals 0 eV)

`E(k)_tighten_bulk.dat` energy dispersion E(k) (original output) (shifted so that valence band edge of Gamma k vector equals 0 eV (default) (shifted so that valence band edge of first k vector equals 0 eV (`debug-level = 10`)

If `k-vectors-sample-type = xyz`, then the $|\mathbf{k}|$ vectors are in units of [1/nm] of the files `out_ek_tighten_bulk_new*.dat`.

`hamtightenout_bulk.txt`

`driver_file_sl.in`

'hamtightenout.dat': ham_output_file = 'hamout.dat'

Strain

```
$simulation-flow-control
 ...
 strain-calculation = homogeneous-strain
```

Include biaxial strain for superlattice *tighten* code.

```
strain-calculation = no-strain
```

No strain is considered for superlattice *tighten* code, i.e. strain tensor is zero and each layer has equilibrium lattice constant in growth direction.

In any case, the lattice constant in the lateral growth direction is the one of the substrate material specified in

```
$domain-ccordinates
 ...
 pseudomorphic-on = GaAs
```

for both, homogeneous-strain and no-strain. Without strain, the vertical lattice constant is the equilibrium lattice constant of the layer material.

$tight-binding

This is preliminary. It is not implemented yet.

```
!-------------------------------------------------------!
$tight-binding                                optional !

 tight-binding-on               character     required ! 'yes' / 'no'
 atomic-positions-out           character     optional ! 'yes' / 'no'

! -> TIGHT BINDING DRIVER INPUT
 screen-output                  integer       optional ! 0 (no screen output), 1␣
↪(minimum screen output), 2 (detailed screen output)
 input-directory                character     required ! root path for the whole TB␣
↪simulation, e.g. 'TB_data\'
 input-parse                    character     required ! name of the TB parse file,␣
↪e.g. 'data_driver'
 TIGHTEN-state-filename         character     required ! name of the reference␣
↪states and couplings file, e.g. 'state.dat'

! SUPERCELL
 supercell-start                double_array  required ! x,y,z [nm]
 supercell-end                  double_array  required ! x,y,z [nm]

! -> TIGHT BINDING INPUT
 nearest-neighbors-order        integer       required ! e.g. 1
 floating-distance              character     required ! 'yes' / 'no', e.g. 'yes'
 nearest-distance-accuracy      double        required ! (%), e.g. 20.0
 relativistic                   character     required ! 'yes' / 'no', e.g. 'yes'
 random-alloy                   character     required ! alloy -> 'yes' / 'no', e.g.
↪'no'
 number-of-alloy-loops          integer       required ! alloy
```

(continues on next page)

(continued from previous page)

```
 periodic-boundaries              character     required ! 'x','xy','xyz' and cyclic␣
↪permutations
 k-vector-filename                character     required !

! -> BASIS INPUT
 basis-output                     integer       required ! 0 (none), 1 (generate xmol␣
↪xyz file), 2 (generate basis file for strain calculation + xmol xyz file), 3␣
↪(calculate strain + generate xmol xyz file), 4 (add strain from data file +␣
↪generate xmol xyz file)
 xmol-basis-filename              character     required !
 strain                           character     required ! 'yes' / 'no'
 strain-directory                 character     required ! e.g. 'strain1\'
 basis-coord-filename             character     required ! e.g. 'basis_coord.dat'
 basis-strain-filename            character     required ! e.g. 'basis_coord_strained.
↪dat'
 potential                        character     required ! 'yes' / 'no', e.g. no
 potential-filename               character     required ! e.g. 'TB_potential.dat'

! -> SPARSE MATRIX INDEX INPUT
 sparse-matrix-format             character     required ! 'upper'
 sparse-matrix-index-filename     character     required ! e.g. 'indices_sparse.dat'
 sparse-matrix-values-filename    character     required ! e.g. 'values_sparse.dat'

! -> DIAGONALIZATION INPUT
 diagonalization                  character     required ! 'yes' / 'no', e.g. yes
 diagonalization-method           character     required ! 'lanczos','davidson'
 maximum-number-of-iterations     integer       required ! e.g. 100000
 tolerance                        double        required ! e.g. 1.0d-6
 number-of-eigenvalues            integer       required ! e.g. 20
 eigenvalues-filename             character     required ! e.g. 'QW_gap.dat'
 result-format                    integer       required ! e.g. 0
 guess-energy-value               double_array  optional ! davidson (complex number in␣
↪[eV]), e.g. (0.7,0.0) -> '0.7 0.0'
 eigenstates                      character     optional ! davidson 'yes' / 'no', e.g.␣
↪no
 eigenstates-filename             character     optional ! davidson e.g. 'QW_gap.dat'
 conduction                       character     optional ! lanczos 'yes' / 'no', e.g.␣
↪yes
 conduction-guess                 double_array  optional ! lanczos e.g. 1.2 1.2
 valence                          character     optional ! lanczos 'yes' / 'no', e.g.␣
↪yes
 valence-guess                    double_array  optional ! lanczos e.g. 0.5 0.5

! -> Dangling bonds saturation
 saturated-bond-length-factor     double        required ! [nm]
 saturated-onsite-energy-H        double        required ! [eV]

$end_tight-binding                              optional !
!-------------------------------------------------------!
```

$warnings

Warnings can be switched on or off. It is recommended to use the default settings which are specified in the database file `database_nn3.in`. Currently, changing the default value does not have much effect.

```
$warnings                          optional
 warnings            logical       required
$end_warnings                      optional
```

Specify here if warnings should be written to the `.log` file.

warnings

type
logical

presence
required

options
`.FALSE.`

`.TRUE.`

default
uses value specified in `database_nn3.in`

Example

```
!----------------------------!
$warnings                    !
 warnings = .TRUE.           ! switch on warnings
$end_warnings                !
!----------------------------!
```

$quantum-bound-states

(1D only)

Finds out all eigenfunctions which are localized within a certain region where one expects bound states for example. The quantum states are specified by a certain threshold fraction of psi within the region [`x-left`, `x-right`]. This is necessary for large quantum clusters which extend far beyond the region of interest and therefore have many irrelevant eigenstates.

```
$quantum-bound-states                    optional
 set-number                integer       required
 quantum-region            integer       optional
 num-schroedinger-equation integer       optional
 charge                    character     optional
 x-left                    double        optional
 x-right                   double        optional
 threshold                 double        optional
$end_quantum-bound-states                optional
```

Syntax

set-number

type
integer

presence
required

example
`1`

Number to distinguish different sets of localized states. Has to be in ascending order.

quantum-region

type
integer

example
`1`

Number of quantum cluster in which to look for localized states.

num-schroedinger-equation

type
integer

example
`1`

Number of Schrödinger equation in which to look for localized states.

charge

type
character

value
`el` or `hl`

Flag whether electrons or holes are regarded.

x-left

type
double

unit
`[nm]`

example
`20.0`

left boundary of localization region

x-right

type
double

unit
`[nm]`

example
`40.0`

right boundary of localization region

threshold

type
double

unit
`[]`

example
`0.6`

Minimum fraction of ψ^2 of certain eigenstate within [x-left, x-right] to be regarded as localized state.

$quantumstate-recombination-rates

Relevant data for generation and recombination rates in quantum clusters. In quantum clusters, rates for different eigenstates can be provided and a net generation and recombination rate will be calculated.

$quantumstate-recombination-rates		**optional**
rate-number	**integer**	**required**
quantum-region	**integer**	**optional**
x-left	**double**	**optional**
x-right	**double**	**optional**
threshold-frac	**double**	**optional**
vb-eigenstate	**integer**	**optional**
cb-eigenstate	**integer**	**optional**
netto-gen-rate	**double**	**optional**
$end_quantumstate-recombination-rates		**optional**

Syntax

rate-number

type
integer

presence
required

example
1

It has to be numbered from 1, 2, 3, ...

quantum-region

type
integer

example
1

number of quantum-cluster

x-left

type
integer

unit
[nm]

example
10.0

left boundary of relevant integration range

x-right

type
integer

unit
[nm]

example
20.0

right boundary of relevant integration range

threshold-frac

type
double

unit
[]

example
20.0

The quantum states are specified by a a certain threshold fraction of ψ^2 within the region [x-left, x-right]. This is necessary for large quantum regions which extend far beyond the region of interest and therefore have many irrelevant eigenstates. The generation rate is proportional to netto_gen_rate * psiV(x)^2 * Fermi_function(x).

vb-eigenstate

type
integer

example
1

relevant valence band eigenstate number

cb-eigenstate

type
integer

example
1

relevant conduction band eigenstate number

netto_gen_rate

type
double

example
?

unit
?

net generation rate

$NEGF-spintronics

NEGF for spintronics: Spin transport based on the Non-equilibrium Green's functions (NEGF) method.

This part is based on the PhD thesis of *[KubisPhD2009]*.

This is preliminary. It is not useful for nondevelopers.

```
$NEGF-spintronics                                optional !
 NEGF-spintronics-on              character      required ! yes/no
 directory-NEGF-spintronics       character      optional !
 include-spin                     character      optional ! yes/no
 include-scattering               character      optional ! yes/no
 scattering-iterations            integer        optional !
 grid-points-in-energy            integer        optional ! number of grid points in␣
↪energy
 minimum-energy                   double         optional ! minimum of considered␣
```

(continues on next page)

(continued from previous page)

```
↪energies in [eV]
 bulk-inversion-asymmetry        double          optional ! BIA [eVnm]
 structure-inversion-asymmetry   double          optional ! SIA (Rashba) [eVnm]
 magnetic-field-B-left           double_array    optional ! B_x, B_y, B_z [T]
 magnetic-field-B-right          double_array    optional ! B_x, B_y, B_z [T]
 magnetic-field-B-up             double_array    optional ! B_x, B_y, B_z [T]
 magnetic-field-B-down           double_array    optional ! B_x, B_y, B_z [T]
 Fermi-level-left                double          optional ! [eV]
 Fermi-level-right               double          optional ! [eV]
 Fermi-level-up                  double          optional ! [eV]
 Fermi-level-down                double          optional ! [eV]
 magnetic-contacts               character       optional ! yes/no
 contacts-left-and-right         character       optional ! yes/no
 calculate-contact-left          character       optional ! yes/no
 calculate-contact-right         character       optional ! yes/no
 calculate-contact-up            character       optional ! yes/no
 calculate-contact-down          character       optional ! yes/no
 calculate-transmission          character       optional ! yes/no

$end_NEGF-spintronics                            optional !
```

$global-parameters

Global parameters are general parameters which are valid all over the device. See also *$global-parameters* under database section.

```
$global-parameters                              optional
 lattice-temperature                double      required
 temperature-sweep-active           character   optional
 temperature-sweep-step-size        double      optional
 temperature-sweep-number-of-steps  integer     optional
 data-out-every-nth-step            integer     optional
$end_global-parameters                          optional
```

The lattice temperature is given in Kelvin. The lowest allowed temperature can be set in *$program_restrictions*.

Note: $global-parameters is optional. If it is not present, the default value for the temperature is taken from *$global-parameters* of the database file. Currently, the default temperature is set to 300 K.

Note: The band gaps (namely the conduction band edges only) are adjusted when the temperature changes. This feature can be switched off. See *$numeric-control*: `varshni-parameters-on = no`

In addition, the lattice constants depend on temperature. This can be switched off as well. See *$numeric-control*: `lattice-constants-temp-coeff-on = no`

Temperature sweep

It is possible to sweep over the temperature, i.e. to vary the temperature stepwise. This is similar to

- electric field sweep (*$electric-field*)
- magnetic field sweep (*$magnetic-field*)
- voltage sweep (*$voltage-sweep*)
- doping concentration sweep (*$doping-function*)

- alloy sweep (*$alloy-function*).

The output is labeled with `..._ind000.dat`, `..._ind001.dat`, `..._ind002.dat`, ... where the index refers to the number of the temperature sweep step.

Example

```
!-------------------------------------------!
 $global-parameters
  lattice-temperature               = 300.0  ! [K]

  temperature-sweep-active          = yes    ! yes/no
  temperature-sweep-step-size       = -10.0  ! [K], i.e. in this case -10 K
  temperature-sweep-number-of-steps = 20     ! number of temperature sweep␣
↪steps
  data-out-every-nth-step           = 10     ! (optional, default = 1)

  $end_global-parameters
 !-------------------------------------------!
```

In this example, the temperature is varied starting from 300 K, and then reducing the temperature 20 times in steps of `-10.0`, ending at 100 K. Obviously, increasing the temperature is also possible. If you do not want to write out all data specified in the output section for every step, you have to enter an integer number greater than one. If you specify 1, then output files will be generated for each temperature sweep. This is useful if you want to fill out all unused space on your hard disk.

Restrictions

```
flow-scheme = 13
flow-scheme = 130
flow-scheme = 14
flow-scheme = 140
```

have some special functionality so far.

130/140: same as 13/14 but including self-consistent Poisson-Schrödinger.

```
flow-scheme = 13: vary temperature as T
     lattice-temperature               = 5.0  ! 5 Kelvin
     temperature-sweep-active          = yes  !
     temperature-sweep-step-size       = 5.0  ! increase by 5 K
     temperature-sweep-number-of-steps = 250  !
     data-out-every-nth-step           = 50   !

flow-scheme = 14: vary temperature as 1000/T
     lattice-temperature               = 1250  ! 1250 Kelvin
     temperature-sweep-active          = yes   !
     temperature-sweep-step-size       = -5.0  ! decrease by 5 K
     temperature-sweep-number-of-steps = 250   !
     data-out-every-nth-step           = 50    !
```

See tutorial Electron concentration in doped semiconductors (Si, Ge, GaAs) for details.

$output-grid

Write grid coordinates to output file. (This keyword is not very useful.)

```
$output-grid                        optional
 grid-coordinate    integer_array   required
 output-file        character       required
$end_output-grid                    optional
```

Example

```
!-----------------------------------!
$output-grid                        !
 grid-coordinate = 1 0 0            ! specify coordinate axis (1 0 0) or (0␣
↪1 0) or (0 0 1)
 output-file     = grid_x.dat       ! specify output file name
                                    !
 grid-coordinate = 0 1 0            !
 output-file     = grid_y.dat       !
$end_output-grid                    !
!-----------------------------------!
```

$output-geometry

Write output files that contain for the material grid points:

- cluster numbers
- quantum cluster numbers
- current cluster numbers

```
$output-geometry                        optional
 cluster-numbers            character   required
 quantum-cluster-numbers    character   optional
 current-cluster-numbers    character   optional
$end_output-geometry                    optional
```

Example

The output filenames can be chosen by the user.

```
$output-geometry
 cluster-numbers         = geometry_cluster-numbers.dat
 quantum-cluster-numbers = geometry_quantum-cluster-numbers.dat
 current-cluster-numbers = geometry_current-cluster-numbers.dat
$end_output-geometry
```

$output-raw-data

Unformatted data output

In order to restart the calculation not from the very beginning but from the last calculated step on has to save the data including all internal structures. This is done by the keyword `$output-raw-data`. In order to keep files small, the format is binary and is not readable by outside programs.

The unformatted data can be reused in the program to continue some calculations. For details see for example `flow-scheme` in *$simulation-flow-control*.

```
$output-raw-data                          optional
 destination-directory     character      required
 potential                 character      optional
 fermi-levels              character      optional
 kp-eigenstates            character      optional
 strain                    character      optional
$end_output-raw-data                      optional
```

Syntax

destination-directory

type
character

presence
required

example
`my-directory/`

example
`raw_data/`

Name of directory to which the files should be written. The folder name has to include the backslash \ (or slash /).

potential

type
character

presence
optional

options
`yes` or `no`

name of output file
`potentials_store1D/2D/3D.raw`

file format
binary (unformatted)

Flag whether to save the electrostatic potential.

fermi-levels

type
character

presence
optional

options
`yes` or `no`

name of output file
`fermi_store1D/2D/3D.raw`

file format
binary (unformatted)

Flag whether to save the quasi-Fermi levels for electrons and holes.

kp-eigenstates

type
character

presence
optional

options
`yes` or `no`

name of output file
`schroedinger_store1D/2D/3D.raw`

name of output file
`kp_store1D/2D/3D.raw`

file format
binary (unformatted)

Flag whether to save the single-band or $\mathbf{k} \cdot \mathbf{p}$ eigenstates and wave functions.

strain

type
character

presence
optional

options
`yes` or `no`

name of output file
`strain_store1D/2D/3D.raw`

file format
binary (unformatted)

Flag whether to save the strain tensor.

Example

```
$output-raw-data
 destination-directory = raw_data/
 potential             = yes        ! 'yes' or 'no'
 fermi-levels          = yes        ! 'yes' or 'no'
 kp-eigenstates        = yes        ! 'yes' or 'no'
 strain                = yes        ! 'yes' or 'no'
$end_output-raw-data
```

$input-filename

Specification of input file name

This is optional (and deprecated). It is much more convenient to start the simulation using command line argument `--inputfile "D:\test\Quantum Well.in"`. The only reason for the existence of this keyword is to start a simulation without command line arguments, e.g. for testing purposes. The name of the input file to be run can be written into the file `keywords.val` within the keyword `$input_filename` which must be the **first** (!) entry in this file. Apart from this modification `keywords.val` must not be changed.

The specification of the input filename is done as a first entry after `$input_filename` in the file `keywords.val` as follows:

```
$input_filename                      optional
 input-filename          character   optional
$end_input_filename                  optional
```

Example 1

```
!-------------------------------------------!
$input_filename                  optional  !
 input_file1.in        character   optional  ! Reads in "input_file1.in".
!1D_Quantum_well.in    character   optional  ! Reads in "1D_Quantum_well.in
↪" if the comment sign "!" is removed.
$end_input_filename              optional  !
!-------------------------------------------!
```

A comment can be inserted using ! or #.

Example 1

```
!-------------------------------------------------------------------!
$input_filename                                          optional  !
 "D:\My nextnano input files\input_file1.in"   character  optional  ! Reads␣
↪in "input_file1.in".
$end_input_filename                                      optional  !
!-------------------------------------------------------------------!
```

$Monte-Carlo

Monte Carlo transport: Here, the parameters for the Monte Carlo procedure and the mobility calculations are defined.

```
$Monte-Carlo                                optional                !
 Monte-Carlo-transport                      character     required  !
 destination-directory                      character     required  !
 grid-position                              double_array  required  ! [nm]
 energy-dispersion                          character     required  !
 charge-carrier-type                        character     required  !
 number-of-simulated-carriers               integer       required  !
 number-of-timesteps                        integer       required  !
 timestep                                   double        required  ! [fs]
 number-of-electric-field-steps             integer       required  !
 electric-field-start-value                 double_array  required  ! [V/m]
 electric-field-step                        double        required  ! [V/m]
 electric-field-step-factor                 double        required  !
 start-averaging-after-timesteps            integer       required  !
 execute-averaging-after-timesteps          integer       required  !
 impurity-background-doping-concentration   double        required  ! [cm-3]
 doping-concentration                       double        required  ! [cm-3]
 LA-phonon-scattering                       character     optional  !
 LO-phonon-scattering                       character     optional  !
 TA-phonon-scattering                       character     optional  !
 TO-phonon-scattering                       character     optional  !
 acoustic-phonon-scattering                 character     optional  !
 polar-optical-phonon-scattering            character     optional  !
 plasmon-scattering                         character     optional  !
 ionized-impurity-scattering                character     optional  !
 surface-roughness-scattering               character     optional  !
 electron-hole-scattering                   character     optional  !
 impact-ionization-scattering               character     optional  !
                                                                    !
 alloy-scattering                           character     optional  !
 alloy-disorder-scattering-potential        double        optional  ! [eV]
```

(continues on next page)

(continued from previous page)

```
                                                                    !
 mass-density                              double        optional  ! [kg/m3]
 sound-velocity                            double        optional  ! [m/s]
 acoustic-deformation-potential            double        optional  ! [eV]
                                                                    !
 quantum-well-width                        double        optional  ! [nm]
 spacer-width                              double_array  optional  ! [nm]
 remote-doping-sheet-density               double_array  optional  ! [cm-2]
 2DEG-sheet-density                        double        optional  ! [cm-2]
 2DEG-sheet-density-number-of-subbands     double        optional  !
$end_Monte-Carlo                                         optional  !
```

Syntax

```
destination-directory = Monte_Carlo/
```

Name of directory to which the Monte Carlo files should be written.
Directory name has to include the slash.

```
Monte-Carlo-transport = yes    ! Monte Carlo transport calculations switched on
                      = no     ! Monte Carlo transport calculations switched off
                      = simple ! Monte Carlo transport calculations switched off and␣
↪a simple 2DEG algorithm is employed
```

```
grid-position = 10.0              ! grid position in units of [nm]
                                  ! take material parameters from the material␣
↪located at this point
grid-position = 10.0              ! x = 10 [nm]                              (1D)
              = 10.0  20.0        ! x = 10 [nm], y = 20 [nm]                 (2D)
              = 10.0  20.0  20.0  ! x = 10 [nm], y = 10 [nm], z = 20 [nm]    (3D)
```

```
energy-dispersion = kp        ! use nonparabolic and anisotropic energy dispersion␣
↪E(k) calculated from k.p theory, i.e.
                              ! take into account k.p           dispersion into␣
↪scattering rates, overlap factors and selection of final states
                  = parabolic ! use parabolic      and     isotropic energy␣
↪dispersion E(k) (single-band effective-mass approximation), i.e.
                              ! take into account parabolic dispersion into␣
↪scattering rates, overlap factors and selection of final states
```

```
charge-carrier-type = holes     ! Monte Carlo transport calculations for holes
                    = electrons ! (not implemented yet)
```

```
number-of-simulated-carriers = 10000 ! number of simulated electrons or holes,␣
↪respectively
                                     ! (depending on charge-carrier-type = ...)
```

```
number-of-timesteps = 600  ! number of timesteps
timestep            = 10.0 ! timestep in units of [fs]
```

total time = `number-of-timesteps` * `timestep` = 600 * 10 fs = 6000 fs = 6 ps

```
number-of-electric-field-steps = 5          ! number of calculated points for␣
↪increasing electrical field (starting value: 1 kV/cm)
electric-field-start-value     = 1.0e5  0.0 ! [V/m], F = (Fx,Fy), i.e. in this case␣
↪Fx = 1.0 * 10^5 V/m = 1 kV/cm,
                                            !                                  Fy = 0 V/m
electric-field-step            = 1.0e3      ! increase electric field strength along␣
↪the channel by this magnitude (for each electric field step)
                                            ! [V/m], i.e. in this case 1.0 * 10^5 V/m,
↪  Units: 1 kV/cm = 1 * 10^5 V/m = 1e5 V/m
                                            ! (1.0d3 = 0.01 kV/cm = 1000 V/m = 1 kV/m␣
↪= 10 V/cm)
electric-field-step-factor     = 1.1        ! increase electric field strength along␣
↪the channel by this factor        (for each electric field step)
                               = 1.0        ! [] linear increase
```

$F_{x,i+1} = F_{x,i}$ * electric-field-step-factor + electric-field-step

$F_{y,i+1} = F_{y,i}$ where F_x and F_y are the electric field magnitude along the y or y direction, respectively.

```
start-averaging-after-timesteps   = 400  ! number of timesteps after which the␣
↪averaging mechanism is started
execute-averaging-after-timesteps = 1    ! number of timesteps after which the␣
↪averaging mechanism is executed
                                         ! (= 1: each timestep is used for averaging)
```

```
impurity-background-doping-concentration = 0.5e15  ! in units of [cm^-3]  ==> 0.5e15␣
↪= 0.5 * 10^15 cm-3
```

The impurity background doping concentration which is relevant for the impurity scattering mechanism is given in units of [cm^-3]. Doping with fully ionized dopants is assumed in this case (during the Monte Carlo procedure only). The whole simulation region will be covered with this background doping during the Monte Carlo procedure. It will contribute additively to the additional doping regions specified inside the input file (*$doping-function*). It also enters the scattering rates for impurity scattering: If it is set to zero, the integral will not converge, so don't do it...

```
doping-concentration = 1e18 ! in units of [cm^-3]  ==> 1e18 = 1 * 10^18 cm^-3
```

The doping concentration is given in units of [cm^-3]. This value for the doping concentration is used for the impurity scattering table. It is relevant for electron-hole scattering and ionized impurity scattering.

Flags to switch on/off certain scattering models

```
LA-phonon-scattering = yes ! include    LA phonon scattering (default)
                     = no  ! switch off LA phonon scattering
```

This affects the subroutines

- total scattering probability due to LA phonon emission in first conduction band
- total scattering probability due to LA phonon emission among hole bands
- total scattering probability due to LA phonon absorption in first conduction band
- total scattering probability due to LA phonon absorption among hole bands

```
LO-phonon-scattering = yes  ! include    LO phonon scattering (default)
                     = no   ! switch off LO phonon scattering
```

This affects the subroutines

- total scattering probability due to LO phonon emission in first conduction band

- total scattering probability due to LO phonon emission among hole bands
- total scattering probability due to LO phonon emission among 2D hole subbands
- total scattering probability due to LO phonon absorption in first conduction band
- total scattering probability due to LO phonon absorption among hole bands
- total scattering probability due to LO phonon absorption among 2D hole subbands

```
TA-phonon-scattering = yes  ! include    TA phonon scattering (default)
                     = no   ! switch off TA phonon scattering
```

This affects the subroutines

- total scattering probability due to TA phonon emission in first conduction band
- total scattering probability due to TA phonon emission among hole bands
- total scattering probability due to TA phonon absorption in first conduction band
- total scattering probability due to TA phonon absorption among hole bands

```
TO-phonon-scattering = yes  ! include    TO phonon scattering (default)
                     = no   ! switch off TO phonon scattering
```

This affects the subroutines

- total scattering probability due to TO phonon emission in first conduction band
- total scattering probability due to TO phonon emission among hole bands
- total scattering probability due to TO phonon absorption in first conduction band
- total scattering probability due to TO phonon absorption among hole bands

```
acoustic-phonon-scattering = yes  ! include    acoustic phonon scattering (default)
                           = no   ! switch off acoustic phonon scattering
```

This affects the subroutines

- total scattering probability due to acoustic scattering in first conduction band
- total scattering probability due to acoustic scattering among hole bands
- total scattering probability due to acoustic scattering among 2D hole subbands

The acoustic phonon scattering rates are linear functions of temperature.

```
polar-optical-phonon-scattering = yes  ! include    polar optical phonon scattering␣
↪(default)
                                = no   ! switch off polar optical phonon scattering
```

This affects the subroutines

- total scattering probability due to polar optical phonon emission
- total scattering probability due to polar optical phonon absorption

```
plasmon-scattering = yes  ! include    plasmon scattering (default)
                   = no   ! switch off plasmon scattering
```

This affects the subroutines

- total scattering probability due to heavy hole - plasmon absorption in first conduction band
- total scattering probability due to Gamma - plasmon absorption in hole bands
- total scattering probability due to heavy hole - plasmon emission in first conduction band

- total scattering probability due to Gamma - plasmon emission in hole band

```
ionized-impurity-scattering = yes  ! include    TO phonon scattering (default)
                            = no   ! switch off TO phonon scattering
```

This affects the subroutines

- total scattering probability due to ionized impurities
- total scattering probability due to ionized impurity scattering among 2D hole subbands

```
surface-roughness-scattering = yes  ! include    surface roughness scattering␣
↪(default)
                             = no   ! switch off surface roughness phonon scattering
```

This affects the subroutines

- total scattering probability due to surface roughness scattering among 2D hole subbands

```
electron-hole-scattering = yes  ! include    electron-hole scattering (default)
                         = no   ! switch off electron-hole scattering
```

This affects the subroutines

- total scattering probability due to electron-hole scattering in first conduction band
- total scattering probability due to electron-hole scattering in hole bands

```
impact-ionization-scattering = yes  ! include    impact ionization scattering (Kane␣
↪model) (default)
                             = no   ! switch off impact ionization scattering (Kane␣
↪model)
```

This affects the subroutines

- total scattering probability due to impact ionization scattering (Kane model)

```
alloy-scattering = yes  ! include    alloy scattering (default)
                 = no   ! switch off alloy scattering
```

This affects the subroutines

- total scattering probability due to alloy scattering in first conduction band
- total scattering probability due to alloy scattering among hole bands

```
alloy-disorder-scattering-potential = 1.0 ! [eV] (optional parameter)
                                    = 1.0 ! [eV]  1 eV for AlGaAs
                                    = 0.6 ! [eV] ~0.6 eV for InGaAs
```

If `alloy-disorder-scattering-potential` is not present, then this value will be calculated internally from the conduction (or valence) band offset of the two binary end points for each grid point, e.g. Al(x)Ga(1-x)As: CBO(AlAs) - CBO(GaAs) Some more information about the used parameters for each grid point is contained in the file `AlloyScatteringInfo.dat`.

Material parameters

The following parameters are used for deformation potential acoustic phonon scattering.

```
mass-density                  = 5.79e3 ! [kg/m3] lattice density
sound-velocity                = 3.7e3  ! [m/s]   velocity of longitudinal elastic␣
↪waves
acoustic-deformation-potential = 7.2    ! [eV]
```

The following parameters are used for the *simple algorithm* (for delta-doped 2DEGs) to calculate the mobility.

```
quantum-well-width                      = 10.0      ! [nm]
spacer-width                            = 20.0      ! [nm]
remote-doping-sheet-density             = 1e12      ! [cm-2]
2DEG-sheet-density                      = 0.071e12  ! [cm-2]
2DEG-sheet-density-number-of-subbands = 3           ! (default: 1)
```

If `2DEG-sheet-density` is not present, then the 2DEG density is calculated automatically where the number of subbands that are taken into account can be specified (default: 1). If two remote doping regions should be taken into account, one can input an array of values.

```
spacer-width                 = 20.0  10.0  ! [nm]   spacer width of first and 2nd␣
↪doping region
remote-doping-sheet-density = 1e12  1e11  ! [cm-2] remote doing density of first and␣
↪2nd doping region
```

Monte Carlo

The Monte Carlo algorithm that is used inside *nextnano*[3] is similar to the following publication:

> Subband structure and mobility of two-dimensional holes in strained Si/SiGe MOSFET's
> R. Oberhuber, G. Zandler and P. Vogl
> Physical Review B 58, 9941 (1998)

Details

The carrier transport in quantized 2D channels is computed in terms of a momentum space ensemble Monte Carlo procedure for a spatially homogeneous channel with constant electric fields invoking consistently calculated 2D scattering rates (see subroutine `scattering` for details on scattering).

For electrons and a single conduction band, a similar procedure was carried out previously (M.V. Fischetti, S.E. Laux, PRB 48, 2244 (1993)).

The equations of motion are integrated with the numerically determined, fully nonparabolic subband dispersions E(k) in analogy to full band Monte Carlo methods that have been developed previously for bulk (M.V. Fischetti, S.E. Laux, PRB 38, 9721 (1988)).

Both the hole dispersion relations E(k) and the inverse relation k(E) have been determined by a 2D discretization of k space.

The Monte Carlo method requires both the individual as well as the total scattering rates. The latter can only be obtained, for each initial state in subband ... with energy E(k), by a numerical integration of Eq. (...) over all final states. This amounts to integrating over the wavevectors q,q with the numerically determined 2D band structure.

Restrictions: So far, this Monte Carlo procedure only makes sense for 1D simulations with quantum confinement.

Note: Do not forget to specify the parameters for the density of states (DOS) calculation. The DOS is used for the calculation of the scattering tables.

```
$output-kp-data
 ...
 DOS-density-of-states = ...
 DOS-Emin-Emax         = ...
 DOS-points            = ...
```

Scattering mechanisms

- polar optical phonon emission
- ionized impurities
- polar optical phonon absorption
- LO phonon emission in first conduction band
- LO phonon emission among hole bands
- LO phonon emission among 2D hole subbands
- LO phonon absorption in first conduction band
- LO phonon absorption among hole bands
- LO phonon absorption among 2D hole subbands
- TO phonon emission in first conduction band
- TO phonon emission among hole bands
- TO phonon absorption in first conduction band
- TO phonon absorption among hole bands
- LA phonon emission in first conduction band
- LA phonon emission among hole bands
- LA phonon absorption in first conduction band
- LA phonon absorption among hole bands
- TA phonon emission in first conduction band
- TA phonon emission among hole bands
- TA phonon absorption in first conduction band
- TA phonon absorption among hole bands
- acoustic scattering in first conduction band
- acoustic scattering among hole bands
- acoustic scattering among 2D hole subbands
- alloy scattering in first conduction band
- alloy scattering among hole bands
- electron-hole scattering in first conduction band
- electron-hole scattering in hole bands
- heavy hole plasmon absorption in first conduction band
- Gamma plasmon absorption in hole bands
- heavy hole plasmon emission in first conduction band
- Gamma plasmon emission in hole band
- surface roughness scattering among 2D hole subbands
- ionized impurity scattering among 2D hole subbands

Output

Output files for k.p data

Density of states (DOS)

The two-dimensional density of states (DOS) is written to the files `Schroedinger_kp/DOS_hl_6x6kp.dat` and `DOS_hl_6x6kp_sum.dat` (and similar for electrons). The DOS has been calculated from the energy dispersion $E(\mathbf{k}) = E(k_x, k_y)$ and is used inside the Monte Carlo procedure (e.g. scattering tables).

For details, see *$output-kp-data*.

Output files for transport data

After the total simulation time is reached, the averaged transport quantities are written to the output files with the subroutine `write_outputMC`. This is done for each value of the applied electric field. The results of the calculated transport quantities are stored in the files:

- `mobility10.dat`
- `mobility11.dat`
- `subband_energy.dat`
- `subband_velocity_x.dat`
- `subband_density.dat`

The growth direction is along [001], so the carrier channel is perpendicular to this direction. The electric field is applied in the [100] direction, so the output file `mobility10.dat` contains the mobility into this direction. To investigate the anisotropy in the two-dimensional plane, also the mobility calculated from the velocity in [110] direction is stored (`mobility11.dat`).

mobility10.dat
: This file contains the calculated low field mobility, both calculated from the velocity (4th column of the file) and the diffusion constant average (5th column) obtained by the Monte Carlo procedure. The value of the electric field in [kV/cm] is given in the 1st column, and the corresponding velocity in x direction, average energy, mobility from velocity and mobility from diffusion constant are listed in the following columns.

```
electric field [kV/cm] vx [cm/s] Eav [eV] μv,x [cm2/Vs] μDiff,x [cm2/Vs]
```

$\mu_{v,x} = v_x / F_x$ where F_x is the *electric field* denoted in the 1st column.

mobility11.dat
: This file contains the mobility along the [11] direction in the two-dimensional k space, and is similar to the file `mobility10.dat` discussed above.

```
electric field [kV/cm] vxy [cm/s] Eav [eV] μv,xy [cm2/Vs]
```

subband_energy.dat
: This file contains for each electric field value (1st column) the average kinetic energy within each subband (2nd column: subband 1, 3rd column: subband 2, ...) in units of [eV].

```
electric field [kV/cm] Eav,1 Eav,2 Eav,3 Eav,4 Eav,5 Eav,6 ...
```

subband_velocity_x.dat
: This file contains the average velocity in each subband in units of [cm/s]. Its structure is similar to the previous file.

```
electric field [kV/cm] vav,1 vav,2 vav,3 vav,4 vav,5 vav,6 ...
```

subband_density.dat
: This file gives the occupation of each subband. $N_{\mathrm{av},i}$ is the average number of particles in subband i. The sum over all $N_{\mathrm{av},i}$ must be equal to the number of particles that have been simulated, i.e. `number-of-simulated-carriers`.

```
electric field [kV/cm] Nav,1 Nav,2 Nav,3 Nav,4 Nav,5 Nav,6 ...
```

The following keywords are not documented yet in the new documentation, i.e. they are linked to the old documentation.

$domain-coordinates

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/domain-coordinates.htm

$material

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/material.htm

$strain-minimization-model

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/strain-minimization-model.htm

$poisson-boundary-conditions

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/poisson-boundary-conditions.htm

$quantum-dot-layer-density

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-dot-layer-density.htm

$output-strain

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-strain.htm

$output-bandstructure

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-bandstructure.htm

$output-1-band-schroedinger

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-1-band-schroedinger.htm

The output of the eigenvalues and eigenfunctions for the single-band Schrödinger equations (`effective-mass`) is controlled by this keyword. All eigenfunctions and eigenvalues between `cb-min-ev` and `cb-max-ev` are written out for each band.

$output-1-band-schroedinger		**optional**
destination-directory	**character**	**required**
sg-structure	**character**	**optional**
effective-mass-tensor	**character**	**optional**
complex-wave-functions	**character**	**optional**

(continues on next page)

(continued from previous page)

```
 eigenvalues-only                    character       optional
 scale                               double          optional
 shift-wave function-by-eigenvalue    character       optional

 conduction-band-numbers             integer_array   optional
 cb-min-ev                           integer         optional
 cb-max-ev                           integer         optional

 valence-band-numbers                integer_array   optional
 vb-min-ev                           integer         optional
 vb-max-ev                           integer         optional

 interband-matrix-elements           character       optional
 intraband-matrix-elements           character       optional
 intraband-lifetime                  character       optional
 intraband-matrix-elements-operator  character       optional

 stark-effect-out                    character       optional
 voltage-offset                      double          optional
 lever-arm-length                    double          optional

 resonance-bound-states              integer_array   optional
 resonance-incidence                 integer_array   optional

$end_output-1-band-schroedinger                      optional
```

Example

```
$output-1-band-schroedinger

 destination-directory             = Schroedinger_1band/
 sg-structure                      = yes
 effective-mass-tensor             = yes
 complex-wave-functions            = yes
 scale                             = 1.0

 conduction-band-numbers           = 1 2 3
 cb-min-ev                         = 1
 cb-max-ev                         = 10

 valence-band-numbers              = 1 2 3
 vb-min-ev                         = 1
 vb-max-ev                         = 10

 interband-matrix-elements         = yes
 intraband-matrix-elements         = yes

$end_output-1-band-schroedinger
```

Syntax

destination-directory

example
`my-directory/`, `Schroedinger_1band/`

Name of directory to which the files should be written. Directory name has to include the slash.

sg-structure

example
yes or no

default
no

Flag whether to write out the Schrödinger structure file (`sg_info.txt`). This file describes the internal number and degeneracy of the Schrödinger equations that have to be solved.

Note: If the energy bands are split due to strain, e.g. X valley and L valley, then the Schrödinger equation has to be solved for different band edges. If the masses are anisotropic, then for each mass valley a separate Schrödinger equation has to be solved.

Note: The files here have different labels:

`cb3` = conduction band no. 3 (1 = Gamma band, 2 = L band, 3 = X band)
`qc1` = quantum cluster no. 1
`sg3` = no. of Schrödinger equation to be solved
`deg1` = degeneracy of Schrödinger equation to be solved

Output of effective mass tensor

effective-mass-tensor

example
yes or no

default
no

Flag whether to write out the effective mass tensor $(1/m)_{ij}$ for each Schrödinger equation to be solved. Output are 6 components of the symmetric 3x3 matrix for each grid point. The file is called `*mass_tensor*.dat`.

Example for 1D output: `cb3_mass_tensor_qc1_sg3_deg1.dat`

```
position[nm]    (1/m)_xx        (1/m)_yy        (1/m)_zz        (1/m)_xy        (1/m)_xz  ␣
↪     (1/m)_yz
...
0.350750E+002   0.769231E+000   0.434783E+001   0.434783E+001   0.000000E+000   0.
↪000000E+000   0.000000E+000
...
                1/1.3 = 0.769    1/0.23 = 4.434   1/0.23 = 4.34
```

In this example the mass tensor is diagonal and $(1/m)_{xx}$‘ contains 1/(longitudinal mass m_l) and $(1/m)_{yy} = (1/m)_{zz}$ contains 1/(transverse mass m_t) of the X valley of GaAs.

The effective masses are specified in the database.

```
$binary-zb-default
 binary-type            = GaAs-zb-default
 conduction-band-masses = 0.067 0.067  0.067   ! Gamma (isotropic)
                          1.9   0.0754 0.0754  ! L   (ml mt mt)
                          1.3   0.23   0.23    ! X   (ml mt mt)
```

More information on effective masses.

Note that the labels x, y and z of the mass tensor output are defined with respect to the *simulation* coordinate system (and not *crystal* coordinate system).

complex-wave-functions

example
`yes` or `no`

default
`no`

Flag whether to print out the wave functions ψ (amplitudes) including real and imaginary parts in addition to the output of the probability densities ψ^2. The amplitudes of a single-band Hamiltonian are typically real and the imaginary part is zero. This does not hold for nonzero magnetic fields or nonzero superlattice vectors. Depending on the algorithm inside the code (e.g. storage of the Hamiltonian in a real or complex array, or usage of a real or complex eigenvalue solver), the output contains only the real part, or the real part and the imaginary part (which is zero in most cases). In the case of a complex eigenvalue solver, the imaginary part might be nonzero even for zero magnetic field or no superlattice vector. If this is the case, the wave functions could be converted into a real basis. (It would have been better to call this specifier `amplitudes` rather than `complex-wave-functions`.)

eigenvalues-only

example
`yes` or `no`

default
`no`

Sometimes one is only interested in plotting out the eigenvalues but not the eigenfunctions. Nevertheless, internally in the program the eigenfunctions are used, e.g. for calculating the density.

$output-kp-data

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-kp-data.htm

$output-densities

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-densities.htm

$output-current-data

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-current-data.htm

$output-file-format

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-file-format.htm

$output-section

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-section.htm

$output-material

Writes material parameters on grid coordinates to output files.

```
$output-material                                optional
 destination-directory                character    required

 number-of-conduction-bands           character    optional
 conduction-band-energies             character    optional
 conduction-band-masses               character    optional
 conduction-band-degeneracies         character    optional
 conduction-band-nonparabolicities    character    optional

 number-of-valence-bands              character    optional
 valence-band-energies                character    optional
 valence-band-masses                  character    optional
 valence-band-degeneracies            character    optional
 valence-band-nonparabolicities       character    optional

 static-dielectric-constants          character    optional
 optical-dielectric-constants         character    optional
 lattice-constants                    character    optional
 doping-concentration                 character    optional

 pressure                             character    optional
 elastic-constants                    character    optional
 absolute-deformation-potentials-cbs  character    optional
 absolute-deformation-potential-vb    character    optional
 uniax-cb-deformation-potentials      character    optional
 uniax-vb-deformation-potentials      character    optional
 spontaneous-polarization             character    optional
 piezoelectric-polarization           character    optional

 Luttinger-parameters                 character    optional
 6x6kp-parameters                     character    optional
 8x8kp-parameters                     character    optional
 LO-phonon-energy                     character    optional

 grid-position                        double_array optional
$end_output-material                              optional
```

destination-directory

type
character

example
`material_parameters/`

Name of directory to which the files should be written. Directory name has to include the slash (for DOS and / for Linux)

number-of-conduction-bands

type
character

example
`num-cbands`

Name of file for storing the number of conduction bands at each grid point.

conduction-band-energies

type
character

example
`cb-energies`

Name of file for storing the conduction band energies E_c^Γ, E_c^L, E_c^X at each grid point.

conduction-band-masses

type
character

example
`cb-masses`

Name of file for storing the conduction band masses m^Γ, m^L, m^X at each grid point.

conduction-band-degeneracies

type
character

example
`cb-degeneracy`

Name of file for storing the degeneracies of the conduction bands at Γ, L, X for each grid point.

conduction-band-nonparabolicities

type
character

example
`cb-nonpar`

Name of file for storing the non-parabolicities of the conduction bands Γ, L, X for each grid point.

number-of-valence-bands

type
character

example
`num-vbands`

Name of file for storing the number of valence bands at each grid point.

valence-band-energies

type
character

example
`vb-energy`

Name of file for storing the average of the three valence band energies $E_{\mathrm{v,av}}$ at each grid point.

valence-band-masses

type
character

example
`vb-masses`

Name of file for storing the valence band masses m_{hh}, m_{lh}, m_{so} at each grid point.

valence-band-degeneracies

type
character

example
`vb-degeneracy`

Name of file for storing the valence band degeneracies of the hh, lh, so states at each grid point.

valence-band-nonparabolicities

type
character

example
`vb-nonpar`

Name of file for storing the valence band non-parabolicities of the hh, lh, so states at each grid point.

static-dielectric-constants

type
character

example
`st-dielc`

Name of file for storing the static dielectric constant $\epsilon(0)$ at each grid point.

optical-dielectric-constants

type
character

example
`op-dielc`

Name of file for storing the optical dielectric constant $\epsilon(\infty)$ at each grid point.

doping-concentration

type
character

example
`doping_concentration`

Name of file for storing the doping concentration in units of $1 \cdot 10^{18}\ 1/\mathrm{cm}^3$ at each grid point.

lattice-constants

type
character

example
`lattice-constants`

Name of file for storing the lattice-constant a (zincblende) or a, c (wurtzite) at each grid point.

pressure

type
character

example
`pressure`

Name of file for storing the pressure in units of $[\mathrm{GPa}]$ at each grid point.

elastic-constant

type
character

example
`elastic-constants`

Name of file for storing the elastic constants c_{11}, c_{12}, c_{44} (zincblende) or c_{11}, c_{12}, c_{13}, c_{33}, c_{44} (wurtzite) at each grid point.

absolute-deformation-potentials-cbs

type
character

example
`abs_def_cb`

Name of file for storing the absolute deformation potentials for the conduction band at Γ, L, X at each grid point.

absolute-deformation-potential-vb

type
character

example
`abs_def_vb`

Name of file for storing the absolute deformation potential for the valence band at each grid point.

uniax-cb-deformation-potentials

type
character

example
`uniax_cb_def`

Name of file for storing the uniaxial deformation potential conduction band at Γ, L, X at each grid point.

uniax-vb-deformation-potentials

type
character

example
`uniax_vb_def`

Name of file for storing the uniaxial deformation potentials b, d for the valence band at each grid point.

spontaneous-polarization

type
character

example
`pyro_polarization`

Name of file for storing the pyroelectric polarization constant at each grid point.

piezoelectric-polarization

type
character

example
`piezo_polarization`

Name of file for storing the piezoelectric polarization constants e_{14} (zincblende) or e_{33}, e_{31}, e_{15} (wurtzite) in units of $[\mathrm{C}/\mathrm{cm}^2]$ at each grid point.

Luttinger-parameters

type
character

example
`Luttinger`

Name of file for storing the Luttinger parameters γ_1, γ_2, γ_3 at each grid point.

6x6kp-parameters

type
character

example
`6x6kp`

Name of file for storing the 6-band $\mathbf{k}.\mathbf{p}$ parameters L, M, N, Δ_{so} at each grid point.

8x8kp-parameters

type
character

example
`8x8kp`

Name of file for storing the 8-band $\mathbf{k}.\mathbf{p}$ parameters L', M', N', B, P, S, Δ_{so} at each grid point.

LO-phonon-energy

type
character

example
`LO_phonon_energy`

Name of file for storing the LO-phonon energy E_{LO} in units of eV at each grid point.

grid-position

type
double_array

example

`10d0` (1D)
`10d0 20d0` (2D)
`10d0 20d0 30d0` (3D)

Prints out the material parameters for the binary (or ternary) material at this position. This feature can be used to calculate the material parameters of a ternary, and print it to a file. This data format can then be modified and read in from the database or input file ("copy and paste"). Example: MaterialParametersGridPointAl(x)Ga(1-x)N.dat

```
$binary-wz-default                                                    ␣
↪     !
 binary-type                  = Al(x)Ga(1-x)N,x=0.400,1-x=0.600-wz-
↪default
 conduction-bands             = 3                                    ␣
↪     !
 conduction-band-masses       = 0.208000D+00  0.208000D+00  0.
↪252000D+00     ! [m0]
                                0.946800D+00  0.286800D+00  0.
↪493200D+00     ! [m0]
                                0.143400D+01  0.300000D+00  0.
↪376000D+00     ! [m0]
 conduction-band-degeneracies = 2 8 6                                ␣
↪     !
...
```

Example

```
$output-material
 destination-directory               = material_parameters/
 number-of-conduction-bands          = cb-nums.dat
 conduction-band-energies            = cb-energies.dat
 conduction-band-masses              = cb-masses.dat
 conduction-band-nonparabolicities   = cb-nonpara.dat
 conduction-band-degeneracies        = cb-deg.dat
 number-of-valence-bands             = vb-nums.dat
 valence-band-energies               = vb-energy.dat
 valence-band-masses                 = vb-masses.dat
 valence-band-nonparabolicities      = vb-nonpara.dat
 valence-band-degeneracies           = vb-deg.dat
 static-dielectric-constants         = static-dielectric-constants.dat
 optical-dielectric-constants        = optical-dielectric-constants.dat
 lattice-constants                   = lattice-constants.dat
 pressure                            = pressure.dat
 elastic-constants                   = elastic-constants.dat
 absolute-deformation-potentials-cbs = cb-abs-defpots.dat
 absolute-deformation-potential-vb   = vb-abs-defpot.dat
 uniax-cb-deformation-potentials     = cb-uniax-defpots.dat
 uniax-vb-deformation-potentials     = vb-uniax-defpots.dat
 Luttinger-parameters                = Luttinger-parameters.dat
 6x6kp-parameters                    = 6x6kp-parameters.dat
 8x8kp-parameters                    = 8x8kp-parametersv
 LO-phonon-energy                    = LO_phonon_energy.dat
 piezoelectric-polarization          = piezo-constants.dat
 spontaneous-polarization            = pyro-constants.dat
 doping-concentration                = doping-concentration.dat

 grid-position                       = 5d0
$end_output-material
```

$doping-function

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/doping-function.htm

$interface-states

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/interface-states.htm

$global-settings

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/global-settings.htm

$simulation-flow-control

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/simulation-flow-control.htm

$grid-specification

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/grid-specification.htm

$import-data-on-material-grid

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/import-data-on-material-grid.htm

$binary-zb-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/binary-zb-default.htm

$binary-wz-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/binary-wz-default.htm

$ternary-zb-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/ternary-zb-default.htm

$ternary-wz-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/ternary-wz-default.htm

$alloy-function

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/alloy-function.htm

$numeric-control

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/numeric-control.htm

$CBR-current

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/CBR-current.htm

$quantum-regions

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-regions.htm

$quantum-model-electrons

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-model-electrons.htm

$quantum-model-holes

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-model-holes.htm

$voltage-sweep

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/voltage-sweep.htm

$electric-field

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/electric-field.htm

$simple-drift-models

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/simple-drift-models.htm

$NEGF

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/NEGF.htm

$electrolyte

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/electrolyte.htm

$electrolyte-ion-content

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/electrolyte-ion-content.htm

Keywords: This section contains the definition of the allowed syntax in your input file.

The old documentation of the keywords for *nextnano*[3] is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/keywords.htm

7.3 Material Database

7.3.1 The database keywords

Here you can find detailed descriptions about the keywords and specifiers in the default parameter database input file `database_nn3.in` and the general definitions file `database_nn3_keywords.val`.

The scheme is

- `keywords.val` (general definitions) <== `input_file.in` (input file)
- `database_nn3_keywords.val` (general definitions) <== `database_nn3.in` (database file)

Valid keywords for the parameters of the default parameter database are listed in `database_nn3_keywords.val`. This is the place to enter new keywords if one wants to add new parameters. In principle, one should only enter into this database keywords, which are marked as `required`. Through this it is warranted that corresponding entries in the database actually do occur. The input parser checks this, but only their existence. Additionally, the new keywords have to be entered into `keywords.val`. E.g. for zinc blende materials the keyword is `$binary_zb_default`. On this basis, the entries in the data base `database_nn3.in` have to be complemented for the newly declared values for all affected known materials. This also includes the adaption of changes in the data types of already existing parameters (e.g. `double` ==> `double_array`).

Material parameters can be found in the review articles of Vurgaftman and Meyer (*[VurgaftmanJAP2001]*, *[VurgaftmanJAP2003]*).

7.3.2 Keywords

$binary-zb-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/binary-zb-default.htm

$binary-wz-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/binary-wz-default.htm

$ternary-zb-default

Ternary zinc blende parameters

Parameters for zinc blende type ternary alloys. This set of parameters refers to the binary constituents and their material parameters. Here, the bowing parameters for interpolation between these binaries are specified.

A bowing parameter b is defined as follows for the material parameter Q. Note that there is a minus sign in front of the term $bx(1-x)$.

$Q[\mathrm{A}_x\mathrm{B}_{1-x}\mathrm{C}] = x \cdot Q[\mathrm{AC}] + (1-x) \cdot Q[\mathrm{BC}] - b \cdot x \cdot (1-x)$

b is defined as $b = 4Q(\mathrm{A}_{0.5}\mathrm{B}_{0.5}\mathrm{C}) - 2(Q[\mathrm{AC}] + Q[\mathrm{BC}])$.

The advantage of the bowing model is that it requires knowledge of the relevant quantity only at a composition x=0.5 together with the values for the binaries.

Please check the keywords section for more details: $ternary-zb-default

For the meaning of the material parameters and its units, please check the keywords section for $binary-zb-default for more details. The units for binary material parameters and for the bowing parameters are the same.

$ternary-zb-default		**required**
ternary-type	**character**	**required**
binary(x)	**character**	**required**
binary(1-x)	**character**	**required**
bow-conduction-band-masses	**double_array**	**optional**
bow-conduction-band-nonparabolicities	**double_array**	**optional**
bow-conduction-band-energies	**double_array**	**optional**
bow-valence-band-masses	**double_array**	**optional**
bow-valence-band-nonparabolicities	**double_array**	**optional**
bow-valence-band-energies	**double_array**	**optional**
bow-band-gaps	**double_array**	**optional**
bow-static-dielectric-constants	**double_array**	**optional**
bow-optical-dielectric-constants	**double**	**optional**
bow-lattice-constants	**double_array**	**optional**
bow-elastic-constants	**double_array**	**optional**
bow-piezo-electric-constants	**double_array**	**optional**
bow-abs-deformation-pot-vb	**double**	**optional**
bow-abs-deformation-pots-cbs	**double_array**	**optional**
bow-uniax-vb-deformation-pots	**double_array**	**optional**
bow-uniax-cb-deformation-pots	**double_array**	**optional**
bow-Luttinger-parameters	**double_array**	**optional**
bow-6x6kp-parameters	**double_array**	**optional**
bow-8x8kp-parameters	**double_array**	**optional**
bow-LO-phonon-energy	**double**	**optional**
bow-band-shift	**double**	**optional**

(continues on next page)

(continued from previous page)

```
 band-shift                                    double          optional
$end_ternary-zb-default                                        required
```

Example 1

If no bowing parameters are specified, linear interpolation is done.

```
!------------------------------------------------------------!
$ternary-zb-default
 ternary-type                 = Al(x)In(1-x)Sb-zb-default
 binary(x)                    = AlSb-zb-default
 binary(1-x)                  = InSb-zb-default
$end_ternary-zb-default
!------------------------------------------------------------!
```

Example 2

If a bowing parameter is nonzero, bowing is used. Bowing can be quite complicated, especially if

- valence band offset (average valence band edge energy)
- spin-orbit splitting energy
- band gap

are bowed simultaneously.

Note that you can use `band-shift` to shift the resulting band edges by the desired amount.

It is hard to find bowing parameters for a default material database that works for every alloy parameter.

```
!------------------------------------------------------------------------!
$ternary-zb-default
 ternary-type                 = In(x)Ga(1-x)As-zb-default
 binary(x)                    = InAs-zb-default
 binary(1-x)                  = GaAs-zb-default

 bow-conduction-band-masses   = 0.0091   0.0091   0.0091                 !␣
↪=> 0.04300 [m0] (for In0.53Ga0.47As)
                                0.0      0.0      0.0                    !␣
↪no bowing for L
                                0.0      0.0      0.0                    !␣
↪no bowing for X

 bow-band-gaps                = 0.477    0.33     1.4                    !␣
↪[Vurgaftman1]
 bow-conduction-band-energies = 0.477    0.33     1.4                    !␣
↪[Vurgaftman1] for gaps. This is good (better than zero bowing).

 bow-valence-band-masses      = -0.145   -0.145   -0.145                 !␣
↪hh along [001]
                                 0.0202   0.0202   0.0202               !␣
↪lh along [001]
                                 0.0      0.0      0.0                  !␣
↪so

 bow-valence-band-energies    = -0.38                                   !␣
↪[Vurgaftman1] (valence band offset bowing) (Does this value refer to the␣
↪valence band edge or to the average valence band edge?)
!bow-valence-band-energies    = -0.0511107                              !␣
```

(continues on next page)

(continued from previous page)

```
↪to get a band gap of 0.816 eV (for In0.53Ga0.47As)

 bow-abs-deformation-pots-cbs = 2.61     2.61     2.61                    !␣
↪[Vurgaftman1] (Gamma) absolute deformation potentials of conduction band␣
↪minima (Gamma, L, X)

 bow-Luttinger-parameters     = 11.52388599   5.191489362   5.67282216    !␣
↪[Vurgaftman1] to get original values of Alavi et al. for In0.53Ga0.47As,␣
↪recommended by Vurgaftman
!bow-Luttinger-parameters     = 0.0      0.0      0.0                     !␣
↪gamma3 - gamma2 =  0.481 [Vurgaftman1]
                                0.0      0.0                             !
 bow-6x6kp-parameters         = -32.28984344 -1.140907266 -34.03693296    !␣
↪L = -28.73, M = -3.65, N = -29.04 (for In0.53Ga0.47As)
                                 0.15                                    !␣
↪Delta_so [Vurgaftman1]
 bow-8x8kp-parameters         = 0.0          -1.140907266  0.0            !
                                0.0  -1.48          3.54                 !␣
↪bowing(E_P) = -1.48 eV ([Vurgaftman1] => E_P = 25.3 eV), bowing(S) = 2 *␣
↪bowing(F) = 2 * 1.77 ([Vurgaftman1])
!                               0.0  12.1678040    3.54                  !␣
↪bowing(E_P) = 12.1678040 eV (Sirtori => E_P = 21.9 eV = 21.5 * 0.53 + 28.
↪8 * 0.47 - 0.53 * 0.47 * 12.16780409)

$end_ternary-zb-default
!-------------------------------------------------------------------------!
```

Example 3

You can define special ternaries for certain conditions, e.g. low temperature or high In content.

```
!-------------------------------------------------------------------!
$ternary-zb-default
 ternary-type                 = Al(x)In(1-x)Sb-4K-zb-default
!ternary-type                 = Al(x)In(1-x)Sb-zb-default
 binary(x)                    = AlSb-zb-default
 binary(1-x)                  = InSb-zb-default

 ! Define here special bowing parameters for 4 K.
 ...

$end_ternary-zb-default
!-------------------------------------------------------------------!
```

`ternary-type` e.g. `Al(x)Ga(1-x)As-zb-default`, must be a defined ternary material. This string is usually a known material. If a material-type with material-model = ternary-zb-default is specified within the $material keyword which is unknown, you have to provide a complete set of input data for this material type. In this case the material-type must be equal to string. However, the binary constituents can still be either known or unknown binary materials.

`binary(x)` and `binary(1-x)` must be a binary material of type `binary-zb-default`, e.g. `AlSb-zb-default`, must be a defined binary material. This string can be either a known binary or an arbitrary name. In case this binary is not a known material, you will be prompted for all material parameters.

band-shift

type
double

presence
optional

value
`0.0`

If nonzero, the resulting ternary material is shifted by this amount (independent of the alloy-conent). So the default value should be `0.0`.

bow-valence-band-energies

type
double

presence
optional

value
`0.0`

This bowing applies to the average valence band edge energy and not to the valence band maximum.

$ternary-wz-default

Ternary wurtzite parameters

Parameters for wurtzite type ternary alloys. This set of parameters refers to the binary constituents and their material parameters. Here, the bowing parameters for interpolation between these binaries are specified.

```
$ternary-wz-default                                    required
 ternary-type                              character   required
 binary(x)                                 character   required
 binary(1-x)                               character   required
 bow-conduction-band-masses                double_array optional
 bow-conduction-band-nonparabolicities     double_array optional
 bow-conduction-band-energies              double_array optional
 bow-valence-band-masses                   double_array optional
 bow-valence-band-nonparabolicities        double_array optional
 bow-valence-band-energies                 double_array optional !
↪"average" valence band edge energy E_v (without splittings)
 bow-band-gaps                             double_array optional
 bow-static-dielectric-constants           double_array optional
 bow-optical-dielectric-constants          double_array optional ! for␣
↪zincblende, this is double
 bow-lattice-constants                     double_array optional
 bow-piezo-electric-constants              double_array optional
 bow-pyro-polarization                     double_array optional ! for␣
↪zincblende, this does not exist
 bow-abs-deformation-pot-vb                double         optional ! not␣
↪used in wurtzite
 bow-abs-deformation-pots-cbs              double_array optional
 bow-uniax-vb-deformation-pots             double_array optional
 bow-uniax-cb-deformation-pots             double_array optional ! not␣
↪used in wurtzite
 bow-elastic-constants                     double_array optional
 bow-Luttinger-parameters                  double_array optional
 bow-6x6kp-parameters                      double_array optional
 bow-8x8kp-parameters                      double_array optional
 bow-LO-phonon-energy                      double_array optional ! for␣
↪zincblende, this is double
 bow-band-shift                            double         optional
```

(continues on next page)

(continued from previous page)

```
 band-shift                              double          optional
$end_ternary-wz-default                                  required
```

Here, no further explanations are given. The meaning is analogous to *$ternary-zb-default*.

$default-materials

Default materials.

```
$default-materials                                       required
 material-name                  character    required
 material-type                  character    required
 material-model                 character    required
 material-property              character    optional
$end_default-materials                                   required
```

Explanation of specifiers.

material-name

type
character

presence
required

value
e.g. `Si`

An arbitrary name can be chosen, e.g. `Si` or `silicon`.

material-type

type
character

presence
required

value
e.g. `Si-zb-default`

The material parameters of `Si-zb-default` have to be defined in the database.

material-model

type
character

presence
required

value
`binary-zb-default`, `binary-wz-default`, `ternary-zb-default`, `ternary-wz-default`; not supported yet: `quaternary-wz-default`

Zinc blende (zb) and wurtzite (wz) binary and ternary alloys are supported. Materials with diamond crystal structure can be specified using zinc blende. SiGe alloys are called "ternaries" even if only two atoms are involved.

For alloys (ternaries) only.

material-property

type
character

presence
optional

value
swap-(x)-and-(1-x)

If material is a ternary material, this value can be set to `swap-(x)-and-(1-x)`. This allows to specify 2 different types of SiGe alloys that contain the same material parameters but x and $y = (1 - x)$ are interchanged.

```
material-name = Si(1-x)Ge(x)    material-type = Si(1-x)Ge(x)-zb-default   ␣
↪material-model = ternary-zb-default
material-name = Si(x)Ge(1-x)    material-type = Si(1-x)Ge(x)-zb-default   ␣
↪material-model = ternary-zb-default   material-property = swap-(x)-and-(1-
↪x)
```

Here, `material-name` is swapped but both refer to the same `material-type`.

You can also define your own materials, e.g. GaAs with parameters for 4 K, or explicit parameters for a ternary, e.g. `Al(x)In(1-x)P` with a fixed value of x=0.52: $Al_{0.52}In_{0.48}P$

```
material-name  = GaAs-4K
material-model = binary-zb-default
material-type  = GaAs-4K-zb-default

material-name  = Al0.52In0.48P
material-model = binary-zb-default
material-type  = Al0.52In0.48P-zb-default
```

Further examples

```
!-----------------------------------------------------------!
$default-materials

 material-name  = Si
 material-type  = Si-zb-default
 material-model = binary-zb-default

 !-------------------------------------------------------------------------
↪----------------------------------
 ! 'alias' materials: Here, we specify 4 different types of AlGaAs alloy␣
↪that contain the same material parameters.
 !-------------------------------------------------------------------------
↪----------------------------------
 material-name = Al(x)Ga(1-x)As    material-type = Al(x)Ga(1-x)As-zb-
↪default    material-model = ternary-zb-default
 material-name = Ga(1-x)Al(x)As    material-type = Al(x)Ga(1-x)As-zb-
↪default    material-model = ternary-zb-default
 material-name = Al(1-x)Ga(x)As    material-type = Al(x)Ga(1-x)As-zb-
↪default    material-model = ternary-zb-default   material-property = swap-
↪(x)-and-(1-x)
 material-name = Ga(x)Al(1-x)As    material-type = Al(x)Ga(1-x)As-zb-
↪default    material-model = ternary-zb-default   material-property = swap-
↪(x)-and-(1-x)

 material-name = Al(x)Ga(1-x)N
 material-type = Al(x)Ga(1-x)N-wz-default
 material-model = ternary-wz-default

$end_default-materials
!-----------------------------------------------------------!
```

$default-material-models

Default material models.

```
$default-material-models                        required
 material-model-number          integer         required
 material-model                 character       required
 needs-alloy-function           logical         required
$end_default-material-models                    required
```

Zinc blende and wurtzite materials are supported. Binary and ternaries are possible. The latter need an alloy function. Quaternarier are not supported yet.

Example

```
!-----------------------------------------------!
$default-material-models

 ! Zinc blende

 material-model-number = 1
 material-model        = binary-zb-default
 needs-alloy-function  = .FALSE.

 material-model-number = 2
 material-model        = ternary-zb-default
 needs-alloy-function  = .TRUE.

 ! Wurtzite

 material-model-number = 3
 material-model        = binary-wz-default
 needs-alloy-function  = .FALSE.

 material-model-number = 4
 material-model        = ternary-wz-default
 needs-alloy-function  = .TRUE.

 ! not implemented yet:

 material-model-number = 5                      ! not supported yet
 material-model        = quaternary-zb-default  ! not supported yet
 needs-alloy-function  = .TRUE.                 ! not supported yet

$end_default-material-models
!-----------------------------------------------!
```

$Auger-recombination

More information on the Physics: Auger recombination processes in semiconductor heterostructures

For devices with an extremely high carrier concentration the Auger process is the dominant recombination channel. The process involves three particles and therefore scales with the third power of the carrier densities. The phonon-assisted Auger recombination rate, which plays an important role especially at high carrier injection, respectively high doping levels, will be modeled in the program by the following equation:

$R_{\mathrm{Aug}} = (C_{\mathrm{n}} n + C_{\mathrm{p}} p)(np - n_{\mathrm{i}}^2)$

where n is the electron density, p is the hole density and n_{i} is the intrinsic density.

$Auger-recombination		**optional**
material-name	**character**	**required**
number-of-parameters	**integer**	**required**
n-C	**double**	**optional**
p-C	**double**	**optional**
n-bow-C	**double**	**optional**
p-bow-C	**double**	**optional**
$end_Auger-recombination		**optional**

Explanation of specifiers.

material-name

type
character

presence
required

value
e.g. `GaAs`

Name of material to which this set of parameters applies. Name has to be listed in *$default-materials*.

number-of-parameters

type
integer

presence
required

value
e.g. `2`

Control parameter if the number of parameters provided is the same as demanded.

There are two sets of parameters, one for electrons (`n`) and one for holes (`p`).

n-C

type
double

presence
required

value
e.g. `1.0e-30`

unit
`[cm^6/s]`

The C_n parameter for electrons as specified in the equation above. The order of magnitude is around 10^{-30} `[cm^6/s]`.

n-bow-C

type
double

presence
optional

value
e.g. `0.0`

unit
[cm^6/s]

For ternary alloys there are also bowing parameters possible. `n-bow-C = 0.0` means zero bowing, i.e. linear interpolation is used.

p-C

unit
[cm^6/s]

Same as `n-C` but for holes.

p-bow-C

unit
[cm^6/s]

Same as `n-bow-C` but for holes.

Example

```
!---------------------------------------------!
$Auger-recombination                          !
                                              !
 material-name        = Si                    !
 number-of-parameters = 2                     !
 n-C                  = 2.8e-31               ! [cm^6/s]
 p-C                  = 9.9e-31               ! [cm^6/s]
                                              !
 material-name        = GaAs                  !
 number-of-parameters = 2                     !
 n-C                  = 1.0e-30               ! [cm^6/s]
 p-C                  = 1.0e-30               ! [cm^6/s]
                                              !
 material-name        = Al(x)Ga(1-x)As        !
 number-of-parameters = 2                     !
 n-bow-C              = 0.0                   ! [cm^6/s]
 p-bow-C              = 0.0                   ! [cm^6/s]
                                              !
$end_Auger-recombination                      !
!---------------------------------------------!
```

There is also a keywords section in the input file for *$Auger-recombination* which you can use to overwrite default material parameters.

$direct-recombination

The simplest process for the generation and recombination of electron-hole pairs is the direct process via the emission or absorption of a photon (radiative recombination). This is important for light emitting devices. The local spontaneous emission rate is approximated as

$R_{dir} = C(np - n_i^2)$

where C is called the *bimolecular recombination coefficient* (often abbreviated with the letter B), n is the electron density, p is the hole density and n_i is the intrinsic density.

This simple equation includes the full spectrum of photons generated by spontaneous band-to-band transition processes. By definition, the radiative recombination rate (= spontaneous electron recombination rate) per unit volume of an electron-hole pair is equivalent to the spontaneous photon generation rate per unit volume, because obviously, each time an electron recombines with a hole radiatively, a photon is emitted.

$direct-recombination		**optional**
material-name	**character**	**required**
number-of-parameters	**integer**	**required**
C-opt	**double**	**optional**
bow-C-opt	**double**	**optional**
$end_direct-recombination		**optional**

Explanation of specifiers.

material-name

type
character

presence
required

value
e.g. `GaAs`

Name of material to which this set of parameters applies. Name has to be listed in *$default-materials*.

number-of-parameters

type
integer

presence
required

value
e.g. `1`

Control parameter if the number of parameters provided is the same as demanded.

C-opt

type
double

presence
required

value
e.g. `7.2e-10`

unit
`[cm^3/s]`

The parameters are specified as shown in the tables above. The order of magnitude is around 10^{-10} `[cm^3/s]`.

bow-C-opt

type
double

presence
optional

value
e.g. `0.0`

unit
`[cm^3/s]`

For ternary alloys there are also bowing parameters possible. `bow-C-opt = 0.0` means zero bowing, i.e. linear interpolation is used.

Example

```
!-------------------------------------------!
$direct-recombination                       !
                                            !
 material-name        = GaAs                !
 number-of-parameters = 1                   !
 C-opt                = 7.2e-10             ! [cm^3/s]
                                            !
 material-name        = GaN                 !
 number-of-parameters = 1                   !
 C-opt                = 2e-10               ! [cm^3/s]
                                            !
 material-name        = Al(x)Ga(1-x)As      !
 number-of-parameters = 1                   !
 bow-C-opt            = 0.0                 ! [cm^3/s]
                                            !
$end_direct-recombination                   !
!-------------------------------------------!
```

There is also a keywords section in the input file for *$direct-recombination* which you can use to overwrite default material parameters.

$SRH-recombination

The generation/recombination process can be assisted by impurities. This is modeled by the Shockley-Read-Hall model (SRH). 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.

$R_{\mathrm{SRH}} = \frac{pn - n_{\mathrm{i}}^2}{\tau_{\mathrm{p}}(n+n_{\mathrm{i}}) + \tau_{\mathrm{n}}(p+n_{\mathrm{i}})}$

where

$\tau_{\mathrm{p,n}}\left(N_{\mathrm{D}} + N_{\mathrm{A}}\right) = \frac{\tau_{\mathrm{p0,n0}}}{1 + \frac{N_{\mathrm{D}}+N_{\mathrm{A}}}{N_{\mathrm{n,p,ref}}}}$

and n is the electron density, p is the hole density and n_{i} is the intrinsic density.

N_{D} and N_{A} are the donor and acceptor concentrations, respectively.

$N_{\mathrm{n,ref}}$ and $N_{\mathrm{p,ref}}$ are the reference doping concentrations for electrons and holes, respectively.

τ_{n0} and τ_{p0} are the zero doping scattering times for electrons and holes, respectively.

$SRH-recombination		**optional**
material-name	**character**	**required**
number-of-parameters	**integer**	**required**
n-N-ref	**double**	**optional**
n-tau	**double**	**optional**
p-N-ref	**double**	**optional**
p-tau	**double**	**optional**
n-bow-N-ref	**double**	**optional**
n-bow-tau	**double**	**optional**
p-bow-N-ref	**double**	**optional**
p-bow-tau	**double**	**optional**
$end_SRH-recombination		**optional**

Explanation of specifiers.

material-name

type
character

presence
required

value
e.g. `GaAs`

Name of material to which this set of parameters applies. Name has to be listed in *$default-materials*.

number-of-parameters

type
integer

presence
required

value
e.g. `4`

Control parameter if the number of parameters provided is the same as demanded.

There are two sets of parameters, one for electrons (`n`) and one for holes (`p`).

n-N-ref

type
double

presence
required

value
e.g. `7.1e15`

unit
`[cm^-3]`

The $N_{\mathrm{n,ref}}$ parameter for electrons as specified in the equation above.

n-tau

type
double

presence
required

value
e.g. `4.26e-4`

unit
`[s]`

The τ_{n} parameter for electrons as specified in the equation above.

n-bow-N-ref

type
double

presence
optional

value
e.g. `0.0`

unit
`[cm^-3]`

For ternary alloys there are also bowing parameters possible. `n-bow-N-ref = 0.0` means zero bowing, i.e. linear interpolation is used.

n-bow-tau

type
double

presence
optional

value
e.g. `0.0`

unit
`[s]`

For ternary alloys there are also bowing parameters possible. `n-bow-tau = 0.0` means zero bowing, i.e. linear interpolation is used.

p-N-ref

unit
`[cm^-3]`

Same as `n-N-ref` but for holes.

p-tau

unit
`[s]`

Same as `n-tau` but for holes.

p-bow-N-ref

unit
`[cm^-3]`

Same as `n-bow-N-ref` but for holes.

p-bow-tau

unit
`[s]`

Same as `n-bow-tau` but for holes.

Example

```
!-------------------------------------------!
$SRH-recombination                          !
                                            !
 material-name        = Si                  !
 number-of-parameters = 4                   !
 n-N-ref              = 7.1e15              ! [cm^-3]
 n-tau                = 4.26e-4             ! [s]
 p-N-ref              = 7.1e15              ! [cm^-3]
 p-tau                = 3.95e-4             ! [s]

 material-name        = GaAs                !
 number-of-parameters = 4                   !
 n-N-ref              = 1.0e19              ! [cm^-3]
 n-tau                = 1.0e-9              ! [s]
 p-N-ref              = 1.0e18              ! [cm^-3]
 p-tau                = 1.0e-9              ! [s]

 material-name        = Al(x)Ga(1-x)As      !
 number-of-parameters = 4                   !
```

(continues on next page)

(continued from previous page)

```
 n-bow-N-ref            = 0.0                    ! [cm^-3]
 n-bow-tau              = 0.0                    ! [s]
 p-bow-N-ref            = 0.0                    ! [cm^-3]
 p-bow-tau              = 0.0                    ! [s]
                                                 !
$end_SRH-recombination                           !
!------------------------------------------------!
```

There is also a keywords section in the input file for *$SRH-recombination* which you can use to overwrite default material parameters.

$mobility-model-arora

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-arora.htm

$mobility-model-constant

The constant mobility model is due to lattice scattering (phonon scattering) and depends only on the temperature. The lattice atoms oscillate about their equilibrium sites at finite temperature leading to a scattering of carriers which results in a temperature dependent mobility μ_{const}. μ_{L} is the mobility due to bulk phonon (lattice) scattering. For all semiconductors the temperature dependent lattice mobility is modeled by a power law. 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).

This model is suited for undoped structure.

Note: The γ exponents `n-gamma-lattice-temp`, `p-gamma-lattice-temp` have opposite sign in both the PhD thesis of V. Palankovski and in the MINIMOS documentation compared to the implementation of *nextnano*[3].

In this model the mobility is constant and depends only on the temperature T. The parameters in the database are given for electrons and holes,

$$\mu_{\text{const}}(T) = \mu_{\text{L}} \left(\frac{T}{T_0} \right)^{-\gamma},$$

where $T_0 = 300$ K.

```
$mobility-model-constant                    optional
 material-name                  character   required
 number-of-parameters           integer     required
 n-mu-lattice-temp              double      optional
 n-gamma-lattice-temp           double      optional
 p-mu-lattice-temp              double      optional
 p-gamma-lattice-temp           double      optional

 !------------------------------
 ! Bowing parameters for alloys
 !------------------------------

 n-bow-mu-lattice-temp          double      optional
 n-bow-gamma-lattice-temp       double      optional
 p-bow-mu-lattice-temp          double      optional
 p-bow-gamma-lattice-temp       double      optional
$end_mobility-model-constant                optional
```

Explanation of specifiers.

material-name

type
character

presence
required

value
e.g. `Si`

Name of material to which this set of parameters applies. Name has to be listed in *$default-materials*.

number-of-parameters

type
integer

presence
required

value
e.g. `4`

Control parameter if the number of parameters provided is the same as demanded.

There are two sets of parameters, one for electrons (`n`) and one for holes (`p`).

n-mu-lattice-temp

type
double

presence
required

value
e.g. `1417`

unit
`[cm^2/Vs]`

$\mu_{L,n}$ is the bulk phonon mobility for electrons.

n-gamma-lattice-temp

type
double

presence
required

value
e.g. `2.5`

unit
`[]`

γ_n is the exponent of the temperature dependence for electrons.

Bowing parameters

n-bow-mu-lattice-temp

type
double

presence
optional

value
e.g. `0.0`

unit
[cm^2/Vs]

For ternary alloys there are also bowing parameters possible. `n-bow-mu-lattice-temp = 0.0` means zero bowing, i.e. linear interpolation is used.

n-bow-gamma-lattice-temp

type
double

presence
optional

value
e.g. `0.0`

unit
[]

Bowing parameters `n-bow-gamma-lattice-temp = 0.0` means zero bowing, i.e. linear interpolation is used.

p-mu-lattice-temp

unit
[cm^2/Vs]

Same as `n-mu-lattice-temp` but for holes, $\mu_{\mathrm{L,p}}$.

p-gamma-lattice-temp

unit
[]

Same as `n-gamma-lattice-temp` but for holes, γ_{p}.

p-bow-mu-lattice-temp

unit
[cm^2/Vs]

Same as `n-bow-mu-lattice-temp` but for holes.

p-bow-gamma-lattice-temp

unit
[]

Same as `n-bow-gamma-lattice-temp` but for holes.

Example

```
!-------------------------------------------
$mobility-model-constant

 material-name             = GaAs
 number-of-parameters      = 4
 n-mu-lattice-temp         = 8500          ! [cm^2/Vs]  PhD thesis V.␣
↪Palankovski
 n-gamma-lattice-temp      = 2.2           ! []         PhD thesis V.␣
↪Palankovski but opposite sign compared to MINIMOS
 p-mu-lattice-temp         = 800           ! [cm^2/Vs]  PhD thesis V.␣
↪Palankovski
 p-gamma-lattice-temp      = 0.9           ! []         PhD thesis V.␣
↪Palankovski but opposite sign compared to MINIMOS

 material-name             = Al(x)Ga(1-x)As
```

(continues on next page)

(continued from previous page)

```
 number-of-parameters      = 4
 n-bow-mu-lattice-temp     = 0.0            ! [cm^2/Vs]
 n-bow-gamma-lattice-temp  = 0.0            ! []
 p-bow-mu-lattice-temp     = 0.0            ! [cm^2/Vs]
 p-bow-gamma-lattice-temp  = 0.0            ! []

$end_mobility-model-constant
!-------------------------------------------
```

$mobility-model-dar

For the Darwish mobility model, see *[DarwishIEEE1997]* and *[Klaassen1992]*.

```
$mobility-model-dar                          optional
 material-name                  character    required
 number-of-parameters           integer      required

 n-acoust-B                     double       optional
 n-acoust-C                     double       optional
 n-ac-C-exponent                double       optional
 n-acoust-kappa                 double       optional
 n-bulk-mu-min                  double       optional
 n-bulk-mu-max                  double       optional
 n-bulk-mu-Nref1                double       optional
 n-bulk-thetan                  double       optional
 n-bulk-alpha1                  double       optional
 n-bulk-f-CW                    double       optional
 n-bulk-f-BH                    double       optional
 n-bulk-P-CW-const              double       optional
 n-bulk-P-BH-const              double       optional
 n-bulk-G-s1                    double       optional
 n-bulk-G-s2                    double       optional
 n-bulk-G-s3                    double       optional
 n-bulk-G-s4                    double       optional
 n-bulk-G-s5                    double       optional
 n-bulk-G-s6                    double       optional
 n-bulk-G-s7                    double       optional
 n-bulk-F-r1                    double       optional
 n-bulk-F-r2                    double       optional
 n-bulk-F-r3                    double       optional
 n-bulk-F-r4                    double       optional
 n-bulk-F-r5                    double       optional
 n-bulk-F-r6                    double       optional
 n-sr-delta                     double       optional
 n-sr-alpha                     double       optional
 n-sr-etha                      double       optional
 n-sr-A                         double       optional
 n-a-E                          double       optional
 n-b-E                          double       optional
 n-E0-saturation                double       optional
 n-bulk-Z-c                     double       optional
 n-bulk-Z-Nref                  double       optional

 p-acoust-B                     double       optional
 p-acoust-C                     double       optional
```

(continues on next page)

(continued from previous page)

```
p-ac-C-exponent                 double    optional
p-acoust-kappa                  double    optional
p-bulk-mu-min                   double    optional
p-bulk-mu-max                   double    optional
p-bulk-mu-Nref1                 double    optional
p-bulk-thetan                   double    optional
p-bulk-alpha1                   double    optional
p-bulk-f-CW                     double    optional
p-bulk-f-BH                     double    optional
p-bulk-P-CW-const               double    optional
p-bulk-P-BH-const               double    optional
p-bulk-G-s1                     double    optional
p-bulk-G-s2                     double    optional
p-bulk-G-s3                     double    optional
p-bulk-G-s4                     double    optional
p-bulk-G-s5                     double    optional
p-bulk-G-s6                     double    optional
p-bulk-G-s7                     double    optional
p-bulk-F-r1                     double    optional
p-bulk-F-r2                     double    optional
p-bulk-F-r3                     double    optional
p-bulk-F-r4                     double    optional
p-bulk-F-r5                     double    optional
p-bulk-F-r6                     double    optional
p-sr-delta                      double    optional
p-sr-alpha                      double    optional
p-sr-etha                       double    optional
p-sr-A                          double    optional
p-a-E                           double    optional
p-b-E                           double    optional
p-E0-saturation                 double    optional
p-bulk-Z-c                      double    optional
p-bulk-Z-Nref                   double    optional

!------------------------------
! Bowing parameters for alloys
!------------------------------

n-bow-acoust-B                  double    optional
n-bow-acoust-C                  double    optional
n-bow-ac-C-exponent             double    optional
n-bow-acoust-kappa              double    optional
n-bow-bulk-mu-min               double    optional
n-bow-bulk-mu-max               double    optional
n-bow-bulk-mu-Nref1             double    optional
n-bow-bulk-thetan               double    optional
n-bow-bulk-alpha1               double    optional
n-bow-bulk-f-CW                 double    optional
n-bow-bulk-f-BH                 double    optional
n-bow-bulk-P-CW-const           double    optional
n-bow-bulk-P-BH-const           double    optional
n-bow-bulk-G-s1                 double    optional
n-bow-bulk-G-s2                 double    optional
n-bow-bulk-G-s3                 double    optional
n-bow-bulk-G-s4                 double    optional
n-bow-bulk-G-s5                 double    optional
```

(continues on next page)

(continued from previous page)

n-bow-bulk-G-s6	double	optional
n-bow-bulk-G-s7	double	optional
n-bow-bulk-F-r1	double	optional
n-bow-bulk-F-r2	double	optional
n-bow-bulk-F-r3	double	optional
n-bow-bulk-F-r4	double	optional
n-bow-bulk-F-r5	double	optional
n-bow-bulk-F-r6	double	optional
n-bow-sr-delta	double	optional
n-bow-sr-alpha	double	optional
n-bow-sr-etha	double	optional
n-bow-sr-A	double	optional
n-bow-a-E	double	optional
n-bow-b-E	double	optional
n-bow-E0-saturation	double	optional
n-bow-bulk-Z-c	double	optional
n-bow-bulk-Z-Nref	double	optional
p-bow-acoust-B	double	optional
p-bow-acoust-C	double	optional
p-bow-ac-C-exponent	double	optional
p-bow-acoust-kappa	double	optional
p-bow-bulk-mu-min	double	optional
p-bow-bulk-mu-max	double	optional
p-bow-bulk-mu-Nref1	double	optional
p-bow-bulk-thetan	double	optional
p-bow-bulk-alpha1	double	optional
p-bow-bulk-f-CW	double	optional
p-bow-bulk-f-BH	double	optional
p-bow-bulk-P-CW-const	double	optional
p-bow-bulk-P-BH-const	double	optional
p-bow-bulk-G-s1	double	optional
p-bow-bulk-G-s2	double	optional
p-bow-bulk-G-s3	double	optional
p-bow-bulk-G-s4	double	optional
p-bow-bulk-G-s5	double	optional
p-bow-bulk-G-s6	double	optional
p-bow-bulk-G-s7	double	optional
p-bow-bulk-F-r1	double	optional
p-bow-bulk-F-r2	double	optional
p-bow-bulk-F-r3	double	optional
p-bow-bulk-F-r4	double	optional
p-bow-bulk-F-r5	double	optional
p-bow-bulk-F-r6	double	optional
p-bow-sr-delta	double	optional
p-bow-sr-alpha	double	optional
p-bow-sr-etha	double	optional
p-bow-sr-A	double	optional
p-bow-a-E	double	optional
p-bow-b-E	double	optional
p-bow-E0-saturation	double	optional
p-bow-bulk-Z-c	double	optional
p-bow-bulk-Z-Nref	double	optional
$end_mobility-model-dar		optional

Note: The mobility models `mobility-model-dar` (Darwish) has been developed for silicon only.

Note: In nextnano GmbH we use the nominal dopant concentration as specified in the input file and not the ionized one.

Example

```
!-----------------------------------------!
$mobility-model-dar

 material-name              = Si
 number-of-parameters       = 70

 n-acoust-B                 =  3.61e7       !  [cm/s]
 n-acoust-C                 =  1.70e4       !  [(cm^2/Vs)(V/cm)**(1/
↪3)cm**(3tau)]
 n-ac-C-exponent            =  0.0233       !  []
 n-acoust-kappa             =  1.7          !  []
 n-bulk-mu-min              =  52.2         !  [cm^2/Vs]
 n-bulk-mu-max              =  1417.0       !  [cm^2/Vs]
 n-bulk-mu-Nref1            =  9.68e16      !  [1/cm^3]
 n-bulk-thetan              =  1.0          !  []
 n-bulk-alpha1              =  0.68         !  []
 n-bulk-f-CW                =  2.495        !  []
 n-bulk-f-BH                =  3.828        !  []
 n-bulk-P-CW-const          =  3.97e13      !  []
 n-bulk-P-BH-const          =  1.36e20      !  []
 n-bulk-G-s1                =  0.89233      !  []
 n-bulk-G-s2                =  0.41372      !  []
 n-bulk-G-s3                =  0.19778      !  []
 n-bulk-G-s4                =  0.28227      !  []
 n-bulk-G-s5                =  0.005978     !  []
 n-bulk-G-s6                =  1.80618      !  []
 n-bulk-G-s7                =  0.72169      !  []
 n-bulk-F-r1                =  0.7643       !  []
 n-bulk-F-r2                =  2.2999       !  []
 n-bulk-F-r3                =  6.5502       !  []
 n-bulk-F-r4                =  2.3670       !  []
 n-bulk-F-r5                = -0.01552      !  []
 n-bulk-F-r6                =  0.6478       !  []
 n-sr-delta                 =  3.58e18      !  [V/s]
 n-sr-alpha                 =  6.85e-21     !  []
 n-sr-etha                  =  0.0767       !  []
 n-sr-A                     =  2.58         !  []
 n-a-E                      =  2.0          !  []
 n-b-E                      =  0.5          !  []
 n-E0-saturation            =  8.0e3        !  [V/cm]
 n-bulk-Z-c                 =  0.21         !  []
 n-bulk-Z-Nref              =  4.0e20       !  [1/cm^3]

 p-acoust-B                 =  1.51e7       !  [cm/s]
 p-acoust-C                 =  4.18e3       !  [(cm^2/Vs)(V/cm)**(1/
↪3)cm**(3tau)]
 p-ac-C-exponent            =  0.0119       !  []
```

(continues on next page)

(continued from previous page)

```
 p-acoust-kappa              =  0.9          !  []
 p-bulk-mu-min               =  44.9         !  [cm^2/Vs]
 p-bulk-mu-max               =  470.5        !  [cm^2/Vs]
 p-bulk-mu-Nref1             =  2.23e17      !  [1/cm^3]
 p-bulk-thetan               =  1.0          !  []
 p-bulk-alpha1               =  0.719        !  []
 p-bulk-f-CW                 =  2.495        !  []
 p-bulk-f-BH                 =  3.828        !  []
 p-bulk-P-CW-const           =  3.97e13      !  []
 p-bulk-P-BH-const           =  1.36e20      !  []
 p-bulk-G-s1                 =  0.89233      !  []
 p-bulk-G-s2                 =  0.41372      !  []
 p-bulk-G-s3                 =  0.19778      !  []
 p-bulk-G-s4                 =  0.28227      !  []
 p-bulk-G-s5                 =  0.005978     !  []
 p-bulk-G-s6                 =  1.80618      !  []
 p-bulk-G-s7                 =  0.72169      !  []
 p-bulk-F-r1                 =  0.7643       !  []
 p-bulk-F-r2                 =  2.2999       !  []
 p-bulk-F-r3                 =  6.5502       !  []
 p-bulk-F-r4                 =  2.3670       !  []
 p-bulk-F-r5                 = -0.01552      !  []
 p-bulk-F-r6                 =  0.6478       !  []
 p-sr-delta                  =  4.10e15      !  [V/s]
 p-sr-alpha                  =  7.82e-21     !  []
 p-sr-etha                   =  0.123        !  []
 p-sr-A                      =  2.18         !  []
 p-a-E                       =  1.0          !  []
 p-b-E                       =  1.0          !  []
 p-E0-saturation             =  1.95e4       !  [V/cm]
 p-bulk-Z-c                  =  0.50         !  []
 p-bulk-Z-Nref               =  7.2e20       !  [1/cm^3]

$end_$mobility-model-dar
!-----------------------------------------!
```

$mobility-model-lom

For the Lombardi mobility model, see *[LombardiIEEE1988]*.

```
$mobility-model-lom                          optional
 material-name                  character    required
 number-of-parameters           integer      required

 n-acoust-B                     double       optional
 n-acoust-C                     double       optional
 n-ac-C-exponent                double       optional
 n-bulk-mu-zero                 double       optional
 n-bulk-mu-max                  double       optional
 n-bulk-mu-1                    double       optional
 n-bulk-gamma                   double       optional
 n-bulk-Cr                      double       optional
 n-bulk-C0                      double       optional
 n-bulk-alpha                   double       optional
 n-bulk-beta                    double       optional
```

(continues on next page)

(continued from previous page)

```
n-sr-delta                      double    optional
n-alpha-E                       double    optional
n-beta-E                        double    optional
n-E0-saturation                 double    optional
n-v0-saturation                 double    optional
n-temp-dependence-v             double    optional
n-kappa-v                       double    optional
n-T0-v-saturation               double    optional

p-acoust-B                      double    optional
p-acoust-C                      double    optional
p-ac-C-exponent                 double    optional
p-bulk-mu-zero                  double    optional
p-bulk-mu-max                   double    optional
p-bulk-mu-1                     double    optional
p-bulk-gamma                    double    optional
p-bulk-Cr                       double    optional
p-bulk-C0                       double    optional
p-bulk-alpha                    double    optional
p-bulk-beta                     double    optional
p-sr-delta                      double    optional
p-alpha-E                       double    optional
p-beta-E                        double    optional
p-E0-saturation                 double    optional
p-v0-saturation                 double    optional
p-temp-dependence-v             double    optional
p-kappa-v                       double    optional
p-T0-v-saturation               double    optional

!------------------------------
! Bowing parameters for alloys
!------------------------------

n-bow-acoust-B                  double    optional
n-bow-acoust-C                  double    optional
n-bow-ac-C-exponent             double    optional
n-bow-bulk-mu-zero              double    optional
n-bow-bulk-mu-max               double    optional
n-bow-bulk-mu-1                 double    optional
n-bow-bulk-gamma                double    optional
n-bow-bulk-Cr                   double    optional
n-bow-bulk-C0                   double    optional
n-bow-bulk-alpha                double    optional
n-bow-bulk-beta                 double    optional
n-bow-sr-delta                  double    optional
n-bow-alpha-E                   double    optional
n-bow-beta-E                    double    optional
n-bow-E0-saturation             double    optional
n-bow-v0-saturation             double    optional
n-bow-temp-dependence-v         double    optional
n-bow-kappa-v                   double    optional
n-bow-T0-v-saturation           double    optional

p-bow-acoust-B                  double    optional
p-bow-acoust-C                  double    optional
p-bow-ac-C-exponent             double    optional
```

(continues on next page)

(continued from previous page)

```
 p-bow-bulk-mu-zero                double    optional
 p-bow-bulk-mu-max                 double    optional
 p-bow-bulk-mu-1                   double    optional
 p-bow-bulk-gamma                  double    optional
 p-bow-bulk-Cr                     double    optional
 p-bow-bulk-C0                     double    optional
 p-bow-bulk-alpha                  double    optional
 p-bow-bulk-beta                   double    optional
 p-bow-sr-delta                    double    optional
 p-bow-alpha-E                     double    optional
 p-bow-beta-E                      double    optional
 p-bow-E0-saturation               double    optional
 p-bow-v0-saturation               double    optional
 p-bow-temp-dependence-v           double    optional
 p-bow-kappa-v                     double    optional
 p-bow-T0-v-saturation             double    optional
$end_mobility-model-lom                      optional
```

Note: The mobility models `mobility-model-lom` (Lombardi) has been developed for silicon only.

Note: In nextnano GmbH we use the nominal dopant concentration as specified in the input file and not the ionized one.

Example

```
!---------------------------------------!
$mobility-model-dar

 material-name             = Si
 number-of-parameters      = 38

 n-acoust-B                = 4.75e7     ! [cm/s]
 n-acoust-C                = 1.74e5     ! [see paper]
 n-ac-C-exponent           = 0.125      ! []
 n-bulk-mu-zero            = 52.2       ! [cm^2/Vs]
 n-bulk-mu-max             = 1417.0     ! [cm^2/Vs]
 n-bulk-mu-1               = 43.4       ! [cm^2/Vs]
 n-bulk-gamma              = 2.5        ! []
 n-bulk-Cr                 = 9.68e16    ! [1/cm^3]
 n-bulk-C0                 = 3.43e20    ! [1/cm^3]
 n-bulk-alpha              = 0.680      ! []
 n-bulk-beta               = 2.0        ! []
 n-sr-delta                = 5.82e14    ! [V/s]
 n-alpha-E                 = 2.0        ! []
 n-beta-E                  = 0.5        ! []
 n-E0-saturation           = 8.0e3      ! [V/cm]
 n-v0-saturation           = 1.03e7     ! [cm/s]   see SIMBA documentation
 n-temp-dependence-v       = 0.0        ! [cm/Ks]  see SIMBA documentation
 n-kappa-v                 = 2          ! []       see SIMBA documentation
 n-T0-v-saturation         = 300.0      ! [K]      see SIMBA documentation

 p-acoust-B                = 9.93e7     ! [cm/s]
 p-acoust-C                = 8.84e5     ! [see paper]
```

(continues on next page)

(continued from previous page)

```
 p-ac-C-exponent            = 0.0317      ! []
 p-bulk-mu-zero             = 44.9        ! [cm^2/Vs]
 p-bulk-mu-max              = 470.5       ! [cm^2/Vs]
 p-bulk-mu-1                = 29.0        ! [cm^2/Vs]
 p-bulk-gamma               = 2.2         ! []
 p-bulk-Cr                  = 2.23e17     ! [1/cm^3]
 p-bulk-C0                  = 6.1e20      ! [1/cm^3]
 p-bulk-alpha               = 0.719       ! []
 p-bulk-beta                = 2.0         ! []
 p-sr-delta                 = 2.05e14     ! [V/s]
 p-alpha-E                  = 1.0         ! []
 p-beta-E                   = 1.0         ! []
 p-E0-saturation            = 1.95e4      ! [V/cm]
 p-v0-saturation            = 1.03e7      ! [cm/s]   see SIMBA documentation
 p-temp-dependence-v        = 0.0         ! [cm/Ks]  see SIMBA documentation
 p-kappa-v                  = 1           ! []       see SIMBA documentation
 p-T0-v-saturation          = 300.0       ! [K]      see SIMBA documentation

$end_$mobility-model-lom
!---------------------------------------!
```

$mobility-model-masetti

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-masetti.htm

$mobility-model-minimos

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-minimos.htm

$mobility-model-simba

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-simba.htm

$mobility-models

Several mobility models can be chosen that consider phonon scattering, impurity scattering and electric field dependence.

```
$mobility-models                        required
 model-name                 character   required
 model-type-number          integer     required
$end_mobility-models                    required
```

Note: The mobility models `mobility-model-lom` (Lombardi) and `mobility-model-dar` (Darwish) have been developed for silicon only.

Example

```
!-----------------------------------------------!
$mobility-models
 model-name        = mobility-model-simba-0    ! SIMBA, parallel E field␣
↪dependence according to model 0
 model-type-number = 1

 model-name        = mobility-model-simba-1    ! SIMBA, parallel E field␣
↪dependence according to model 1
 model-type-number = 2

 model-name        = mobility-model-simba-2    ! SIMBA, parallel E field␣
↪dependence according to model 2
 model-type-number = 3

 model-name        = mobility-model-simba-3    ! SIMBA, parallel E field␣
↪dependence according to model 3
 model-type-number = 4

 model-name        = mobility-model-simba-4    ! SIMBA, parallel E field␣
↪dependence according to model 4
 model-type-number = 5

 model-name        = mobility-model-simba-5    ! SIMBA, parallel E field␣
↪dependence according to model 5
 model-type-number = 6

 model-name        = mobility-model-simba-0e   ! with perpendicular E field␣
↪dependence
 model-type-number = 7

 model-name        = mobility-model-simba-1e   ! with perpendicular E field␣
↪dependence
 model-type-number = 8

 model-name        = mobility-model-simba-2e   ! with perpendicular E field␣
↪dependence
 model-type-number = 9

 model-name        = mobility-model-simba-3e   ! with perpendicular E field␣
↪dependence
 model-type-number = 10

 model-name        = mobility-model-simba-4e   ! with perpendicular E field␣
↪dependence
 model-type-number = 11

 model-name        = mobility-model-simba-5e   ! with perpendicular E field␣
↪dependence
 model-type-number = 12

 model-name        = mobility-model-lom        ! Lobardi model, for Si only
 model-type-number = 13

 model-name        = mobility-model-dar        ! Darwish model, for Si only
 model-type-number = 14

 model-name        = mobility-model-constant   ! constant mobility model,␣
```

(continues on next page)

(continued from previous page)

```
↪for undoped structures only (phonon scattering)
 model-type-number = 15

 model-name        = mobility-model-masetti    ! Masetti model    - doping␣
↪dependent model (phonon and impurity scattering)
 model-type-number = 16

 model-name        = mobility-model-arora      ! Arora model      - doping␣
↪dependent model (phonon and impurity scattering)
 model-type-number = 17

 model-name        = mobility-model-minimos    ! Minimos 6 model - doping␣
↪dependent model (phonon and impurity scattering)
 model-type-number = 18

$end_mobility-models
!-----------------------------------------------!
```

$transport-models

Charge carrier models.

$transport-models		**required**
model-name	**character**	**required**
model-type-number	**integer**	**required**
$end_transport-models		**required**

Transport model for drift-diffusion equation.

model-name

type
character

presence
required

value
simple-drift-model

More information available here: *$simple-drift-models*

model-type-number

type
integer

presence
required

value
1

Only 1 model implemented so far.

Example

```
!---------------------------------------!
$transport-models                       !                                ␣
↪       !
 model-name        = simple-drift-model  !
```

(continues on next page)

(continued from previous page)

```
 model-type-number = 1                    !
$end_transport-models                     !                                  ␣
↪      !
!-----------------------------------------!
```

$buffer-constant-A(T)

Constant $A(T)$ used for buffer calculations: The pK_a value depends on the ionic strength.

$buffer-constant-A(T)		**optional**
constant-Centigrade-to-Kelvin	**double**	**required**
T_A(T)	**double_array**	**required**
$end_buffer-constant-A(T)		**optional**

Example

```
!-------------------------------------------------------------!
$buffer-constant-A(T)                                         !
 constant-Centigrade-to-Kelvin = 273.15                       ! Kelvin = Celsius +␣
↪273.15
                                                              !
 !============================================================!
 !                  first column: T[C]   second column: A(T)  !
 !============================================================!
 T_A(T)                          =   0.0          0.4918      !   0° C = 273.15 K
                                    10.0          0.4989      !  10° C = 283.15 K
                                    20.0          0.5070      !  20° C = 293.15 K
                                    25.0          0.5114      !  25° C = 298.15 K
                                    30.0          0.5161      !  30° C = 303.15 K
                                    37.0          0.5321      !  37° C = 310.15 K
                                    40.0          0.5262      !  40° C = 313.15 K
                                    50.0          0.5373      !  50° C = 323.15 K
                                    60.0          0.5494      !  60° C = 333.15 K
                                    70.0          0.5625      !  70° C = 343.15 K
                                    80.0          0.5767      !  80° C = 353.15 K
                                    90.0          0.5920      !  90° C = 363.15 K
                                   100.0          0.6086      ! 100° C = 373.15 K
$end_buffer-constant-A(T)                                     !
!-------------------------------------------------------------!
```

The left column of the specifier `T_A(T)` contains the temperature in degrees of Centigrade (Celsius) between 0° C and 100° C.

The right column of the specifier `T_A(T)` contains the corresponding value of the constant A as a function of temperature T, i.e. A(T).

The values are taken from page 30 of *[Beynon1996]*.

They can also be approximated by a second-order polynomial (*[Beynon1988]*):

$A(T) = 0.4918 + 0.0006614T + 0.000004975T^2$

If the keyword `$buffer-constant-A(T)` is present in the input file, the values for this keyword in the database are overwritten.

Physical significance of this parameter

The ionic strength of an electrolyte influences the pK_a value of the buffer. This dependence can be described by the following equation (sometimes known as the Debye-Hückel relationship) where the constant A(T) enters.

$$pK'_a = pK_a + (2z_a - 1)[AI^{1/2}/(1 + I^{1/2}) - 0.1I]$$

where I is the ionic strength and z_a is the charge on the conjugate acid species. pK'_a is the modified pK_a value. The value of A (sometimes called Debye-Hückel parameter) is about 0.5 but it is temperature dependent.

Internally, the program takes the temperature T_0 that is given in the input file under the keyword *$global-parameters* (in units of Kelvin) and interpolates linearly between the two appropriate neighboring $A(T)$ values to find the value for $A(T_0)$. The conversion between temperature in Kelvin and Centigrade is done by the constant: `constant-Centigrade-to-Kelvin = 273.15`

Example

```
lattice-temperature = 288.15    ! 288.15 [K] = 15° [C] + 273.15 [K]

A(T = 10° C) = 0.4989
A(T = 20° C) = 0.5070
```

=> Internally the program calculates the value for A(T = 15° C) = (0.4989 + 0.5070)/2 = 0.50295.

The following interpolation formula is used:

```
A(T = x°C) = A(T_i) + slope * ('lattice-temperature' - 'constant-Centigrade-to-Kelvin
↪' - Ti) =
           = A(T_i) + slope * ('lattice-temperature' - '273.15'                    ␣
↪ - Ti) =
```

where

```
slope = ( A(T_(i+1)) - A(T_i) ) / ( T_(i+1) - T_i )
```

and it holds:

```
T_i  <  'lattice-temperature' - '273.15'T_i  <  T_(i+1)
```

T_{i+1} and T_i are the closest temperature points above and below the specified temperature `lattice-temperature`.

- If the `lattice-temperature` is smaller than the smallest value of A(T), the smallest A(T) value is taken.
- If the `lattice-temperature` is larger than the largest value of A(T), the largest A(T) value is taken.

The value of A *always* depends on temperature. This can only be switched off by specifying only *one* value of T and A(T) in the database or in the input file. The values for T and A(T) that are specified in the database can be overwritten in the input file. For details, have a look at the input file keyword *$buffer-constant-A(T)*.

$buffer-solutions

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/buffer-solutions.htm

$physical-constants

The base system for units is SI.

The following physical constants are used within *nextnano*[3].

```
$physical-constants                 required
 electron-charge          double    required  ! [As]    = [C]            e: ␣
↪      elementary charge
 electron-mass            double    required  ! [kg]                     m0:␣
↪      electron mass
 planck-constant          double    required  ! [Js]                     h_
```

(continues on next page)

(continued from previous page)

```
↪bar:    Planck's constant
 speed-of-light           double   required  ! [m/s]            (exact) c: ␣
↪      speed of light in vacuum
 boltzmann-constant       double   required  ! [J/K]                    k_
↪B:      Boltzmann constant
 vacuum-permittivity      double   required  ! [As/Vm] = [F/m]  (exact)␣
↪epsilon0: electric constant
 avogadro-number          double   required  ! [1/mol]                  N_
↪A:      Avogadro number
$end_physical-constants             required
```

Example

```
!-----------------------------------------!
$physical-constants                       !
 electron-charge     = -1.6021766208e-19  ! [C] = [As]       -1.
↪6021766208(98)e-19
 electron-mass       =  9.10938356e-31    ! [kg]              9.
↪10938356(11)e-31
 planck-constant     =  6.626070040e-34   ! [Js]              6.
↪626070040(81)e-34
 speed-of-light      =  2.99792458e+8     ! [m/s]    (exact)
 boltzmann-constant  =  1.38064852e-23    ! [J/K]             1.
↪38064852(79)e-23
 vacuum-permittivity =  8.854187817e-12   ! [As/Vm]  (exact)  8.854187817...
↪e-12
 avogadro-number     =  6.022140857e+23   ! []                6.
↪022140857(74)e+23
$end_physical-constants                   !
!-----------------------------------------!
```

These SI units were taken on 2019-05-10 from https://physics.nist.gov/cuu/Constants/index.html. The number in parentheses is the numerical value of the standard uncertainty referred to the corresponding last digits of the quoted result.

Further constants

- `reduced_planck_constant` (Planck constant over 2 pi) is calculated internally inside the program: `planck-constant/(2pi) = h/2pi`

```
reduced_planck_constant      =  1.054571800139113e-034   ! [Js]␣
↪(calculated internally from other constants)
                                1.054571800(13)e-34      ! [Js] (NIST)
```

- `bohr_radius` is calculated internally inside the program:

```
4 * pi * vacuum_permittivity * reduced_planck_constant^2 / (electron_
↪mass * electron_charge^2)
bohr_radius                  =  0.5291772105267628e-010  ! [m]␣
↪(calculated internally from other constants)
                                0.52917721067(12)e-10    ! [m] (NIST)
```

- `hydrogen_ionization_energy_J` (Rydberg constant times hc in `[J]`) is calculated internally inside the program:

```
electron_mass * electron_charge^4 / (32 * pi^2 * reduced_planck_constant^
↪2 * vacuum_permittivity^2)
hydrogen_ionization_energy_J =  2.179872325695729e-018   ! [J]␣
```

(continues on next page)

(continued from previous page)

```
↪(calculated internally from other constants)
                                2.179872325(27)e-18       ! [J] (NIST)
```

- `hydrogen_ionization_energy_eV` (Rydberg constant times hc in `[eV]`) is calculated internally inside the program:

```
hydrogen_ionization_energy_J / electron-charge
hydrogen_ionization_energy_eV = 13.6056930140903          ! [eV]␣
↪(calculated internally from other constants)
                                13.605693009(84)          ! [eV] (NIST)␣
↪corresponds to hydrogen_ionization_energy_J
```

- `Hartree_eV` (Hartree energy in `[eV]`) is calculated internally inside the program:

```
2 * hydrogen_ionization_energy_eV
Hartree_eV                    =  27.2113860281805         ! [eV]␣
↪(calculated by nextnano³ from other constants)
                                 27.21138602(17)          ! [eV] (NIST)
```

Derived constants

- `[h_bar^2/(2*m0)]`

```
h2b2m_Jm2   = reduced_planck_constant^2 / (2*electron_mass) =
            = 6.104264214606464e-039 [J  m^2]
h2b2m_evAA2 = h2b2m_Jm2 / ABS(electron_charge) * (1d10)^2 =
            = 3.80998208022688       [eV AA^2]    ! AA = Angstrom
```

- From the Boltzmann constant k_B, one obtains $k_B T$ at room temperature in units of `[eV]`:

```
kBT = 0.02585199101... [eV] (T = 298.15 K = 25°C)
kBT = 0.02569257040... [eV] (T = 300 K)
```

For input scaling factors, see *$input-scaling-factors*.

Conversion factors

- µm <=> eV: h * c / e * 10^6 = 1.23984197

  Example

  1.23984 / 8.4 µm = 0.1476 eV

  1.23984 / 0.1476 eV = 8.4 µm

- µm <=> THz: c * 10^{-6} = 299.792458

  Example

  299.79 / 8.4 µm = 35.69 THz

  299.79 / 35.69 THz = 8.4 µm

- 10^{18} cm^{-3} <=> M: 602.21415

  Example

  30.11 / 602.2 = 0.050 M

  0.050 M * 602.2 = 30.11

- 10^{18} cm^{-3} <=> mM: 0.60221415

  Example

30.11 / 0.6022 = 50 mM

50 mM * 0.6022 = 30.11

$input-scaling-factors

The base system for units is SI. However, most input quantities are assumed to be given in scaled form. For physical constants, see *$physical-constants*.

```
$input-scaling-factors                  required
 q0                          double     required
 m0                          double     required
 l0                          double     required
 e0                          double     required
 n3d0                        double     required
 n2d0                        double     required
 n1d0                        double     required
 eps0                        double     required
 phi0                        double     required
 prs0                        double     required
 t0                          double     required
 kp_k^0_zb                   double     required  ! zb = zinc blende
 kp_k^2_zb                   double     required  ! zb = zinc blende
 kp_k^0_wz                   double     required  ! wz = wurtzite
 kp_k^2A_wz                  double     required  ! wz = wurtzite
 kp_k^2B_wz                  double     required  ! wz = wurtzite
$end_input-scaling-factors              required
```

Example

```
!-------------------------------------!
$input-scaling-factors                 !
 q0          =  1.6021766208e-19       ! charge [|electron charge e|]
 m0          =  9.10938356e-31         ! mass   [free electron mass m0]
 l0          =  1e-9                   ! length [nm]
 e0          =  1.6021766208e-19       ! energy [eV]
 n3d0        =  1e24                   ! 3-dimensional number density [10^18␣
↪[cm^-3]]
 n2d0        =  1e16                   ! 2-dimensional number density [10^12␣
↪cm^-2]
 n1d0        =  1e8                    ! 1-dimensional number density [10^6 ␣
↪cm^-1]
 eps0        =  8.854187817e-12        ! dielectric constants [vacuum-
↪permittivity] (exact) 8.854187817...e-12
 phi0        =  1.0                    ! electrostatic potential [V]
 prs0        =  1e9                    ! pressure [GPa], e.g. for elastic␣
↪constants
 t0          =  1e-9                   ! time [ns]
                                       !
 kp_k^0_zb   =  1.6021766208e-19       ! spin-orbit coupling energy,␣
↪deformation potentials [eV]
 kp_k^2_zb   =  6.104264214606464e-39  ! k.p parameter L,M,N,B  [h_bar^2/
↪(2*m0)]
                                       !
 kp_k^0_wz   =  1.6021766208e-19       ! spin-orbit coupling energies,␣
↪deformation potentials [eV]
 kp_k^2A_wz  =  1.0                    ! k.p parameter A1,A2,A3,A4,A5,A6 [],␣
↪multiplied internally with [h_bar^2/(2*m0)]
```

(continues on next page)

(continued from previous page)

```
 kp_k^2B_wz =  6.104264214606464e-39  ! k.p parameter B1,B2,B3 [h_bar^2/
↪(2*m0)] [J/m^2]
$end_input-scaling-factors           !
!-------------------------------------!
```

It is assumed that input numbers are scaled to these *units*.

Note: `[h_bar^2/(2*m0)] = kp_k^2_zb = kp_k^2B_wz`

$global-parameters

Global parameters are general parameters which are valid all over the device. See also *$global-parameters* in Keywords section.

```
$global-parameters                      required
 lattice-temperature        double      required
$end_global-parameters                  required
```

Lattice temperature

lattice-temperature

type
double

presence
required

unit
`[K]`

value
e.g. `300.0`

Note: The band gaps, i.e. the conduction band edges, are adjusted when the temperature changes. This feature can be switched off.

See *$numeric-control*: `varshni-parameters-on`

In addition, the lattice constants depend on temperature. This can be switched off as well.

See *$numeric-control*: `lattice-constants-temp-coeff-on`

Example

```
!-----------------------------------!
$global-parameters
 lattice-temperature = 300.0        ! 300 K
$end_global-parameters
!-----------------------------------!
```

$program_restrictions

These are general restrictions to the program.

```
$program_restrictions                          required
 min_dimension_domain              integer     required
 max_dimension_domain              integer     required

 minimum-grid-factor               double      required
 maximum-grid-factor               double      required

 delta-to-treat-grid-factor-as-one double      required

 minimum-grid-width                double      required
 maximum-grid-width                double      required

 maximum-number-of-grid-points     integer     required

 lowest-temperature-allowed        double      required
$end_program_restrictions                      required
```

Settings for domain dimensions.

min_dimension_domain

type
integer

presence
required

value
1

The minimum is 1, i.e. for one-dimensional simulations. A value of 0 makes sense for bulk simulations. However, this is not supported yet.

max_dimension_domain

type
integer

presence
required

value
3

The maximum is 3, i.e. for three-dimensional simulations.

Settings for grid spacing.

minimum-grid-factor

type
double

presence
required

value
0.1

Be careful, geometric series grow up very rapidly.

maximum-grid-factor

type
double

presence
required

value
`10.0`

Be careful, geometric series grow up very rapidly.

delta-to-treat-grid-factor-as-one

type
double

presence
required

value
`1.0e-3`

To avoid numerical inaccuracies, everything in the interval $1.0 \pm \Delta$ is treated as 1.0.

minimum-grid-width

type
double

presence
required

value
`0.01`

Only a guess, if exceeded a warning message is printed to standard output. Currently, this value is irrelevant and is ignored in the program, i.e. no warning message is printed.

maximum-grid-width

type
double

presence
required

value
`200.0`

Only a guess, if exceeded a warning message is printed to standard output. Currently, this value is irrelevant and is ignored in the program, i.e. no warning message is printed.

maximum-number-of-grid-points

type
integer

presence
required

value
`1000000` = 100 * 100 * 100

Only a guess, if exceeded a warning message is printed to standard output.

lowest-temperature-allowed

type
double

presence
required

value
`0.0`

Note: Convergence is more difficult to achieve if the temperature is set to a very small value or to even `0.0 [K]`.

Example

```
!-----------------------------------------------!
$program_restrictions                           !
 min_dimension_domain              = 1          !
 max_dimension_domain              = 3          !
 minimum-grid-factor               = 0.1        !
 maximum-grid-factor               = 10.0       !
 delta-to-treat-grid-factor-as-one = 1.0e-3     !
 minimum-grid-width                = 0.01       !
 maximum-grid-width                = 200.0      !
 maximum-number-of-grid-points     = 1000000    !
!lowest-temperature-allowed        = 0.1        ! [K]
 lowest-temperature-allowed        = 0.0        ! [K]
$end_program_restrictions                       !
!-----------------------------------------------!
```

$input_filename

Only to specify input file name.

```
$input_filename                          optional
 ../Syntax/database_nn3.in     character  optional
$end_input_filename                      optional
```

Any keyword with valid syntax. Must be set to optional. The first string in the third line is the name of the input file.

$domain-coordinates-defaults

Specify axis for growth direction with respect to simulation coordinate system.

```
$domain-coordinates-defaults                    required
 numbers-per-axis                integer        required
 growth-coordinate-axis          integer_array  required
 axes-possible                   integer_array  required
$end_domain-coordinates-defaults                required
```

Define the possible values for `growth-coordinate-axis` in keyword *$domain-coordinates*. They are only relevant for

```
strain-calculation = homogeneous-strain
```

or

```
strain-calculation = hydrostatic-strain
```

Right now, the values specified here are essentially not relevant because `growth-coordinate-axis` is now required in the input file if it is necessary for the strain model used.

Example

```
!-------------------------------------!
$domain-coordinates-defaults          !
 numbers-per-axis       = 3           ! only 3 numbers can be specified
 growth-coordinate-axis = 0  0  1     ! along z axis (default)
 axes-possible          = 0  0  1     ! along z axis (default)
                          0  1  0     ! along y axis
                          1  0  0     ! along x axis
$end_domain-coordinates-defaults      !
!-------------------------------------!
```

$zb-default-expectations

Number of expected parameters for zinc blende materials.

```
$zb-default-expectations                    required
 number-of-8x8kp-parameters      integer    required
 number-of-6x6kp-parameters      integer    required
 number-uniax-vb-def-pots        integer    required
$end_zb-default-expectations                required
```

Specify here how many parameters are expected.

number-of-8x8kp-parameters

type
integer

presence
required

value
6

We need 6 parameters for the 8-band $\mathbf{k} \cdot \mathbf{p}$ model in zinc blende.

number-of-6x6kp-parameters

type
integer

presence
required

value
4

We need 4 parameters for the 6-band $\mathbf{k} \cdot \mathbf{p}$ model in zinc blende.

number-of-uniax-vb-def-pots

type
integer

presence
required

value
2

We need 2 parameters for the uniaxial valence band deformation potentials in zinc blende (2 numbers usually: b, d).

Example

```
!---------------------------------------!
$zb-default-expectations                !
 number-of-8x8kp-parameters = 6         !
 number-of-6x6kp-parameters = 4         !
 number-uniax-vb-def-pots   = 2         ! b, d
$end_zb-default-expectations            !
!---------------------------------------!
```

$wz-default-expectations

Number of expected parameters for wurtzite materials.

```
$wz-default-expectations                    required
 number-of-8x8kp-parameters      integer    required
 number-of-6x6kp-parameters      integer    required
 number-uniax-vb-def-pots        integer    required
$end_wz-default-expectations                required
```

Specify here how many parameters are expected.

number-of-8x8kp-parameters

type
integer

presence
required

value
13

We need 13 parameters for the 8-band $\mathbf{k} \cdot \mathbf{p}$ model in wurtzite.

number-of-6x6kp-parameters

type
integer

presence
required

value
9

We need 9 parameters for the 6-band $\mathbf{k} \cdot \mathbf{p}$ model in wurtzite.

number-of-uniax-vb-def-pots

type
integer

presence
required

value
6

We need 6 parameters for the uniaxial valence band deformation potentials in wurtzite (6 numbers usually: d1, d2, d3, d4, d5, d6).

Example

```
!---------------------------------------!
$zb-default-expectations                !
 number-of-8x8kp-parameters = 13        !
 number-of-6x6kp-parameters = 9         !
 number-uniax-vb-def-pots   = 6         ! d1, d2, d3, d4, d5, d6
$end_zb-default-expectations            !
!---------------------------------------!
```

$zb-restrictions

Some restrictions apply for zinc blende materials.

```
$zb-restrictions                                    required
 miller-size                    integer             required
 miller-default-direction-of-x  integer_array       required
 miller-default-direction-of-y  integer_array       required
 direction-cosines              double_array        required
 miller-direction-of-cx         integer_array       required
 miller-direction-of-cy         integer_array       required
 miller-direction-of-cz         integer_array       required
 lattice-constants-for-cxyz     double_array        required
 gamma-is-cb-number             integer             required
 l-is-cb-number                 integer             required
 x-is-cb-number                 integer             required
$end_zb-restrictions                                required
```

Explanations

miller-size

type
integer

presence
required

value
3

There are three Miller indices altogether that define the (hkl) **plane**. Coincidently, in zincblende the [hkl] vector is perpendicular to the (hkl) plane.

miller-default-direction-of-x

type
integer_array

presence
required

Three-digit Miller indices of the (hkl) plane perpendicular to x axis of simulation coordinate system.

miller-default-direction-of-y

type
integer_array

presence
required

Three-digit Miller indices of the (hkl) plane perpendicular to y axis of simulation coordinate system.

```
miller-default-direction-of-x = 1 0 0  ! (100 plane) ==> [100] direction
miller-default-direction-of-y = 0 1 0  ! (010 plane) ==> [010] direction
```

This corresponds to x axis and y axis, respectively, in simulation coordinate system, i.e. the x axis is perpendicular to the (1 0 0) plane, i.e. x axis is along the [100] direction in zinc blende, and the y axis is perpendicular to the (0 1 0) plane, i.e. y axis is along the [010] direction in zinc blende.

These value can be overwritten in *$domain-coordinates* (`hkl-x-direction`, `hkl-y-direction`, `hkl-z-direction`).

Direction cosines

direction-cosines

type
double_array

presence
required

value
`0.0 0.0 0.0`

Direction cosines between lattice vectors. g1*g2, g2*g3, g1*g3 gi ... unit vectors in lattice directions.

Direction cosine refers to the cosine of the angle between any two vectors. Direction cosines are useful for forming direction cosine matrices that express one set of orthonormal basis vectors in terms of another set, or for expressing a known vector in a different basis. For zinc blende, we use:

```
        (  1  0  0 )
g_ik  = (  0  1  0 )
        (  0  0  1 )
```

Three-digit Miller indices of the (hkl) plane:

miller-direction-of-cx

type
integer_array

value
`1 0 0`

Corresponds to x axis in crystal coordinate system, i.e. the x axis is perpendicular to the (1 0 0) plane, i.e. x axis is along the [100] direction in zinc blende.

miller-direction-of-cy

type
integer_array

value
`0 1 0`

Corresponds to y axis in crystal coordinate system, i.e. the y axis is perpendicular to the (0 0 1) plane, i.e. y axis is along the [010] direction in zinc blende.

miller-direction-of-cz

type
integer_array

value
`0 0 1`

Corresponds to z axis in crystal coordinate system, i.e. the z axis is perpendicular to the (0 0 1) plane, i.e. z axis is along the [001] direction in zinc blende.

```
miller-direction-of-cx =  1 0 0  ! (100) plane ("[100] direction")
miller-direction-of-cy =  0 1 0  ! (010) plane ("[010] direction")
miller-direction-of-cz =  0 0 1  ! (000) plane ("[001] direction")
```

These are the default orientations.

lattice-constants-for-cxyz

type
double_array

value
`1.0 1.0 1.0`

Lattice constants to interpret the Miller indices.

Define how how the bands are labeled.

gamma-is-cb-number

type
integer

value
1

The conduction band minimum at the *Gamma* point has the number 1.

l-is-cb-number

type
integer

value
2

The conduction band minimum at the *L* point has the number 2.

x-is-cb-number

type
integer

value
3

The conduction band minimum at the *X* point (or *Delta* in Si or Ge) has the number 3.

Example

```
!-------------------------------------------------!
$zb-restrictions
 miller-size                  = 3
 miller-default-direction-of-x = 1 0 0
 miller-default-direction-of-y = 0 1 0
 direction-cosines            = 0.0   0.0   0.0   ! g1*g2, g2*g3, g1*g3 gi␣
↪... unit vectors in lattice directions
 miller-direction-of-cx       = 1 0 0
 miller-direction-of-cy       = 0 1 0
 miller-direction-of-cz       = 0 0 1
 lattice-constants-for-cxyz   = 1.0   1.0   1.0
 gamma-is-cb-number           = 1
 l-is-cb-number               = 2
 x-is-cb-number               = 3
$end_zb-restrictions
!-------------------------------------------------!
```

$wz-restrictions

Some restrictions apply for wurtzite materials.

```
$wz-restrictions                                    required
 miller-size                        integer         required
 miller-default-direction-of-x      integer_array   required
 miller-default-direction-of-y      integer_array   required
 direction-cosines                  double_array    required
 miller-direction-of-cx             integer_array   required
 miller-direction-of-cy             integer_array   required
 miller-direction-of-cz             integer_array   required
 lattice-constants-for-cxyz         double_array    required
$end_wz-restrictions                                required
```

Explanations

miller-size

type
integer

presence
required

value
4

There are four Miller-Bravais indices altogether that define the (hkil) **plane**.

Note: They do not define the [hkil] direction.

Usually for wurtzite, the four-digit Miller-Bravais indices (h k i l) are used.

miller-default-direction-of-x

type
integer_array

presence
required

Four-digit Miller indices of the (hkil) plane perpendicular to x axis of simulation coordinate system.

miller-default-direction-of-y

type
integer_array

presence
required

Four-digit Miller indices of the (hkil) plane perpendicular to y axis of simulation coordinate system.

```
miller-default-direction-of-x =  1 0 -1 0  ! ( 1 0 -1 0) plane
miller-default-direction-of-y = -1 2 -1 0  ! (-1 2 -1 0) plane
```

This corresponds to x axis and y axis, respectively, in simulation coordinate system, i.e. the x axis is perpendicular to the (1 0 -1 0) plane and the y axis is perpendicular to the (-1 2 -1 0) plane, respectively.

These value can be overwritten in *$domain-coordinates* (`hkil-x-direction`, `hkil-y-direction`, `hkil-z-direction`).

Note: It holds for the four-digit Miller-Bravais indices (h k i l): i = - (h + k), i.e. i is not independent.

Direction cosines

direction-cosines

type
double_array

presence
required

value
`-0.5 0.0 0.0`

Direction cosines between lattice vectors. g1*g2, g2*g3, g1*g3 gi ... unit vectors in lattice directions.

Direction cosine refers to the cosine of the angle between any two vectors. Direction cosines are useful for forming direction cosine matrices that express one set of orthonormal basis vectors in terms of another set, or for expressing a known vector in a different basis. For wurtzite, we use:

```
        (  1   -0.5  0 )
g_ik  = ( -0.5  1    0 )
        (  0    0    1 )
```

Four-digit Miller indices of the (hkil) plane:

miller-direction-of-cx

type
integer_array

value
`1 0 -1 0`

Corresponds to x axis in cartesian crystal coordinate system, i.e. the x axis is perpendicular to the (1 0 -1 0) plane.

miller-direction-of-cy

type
integer_array

value
`-1 2 -1 0`

Corresponds to y axis in cartesian crystal coordinate system, i.e. the y axis is perpendicular to the (-1 2 -1 0) plane.

miller-direction-of-cz

type
integer_array

value
`0 0 0 1`

Corresponds to z axis in cartesian crystal coordinate system, i.e. the z axis is perpendicular to the (0 0 0 1) plane, i.e. axis parallel to sixfold rotational axis in wurtzite which is coincidently also the [0001] direction.

```
miller-direction-of-cx =  1 0 -1 0  ! (10-10) plane
miller-direction-of-cy = -1 2 -1 0  ! (-12-10) plane
miller-direction-of-cz =  0 0  0 1  ! (0001) plane ("[0001] direction")
```

These are the default orientations.

lattice-constants-for-cxyz

type
double_array

value
`1.0 1.0 1.6329931618554520654648560498039`

Lattice constants to interpret the Miller-Bravais indices: 1.0, 1.0, $\sqrt{8/3}$. Here, we take the ideal wurtzite ratio of $c/a = \sqrt{8/3}$.

In wurtzite, there are three coordinate axis in the basal plane, $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, and the $\mathbf{c}$ direction perpendicular to it. There are different definitions for it.

- $\mathbf{a}_1$ = [10-10], $\mathbf{a}_2$ = [-12-10], $\mathbf{a}_3$ = [...], $\mathbf{c}$ = [0001] (used by *nextnano*[3]) ==> $\mathbf{a}_1 = \sqrt{3}/2a\mathbf{x} - a/2\mathbf{y}$, $\mathbf{a}_2 = a\mathbf{y}$, $\mathbf{c} = c\mathbf{z}$
- $\mathbf{a}_1$ = [2-1-10], $\mathbf{a}_2$ = [-12-10], $\mathbf{a}_3$ = [-1-120], $\mathbf{c}$ = [0001] ==> $\mathbf{a}_1$ = [2-1-10] $a/\sqrt{6}$, $\mathbf{a}_2$ = [-12-10] $a/\sqrt{6}$, $\mathbf{c}$ = [0001] c = [0, 0, 0, 3 λ] $a/\sqrt{6}$, where $\lambda = \sqrt{2/3}c/a$.

Example

```
!-----------------------------------------------------!
$wz-restrictions
 miller-size                    =  4
 miller-default-direction-of-x  =  1  0 -1  0
 miller-default-direction-of-y  = -1  2 -1  0
 direction-cosines              = -0.5   0.0   0.0  ! g1*g2, g2*g3, g1*g3 gi␣
↪... unit vectors in lattice directions
 miller-direction-of-cx         =  1  0 -1  0
 miller-direction-of-cy         = -1  2 -1  0
 miller-direction-of-cz         =  0  0  0  1
 lattice-constants-for-cxyz     =  1.0    1.0    1.
↪6329931618554520654648560498039
$end_wz-restrictions
!-----------------------------------------------------!
```

$region-default

Default priority in regions.

```
$region-default                          required
 minimum-priority          integer       required
$end_region-default                      required
```

Minimum priority.

minimum-priority

type
integer

presence
required

value
`0`

Change only with care. `0` is a good idea. A negative value might also work.

Example

```
!----------------------------!
$region-default               !
 minimum-priority = 0         !
```

(continues on next page)

(continued from previous page)

```
$end_region-default          !
!----------------------------!
```

$known-doping-functions

Several doping functions are possible. For more information see *$doping-function*.

```
$known-doping-functions                          required
 doping-function-number             integer     required
 doping-function-name               character   required
$end_known-doping-functions                      required
```

Example

```
!-----------------------------------!
$known-doping-functions

 doping-function-number = 1
 doping-function-name   = constant   ! constant doping profile

 doping-function-number = 2
 doping-function-name   = gauss-1d   ! Gaussian shaped doping profile

 doping-function-number = 3
 doping-function-name   = step-1d

 doping-function-number = 4
 doping-function-name   = well-1d

 doping-function-number = 5
 doping-function-name   = linear     ! linear doping profile

$end_known-doping-functions
!-----------------------------------!
```

$known-function-names

Several functions are possible. For more information see *$alloy-function*.

```
$known-function-names                    required
 function-number             integer     required
 function-name               character   required
$end_known-function-names                required
```

Alloy functions.

function-number

type
integer

presence
required

value
1, 2, ...

Dense numbering of integers, starting from 1, 2, 3, ...

function-name

type
character

presence
required

value
`constant`, `linear`, `import-alloy-profile`, `alloy-profile-defined-by-function`, ... (see below)

Name of alloy profile.

Example

```
!-----------------------------------------------------!
 $known-function-names
  function-number = 1
  function-name   = gaussian-3d

  function-number = 2
  function-name   = gaussian-2d

  function-number = 3
  function-name   = gaussian-1d

  function-number = 4
  function-name   = one-m-gaussian-3d

  function-number = 5
  function-name   = one-m-gaussian-2d

  function-number = 6
  function-name   = one-m-gaussian-1d

  function-number = 7
  function-name   = constant

  function-number = 8
  function-name   = linear

  function-number = 9
  function-name   = bilinear

  function-number = 10
  function-name   = xy-gaussian-3d

  function-number = 11
  function-name   = xy-gaussian-2d

  function-number = 12
  function-name   = xy-gaussian-1d

  function-number = 13
  function-name   = xy-one-m-gaussian-3d

  function-number = 14
  function-name   = xy-one-m-gaussian-2d
```

(continues on next page)

(continued from previous page)

```
  function-number = 15
  function-name   = xy-one-m-gaussian-1d

  function-number = 16
  function-name   = xy-constant

  function-number = 17
  function-name   = xy-linear

  function-number = 18
  function-name   = xy-bilinear

  function-number = 19
  function-name   = inv-triangle

  function-number = 20
  function-name   = trumpet

  function-number = 21
  function-name   = parabolic

  function-number = 22
  function-name   = biparabolic

  function-number = 23
  function-name   = triparabolic

  function-number = 24
  function-name   = Fermi-function

  function-number = 25
  function-name   = onion

  function-number = 26
  function-name   = import-alloy-profile

  function-number = 27
  function-name   = alloy-profile-defined-by-function

$end_known-function-names
!-------------------------------------------------------!
```

$known-impurity-types

Several impurity types are possible.

```
$known-impurity-types                 required
 impurity-type-name         character  required
 impurity-type-number       integer    required
$end_known-impurity-types              required
```

Example

```
!---------------------------------------!
$known-impurity-types                                 ␣
```

(continues on next page)

(continued from previous page)

```
↪    !

 impurity-type-name   = n-type
 impurity-type-number = 1

 impurity-type-name   = p-type
 impurity-type-number = 2

 impurity-type-name   = trap
 impurity-type-number = 3

$end_known-impurity-types                                                    ␣
↪    !
!---------------------------------------!
```

$interface-state-limitations

Several doping functions are possible. For more information see *$doping-function*.

```
$interface-state-limitations                  required
 known-state-type-name             character  required
$end_interface-state-limitations              required
```

Example

```
!---------------------------------------!
$interface-state-limitations            !
 known-state-type-name = trap           ! charge density due to trap states
 known-state-type-name = fixed-charge   ! fixed charge density
 known-state-type-name = electrolyte    ! charge density due to electrolyte␣
↪(site-binding model)
 known-state-type-name = gas            ! charge density due to gas␣
↪adsorption
 known-state-type-name = k.p            ! interface k.p Hamiltonian
$end_interface-state-limitations        !
!---------------------------------------!
```

$quantum-model-electrons

Several quantum models for the electrons are possible.

```
$quantum-models-electrons                 required
 model-name                 character     required
 model-type-number          integer       required
$end_quantum-models-electrons              required
```

Example

```
!-----------------------------------------------!
$quantum-models-electrons
 model-name        = effective-mass
 model-type-number = 1

 model-name        = 8x8kp
```

(continues on next page)

(continued from previous page)

```
 model-type-number = 2
$end_quantum-models-electrons
!-----------------------------------------------!
```

$quantum-model-holes

Several quantum models for the holes are possible.

```
$quantum-models-holes                    required
 model-name                  character   required
 model-type-number           integer     required
$end_quantum-models-holes                required
```

Example

```
!-----------------------------------------------!
$quantum-models-holes
 model-name        = effective-mass
 model-type-number = 1

 model-name        = 8x8kp
 model-type-number = 2

 model-name        = 6x6kp
 model-type-number = 3
$end_quantum-models-holes
!-----------------------------------------------!
```

$separation-models-electrons

More details about the separation model can be found in the glossary.

```
$separation-models-electrons                    required
 model-name                         character   required
 model-type-number                  integer     required
$end_separation-models-electrons                required
```

Example

```
!-------------------------------------!
$separation-models-electrons
 model-name        = eigenvalue
 model-type-number = 1

 model-name        = energy
 model-type-number = 2

 model-name        = edge-model
 model-type-number = 3
$end_separation-models-electrons
!-------------------------------------!
```

$separation-models-holes

More details about the separation model can be found in the glossary.

```
$separation-models-holes                          required
 model-name                          character    required
 model-type-number                   integer      required
$end_separation-models-holes                      required
```

Example

```
!-------------------------------------!
$separation-models-holes
 model-name        = eigenvalue
 model-type-number = 1

 model-name        = energy
 model-type-number = 2

 model-name        = edge-model
 model-type-number = 3
$end_separation-models-holes
!-------------------------------------!
```

$method-of-brillouin-zone-integration

More details about the method of Brillouin zone integration can be found in the glossary.

```
$method-of-brillouin-zone-integration                 required
 model-name                                character    required
 model-type-number                         integer      required
$end_method-of-brillouin-zone-integration             required
```

Example

```
!---------------------------------------!
$method-of-brillouin-zone-integration
 model-name        = special-axis
 model-type-number = 1

 model-name        = simple-integration
 model-type-number = 2

 model-name        = gen-dos
 model-type-number = 3
$end_method-of-brillouin-zone-integration
!---------------------------------------!
```

$k-range-determination-methods

This makes only sense for $\mathbf{k} \cdot \mathbf{p}$ calculations in 1D ($k_\parallel = (k_x, k_y)$) and 2D ($k_\parallel = (k_z)$) but not in 3D.

1. Solve Schrödinger equation for $(k_x, k_y) = (0, 0)$.
2. Define a set of $k_\parallel$ that one needs and solve $\mathbf{k} \cdot \mathbf{p}$ Schrödinger equation for every $k_\parallel$.

```
$k-range-determination-methods                    required
 model-name                          character    required
 model-type-number                   integer      required
$end_k-range-determination-methods
```

Two models are supported.

model-name

value
bulk-dispersion-analysis

model-type-number

value
1

Here, the range for $k_\parallel$ is determined automatically by the program using the bulk energy dispersion $E(k)$. More information…

model-name

value
k-max-input

model-type-number

value
2

A maximum value k_{max} of $k_\parallel$ has to be specified in the input file.

Example

```
!-------------------------------------------!
$k-range-determination-methods
 model-name        = bulk-dispersion-analysis
 model-type-number = 1

 model-name        = k-max-input
 model-type-number = 2
$end_k-range-determination-methods
!-------------------------------------------!
```

$tight-binding-parameters

This part is preliminary. It is not working yet.

```
$tight-binding-parameters                         optional !

 material-name                       character    required !

 ! basis
 number-of-ions                      integer      required !
 name-ion-1                          character    required !
```

(continues on next page)

(continued from previous page)

```
name-ion-2                              character     optional !

! states
spin-orbit-ion-1-p                      double        required ! ion 1 spin␣
↪orbit (p)
spin-orbit-ion-1-d                      double        required ! ion 1 spin␣
↪orbit (d)
spin-orbit-ion-2-p                      double        optional ! ion 2 spin␣
↪orbit (p)
spin-orbit-ion-2-d                      double        optional ! ion 2 spin␣
↪orbit (d)

! interaction between s and p orbitals
! interaction between s and d orbitals (in our case: zero)

number-of-orbitals-per-ion-1            integer       required !
number-of-orbitals-per-ion-2            integer       optional !
onsite-ion-1-s                          double        required ! ion 1␣
↪onsite (s) [eV]
onsite-ion-1-p                          double        required ! ion 1␣
↪onsite (p) [eV]
onsite-ion-1-s*                         double        required ! ion 1␣
↪onsite (s*) [eV]
onsite-ion-1-d                          double        required ! ion 1␣
↪onsite (d) [eV]
onsite-ion-2-s                          double        optional ! ion 2␣
↪onsite (s) [eV]
onsite-ion-2-p                          double        optional ! ion 2␣
↪onsite (p) [eV]
onsite-ion-2-s*                         double        optional ! ion 2␣
↪onsite (s*) [eV]
onsite-ion-2-d                          double        optional ! ion 2␣
↪onsite (d) [eV]
offset-ion-1                            double        required ! ion 1 [eV]
offset-ion-2                            double        optional ! ion 2 [eV]

! Tight-binding couplings
number-of-pairs                         integer       required !
pair-1-ion-1                            character     required !
pair-1-ion-2                            character     optional !
pair-1-number-of-parameters             integer       required !
pair-1-nearest-neighbors                integer       required !
ion-ion-coupling-1                      character     required !
ion-ion-coupling-1-sss                  double_array  required ! [eV],[]
ion-ion-coupling-1-sps                  double_array  required ! [eV],[]
ion-ion-coupling-1-ss*s                 double_array  required ! [eV],[]
ion-ion-coupling-1-sds                  double_array  required ! [eV],[]
ion-ion-coupling-1-pss                  double_array  required ! [eV],[]
ion-ion-coupling-1-pps                  double_array  required ! [eV],[]
ion-ion-coupling-1-ps*s                 double_array  required ! [eV],[]
ion-ion-coupling-1-pds                  double_array  required ! [eV],[]
ion-ion-coupling-1-s*ss                 double_array  required ! [eV],[]
ion-ion-coupling-1-s*ps                 double_array  required ! [eV],[]
ion-ion-coupling-1-s*s*s                double_array  required ! [eV],[]
ion-ion-coupling-1-s*ds                 double_array  required ! [eV],[]
ion-ion-coupling-1-dss                  double_array  required ! [eV],[]
```

(continues on next page)

(continued from previous page)

```
 ion-ion-coupling-1-dps                double_array  required ! [eV],[]
 ion-ion-coupling-1-ds*s               double_array  required ! [eV],[]
 ion-ion-coupling-1-dds                double_array  required ! [eV],[]
 ion-ion-coupling-1-ppp                double_array  required ! [eV],[]
 ion-ion-coupling-1-pdp                double_array  required ! [eV],[]
 ion-ion-coupling-1-dpp                double_array  required ! [eV],[]
 ion-ion-coupling-1-ddp                double_array  required ! [eV],[]
 ion-ion-coupling-1-ddd                double_array  required ! [eV],[]

 ion-ion-pair-1-reference-distance-nn  double        required ! [nm]

$end_tight-binding-parameters                       optional !
```

Example

```
!-----------------------------------------------------------------------!
$tight-binding-parameters

 !-----------------------------------------------
 ! Si tight-binding parameters (c) Jean Marc Jancu
 !-----------------------------------------------

 material-name                      = Si

 ! basis
 number-of-ions                     = 1         !
 name-ion-1                         = Si

 ! states
 spin-orbit-ion-1-p                 = 0.0195    ! Si spin orbit (p)
 spin-orbit-ion-1-d                 = 0.0000    ! Si spin orbit (d)

 ! interaction between s and p orbtals
 ! interaction between s and d orbitals (in our case: zero)

 number-of-orbitals-per-ion-1       = 4          ! Si
 onsite-ion-1-s                     = -2.0196    ! Si onsite (s)
 onsite-ion-1-p                     =  4.5448    ! Si onsite (p)
 onsite-ion-1-s*                    = 19.6748    ! Si onsite (s*)
 onsite-ion-1-d                     = 14.1836    ! Si onsite (d)
 offset-ion-1                       = 0.0        ! Si  eV

! Tight-binding couplings
 number-of-pairs                    = 1
 pair-1-ion-1                       = Si
 pair-1-ion-2                       = Si
 pair-1-number-of-parameters        = 21
 pair-1-nearest-neighbors           = 1
 ion-ion-coupling-1                 = 'sss sps ss*s sds pss pps ps*s pds␣
↪s*ss s*ps s*s*s s*ds dss dps ds*s dds ppp pdp dpp ddp ddd'
 ion-ion-coupling-1-sss             = -1.9413      3.672d0
 ion-ion-coupling-1-sps             =  2.7836      2.488d0
 ion-ion-coupling-1-ss*s            = -1.6933      0.000d0
 ion-ion-coupling-1-sds             = -2.7998      1.869d0
 ion-ion-coupling-1-pss             =  2.7836      2.488d0
 ion-ion-coupling-1-pps             =  4.1068      2.187d0
```

(continues on next page)

(continued from previous page)

```
 ion-ion-coupling-1-ps*s                =  2.8428      1.919d0
 ion-ion-coupling-1-pds                 = -2.1073      1.830d0
 ion-ion-coupling-1-s*ss                = -1.6933      0.000d0
 ion-ion-coupling-1-s*ps                =  2.8428      1.919d0
 ion-ion-coupling-1-s*s*s               = -3.3081      0.000d0
 ion-ion-coupling-1-s*ds                = -0.7003      2.000d0
 ion-ion-coupling-1-dss                 = -2.7998      1.869d0
 ion-ion-coupling-1-dps                 = -2.1073      1.830d0
 ion-ion-coupling-1-ds*s                = -0.7003      2.000d0
 ion-ion-coupling-1-dds                 = -1.2327      2.000d0
 ion-ion-coupling-1-ppp                 = -1.5934      3.711d0
 ion-ion-coupling-1-pdp                 =  1.9977      2.093d0
 ion-ion-coupling-1-dpp                 =  1.9977      2.093d0
 ion-ion-coupling-1-ddp                 =  2.5145      2.000d0
 ion-ion-coupling-1-ddd                 = -2.4734      2.000d0

 ion-ion-pair-1-reference-distance-nn = 2.351259e-10 ! m -> A ! don't␣
↪change! very sensitive

$end_tight-binding-parameters
!-----------------------------------------------------------------------!
```

$warnings

Warnings can be switched on or off.

```
$warnings                              required
 warnings              logical         required
$end_warnings                          required
```

Specify here if warnings should be written to the `.log` file.

warnings

type
logical

presence
required

options
`.FALSE.`

`.TRUE.`

default
uses value specified in `database_nn3.in`

Example

```
!----------------------------!
$warnings                    !
 warnings = .FALSE.          ! switch off warnings
!warnings = .TRUE.           ! switch on warnings
$end_warnings                !
!----------------------------!
```

7.4 Tutorials

Many of the *nextnano++* tutorials have equivalent input files for *nextnano³*. However, some features are only available in *nextnano³*, e.g. tight-binding, electrolytes, graphene and mobility calculations.

7.4.1 Heterostructures

Quantum Confined Stark Effect (QCSE)

Author: Stefan Birner

The input file used is:

- *1DQuantumConfinedStarkEffect_nn3.in*
- *1DQuantumConfinedStarkEffect_nnp.in*

This tutorial aims to reproduce Figure 3.22 (p. 96) of Paul Harrison's excellent book "*Quantum Wells, Wires and Dots*" (Section 3.12 "Quantum Confined Stark Effect"), 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 are able to look up the relevant material parameters and to check the results for consistency.

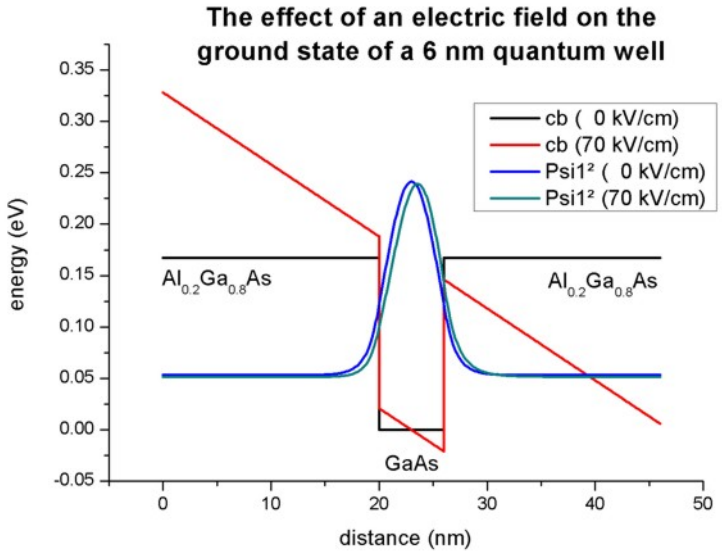
Single quantum well: 20 nm AlGaAs / 6 nm GaAs / 20 nm AlGaAs

Our structure consists of a 6 nm GaAs quantum well that is surrounded by 20 nm $Al_{0.2}Ga_{0.8}As$ barriers on each side. We thus have the following sequence: **20 nm** $Al_{0.2}Ga_{0.8}As$ / 6 nm GaAs / **20 nm** $Al_{0.2}Ga_{0.8}As$. The barriers are printed in bold.

The figure shows the **conduction band edge** and the **square of the ground state electron wave function** (ψ^2) that is confined inside the well for two cases:

- No applied electric field (0 kV/cm)
- Applied electric field (-70 kV/cm)

In the case of an applied electric field, the wave function moves to the right and its ground state energy decreases slightly. The reason is that a charged particle prefers to move to areas of lower potential in order to lower the total energy. (Note that the energies were shifted so that the conduction band edge of GaAs equals 0 eV.) The origin of the electric field is chosen automatically to be the center of the well. This makes it possible to compare energies by varying the applied electric field as shown in this tutorial. For holes, the wave function would move to the left (not shown here) thus making it possible to produce a space charge or a polarization of the charge carriers.



Technical Details

We use

```
$simulation-flow-control
 flow-scheme = 21          ! apply constant electric field
 ...
```

and

```
$numeric-control
 ...
 zero-potential = yes
```

This flow scheme includes the following:

1. We calculate the strain (if any)
2. We calculate the piezo and pyroelectric charges (if any)
3. **We do not solve Poisson's equation** (This is the difference with `flow-scheme = 20`)
4. We apply the electric field
5. We calculate the eigenstates and wave functions by solving Schrödinger's equation (either single-band or $\mathbf{k} \cdot \mathbf{p}$)

Note that in this case, this is not a self-consistent calculation of the Poisson-Schrödinger equation.

Output

a. The **conduction** band edge of the Gamma conduction band can be found at `band_structure / cb1D_Gamma.dat`

b. This file contains the **eigenenergies** of the ground state. The units are in [eV] `Schroedinger_1band / ev_cb1_qc1_sg1_deg1.dat`

For example, the values for zero applied electric field read:

	num_ev	eigenvalue [eV]
nextnano GmbH	1	0.053287
Paul Harrison's book	1	0.05326045 (or 0.053310)

c. This file contains the **eigenenergies** and the **squared wave functions** (ψ^2): `Schroedinger_1band / cb1_sg1_deg1_psi_squared.dat`

d. This file contains the **eigenenergies** and the **wave functions** (ψ): `Schroedinger_1band / cb1_sg1_deg1_psi.dat`

a) and b) can be used to plot the data as shown in the figure above.

Varying the electric field

There are two ways to vary the applied electric field from 0 kV/cm to -70 kV/cm.

Possibility 1

We perform several individual calculations and vary the strength of the electric field by specifying its value for each individual calculation.

```
$electric-field
   electric-field-on        = yes    ! 'yes' / 'no'
   electric-field-strength  =  0.0   ! in units of [V/m] - Here:   0 kV/cm
 ! electric-field-strength  = -5e5   ! in units of [V/m] - Here:  -5 kV/cm
 ! electric-field-strength  = -10e5  ! in units of [V/m] - Here: -10 kV/cm
 ! electric-field-strength  = -15e5  ! in units of [V/m] - Here: -15 kV/cm
 ! electric-field-strength  = -20e5  ! in units of [V/m] - Here: -20 kV/cm
 ! electric-field-strength  = -25e5  ! in units of [V/m] - Here: -25 kV/cm
 ! electric-field-strength  = -30e5  ! in units of [V/m] - Here: -30 kV/cm
 ! electric-field-strength  = -40e5  ! in units of [V/m] - Here: -40 kV/cm
 ! electric-field-strength  = -50e5  ! in units of [V/m] - Here: -50 kV/cm
 ! electric-field-strength  = -60e5  ! in units of [V/m] - Here: -60 kV/cm
 ! electric-field-strength  = -70e5  ! in units of [V/m] - Here: -70 kV/cm
   electric-field-direction = 0 0 1  ! [001] direction, i.e. along z axis
$end_electric-field
```

Possiblity 2

An **alternative** (and much more user friendly approach) would be the usage of an "electric field sweep". Here, only one calculation is necessary. The variation of the electric field strength is done automatically.

```
$electric-field
 electric-field-on                   = yes   ! 'yes' / 'no'
 electric-field-strength             = 0.0   ! in units of [V/m] - Here:  0 kV/cm
 electric-field-direction            = 0 0 1 ! [001] direction, i.e. along z axis.
                                             !
 electric-field-sweep-active         = yes   ! 'yes' / 'no'
 electric-field-sweep-step-size      = -5e5  ! in units of [V/m] - Here:  -5 * 10^5␣
↪V/m =  -5 kV/cm
 electric-field-sweep-number-of-steps = 14   ! 14 steps, starting from 0 kV/cm to -
↪70 kV/cm
$end_electric-field
```

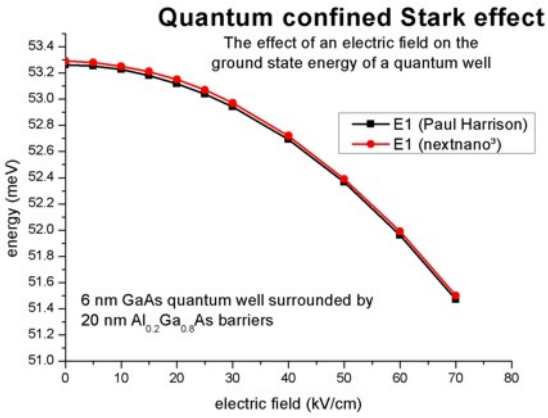
Here, the electric field is varied from 0 kV/cm to -70 kV/cm, in steps of -5 kV/cm. The output of the eigenvalues is then contained in `Schroedinger_1band / electric_ev_cb1_sg1_deg1.dat`.

The first column contains the strength of the electric field in units of [kV/cm]. The second column contains the 1st eigenvalue for the specified electric field in units of [eV]. The third column contains the 2nd eigenvalue for the specified electric field in units of [eV].

The following figure shows the ground state energy of the 6 nm quantum well as a function of the applied electric field strength F. The calculated energies can be represented by a parabolic fit. Over the range of electric fields investigated, the ground state energy can be represented by the parabola:

$$E_1(F) = E_1(0) - 0.000365F^2$$

where $E_1(0)$ refers to the ground state energy at zero electric field (in units of meV). Here, the electric field strength F is given in units of kV/cm. (Note: Paul Harrison's value is 0.00036)



This suppression of the confined energy level by an electric field is called "Quantum Confined Stark Effect (QCSE)". The data that were plotted here are contained in this file. `Schroedinger_1bandelectric[kV/cm]_ev1D_cb1_qc1_sg1_deg1_dir.dat`. The energy levels are contained as a function of the sweep variable "electric field".

The plot is (almost) in agreement with Fig. 3.22 (p. 96) of Paul Harrison's book "*Quantum Wells, Wires and Dots*". The energies differ slightly although we used

- identical effective masses
- identical conduction band offset
- identical grid resolution (0.1 nm)

The only difference is that we use multiple points at the heterointerfaces but this should not explain the difference. The answer lies probably in a tiny inconsistency in the book. On page 96 it says: "where $E_1(0)$ refers to the ground-state energy (53.310 meV) at zero field" However, the value that was used in Fig. 3.22 is $E_1(0)$ = 53.26045 meV. (This value can be found on the CD that accompanies the book.) The next**nano**^{3} value is 53.287 meV which lies in the middle between these two values.

Output

The energy values were taken from the file `Schroedinger_1band / ev_cb1_sg1_deg1.dat`.

For example, the values for zero applied electric field read:

	num_ev	eigenvalue [eV]
nextnano++	1	0.053287
Paul Harrison's book	1	0.05326045 (or 0.053310)

The values for an applied electric field of -70 kV/cm read:

	num_ev	eigenvalue [eV]
nextnano++	1	0.051497
Paul Harrison's book	1	0.051472

Optical interband absorption in a quantum well including excitonic effects

This tutorial presents calculation of interband absorption spectrum in a quantum well including excitonic effects.

There is a separate tutorial that discusses the calculation of the exciton binding energy and exciton Bohr radius of an infinite quantum well: *Exciton energy in quantum wells - Tutorial*

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 absorption spectrum has been calculated using a simple model assuming a parabolic energy dispersion. In order to keep things simple, i.e. to be able to compare our results with analytical formula, we used the same effective mass for electrons and holes ($m_\mathrm{e} = m_\mathrm{h} = 0.065\ m_0$).

The excitonic binding energy E_b has been calculated to be -9.5 meV. Therefore, the absorption spectrum that includes excitonic contributions starts at an energy roughly 10 meV below than band gap. The exciton Bohr radius λ was found to be 13.1 nm.

For Lorentzian broadening we use a linewidth of FWHM = 6 meV, and for Gaussian broadening we use FWHM = 10 meV. The FWHM(Voigt) depends in a complicated way on FWHM(Lorentzian) and FWHM(Gaussian).

Property	Symbol	unit	analytical calculation	nextnano GmbH
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
effective hole mass	m_h	m_0	0.0665	0.0665
refractive index	n_r		3.3	3.3
linewidth (FWHM) Lorentzian	Γ_L	meV	n/a	6
linewidth (FWHM) Gaussian	Γ_G	meV	n/a	10
temperature	T	K	300	300

The Coulomb enhancement factor is given by $S_\mathrm{2D} = \frac{\exp\left(\pi/\sqrt{\Delta}\right)}{\cosh\left(\pi/\sqrt{\Delta}\right)}$, where Δ is the total excess kinetic energy of the electron–hole pair normalized to $E_\mathrm{b}/4$ *[LeverJLT2010]*.

We observe two major contributions to the absorption spectrum:

- A distinct peak a few meV (corresponding to the exciton binding energy E_b) lower than the absorption edge (band gap). This is the signature of the bound exciton.
- Sommerfeld enhancement: In the continuum part of the absorption spectrum, it is scaled via the Coulomb enhancement factor S_2D.

— Begin —

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

The following documentation and figures were generated automatically using *nextnanopy*.

The following Python script was used: `1D_InterbandAbsorption_InfiniteWell_Exciton_nextnano3.py`

The following figures have been generated using *nextnano³*.

The absorption spectrum has been calculated using a simple model assuming a parabolic energy dispersion.

Infinite QW (single-band)

Optical absorption spectrum of bulk crystal and of a quantum well

Optical absorption of a 10 nm quantum well

Optical absorption of a 10 nm quantum well using different broadening functions

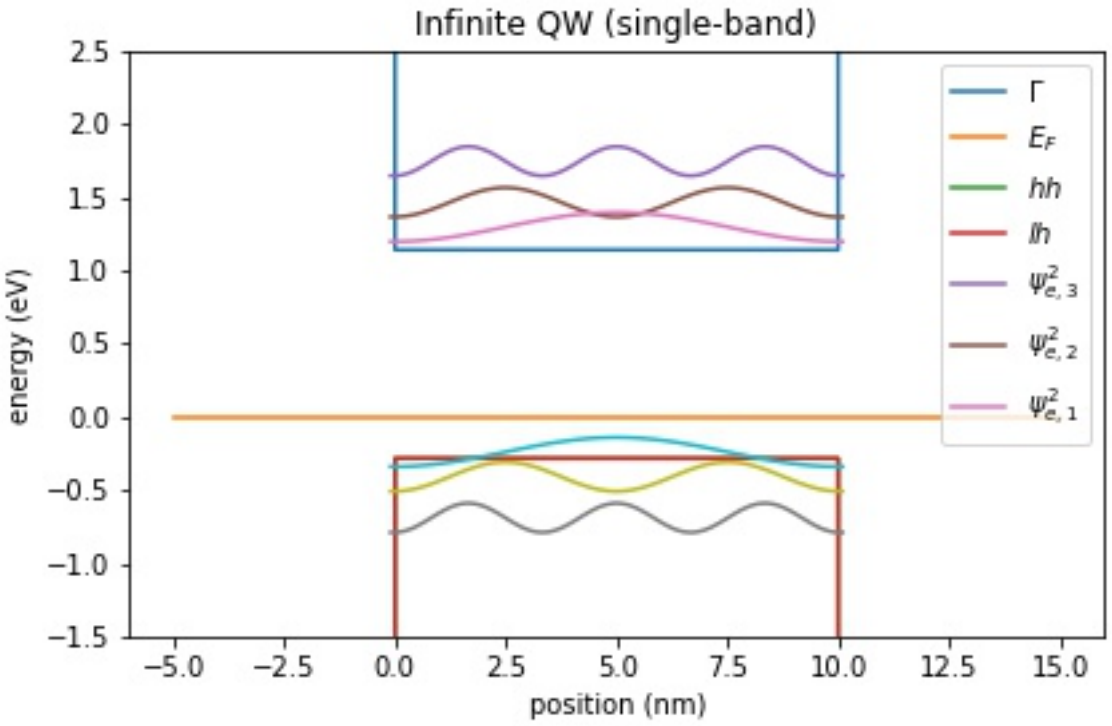


Figure 7.4.1.1: Conduction and valence band edges, Fermi level, electron and holes states of a quantum well

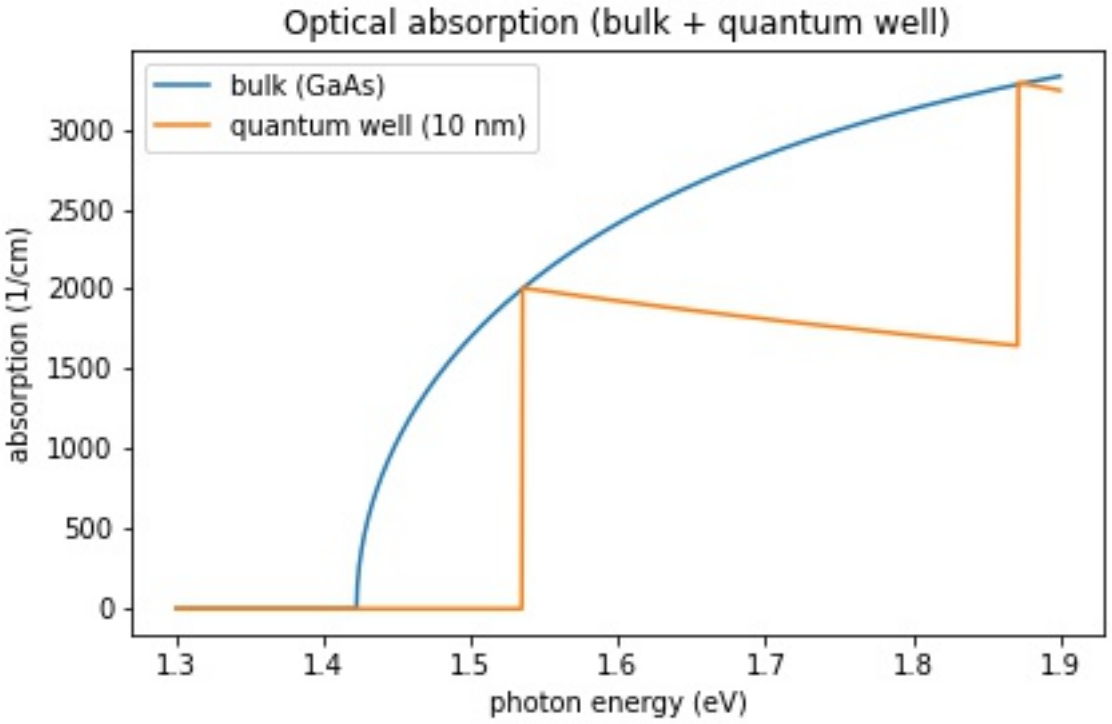


Figure 7.4.1.2: The absorption spectrum of bulk GaAs and a 10 nm infinite quantum well.

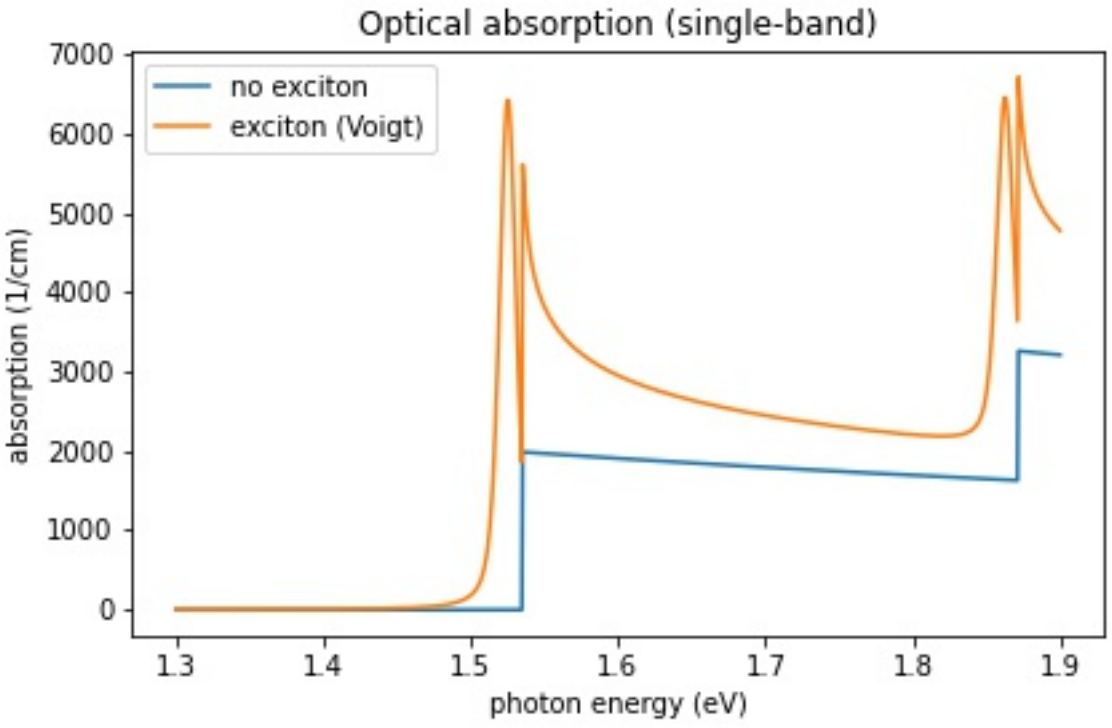


Figure 7.4.1.3: The absorption spectrum of a 10 nm infinite quantum well consisting of GaAs. The absorption spectrum has been calculated with and without excitonic contributions to the spectrum.

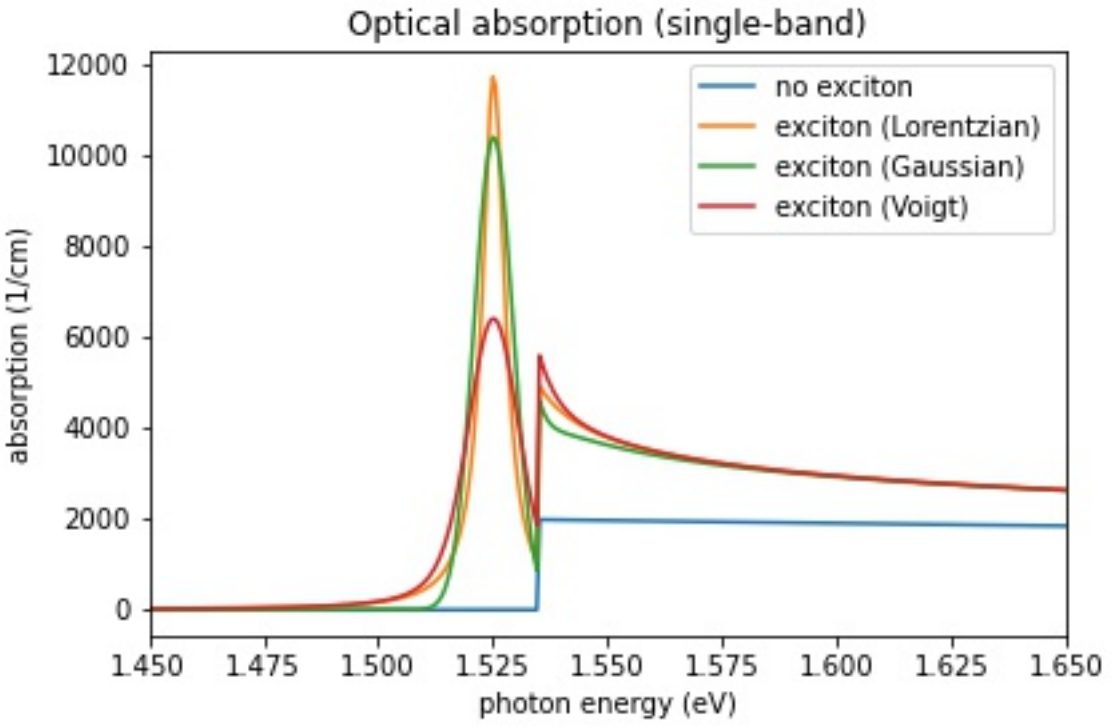


Figure 7.4.1.4: The absorption spectrum of a 10 nm infinite quantum well consisting of GaAs. The absorption spectrum has been calculated with Lorentzian, Gaussian and Voigt broadening function, and without excitonic contributions to the spectrum.

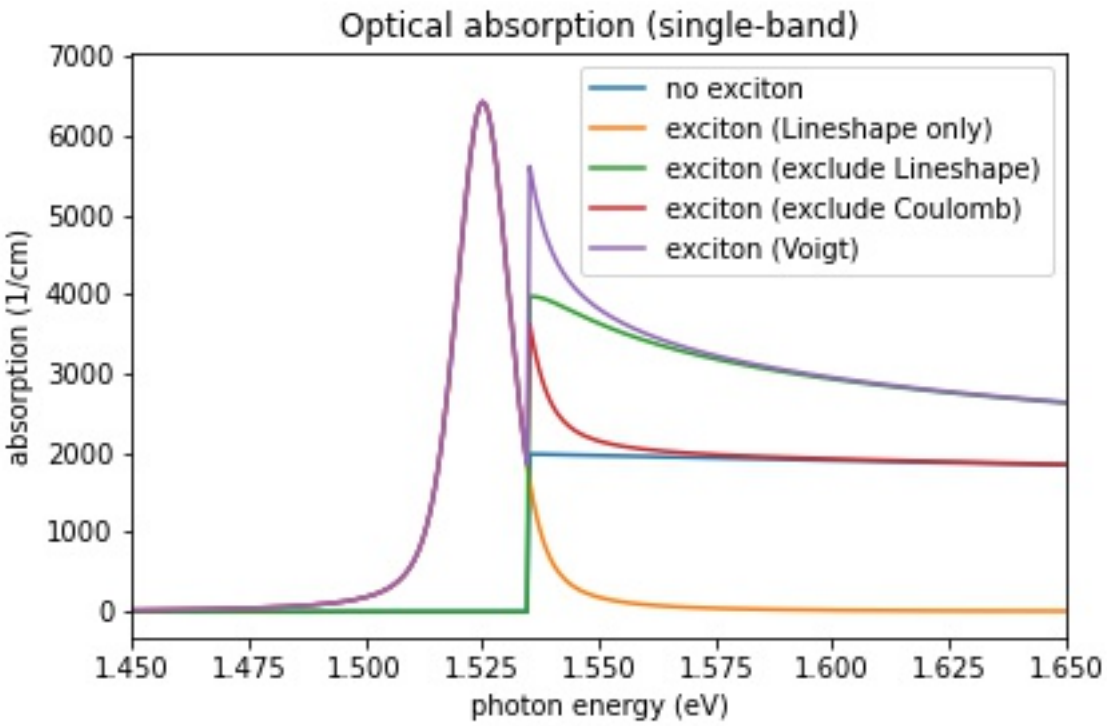


Figure 7.4.1.5: The absorption spectrum of a 10 nm infinite quantum well consisting of GaAs showing the different excitonic contributions. The absorption spectrum has been calculated with a Voigt broadening functions, and without excitonic contributions to the spectrum. In order to see the contributions of the Coulomb enhancement factor and the lineshape peak, each of these contributions can be switched off.

Optical absorption of a 10 nm quantum well showing the different contributions to the excitonic absorption

We acknowledge funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101017194 (SiPho-G).

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

— End —

Schottky barrier

Input Files:

- *1DSchottky_barrier_GaAs_nn3.in*
- *2DSchottky_barrier_GaAs_nn3.in*
- *1DSchottky_barrier_GaAs_ohmic_nn3.in*
- *1DSchottky_barrier_GaAs_SchottkyBarrier0V_nn3.in*
- *1DSchottky_barrier_GaAs_surface_density_nn3.in*
- *1DSchottky_barrier_GaAs_surface_states_acceptor_nn3.in*

Scope:

In this tutorial we simulate Schottky contacts.

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

Consequence: The Schottky barrier sets 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.

The Schottky barrier model implemented in *nextnano++* is basically a Fermi level pinning and does not take into account the work function of the metal: The barrier height is independent of the metal work function and is entirely determined by the surface states and the doping.

```
contacts{
    schottky{                   # Schottky barrier (Fermi level pinning)
        name = contact
        bias = 0.0              # [V] apply voltage
        barrier = 0.53          # [V] GaAs, S.M. Sze, "Physics of Semiconductor␣
↪Devices", p. 275 (2nd ed.)
    }
}
```

The n-type donor concentration in $GaAs$ has been taken to be $1 \cdot 10^{18}$ cm^{-3} (fully ionized). The temperature is set to 300 K.

Figure 7.4.1.6 shows the conduction band edge profile for n-type $GaAs$ in equilibrium with

- a Schottky barrier of 0.53 V, i.e. the conduction band edge is pinned 0.53 eV above the Fermi level (which is at 0 eV)
- a Schottky barrier of 0 V
- an ohmic contact at 10 nm.

(The contact region is from 0 nm to 10 nm but no equations are solved inside the contact region.)

Note that in equilibrium the Femi 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 part of the figure). 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!

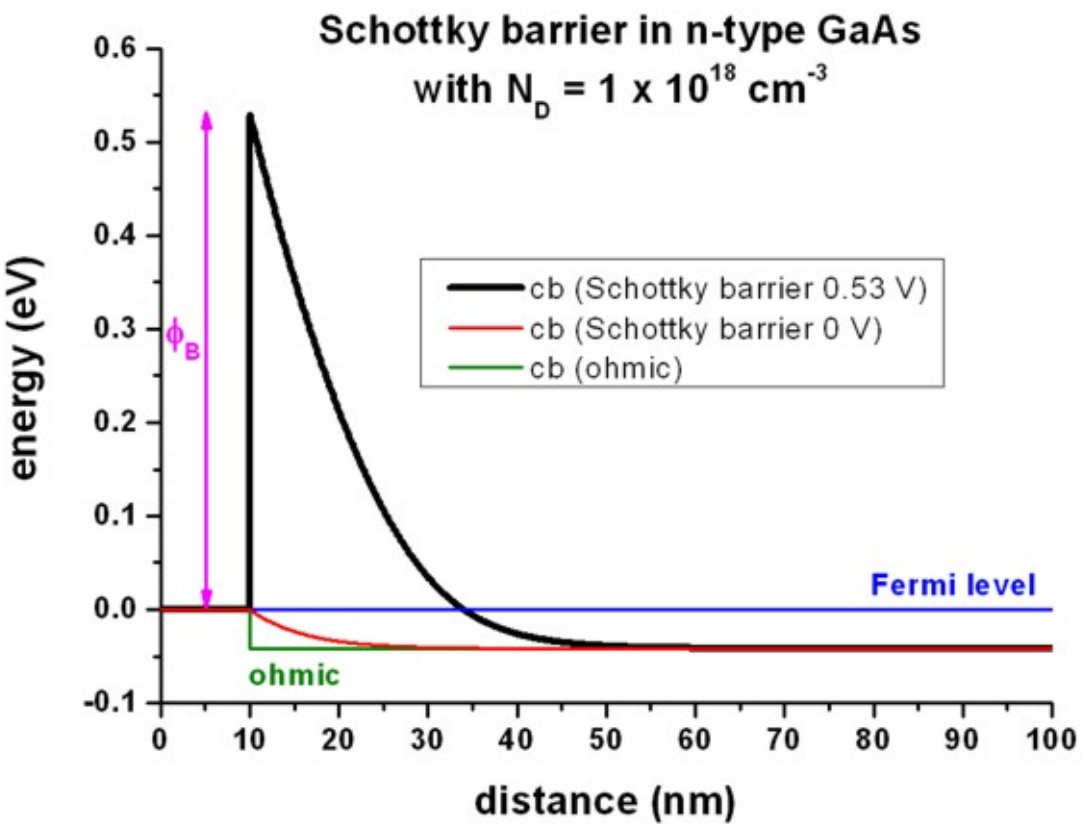


Figure 7.4.1.6: Calculated conduction band profile

An ohmic contact corresponds to a Neumann boundary condition for the Poisson equation, i.e. $d\phi/dx = 0$ (constant electrostatic potential) or flat band condition which is equivalent to $E = 0$. A Schottky barrier ϕ_B is a Dirichlet boundary condition for the Poisson equation, i.e. the value of the conduction band edge at the boundary is fixed with respect to the Fermi level:

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

In this particular example, an artificial Schottky barrier of -0.04184 V would be equivalent to an ohmic contact, (i.e. flat band condition), but only for the same temperature and the same doping concentration.

Interface charges (surface states)

Input file: *1DSchottky_barrier_GaAs_surface_density.in*

Instead of specifying a Schottky barrier, the user can alternatively specify a fixed surface charge density.

```
structure{
    ...
    region{ # charge sheet
        line{ x = [10E0, 10E0 + $Width]}            # between contact/GaAs at 10 nm
        doping{
            constant{
                name = "negative-interface-charge"   # name of impurity
                conc = 2.7675e20                     # doping concentration [cm-3]
            }
        }
    }
}
```

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

Figure 7.4.1.7 shows that the red curve (= "ohmic" contact with interface charge density σ (surface states) of -2.7675 · 10^{12} $|e|$ /cm^2 = -4.4340 · 10^{-3} C/m^2) is equivalent to the black curve (Schottky barrier of 0.53 eV).

A sheet charge density of -2.7675 · 10^{12} cm^{-2} corresponds to a volume charge of -2.7675 · 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 = 387.3 kV/cm

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

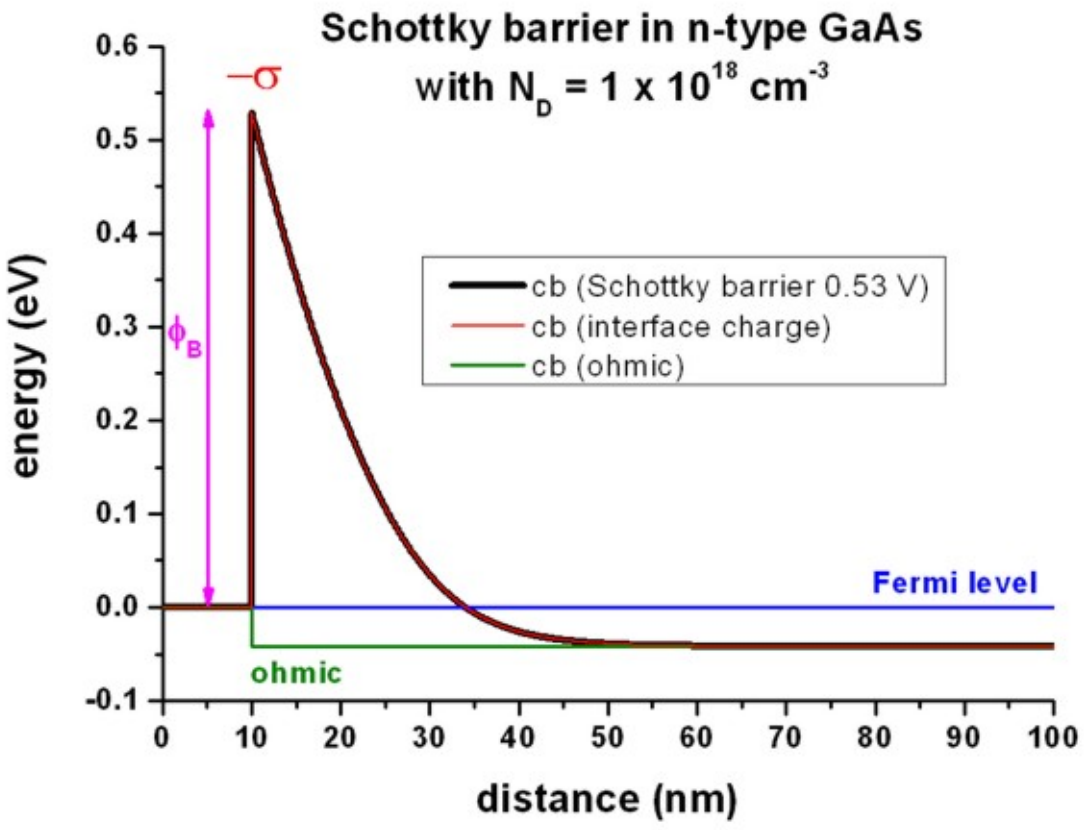


Figure 7.4.1.7: Calculated conduction band profile

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 276.75 $\cdot$ 10^{18} cm^{-3}. This corresponds to a sheet charge density of 2.7675 $\cdot$ 10^{12} cm^{-2} where we assume the states to be fully ionized.

```
impurities{
    ...
    acceptor{ # p-type
        name = "impurity"   # refer to region with name impurity
        energy = -1000E0    # all ionized
        degeneracy = 4      # degeneracy of energy levels, 2 for n-type, 4 for p-type
    }
}
```

The results are the same as shown in Figure 7.4.1.7 for the interface charges.

7.4.2 Band structure

Empirical tight-binding sp^3s^* band structure of GaAs, GaP, AlAs, InAs, C (diamond) and Si

The input files to be used are:

- *1D_TightBinding_bulk_GaAs.in*
- *1D_TightBinding_bulk_GaAs_so.in*
- *1D_TightBinding_bulk_Al0.3Ga0.7As.in*
- *1D_TightBinding_bulk_GaP.in*
- *1D_TightBinding_bulk_GaP_so.in*
- *1D_TightBinding_bulk_AlAs.in*
- *1D_TightBinding_bulk_AlAs_so.in*
- *1D_TightBinding_bulk_C.in*
- *1D_TightBinding_bulk_Si.in*
- *1D_TightBinding_bulk_Ge.in*
- *1D_TightBinding_bulk_InAs_so.in*
- *1D_TightBinding_bulk_AlSb_so.in*
- *1D_TightBinding_bulk_InSb_so.in*
- *1D_TightBinding_bulk_Al0.5In0.5Sb.in*

Empirical tight-binding sp^3s^* band structure of GaAs and GaP

The empirical tight-binding model that is used here is based on the sp^3s^* Hamiltonian, i.e. the 10 x 10 matrix given in Table (A) of *[VoglJPCS1983]*.

In addition, we include spin-orbit coupling leading to a 20 x 20 matrix. The additional terms arising due to spin-orbit coupling are given for instance on p. R5 of *[CarloSST2003]*.

We note that nowadays much better theoretical methods are available for calculating the band structure of bulk materials. However, for educational purposes, the chosen sp^3s^* method should be sufficient.

In this tutorial, we calculate the bulk band structure of

- GaAs, GaP and AlAs *without* spin-orbit coupling using the parameters of *[VoglJPCS1983]* at T = 0 K
- GaAs, GaP and AlAs *including* spin-orbit coupling using the parameters of *[KlimeckSM2000]* at T = 300 K

Input

The values for the tight binding parametrization have to be specified in the input file:

```
$numeric-control
...
!-------------------------------------------------------------------------------
! Tight-binding parameters for GaAs (values of [Klimeck]). The units are [eV].
!-------------------------------------------------------------------------------
!tight-binding-parameters = -3.53284d0              ! Esa (GaAs)
                            0.27772d0              ! Epa
                           -8.11499d0              ! Esc
                            4.57341d0              ! Epc
                           12.33930d0              ! Es_a
                            4.31241d0              ! Es_c
                           -6.87653d0              ! Vss
                            1.33572d0              ! Vxx
                            5.07596d0              ! Vxy
                            0d0                    ! Vs_s_
                            2.85929d0              ! Vsa_pc
                           11.09774d0              ! Vsc_pa
                            6.31619d0              ! Vs_a_pc
                            5.02335d0              ! Vs_c_pa
                            0.32703d0  0.12000d0   ! Delta_so_a Delta_so_c
! Note: a = anion, c = cation
!       s_ = s*
```

For more information about the meaning of these parameters, refer to the above cited references.

Output

The output of the calculated tight-binding band structure can be found in the following file: `TightBinding/BandStructure.dat`

The first column contains the number of the grid point in the Brillouin zone. These grid points run

- from L point to Gamma point (along Lambda)
- from Gamma point to X point (along Delta)
- from X point to the U, K points
- from U,K points to Gamma point (along Sigma)

The next columns are the eigenvalues of the tight-binding Hamiltonian in units of [eV] for each grid point in $\mathbf{k} = (k_x, k_y, k_z)$ space.

The file `TightBinding/BandStructure_without_so.dat` contains the tight-binding band structure without spin-orbit coupling.

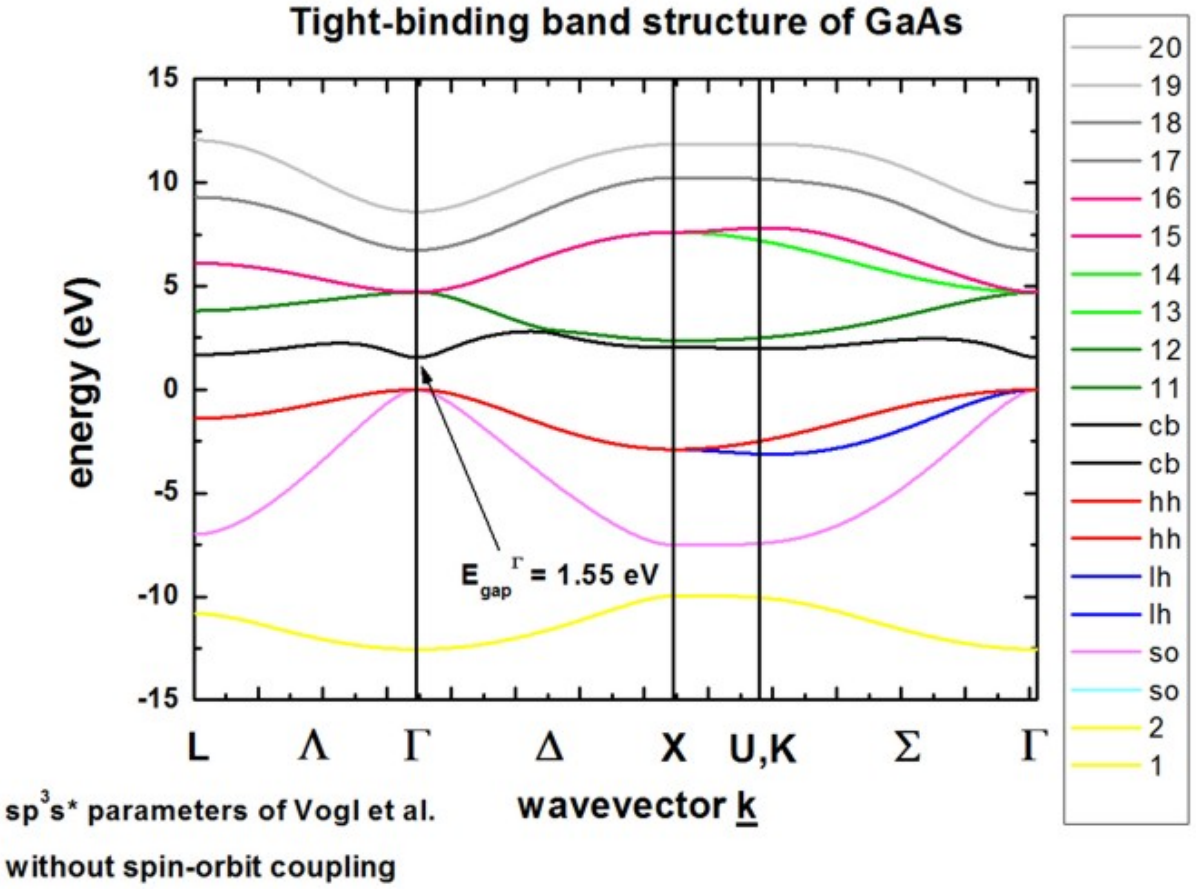
The file `TightBinding/k_vectors.dat` contains for each point the information to which k point it belongs to.

no.	kx	ky	kz	|k|	kx [2pi/a]	ky [2pi/a]	kz [2pi/a]	|k| [2pi/a]
1	0.314159E+00	0.314159E+00	0.314159E+00	0.544140E+00	0.500000E+00	0.500000E+00	0.500000E+00	0.866025E+00

Note: Currently the units of k_x, k_y and k_z do not take into account the lattice constant a. This should be modified. The values for k_x, k_y and k_z in units of [2pi/a] are correct, however. Another improvement would be to calculate and output the three-dimensional energy dispersion E(k_x, k_z, k_z) and two-dimensional slices E(k_x, k_z, 0) through the three-dimensional energy dispersion E(k_x, k_z, k_z) for a constant value of k_z, e.g. k_z =0.

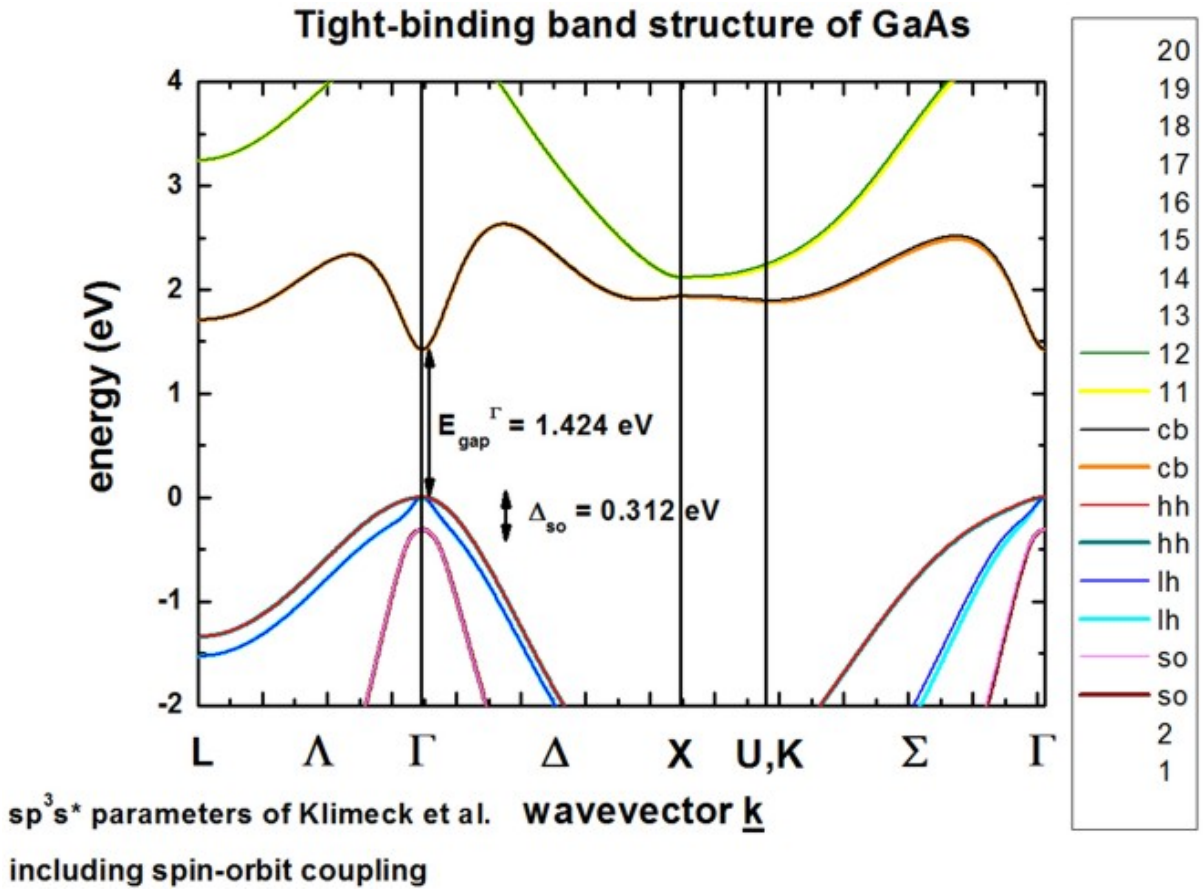
Results

GaAs without spin-orbit coupling from `1D_TightBinding_bulk_GaAs.in`



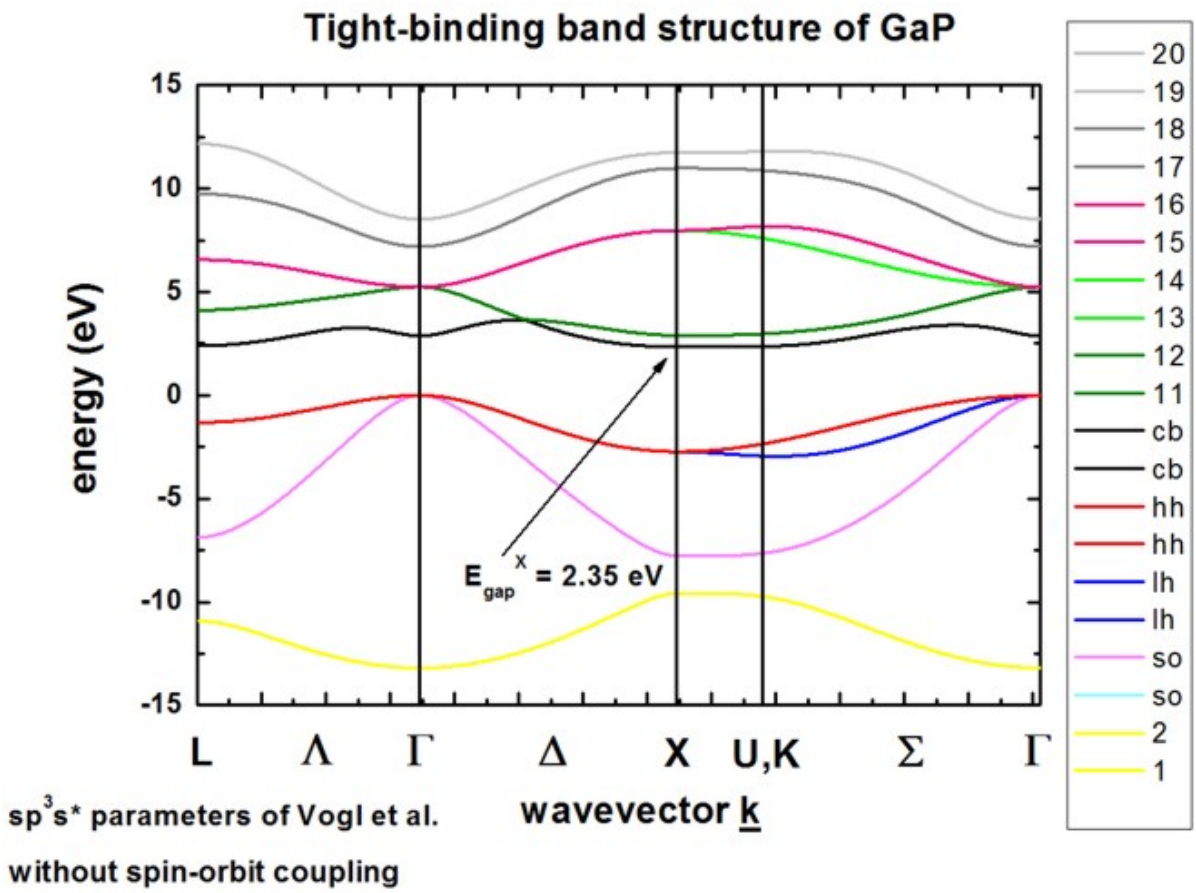
The calculated band structure is in excellent agreement with Fig. 11(d) of *[VoglJPCS1983]*. The conduction band minimum is at the Gamma point (direct band gap). Because spin-orbit coupling is not included in the Hamiltonian, the sp^3s^* empirical tight-binding parameters were taken from *[VoglJPCS1983]* at T = 0 K.

GaAs including spin-orbit coupling from `1D_TightBinding_bulk_GaAs_so.in`



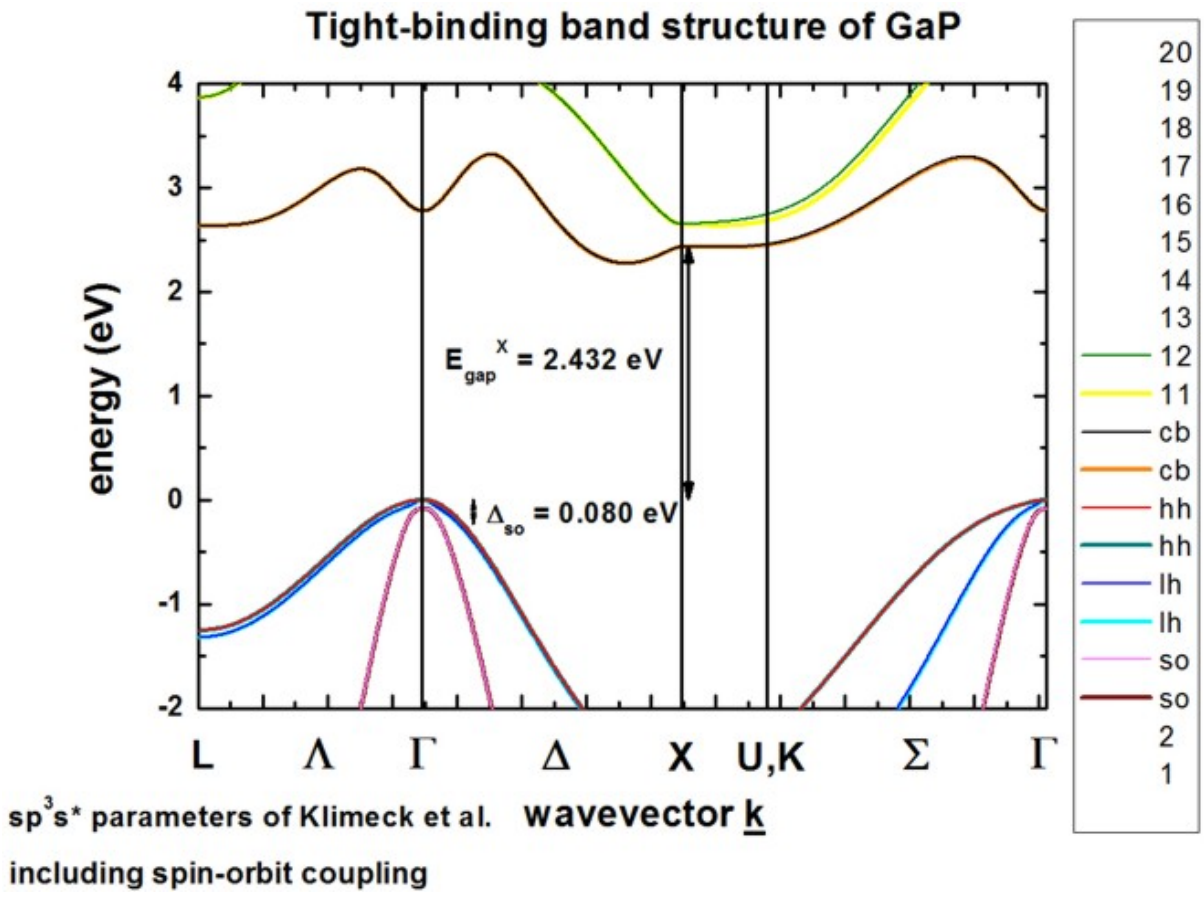
The calculated band structure is in excellent agreement with Fig. 1 of *[KlimeckSM2000]*. The conduction band minimum is at the Gamma point (irect band gap). Spin-orbit coupling lifts the degeneracy of heavy/light hole and split-off hole at the Gamma point. Heavy and light hole are still degerate at the Gamma point. The sp^3s^* empirical tight-binding parameters were taken from *[KlimeckSM2000]* at T = 300 K.

GaP without spin-orbit coupling from `1D_TightBinding_bulk_GaP.in`



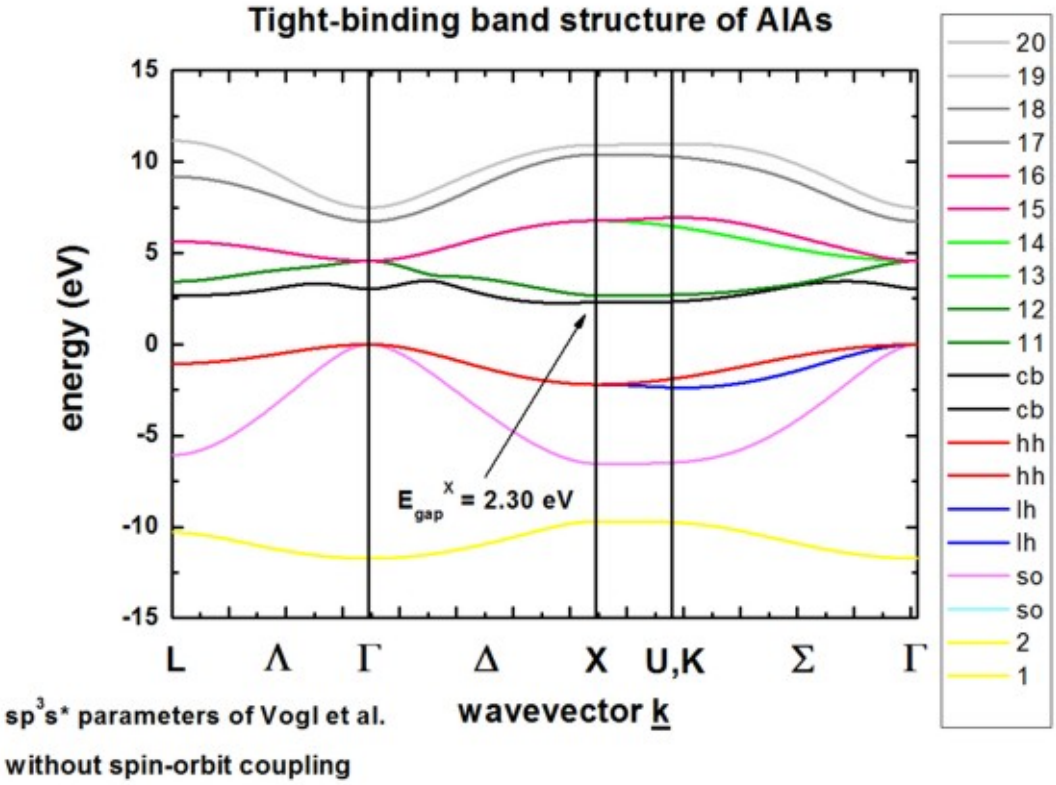
The calculated band structure is in excellent agreement with Fig. 2 of *[VoglJPCS1983]*. The conduction band minimum is calculated to be at the X point (indirect band gap). Because spin-orbit coupling is not included in the Hamiltonian, heavy, light and the split-off hole are degenerate at the Gamma point, i.e at $\mathbf{k} = (k_x, k_y, k_z) = 0$. The sp^3s^* empirical tight-binding parameters were taken from *[VoglJPCS1983]* at T = 0 K.

GaP including spin-orbit coupling from `1D_TightBinding_bulk_GaP_so.in`

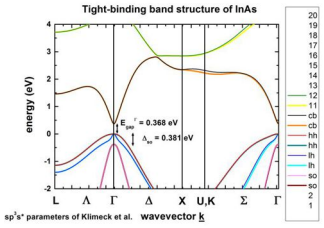


The calculated band structure is in excellent agreement with Fig. 1 of *[KlimeckSM2000]*. The conduction band minimum is in the vincinity of the X point at the Delta line (indirect band gap), so-called *camel's back*. Spin-orbit coupling lifts the degeneracy of heavy/light hole and split-off hole at the Gamma point. Heavy and light hole are still degenerate at the Gamma point. The sp^3s^* empirical tight-binding parameters were taken from *[KlimeckSM2000]* at T = 300 K.

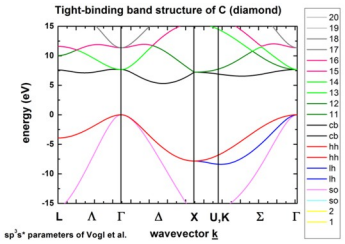
AlAs without spin-orbit coupling from `1D_TightBinding_bulk_AlAs.in`



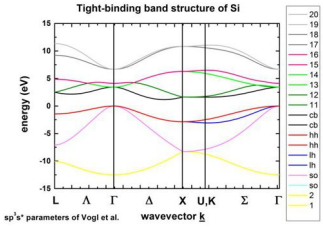
InAs including spin-orbit coupling from `1D_TightBinding_bulk_InAs_so.in`



C (diamond) without spin-orbit coupling from `1D_TightBinding_bulk_C.in`



Si (silicon) without spin-orbit coupling from `1D_TightBinding_bulk_Si.in`



The k space resolution, i.e. the number of grid points on the axis of these plots can be adjusted. This can be done with:

```
$tighten
calculate-tight-binding-tighten = no
destination-directory           = TightBinding/
number-of-k-points              = 50            ! This number corresponds to 50 k␣
↪points between the Gamma point and the X point
                                                ! The number of k points along the␣
↪other directions are scaled accordingly.
```

Author: Stefan Birner, Reinhard Scholz

Tight-binding band structure of graphene

The input file used is:

- *1D_TightBinding_graphene.in*

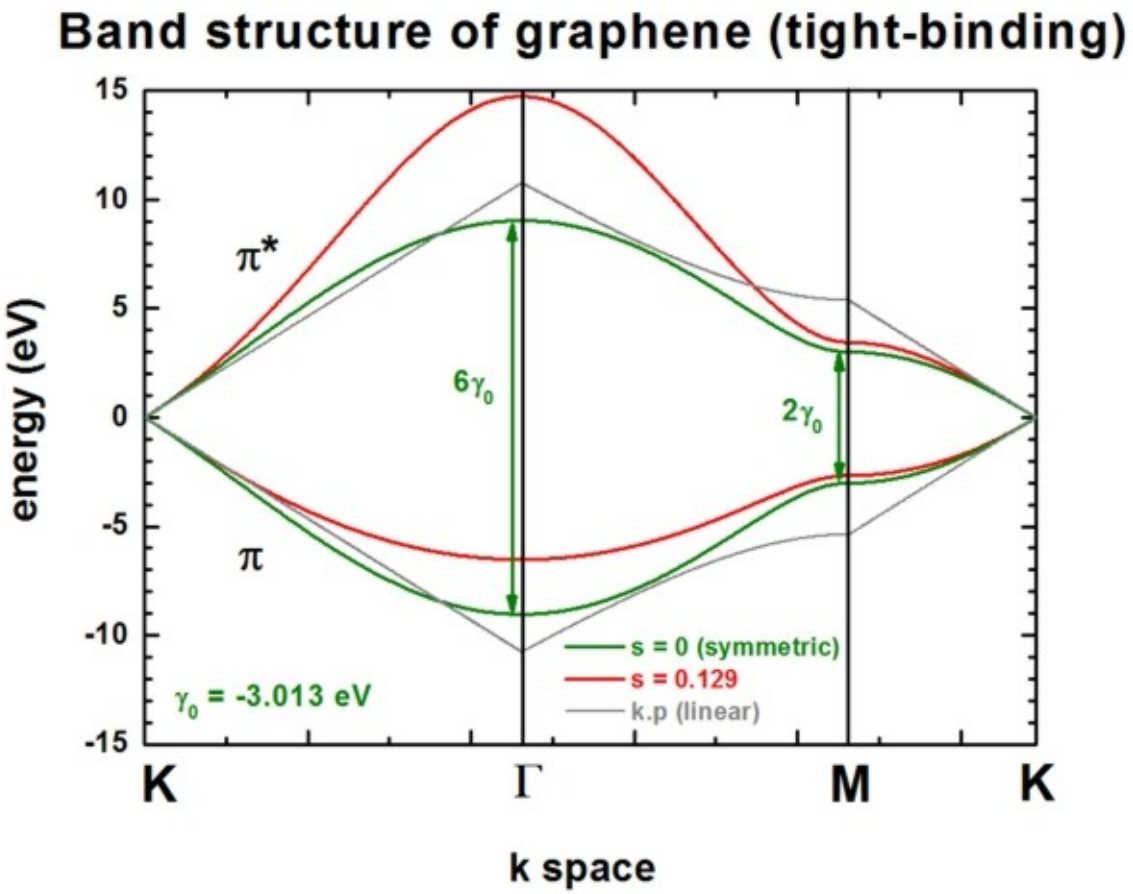
Nearest-neighbor tight-binding approximation

In this tutorial we calculate the bulk band structure of graphene which is a two-dimensional crystal (i.e. a monolayer of graphite) using a standard tight-binding approach. For more details, see for example the article *[SaitoS2001]*

The following figure shows the conduction band pi* (upper part) and valence band pi (lower part) of graphene along special high-symmetry directions in the two-dimensional hexagonal Brillouin zone (**k** space).

The high symmetry points that are used in this graph (from left to right) are:

- K: $\mathbf{k} = (k_x, k_y) = (0, \frac{2}{3}) \times \frac{2\pi}{a}$
- Gamma: $\mathbf{k} = (k_x, k_y) = (0, 0)$
- M: $\mathbf{k} = (k_x, k_y) = (\frac{1}{3}^{1/2}, 0) \times \frac{2\pi}{a}$
- K': $\mathbf{k} = (k_x, k_y) = (\frac{1}{3}^{1/2}, \frac{1}{3}) \times \frac{2\pi}{a}$



Two lines correspond to the case where the s_0 parameter is set to zero, i.e. in that case the dispersion of both pi* and pi is the same (apart from the sign, i.e. they are symmetric with respect to the Fermi level E_F = 0 eV). In this case the splitting energy at Gamma is three times as large as at the M point:

- splitting at Gamma: 6 gamma_0 (for s_0 = 0)
- splitting at M: 2 gamma_0 (for s_0 = 0)

Two lines correspond to the case where s0 = 0.129. pi* and pi are then nonsymmetric and are close to calculations from first principles and experimental data.

The data points are contained in the following file: `TightBinding/BandStructureGraphene.dat` The first column contains integers which refer to the x axis (i.e. numbering of k points), the second column contains the eigenvalue of pi* in units of [eV] (conduction band), the third column contains the eigenvalue of pi in units of [eV] (valence band). The file `TightBinding/k_vectors.dat` contains information about which integer corresponds to which k_x and k_y value.

The general formula for these lines read:

$$E_{+,-} = \frac{[E_{2p} \pm \Gamma_0 w(\mathbf{k})]}{1 \pm s_0 w(\mathbf{k})}$$

The parameters can be specified in the input file:

```
$numeric-control
...
(nearest-neighbor approximation, i.e. 3 parameters)
tight-binding-parameters =  0d0      ! [eV] E2p: site energy of the 2pz atomic orbital␣
↪(orbital energy)
                          -3.013d0 ! [eV] gamma0: C-C transfer energy  (usually it␣
↪holds: -3 eV < gamma0 < -2.5 eV)
                           0.129d0 ! []   s0 = 0.129: denotes the overlap of the␣
```

(continues on next page)

(continued from previous page)

```
↪electronic wave function on adjacent sites
                        !  0d0     ! []   s0 = 0:       denotes the overlap of the␣
↪electronic wave function on adjacent sites
                                    !      (usually it holds: s0 < 0.1. Since this␣
↪value is small, very often, it is neglected.)
```

Then there are two lines that are linear around the K point ($\mathbf{k} \cdot \mathbf{p}$ approximation or linear expansion). Their linear dispersion is independent of the parameter s_0. Thus for small values of k (i.e. with respect to the K point), the energy dispersion can be approximated by a linear dispersion relation.

$$E(k) = E_{2p} \pm \hbar V_F |\mathbf{k}| = E_{2p} \pm 3^{1/2} \Gamma_0 \frac{ka}{2}$$

where

- a is the latticer constant of graphene (a = 0.24612 nm)
- the Fermi velocity of the charge varriers is given by $V_F = 3^{1/2} |\Gamma_| \frac{a}{2\hbar} = 0.98 \times 106\text{m s}^{-1} \simeq 0.003c$
- c is the velocity of light

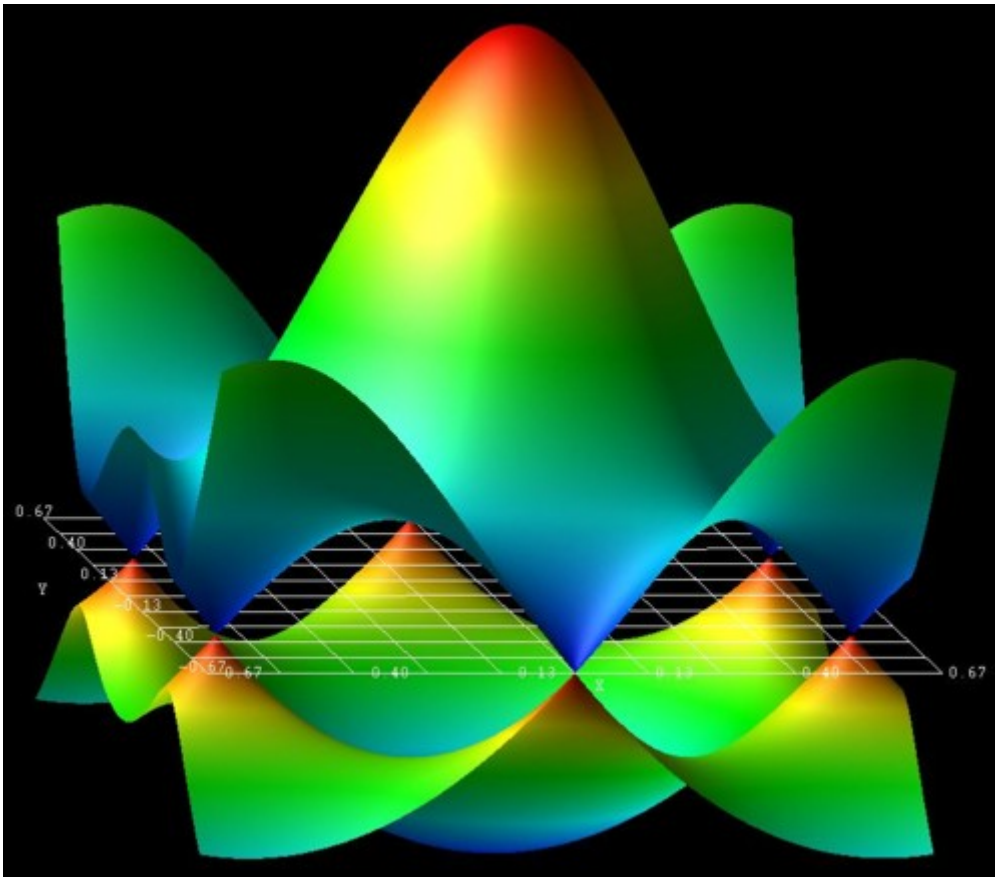
These data points are contained in this file: `TightBinding/BandStructureGraphene_kp.dat`

At the K point, the band gap is zero.

The following figure shows the energy dispersion E(k_x, k_y) of graphene for s_0 = 0.129. At the K points, the band gap is zero. The point in the middle is the Gamma point. k_x is from $[-\frac{2}{3}, \frac{2}{3}]\frac{2\pi}{a}$, the same holds for k_y.

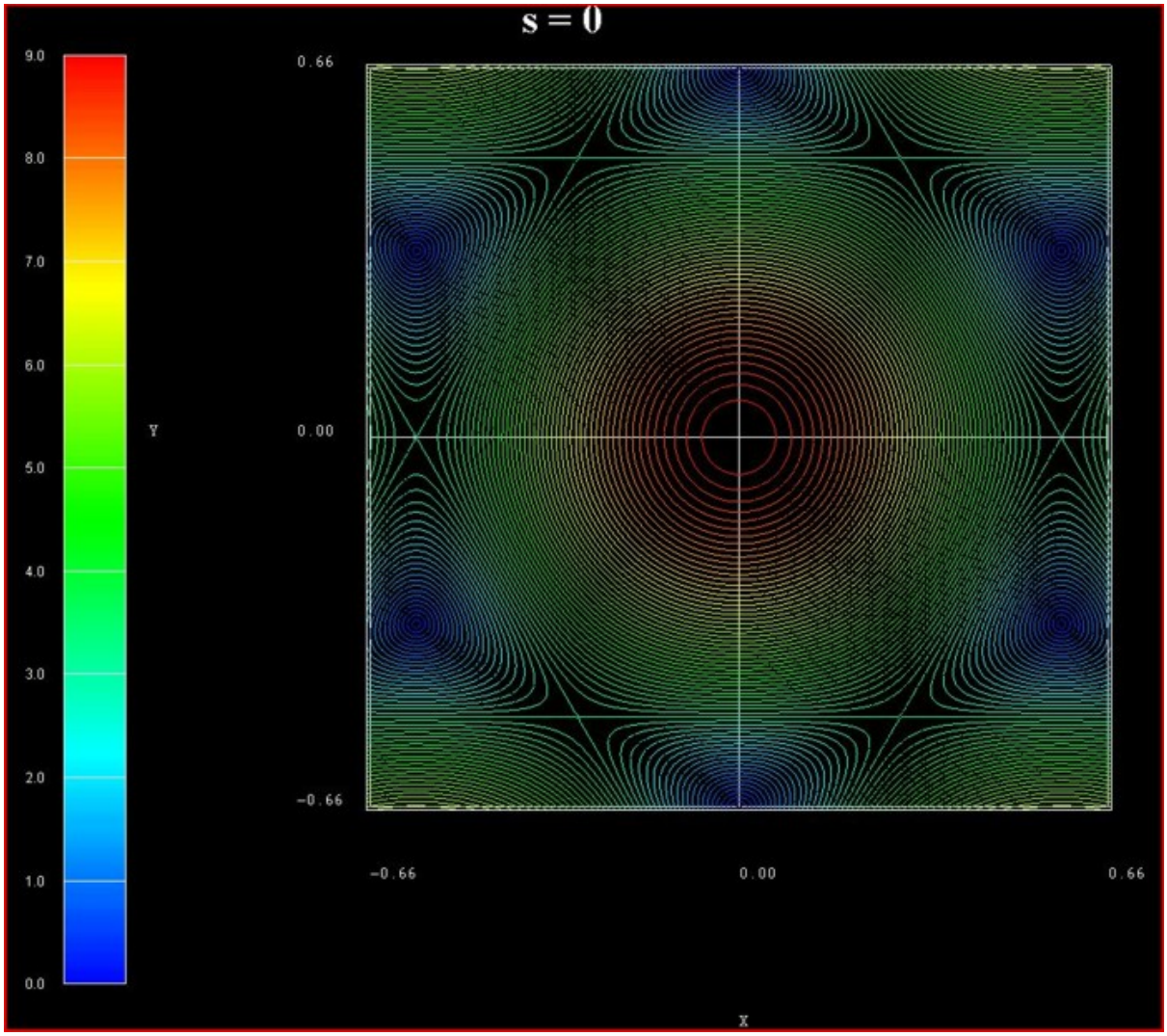
These data points are contained in the files:

- `TightBinding/BandStructureGraphene_cb.vtr`
- `TightBinding/BandStructureGraphene_vb.vtr`



The figure has been generated using the AVS/Express software.

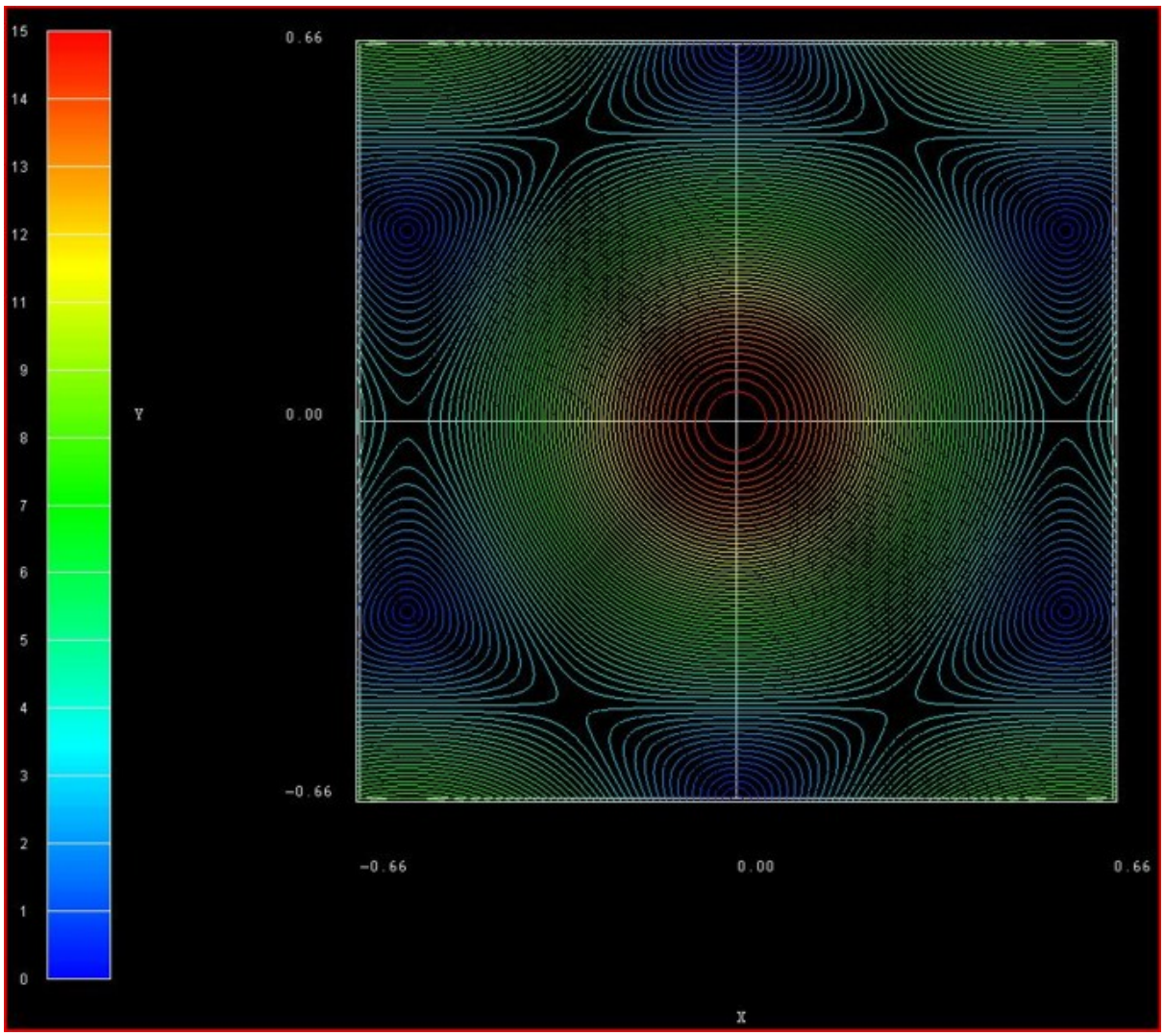
The following figure shows the contour plot of the energy dispersion E(k_x, k_y) of graphene for s_0 = 0 for the conduction band. (Note that the valence band dispersion is identical, apart from the sign, for s_0 = 0.) These data points are contained in the file: `TightBinding/BandStructureGraphene_cb.vtr`

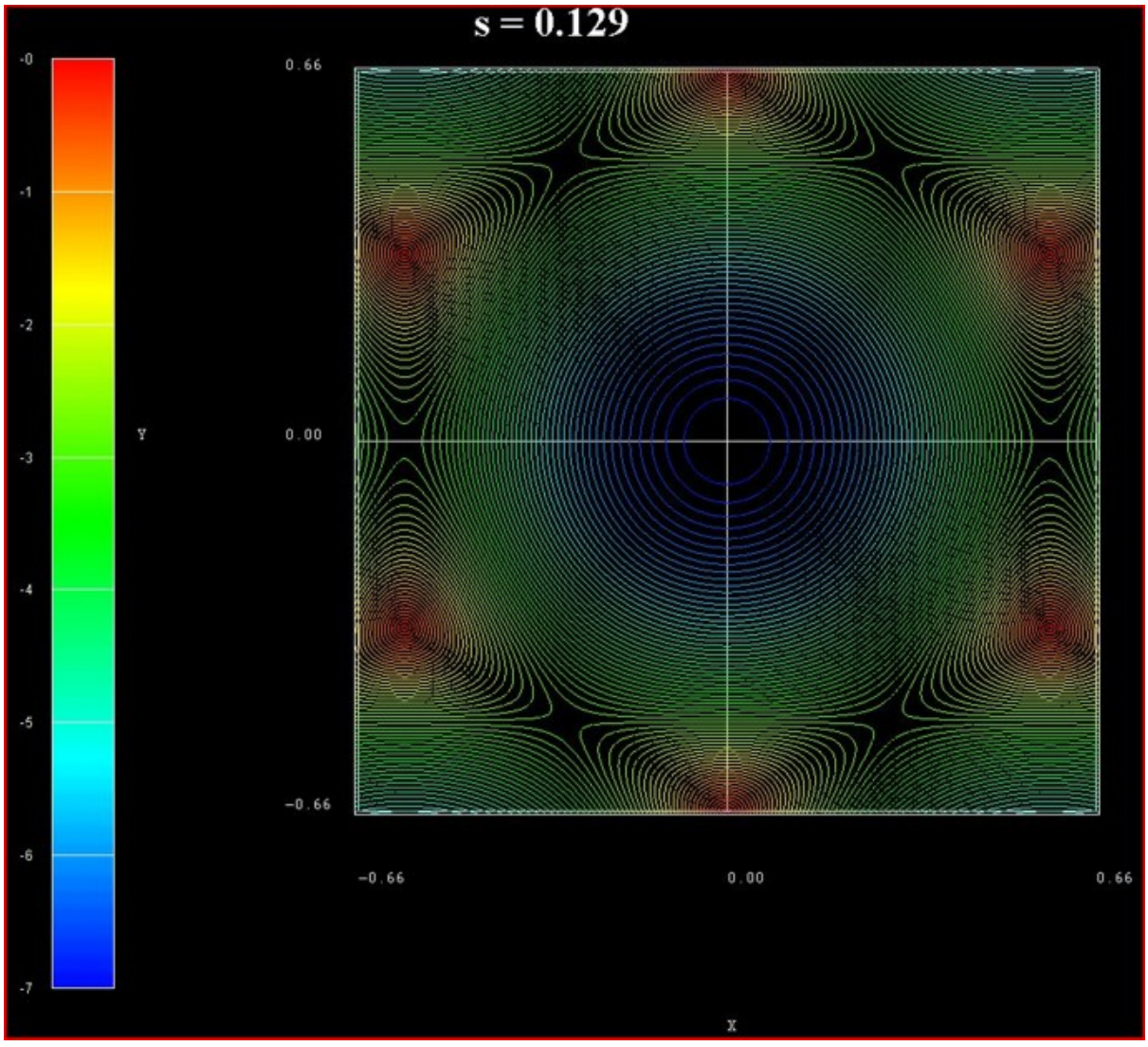


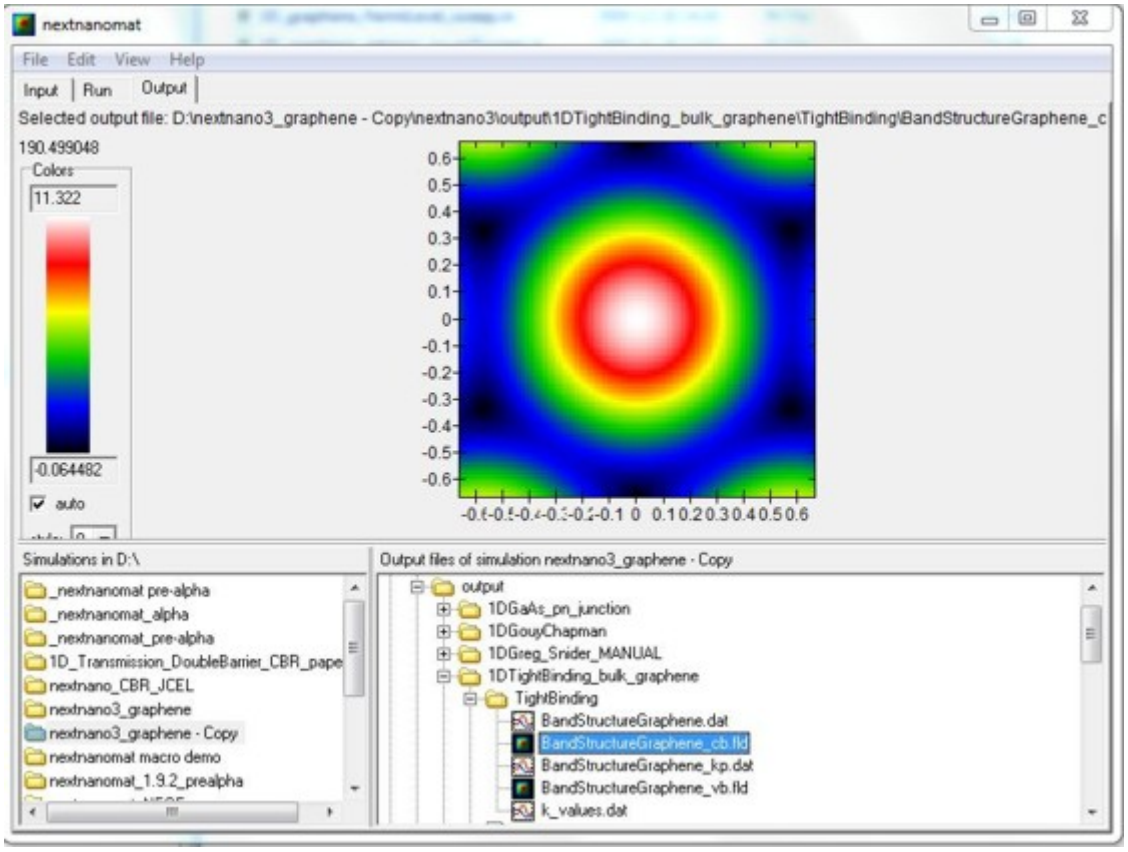
The following figure shows the contour plot of the energy dispersion E(k_x, k_y) of graphene for s_0 = 0.129 for the conduction band. These data points are contained in the file: `TightBinding/BandStructureGraphene_cb.vtr`

The following figure shows the contour plot of the energy dispersion E(k_x, k_y) of graphene for s_0 = 0.129 for the valence band. These data points are contained in the file: `TightBinding/BandStructureGraphene_vb.vtr`

The following figure is the *nextnanomat* screenshot for the conduction band dispersion E(k_x, k_y). The six dark areas correspong to the Dirac points in graphene.


15
14
13
12
11
9
8
7
6
5
4
3
2
1
0
Y
0.66
0.00
-0.66
-0.66
0.00
0.66
X


s = 0.129
-0
-1
-2
-3
-4
-5
-6
-7
Y
0.66
0.00
-0.66
-0.66
0.00
0.66
X



Output Options

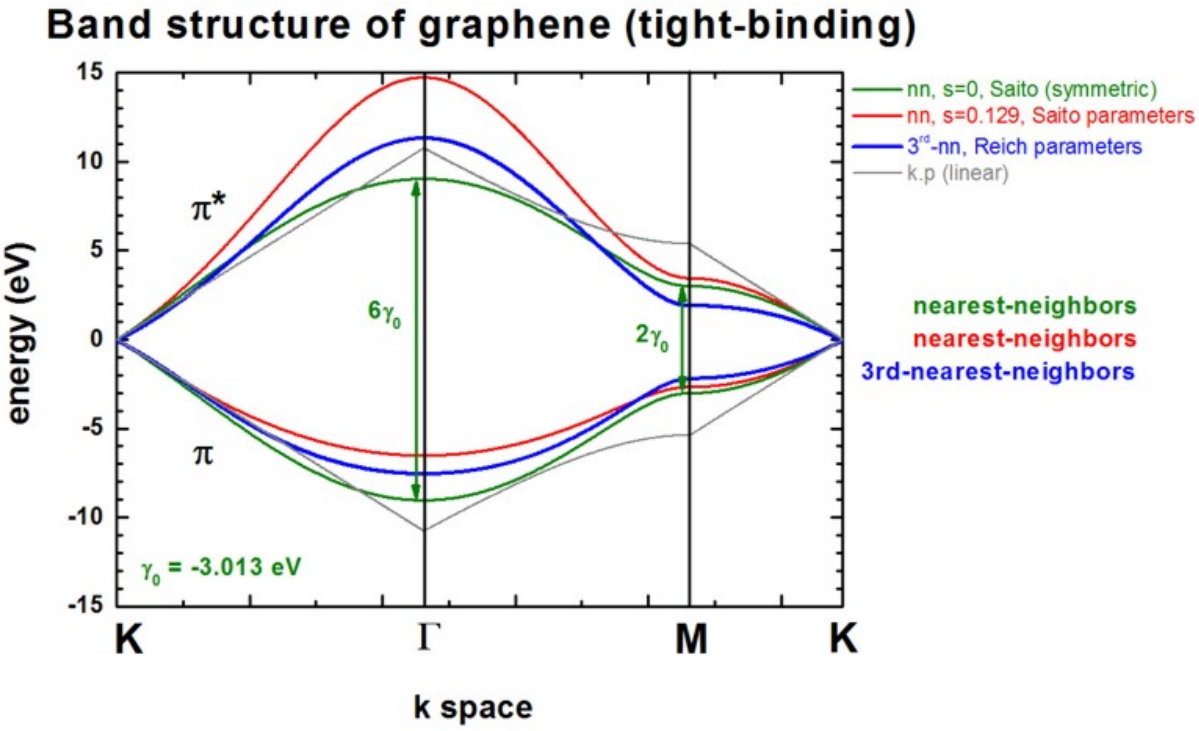
```
!------------------------------------------------------------------------------!
$output-kp-data
 ...
bulk-kp-dispersion-3D = yes  ! In this case, this refers to the bulk 2D tight-binding␣
↪energy dispersion E(kx,ky).
                             ! If yes, then the two-dimensional energy dispersion␣
↪E(kx,ky) is written out.



$tighten
destination-directory = TightBinding/
!------------------------------------------------------------------------------!
! This parameter defines the resolution of the E(kx,ky) energy dispersion.
! E.g. if 'number-of-k-points = 100' then the kx gridding has a total of
! [-100,...,0,...,100] grid points,
! i.e. a total of 2 * 100 + 1 = 201 grid points along the kx direction.
! The same applies for ky direction.
!------------------------------------------------------------------------------!
number-of-k-points = 100                                                ! ==> 2 *␣
↪n + 1
```

Third-nearest-neighbor tight-binding approximation

The following figure shows the band structure of graphene. All lines are identical to the ones shown already above with the exception of the blue lines which is the third-nearest-neighbor tight-binding approximation.



The third-nearest-neighbor tight-binding approximation is described in *ReichPR2002*

The following parameters have been used for the third-nearest neighbor approximation, i.e. 7 parameters:

```
tight-binding-parameters = -0.28d0  ! [eV] E2p: site energy of the 2pz atomic orbital␣
↪(orbital energy)
                           -2.97d0  ! [eV] gamma0: C-C transfer energy (nearest-
↪neighbor, nn)
                            0.073d0 ! []   s0: denotes the overlap of the electronic␣
↪wave function on adjacent sites (nn)
                           -0.073d0 ! [eV] gamma1: (2nd-nn)
                            0.018d0 ! []   s1:     (2nd-nn)
                           -0.33d0  ! [eV] gamma2: (3rd-nn)
                            0.026d0 ! []   s2:     (3rd-nn)
```

The parameters are taken from Reich et al. Note that the band gap at the K point is not exactly zero when using these parameters.)

More Options

Inside the code several options, i.e. several algorihms, exist to setup the tight-binding Hamiltonian:

```
tight-binding-method = bulk-graphene-Saito-nn        !     nn =         nearest-neighbor
                     = bulk-graphene-Scholz-nn       !     nn =         nearest-neighbor
                     = bulk-graphene-Scholz-3rd-nn   ! 3rd-nn = third-nearest-neighbor
                     = bulk-graphene-Reich-3rd-nn    ! 3rd-nn = third-nearest-neighbor
```

Special option for third-nearest-neighbor tight-binding approximation in graphene:

```
tight-binding-calculate-parameter = no     ! (default) use 7 parameters (E2p, gamma0,␣
↪gamma1, gamma2, s0, s1, s2)
                                  = E2p    !             use 6 parameters (gamma0,␣
↪gamma1, gamma2, s0, s1, s2)
                                  = gamma1 !             use 6 parameters (E2p,␣
↪gamma0, gamma2, s0, s1, s2)
```

E_{2p} or gamma1 can be calculated internally in order to force $E(K) = 0$ eV where K is the K point in the Brillouin zone. In that case, only 6 parameters are used (although 7 parameters have to be present in the input file). The parameter to be calculated is simply ignored inside the code.

The k space resolution, i.e. the number of grid points on the axis of these plots can be adjusted.

```
$tighten
calculate-tight-binding-tighten = no              !
destination-directory           = TightBinding/
number-of-k-points              = 50              ! This corresponds to 50 k points␣
↪between the Gamma point and the M point.
                                                  ! The number of k points along the␣
↪other directions are scaled correspondingly.
```

30-band $\mathbf{k} \cdot \mathbf{p}$ band structure calculation

Note: This tutorial is under in development.

Attention: The 30-band model is under development in our tool, therefore related syntax and available functionalities are expected to be changing. Strain effects are not included yet.

Input Files:

bulk_kp_dispersion_Si_SiGe_Ge_30band_nn3.in

Scope of the tutorial:

- Band structure of bulk Si, SiGe, and Ge within 30-band $\mathbf{k} \cdot \mathbf{p}$

Relevant keywords:

- `bulk-kp-dispersion`
- `kp-parameters`
- `k-vectors-sample-type`

Main adjustable parameters in the input file:

- set of parameters - `%30band_parameters`
- path of the band structure - `%Bandstructure`
- parameters controlling mole fraction of the modeled alloy `%AlloySweepActive`, `%AlloySweepSize`, `%AlloySweepSteps`

Relevant output Files:

- kp_bulk\bulk_kp30kp_dispersion*.dat (band structure)

Introduction

This tutorial shows how to calculate band structure of $Si_{1-x}Ge_x$ alloys within 30-band $\mathbf{k} \cdot \mathbf{p}$ model. It answers the following questions:

- How to trigger 30-band model in *nextnano*[3]?
- How to define path on which you want to calculate the band structure?
- Which output file contain the band structure computed within the 30-band model?

The band structures presented in this tutorial are in agreement with those reported by *M. Cardona, F. H. Pollak* and *Rideau et al.* (except vicinity of the K point for the latter, which is under investigation.)

The Input File

30-band $\mathbf{k} \cdot \mathbf{p}$ model can be called to compute bulk dispersion by specifying **bulk-kp-dispersion = 30-band** inside the group **$output-kp-data**.

```
$output-kp-data
bulk-kp-dispersion = 30-band
$end_output-kp-data
```

The path along which the band structure is computed can be specified inside the group **$tighten** by assigning one of available paths (use *autocomplete* feature in *nextnanomat*) to the attribute **k-vectors-sample-type**.

```
$tighten
k-vectors-sample-type = L-Gamma-X-W-K-L-W-X-K-Gamma
$end_tighten
```

Parameters are currently hard-coded and available only for SiGe alloys. In this tutorial two sets of parameters are used to reproduce results from *M. Cardona, F. H. Pollak* and *Rideau et al.*. Switching between the different sets of parameters is done by setting **kp-parameters** to **kp-parameters = Rideau** or to **kp-parameters = Cardona-Pollak** inside the group **$numeric-control**.

```
$numeric-control
kp-parameters = Rideau
$end_numeric-control
```

Output files

The band structure generated after running the input file *bulk_kp_dispersion_Si_SiGe_Ge_30band_nn3.in* can be found in a file *...kp_bulkbulk_kp30kp_dispersion_BrillouinZone1_L-Gamma-X-W-K-L-W-X-K-Gamma.dat*, where the first column contains indexes of following wave vectors along the path and all the following columns contain eigenvalues starting with the highest ones. Exact coordinates of wave vectors for each point of the computed band structure can be found in a related file *...kp_bulkbulk_kp30kp_dispersion_BrillouinZone1_L-Gamma-X-W-K-L-W-X-K-Gamma_k_vectors.dat*. There, the first column contains corresponding indexes of each k-point, the three following contain exact coordinates of each wave vector, and the last column stores the length of each wave vector.

Results

Cardona-Pollak parameters: 15 band Hamiltonian

In the paper of *M. Cardona and F. H. Pollak*, 15-band kp Hamiltonian is introduced to model band structures of Si and Ge. The 15-band Hamiltonian disregards spin-orbit interactions, resulting in spin degeneracy across all bands. For this tutorial, nextnano GmbH computes all 30 bands, while keeping spin-orbit couplings at zero.

To use the parameters from this paper, change **kp-parameters** to **Cardona-Pollak**

```
$numeric_control
kp_parameters = Cardona-Pollak
$end_numeric_control
```

The variable `%AlloyContent` in the input file can be adjusted to select between 0 (pure Germanium) and 1 (pure Silicon).

The resulting band structures for both pure Germanium and pure Silicon are illustrated in Figures Figure 7.4.2.1, and Figure 7.4.2.2, respectively.

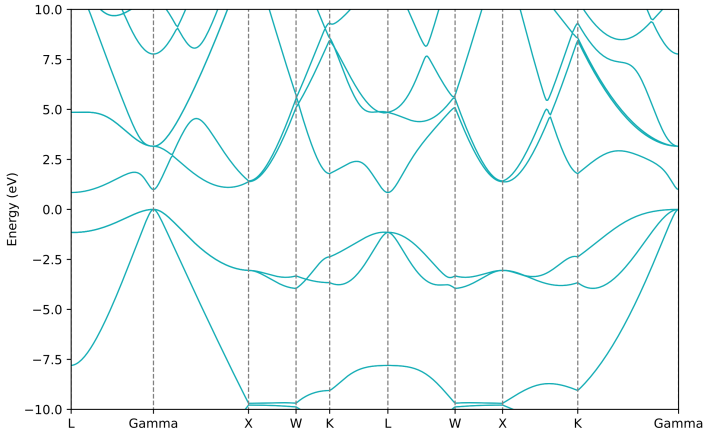


Figure 7.4.2.1: Band structure of Germanium (Ge) computed with Cardona-Pollak set of parameters , `%AlloyContent=1.0`

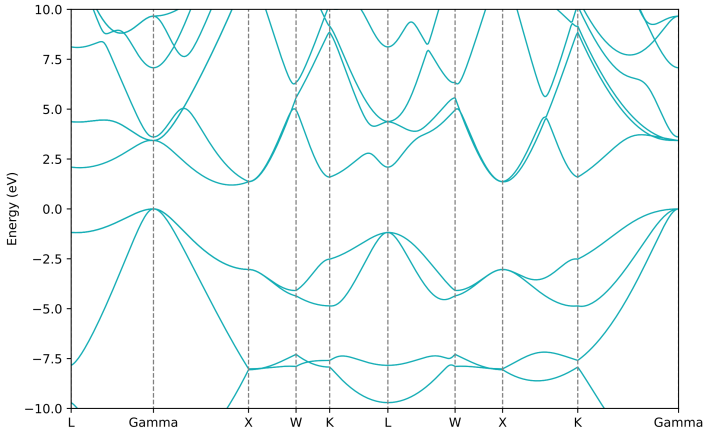


Figure 7.4.2.2: Band structure of Silicon (Si) computed with Cardona-Pollak set of parameters, `%AlloyContent=0.0`

Rideau et al parameters.

In the paper of *Rideau et al.*, the full set of parameters for 30-band Hamiltonian is given, including the spin-orbit couplings. The parameters are also proved to be valid for any alloy content in $Si_{1-x}Ge_x$alloys.

Band structures of pure Germanium , Silicon-Germanium alloy and pure Silicon generated within the input file are shown in the figures Figure 7.4.2.3, Figure 7.4.2.4 and Figure 7.4.2.5, respectively.

To use the parameters from this paper, change **kp-parameters** to **Rideau**

```
$numeric-control
kp-parameters = Rideau
$end_numeric-control
```

The variable `%AlloyContent` should be adjusted accordingly in the input file to choose the material.

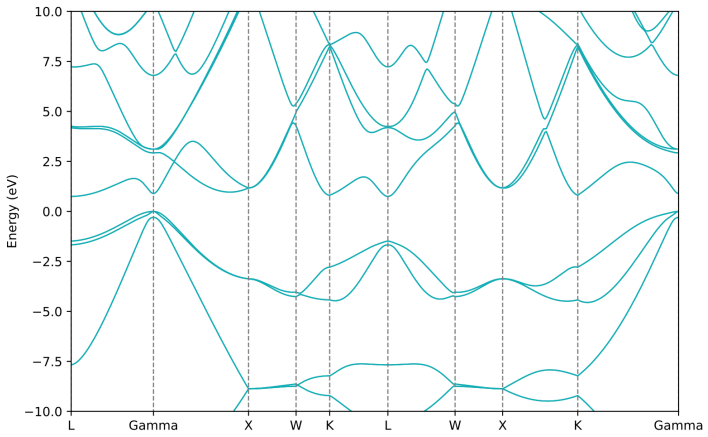


Figure 7.4.2.3: Band structure of Germanium (Ge), `%AlloyContent=1.0`, spin-orbit coupling included.

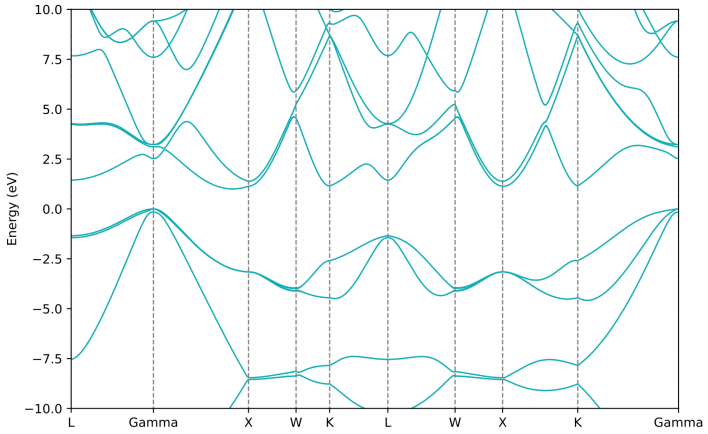


Figure 7.4.2.4: Band structure of Silicon Germanium ($Si_{0.5}Ge_{0.5}$), `%AlloyContent=0.5`, spin-orbit coupling included.

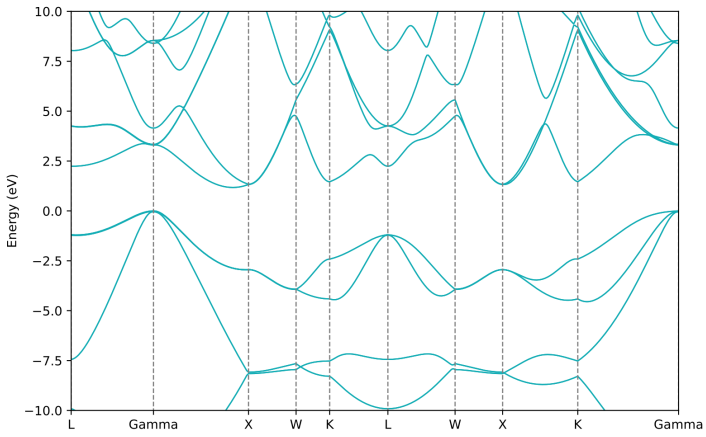


Figure 7.4.2.5: Band structure of Silicon (Si), `%AlloyContent=0.0`, , spin-orbit coupling included.

Acknowledgment

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 grant agreement No 101017194.

7.4.3 Transmission

Efficient method for the calculation of ballistic quantum transport - The CBR method (2D example)

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.

The following two *nextnano*[3] input files are based on the paper *[MamaluyCBR2003]*

- `2D_CBR_MamaluySabathilJAP2003.in`
- `2D_CBR_MamaluySabathilJAP2003_holes.in`

the latter of which simulates holes instead of electrons.

- The device consists of three leads that are called 'source', 'gate' and 'drain' in this example.
- In the middle of the device a potential barrier of two-dimensional Gaussian shape effectively expels the electrons from the center.
- In the upper part of the device, a thin tunneling double barrier is present.
- The device dimensions are 20 nm x 20 nm.

A detailed description of the device can be found in Section V of *[MamaluyCBR2003]*.

- The effective electron mass is assumed to be constant throughout the device and equal to 0.3 m_0.
- The device region consists of 41 x 41 = 1681 grid points, which is equivalent to a grid spacing of 0.5 nm. This means that the device Hamiltonian is a matrix of size 1681 x 1681.
- The tunneling barriers have a width of 1 nm each and are separated by 3 nm.
- The maximum height of the Gaussian barrier is $E_{c,0}$ = 1 eV, the height of the double barriers is 0.4 eV.

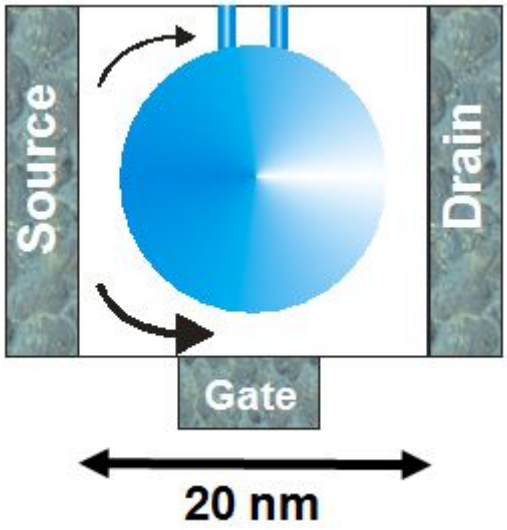
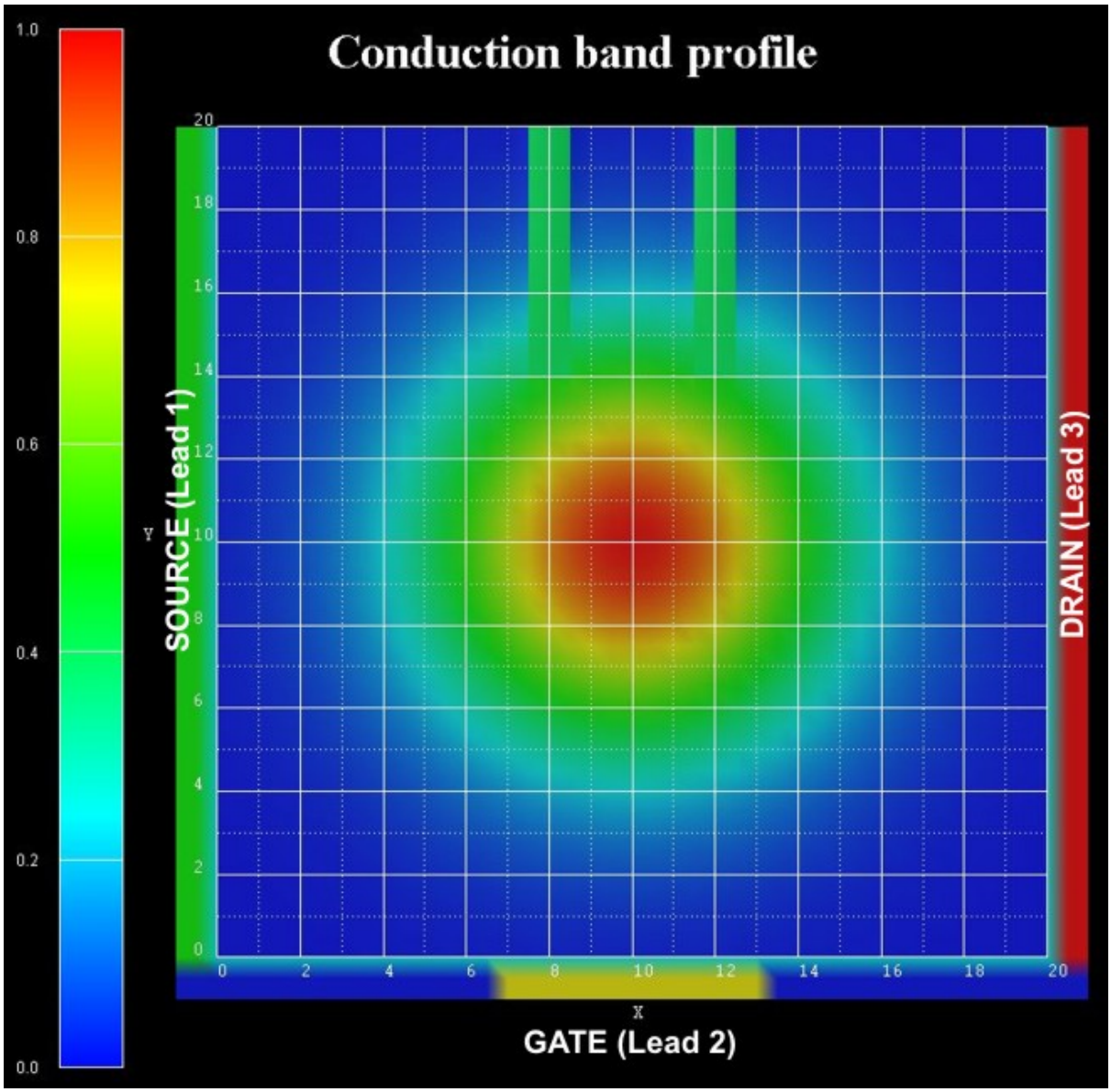
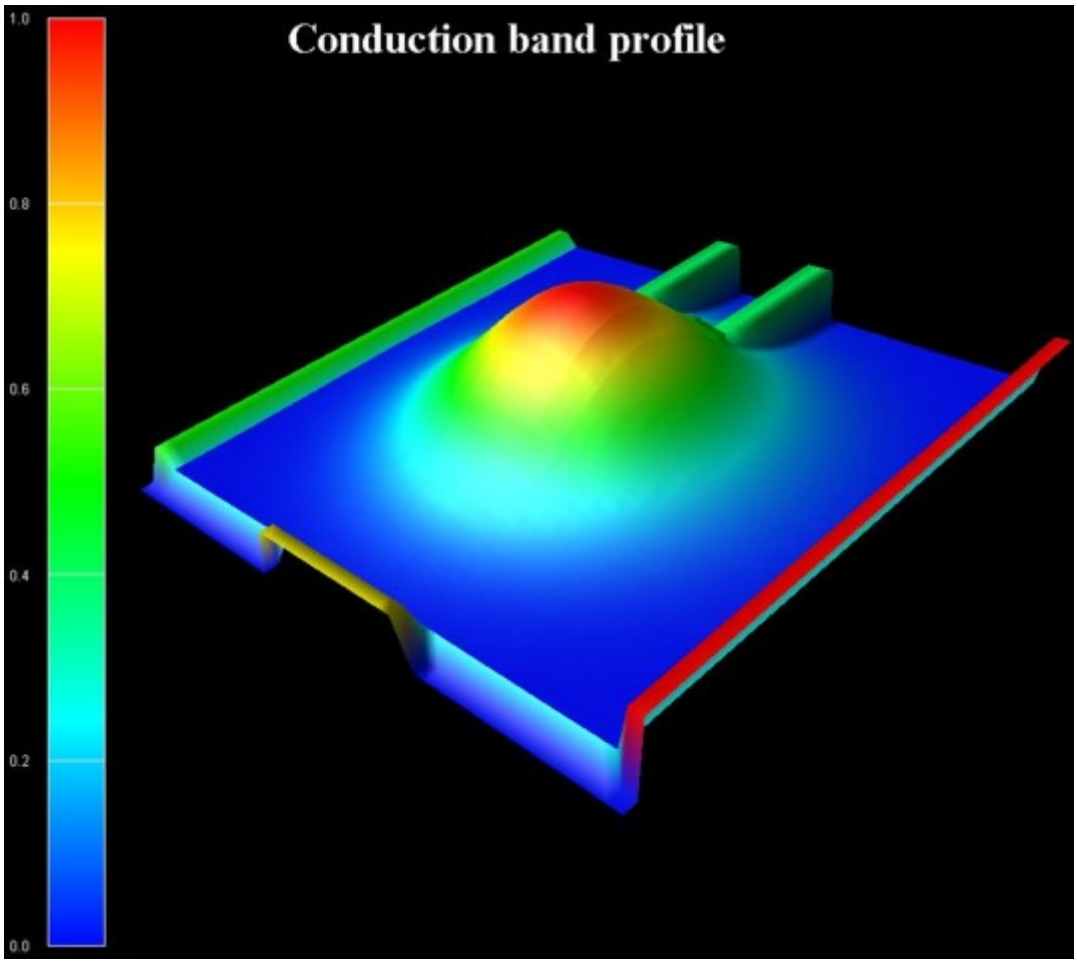


Figure 7.4.3.1: Schematic sketch of the device showing two possible paths of the electrons



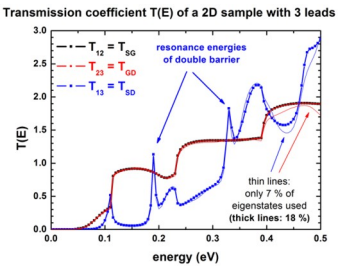
- The conduction band profile is given by $E_c = E_{c,0} \exp\left[-(x^2 + y^2)/a^2\right]$ where x and y are with respect to the center of the device, and a = 5 nm.
- The conduction band profile is achieved by using an appropriate ternary material having a 2D Gaussian alloy profile.
- The lower gate is 6 nm long, all other leads are 20 nm long.



The following figure shows the calculated transmission coefficients of the various lead combinations T_{12}, T_{23} and T_{13}. For the thick lines 18 % (303 of 1681) of all eigenvectors were used whereas for the thin lines only 7 % (118 of 1681) had to be calculated, i.e. 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.

The transmission coefficient can be found in this file:

CBR_data1/transmission2D_cb_sg001_ind000_CBR.dat



The *nextnano*[3] results differ slightly from the *[MamaluyCBR2003]* paper. Reason: The potential energy profile in the device and in the leads is not exactly identical, as well as the dimensions of the barriers. Therefore the eigenenergies and wave functions in the device, and in the leads differ slightly which explains the small deviations.

The eigenstates # 16 is a resonance state of the lower transmission path.

- 1st resonance: # 16: 0.123 eV
- Its square of the wave function is shown below.

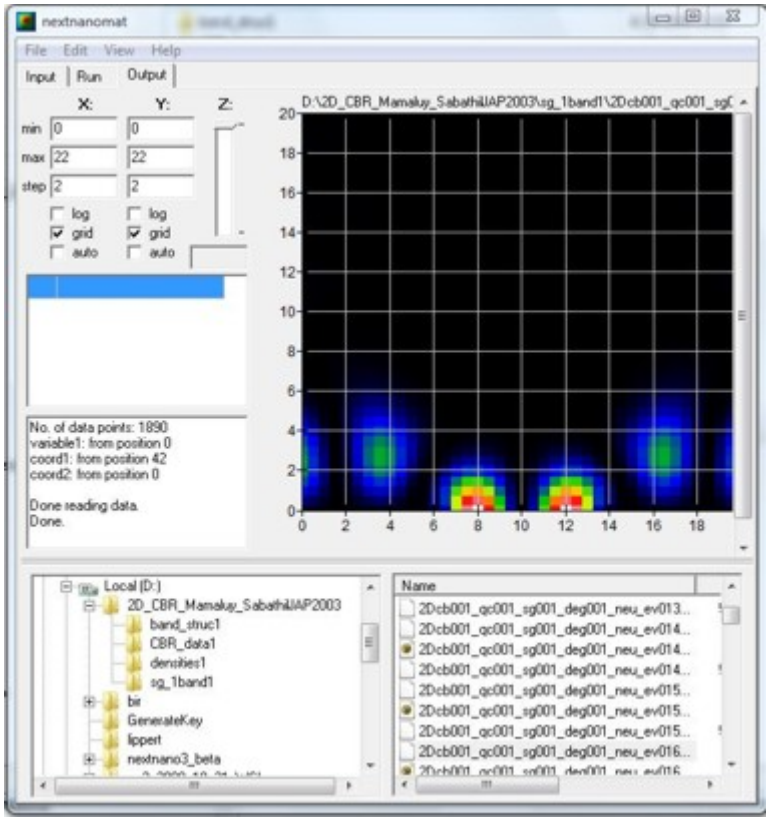
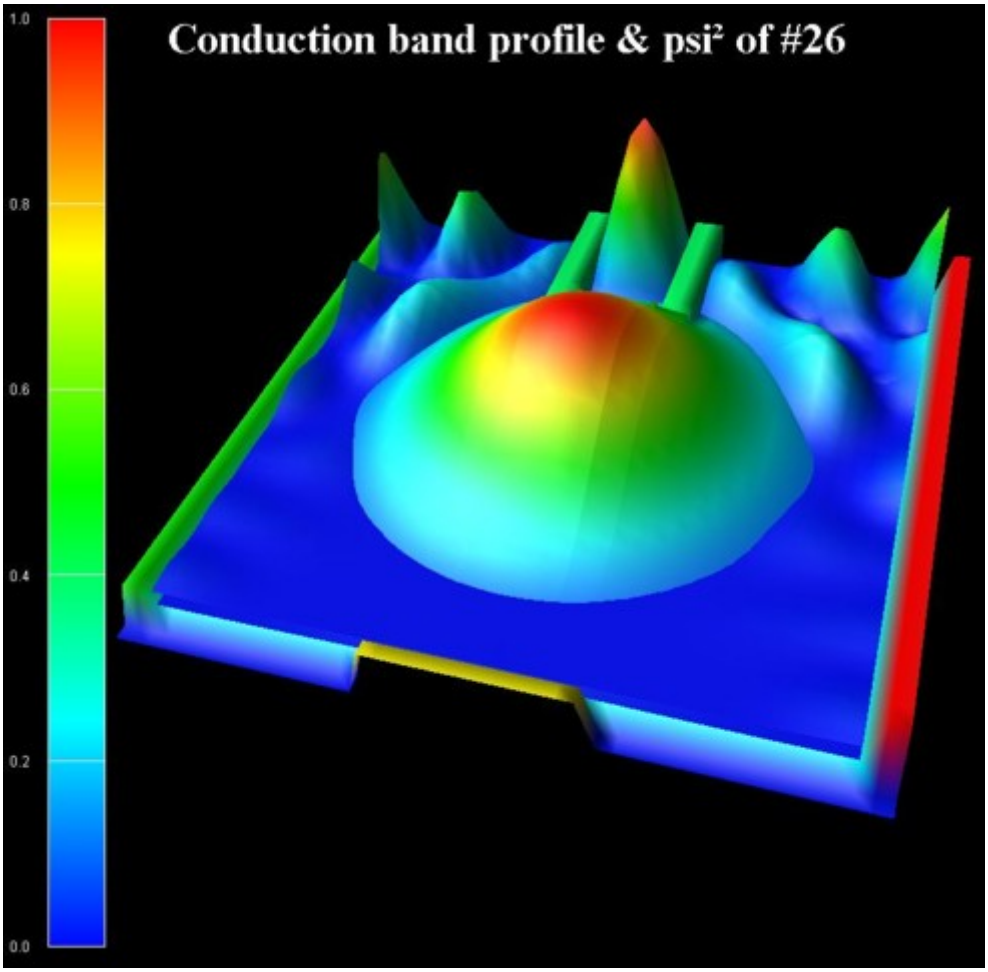


Figure 7.4.3.2: The *nextnanomat* screenshot (nextnanomat has been developed by Jörg Ehehalt)

The eigenstates # 26 and # 29 are resonance states of the double barrier.

- **1st resonance:**
  - # 26: 0.182 eV (delocalized)
  - # 29: 0.196 eV (more localized)
- **2nd resonance:**
  - # 55: 0.322 eV (delocalized)
  - # 57: 0.333 eV (more localized)
  - # 60: 0.345 eV (delocalized)
  - # 63: 0.359 eV (more localized)

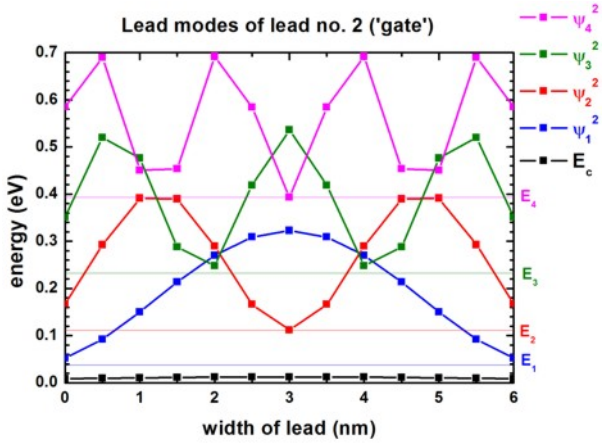
The following figure shows the conduction band profile together with the square of the wave function of the 26th eigenstate. One can clearly see that it is a resonance state of the double barrier and corresponds to the second peak in the blue transmission curve T_{13} from source to drain around 180 meV.

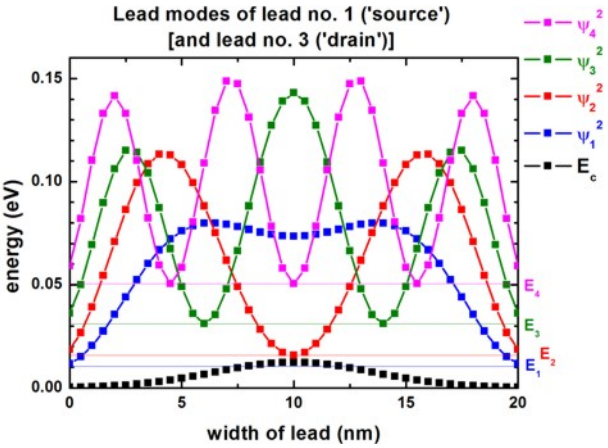

Conduction band profile & psi² of #26
1.0
0.8
0.6
0.4
0.2
0.0

Lead modes

The following two figures show 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 transmission shows a step-like behavior which is related to the energies of lead no. 2 ('gate').

The lead modes (eigenvalues, psi, psi^2, band edge profile, ...) can be found in these files:
CBR_data1/modes_lead00*_sg001_*.dat





Technical details

Definition of contacts

For each contact (lead), a quantum cluster ("lead quantum cluster") has to be defined because in each lead, a one-dimensional Schrödinger equation has to be solved which gives us the lead modes (i.e. energies and eigenvectors of the leads). In addition, a quantum cluster is required for the device itself ("main quantum cluster").

```
!-----------------------------------------------!
$quantum-regions                                !

region-number    = 1                          ! 'device'
base-geometry    = rectangle                  !
region-priority = 2                           !
x-coordinates    = 0d0 20d0                   ! [nm]  width of 'device' = 20 nm
y-coordinates    = 0d0 20d0                   ! [nm]  width of 'device' = 20 nm
```

(continues on next page)

(continued from previous page)

```
region-number    = 2                                    ! 'source'
base-geometry    = rectangle                            !
region-priority  = 1                                    !
x-coordinates    = -1d0 -0.5d0                          ! [nm]  (including 2 grid points␣
↪along this direction)
y-coordinates    =  0d0 20d0                            ! [nm]  length of 'lead 1' = 20 nm

region-number    = 3                                    ! 'gate'
base-geometry    = rectangle                            !
region-priority  = 1                                    !
x-coordinates    =  7d0 13d0                            ! [nm]  length of 'lead 2' =  6 nm
y-coordinates    = -1d0 -0.5d0                          ! [nm]  (including 2 grid points␣
↪along this direction)

region-number    = 4                                    ! 'drain'
base-geometry    = rectangle                            !
region-priority  = 1                                    !
x-coordinates    = 20.5d0 21d0                          ! [nm]  (including 2 grid points␣
↪along this direction)
y-coordinates    = 0d0    20d0                          ! [nm]  length of 'lead 3' = 20 nm

$end_quantum-regions                                     !
!-------------------------------------------------------!
```

For each quantum cluster, the number of eigenstates to be calculated and its boundary conditions have to be specified.

For the main quantum cluster it holds: For each grid point in the main quantum cluster it is checked if it is at the boundary and if it is in contact to a lead. 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-model-electrons                           !
model-number                    = 1                !
model-name                      = effective-mass   ! single band effective-mass␣
↪Schrödinger equation
cluster-numbers                 = 1
conduction-band-numbers         = 1                ! Gamma band
!number-of-eigenvalues-per-band = 118               ! corresponds to   7.0 % of 1681
number-of-eigenvalues-per-band  = 303              ! corresponds to  18.0 % of 1681
!number-of-eigenvalues-per-band = 1681              ! corresponds to 100.0 % of 1681
quantization-along-axes         = 1 1 0            ! (x,y)-plane
boundary-condition-100          = Neumann          ! Neumann along propagation direction
boundary-condition-010          = Neumann          ! Neumann along propagation direction

lead 1 = source: number of modes per lead = 41 =  maximum number of relevant quantum␣
↪grid points in lead 1
!-------------------------------------------------!
model-number                    = 2                !
...
cluster-numbers                 = 2                !
number-of-eigenvalues-per-band  = 41               ! calculate 41 lead modes
boundary-condition-010          = Dirichlet        ! Dirichlet perpendicular to␣
↪propagation direction

!-------------------------------------------------!
! lead 2 = gate:   number of modes per lead = 13 = maximum number of relevant quantum␣
```

(continues on next page)

(continued from previous page)

```
↪grid points in lead 2
!-------------------------------------------------!
model-number                    = 3               !
...
cluster-numbers                 = 2               !
number-of-eigenvalues-per-band = 13               ! calculate 13 lead modes
boundary-condition-100          = Dirichlet       ! Dirichlet perpendicular to␣
↪propagation direction

!-------------------------------------------------!
! lead 3 = drain:  number of modes per lead = 41 = maximum number of relevant quantum␣
↪grid points in lead 3
!-------------------------------------------------!
model-number                    = 4               !
...
cluster-numbers                 = 2               !
number-of-eigenvalues-per-band = 41               ! calculate 41 lead modes
boundary-condition-010          = Dirichlet       ! Dirichlet perpendicular to␣
↪propagation direction

$end_quantum-model-electrons                       !
!-------------------------------------------------!
```

```
!-------------------------------------------------!
$CBR-current                                       !
                                                 !
destination-directory = CBR_data1/                ! directory for output and data files
calculate-CBR         = yes                       ! flag: "yes"/"no"
                                                 !
main-qr-num           = 1                         ! number of main quantum cluster for␣
↪which transport is calculated: 'device'
num-leads             = 3                         ! total number of leads attached to␣
↪main region
lead-qr-numbers       = 2  3  4                   ! quantum cluster numbers␣
↪corresponding to each lead
propagation-direction = 1  2  1 ! '1'=x, '2'=y    ! direction of propagation (1,2,3)␣
↪for  each lead
num-modes-per-lead    = 41 13 41                  ! number of modes per lead used for␣
↪CBR calculation
                                                 ! must be <= number of eigenvalues␣
↪specified in corresponding quantum model
!num-eigenvectors-used = 118                       ! corresponds to   7.0 % of 1681
num-eigenvectors-used = 303                       ! corresponds to 18.0 % of 1681
!num-eigenvectors-used = 1681 ! 1681=41*41=N_x*N_y! number of eigenvectors in main␣
↪quantum cluster used for CBR calculation
E-min                 = 0.0d0                     ! [eV] lower boundary for␣
↪transmission energy interval
E-max                 = 0.5d0                     ! [eV] upper boundary for␣
↪transmission energy interval
num-energy-steps      = 100                       ! number of energy steps in T(E)
                                                 !
$end_CBR-current                                   !
!-------------------------------------------------!
```

For each energy E (num-energy-steps = 100) where the transmission coefficient T(E) has to be calculated, a matrix of size 95 x 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 was about 30 seconds for 303 eigenstates.

For further information, please study this section: $CBR-current .

Transmission through a 3D nanowire (3D example)

In this tutorial we apply the Contact Block Reduction (CBR) method to a simple GaAs nanowire of cuboidal shape, using the input files

- `transmission-nanowire_GaAs_3D_nnp.in`

the latter of which is for holes instead of electrons.

The corresponding tutorial for *nextnano++* can be found *here*.

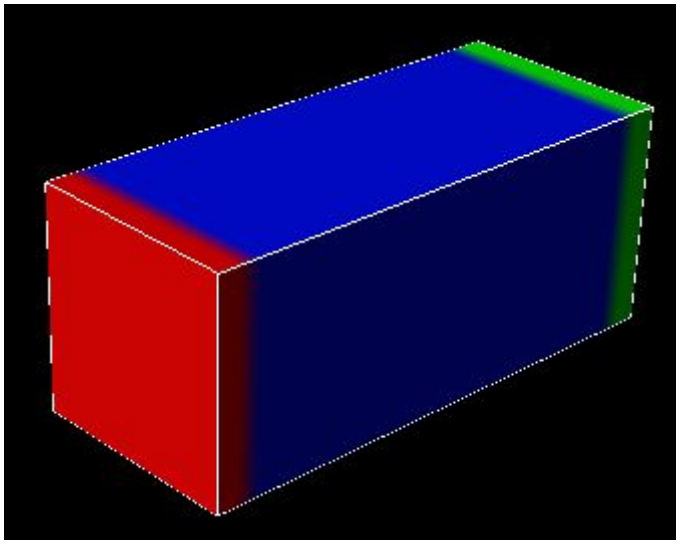
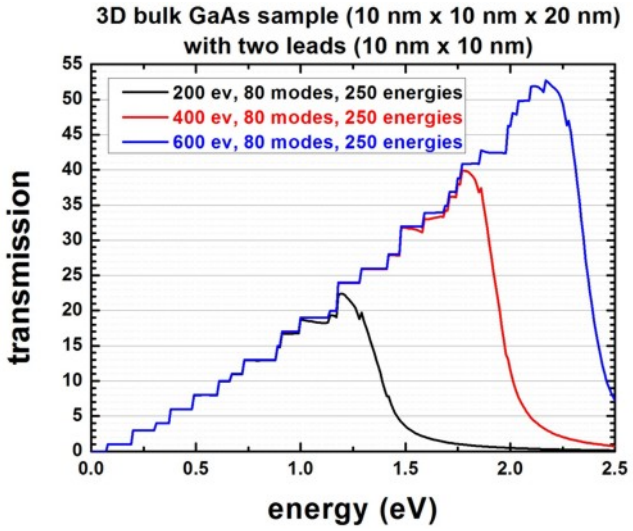


Figure 7.4.3.3: Schematic sketch of the 3D device showing the GaAs region that is placed between two contacts (red and green leads)

- The device consists of two leads of 10 nm x 10 nm each. Each lead has a total of 121 grid points (11 x 11 grid points).
- The leads are described as quantum regions, i.e. in each lead (quantum region) a two-dimensional Schrödinger equation has to be solved to obtain the eigenenergies and wave functions of the lead modes.
- The device dimensions are 10 nm x 10 nm x 20 nm.
- 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.
- The device region consists of 11 x 11 x 21 = 2541 grid points, which is equivalent to a grid spacing of 1.0 nm. This means that the device Hamiltonian is a matrix of size 2541 x 2541.
- The conduction band profile is constant and set to E_c = 0 eV.

The following figure shows the calculated transmission coefficient as a function of energy between the leads 1 and 2.



- For the blue lines 23.6 % (600 of 2541) of all eigenvectors were used whereas for the red lines only 15.7 % (400 of 2541) had to be calculated, i.e. one does not have to calculate all eigenvalues of the device Hamiltonian which grossly reduces CPU time.
- For the black lines 7.9 % (200 of 2541) of all eigenvectors were used.

A small percentage of eigenvalues suffices for T(E) in relevant energy range of interest. Note that the transmission drops significantly once the cutoff energy of the highest eigenvector taken into account is reached.

The transmission coefficient can be found in this file:

`CBR_data1/transmission3D_cb_sg001_CBR.dat`

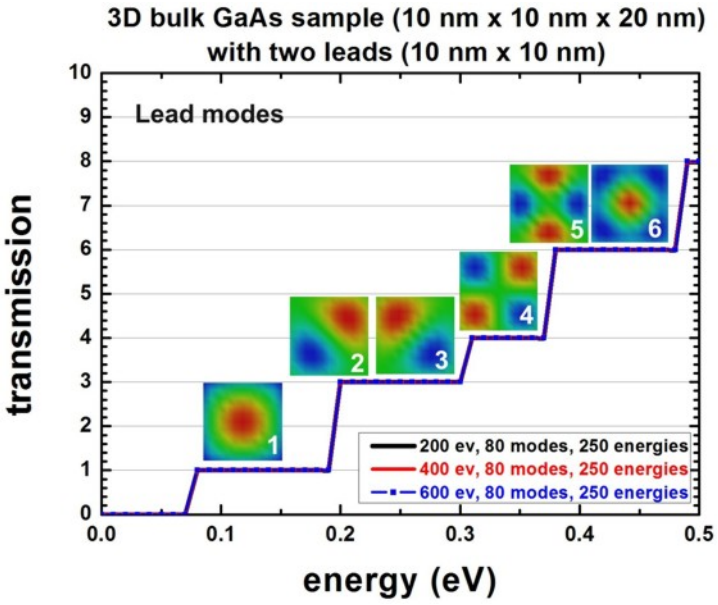
In the following, the same data as above is shown again as a zoom into the energy range 0 eV - 0.5 eV. The colored figures show the wave function amplitude of the lowest energy lead modes.

They can be found in these files:

- `CBR_data1/2Dmodes_lead001_sg001_psi_ev001.dat`
- `...`
- `CBR_data1/2Dmodes_lead001_sg001_psi_ev080.dat`
- `CBR_data1/2Dmodes_lead002_sg001_psi_ev001.dat`
- `...`
- `CBR_data1/2Dmodes_lead002_sg001_psi_ev080.dat`

- Once the energy reaches 78 meV, the first lead mode energy is reached and then this mode transmits perfectly, giving a transmission of 1.
- The second and third lead mode states are degenerate due to the symmetry of the lead cross-section, thus they have the same energy (191 meV). Consequently, once the energy of 191 meV is reached, the transmission increases by 2.
- The total transmission is now equal to 3 as all lead modes transmit perfectly.
- The energy of the 4th lead mode is at 305 meV.
- The degeneracy of the 5th and 6th mode is accidental. They have the same energy.

As one can clearly see, in this low energy limit, it is sufficient to calculate only a few percent of all eigenfunctions of the device Hamiltonian. For the leads, in all cases 80 eigenstates have been calculated.

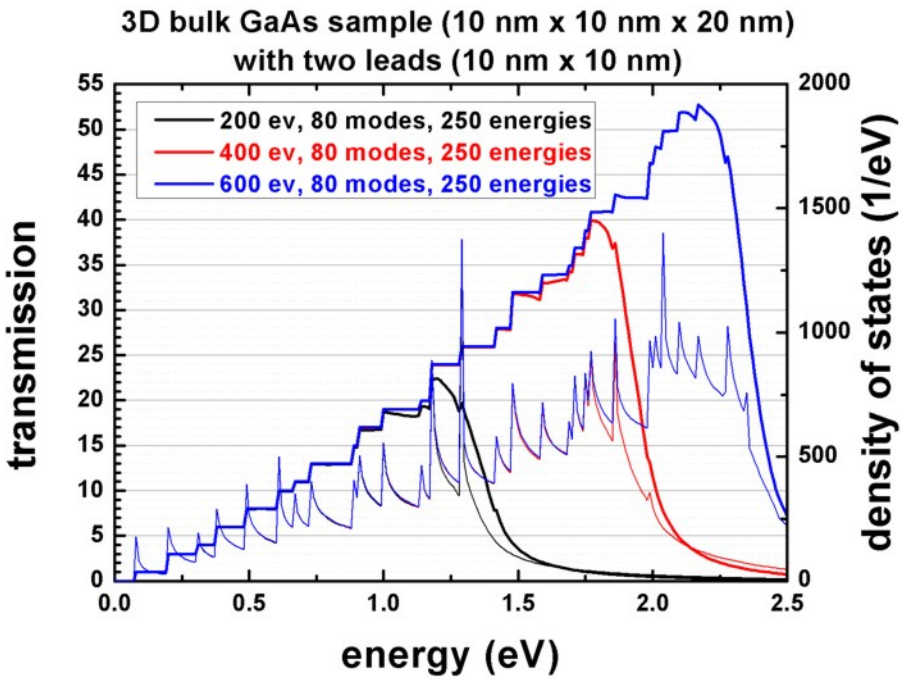


Density of states

Using

```
calculate-CBR-DOS = yes  ! flag: yes / no
```

the CPU time increases but information like the density of states (DOS) can be obtained. This is shown in the next figure where the same transmission data as above is plotted but this time including the density of states. Again, the colors indicate taking into account 200, 400 or 600 eigenvectors of the decoupled system (closed system).



Technical details

Definition of contacts

For each contact (lead), a quantum cluster ("lead quantum cluster") has to be defined because in each lead, a two-dimensional Schrödinger equation has to be solved which gives us the lead modes (i.e. energies and eigenvectors of the leads). In addition, a quantum cluster is required for the device itself ("main quantum cluster").

```
!-----------------------------------------------------!
$quantum-regions                                      !

region-number   = 1                                   ! 'device'
base-geometry   = cuboid                              !
region-priority = 1                                   !
x-coordinates   = -5d0  5d0                           ! [nm]  width of 'device' along x␣
↪= 10 nm
y-coordinates   = -5d0  5d0                           ! [nm]  width of 'device' along y␣
↪= 10 nm
z-coordinates   =  0d0 20d0                           ! [nm]      device length along z␣
↪= 20 nm

region-number   = 2                                   ! 'source' (lead 1)
base-geometry   = cuboid                              !
region-priority = 2                                   !
x-coordinates   = -5d0  5d0                           ! [nm]  width of 'lead 1' along x␣
↪= 10 nm
y-coordinates   = -5d0  5d0                           ! [nm]  width of 'lead 1' along y␣
↪= 10 nm
z-coordinates   = -2d0 -1d0                           ! [nm]  (including 2 grid points␣
↪along this direction)

region-number   = 3                                   ! 'gate'   (lead 2)
base-geometry   = cuboid                              !
region-priority = 2                                   !
x-coordinates   = -5d0  5d0                           ! [nm]  width of 'lead 1' along x␣
↪= 10 nm
y-coordinates   = -5d0  5d0                           ! [nm]  width of 'lead 1' along y␣
↪= 10 nm
z-coordinates   = 21d0 22d0                           ! [nm]  (including 2 grid points␣
↪along this direction)

!-----------------------------------------------------!
```

For each quantum cluster, the number of eigenstates to be calculated.

```
!-------------------------------------------------!
$quantum-model-electrons                          !
...
cluster-numbers                 = 1
...
number-of-eigenvalues-per-band = 200             ! corresponds to 7.87 % of 2541
!number-of-eigenvalues-per-band = 400             ! corresponds to 15.74 % of 2541
!number-of-eigenvalues-per-band = 600             ! corresponds to 23.61 % of 2541



!-------------------------------------------------!
```

(continues on next page)

(continued from previous page)

```
! lead 1 = source: number of modes per lead = 80   maximum number of relevant quantum␣
↪grid points in lead 1 = 121
!-------------------------------------------------!
...
cluster-numbers                = 2                !
number-of-eigenvalues-per-band = 80               ! calculate 80 lead modes,␣
↪corresponds to 66.1 % of 121


!-------------------------------------------------!
! lead 2 = source: number of modes per lead = 80   maximum number of relevant quantum␣
↪grid points in lead 1 = 121
!-------------------------------------------------!
...
cluster-numbers                = 3                !
number-of-eigenvalues-per-band = 80               ! calculate 80 lead modes,␣
↪corresponds to 66.1 % of 121

$end_quantum-model-electrons                       !
!-------------------------------------------------!
```

```
!-------------------------------------------------!
$CBR-current                                       !
                                                  !
destination-directory = CBR_data1/                 ! directory for output and data files
calculate-CBR         = yes                        ! flag: "yes"/"no"
                                                  !
main-qr-num           = 1                          ! number of main quantum cluster for␣
↪which transport is calculated: 'device'
num-leads             = 2                          ! total number of leads attached to␣
↪main region
lead-qr-numbers       = 2  3                       ! quantum cluster numbers␣
↪corresponding to each lead
propagation-direction = 3  3    ! '3'=z,           ! direction of propagation (1,2,3)␣
↪for  each lead
num-modes-per-lead    = 80 80                      ! number of modes per lead used for␣
↪CBR calculation
                                                  ! must be <= number of eigenvalues␣
↪specified in corresponding quantum model
!num-eigenvectors-used = 200                       ! corresponds to 7.87 % of 2541
!num-eigenvectors-used = 400                       ! corresponds to 15.74 % of 2541
num-eigenvectors-used = 600                       ! corresponds to 23.61 % of 2541
!num-eigenvectors-used = 2541 ! 2541=11*11*21=N_x*N_y*N_y! number of eigenvectors in␣
↪main quantum cluster used for CBR calculation
E-min                 = 0.0d0                      ! [eV] lower boundary for␣
↪transmission energy interval
E-max                 = 2.5d0                      ! [eV] upper boundary for␣
↪transmission energy interval
num-energy-steps      = 250                        ! number of energy steps in T(E)
                                                  !
$end_CBR-current                                   !
!-------------------------------------------------!
```

Loop over energy (parallelization)

For each energy E (num-energy-steps = 250) where the transmission coefficient T(E) has to be calculated, a matrix of size 160 x 160 has to be inverted.

The size of 160 is determined by the sum over the number of lead modes taken into account for each lead.

The upper limit would be the number of grid points in each lead that are in contact to the device, i.e. in this example where each lead has 11 x 11 = 121 grid points, the maximum size of the matrix to be inverted could be 242 = 121 + 121.

- Lead 1 (Source): 121 grid points (80 lead modes taken into account)
- Lead 2 (Drain): 121 grid points (80 lead modes taken into account)
  - in total: 242 grid points (but only 80 modes are taken into account for each lead: 160 = 80 + 80)
  - The total CPU time for calculation of the transmission T(E) in this example was less than a minute for 200 eigenstates, 80 lead modes in each lead, and 250 energy steps.

This loop can be executed in parallel to improve computational performance.

```
$global-settings
...
!number-of-parallel-threads   = 2  ! (for dual-core CPU)
number-of-parallel-threads    = 4  ! (for quad-core CPU)
```

For further information, please study this section: $CBR-current .

1D - Quantum Tunneling: Comparison of CBR- and WKB approaches with the exact answer

Input Files:
1D_transmission_GaAs_analytic_nn3.in

Scope of the tutorial:

- General information about tunneling
- Comparison of the exact expression for the tunneling probability with that calculated by using CBR-based simulations and the semiclassical (WKB) method *[Liang]*

Main adjustable parameters:

- upper boundary for transmission energy - `%E_max`
- analytical expression for the potential barrier - `potential-from-function`
- the effective mass of the electron - `%effective_mass`

Relevant output Files:

- *Results\BandEdges.dat* (energy profile)
- *Results\Transmission_cb_sg1_deg1.dat* (transmission)

Note: This tutorial is in progrees and will be finalized soon.

Tunneling of particles through a potential barrier is one of the most well-known quantum phenomenon which does not exist in a classical world. Understanding the quantum tunneling and tunneling currents is very important for engineering and fabricating various elements of electronics.

As any quantum process, tunneling is described by the correspondent probability, $\mathcal{T}$, which depends on the energy of a propagating electron. There are only few potential shapes for which the analyhtical (at least approximate) expression for $\mathcal{T}$ is known. Generically, one has to use either approximate or numercial methods to calculate $\mathcal{T}$.

In this tutorial, we discuss the tunneling probability through 1D potential defined as:

$$V(x) = V_0/\cosh^2(a(x - x_0)). \tag{7.4.3.1}$$

Here V_0 is the height of the potential barrier, and α^{-1} describes the width of the barrier, see the example in Figure 7.4.3.4. Eq. (7.4.3.1) belongs to these rare cases where the exact answer for $\mathcal{T}$ is known *[Landau Lifshitz]*, *section §123 The general theory of scattering*:

$$\begin{aligned}
\mathcal{T}(E) &= \frac{\sinh^2(\pi k(E)/\alpha)}{\sinh^2(\pi k(E)/\alpha) + \cos^2(\pi/2\sqrt{1-\mathcal{U}})}, \ \mathcal{U} < \infty; \\
\mathcal{T}(E) &= \frac{\sinh^2(\pi k(E)/\alpha)}{\sinh^2(\pi k(E)/\alpha) + \cosh^2(\pi/2\sqrt{\mathcal{U}-\infty})}, \ \mathcal{U} > \infty; \\
\mathcal{U} &= \frac{8mV_0}{\hbar^2\alpha^2}, \ k(E) = \sqrt{2mV_0}/\hbar;
\end{aligned} \tag{7.4.3.2}$$

where $E, k,$ and m are the electron energy, wave vector, and effective mass, respectively; $\hbar$ is the Planck constant.

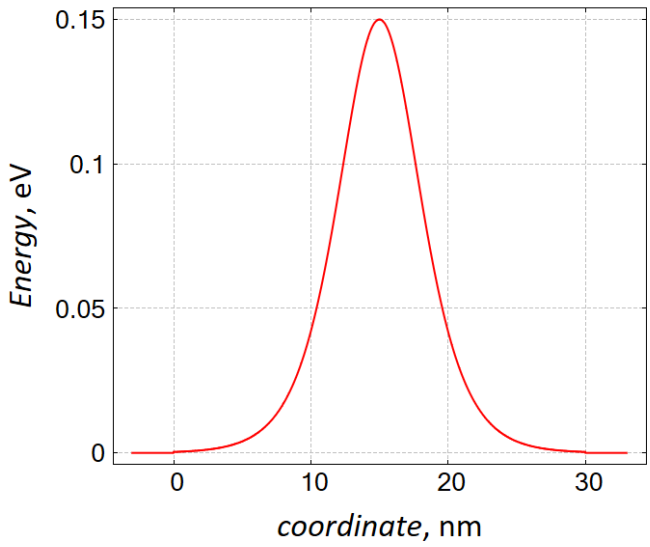


Figure 7.4.3.4: Potential $V(x)$, Eq. (7.4.3.1), for which the exact answer for the transmission probability is known, with $V_0 = 0.15$ eV, $x_0 = 15$ nm, and $\alpha^{-1} = 4$ nm. For the consistency with the usual scattering approach *[Landau Lifshitz]*, *section §123 The general theory of scattering*, the potential is finite only in the central region (device), $0 \leq x \leq 30$ nm and is set to 0 outside of this region. The energy origin (i.e. the bottom of the conduction band) is fixed: $E_c = 0$.

The comparison of the transmission, which is obtained numerically (CBR), semi-analytically (WKB) and analytically, Eq. (7.4.3.2), is shown in Figure 7.4.3.5. The integration over the coordiante (between two turning points) in the WKB answer

$$\mathcal{T}_{\mathrm{WKB}}(E) \simeq \exp\left(-2\int_{x_1}^{x_2} k(x)\mathrm{d}x\right) \tag{7.4.3.3}$$

has been performed numerically.

One cas see that CBR is able to reproduce the exact answer with a high accuracy. Clearly, the WKB approach can be used to claculate transmission only in the semiclassical regime where transmission is small.

Exercises:

- Check how the agreement between the exact answer, Eq. (7.4.3.2), and the numerical results of the CBR approach depends on the number of the (device) eigenstates which are taken into account.
- Using the sample file of this tutorial, calculate transmission of an electron through the parabolically shaped potential which excists only inside the devie. Compare CBR results with approximate WKB answer. Explain your observations.

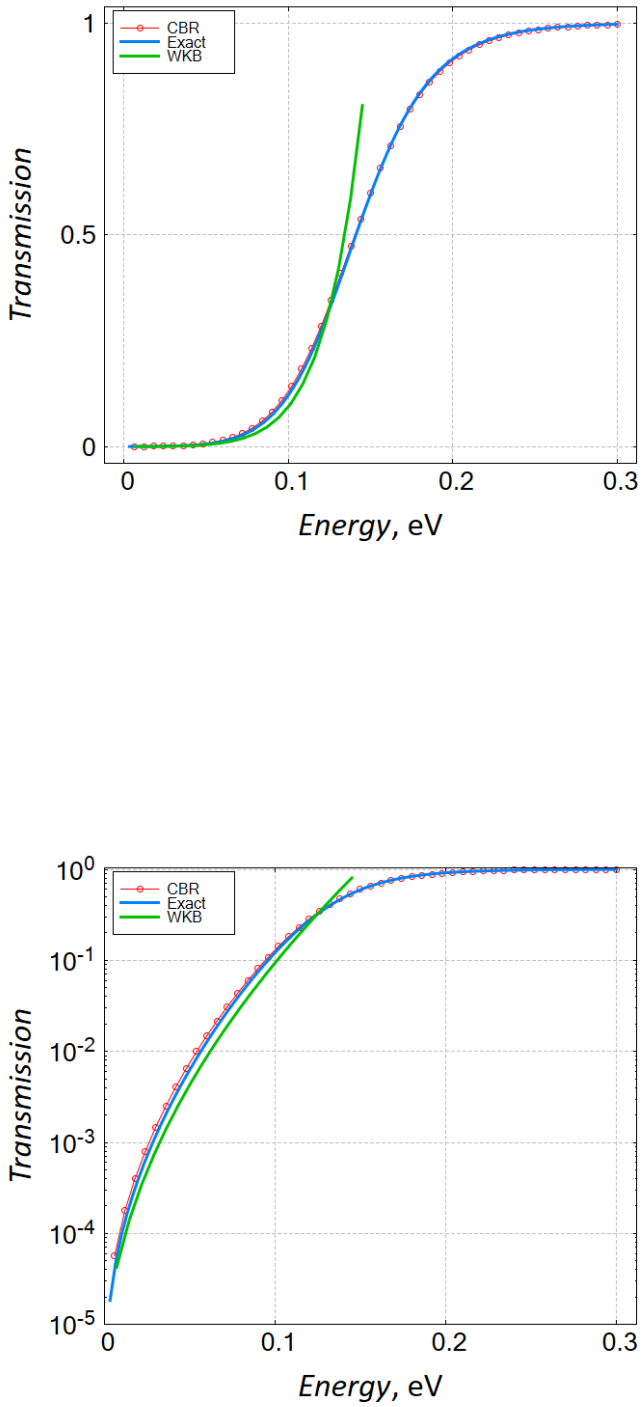


Figure 7.4.3.5: Energy dependence of transmission through the potential shown on Figure 7.4.3.4. The upper (linear scales) and lower (linear-log scales) panels shown answers which are obtained with the help of the CBR (red) and WKB (green) methods. The blue curve is the exact answer, Eq. (7.4.3.2). We have used in the input file the effective mass of the electron in GaAs. Note that finite temperature is irrelevant for these simulations.

1D - Quantum Tunneling and Fowler-Nordheim theory

Input Files:
1D_quantum_tunneling_GaAs_nn3.in

Scope of the tutorial:

- General information about tunneling and tunneling current
- Comparison of the Fowler-Nordheim theory and the CBR-based simulations

Main adjustable parameters:

- upper boundary for transmission energy - `%E_max`
- the barrier widths - `%Delta_x = %Barrier_max - %Barrier_min`
- the barrier heights - `%Barrier_Height`
- the electric field - `%ELECTRIC_FIELD`
- 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`
- dimensionality of the interface which separates leads and connecting wires - `%DIM_LEAD`

Relevant output Files:

- *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)

Tunneling of particles through a potential barrier is one of the most well-known quantum phenomenon which does not exist in a classical world. Understanding the quantum tunneling and tunneling currents is very important for engineering and fabricating various elements of electronics and nanodevices, including qubits. There are several seminal theories which describe these effects in various setups, for example, *Tsu-Esaki theory of tunneling in quantum superlattices [TsuEsaki]* and *Fowler-Nordheim theory for field electron emission [FowlerNordheim]* with *applications to semiconductor structures [LenzlingerSnow]*. This tutorial addresses the Fowler-Nordheim tunneling current in a semiconductor multilayer structure, see Figure 7.4.3.6.

If one considers the tunneling current along one axis and the semiconductor structure is assumed to be homogeneous along the other two directions, simulations can be reduced to effectively one-dimensional Landauer-like setup, see the upper panel of Figure 7.4.3.7, where the device is connected to two leads with different chemical potentials. The current flows from the material with a larger chemical potential to that with a smaller one.

Following the standard approach, we have introduced wires which connect the device and the leads. These wires are optional in the current tutorial. The leads correspond to the left and right materials in Figure 7.4.3.6 and, therefore, are three-dimensional semiconductors. Their chemical potentials are different if the left and right materials are different (e.g., differently doped semiconductors). Alternatively, the chemical potentials are shifted by an applied DC voltage, $\mu_L - \mu_R = eV_{dc}$, with $\mu_{L,R}$ being measured from the bottom of the conduction-band, $E_c = 0$.

> **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]*.

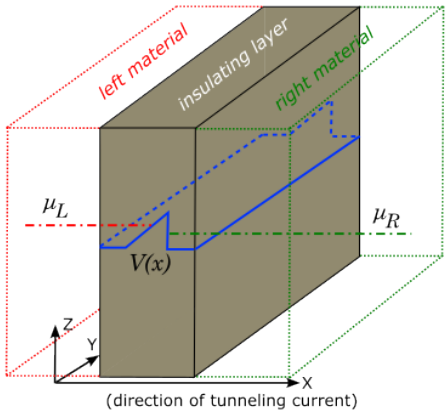


Figure 7.4.3.6: Sketch of a setup for the tunneling current in the Fowler-Nordheim theory. Left (red lines) and right (green lines) materials with different chemical potentials, $\mu_{L,R}$, are separated by an insulating layer (brown region) which makes a potential barrier (blue lines). Everything is homogeneous in {y, z}-directions. The tunneling through the potential barrier occurs in x-direction.

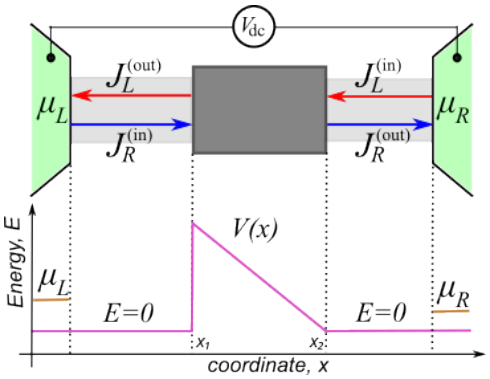


Figure 7.4.3.7: Upper panel: Landauer-like setup. Left and right leads (green regions) are connected to a semiconductor device (dark gray square) via connecting wires (light gray regions). The leads correspond to the left and right materials in Figure 7.4.3.6, hence, they are three-dimensional. Lower panel: chemical potentials of the leads (orange lines) and the energy of the potential barrier (magenta lines).

The bias V_{dc} results in the electric current through the device. For simplicity, the connecting wires and the device are treated as purely one-dimensional. The wires are ideal conductors. The device contains a triangular potential barrier, the lower panel of Figure 7.4.3.7, with $V(x_1) = V_{max}, V(x_2) = 0$. Such a shape of the potential energy can be caused by an electric field, F: $V(x) = V_{max} - eF(x - x_1)$, with $x_1 \leq x \leq x_2$ and $F = V_{max}/e(x_2 - x_1)$; e denotes the electron charge. If $\mu_{L,R} < V_{max}$, classical transport of the electrons is impossible and one comes across purely quantum current supported by tunneling of the electrons.

The total current is the difference of currents flowing in opposite directions: $J = J_R^{(\text{in})} - J_L^{(\text{out})} = J_R^{(\text{out})} - J_L^{(in)}$. Here, upper indices indicate whether a given current flows into or from the device. The Landauer formula allows one to express J via the transmission probability of tunneling through the barrier, $\mathcal{T}$:

$$J = 2e \int \frac{dk_x}{2\pi} v_x(k_x) \mathcal{T}(k_x)(n_L(k_x) - n_R(k_x)), \tag{7.4.3.4}$$

where $n_{L/R}(k_x) = \int \frac{dk_{y,z}}{(2\pi)^2} f_{L/R}(k_{x,y,z})$ are the electron densities at interfaces between left/right material and the insulating layer; v_x is the electron velocity along x-axis. The electrons in the left/right materials (i.e., in the leads) are described by the Fermi-Dirac distribution functions, $f_{L,R}$, The transmission probability in our simplified model depends only on the longitudinal wave-vector of the electron, k_x, and is independent on the direction of tunneling (either from the left lead to the right one or vice versa). We assume further that there is no temperature gradient and the temperature, T, is the same in all parts of the circuit. We note that, in the setup shown in Figure 7.4.3.6, J is the current per unit transverse cross-section.

$\mathcal{T}$ can be found analytically for the one-dimensional triangular barrier, see, e.g., *secton in the book by Shi-Dong Liang [Shi-Dong-Liang]*. In a general case, one can use either approximate analytical methods, like WKB, or calculate the transmission probability numerically by solving the Schrödinger equation. We have used *Contact Block Reduction method [CBR]* for numerical simulations, see also tutorial. Results of the CBR based simulations are shown in Figure 7.4.3.8.

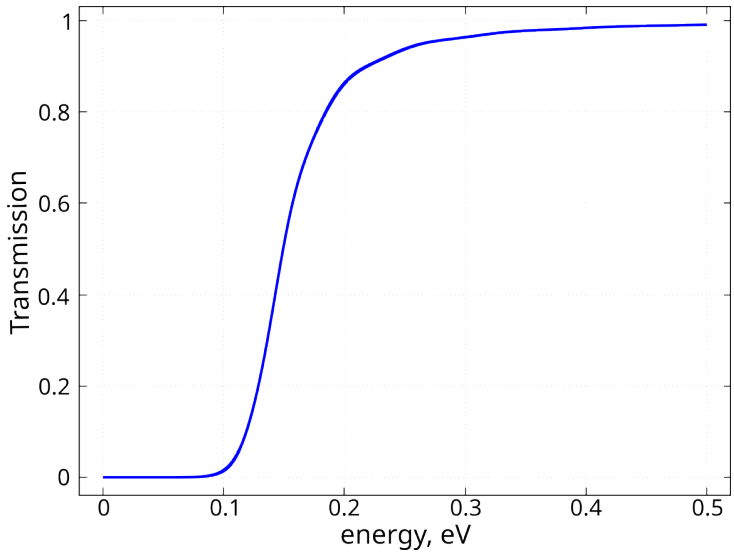


Figure 7.4.3.8: Energy dependence of the transmission probability through a triangular barrier, see Figure 7.4.3.7, with parameters $V_{max} = 0.161$ eV, $x_2 - x_1 = 30$ nm. Simulations have been done for the conductors made from GaAs and the potential barrier from $\text{Al}_x\text{In}_{1-x}\text{As}$.

An additional information can be obtained from calculations of the local density of states, see Figure 7.4.3.9 which illustrates, how lead modes enter the device when only one lead is connected. The modes with higher energies have a larger penetration depth.

After integration (7.4.3.4) over the electron momentum (or, equivalently, over the electron energy) one obtains the tunneling current. One example of its IV-characteristics at zero temperature is shown in Figure 7.4.3.10. In the Fowler-Nordheim theory, one usually assumes that the drain (the right lead in the above setup) is the vacuum. It does not contain conduction electrons, i.e., this corresponds to the limit $\mu_R \to 0$. This theory predicts the following tunneling current in isotropic materials: $J_{FN} \simeq \frac{A_{\text{FN}}}{\phi} \exp\left(-B_{\text{FN}}\phi^{3/2}\right)$. Here $\phi = V_{max} - \mu_L$, $A_{\text{FN}} = \frac{e^3}{16\pi^2\hbar} F^2$,

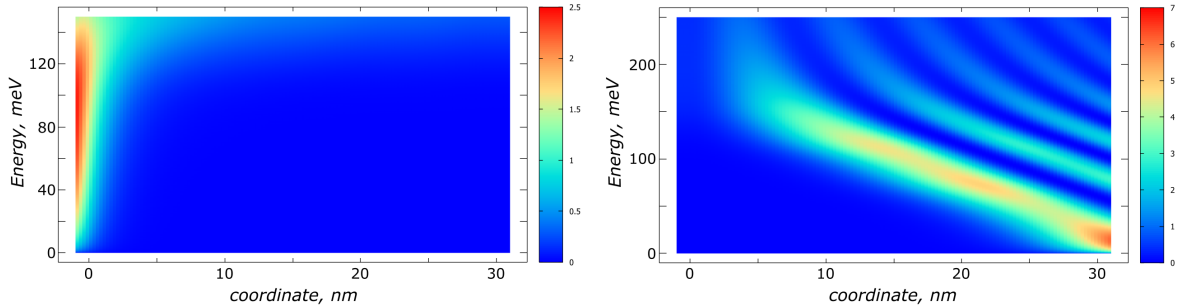


Figure 7.4.3.9: Local Density of states which illustrates how lead modes enter the device when only one lead (the left/right lead in the left/right panel) is connected. Mind the different scale of the color scheme in the left and right panels.

$B_{\mathrm{FN}} = -4\sqrt{2m^*}/3\hbar eF$; $\hbar$ and m^* are the Planck constant and the effective mass of the electron in a given material, respectively.

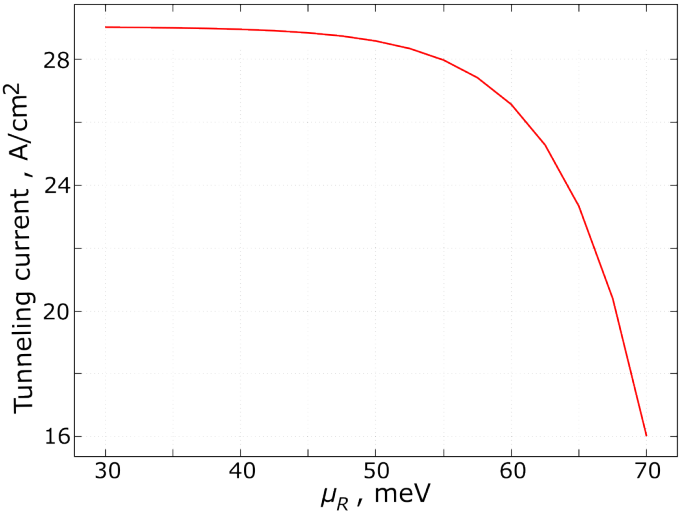


Figure 7.4.3.10: Dependence of the tunneling current on the chemical potential μ_R at $T = 0$ K at fixed value of $\mu_L = 75$ meV. The DC bias is $V_{dc} = \mu_L - \mu_R$. The length of the connecting wires has been chosen as 1 nm. Other parameters are the same as in Figure 7.4.3.8.

The disagreement between the constant B_{FN} and its numerical counterpart, which can be obtained from the fitting of the numerically obtained IV-characteristics, Figure 7.4.3.10, by the Fowler-Nordheim-like equation, is very small; it is close to 2% with the parameters used in this tutorial. However, the constant A_{FN} substantially underestimates its numerically obtained value. This is related to the approximate calculations of the transmission in the Fowler-Nordheim theory and to the difference between the tunneling from a semiconductor to vacuum and between two different semiconductor materials, Figure 7.4.3.6.

The finite temperature changes the energy distribution of the electrons in the leads which results in a temperature-dependent contribution to the tunneling current, see the main panel of Figure 7.4.3.11. The Fowler-Nordheim theory predicts that $J(T) \simeq J(T=0)(1+(T/T_c)^2+O\left[(T/T_c)^4\right])$ with $T_c = \sqrt{3}eF\,\hbar/2\pi k_B\sqrt{m^*\phi}$, $T/T_c \ll 1$, and k_B being the Boltzmann constant.

Fitting the numerically obtained result by the quadratic temperature dependence yields $T_c = 58.56$ K, see the inset of Figure 7.4.3.11 which reasonably agrees with the theoretical value of the Fowler-Nordheim theory $T_c^{\mathrm{FN}} \simeq 64.4$ K.

Exercise:

- Take all parameters from the example shown in Figure 7.4.3.10 but, instead of changing the

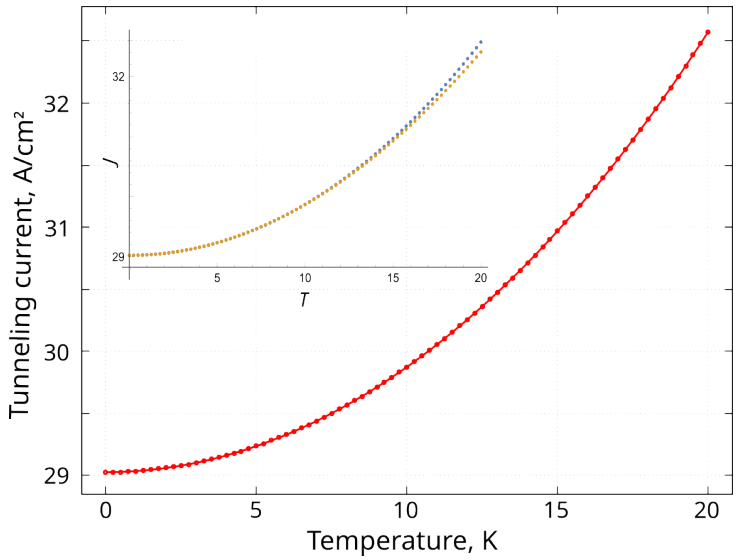


Figure 7.4.3.11: Main panel: Temperature dependence of the tunneling current at $\mu_R = 30$ meV. Other parameters are the same as in previous Figures. Inset: Comparison of numerically obtained results (blue dots) and their fitting by the quadratic temperature dependence (orange dots) with $T_c = 58.56$ K

value of μ_R, fix the applied voltage $V_{\rm dc}$ and calculate numerically the dependence of the tunneling current on the mean value of the chemical potential, $\bar{\mu} = (\mu_L + \mu_R)/2$. Explain your results.

7.4.4 Electrolyte

Poisson–Boltzmann equation: The Gouy–Chapman solution

The following *nextnano*[3] input file were used:

- `1DGouyChapman_template.in`
- `2DGouyChapman_template.in`
- `3DGouyChapman_template.in`

We solve the Poisson–Boltzmann equation for a monovalent salt, i.e. NaCl (Na^+ Cl^-). For this particular case, our numerical solution of the Poisson–Boltzmann equation can be compared to the analytical one-dimensional Gouy–Chapman solution for a monovalent and symmetric salt.

The temperature is set to 298.15 K = 25°C, the static dielectric constant of water is set to 78.

The electrolyte region (0 nm - 200 nm) contains the following ions (*$electrolyte-ion-content*):

```
!------------------------------------------------------------
! The electrolyte (NaCl) contains two types of ions:
!   1)  100 mM singly charged cations (Na+)    <- 100 mM NaCl
!   2)  100 mM singly charged anions  (Cl-)    <- 100 mM NaCl
!------------------------------------------------------------
$electrolyte-ion-content

 ion-number        = 1           ! singly charged cations
 ion-valency       =  1.0        ! charge of the ion: Na+
 ion-concentration = 100e-3      ! [M] = [mol/l] = 1d-3 [mol/cm^3]
 ion-region        = 0.0  200.0  ! electrolyte region

 ion-number        = 2           ! singly charged anions
```

(continues on next page)

(continued from previous page)

```
ion-valency       = -1.0        ! charge of the ion: Cl-
ion-concentration = 100e-3      ! [M] = [mol/l] = 1d-3 [mol/cm^3]
ion-region        = 0.0  200.0  ! electrolyte region
```

We vary the NaCl concentration (`ion-concentration`) from 0.1 mM to 1 M.

- 0.1 mM
- 1 mM
- 10 mM
- 0.1 M
- 1 M

We assume an interface charge density (*$interface-states*) between the oxide and the electrolyte of -0.2 C/m^2 = -124.83 · 10^{12} cm^{-2}.

```
$interface-states
 ...
 state-number      = 1               ! between contact / electrolyte at 0 nm
 state-type        = fixed-charge    ! sigma
 interface-density = -124.830193e12  ! -0.2 [C/m^2] = -124.830193 * 10^12␣
↪[cm^-2]
```

The pH value (*$electrolyte*) is 7, i.e. neutral.

```
$interface-states
 ...
 state-number      = 2               ! between contact / electrolyte at 0 nm
 state-type        = electrolyte
```

```
$electrolyte
 ...
 pH-value          = 7.0  ! pH = -lg(concentration) = 7 -> concentration in␣
↪[M]=[mol/l]
```

The following figure shows the electrostatic potential for different salt concentrations (0.1 mM, 1 mM, 10 mM, 0.1 M and 1 M) at a fixed surface charge of -0.2 C/m^2. The potential at the surface at 0 nm, that arises due to the fixed surface charge density that is in contact with the electrolyte, is screened by the ions in the solution and the resulting distribution of the ions depends on the value of the spatially varying electrostatic potential.

The *Debye screening lengths* (`DebyeScreeningLength.txt`) are indicated by the squares and the values are:

NaCl salt concentration	Debye screening length (nm)
1 M	0.303
0.1 M	0.959
10 mM	3.032
1 mM	9.589
0.1 mM	30.308

Note: For 0.1 mM, the concentrations of the H_3O^+ and OH^- ions slighly influence the Debye screening length (last two digits) because in the *nextnano*^{3} simulations these ions are always present and their concentrations depend on the pH value.

For a definiton of the Debye screening length, have a look here: *$electrolyte*

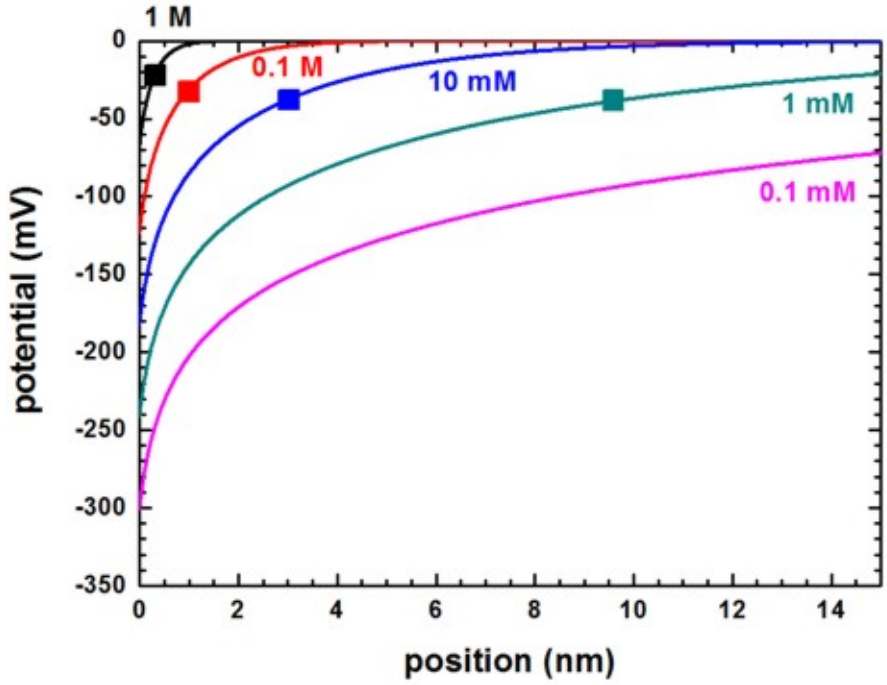


Figure 7.4.4.1: Electrostatic potential for different salt concentrations. The interface is at 0 nm.

The following figure shows the Debye screening length for a monovalent salt such as NaCl as a function of the salt concentration.

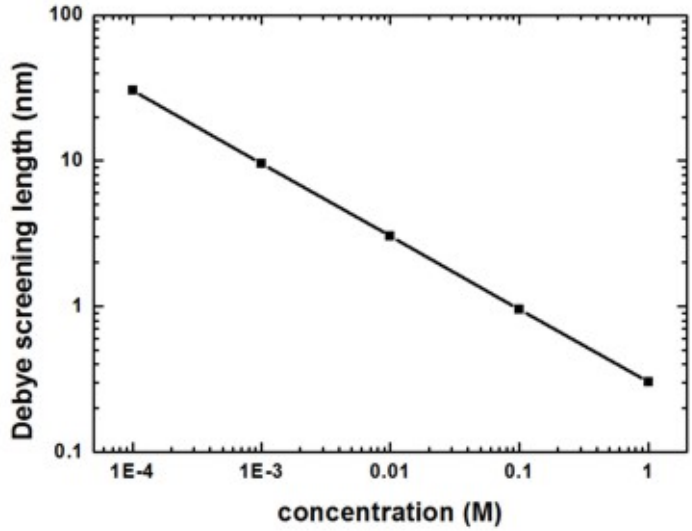


Figure 7.4.4.2: Debye screening length for NaCl as a function of salt concentration

For a monovalent salt the nominal value of the salt concentration is equal to the *ionic strength* which is a measure for the screening of charges in a solution.

The surface potential can be found in this file: `InterfacePotentialDensity_vs_pH1D.dat`

It reads for a salt concentration of 0.1 M NaCl:

```
pH value    interface potential [V]    interface density (1*10^12 [e/cm^2])
7.000000    -0.123478240580906         -124.830193000000
```

The following figure shows the ion distribution for a 0.1 M NaCl electrolyte. The multiples of the Debye screening

lengths (κ^{-1} = 0.959 nm) are indicated by the vertical lines. The negative surface charge is screened by the positive Na^+ ions whereas the negatively charged Cl^- ions are repelled from the surface. At about 5 nm both ions reach their equilibrium concentration of 0.1 M.

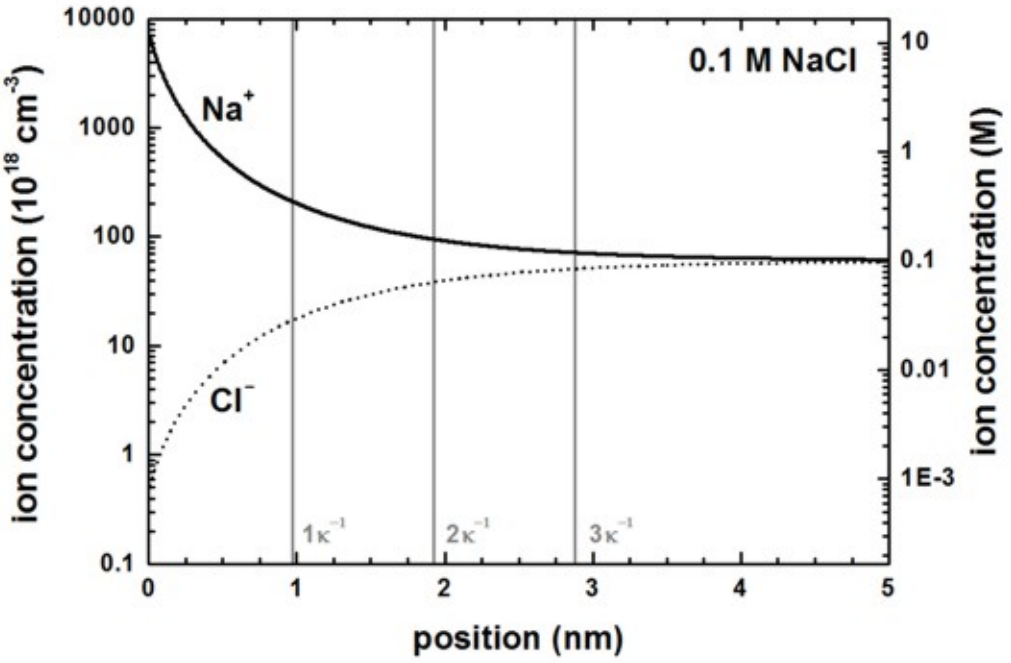


Figure 7.4.4.3: Ion concentrations as a function of distance from the surface at 0 nm

Linearization of the Gouy–Chapman model

In this approximation which is only valid for high salt concentrations or small surface charges, the surface charge and the surface potential can be related through the basic capacitor equation

$$\sigma_s = \phi_s C_{DL}$$

where C_{DL} is the capacitance per unit area of the electric double layer.

The following figure shows the surface potential at the solid/electrolyte interface as a function of interface charge for a monovalent salt such as NaCl at different salt concentrations calculated with the Poisson–Boltzmann equation (symbols). The solid lines are the solutions of the analytical Grahame equation for a monovalent salt which relates surface potential to surface charge. The dotted lines are the solutions of the Debye–Hückel approximation for a monovalent salt which relates surface potential to surface charge.

It can be clearly seen that only for high salt concentrations or small surface charges the linearization is valid.

The linearization of the Poisson–Boltzmann equation is called *Debye–Hückel approximation*. It can be switched on using:

```
! electrolyte-equation = Poisson-Boltzmann            ! Poisson-Boltzmann␣
↪equation (default)
  electrolyte-equation = Debye-Hueckel-approximation  ! Debye-Hückel␣
↪approximation
```

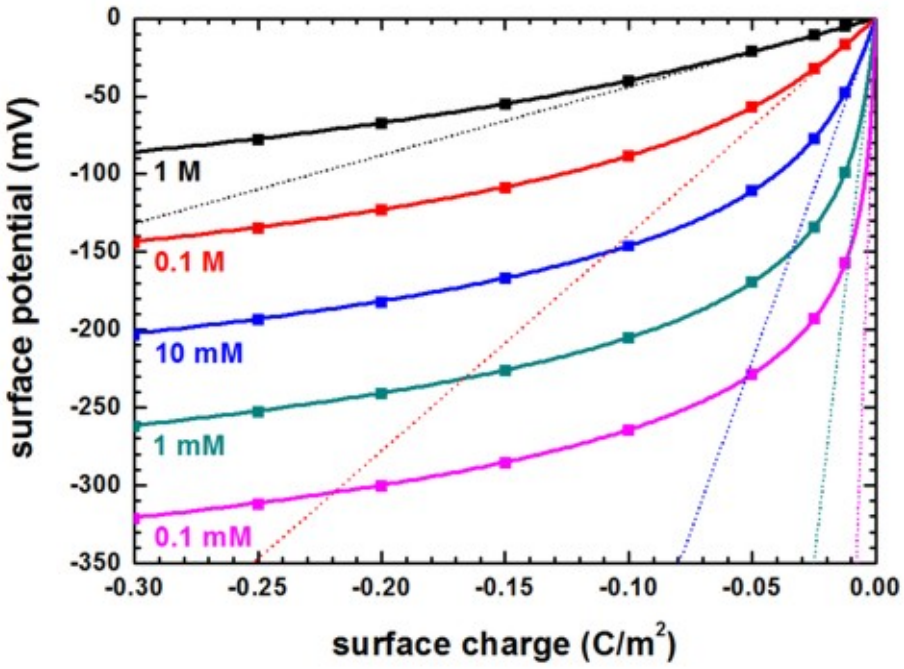


Figure 7.4.4.4: Surface potential vs. surface charge for different salt concentrations

Capacitance

The numerical Poisson–Boltzmann calculations for the capacitance (using the same data as in the previous figure) are shown next. The capacitance increases rapidly for higher potentials but at very small surface potentials, the capacitance is equal to the approximation of parallel plate capacitor model of the electric double layer. This is expected because in the limit of low potentials, the solution of the Poisson-Boltzmann equation must converge to the solution of the Debye–Hückel equation.

— DEV — Charge accumulation at the diamond–electrolyte interface

Attention: This tutorial is under construction

Input Files:

- *2DHG_Diamond_Electrolyte_Birner_Stefan_2011_1D_sPM_nn3*
- *2DHG_Diamond_Electrolyte_Birner_Stefan_2011_1D_ePM_nn3*

Relevant output files:

- *band_structure\BandEdges.dat*
- *band_structure\potential.dat*
- *density\density_hl.dat*
- *density\density_space_charge.dat*
- *density\density_ion_total.dat*
- *density\density_hl.dat*
- *density\density_ion1ion10_M.dat*
- *Shroedinger_kp\kp6_psi_squared_hl_kpar001_1D_shift.dat*

Scope of the tutorial:

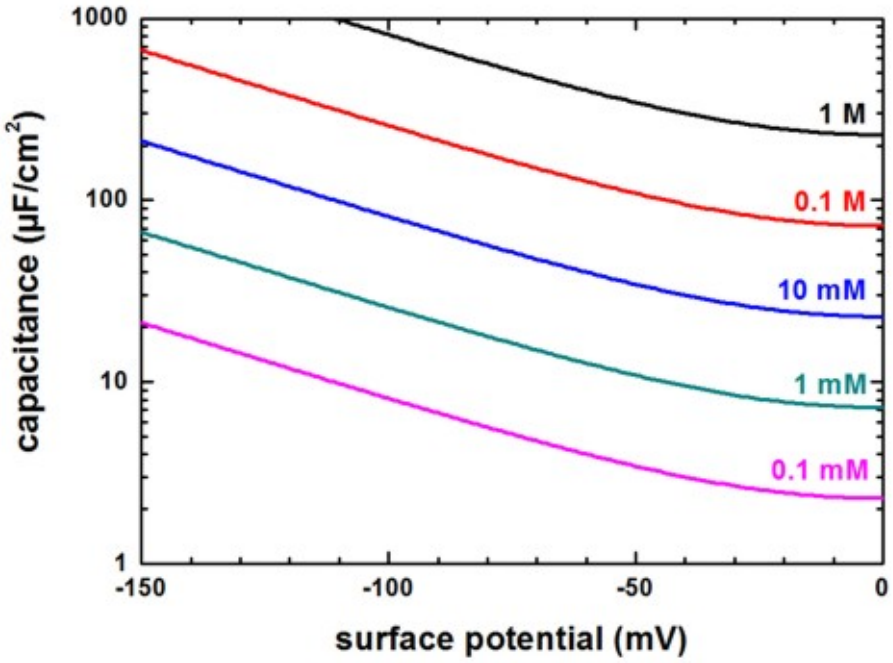


Figure 7.4.4.5: Capacitance vs. surface potential

- Poisson-Boltzmann equation
- Introduce the PMF factor

This tutorial aims to briefly summarize the results obtained in *[BirnerPhD2011]* with *nextnano*[3].

Introduction

The standard Poisson-Boltzmann equation

The standard Poisson-Boltzmann (PB) equation, which is described below, enables us to evaluate charge distributions for ions around charged surfaces.

$$\nabla\left(\varepsilon_r \nabla\varphi\left(x\right)\right) = -\sum_i z_i e C_{0,i} \exp\left(-\frac{z_i e\left(\varphi\left(x\right) - U_G\right)}{k_B T}\right)$$

Here, each variable is shown in the table.

Table 7.4.4.1: variables in the standard Poisson-Boltzmann equation

ε_r	The dielectric constant
$\varphi\left(x\right)$	The electrostatic potential at a point x
$\rho\left(x\right)$	The carrier density at a point x
z_i	The valency of an ion i
e	The elementary electric charge
$C_{0,i}$	The bulk concentration of an ion i at a point x
U_G	The potential of the reference gate electrode.
k_B	Boltzmann constant
T	The temperature

The extended Poisson-Boltzmann equation

Refer to these tutorials, the paper, and the thesis for detailed information of this model. *$electrolyte*, PMF, *[Schwierz2010]*, and *[BirnerPhD2011]*.

The standard Poisson-Boltzmann equation in *nextnano*[3] does not consider a spatially varying dielectric constant ($\varepsilon_r(x)$), which means that it is equivalent to a homogeneous dielectric constant of water ($\varepsilon_r = 78$). On the other hand, the extended Poisson-Boltzmann equation takes into account the spatially varying dielectric constant, and the potential of mean force (PMF) for the electrolyte ions. The equation is given by

$$\nabla\left(\varepsilon_r(x)\nabla\varphi(x)\right) = -\sum_i z_i e C_{0,i} \exp\left(-\frac{z_i e\left(\varphi(x) - U_G\right) + V_{PMF,i}(x)}{k_B T}\right)$$

, where the potential energy term $V_{PMF,i}(x)$ is the spatially varying potential of mean force of ion species i.

Instead of assuming a constant value of $\varepsilon_r = 78$ for the dielectric constant of water, a local dielectric constant, depending on x, for the electrolyte is employed for all PMF calculations.

The local dielectric constant $\varepsilon_r(x)$ to vary in the electrolyte as a function of distance x from the solid-liquid interface at $x_0 = 0\ nm$ is assumed as

$$\varepsilon_r(x) = \varepsilon_{r,s} + \left(\varepsilon_r^{H_2O} - \varepsilon_{r,s}\right)\frac{\rho^{H_2O}(x)}{\rho_0^{H_2O}}$$

As you can see, the dielectric constant is assumed to be proportial to the water density profile $\rho^{H_2O}(x)$. $\rho_0^{H_2O}$ is the bulk density of water, and $\varepsilon_r^{H_2O}$ is the dielectric constant of water. The density profile is obtained by molecular dynamics simulations approximated by a fitting function. $\varepsilon_{r,s}$ is the relative dielectric constant at the surface of the electrolyte. Therefore, it can become the constant of vacuum ($\varepsilon_{r,s} = 1$) if there is a distance of a few Angstrom where there are no ions or water molecules away from the solid-liquid interface (equivalent to the hydrophobic interface).

In this tutorial, we use these Poisson-Boltzmann equations to calculate charge distributions at diamond-electrolyte interfaces and reveal the influence of the hydrophobic interface on the distributions.

Hydrophobic interaction and charge accumulation at the diamond-electrolyte interface

Refer to the detail of the system used in the simulations in the thesis *[BirnerPhD2011]*. The figure below (Figure 7.4.4.6) shows the calculated valence band edge energy (black lines) of the hydrogen-terminated diamond-electrolyte interface for an applied gate voltage $U_G = -0.2\ V$.

By adjusting the gate potential applied between the reference electrode in the electrolyte and a contact on the diamond, you can shift the valence band edge with respect to the Fermi level ($E_F = 0\ eV$). This modifies the confinement potential of the resulting triangular-well, and thus also the positive charge density (light blue lines) of the 2DHG in the diamond, leading to a band bending at the surface.

The hole density is mirrored by the corresponding total ion charge density (net negative charge) in the electrolyte indicated in the violet color. The arrow indicates the region of low water density (hydrophobic region) which has a width of approximately $0.3\ nm$.

The same situation is shown in Figure 7.4.4.7.

In the left part, the three highest eigenstates (shown in the form of their probabilities) at $k_{||}$ = 0 are shown for both the hydrophobic (light green solid lines) and the standard PB approach (purple dotted lines). In fact, the states at $k_{||}$ = 0 are twofold degenerate due to spin, so essentially six states are shown. In the right part, the distribution of the Cl^- and the Na^+ ions is shown for both models. In the bulk electrolyte, both ions reach their equilibrium concentration of $50\ mM$. The potential of mean force (PMF) for the Na^+ ions causes the local maximum in the Na^+ ion density profile. For the standard PB model, both the 2DHG and the Cl^- ions are closer to the interface. The second and the third eigenstate have almost the same energy (also separated by $4\ meV$ for both the hydrophobic and the standard PB model) and the same shape. This is however difficult to see in this figure. The 2DHG sheet charge density for the hydrophobic model is $\sigma = 3.8 \times 10^{12} cm^{-2}$ and for the standard PB model $\sigma = 10 \times 10^{12} cm^{-2}$.

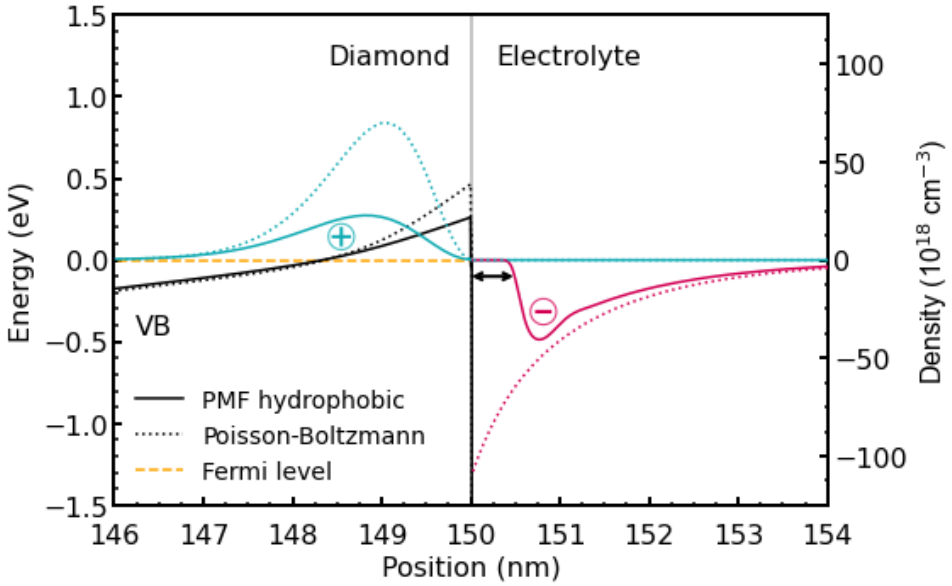


Figure 7.4.4.6: Calculated valence band edge energy (black lines) of the hydrogen-terminated diamond-electrolyte interface for an applied gate voltage $U_G = -0.2\ V$. The density of the accumulated holes (light blue lines) is mirrored by the corresponding total ion charge density (net negative charge) in the electrolyte (violet lines). The results of the standard PB calculation are shown in dotted lines whereas the results obtained with the extended PB model are shown in solid lines. The arrow indicates the region of low water density (hydrophobic region).

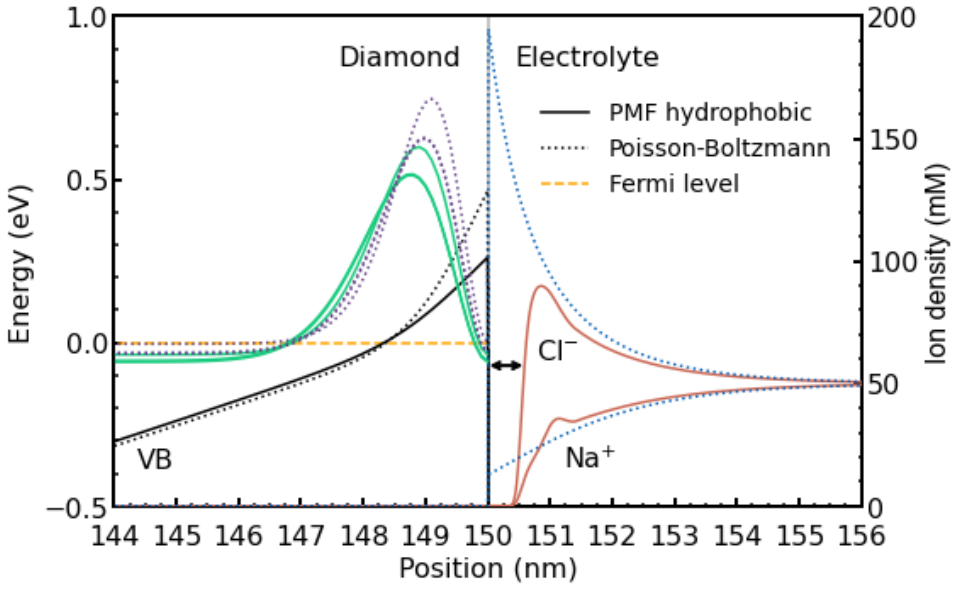


Figure 7.4.4.7: Calculated valence band edge energy (black lines) for an applied gate voltage $U_G = -0.2\ V$. The arrow indicates the hydrophobic region. The probabilities of the three highest eigenstates at $k_{||} = 0$ are shown in the left part for both models (light green lines and purple dotted lines). In the right part, the distribution of the Na^+ and Cl^- ions is shown as well. The results of the standard PB calculation are shown in dotted lines whereas the results obtained with the extended PB model are shown in solid lines.

Figure 7.4.4.8 shows the calculated electrostatic potential and valence band energy (black lines) under the same situation as in the previous figures.

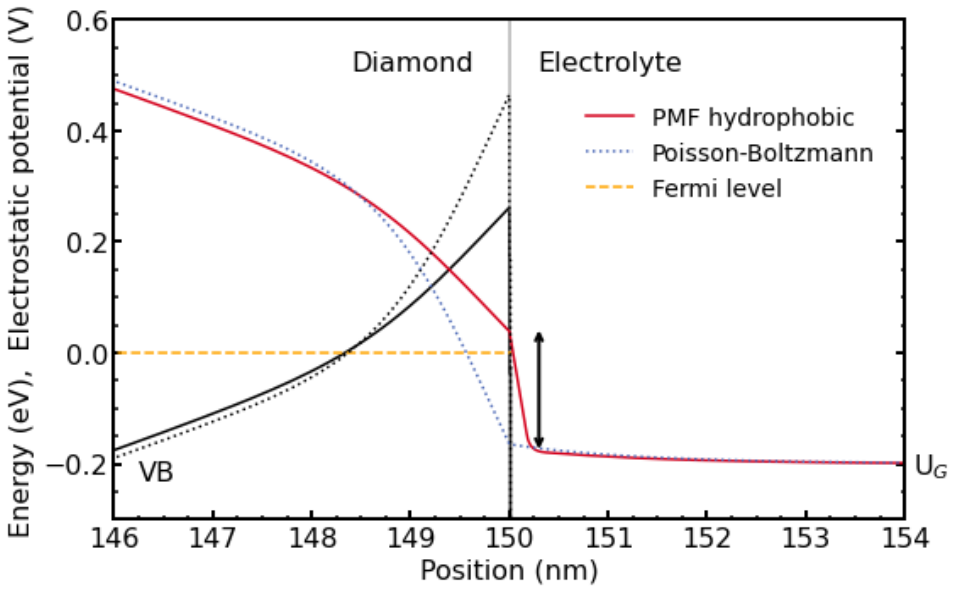


Figure 7.4.4.8: Calculated electrostatic potential and valence band edge (black lines) of the hydrogen-terminated diamond-electrolyte interface under the same situation as in the previous figures. The arrow indicates the large potential drop in the electrolyte region for the hydrophobic model (red solid line). The potential drop in the electrolyte for the standard PB model (blue dotted line) is much smaller. The results of the standard Poisson-Boltzmann calculation are shown in dotted lines whereas the results obtained with the new extended Poisson-Boltzmann model are shown in solid lines.

The applied gate voltage U_G in the electrolyte cannot be directly related to the electrostatic potential in the diamond because a part of the applied voltage drops in the electrolyte region close to the interface. The electrostatic potential distribution in Figure 7.4.4.8 reveals the potential drop across the diamond-electrolyte interface. There is a striking difference between the two models. The arrow indicates the large potential drop in the electrolyte region for the hydrophobic model (red solid line). The potential drop in the electrolyte for the standard PB model (blue dotted line) is much smaller because here the ions are allowed to approach the interface infinitely close and additionally, the dielectric constant of water is very high even close to the surface. Both effects minimize the potential drop in the electrolyte. Recall that the dielectric constant for the standard PB model has a value of $\epsilon_r = 78$ up to the interface, whereas the extended PB model assumes a dielectric constant of $\epsilon_r = 1$ in the hydrophobic 'gap' where no ions are present. As a result, the valence band edge for the hydrophobic model is closer to the Fermi level resulting in a lower 2DHG density.

The above results show that the hydrophobicity (or hydrophilicity) of the surface of most biosensor devices has a significant impact on the behavior of the devices, since the sensing signals are generated by the potential change across the interface.

Unfortunately, many *nextnano*[3] tutorials have not been moved yet to the new documentation.

Here is a list of these old tutorials.

- Tutorials indicated by (++) are also available for *nextnano++*.
- Tutorials indicated by (Ma) already moved to the new documentation.

7.4.5 Quantum Mechanics

- (++) (Ma) Triangular well
- (++) (Ma) Parabolic quantum well
- (++) (Ma) Double quantum well
- (++) (Ma) Dispersion in infinite superlattices: Minibands (Kronig-Penney model)
- (++) (Ma) Quantum corral (2D)
- (++) (Ma) Quantum wire (2D)
- (++) (Ma) Artificial atom (3D)
- (++) (Ma) Efficient method for the calculation of ballistic quantum transport - The CBR method (2D example)
- (++) (Ma) Transmission function - NEGF
- (++) (Ma) Transmission through a 3D nanowire (3D)
- (++) (Ma) Exciton correction in 1D quantum wells

7.4.6 Semiconductor Physics

- (++) (Ma) p-n junction
- Electron density in doped semiconductors (Si, Ge, GaAs) - Compensated semiconductors
- Density in n-doped GaAs - Comparison of classical, quantum, k.p and full-band density k.p approach
- (++) (Ma) Poisson equation (for different charge density profiles)

7.4.7 Strain and Piezoelectricity

- (++) Strain: Band shifts and splittings due to conduction and valence band deformation potentials
- (++) (Ma) Band gap of strained AlGaInP on GaAs substrate
- (++) Strain and displacement tensors along different growth directions
- Growth of layers on strained (or stressed) substrates (biaxial and uniaxial)
- strained silicon
- (++) Piezoelectric fields due to strain (Quantum well)
- Piezoelectric field in InAs/GaAs QWs grown along the [111] orientation
- (++) GaN/AlN wurtzite structure: Strain, piezo and pyroelectric charges in wurtzite
- (++) (Ma) Strain effects in freestanding three-dimensional nitride nanostructures (3D)

7.4.8 Magnetic field

- (++) (Ma) Landau levels in bulk GaAs (magnetic field) (2D)
- (Ma) Fock-Darwin states of a 2D parabolic potential (magnetic field) (2D)
- (Ma) Fock-Darwin states of a 2D parabolic, anisotropic (elliptical) potential in a magnetic field (2D)
- Vertically coupled quantum wires in a longitudinal magnetic field (2D)

7.4.9 Heterostructures

- Strained AlAs QW - Crossover of ground states
- (++) (Ma) Schrödinger equation of a two-dimensional core-shell structure Hexagonal 2DEG - Two-dimensional electron gas in a delta-doped hexagonal shaped GaAs/AlGaAs nanowire heterostructure (2D)
- (++) (Ma) Electron and hole wave functions in a T-shaped quantum wire grown by CEO (cleaved edge overgrowth) (2D)
- (++) (Ma) Electron and hole wave functions in a strained T-shaped quantum wire grown by CEO (cleaved edge overgrowth) (2D)
- (++) (Ma) Cubic and cuboidal shaped quantum dots (3D)
- Nanocrystals (3D)
- (++) (Ma) Artificial quantum dot crystal - Superlattice dispersion (minibands) (3D)
- Intermediate-band solar cell (artificial quantum dot crystal) (3D)
- (++) (Ma) Energy levels in a pyramidal shaped InAs/GaAs quantum dot including strain and piezoelectric fields (3D)
- Exciton correction in 2D quantum wires (2D)
- Exciton and biexciton correction in idealistic 3D cubic quantum dots (3D)
- (++) (Ma) QD molecule (3D)
- Cleaved edge overgrowth quantum dots (CEO QDs) (3D)
- (++) (Ma) Hexagonal shaped GaN quantum dot embedded in AlN (wurtzite) (3D)

7.4.10 k.p

- (++) (Ma) k.p dispersion in bulk GaAs (strained / unstrained)
- (++) (Ma) k.p dispersion in bulk GaN (strained / unstrained) (wurtzite)
- (++) (Ma) k.p dispersion in bulk unstrained ZnS, CdS, CdSe and ZnO (wurtzite)
- (++) (Ma) k|| energy dispersion of holes in a GaAs/AlAs quantum well - How to modify database parameters in the input file.
- (++) (Ma) k|| energy dispersion of holes in a GaN/AlGaN quantum well
- k|| energy dispersion of holes in unstrained and strained silicon inversion layers
- Electron density of states (DOS) of a GaAs quantum well and Hole density of states (DOS) of a Si hole channel
- Self-consistent 6-band k.p calculations of holes in strained Si/SiGe MOSFETs
- (++) (Ma) Energy dispersion of a cylindrically shaped GaN nanowire (2D)
- (++) (Ma) Artificial atom (Si QD) - 6-band k.p (3D)
- Single-band is a special case of 8-band k.p if the electrons are decoupled from the holes
  - Self-consistent calculation of the quantum mechanical density within the single-band approximation and 8-band k.p
  - Calculation of the quantum mechanical density from the k.p dispersion (no self-consistency)
- 2D Tutorial of the 1D tutorial: Single-band ('effective-mass') is a special case of 8-band k.p ('8x8kp') if the electrons are decoupled from the holes (2D)
  - Calculating the quantum mechanical density within the single-band approximation and 8-band k.p self-consistently

- Calculation of the quantum mechanical density from the k.p dispersion (no self-consistency)

7.4.11 T2SL

- (++) (Ma) InAs / In0.4Ga0.6Sb superlattice dispersion with 8-band k.p (type-II band alignment)
- (++) (Ma) InAs / GaSb broken gap quantum well (BGQW) (type-II band alignment)
- HgTe/CdTe quantum well (available on request; $\mathbf{k} \cdot \mathbf{p}$ and tight-binding)

7.4.12 2DEGs

Mobility in two-dimensional electron gases (2DEGs)

Author: Stefan Birner

The input files used in this tutorial are:

- *1DInSb_mobility_ShaoFig3.in*
- *1DGaAs_mobility_WalukiewiczFig2.in*
- *1DGaAs_mobility_WalukiewiczFig3.in*
- *1DInGaAs_mobility_WalukiewiczFig8.in*
- *1DGaN_mobility_WalukiewiczFig4.in*
- *1DGaN_mobility_WalukiewiczFig5.in*

Table of Contents

Mobility in delta-doped InSb quantum wells

This tutorial is based on the following paper:

- Carrier Mobilities in delta-doped Heterostructures, Y. Shao, S.A. Solin, L.R. Ram-Mohan, arXiv: Materials Science (2006)

Our implementation is based on the equations that are given in this paper (with the exception of Eq. (3.5) where we added a factor of $\frac{1}{4\pi}$ because of SI units).

We calculate the mobility in a 40 nm InSb quantum well that is surrounded by and strained with respect to $Al_{0.15}In_{0.85}Sb$ barriers.

At z = -20 nm, there is a delta-doping layer with a sheet doping density of $1 \times 10^{12} cm^{-2}$. The delta-doping layer is separated from the InSb QW by a 40 nm $Al_{0.15}In_{0.85}Sb$ spacer layer.

```
quantum-well-width          = 40.0 ! [nm]    [Shao] Fig. 3
spacer-width                = 40.0 ! [nm]    [Shao] Fig. 3
remote-doping-sheet-density = 1e12 ! [cm^-2] [Shao] Fig. 3
```

We calculate all properties for different temperatures. This can be done as follows:

```
$global-parameters
lattice-temperature               = 1.0   ! start value T = 1 [K]
temperature-sweep-active          = yes   ! 'yes' / 'no'
temperature-sweep-step-size       = 10.0  ! increase temperature each time by T = 10␣
↪[K]
temperature-sweep-number-of-steps = 31    ! increase temperature 31 times
data-out-every-nth-step           = 1     ! output all data for every temperature␣
```

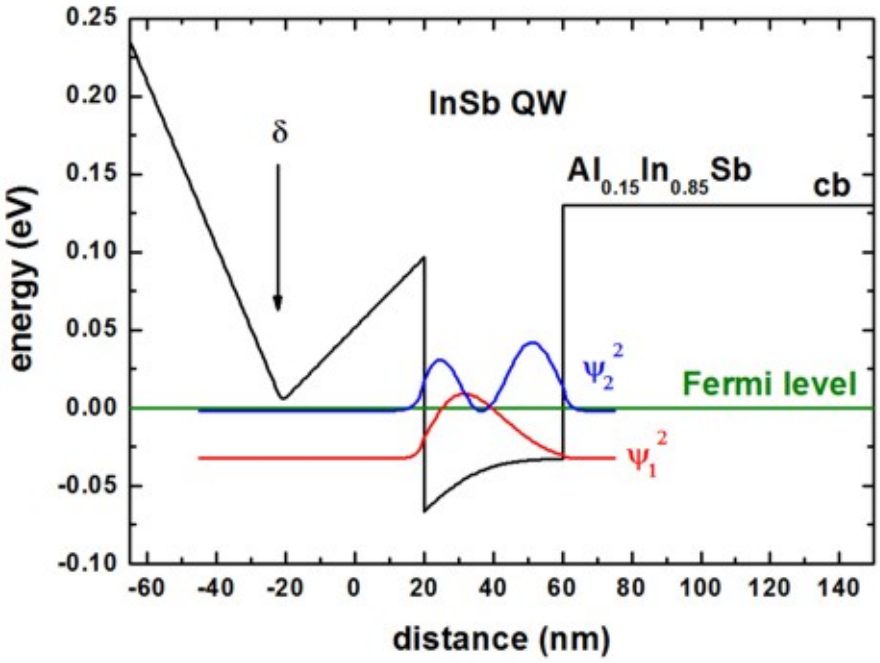
(continues on next page)

(continued from previous page)

```
↪sweep
$end_global-parameters
```

All output files are labelled with an index, starting from zero, that refers to each individual temperature sweep: `.../..._ind000....dat` Here, the index runs from 0 (000) to 30 (030), i.e. 31 output files for each property in total.

We note that band gaps and lattice constants depend on temperature. This is taken into account automatically for each temperature sweep. The following figure shows the conduction band edge, the Fermi level and the square of the lowest two electron wave functions at T = 1 K.



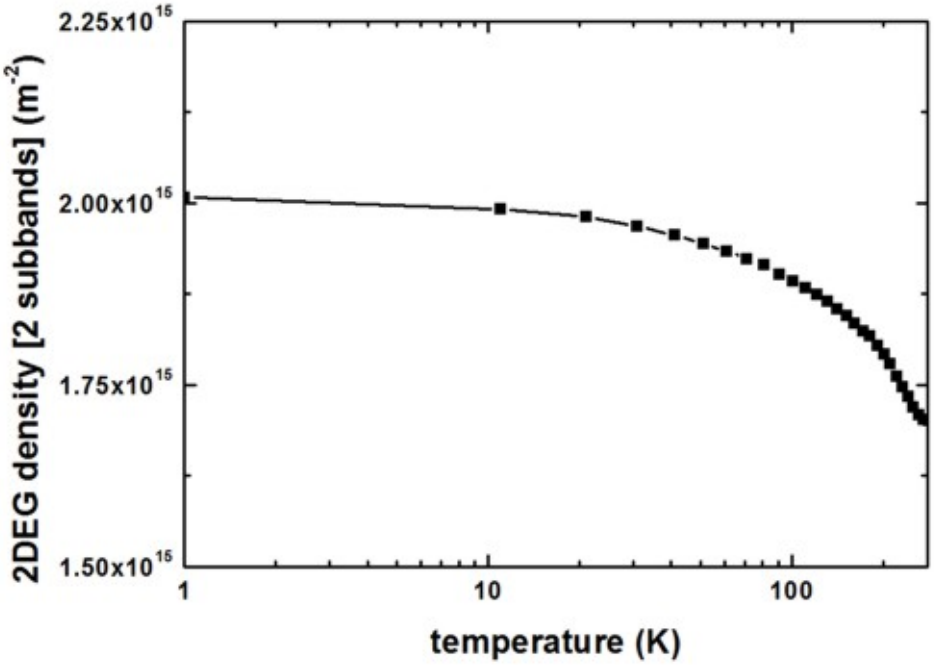
To plot such a figure, the following output files are needed:

- `band_structure/cb1D_001_ind000.dat`
  - 1st column: distance [nm]
  - 2nd column: conduction band edge at the Gamma point [eV]
  - Note: The index 'ind000' refers to the temperature sweep. Here, index 0 means T = 1 K.
- `current/fermi1Del_ind000.dat`
  - 1st column: distance [nm]
  - 2nd column: Fermi level of the electrons [eV]
  - In this tutorial, the Fermi level is always equal to 0 eV.
- `Schroedinger_1band/cb001_ind000_sg1_deg1.dat`
  - 1st column: distance [nm]
  - n columns: n energies of the eigenstates [eV]
  - n columns: n squares of the wave functions (ψ^2) [eV]
  - In the figure, we plotted the columns for ψ^2 of the two lowest states: ψ_1^2, ψ_2^2

The following figure shows the sheet electron density as a function of temperature. We considered the two lowest subbands for calculating the 2DEG density (the spin degeneracy of the subbands is included):

```
2DEG-sheet-density-number-of-subbands = 2
```

Our results differ from the results of Fig. 3(b) of the Shao's paper. (Some obvious discrepancies are the conduction band offset (We used ~0.15 eV whereas Shao used ~0.25 eV.) and the Schottky barrier height.)

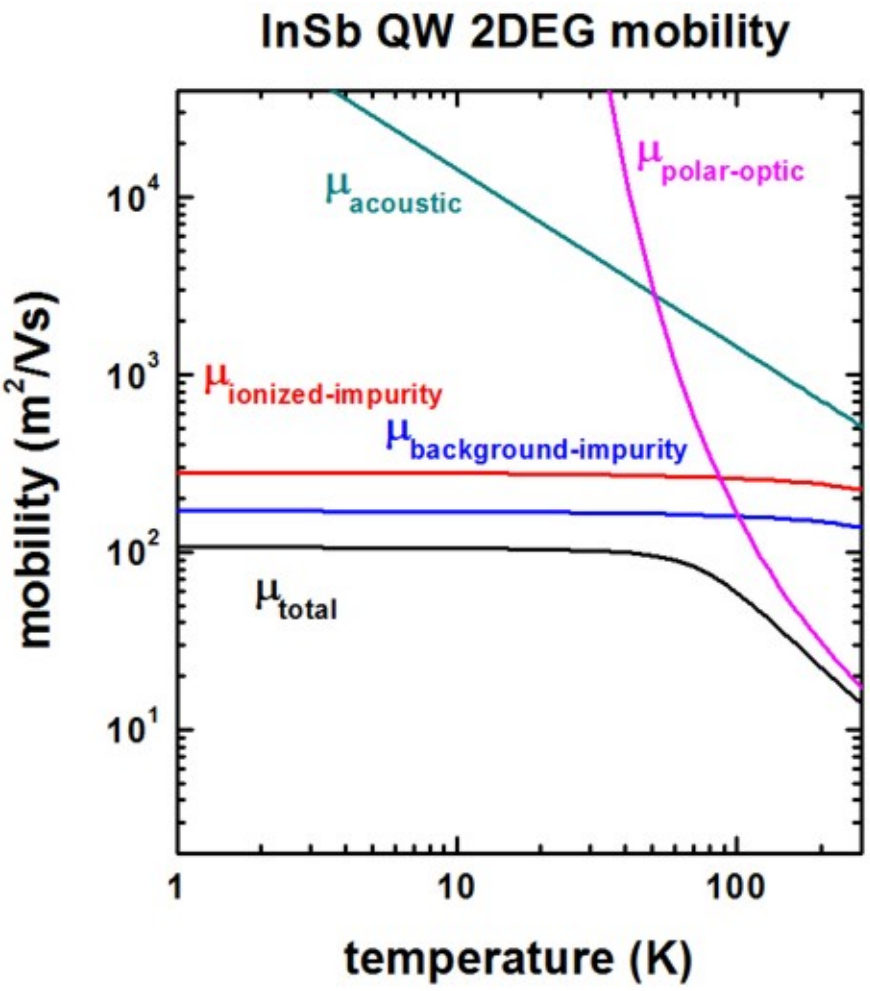


To plot such a figure, the following output file was used:

- `Monte_Carlo/mobility_TemperatureSweep.dat`
  - 1st column: temperature [K]
  - last column: electron sheet density of the lowest subband(s) [m^-2]

The following figure shows the calculated 2DEG mobility as a function of temperature. The relevant data can be found in this file:

- `Monte_Carlo/mobility_TemperatureSweep.dat`
  - 1st column: temperature [K]
  - 2nd column: total mobility [m2/Vs]
  - 3rd column: mobility due to ionized impurity scattering [m2/Vs]
  - 4th column: mobility due to background impurity scattering [m2/Vs]
  - 5th column: mobility due to deformation potential acoustic phonon scattering [m2/Vs]
  - 6th column: mobility due to polar optic (LO) phonon scattering [m2/Vs]



We included the following scattering mechanisms:

```
ionized-impurity-scattering     = yes ! [Shao] (including remote and background␣
↪ionized impurity scattering)
acoustic-phonon-scattering      = yes ! [Shao]
polar-optical-phonon-scattering = yes ! [Shao]
alloy-scattering                = no  ! [Shao]
```

We now discuss the agreement/disagreement compared to Fig. 3(a) of the [Shao] paper.

- The mobility due to acoustic phonon scattering is in excellent agreement.
- The mobility due to polar optic LO phonon scattering is in excellent agreement if one takes into account that [Shao] forgot to include the factor of 1/(4 pi) due to SI units.
- The mobility due to ionized and background impurity scattering differs significantly. It seems that the disagreement is not only due to the different sheet density that has been used. We used the following value

```
impurity-background-doping-concentration = 5e15 ! [cm^-3] [Shao] Fig. 3
```

The following InSb material parameters have been used:

```
conduction-band-masses          = 0.0135   0.0135   0.0135   ! [m0] [Shao]
                                  ...                        !
static-dielectric-constants     = 16.82    16.82    16.82    ! [Shao] epsilon(0)
optical-dielectric-constants    = 15.7                       ! [Shao] epsilon(infinity)
LO-phonon-energy                = 0.025                      ! [eV] [Shao] (optical␣
↪phonon energy)
```

(continues on next page)

(continued from previous page)

```
mass-density                    = 5.79e3                    ! [kg/m^3]   [Shao]
sound-velocity                  = 3.7e3                     ! [m/s]      [Shao]
acoustic-deformation-potential = 7.2                        ! [eV]       [Shao]
```

Mobility in doped GaAs quantum wells

The following input files were used:

- *1DGaAs_mobility_WalukiewiczFig2.in* (Experiment of Hiyamizu et al.)
- *1DGaAs_mobility_WalukiewiczFig3.in* (Experiment of DiLorenzo et al.)

Here, we test our algorithm to results on GaAs 2DEGs of another publication. We note that our algorithm is suitable for delta-doped 2DEGs but the GaAs examples are not delta-doped.

This section is based on the following paper:

[Walukiewicz]
Electron mobility in modulation-doped heterostructures
W. Walukiewicz, H.E. Ruda, J. Lagowski, H.C. Gatos
Physical Review B 30, 4571 (1984)

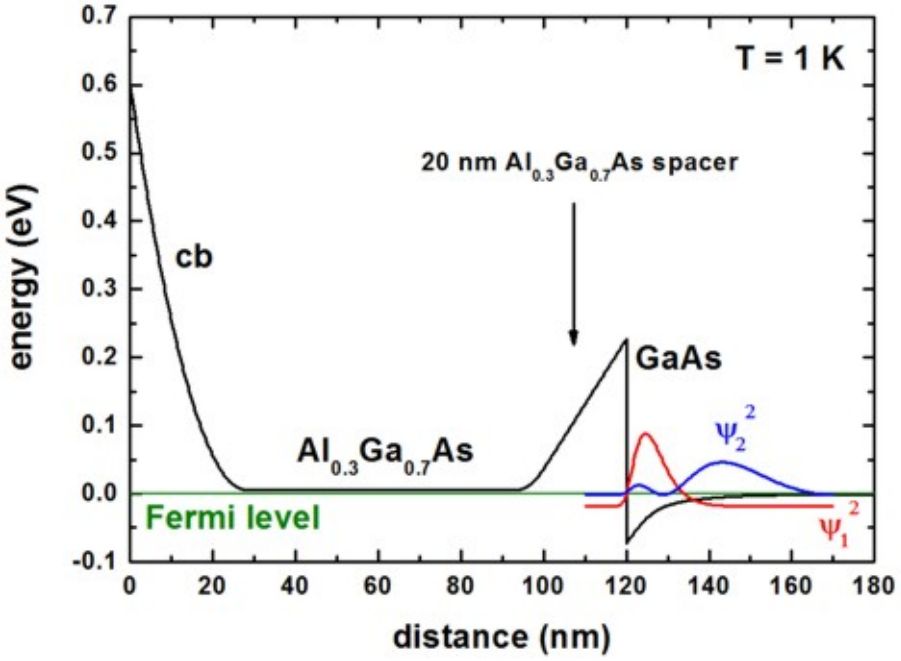
The experimental data is baed on:

[Walukiewicz, Fig. 2]: [Hiyamizu]
Improved Electron Mobility Higher than 10^6 cm^2/Vs in Selectively Doped GaAs/N-AlGaAs Heterostructures grown by MBE
S. Hiyamizu, J. Saito, K. Nanbu
Japan. J. Appl. Phys. 22, L609 (1983)

[Walukiewicz, Fig. 3]: [DiLorenzo]
Material and device considerations for selectively doped heterojunction transistors
J.V. DiLorenzo, R. Dingle, M. Feuer, A.C. Gossard, R. Hendel, J.C.M. Hwang, A. Kastalsky, V.G. Keramidas, R.A. Kiehl, P. O'Connor
IEEE-IEDM (International Electron Devices Meeting) 28, 578 (1982)

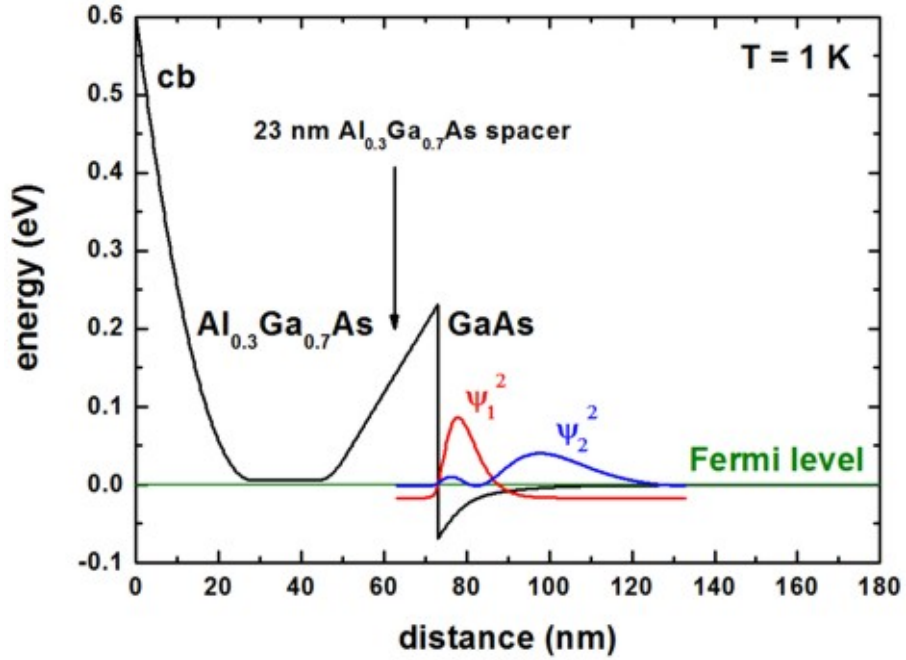
Conduction band profile and wave functions

[Walukiewicz, Fig. 2]: [Hiyamizu]




```
! 20 nm Al0.3Ga0.7As spacer
spacer-width = 20.0   ! 20 [nm]
```

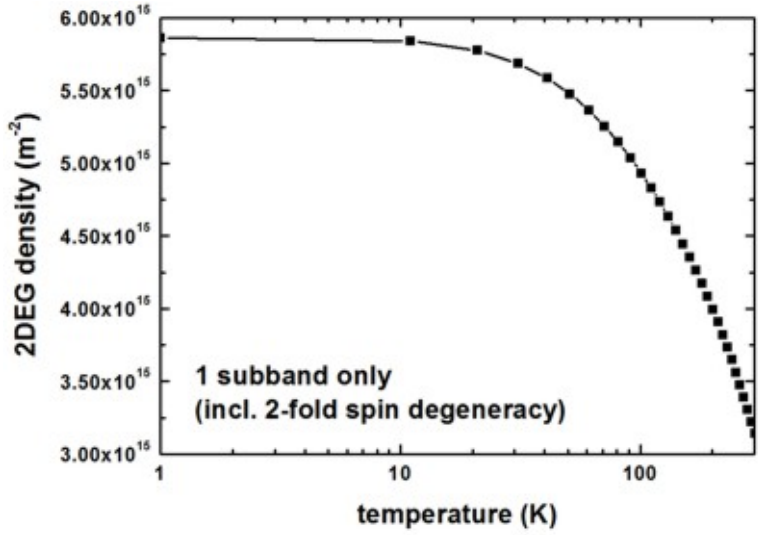
[Walukiewicz, Fig. 3]: [DiLorenzo]



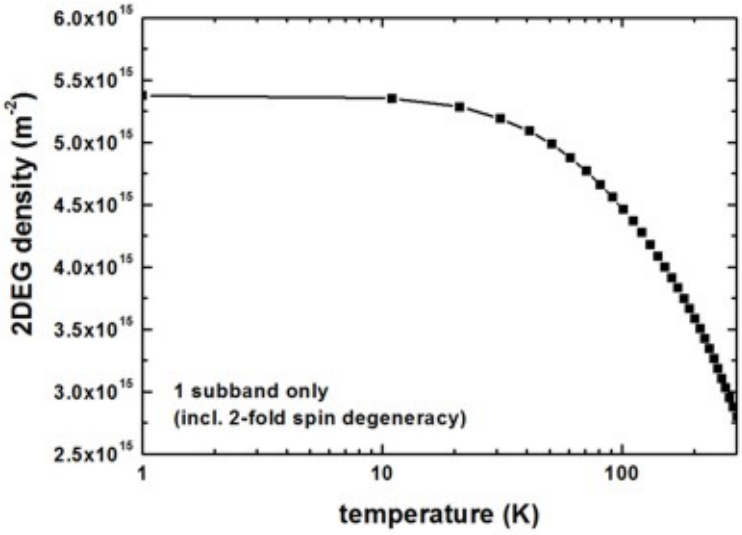

```
! 23 nm Al0.3Ga0.7As spacer
spacer-width = 23.0   ! 23 [nm]
```

2DEG sheet density

[Walukiewicz, Fig. 2]: [Hiyamizu]

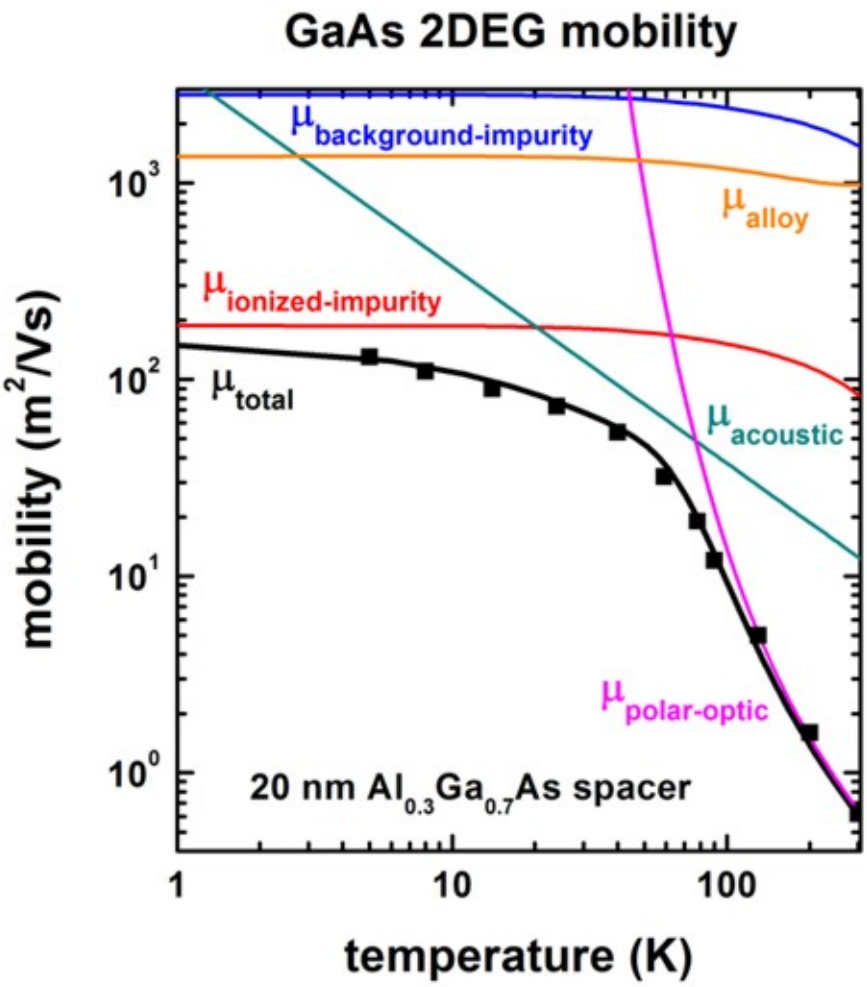


[Walukiewicz, Fig. 3]: [DiLorenzo]



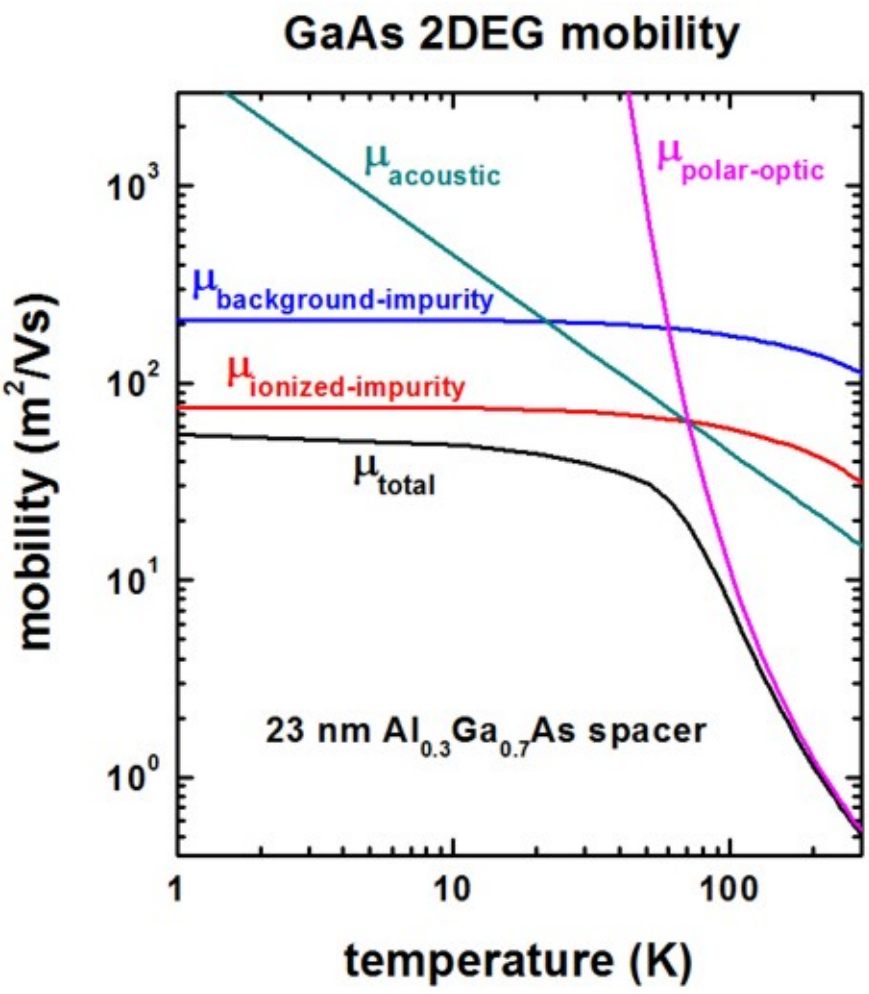
Mobility

[Walukiewicz, Fig. 2]: [Hiyamizu]




```
impurity-background-doping-concentration = 9e13 ! [cm-3]
remote-doping-sheet-density       = 3.5e11      ! [cm-2] (to fit experiment)
(remote-doping-sheet-density       = 1.948344e11 ! [cm-2])
![Walukiewicz] Fig. 2: 8.6 * 1016 [cm-3] ==> 8.6 * 1016 [cm-3]2/3 = 1.948344e11 (?)
```

[Walukiewicz, Fig. 3]: [DiLorenzo]




```
impurity-background-doping-concentration = 1e15 ! [cm-3]
remote-doping-sheet-density              = 1e12 ! [cm-2]
```

Differences with respect to Walukiewicz [Hiyamizu] paper

Walukiewicz used a 2DEG density of $3 \times 10^{11}\text{cm}^{-2}$.

```
conduction-band-masses = 0.067   0.067   0.067   ! [m0]
                         ...                     ! a higher value than for bulk␣
↪because of nonparabolicity
```

Here we used 0.067 as this gives better agreement to the mobility at higher temperatures and this is the usually accepted material parameter for GaAs.

```
quantum-well-width = 13.0 ! [nm] (13 nm seems to be a better value than 20 nm.)

alloy-scattering   = yes
```

Alloy scattering is relevant for the part of the wave function that penetrates into the AlGaAs barrier.

The squares are experimental values of Fig. 5 in:

Improved Electron Mobility Higher than 106 cm^2/Vs in Selectively Doped GaAs/N-AlGaAs Heterostructures Grown by MBE

S. Hiyamizu et al.
Jpn. J. Appl. Phys. 22, L609 (1983)

The following GaAs material parameters have been used:

```
static-dielectric-constants    = 12.9    12.9    12.9    ! [Walukiewicz] epsilon(0)
optical-dielectric-constants   = 10.9                    ! [Walukiewicz]␣
↪epsilon(infinity)
LO-phonon-energy               = 0.036                   ! [eV] [Walukiewicz]␣
↪(optical phonon energy)

mass-density                   = 5.318e3                 ! [kg/m^3]  [Davies] p. 411
sound-velocity                 = 5.29e3                  ! [m/s]     [X.L. Lei, J.␣
↪Phys. C 18, L593 (1985)]
acoustic-deformation-potential = 7.0                     ! [eV]      [Walukiewicz]
```

Into the equation for the **deformation potential acoustic phonon scattering**, the quantum well width is an input parameter. We used a value of 13 nm which corresponds roughly to the extension of the ground state wave function inside the "triangular" QW.

Differences with respect to Walukiewicz [DiLorenzo] paper

In Walukiewicz's paper, the background impurity scattering is dominating the ionized impurity scattering. We found the opposite.

Walukiewicz used a 2DEG density of $2.2 - 3.8 \times 10^{11}\mathrm{cm}^{-2}$

```
conduction-band-masses = 0.076   0.076   0.076   ! [m0] [Walukiewicz]
                         ...                     ! a higher value than for bulk␣
↪because of nonparabolicity. This is also the value used by Walukiewicz.




quantum-well-width = 20.0 ! [nm] (This value might be too large. See left where 13 nm␣
↪was used.)

alloy-scattering   = no
```

Mobility in doped InGaAs quantum wells

The input file used is:

- *1DInGaAs_mobility_WalukiewiczFig8.in* (Experiment of Kastalsky et al.)

Here, we test our algorithm to results on InGaAs 2DEGs of another publication. We note that our algorithm is suitable for delta-doped 2DEGs but the InGaAs examples are not delta-doped.

This tutorial is based on the following paper:

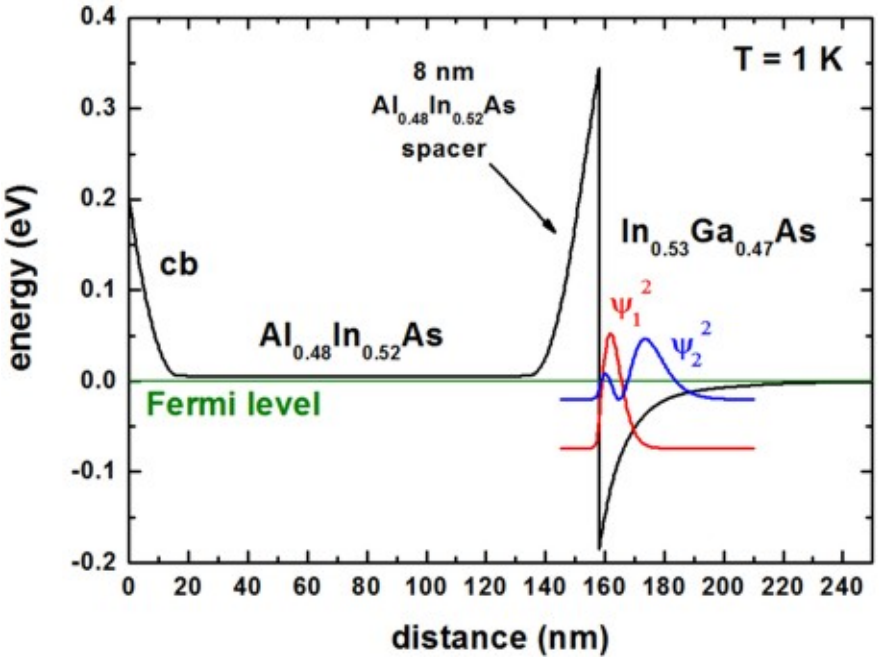
[Walukiewicz]
Electron mobility in modulation-doped heterostructures
W. Walukiewicz, H.E. Ruda, J. Lagowski, H.C. Gatos
Physical Review B 30, 4571 (1984)

The experimental data is based on:

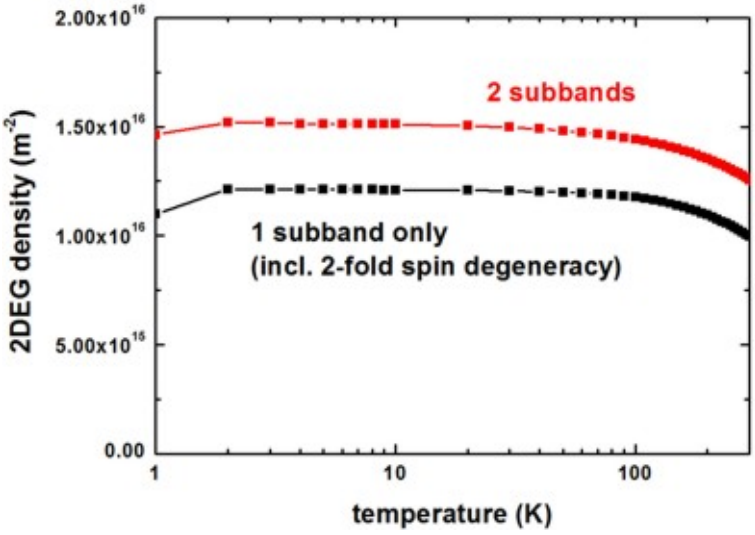
[Walukiewicz, Fig. 8]: [Kastalsky]
Two-dimensional electron gas at a molecular beam epitaxial-grown, selectively doped, In0.53Ga0.47As-In0.48Al0.52As interface
A. Kastalsky, R. Dingle, K.Y. Cheng, A.Y. Cho
Applied Physics Letters 41, 274 (1982)

[Kastalsky] and [Walukiewicz] have used $In_{0.52}Al_{0.48}As$ whereas we used $In_{0.52}Al_{0.48}As$ which is lattice matched to InP and $In_{0.53}Al_{0.47}As$.

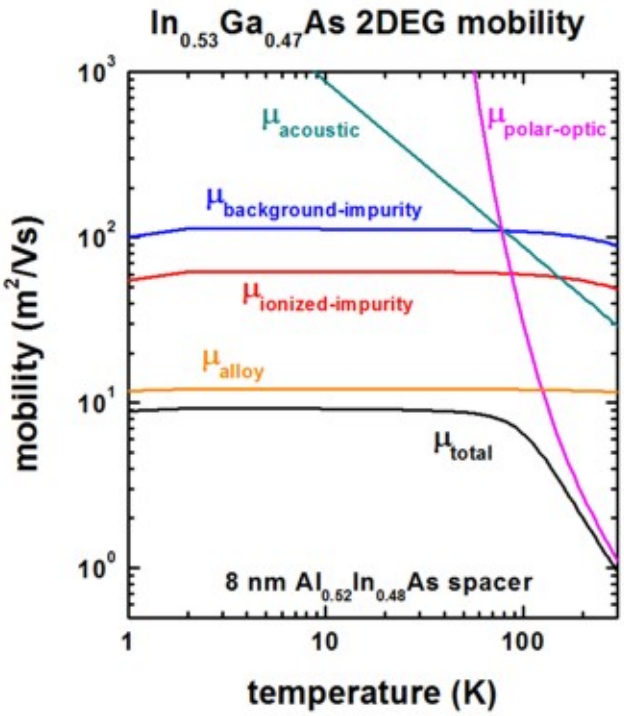
The conduction band profile is shown in the following figure. Here, two subbands (ψ_1^2, ψ_2^2) are occupied although our implementation of calculating the mobility is only applicable to one occupied subband. Note that the 2DEG is located in an alloy, i.e. InGaAs. Thus we expect that alloy scattering has a significant effect on the total mobility.



The following figure shows the subband density of the **first subband** and of the first two subbands as a function of temperature. Inside the mobility algorithm, only the density of the first subband has been considered.



The following figure shows the mobility as a function of temperature. At temperatures below 100 K, the total mobility is dominated by alloy scattering.



Our results for the mobility are in reasonable agreement with Fig. 8 of the paper of [Walukiewicz].

The following parameters were used:

```
ionized-impurity-scattering                = yes     ! (including remote and background␣
↪ionized impurity scattering)
acoustic-phonon-scattering                 = yes     !
polar-optical-phonon-scattering            = yes     !
alloy-scattering                           = yes     !
quantum-well-width                         = 15.0    ! [nm] 15 nm seems to be a␣
↪reasonable approximation for the triangular well
spacer-width                               = 8.0     ! [nm]
impurity-background-doping-concentration = 1.0e16  ! [cm-3]
remote-doping-sheet-density                = 1.0e12  ! [cm-2]
2DEG-sheet-density-number-of-subbands      = 1
alloy-disorder-scattering-potential        = 0.60    ! [eV] InGaAs bulk value [J.R.␣
↪Hayes et al. (1982)]

!-------------------------------------------------------------------------------
! In0.53Ga0.47As material parameters
!-------------------------------------------------------------------------------
mass-density                  = 5.5025e3  ! [kg/m^3] InGaAs            [Wen et al.,␣
↪JAP 100, 103516 (2006)]
sound-velocity                = 4.753e3   ! [m/s]    In0.53Ga0.47As[111] [Wen et al.,
↪ JAP 100, 103516 (2006)]
acoustic-deformation-potential = 7.0       ! [eV]     InGaAs            [Walukiewicz]
```

Mobility in doped GaN quantum wells

The input files used are:

- *1DGaN_mobility_WalukiewiczFig4.in*
- *1DGaN_mobility_WalukiewiczFig5.in*

Here, we test our algorithm to results on GaN 2DEGs of another publication. We note that our algorithm is suitable for delta-doped 2DEGs but the GaN examples are not delta-doped.

This tutorial is based on the following paper:

[WalukiewiczGaN]
Electron mobility in AlxGa1-xN/GaN heterostructures
L. Hsu, W. Walukiewicz
Physical Review B 56, 1520 (1997)

Note: To be consistent with the paper of Walukiewicz, no (!) piezo- and pyroelectricity is included.

The following GaN material parameters have been used:

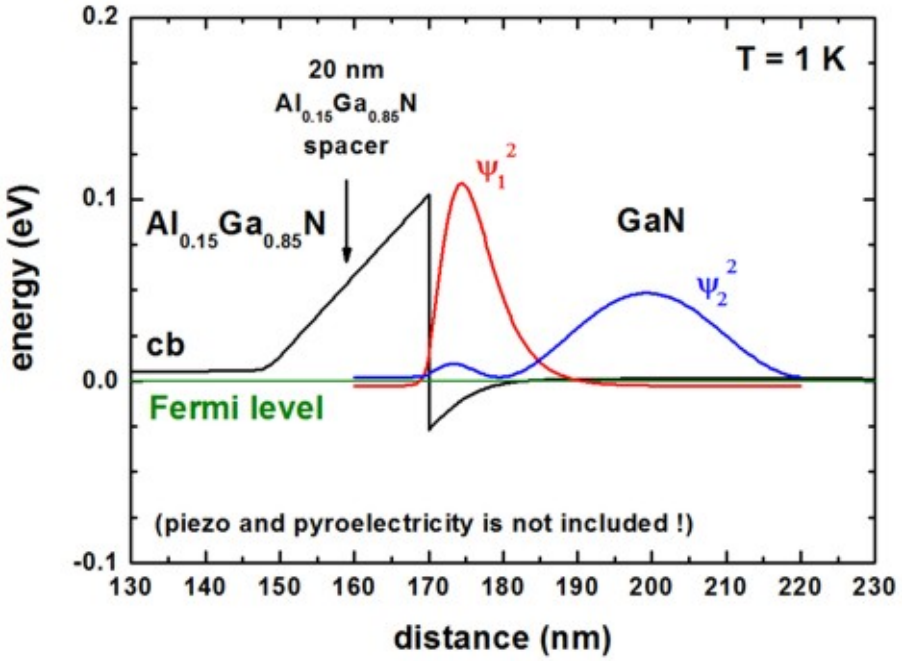
```
conduction-band-masses          = 0.21     0.21     0.21       ! [m0] [WalukiewiczGaN]
static-dielectric-constants     = 9.5      9.5      9.5        ! [WalukiewiczGaN]␣
↪epsilon(0)
optical-dielectric-constants    = 5.35     5.35     5.35       ! [WalukiewiczGaN]␣
↪epsilon(infinity)
LO-phonon-energy                = 0.0905   0.0905   0.0905     ! [eV] [WalukiewiczGaN]␣
↪(optical phonon energy)

mass-density                    = 6.1e3                        ! [kg/m^3] ␣
↪[WalukiewiczGaN]
sound-velocity                  = 6.6e3                        ! [m/s]    ␣
↪[WalukiewiczGaN]
acoustic-deformation-potential = 8.5                           ! [eV]     ␣
↪[WalukiewiczGaN]
alloy-disorder-scattering-potential = 2.3 ! [eV] conduction band offset GaN/AlN
```

Here, alloy scattering is only relevant for the part of the wave function that penetrates into the AlGaN barrier.

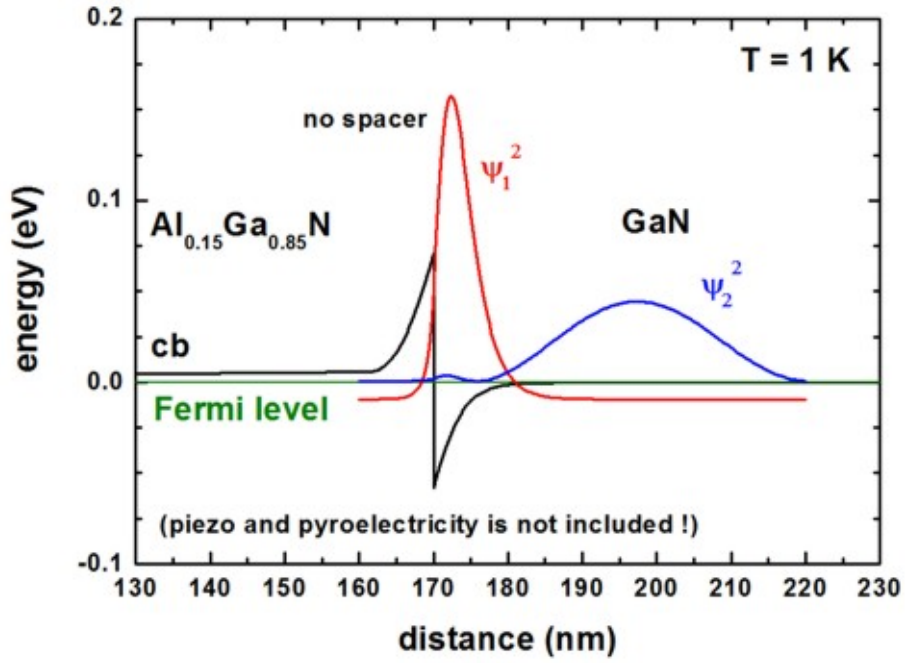
Conduction band profile and wave functions [WalukiewiczGaN]

[WalukiewiczGaN] Fig. 4




```
! 20 nm Al0.15Ga0.85N spacer
spacer-width = 20.0   ! 20 [nm]
```

[WalukiewiczGaN] Fig. 5

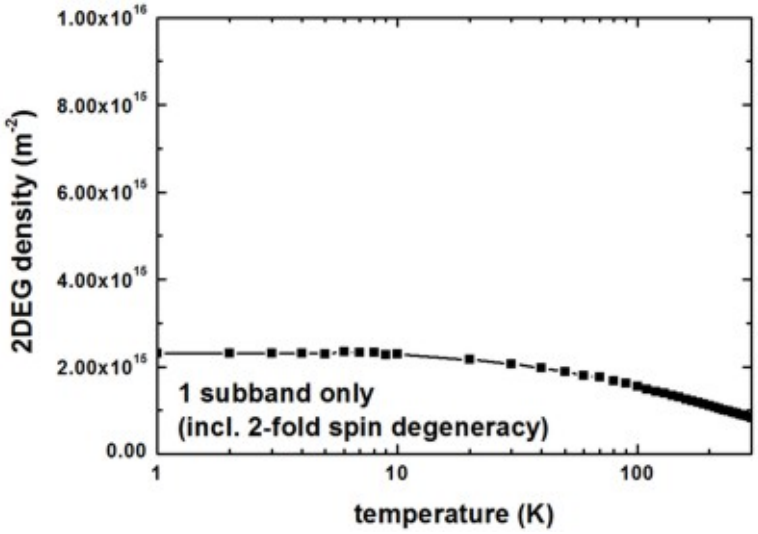



```
! no spacer
spacer-width = 0.0   ! 0 [nm]
```

2DEG sheet density

[WalukiewiczGaN] Fig. 4

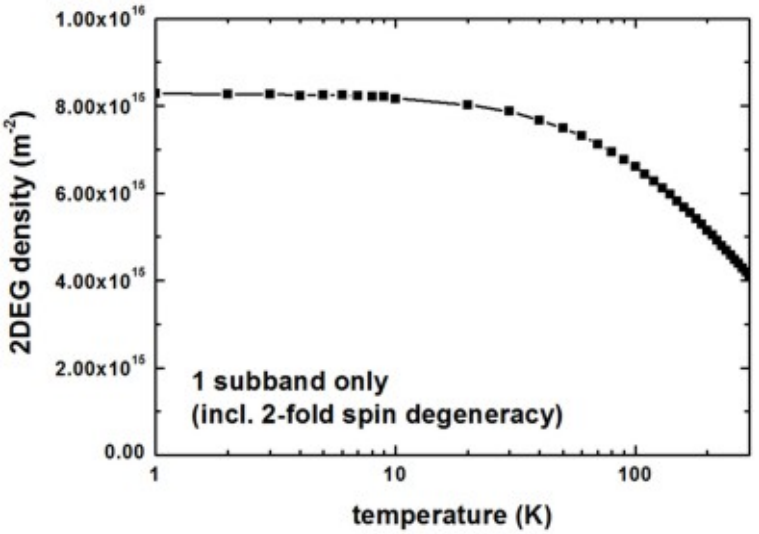
Into the equation for the **deformation potential acoustic phonon scattering**, the quantum well width is an input parameter. We used values which correspond roughly to the extension of the ground state wave function inside the "triangular" QW.



[WalukiewiczGaN] used a 2DEG density of 6.2 * 1015 m-2.

```
quantum-well-width = 15.0 ! 15 [nm]
```

[WalukiewiczGaN] Fig. 5

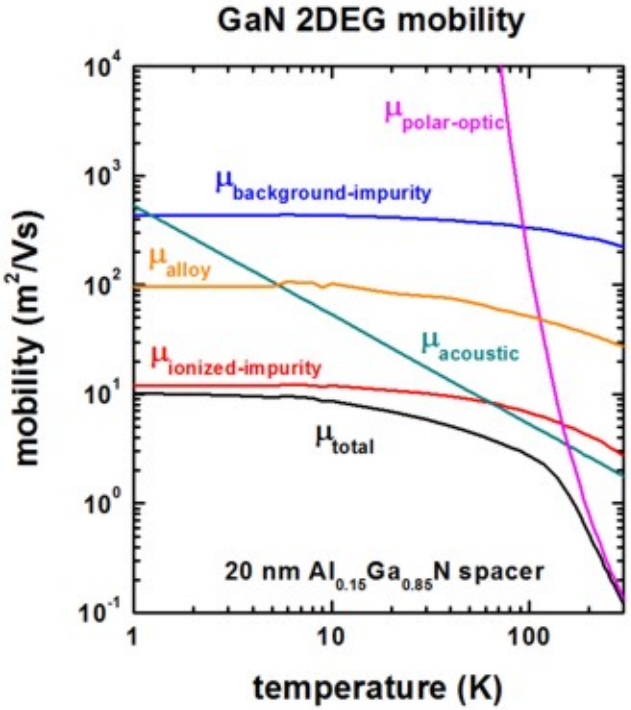


[WalukiewiczGaN] used a 2DEG density of 1.59 * 1016 m-2.

```
quantum-well-width = 10.0 ! 10 [nm]
```

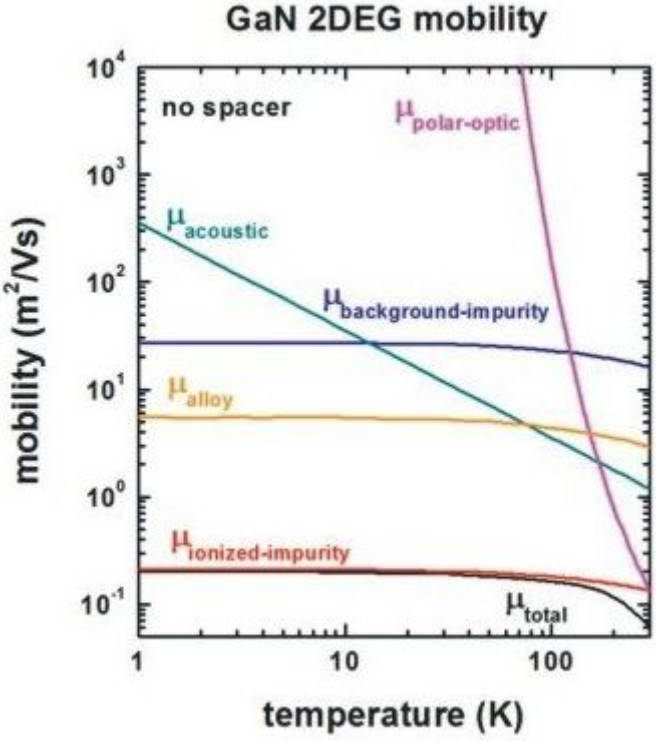
Mobility

[WalukiewiczGaN] Fig. 4




```
impurity-background-doping-concentration = 1e14 ! [cm-3]
remote-doping-sheet-density       = 7.883735e11 ! [cm-2]
![WalukiewiczGaN] Fig. 4: 7 * 1017 [cm-3] ==> 7 * 1017 [cm-3]2/3
```

[WalukiewiczGaN] Fig. 5



```
impurity-background-doping-concentration = 4e15 ! [cm-3]
remote-doping-sheet-density              = 1e12 ! [cm-2]
```

Final remark: In principle, the results of these GaN 2DEGs are not reliable as piezo- and pyroelectricity have to be included. Our results disagree quantitatively with the results of [WalukiewiczGaN]. However, it is not clear, which material parametes he used for the conduction band offset and the alloy scattering.

Further hints

If two remote doping regions should be taken into account, one can input an array of values.

```
spacer-width                          = 20.0   10.0   ! [nm]   spacer width of 1st␣
↪and 2nd doping region
remote-doping-sheet-density           = 1e12   1e11   ! [cm-2]  remote doing density␣
↪of 1st and 2nd doping region
```

- (++) (Ma) Schrödinger-Poisson
- (++) (Ma) Si/SiGe MODQW (Modulation Doped Quantum Well)
- (++) (Ma) HEMT structure (High Electron Mobility Transistor)
- inverted High Electron Mobility Transistor (HEMT)
- (++) (Ma) Two-dimensional electron gas in an AlGaN/GaN field effect transistor
- (++) (Ma) Single-electron transistor (3D)

7.4.13 Optoelectronics

- (++) (Ma) Cascade solar cell (AlGaAs/InGaAs Tandem solar cell)
- (++) (Ma) Quantum Cascade Laser (simple structure)
- (++) (Ma) Quantum Cascade Laser
- (++) (Ma) Intraband transitions
- (++) (Ma) Interband transitions
- (++) (Ma) Intersubband transitions in InGaAs/AlInAs multiple quantum well systems
- (++) (Ma) Scattering times for electrons in unbiased and biased single and multiple quantum wells
- Optical absorption of an InGaAs quantum well

7.4.14 Electronics

- (++) (Ma) Capacitance-Voltage (C-V) curve of a “metal”-insulator-semiconductor (MIS) structure
- p-Si / SiO2 / poly-Si structure (MOSFET with inversion channel due to applied gate voltage)
- (++) (Ma) Resistance in bulk n-type doped silicon
- (++) (Ma) n-i-n Si resistor (classical and quantum mechanical current calculation)
- I-V characteristics of an n-doped Si structure
- (++) (Ma) I-V characteristics of an n-i-n-doped Si structure
- (++) (Ma) Ultrathin-body Double Gate FET - Double Gate MOSFET (Metal Oxide Semiconductor Field Effect Transistor) (2D)
- (++) (Ma) Double Gate MOSFET (5 nm) (quantum mechanical calculation) (2D)

- (Ma) Electron wave functions of a 2D slice of a Triple Gate MOSFET (2D)

7.4.15 NEGF

- Ballistic current in a resonant tunneling diode (RTD)

7.4.16 Electrolyte

for e.g. Ion-Sensitive Field Effect Transistors (ISFETs)

- Extended Poisson-Boltzmann equation: Potentials of mean force (PMF)
- Detection of proteins with a SOI (silicon-on-insulator) electrolyte sensor
- Si-SiO2 electrolyte sensor
- Electrolyte Gate AlGaN/GaN Field Effect Transistor as pH Sensor
- Diamond biosensors (available on request)

7.4.17 Graphene

- Carrier statistics of graphene sheets

7.4.18 Hello World

- Simple SiGe structure (This is a very ancient tutorial and was written 20 years ago. Meanwhile the software has evolved.)
  - Step 1: Simple SiGe structure (input_file1.in)
  - Step 2: Include current (input_file2.in)
  - Step 3: Include quantum models (1-band Schrödinger equation) (input_file3.in)
  - Step 4: Read in data and solve k.p (input_file4.in)

  Four example input files are provided in this tutorial which represent typical simulation settings that are connected to each other. These files contain extensive comments to guide you through your first experience with the input system.

7.5 Command Line

Command line usage:

```
nextnano3_Intel_64bit.exe --inputfile 1D_HEMT.in --database database_nn3.in --license
License_nnp.lic --outputdirectory 1D_HEMT/
```

Currently *nextnano*[3] provides the following options:

-i input_file, --inputfile input_file Name of input file to be read in.

Example:

- `--inputfile 1D_GaAs_HEMT.in`
- `--inputfile "1D GaAs HEMT.in"`

Blanks in file names are supported if using quotation marks " ".

Default value: Entry in `keywords.val`.

-d database_file, --database database_file Name of material parameter database file to be read in.

Example:

- `--database database_modified_my_Si_parameters.in`
- `--database "my database_nn3.in"`

Blanks in file names are supported if using quotation marks " ".

Default value: Entry in file `database_nn3_keywords.val`, i.e. `database_nn3.in`.

-l license_file, --license license_file Name of license file to be read in.

Example:

- `--license license.dat`
- `--license License_nnp.lic`
- `--license H:\Documents\nextnano\license\license.dat`
- `--license "H:\My Documents\nextnano\license\license.dat"`

Default value: `License_nnp.lic`

-o output_directory, --outputdirectory output_directory Directory name where to output results.

Example:

- `--outputdirectory  my_output_foldername`
- `--outputdirectory  my_output_foldername/`
- `--outputdirectory "my output foldername/"`
- `--outputdirectory "my output\my folder\"`
- `--outputdirectory "<name_of_input_file>"`
- `--outputdirectory "<name_of_input_file>\output"`
- `--outputdirectory "E:\My Documents\nextnano_output\<name_of_input_file>"`

In the last three examples the string `<name_of_input_file>` is provided and *not* the name of the input file. This special string `<name_of_input_file>` will be replaced automatically with the actual name of the input file.

Blanks in folder names are supported if using quotation marks " ".

-t i, --threads i Number of parallel threads (`i` = integer) to benefit from parallelization on multi-core CPUs.

Example:

- `--threads 1` uses 1 thread (default)
- `--threads 4` uses 4 threads (only works with an executable that had been compiled with OpenMP option)
- `--threads 0` automatic decision, i.e. use value specified in input file, or use default value if not specified in input file

This setting only affects parallelization of e.g. $k_{||}$ vectors or energy grid points (NEGF). Parallelization of calls to MKL are still parallelized automatically.

--debuglevel i Generate additional debug information (`i` = integer).

Example:

- `--debuglevel 3`

The integer number has the following meaning:

- `0`: no debug information at all (default)
- `1`: modest debug information
- `2`: more debug information
- `3`: even more debug information
- `n`: any other integer number is also possible, some have specific meanings (for developers only)
- `> 1000`: Here, the files `keywords_nn3.xml` and `database_nn3_keywords_nn3.xml` are created, which are used by *nextnanomat* for its auto completion feature.
- `< 0`: automatic decision, i.e. use value specified in input file, or use default value if not specified in input file

--log Generate a `*.log` file of screen output (standard output).

The file will be written to: `<outputdirectory>/<inputfilename>_log.log`

--cancel i Automatically cancel simulation after `i` minutes (*kill*, i.e. program stops immediately).

If value is <= `0`, the simulation is not canceled.

Example:

- `--cancel 10`

--softkill i Automatically cancel simulation after `i` minutes (*soft kill*, i.e. program exits iteration cycle and writes output).

If value is <= `0`, the simulation is not canceled.

Example:

- `--softkill 10`

Alternative option: If the user saves a file called `SOFT_KILL` (without file extension) into the output folder of the currently running simulation, a *soft kill* will be performed.

--system operating_system Flag indicating on which operating system the executable is executed.

Example:

- `--system default` (default)
- `--system windows`
- `--system linux`

Available options are `default`, `windows`, `linux`, `unix` and `mac`.

When `default` is specified, the corresponding setting of your executable is applied.

Typically, it is not required to specify this flag as the executables had been compiled for the respective operating systems (and thus this setting has already been applied automatically).

Currently, the internal settings for `linux`, `unix`, `mac` are identical.

--condor Flag that is necessary if job is submitted to *HTCondor*.

-p, --parse Parse input file and quit.

--parse-keywords	Parse `keywords.val` file. By default, the content of the file `keywords.val` is read in from within the source code.
--parse-database	Parse `database_nn3_keywords.val` file. By default, the content of the file `database_nn3_keywords.val` is read in from within the source code.
-v, --version	Show version number only.
-h, --help	Display available command line options.

Note: For versions before 2021, the double-hyphen flags `--` are not supported. Please use instead the appropriate single-hypen flags `-`:

- `-inputfile`
- `-database`
- `-license`
- `-outputdirectory`
- `-help` (Displays allowed options.)

7.6 Release Notes

7.6.1 2.6.4.b (2024-08-05)

- new output of doping ionization ratio
- minor bugfix for $\mathbf{k} \cdot \mathbf{p}$ solver
- Altermatt incomplete ionization model implemented

7.6.2 2.6.1.b (2024-04-02)

- minor changes

7.6.3 2.6.0.b (2024-02-01)

- multiple improvements of messages in log file
- scaling bugfix for lifetimes
- bugfix for drift velocity
- parametrization bugfix for 30-band $\mathbf{k} \cdot \mathbf{p}$ model

7.6.4 2.5.4.b (2023-07-27)

syntax

- `MIN()` and `MAX()` functions introduced
- `!WHEN` statement introduced for conditional lines
- `!TEXT` and `!ENDTEXT` statements introduced for multi-line comments
- `!DATA` statement got introduced for post-processor

output

- output of material parameters in the *nextnano++* database style

models

- excitonic absorption
- tunneling current for CBR

7.6.5 2.3.8.b (2022-09-29)

- Bowing parameters of band edges has been updated in the default database.
- 30-band $\mathbf{k} \cdot \mathbf{p}$ model for bulk crystals without strain has been implemented.

7.6.6 2.3.7.b (2022-08-03)

- band-anti-crossing (BAC) model for nitrides as an experimental feature

---

7.6.7 EARLIER

- Bulk band structure with 14-band and 30-band k.p
- Improvements for CBR
- New 2D CBR input files (QPC)
- Improvement of 2DEG mobility input files (e.g. GaN)
- Synonyms in material database (e.g. Al(x)In(x)As and In(x)Al(1-x)As)
- Input files for tutorial: "Intersubband absorption in infinite QW"
- Input files for intersubband absorption in Ge/SiGe QWs
- Added more tutorial input files to samples folder
- Improvements for intersubband absorption (single-band, simple k.p model, k.p)
- Grid lines can be independent of region object boundaries (has always been the case for nn++)
- 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
- 2nd order piezo constants and 2nd order piezo tutorial

- Bulk band structure with pseudopotential method
- Bulk band structure with tight-binding method (improvements)
- Piezo/pyro tutorial for arbitrary (hkil) in wurtzite

CHAPTER

EIGHT

NEXTNANO.NEGF

The *nextnano.NEGF* tool is designed for accurate quantum simulations of semiconducting devices using the Non-Equilibrium Green Functions (**NEGF**) formalism. It has been originally developed to simulate electron dynamics and gain in quantum cascade lasers (**QCLs**) and superlattices. It can also be apply to infrared detectors such as quantum well infrared detectors (**QWIPs**) and quantum cascade detectors (**QCDs**), as well as transport in resonant tunneling diodes (**RTDs**). It is progressively extended to further applications such as interband devices.

Note: This documentation section is under development. More information about *nextnano.NEGF* can be found at: https://nextnano-docu.northeurope.cloudapp.azure.com/dokuwiki/doku.php?id=qcl:input_file

8.1 Overview

The *nextnano.NEGF* tool is designed for accurate quantum simulations of semiconducting devices using the Non-Equilibrium Green Functions (**NEGF**) formalism. It has been originally developed to simulate electron dynamics and gain in quantum cascade lasers (**QCLs**) and superlattices. It can also be apply to infrared detectors such as quantum well infrared detectors (**QWIPs**) and quantum cascade detectors (**QCDs**), as well as transport in resonant tunneling diodes (**RTDs**). It is progressively extended to further applications such as interband devices.

This simulation tool relies on the NEGF formalism, which has been shown to be a powerful framework to model quantum transport in presence of scattering processes.

The following scattering mechanisms are included:

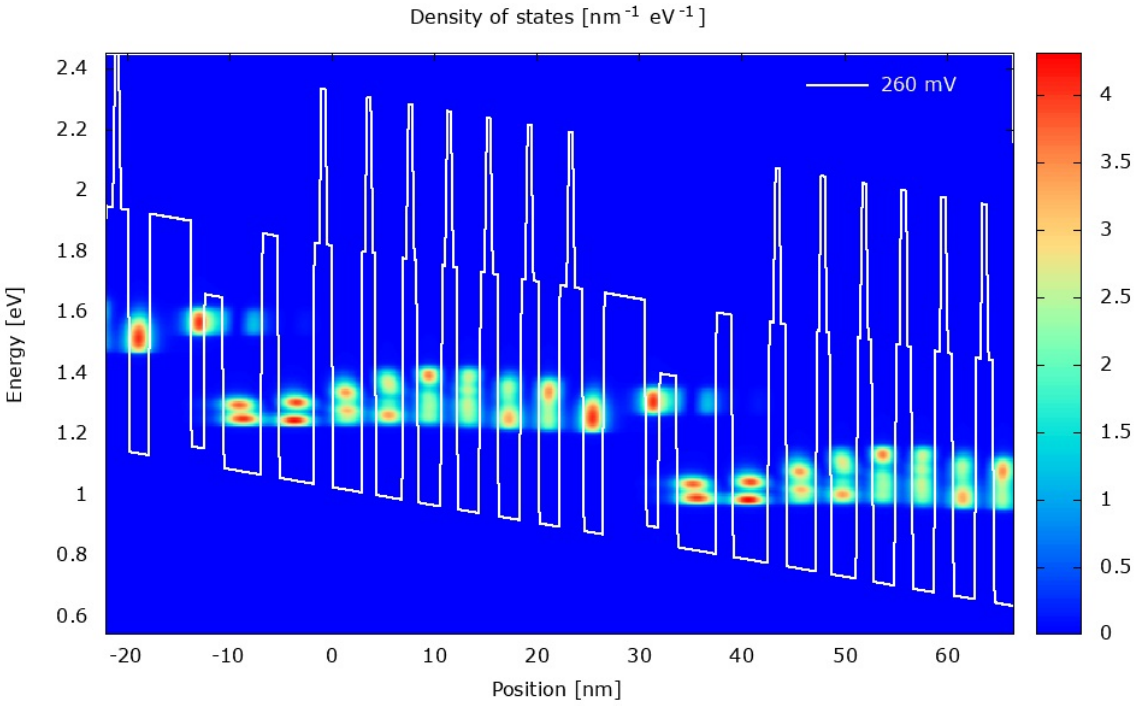
- Longitudinal polar-optical phonon scattering (polar LO phonon scattering)
- Acoustic phonon scattering
- Charged impurity scattering
- Interface roughness scattering
- Alloy scattering
- Electron-electron scattering

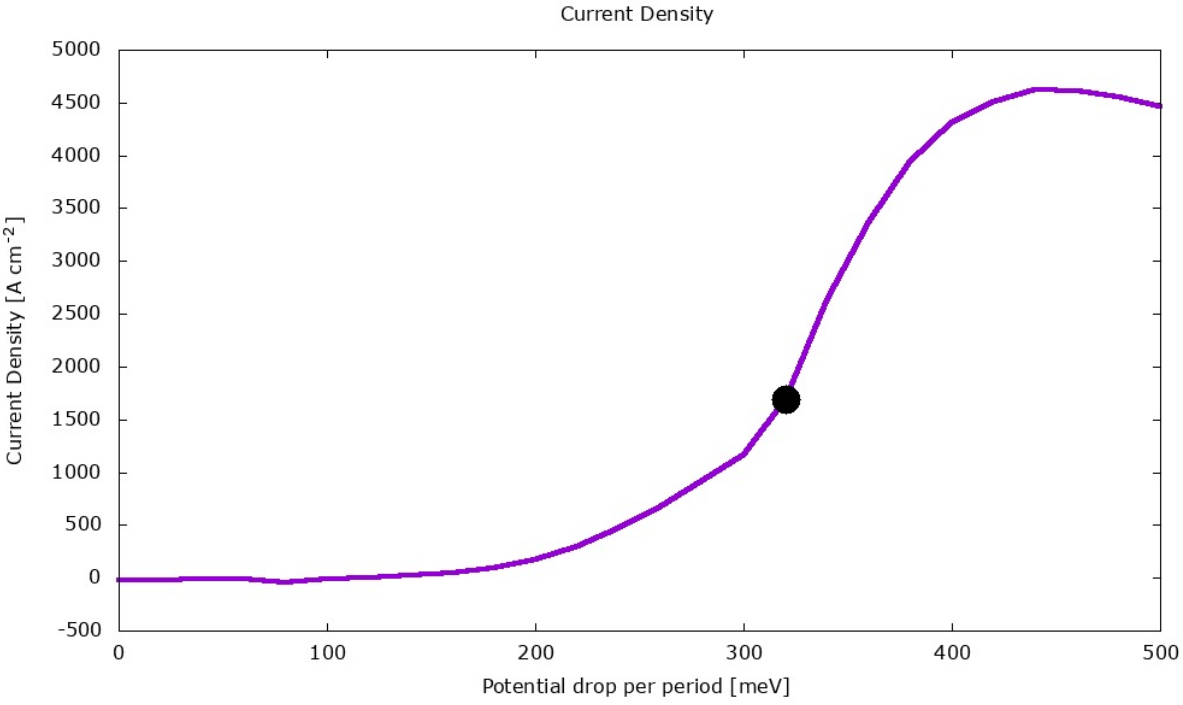
A possible application is illustrated below with the calculation of current-voltage characteristics in a QCL.

See also this video: nextnanoNEGF_video_ITQW_2017_1080p.mp4

Note: We think it makes sense to

1. **first** get familiar with the *nextnanomat* user interface (using the *nextnano++* tool) and
2. **then** have a look at *nextnano.NEGF*.


Density of states [nm -1 eV -1]
260 mV
Energy [eV]
Position [nm]
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
-20
-10
0
10
20
30
40
50
60
4
3.5
3
2.5
2
1.5
1
0.5
0


Current Density
Current Density [A cm -2]
Potential drop per period [meV]
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
-500
0
100
200
300
400
500

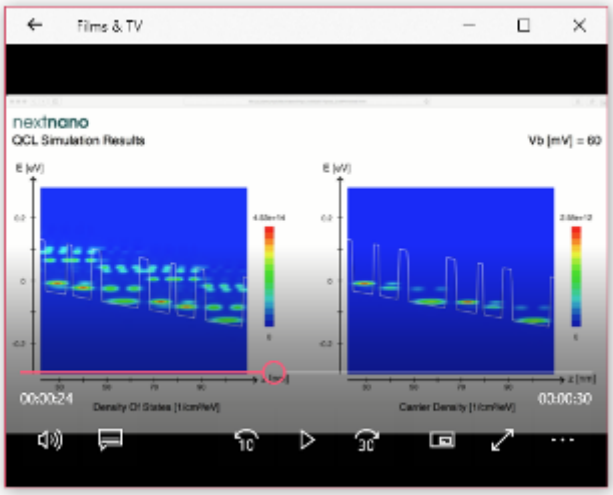


Figure 8.1.1: This video (mpg4, 23 MB) shows how a QCL works. The first 10 seconds summarize the features of our software, the remaining 20 seconds show the animated simulation results. At 20 seconds, the QCL starts to lase: The electron density (figure on the right) shows the population inversion, i.e. the higher state has a larger electron population than the lower lasing state.

You can solve the Schrödinger equation for a QCL heterostructure subject to an electric field using the *nextnano++* tool.

You can find simple QCL wave function calculation (including intersubband transition matrix elements) examples here:

```
C:\Program Files\nextnano\2020_12_09\Sample files\nextnano++ sample files\
```

Start with these input files:

- *1DQCL_simple_nnp.in* in *Simple quantum cascade structure*

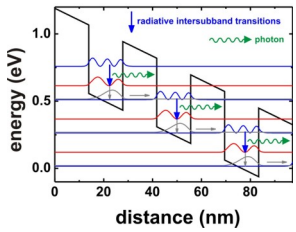


Figure 8.1.2: A toy model of a QCL scheme

- *THzQCL_Andrews_Vienna_MatSciEng2008_nnp.in* in *Quantum-Cascade Lasers*

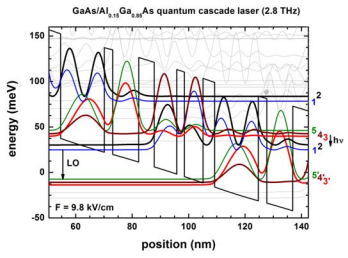


Figure 8.1.3: Square of the wave functions in a THz QCL

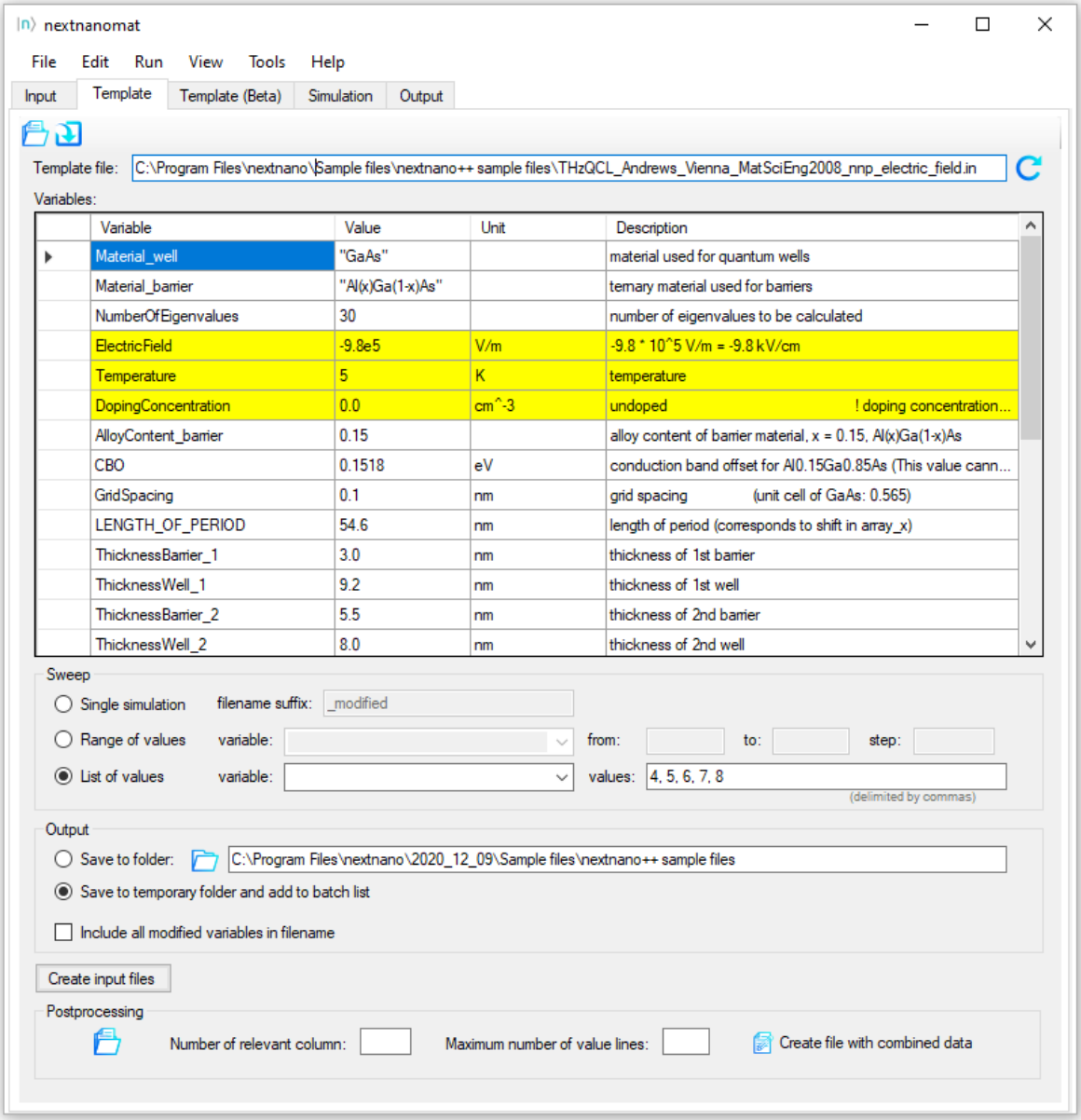


Figure 8.1.4: Template feature of *nextnanomat* for parameter sweeps.

Once you have gained some familiarity with the *nextnanomat* user interface, you can eventually start using *nextnano.NEGF*.

Note:

1. Install the nextnano software.

   Installation

2. Once done, install *nextnano.NEGF*.

You can download *nextnano.NEGF* from here:

Unzip the file, store it to a useful location on your hard disk, and set the path to the `nextnano.NEGF.exe` exectuable in *nextnanomat* ==> *Tools* ==> *Options* ==> *Simulation* ==> *nextnano.NEGF executable file*.

Please make sure to check in Windows File Explorer ==> *View* ==> *File name extensions*. If you don't check this option, you will only see:

- `nextnano.NEGF` (without file extension `.exe.`) and not
- `nextnano.NEGF.exe`

Sample input files can be found in the `\nextnano.NEGF sample files\` folder. We recommend starting with this example: Open `THz_QCL_GaAs_AlGaAs_Fathololoumi_OptExpress2012_10K-FAST.xml` and press the run button. This is *only* a test file that runs fast (typically 10 minutes). For reliable calculations, please use the corresponding file `THz_QCL_GaAs_AlGaAs_Fathololoumi_OptExpress2012-MEDIUM.xml` which takes more CPU time (~12 hours) than `*FAST.xml` but produces much more accurate results. If the calculations take too long, your CPU might not be very fast or you do not have sufficient RAM. In that case, please contact us. You need at least 16 GB RAM, 32 GB are better. Please note that you should run the larger input files on a computer having 32 GB RAM. 16 GB might be not enough and require a change in the settings for the number of parallel threads used for the gain calculation. We recommend a Windows PC with 64 GB RAM, and a recent i7 CPU (e.g. a 6-core i7-8700). The whole PC is then around USD 1,000.

If the tool does not run, you might have to specify the `Material_Database.xml` file in *Tools* ==> *Options* ==> *Material database*.

We hope you enjoy using *nextnano.NEGF*. The technical part of *nextnano.NEGF* documentation is currently being updated and migrated to here from: https://www.nextnano.com/nextnano.NEGF/

Please let us know if you encounter any difficulty or notice any strange simulation results: Contact nextnano

Note also these slides on the implemented physics: Tutorial_IQCLSW_2018_Grange

8.2 Keywords

Note: Page under construction.

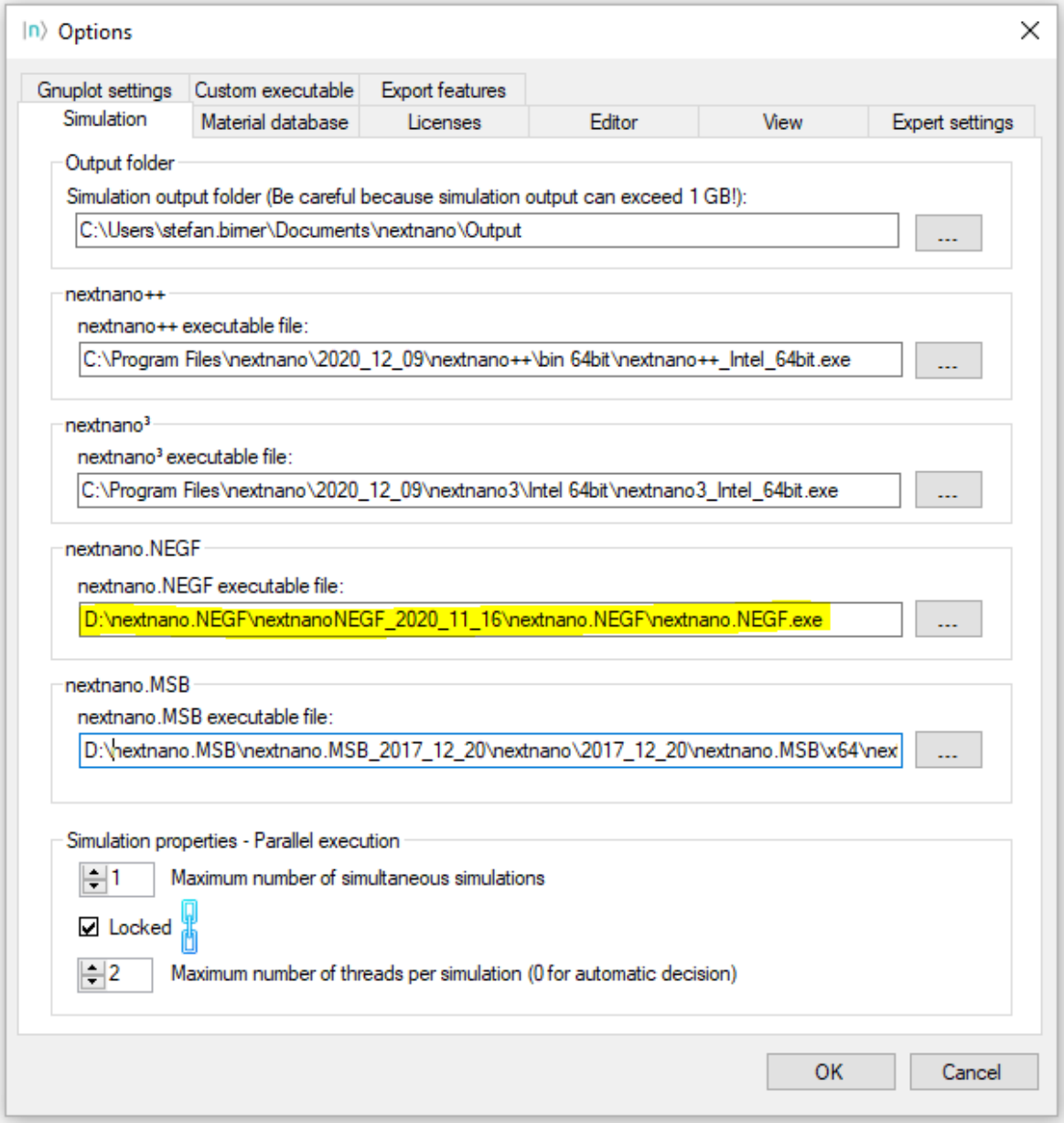


Figure 8.1.5: Set the path to the `nextnano.NEGF.exe` executable.

8.2.1 Temperature

Specifies the lattice temperature. It is used to calculate:

- Varshni shift of band edges
- Phonon bath

```
Temperature = 300  # K
```

Note: This does not affect the carrier temperature. The latter will be calculated from subband occupations once the NEGF solution has converged.

8.2.2 Bias{ }

```
Bias{
   PotentialDrop = 50  # meV
}
```

8.2.3 SweepParameters{ }

```
SweepParameters{
   SweepType =

   SchroedingerOnly =

   Min =
   Max =
   Delta =
   DeltaSmall =

   MinV =
   MaxV =
   DeltaV =

   MinT =
   MaxT =
   DeltaT =

   NThreads =
}
```

8.2.4 Equilibrium{ }

```
Equilibrium{
   Fermi =
   Broadening =
}
```

8.2.5 Crystal{ }

```
Crystal{
   CrystalStructure = "Zincblende"

   Orientation{
      zAxis{
             h = 0
                 k = 0
                 l = 1
          }

      yAxis{
             h = 0
                 k = 1
                 l = 0
          }
   }

   MaterialSubstrate{
      Name =
      AlloyComposition =
   }

   Strain = yes
   Piezoelectricity = yes
   Pyroelectricity = yes
}
```

8.2.6 Contacts{ }

```
Contacts{
   DensityLeft =
   DensityRight =

   MaterialLeft =
   MaterialRight =

   ChargeNeutral =
   DeviceNeutral =

   Broadening =
   Ballistic = no
}
```

8.2.7 Hybrid{ }

```
Hybrid{
   Broadening =
   SeparationLeft =
   SeparationRight =
   OffsetContact =
   IterPoisson =
   eeNEGF =
   Ballistic =
}
```

8.2.8 Materials{ }

```
Materials{
   Material{
      Name =
          AlloyComposition =
          Alias = "well"

      EffectiveMassFromKpParameters = no
          RescaleS = yes  # default: yes
          RescaleSTo = 0

          Overwrite{
             # This keyword directly overwrites the mat params of alloys␣
↪for the composition specified in the input file.
          }
   }

   NumberOfBands = 3
   UseConductionBandOffset =
   NonParabolicity =
   NonParabolicityRelative =
   InPlaneNonParabolicity =
   TemperatureDependentEightBandDKKParameters = no
   ValleyDegeneracy =
}
```

8.2.9 OverwriteMaterialdatabase{ }

```
OverwriteMaterialDatabase{
   Material{
      # This keyword overwrites the mat params of individual binaries
   }
}
```

8.2.10 Structure{ }

(Formerly Superlattice)

```
Structure{
   Layer{
      Material =
          MaterialLeft =
          MaterialRight =
          Thickness =
   }

   InterfaceWidth =

   Doping{
      DopingType = N  # default is N
          DopingStart =
          DopingEnd =
          DopingSpecification =
          DopingDensity =
   }

   AnalysisSeparator{
      SeparatorPosition =
   }
}
```

8.2.11 Scattering{ }

```
Scattering{
   MaterialForScatteringParameters = "well"

   InterfaceRoughness{
      InterfaceWidth =
      AmplitudeInZ =
      InterfaceAutoCorrelationType =
      CorrelationLengthInXY =
      AxialCorrelationLength =
   }

   ThreadingDislocations{
      Density =
      FillingFactor =
      TypePNotN =
   }
```

(continues on next page)

(continued from previous page)

```
   AcousticPhononScattering = yes
   AcousticPhononScatteringEnergyMax =

   MonochromaticLOPhonon =
   TuneLOPhononScattering =
   LOPhononCouplingStrength =
   LOPhononDeformationPotential =

   ScreeningTemperatureType =
   TemperatureOffsetParameter =
   AccuracySelfConsistentElectronTemperature =
   ElectronTemperatureForScreening =
   ImpurityScatteringStrength =

   AlloyScattering = yes
   AlloyScatteringStrength =

   # advanced settings
   SeparateScattering =
   AnharmonicityStrength =
   MinimizeIFRBroadening =

   ElectronElectronScattering =
   HomogeneousCoulomb =
   CoarseGridCoulomb =
   TuneElectElectScatteringStrength =
   ElectElectScatteringStrength =

   PhononDamping =

   InterfacesCutOff =
}
```

8.2.12 Poisson

```
Poisson = yes
iterationPoissonStart =
factPoisson =
```

8.2.13 LateralDiscretization{ }

(Formerly LateralMotion)

Specifies the numerical discretization for the directions perpendicular to the growth axis (*nextnano.NEGF* considers a cylindrical structure along the growth axis).

The parameters are assumed to be homogeneous along the structure, and hence must be taken from a single material. This material is indicated by `MaterialForLateralMotion`. The energy spacing between the ground and first excited in-plane modes is specified with `Value` [meV]. This determines the radius of the cylinder used for the simulation.

```
LateralDiscretization{
   MaterialForLateralMotion = "well"
   Value = 5
```

(continues on next page)

(continued from previous page)

```
   DiagonalIncoherentScattering =
}
```

Attention: There is a further parameter for the in-plane motion, `EnergyRangeLateral` in *SimulationParameter{ }*, which sets the cut-off energy (i.e. the energy range) for the subband dispersion.

Dispersion

```
LateralDiscretization{
   Dispersion{
   }
}
```

8.2.14 SimulationParameter{ }

Axial energy cut-off

```
# By energy cut-off value [meV]
SimulationParameter{
   EnergyRangeAxial = 60              # unit = meV  # z-direction, ␣
↪evaluated from ground state
   EnergyRangeAxialValence = 50       # unit = meV  # z-direction, ␣
↪evaluated from ground state
}

# By number of subbands
SimulationParameter{
   NumberOfConductionSubbands = 1
   NumberOfValenceSubbands = 2
}
```

Heterostructures consist of a large number of subbands, but only a few near the band gap are relevant for the electronics and optoelectronics (Figure 8.2.14.1 left). *nextnano.NEGF* therefore selects a limited number of subbands and construct the Green's function basis (Figure 8.2.14.1 right).

You can specify the cut-off either by the energy range measured from the ground states (`EnergyRangeAxial` and/or `EnergyRangeAxialValence`), or by the number of subbands (`NumberOfConductionSubbands` and/or `NumberOfValenceSubbands`). The number of subbands is considered to be without the spin degree of freedom.

Lateral energy cut-off

`EnergyRangeLateral` limits the energy range of the in-plane dispersion. The energy reference is the bottom of the ground states.

```
SimulationParameter{
    EnergyRangeLateral = 200           # unit = meV  # xy-direction,␣
↪evaluated from ground state
}
```

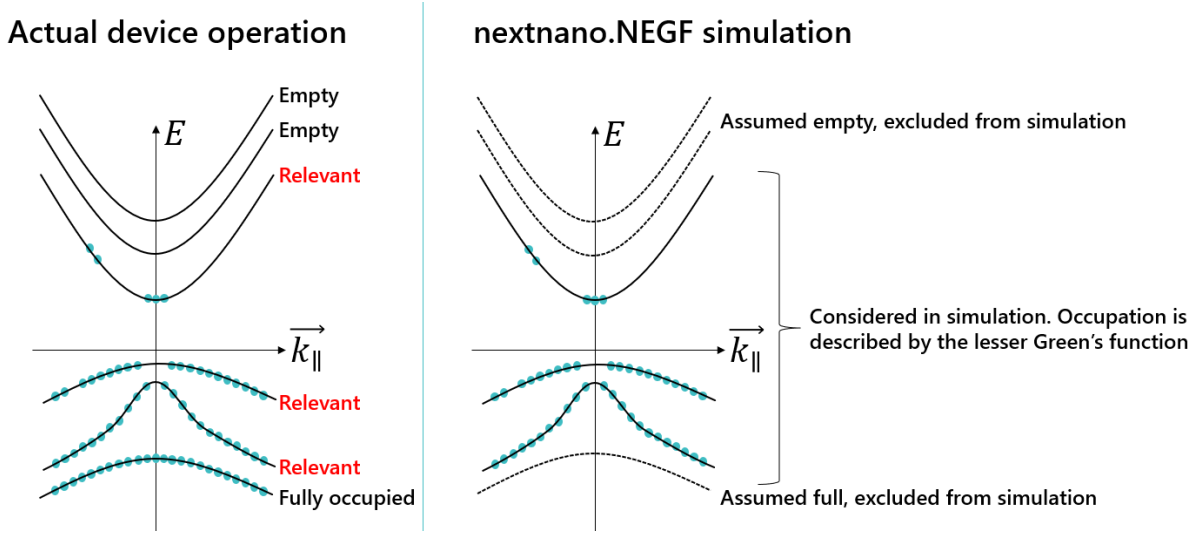


Figure 8.2.14.1: Schematics of miniband selection in the "electron picture", in which the valence band is considered to be almost fully-occupied sea of electrons.

8.2.15 Output{ }

```
Output{
   EnergyResolvedPlots =
   EnergyResolvedPlotsMinimumEnergyGridSpacing =
   EnergyResolvedPlotsFullRealSpaceResolutionForCurrent = yes  # default:␣
↪yes

   EnergyResolvedGain =
   EnergyResolvedGainForEachPhotonEnergy =

   PlotDispersion =

   FLDFormat =
   VTKFormat =
   GnuplotFormat =

   ScaleWaveFunction =
   SXYZBasis = no  # for 8-band k.p model
}
```

8.2.16 Gain{ }

```
Gain{
              # NEGF gain
              GainMethod =
              dEPhot =
              EphotonMin =
              EphotonMax =

              dEPhotSelfConsistent =
              EphotonMinSelfConsistent =
              EphotonMaxSelfConsistent =
              NMaxSelfConsistentIterations =
```

(continues on next page)

(continued from previous page)

```
            ConvergenceFactorGainSelfConsistent =

            SelfConsistentBoundary =

            Gauge = 1  # 0 for Lorenz, 1 for Coulomb Gauge (faster)

            Vmin =
            Vmax =

            CavityLosses =
            GainClamping =
            OverlapFactor =
            FrontMirrorLosses =

            Threshold =
            RefineThreshold =

            # Semiclassical gain
            Linewidth =
            LinewidthMin =

        }
```

Polarization

```
Gain{
   Polarization{
      # TE polarization
      Re = [1, 0, 0]
      Im = [0, 0, 0]
   }
}
```

When `GainMethod = 'semiclassic'`, `Polarization` specifies the electric field polarization used for Fermi's golden rule. The simulation z axis is along the growth direction.

8.2.17 SurfaceRoughness{ }

```
SurfaceRoughness{
   Amplitude =
   CorrelationLength =
}
```

8.2.18 EMfield{ }

Specifies the electromagnetic field used in photo-assisted transport calculation.

- *EMmode*

EMmode

```
EMmode{
   PhotonEnergy = 160   # meV
   ElectricField =
   Intensity =
}
```

8.3 Input Syntax

Note: Page under construction. The input syntax can be found at: https://nextnano-docu.northeurope.cloudapp.azure.com/dokuwiki/doku.php?id=qcl:input_file

Two formats are supported by *nextnano.NEGF*. * the .xml format (supported in all versions) * the .negf format (similar style as *nextnano++* .in format, available only in the new C++ version)

The new format with **extension 'negf'** features:

- **Syntax validation before the simulation** - the parser checks the syntactical correctness as well as the logic and, if invalid, prints an error message in the log specifying what exactly is wrong in your file.
- Variables starting with `$` sign can be used not only in the input but also database files. The new default database *Material_Database.negf* contains a switch `$NEGF` / `$MSB` / `$nnp` to use the default parameters of *nextnano.NEGF*, *nextnano.MSB*, and *nextnano++*, respectively. Naturally, the new format will look much familiar if you have experience with *nextnano++*.
- **The *nextnanomat* feature *Tools > Convert .xml Input File to .negf Input File* can convert your xml input files to the new format. On Linux, you can run the executable *syntax_converter* included in the package. After conversion, please note that**
  - Most of the XML comments `<!-- -->` are not retained in the conversion. If necessary, please recover them manually. Comments start with the letter `#`.
  - Within the new format, only the new input file keywords are supported. They are replaced automatically by the conversion.
  - IMPORTANT: **Do not convert the XML database to the new format**! The new database *Material_Database.negf* is not backward-compatible and contains extra parameters compared to *Material_Database.xml*.

8.4 Material Database

Note: Coming Soon

Two formats are supported by *nextnano.NEGF*. * the .xml format (supported in all versions) * the .negf format (similar style as *nextnano++* .in format, available only in the new C++ version)

IMPORTANT: The C# and C++ versions of *nextnano.NEGF* need different databases, even in the .xml format. **Please use the ones included in the folder of the executable**. IMPORTANT: **Do not convert the XML database to the new format**! The new database *Material_Database.negf* is not backward-compatible and contains extra parameters compared to *Material_Database.xml*. *Material_Database.negf* is compatible with *nextnano.MSB*.

8.5 Tutorials

Here we describe several example input files.

8.5.1 THz QCLs

GaAs/AlGaAs

- THz QCL - Fatholoumi (2012)
  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, H. C. Liu
  Optics Express 20, 3866 (2012)
- GaAs/Al0.15Ga0.85As terahertz quantum cascade lasers with double-phonon resonant depopulation operating up to 172 K
  R. W. Adams, K. Vijayraghavan, Q. J. Wang, J. Fan, F. Capasso, S. P. Khanna, A. G. Davies, E. H. Linfield, M. A. Belkin
  Applied Physics Letters 97, 13111 (2010)
- Influence of doping on the performance of terahertz quantum-cascade lasers
  A. Benz, G. Fasching, A. M. Andrews, M. Martl, K. Unterrainer, T. Roch, W. Schrenk, S. Golka, G. Strasser
  Applied Physics Letters 90, 101107 (2007)
- 1.9 THz quantum-cascade lasers with one-well injector
  S. Kumar, B. S. Williams, Q. Hu
  Applied Physics Letters 88, 121123 (2006)
- Far-infrared ($\lambda \simeq 87\mu m$) bound-to-continuum quantum-cascade lasers operating up to 90 K
  G. Scalari, L. Ajili, J. Faist, H. Beere, E. Linfield, D. Ritchie, G. Davies
  Applied Physics Letters 82, 3165 (2003)
- Broadband THz lasing from a photon-phonon quantum cascade structure
  G. Scalari, M. I. Amanti, C. Walther, R. Terazzi, M. Beck, J. Faist
  Optics Express 18, 8043 (2010)

InGaAs/AlGaSb

- High performance InGaAs/GaAsSb terahertz quantum cascade lasers operating up to 142 K
  C. Deutsch, M. Krall, M. Brandstetter, H. Detz, A. M. Andrews, P. Klang, W. Schrenk, G. Strasser, K. Unterrainer
  Applied Physics Letters 101, 211117 (2012)

8.5.2 Mid-IR QCLs

InGaAs/AlInAs

- Mid-IR QCL - Yu Slivken Razeghi
  Injector doping level-dependent continuous-wave operation of InP-based QCLs at $\lambda = 7.3\mu m$ above room temperature
  J. S. Yu, S. Slivken, M. Razeghi
  Semiconductor Science and Technology 25, 125015 (2010)

GaAs/AlGaAs

- 300 K operation of a GaAs-based quantum-cascade laser at $\lambda \simeq 9\mu m$
  H. Page, C. Becker, A. Robertson, G. Glastre, V. Ortiz, C. Sirtori
  Applied Physics Letters 78, 3529 (2001)

8.5.3 AlGaAs/GaAs RTD

This tutorial describes the *nextnano.NEGF* simulation of AlGaAs/GaAs resonant tunneling diode (RTD).

Sample input files are available from the sample file folder:

- *RTD_Example_withScattering.xml*
- *RTD_Example_ballistic.xml*

The first input file takes the scattering mechanisms into consideration, while the second input file ignores them.

Table of contents

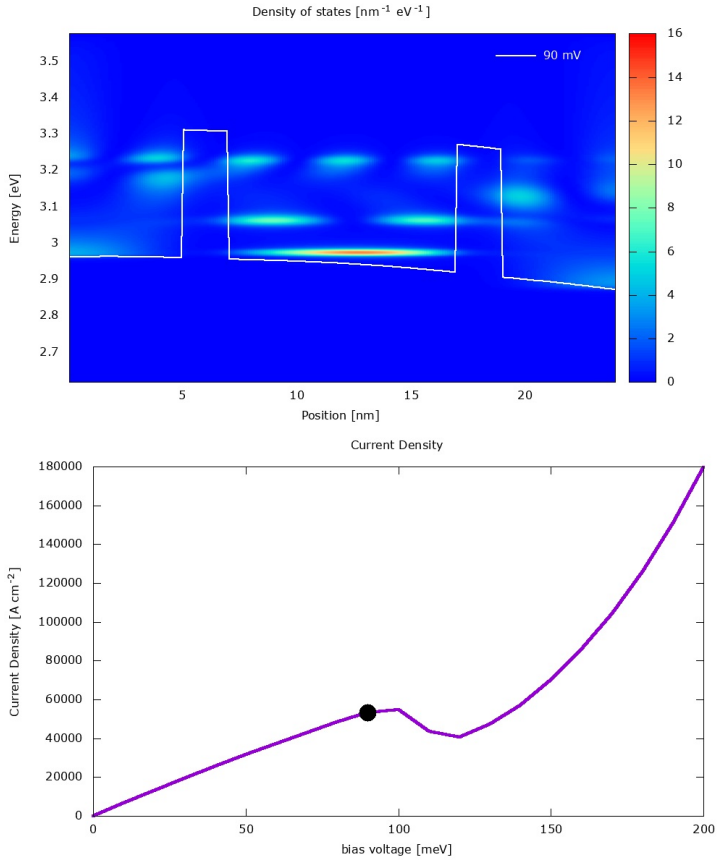


Figure 8.5.3.1: Top: Local density of states calculated from *RTD_Example_withScattering.xml*. The structure consists of GaAs and two AlGaAs barriers. Down: Current-voltage characteristics calculated from *RTD_Example_withScattering.xml*.

Simulation input

Definition of materials

At first, the materials used in the structure have to be defined. Each material is referred by an alias, which is here "well" for GaAs and "barrier" for AlGaAs.

```
  <Materials>

      <Material>
          <Name>GaAs</Name>
          <Alias>well</Alias>
          <Effective_mass_from_kp_parameters>yes</Effective_mass_from_kp_parameters>
      </Material>

      <Material>
          <Name>Al(x)Ga(1-x)As</Name>
          <Alloy_Composition>0.4</Alloy_Composition>
          <Alias>barrier</Alias>
          <Effective_mass_from_kp_parameters>yes</Effective_mass_from_kp_parameters>
      </Material>

      <!-- Model nonparabolicity -->
      <NonParabolicity>yes</NonParabolicity>
      <Number_of_bands>1</Number_of_bands>

      <UseConductionBandOffset>yes</UseConductionBandOffset>

</Materials>
```

It is specified that the effective mass is calculated from the k.p parameters (<Effective_mass_from_kp_parameters>). Also, the effective mass is assumed as energy dependent (<NonParabolicity>) and the model used for the calculation of effective mass is the one for single-band model (<Number_of_bands>).

For the details of the syntaxes of this block, please refer to this page: Syntax of the input file -> Material definition and parameters . The models used for the calculation of the effective mass are described here: Electronic Band Structure .

Defintion of layers

Next, alternating layers consisting of barrier and well have to be specified. In this tutorial, the thickness of each layer is 5.0/**2.0**/10.0/**2.0**/5.0 (nm) where AlGaAs barrier layer is in bold fonts.

```
<Superlattice>

    <Layer>
        <Material>well</Material>
        <Thickness unit="nm">5.0</Thickness>
    </Layer>

    <Layer>
        <Material>barrier</Material>
        <Thickness unit="nm">2.0</Thickness>
```

(continues on next page)

(continued from previous page)

```
    </Layer>

    <Layer>
        <Material>well</Material>
        <Thickness unit="nm">10.0</Thickness>
    </Layer>

    <Layer>
        <Material>barrier</Material>
        <Thickness unit="nm">2.0</Thickness>
    </Layer>

    <Layer>
        <Material>well</Material>
        <Thickness unit="nm">5.0</Thickness>
    </Layer>


    <Analysis_Separator>
        <Separator_Position>6.0</Separator_Position>
        <!-- collector barrier -->
    </Analysis_Separator>

    <Analysis_Separator>
        <Separator_Position>18.0</Separator_Position>
        <!-- injector barrier -->
    </Analysis_Separator>

</Superlattice>
```

The resulting conduction band edge profile can be found in the file called *Conduction_BandEdge.dat*. This file includes the (small) band bending due to the electrostatic potential. At a bias voltage of 0 mV, it looks as follows:

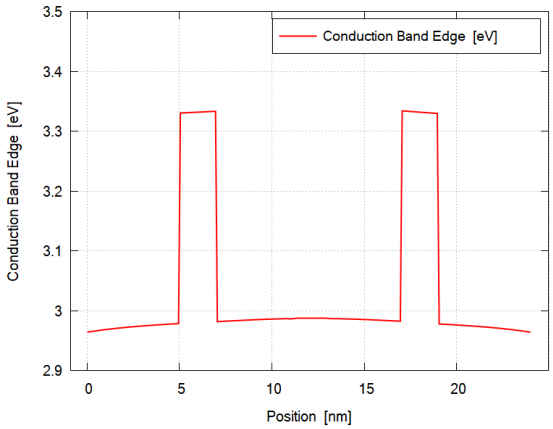


Figure 8.5.3.2: Conduction band edge at a zero bias voltage. This is obtained from *RTD_Example_withScattering.xml*.

Contacts (open boundary condition)

In order to simulate a system with open boundary conditions (instead of the default field-periodic boundary condition), contacts have to be defined by adding a <Contacts> section into the input file.

In *RTD_Example_withScattering.xml* this section is specified as follows:

```
<Contacts>

    <DensityLeft unit="cm^-3">1e18</DensityLeft>
    <DensityRight unit="cm^-3">1e18</DensityRight>

    <MaterialLeft>well</MaterialLeft>
    <MaterialRight>well</MaterialRight>

    <Broadening unit="meV">10.0</Broadening>
    <Ballistic>no</Ballistic>

</Contacts>
```

The carrier densities in the left and right contact have to been defined using the <DensityLeft> and <DensityRight> commands, as shown above. The unit is [cm^{-3}].

The material of the left and right contacts needs to be defined by the command <MaterialLeft> and <MaterialRight>. The string value has to be an alias defined in the <Materials> section.

A broadening energy can be defined by the command <Broadening>. Indeed, scattering is not accounted in the contact, so that this command allows a phenomenological broadening of the density of states in the contact.

The command <Ballistic> can be used to calculate ballistic transport between the contacts (i.e. no scattering process considered) if its value is set to yes. This is the case of *RTD_Example_ballistic.xml*.

Note: In the current version (2020-11-19), only single band calculations are supported for open boundary conditions.

Scattering

The detailed description of the sytaxes for each scattering mechanism is described here: Syntax of the input file -> Scattering processes .

The following scattering mechanisms are included in this tutorial.

- Interface roughness scattering

```
<Interface_Roughness>
<Amplitude_in_Z unit="nm">0.1</Amplitude_in_Z>
    <InterfaceAutoCorrelationType>0</InterfaceAutoCorrelationType> <!-- Correlation␣
↪type: 0=Exponential, 1=Gaussian -->
<Correlation_Length_in_XY unit="nm">8</Correlation_Length_in_XY>
</Interface_Roughness>
```

- Acoustic phonon scattering

```
<!-- Acoustic phonons -->
<Acoustic_Phonon_Scattering>no</Acoustic_Phonon_Scattering> <!-- Comment: Acoustic␣
↪phonons are in general not efficient - can be neglected in most cases -->
<AcousticPhonon_Scattering_EnergyMax unit="meV">3.0</AcousticPhonon_Scattering_
↪EnergyMax> <!-- Maximum acoustic phonon energy -->
```

- Charged impurity scattering

```
<!-- Charged impurities -->

<!-- Effective temperature of the electrons involved in electrostatic screeening: 3␣
↪models available -->
<!-- model #1: Teff = T + Toffset * exp(-T/Toffset) with Toffset specified as␣
↪Temperature_Offset_parameter  -->
<!-- model #2: self-consistent calculation (requires several iterations of the all␣
↪calculation). The ccuracy specified by Accuracy_Self_consistent_Electron_
↪Temperature -->
<!-- model #3: Teff is directly specified by Electron_Temperature_for_Screening-->
<Model_Temperature_for_Screening>1</Model_Temperature_for_Screening> <!-- integer 1,2␣
↪or 3 is required accordingly to the desired model -->

<Temperature_Offset_parameter>150</Temperature_Offset_parameter> <!-- enter Toffset␣
↪for model#1 only such as Teff = T + Toffset * exp(-T/Toffset) -->
<Accuracy_Self_consistent_Electron_Temperature>0.05</Accuracy_Self_consistent_
↪Electron_Temperature>  <!-- for model #2 only: self-consistent calculation until␣
↪the effective temperature convergences below the desired accuracy-->
<Electron_Temperature_for_Screening>200</Electron_Temperature_for_Screening> <!-- for␣
↪model#3 only: the effective temperature is directly specified -->


<ImpurityScattering_Strength>1</ImpurityScattering_Strength> <!-- 1.0 is the normal␣
↪physical calculation. Other values may be used for testing the importance of␣
↪impurity scattering. -->
```

- Electron-electron scattering

```
<Electron_Electron_Scattering>yes</Electron_Electron_Scattering>
```

- Alloy scattering

```
<Alloy_scattering>yes</Alloy_scattering>
```

(*Advanced*) In this sample input file, the Coulomb scatterers (ionized impurities and other charge carriers) are assumed to be homogeneously distributed in order to speed up the calculation.

```
<Homogeneous_Coulomb>yes</Homogeneous_Coulomb>
```

Note: LO-phonon scattering is applied by default as all materials have intrinsic parameters controlling the electron-phonon interaction. We can modify this default behavior by this command.

Poisson equation

```
<Poisson>yes</Poisson>
```

If `yes`, the Poisson equation is included in the program flow. Then the electrostatic mean-field interactions (electron-electron and electron-impurities interactions) can be taken into account.

Lateral motion

```
<Lateral_motion>
    <Material_for_lateral_motion>well</Material_for_lateral_motion>
    <!-- Lateral energy spacing  -->
    <Value unit="meV">10</Value>
</Lateral_motion>
```

In this sample file, the parameters for the lateral motion (i.e. the two-dimensional free motion in the directions perpendicular to the growth axis) are taken from the material of `well`, i.e. GaAs.

Also, the discretization energy for this lateral motion is specified as 10 meV.

Simulation output

Electron eigen states

The electron eigenstates calculated for the whole region biased for each voltage is written in *WannierStark/WannierStark_states.dat*. These states are used as the basis states of the Green's function.

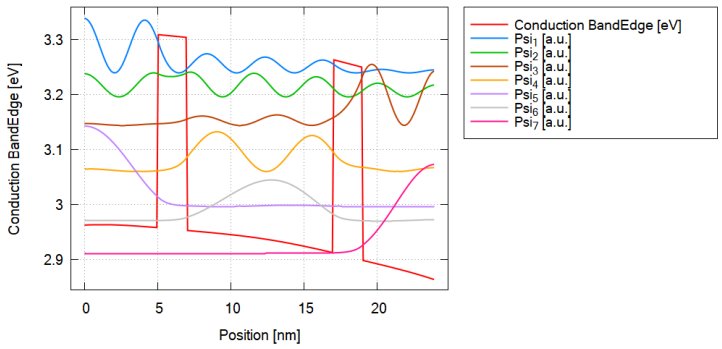


Figure 8.5.3.3: Electron eigenstates for the whole region at the bias of 100 mV.

Note: There is another output *WannierStark/TightBinding_states.dat* that describes the electron eigenstates confined in each GaAs region. These eigenstates are calculated from the Schrödinger equations for each region separated by `<Analysis_Separator>` in `<Superlattice>` section.

For the detailed explanation, please refer to here .

Local density of states

The following figures show the local density of states (LDOS) for the bias of 100 mV written in *2D_plots/DOS_energy_resolved.vtr*. Please note that the scaling of the colormap is different in the two figures. The gnuplot file which generates the gif animation is also available in the top directory as *Animation_DensityOfStates.plt*.

The LDOS tells us where and at which energy electronic states are available that the charge carriers can occupy. The LDOS is shown for $\mathbf{k}_{||} = 0$, i.e. there are also electronic states available for $\mathbf{k}_{||} \neq 0$. But they are not shown in this plot because then the picture could not show the minimum energy of each subband , which is at $\mathbf{k}_{||} = 0$, so nicely.

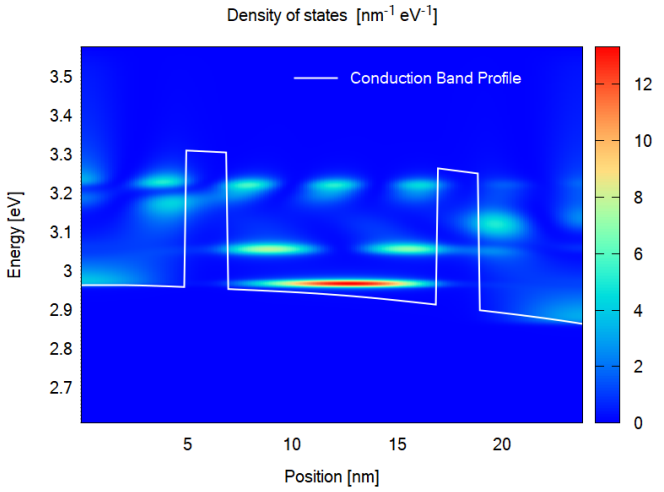


Figure 8.5.3.4: Local density of states calculated with scattering at the bias of 100 mV.

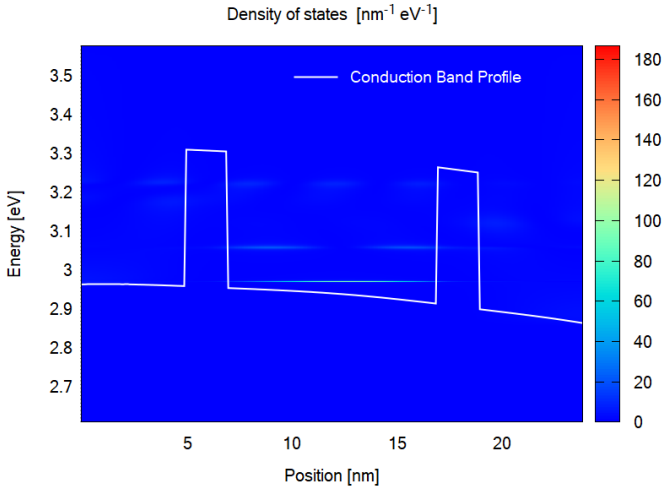


Figure 8.5.3.5: Local density of states calculated with the ballistic condition at the bias of 100 mV.

Note: The gnuplot files that generate the above figures are *DOS_energy_resolved.plt*. The file explorer in *nextnanomat* doesn't show .plt file so please access through your default file explorer.

Electron density

The following figures show the energy resolved electron density $n(x, E)$ for the bias of 100 mV written in *2D_plots/CarrierDensity_energy_resolved.vtr*. Please note that the scaling of the colormap is different in the two figures. The gnuplot file which generates the gif animation is also available in the top directory as *Animation_CarrierDensity.plt*.

The electron density is obtained from occupying the LDOS (for both $\mathbf{k}_{\parallel} = 0$ and $\mathbf{k}_{\parallel} \neq 0$) with charge carriers. It is a non-equilibrium occupation that is not described by a Fermi distribution.

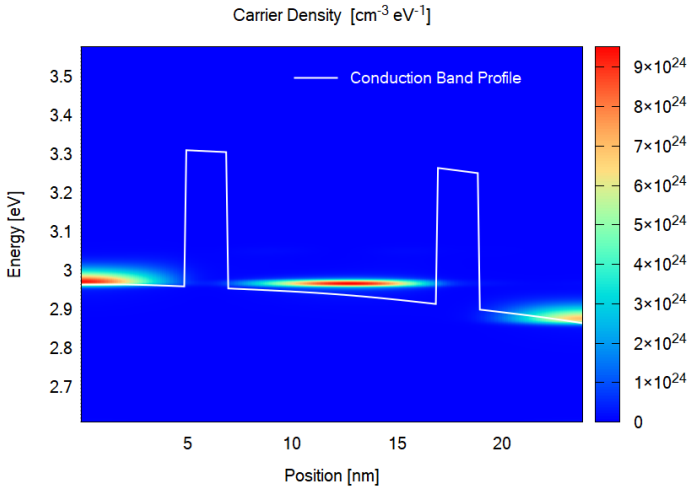


Figure 8.5.3.6: Electron density calculated with scattering at the bias of 100 mV.

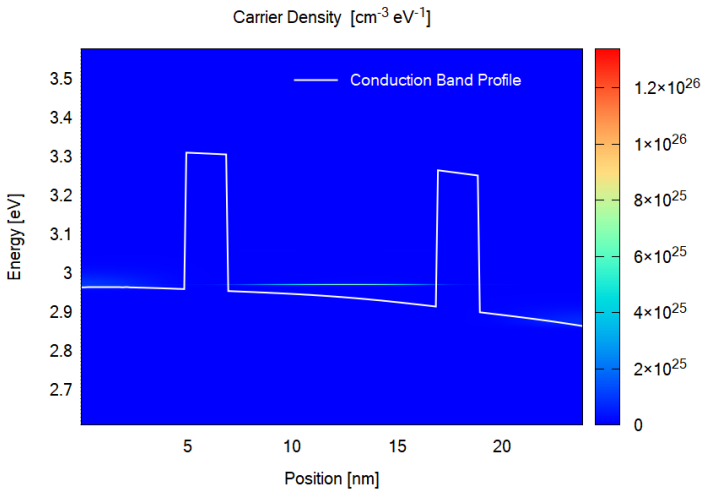


Figure 8.5.3.7: Electron density calculated with the ballistic condition at the bias of 100 mV.

Current density

The following figures show the energy resolved current density $j(x, E)$ for the bias of 100 mV written in *2D_plots/CurrentDensity_energy_resolved.vtr*. Please note that the scaling of the colormap is different in the two figures. The gnuplot file which generates the gif animation is also available as *Animation_CurrentDensity.plt*.

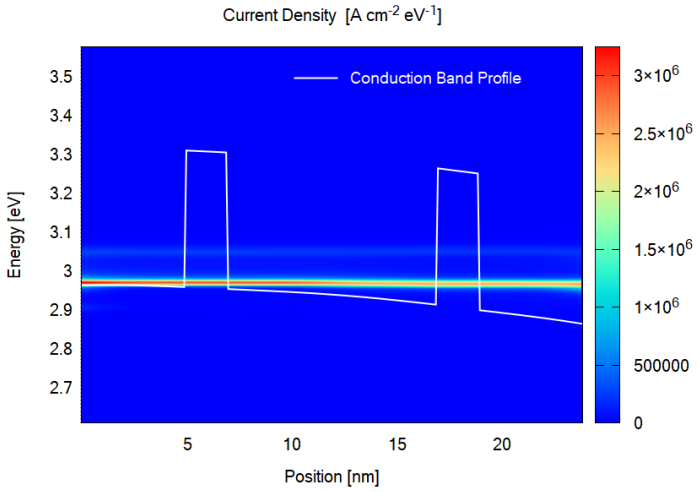


Figure 8.5.3.8: Current density calculated with scattering at the bias of 100 mV.

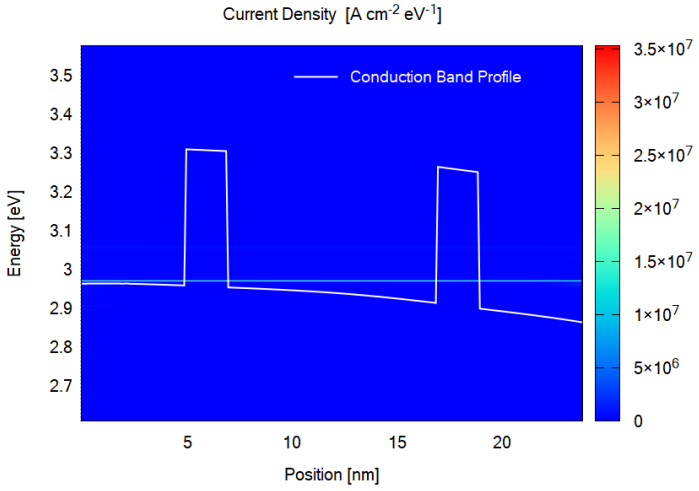


Figure 8.5.3.9: Current density calculated with the ballistic condition at the bias of 100 mV.

Current-voltage characteristics

The following figure shows the current-voltage characteristics calculated both with and without scattering. These results are taken from *Current_vs_Voltage.dat*.

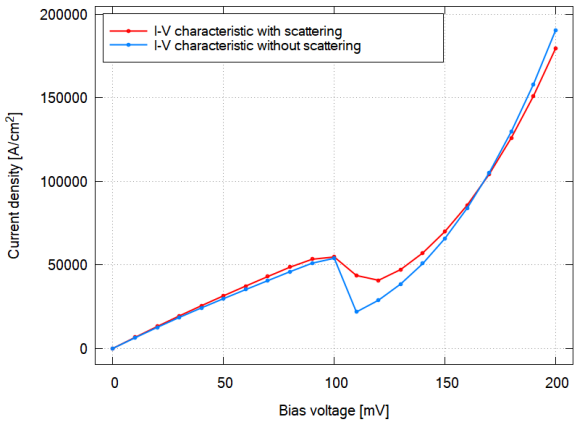


Figure 8.5.3.10: Current-voltage calculated with and without scattering.

8.5.4 Interband cascade laser: gain enhancement by carrier-rebalancing

Attention: This tutorial is under construction.

Summary

We simulate an interband cascade laser (ICL) design of [*VurgaftmanNatComm2011*] under quasi-equilibrium assumption, i.e., assuming that the interband transition is much slower than the intraband counterpart. The calculation includes the band bending due to carrier concentration and reproduces the carrier rebalancing by heavily doping the electron injector region, which was experimentally demonstrated in the paper. Our gain calculation based on Fermi's golden rule confirms the resulting gain enhancement.

Files for the tutorial located in *nextnano.NEGF\examples\quasi_equilibrium_ICLs*:

- *ICL_zb_InAs-GaInSb-GaSb-AlSb_Vurgaftman_NatureComm_2011_quasi-equilibrium_moderateDoping_negf.negf*
- *ICL_zb_InAs-GaInSb-GaSb-AlSb_Vurgaftman_NatureComm_2011_quasi-equilibrium_heavyDoping_negf.negf*
- *plot_gain_*.plt*

To produce the 1D plots (Figure 8.5.4.4, Figure 8.5.4.5, Figure 8.5.4.6, Figure 8.5.4.7), please place the corresponding plt (Gnuplot) file to the root of your output directory and run it.

Relevant output files:

- *Init\EnergyEigenstatesFull0V\EigenStates.dat* Probability distribution of eigenstates of full Hamiltonian
- *(Bias)mV\EnergyEigenstates\EigenStates.dat* Probability distribution of eigenstates recovered from the reduced real space basis, including bias and electrostatic potential
- *(Bias)mV\Gain\Semiclassical_vs_Energy_(light polarization).dat* Gain spectrum for each light polarization
- *(Bias)mV\Gain\SemiclassicalDecomposed_vs_Energy_(light polarization).dat* Gain spectrum for each transition and light polarization
- *(Bias)mV\Gain\Semiclassical_vs_Wavelength_(light polarization).dat* Gain spectrum for each light polarization
- *(Bias)mV\2D_plots\DensityOfStates_ZoneCenter.vtr* Local density of states at the smallest in-plane momentum
- *(Bias)mV\2D_plots\DensityOfStates_WithDispersion.vtr* Local density of states summed over in-plane momentum
- *(Bias)mV\2D_plots\ElectronHoleDensity_ZoneCenter.vtr* Energy-resolved carrier densities at the smallest in-plane momentum
- *(Bias)mV\2D_plots\ElectronHoleDensity_WithDispersion.vtr* Energy-resolved carrier densities summed over in-plane momentum
- *(Bias)mV\FermiLevels.dat* Quasi-Fermi levels and the energy border for distinguishing electrons and holes

Models

8-band $\mathbf{k}\cdot\mathbf{p}$ model

An 8-band $\mathbf{k}\cdot\mathbf{p}$ model is used to find energy eigenstates in the heterostructure grown along the simulation z axis. The non-equilibrium Green's function (NEGF) basis is constructed from them (see *SimulationParameter{ }* for details).

Quasi-equilibrium assumption

In the mode space obtained above, we solve the NEGF equations and Poisson equation self-consistently. Here, we assume that the interband transition in the recombination region (W-shaped quantum well) is much slower than the intraband process. This allows us to define constant quasi-Fermi levels for electrons and holes (Figure 8.5.4.1). They are split by the potential drop per period, which corresponds to device operation at 100% voltage efficiency.

Semiclassical gain

We treat carriers by non-equilibrium Green's functions but consider light as classical electromagnetic field. We restrict ourselves to the dipole approximation, which is valid for submicron period length and mid-infrared light.

The electric field amplitude is considered perturbative. From Fermi's golden rule, we calculate the absorption spectrum, which is (number of photons absorbed per unit volume per unit time) / (number of photons injected per unit area per unit time), for given electric field polarization $\vec{\epsilon}$ and photon energy $\hbar\omega$ *[ChuangOpto1995]*:

$$\alpha(\vec{\epsilon},\omega) = \frac{\pi e^2}{c\varepsilon_0\sqrt{\varepsilon(\omega)}m_0^2\omega}\frac{1}{V}\sum_{n>m}\sum_{\mathbf{k}_\parallel}|\vec{\epsilon}\cdot\vec{\pi}_{nm}(\mathbf{k}_\parallel)|^2[f_m(\mathbf{k}_\parallel)-f_n(\mathbf{k}_\parallel)] \times\frac{1}{\sqrt{2\pi}\sigma}\exp\left\{\left[-\frac{[E_n(\mathbf{k}_\parallel)-E_m(\mathbf{k}_\parallel)-\hbar\omega]^2}{2\sigma^2}\right]\right\}$$

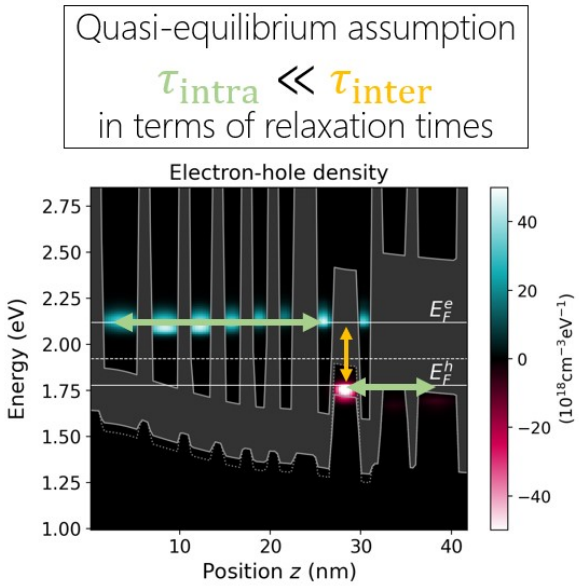


Figure 8.5.4.1: Quasi-equilibrium assumption is the approximation that the electrons and holes thermalize separately in the electron and hole injectors, respectively, due to interband processes being much slower than the intraband ones.

where e, c, m_0 are the elementary charge, vacuum speed of light, and bare electron mass, respectively. ε_0 and $\varepsilon(\omega)$ are the vacuum permittivity and dielectric function at the photon frequency. The normalization volume V is the one of a cylinder with the length of one period and radius determined internally from `LateralDiscretization{ Value }` (see *LateralDiscretization{ }*). $\vec{\pi}_{nm}$ is the in-plane wavevector-dependent momentum matrix elements in the eigenstates of the (reduced) Hamiltonian (see *SimulationParameter{ }* for the mode-space approach in *nextnano.NEGF*). The summation is taken over all possible transitions with positive photon energy. Due to the dipole approximation, the formula involves only the vertical transitions. The standard deviation of the Gaussian distribution is calculated from the input parameter `Equilibrium{ Broadening }` which we denote here by Γ (meV):

$$\sigma = \frac{\Gamma}{2\sqrt{2\log 2}} \tag{8.5.4.1}$$

nextnano.NEGF outputs the minus of the absorption spectrum, i.e., gain spectrum in the folder *(Bias)mV\Gain*.

Input file settings

`Equilibrium{Broadening}` specifies the linewidth of subbands. This phenomenological input parameter replaces scattering calculation. If `Equilibrium{SplitFermi = yes}`, quasi-Fermi levels for electrons and holes are set to match the potential drop per period.

```
Equilibrium{
    Broadening = 30.0
    SplitFermi = yes
}
```

Attention: We are extending the scattering calculation implemented for 1,2, and 3-band models to 8-band. Once implemented, the phenomenological broadening parameter will not be needed.

8-band $\mathbf{k}\cdot\mathbf{p}$ model is used to find energy eigenstates in heterostructure. The NEGF basis is constructed from them.

```
Materials{
    NumberOfBands = 8
}
```

We calculate gain spectrum using Fermi's golden rule, which we call 'semiclassical gain'. Multiple polarizations can be specified for one simulation.

```
Gain{
    GainMethod = "semiclassic"
    Linewidth = 30

    Polarization{
        Re = [1,0,0]
    }
    Polarization{
        Re = [0,0,1]
    }
}
```

To output in-plane k-integrated densities under the *(Bias)mV\2D_plots* folders, we use

```
Output{
    PlotDispersion = 2
}
```

Simulation results

Figure 8.5.4.2 shows the local density of states in one period.

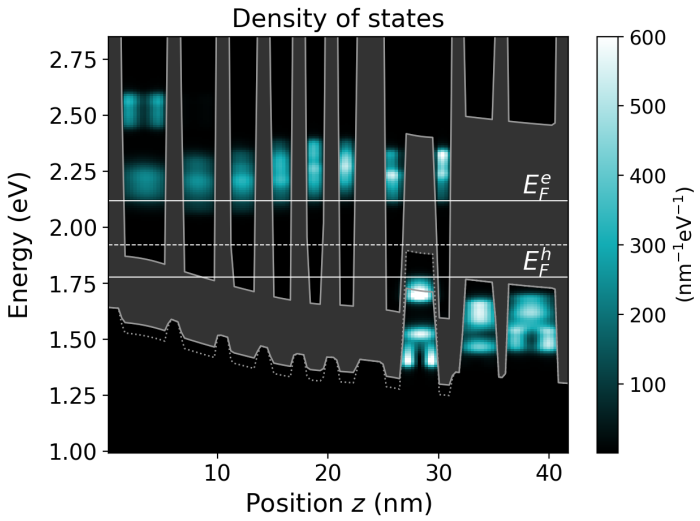


Figure 8.5.4.2: Band edge profile (grey lines), local density of states (colormap), quasi-Fermi levels (white solid lines) and the energy border for distinguishing electrons and holes (dashed white line). The heavy-hole band edge (dotted) is beyond the light-hole (solid) in the $\mathrm{Ga_{0.65}In_{0.35}Sb}$ hole well. (Run *ICL_zb_InAs-GaInSb-GaSb-AlSb_Vurgaftman_NatureComm_2011_quasi-equilibrium_heavyDoping_negf.negf* to reproduce.)

The two input files *ICL_zb_InAs-GaInSb-GaSb-AlSb_Vurgaftman_NatureComm_2011_quasi-equilibrium_*_negf.negf* differ only in the doping profile. 'Moderate doping' and 'Heavy doping' refer to ... in [*VurgaftmanNatComm2011*]. Figure 8.5.4.3 clearly shows the carrier rebalancing in the W-shaped quantum well.

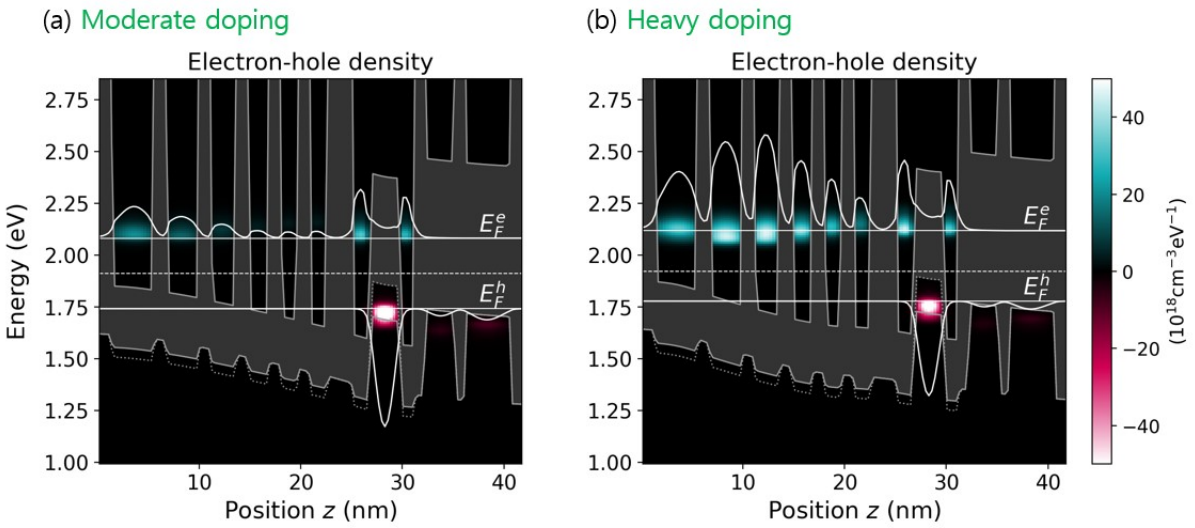


Figure 8.5.4.3: Energy-resolved electron and hole densities for the (a) reference and (b) optimally-doped structures at the potential drop per period of 340 mV. White curves indicate the electron and hole densities integrated above and below the border energy (dashed line). They were scaled by a common factor for (a) and (b).

Furthermore, *nextnano.NEGF* predicts that this carrier rebalancing enhances the gain at target photon energies (Figure 8.5.4.4). This is in agreement with the experimental observation [*VurgaftmanNatComm2011*].

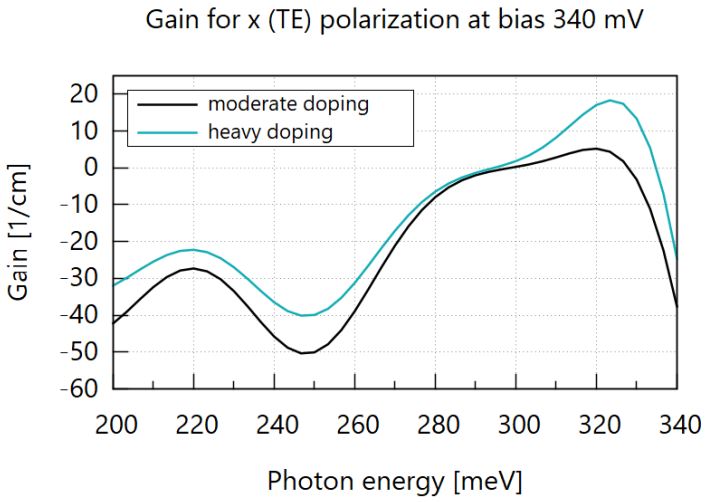


Figure 8.5.4.4: Gain spectra of the structures Figure 8.5.4.3 (a,b), demonstrating an enhancement for the photon energies 310-330 meV.

In addition to the desired gain at the photon energies 310-330 meV, the structure has a significant absorption peak around 250 meV. Gain spectra from individual transitions are stored in *(Bias)mV\Gain\SemiclassicalDecomposed_vs_Energy_(light polarization).dat* (Figure 8.5.4.5). The state index in the file refers to the energy eigenstates, which can be found in *(Bias)mV\EnergyEigenstates\EigenStates.dat*. This decomposed data shows that the dip around 250 meV in Figure 8.5.4.4 arises from valence intersubband absorption. It has been shown to reduce the optical gain in long-wavelength mid-infrared ICLs [*KnoetigLaserPhotRev2022*]. The present structure, however, does not suffer from it as the heavy-hole ground states is energetically much closer to light-hole ground state than to the conduction band ground state, i.e., absorption peak occurs at lower energy than the desired lasing transition.

Larger bias creates more electrons and holes in the W-shaped recombination region, which increases gain (Figure 8.5.4.6). The gain does not saturate in the present model since the feedback from emitted light (cavity field) to carrier transport is not included.

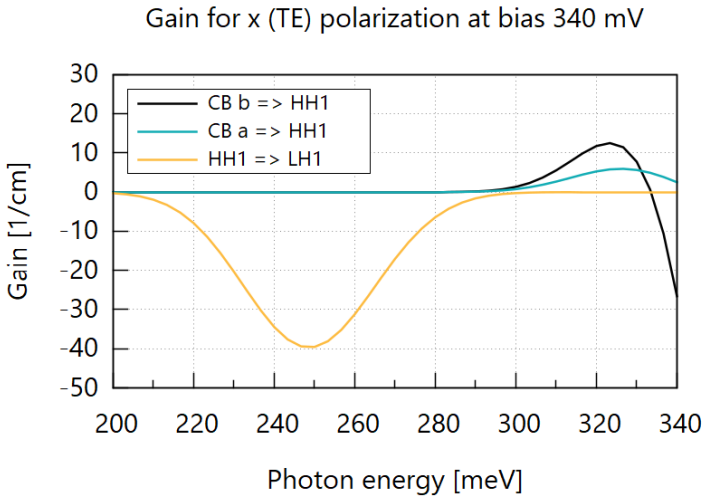


Figure 8.5.4.5: Relevant transitions in the gain spectrum of the heavily-doped case. 'CB a' and 'CB b' refer to the two lowest electron states in the asymmetric W-shaped quantum well. 'HH1' and 'LH1' are the valence band ground states with dominant heavy- and light-hole components at the zone center.

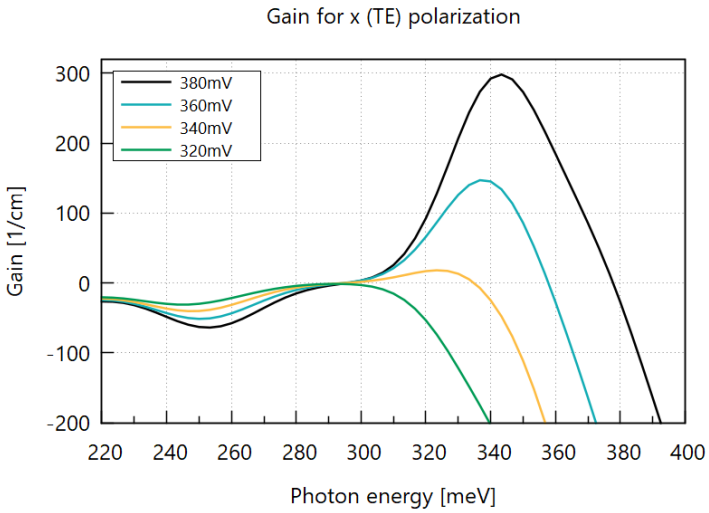


Figure 8.5.4.6: Gain spectra of the structure (b) of Figure 8.5.4.3 for various bias voltages.

Lastly, we investigate the polarization dependence of the gain spectra. While the z (TM) polarization gives only negligible contribution to the radiative interband transition (CB => HH1), it is the dominant channel for intraband devices such as quantum cascade lasers (QCLs). In ICLs, the TM polarization manifests itself in the parasitic absorptions between the states dominated by the same spinor component (Figure 8.5.4.7).

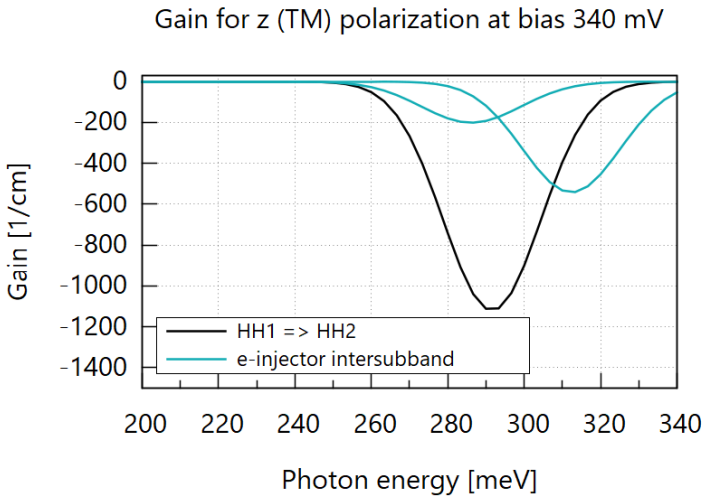


Figure 8.5.4.7: Gain spectra of the structure (b) of Figure 8.5.4.3 for the TM polarization.

Prospects

This model has two major limitations.

Firstly, it relies on a phenomenological broadening parameter Γ in Fermi's golden rule. However, the NEGF formalism is capable of calculating the line broadening from the first principle using fundamental material parameters. We are currently testing interband devices in the presence of interface roughness, alloy disorder, and LO phonon scattering.

The second limitation is the assumption of flat quasi-Fermi levels (Figure 8.5.4.1) and hence the use of equilibrium Green's functions. This hinders charge current (electron transport) calculation. We are developing a more realistic model without this assumption to enable simulation of current-voltage characteristics as well as self-consistent calculation of carrier transport and gain.

8.6 Command Line

Command line usage:

Example on Linux:

```
./nextnano.NEGF++_ubuntu_intel --input-file <path to input file> [--output-folder
<path to output folder>] [--material-database ./Material_Database.negf] [--threads
<integer value>]
```

nextnano.NEGF++ can be invoked from the command-line using these arguments in arbitrary order:

Mandatory arguments include:

-i, --input-file <path to input file> Path to XML input file.

Optional arguments include:

-d <path to database>, --database <path to database> Specify database file.

-l <path to license file>, --license <path to license file> Specify license file.

-o <folder>, --output-folder <folder> Specify output directory. If not specified, ./Output will be used.

-t i, --threads i Set number of parallel threads. Here, `i` threads are specified.

Note: Multi-threading in the NEGF routines

If present, the input parameter

```
SimulationParameter{
   ...
   NMaxThreads = 12
   ...
}
```

sets the maximum number of threads used in the NEGF routines. If `--threads` command line parameter (see above) is specified in the command line in addition, the number of threads used in the NEGF routines is fixed to the latter value.

If this number exceeds the number of **physical** cores, `NMaxThreads` is automatically reduced to the number of physical cores to avoid slowdown of the program. Note that when hyperthreading is activated, the number of physical cores is half of the number of logical processors.

For an automatic setting, do not specify these parameters. Then it will be derived from the number of physical cores and sweep situation.

Parallelization of Temperature-Voltage sweep

This feature will be implemented.

8.7 Release Notes

8.7.1 2024-07-25

8.7.2 2024-06-21

Bug fixes

- Fixed output of energy spectra in `Init/EnergyEigenstatesFull0V/kResolved` which are output if `Output{ PlotDispersion = 2 }`.

Improvements

- Aligned output of effective masses and 8-band k.p parameters in *Init* folder.
- Some simulation parameters in ICL sample files have been improved.
- `EnergyRangeLateral` for 8-band model now works consistently to 3-band: it is compared to the max of the energy ranges of the ground states in the conduction and valence bands.
- Added an option `LateralDiscretization{ OptimizeSampling = yes/no }`. It performs a sampling of the Bessel (i.e. in-plane) modes so that the k‖ points are approximately equidistant. The resolution can be tuned by `LateralDiscretization{ Value }`.

8.7.3 2024-05-23

New features

- 8-band k.p solver is now available by the input file keyword `Materials{ NumberOfBands = 8 }`. Its use for the NEGF calculation is currently limited to equilibrium or quasi-equilibrium cases. It can be used for semiclassical gain calculation. Please check out the interband cascade laser example in the *example* folder.

Bug fixes

- When no bias is specified, assume 0V instead of terminating the program.
- Fixed the zincblende bowing parameters for valence band offset. Now ternaries give consistent band edges to nextnano++.

Improvements

- When the program is called with *–version* command line option, it prints the version number and exit.
- Semiclassical gain calculation can be activated either by setting a linewidth through `Gain{ Linewidth }`. To skip NEGF gain and calculate only with the semiclasscial method, the user can specify `Gain{ GainMethod = 'semiclassic' }`. This method calculates Fermi's golden rule with the reduced space dipole matrix elements for the electric-field polarization along growth.
- Semiclassical gain calculation now takes into account the in-plane k dependence of the dipole matrix elements when the in-plane nonparabolicity is activated.
- Semiclassical gain calculation supports electric-field polarization along in-plane directions if the 8-band model is used (`Gain{ Polarization{ Re = [1,0,0] } }`). Complex polarization vector (e.g. circular polarization) is supported.
- Semiclassical gain calculation now outputs the spectra for individual transitions as well.
- If the in-plane nonparabolicity is considered (either 3 or 8 bands), and if `Output{ PlotDispersion = 2 }`, the in-plane k-dependent dipole matrix elements for the TM polarization are output.
- Ubuntu executable now uses Intel MKL with parallelism when multiple threads are available.
- Added user warnings regarding the lateral energy (k space) resolution.
- Reorganized the in-plane k-dependent output of energy eigenstates.
- Clarified the file names of the 2D colormap outputs.
- Gnuplot files of 2D colormap outputs are generated when `Output{ GnuplotFormat = yes }` (default).
- Voltage-sweep simulations generate GIF animations of carrier density, current density, density of states, emitted power, and gain spectrum. Installation of Gnuplot is needed to generate the .gif animations from the .plt files, see http://www.gnuplot.info/ for more information.
- Bandedge profiles without strain are also included in the output file *Input/BandEdges.dat* when strain is considered.

Syntax changes

- Updated the input file keyword `TuneAlloyScattering` to `AlloyScatteringStrength` in the example files.

8.7.4 2024-03-14

Bug fixes

- The hybrid method has been fixed.
- Fixed bias of band edge profile output when Poisson's equation is not solved

Improvements

- Phonon damping of TO phonons has been added to the database ("PhononDamping") and is considered in the permittivity calculation
- Permittivity output has been added
- Material_Database_Vurgaftman.negf has been added. It overwrites the nextnano.NEGF default values by the ones of Vurgaftman et al., J. Appl. Phys. (2001).
- Possibility of linearly graded alloy

8.7.5 2023-12-01

Bug fixes

- (database) fixed temperature-expansion coefficient of lattice constants, although it is not used in the code yet
- (database) Renamed from Material_DatabaseCpp to Material_Database. The executable uses 'Material_Database.xml' if not specified by the command line parameter *–material-database*
- EnergyReference command (optional in SimulationParameter section) has been fixed. Also the unit has been changed to eV for consistency.
- Fixed output of levels
- Fixed output of valence band edges

Improvements

- Equilibrium calculation. If no contacts nor Fermi energy specified, overall charge neutrality is assumed for the all device.
- Improved the console output of multithreading info
- Print error message to console if LAPACK eigensolver has failed
- User warnings are now listed at the end of simulation as well as saved to the file *Warnings.log* in the output directory for easier check if the simulation ran as intended.

Syntax changes

- the in-plane non-parabolicity is activated by default (command InPlaneNonParabolicity is optional, and set to yes by default)
- The folder WannierStark has been renamed to EnergyEigenstates as the code applies for both periodic (e.g. QCL, QCD) and non-periodic (e.g. QWIP, RTD) structures

---

8.7.6 2023-05-25

Bug fixes

- gain calculation is working
- Bug fixed when nLateralPeriodsForBandStructure > CoherenceLengthInPeriods
- Error fixed in strain calculation
- bessjzero and Correlation data files are now found even if the current working directory is not in the executable folder.
- If the syntax definition files (syntax_database.negf and syntax_input.negf) are not found in the executable directory, search for them in the current working directory.
- (database) Fixed Varshni parameter of AlSb

Improvements

- Faster self-energy calculation
- Support the nn++/nn3 command line arguments —inputfile, —outputdirectory, —database, and —license.
- More strict check of command line parameters –> more meaningful error message
- If .negf format is used, the parsed results of the input and/or database file will be output in the output folder, as is the case in *nextnano++*.
- More robust file loading schemes for bessjzero, Correlation and syntax definition files
- Improved the keyword updates in MSB input files
- (database) added references and TiberCAD default values.
- THz QCL sample files with gain calculation added

8.7.7 2023-02-22

This release includes the following new features compared to the C# version:

1. **Start from equilibrium** option. The NEGF calculation starts by an equilibrium calculation of the Green's functions, which is then then used as an initial condition for the non-equilibrium calculation of the Green's functions.

```
<SimulationParameter>
    ...
    <StartFromEquilibrium> yes </StartFromEquilibrium>
    ...
</SimulationParameter>
```

2. **Equilibrium calculation without scattering calculation**. The following equilibrium section has to be specified, including a broadening parameter.

```
<Equilibrium>
    ...
    <Broadening unit="meV">20.0</Broadening>
    ...
</Equilibrium>
```

3. **Hybrid equilibrium / non-equilibrium equilibrium simulation**. The hybrid section has to be specified, as well as the section

```
<Hybrid>
    <Broadening unit="meV">10.0</Broadening>
            <SeparationLeft unit="nm">20.0</SeparationLeft>
            <SeparationRight unit="nm">30.0</SeparationRight>
            <OffsetContact unit="nm">5.0</OffsetContact>
</Hybrid>
```

4. Added **quantum cascade detector (QCD)** examples.
5. **The nextnano++-style format is available for the input and database files**. The new format with **extension 'negf'** features:
   - **Syntax validation before the simulation** - the parser checks the syntactical correctness as well as the logic and, if invalid, prints an error message in the log specifying what exactly is wrong in your file.
   - Variables starting with `$` sign can be used not only in the input but also database files. The new default database *Material_DatabaseCpp.negf* contains a switch `$NEGF` / `$MSB` / `$nnp` to use the default parameters of *nextnano.NEGF*, *nextnano.MSB*, and *nextnano++*, respectively. Naturally, the new format will look much familiar if you have experience with *nextnano++*.
   - **The *nextnanomat* feature *Tools > Convert .xml Input File to .negf Input File* can convert your xml input files to the new format. On Linux, you can run the executable *syntax_converter* included in the package. After conversion, please note that**
     - The section `Variables` has been deprecated. Please add manually your variables in the *nextnano++* format, i.e., define the variables at the beginning of your input file as `$variable = value` (See *Input Syntax*).
     - Most of the XML comments `<!-- -->` are not retained in the conversion. If necessary, please recover them manually. Comments start with the letter `#`.
     - Within the new format, only the new keywords are supported. They are replaced automatically by the conversion.
     - IMPORTANT: **Do not convert the XML database to the new format**! The new database *Material_DatabaseCpp.negf* is not backward-compatible and contains extra parameters.

8.8 Release Notes (Classic)

8.8.1 2022-06-13

1. Units in **LO-phonon scattering rates** have been fixed.
2. Now the license **nnNEGF.lic** is supported.

8.8.2 2022-04-29

1. The **in-plane nonparabolicity** model introduced in the previous update has been fixed.

```
<Materials>
    ...
    <InPlaneNonParabolicity>yes</InPlaneNonParabolicity>
    ...
</Materials>
```

2. The **LO-phonon scattering rate** output has been fixed.

8.8.3 2022-03-14

1. Linear **alloy grading** is now possible:

```
<Layer> <!-- all the material parameters will be linearily interpolated between the
↪<Material1> and the <Material2> -->
    <Material1>mat1</Material1>
    <Material2>mat2</Material2>
    <Thickness unit="nm">5.0</Thickness>
</Layer>
```

2. The **in-plane nonparabolicity** for **multiband models** can be activated with the command `<InPlaneNonParabolicity>yes</InPlaneNonParabolicity>` inside the block `<Materials>... <Materials>`.

3. In the 2- and 3-band models, **nonparabolicity coefficient** can be used as in input material parameter. If specified, it will be used to overwrite the k.p parameters.

```
<Material>
    <Name>In(x)Ga(1-x)As</Name>
    <Alloy_Composition>0.53</Alloy_Composition>
    <Alias>well</Alias>
    <Effective_mass_from_kp_parameters>no</Effective_mass_from_kp_parameters>
    <Overwrite>
        <NonParabolicity Unit="cm^2">1.5e-14</NonParabolicity>
    </Overwrite>
</Material>
```

4. The **individual scattering rates** of **LO-phonon**, **alloy disorder** and **interface roughness** can be output.

CHAPTER

NINE

NEXTNANO.MSB

9.1 Overview

This documentation describes *nextnano.MSB*, an NEGF quantum transport code based on the MSB method.

The *nextnano.MSB* tool has been developed to simulate *Quantum Cascade Lasers* (QCLs) and *Resonant Tunneling Diodes* (RTDs). The tool calculates current-voltage characteristics and gain.

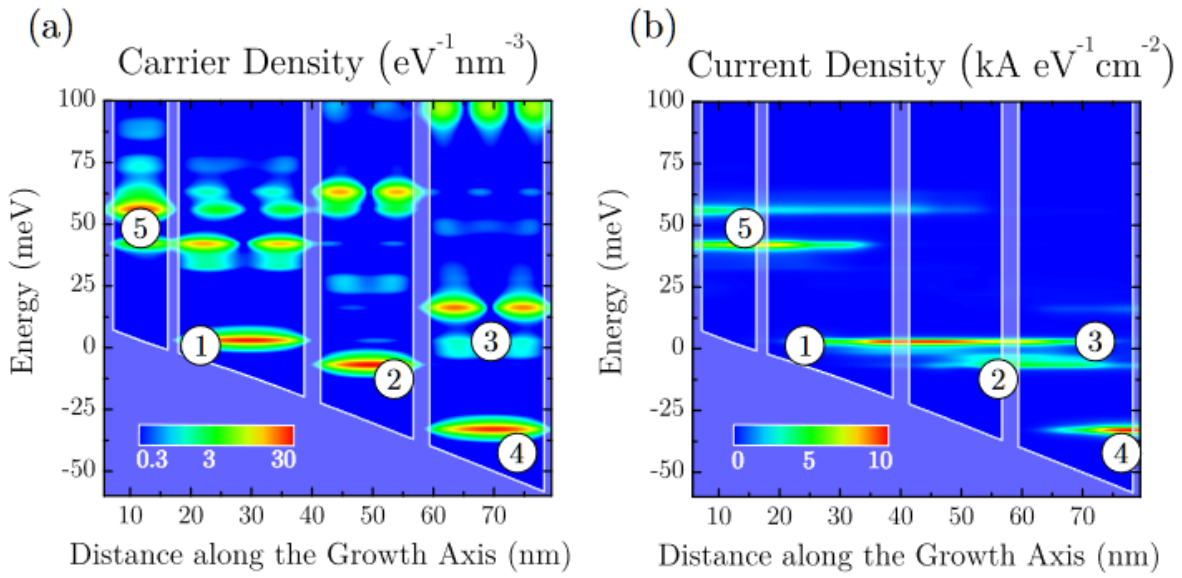


Figure 9.1.1: Calculated electron and current density of a Quantum Cascade Laser

This tool is based on a novel quantum transport method that follows the non-equilibrium Green's function (NEGF) framework but sidesteps any self-consistent calculation of lesser self-energies by replacing them by a quasi-equilibrium expression. This method generalizes the so-called Büttiker probe model but takes into account all relevant individual scattering mechanisms. It is orders of magnitude more efficient than a fully self-consistent NEGF calculation for realistic devices, yet accurately reproduces the results of the latter method. This method opens the path towards realistic three-dimensional quantum transport calculations.

MSB is the acronym of the **M**ulti-**S**cattering **B**üttiker probe model.

The *nextnanomat* GUI can execute the code and visualize the results.

Which scattering mechanisms are included?

- Longitudinal polar-optical phonon scattering (polar LO phonon scattering).
- Acoustic phonon scattering which includes some kind of interface roughness scattering. The latter is intrinsically included.

Where can I find some background on the implemented physics?

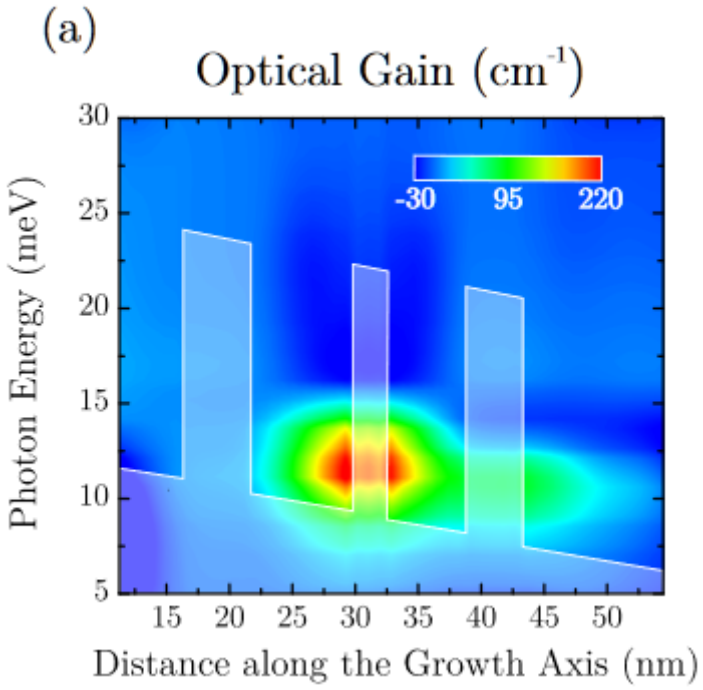


Figure 9.1.2: Calculated gain of a Quantum Cascade Laser

The MSB method is described in the following publications: *[GreckOE2015]*, *[GreckIWCE2010]*, *[Greck-PhD2012]*

Copyright information

The *nextnano.MSB* tool has been developed by Dr. Peter Greck for the nextnano GmbH. It is written in C++.

9.2 Command line arguments

NOTE: nextnano.MSB has been integrated into the nextnano++ executable (since 2021-08).

Below is the old documentation for stand-alone MSB executable.

Command line usage:

```
nextnano.MSB.exe -inputfile ..\my_QCL_input_file.xml -license "H:\My Documents\
nextnano.MSB\License\license.xml" -database Materials.xml  -outputdirectory ..\..\
output\QCL
```

The *nextnano.MSB* tool provides the following options:

-f input_file, -inputfile input_file
Name of input file to be read in.

Example:

- `-inputfile THz_GaAs_QCL.in`

-d database_file, -database database_file
Name of material parameter database file to be read in (`-m` also works).

Example:

- `-database "E:\nextnano.MSB\Materials_no_Varshni.xml"`

-license license_file
Name of license file to be read in.

Example:

- `-license License_nnMSB.lic`

-output_directory output_directory
Directory name where to output results.

Example:

- `-outputdirectory "D:\nextnano\nextnano output\""`

-debug 1
Generate additional debug information.

9.3 Input file syntax

9.3.1 Input file

The structure to be simulated as well as all simulation parameters have to be specified in a text file by the user. In addition to the XML format, the *nextnano++*-style format with the extension .negf is supported (since 2022 Apr).

Both formats of the sample input files are provided in the installation folder, e.g. *THz_QCL_Fathololoumi_OpticsExpress2012.xml* and *THz_QCL_Fathololoumi_OpticsExpress2012.negf*. This documentation explains the syntax in the .negf format, but the keywords and hierarchical structure are identical between two formats except for <Variables> (difference described below).

You can convert the existing XML input files to the new format using *nextnano.MSB* executables later than 2022 Apr. If additional command line option `-p` (or `--parse`) is set, the execution stops after the input file and database have been read and parsed. In case XML files were read in, tentative conversions into the new format are output.

9.3.2 General syntax

```
nextnano.MSB{         # Version = 1.1.0   # This number must be consistent␣
↪to the version of the executable used.
```

In the XML format,

```
<?xml version="1.0" encoding="utf-8"?>

<nextnano.MSB Version="1.1.0">  <!-- This number must be consistent to the␣
↪version of the executable used. -->
```

In XML, Quotation marks are necessary if an equal sign (=) is present, e.g. `AlloyX="Barrier"`.

Comment Section

```
Header{                                    # The Header contains a␣
↪description of the input file.
   Author = "Peter Greck"                  # name of the author of this␣
↪input file
   Version = 1.0                           # version number of this input␣
↪file
   Content = "See comments in input file."  # information that describes␣
↪the content of the input file
   #       Terahertz quantum cascade lasers operating up to ~200 K with␣
↪optimized oscillator strength and improved injection tunneling
   #       S. Fathololoumi et al.
   #       Optics Express 20, 3866 (2012)
   #         ...
```

(continues on next page)

(continued from previous page)

```
   #       CPU time: ...
   }

   # Here comes the main part, see below...
}
```

Output

```
Output{
   Directory                    = Output/my_input_file    # Comment = "Name␣
↪of output folder for calculated results."

   WriteOutputEveryNthIteration = 2                       # Comment =
↪"Determines how frequent output is written."

   MaxNumberOfEigenstates       = 15                      # Comment =
↪"Determines number of eigenstates written in output."
```

If `MaxNumberOfEigenstates` ist not present, 25% of all possible eigenstates are written out. The total number of possible eigenstates corresponds to the total number of spatial grip points. More information on *output of eigenstates*.

```
FormatAsciiEnabled   = true         # Comment = "Enable output for ASCII␣
↪files containing data in columns"
FormatAsciiExt       = .dat         # Comment = "Specify the file extension␣
↪for ASCII files containing data in columns."

FormatGnuPlotEnabled = true         # Comment = "Enable GnuPlot output␣
↪based on text output."
FormatGnuPlotExt     = .gnu.plt     # Comment = "Specify the file extension␣
↪for GnuPlot output."

FormatVTKPlotEnabled = true         # Comment = "Enable VTK output."

FormatAvsEnabled     = true         # Comment = "Enable AVS output."
FormatAvsExt         = .avs         # Comment = "Specify the file extension␣
↪for AVS files."
FormatAvsBinary      = binary       # Comment = "Specify AVS output mode␣
↪[binary|ascii]"
```

Currently, there is no text output included in the code.

```
   FormatTextEnabled    = true      # Comment = "Enable text output."␣
↪(Currently, there is no text output included in the code.)
   FormatTextExt        = .dat      # Comment = "Specify the file extension␣
↪for text files." (Currently, there is no text output included in the code.
↪)
}
```

Variables

New: In the .negf format, variables must be defined with the dollar sign as in *nextnano++* (see *Input Syntax* and sample input file in the installation folder).

Iterator feature is realized by the *nextnanomat* feature *Template* or nextnanopy Sweep. This simply generates multiple input files, which can be run in parallel (see *Options: Simulation*).

```
$Temperature = 300   # (DisplayUnit:K) Temperature to sweep.␣
↪(ListOfValues:100,150,175,190,200) (RangeOfValues:From=100,To=200,Step=11)
$Doping = 1e16       # (DisplayUnit:1/cm^3) Doping concentration
$Barrier = 0.15          # Al concentration of barriers.
```

Below is the documentation of <Variables> in the old XML format input file.

In the XML format, one can define variables (`Constant`) that are used further below in the input file, e.g. `Doping = 1e16`.

```
<Variables>
   <Constant>
      <Name  Comment = "Doping concentration"> Doping </Name>
      <Value Unit = 1/cm^3> 1e16 </Value>
   </Constant>

   <Constant>
      <Name  Comment = "Al concentration of barriers."> Barrier </Name>
      <Value Unit = "[0..1]"> 0.15 </Value>
   </Constant>
```

Iterator Example 1

An `Iterator` can be used to sweep over variables, e.g to do calculations for several temperatures. Here, the iterator accepts a certain *list of values*.

In this example, a calculation for the temperatures `Temperature = 100 [K], 150 [K], ..., 200 [K]` is performed.

```
<Iterator>
   <Name    Comment = "Temperatures to sweep.">Temperature</Name>
   <Values  Unit = K> 100, 150, 175, 190, 200 </Values>
</Iterator>
```

Iterator Example 2

The `Iterator` also accepts a *range of values* from an `Initial` value to a `Final` value with a certain number of `Steps`. Here, the `Final` value and the number of `Steps` is given, and the step width, i.e. the `Delta`, is derived.

In this example, a calculation from `Temperature = 100 [K]` to `Temperature = 200 [K]` is performed for 11 different temperatures.

```
<Iterator>
   <Name    Comment = "Temperatures to sweep.">Temperature</Name>
   <Initial Unit = K> 100 </Initial>
   <Final   Unit = K> 200 </Final>
   <Steps   Comment = "Note that the value is calculated via Initial+[0..
↪Steps-1]*(Final-Initial)/(Steps-1)"> 11 </Steps>
</Iterator>
```

Iterator Example 3

The iterator also accepts a range of values starting from an `Initial` value and performing a certain number of `Steps` with a step width of `Delta`. Here, the step width, i.e. the `Delta`, and the number of sweep `Steps` is given, and from these, the final value is derived.

In this example, a calculation for 11 different temperatures with a step width of `10 [K]`, starting from `Temperature = 100 [K]` is performed.

```
   <Iterator>
      <Name    Comment = "Temperatures to sweep.">Temperature</Name>
      <Initial Unit = K> 100 </Initial>
```

(continues on next page)

(continued from previous page)

```
      <Delta   Unit = K> 10 </Delta>
      <Steps   Comment = "Note that the value is calculated via␣
↪Initial+[0..Steps-1]*Delta">11</Steps>
   </Iterator>
</Variables>
```

Device definition

Temperature as value

```
Device{

   Temperature = 200            # Unit = K  # This is the lattice␣
↪temperature, not the temperature of the electrons.
```

or temperature as a variable.

```
Temperature = $Temperature   # Unit = K  # Here, instead of a number, the␣
↪variable $Temperature is used.
```

Specify grid

This is the grid spacing. It determines the number of grid points in the device. The more grid points, the longer the CPU time. If the grid spacing is modified, then always a slightly different structure is calculated as the barrier height and widths are adjusted to match the grid spacing, see `AdjustBandedge`. Even if the grid spacing exactly matches the layer widths for all variations of grid spacings, in all cases slightly different structures are calculated as the dispersion relations change with grid spacing. If the grid is too fine, then one cannot calculate any more sufficiently large energies. In this case, the results would not be reliable.

```
Grid{
   Spacing = 0.25     # Unit = nm
}
```

Orientation and strain

In order to calculate strain related properties, information on the substrate material is required. Strain can be included (yes) or excluded (no). The crystal structure can be `Zincblende` or `Wurtzite`. The lattice constants of the substrate are used to calculate strain. The (hkl) Miller indices are needed in order to define the orientation of the substrate on which the heterostructure is grown, i.e. the *crystal coordinate system* has to be rotated into the *simulation coordinate system*. The growth direction (simulation axis z) is perpendicular to the substrate (xy plane). The crystal and thus all anisotropic material properties are rotated accordingly.

```
Crystal = Zincblende   # Zincblende or Wurtzite

Substrate{
   Material{
      Base = GaAs      # substrate material
   }
}

Orientation{
   z_axis{    # The heterostructure growth direction is along z (simulation␣
↪direction).
      h = 0   # (hkl) are the Miller indices of the plane perpendicular to␣
↪the z direction.
      k = 0
      l = 1
   }
```

(continues on next page)

(continued from previous page)

```
   y_axis{
      h = 0   # (hkl) are the Miller indices of the plane perpendicular to␣
↪the y direction.
      k = 0
      l = 1
   }
}

Strain = no   # include strain (yes/no)
```

Energy grid

Here, the energy grid spacing is defined. The energy grid is homogeneous. The `Nodes` are the number of energy grid points. The more energy grid points, the longer the CPU time. Example: If one has defined an energy `Range` of `0.3 [eV]`, then by choosing `601 Nodes`, the resulting energy grid spacing is 0.5 `[meV]`. If the barrier height in the conduction band is `0.5 [eV]`, then `Range` should also be around `0.5`. The larger the barrier, the higher the Range.

```
Energy{
   Nodes = 601   # Comment = "Number of energy grid points."
   Range = 0.3   # Unit = eV  # Comment = "Offset is based on conduction␣
↪band edge, i.e. Input/BandEdge_conduction_input.dat."
}
```

Definition of layers

```
# Begin Layers
Layer{
   Thickness = 4   # Unit = nm
   Material{
      Base = GaAs
   }
```

or alternatively use conduction band edge as defined by `ConductionBandOffset`.

```
Material{
   Base = GaAs
   CalculateBandedge = no
}
```

or (default) calculate conduction band edge by using valence band offset (`ValenceBandOffset`) and temperature dependent band gap, i.e. $E_\mathrm{c} = E_\mathrm{v} + E_\mathrm{gap}(T)$, taking into account the Varshni parameters.

```
   Material{
      Base = GaAs
      CalculateBandedge = yes
   }
   Probes = 1.0          # Comment = "Specifies the relative scattering␣
↪strength within this layer. Set to zero to remove all probes from this␣
↪layer."
   Doping = $LeadDoping  # Unit = 1/cm^3  # Here, instead of a number, the␣
↪variable $LeadDoping is used.
}
```

It is very useful to define one (or several) periods of a quantum cascade laser with such a *marker* (`BeginCluster`). This allows one to define a bias per period very conveniently, i.e. a defined electrostatic potential drop per period. Here, a `4.3 [nm]` wide `AlGaAs` (Al(x)Ga(1-x)As) layer with an alloy concentration of `x = 0.15` is defined. The

layer is undoped. Here, two labels, namely `Center` and `QCL`, are assigned to a cluster which begins here and ends at `Layer EndCluster="Center, QCL"`.

```
# Begin Period
Layer{
   BeginCluster = "Center, QCL"
   Thickness = 4.3           # Unit = nm

   # Material{
   #    Base = AlGaAs
   #    AlloyX = $Barrier    # Here, instead of a number, the variable
↪$Barrier is used.
   # }

   Material{
      Base = AlGaAs
      AlloyX = 0.15
   }
   Probes = 1.0
   Doping = 0         # Unit = 1/cm^3
}

Layer{                # Here, a 7.9 nm wide GaAs layer is defined. The layer␣
↪has a doping concentration of 6e16 1/cm^3.
   Thickness = 7.9    # Unit = nm
   Material{
      Base = GaAs
   }
   Probes = 1.0
   Doping = 6e16      # Unit = 1/cm^3
}

Layer{                # Here, a 5.5 nm wide GaAs layer is defined. The layer␣
↪is undoped.
   EndCluster = "Center, QCL"
   Thickness = 5.5    # Unit = nm
   Material{
      Base = GaAs
   }
   Probes = 1.0
   Doping =   0       # Unit = 1/cm^3
}
# End Period

Layer{
   Thickness = 4.0    # Unit = nm    # Here, a 4.0 nm wide GaAs layer is␣
↪defined. The layer has a doping concentration determined by the variable
↪$LeadDoping.
   Material{
      Base = GaAs
   }
   Probes = 1.0
   Doping = $LeadDoping     # Unit = 1/cm^3
}
# End Layers
```

Each `Layer{` can have an additional flag:

```
Layer{
   AdjustBandedge = yes    # Comment = "yes or no"  # default is yes
}
```

The ratio of the desired barrier width (e.g. 1.1 [nm]) to the width used in the simulation (e.g. grid spacing 0.5 [nm] ==> 2*0.5 [nm] = 1.0 [nm]) is added to the barrier height to keep the area of the barrier the same. This approach compensates the loss in accuracy when using a larger grid spacing very well. This applies analogously to the well width. The effect of this flag can be seen in these files in case they are plotted on top of each other:

- Input\BandEdge_conduction_input.dat
- Input\BandEdge_conduction_adjusted.dat

Contacts (leads)

```
# Begin Lead
Lead{
   Name = Source        # Define a source contact with voltage V = 0.000 V.
   Voltage{
      Initial = 0.000  # Unit = V
   }
   Temperature = 300    # Unit = K # Define the temperature of the contact.␣
↪If not defined, then the device temperature is used
}

Lead{
   Name = Drain         # Define a drain contact with voltage V that is␣
↪varied during the calculation starting from V = 0e-3 V.
   Voltage{
      ...
   }
```

(*Check: Probably the terms ``Source`` and ``Drain`` are required. This should be checked.*)

In the defined structure, there is a region (here the cluster is called Center) where the given voltage drops, e.g. Cluster>Center</Cluster. One can include the leads into this region, then the whole voltage drops over the whole structure. Typically, for QCL simulations, the given bias voltage refers to one period only, i.e. the bias voltage per period. In this case, the lead regions are not included. Therefore, the actual voltage drop between the left and the right boundary grid points of the whole structure is larger than the specified value of the voltage drop because the structure is larger than the QCL period as the leads have to be taken into account when calculating the length of the structure.

Example: If one applies 50 mV, this difference corresponds to an energy difference of 50 meV of the conduction band edge of the leftmost and rightmost grid point of the structure, or of the leftmost and rightmost grid point of the region where the bias drops. The effective electric field can be calculated as follows:

field = applied bias / relevant region

Depending on the definition in the input file, the relevant region can include the lead regions or not.

Option 1

A voltage is specified (which typically corresponds to a bias/period).

```
Voltage{
   Cluster = Center
   Initial = 0e-3      # Unit = V
   # Delta = 3e-3      # Unit = V   # For the meaning of Initial, Delta,␣
↪Final and Steps see the documentation of <Iterator>.
   Final   = 60e-3     # Unit = V
   Steps   = 15        # Unit = ""  # If Steps = 1, only one voltage is␣
```

(continues on next page)

(continued from previous page)

```
↪calculated. If Steps = 0, then automatically, Steps = 1 is assumed.
}
```

Option 2

As an alternative, instead of specifying a voltage, one can also specify an electric field. Internally, the relevant bias is calculated from the field, and then the calculation starts.

```
      Voltage{
         SpecifiedByElectricField = yes    # Comment = "Specifies the␣
↪voltage as electric field across the device.
                                           # The units of Initial, Final,␣
↪and Delta are [kV/cm].
                                           # Valid input is [yes|no].␣
↪Default is [no]."> no </SpecifiedByElectricField>
         Initial = 0   # Unit = kV/cm
         # Delta = 5   # Unit = kV/cm     # For the meaning of Initial,␣
↪Delta, Final and Steps see the documentation of <Iterator>.
         Final   = 30  # Unit = kV/cm
         Steps   = 5   #  Unit = ""
      }
   } # End Lead
} # End Device
```

MSB model definition - Multi-scattering Büttiker probe model

MSB single-band (`1Band` = single-band)

```
MSB_1Band{
```

Piezoelectric and pyroelectric charge densities can be included (`yes`) or excluded (`no`) in the Poisson equation.

```
Poisson{
   Piezo = no
   Pyro  = no
}
```

Define numerical parameters

```
MaxGreenIts        = 25                      # Comment = "Max. iterations␣
↪within the G^R cycle."
MaxGreenDelta      = 3e-7   # Unit = 1/nm/eV # Comment = "Max. change␣
↪between two cycles to abort iteration."

MaxProbeIts        = 15                      # Comment = "Max. iterations␣
↪for current conservation calculation. Only used if Core.
↪ProbeMode=iterative"
MaxProbeNorm       = 1e-9   # Unit = A/cm^2  # Comment = "Max. absolute␣
↪leakage current to abort iteration."

MaxPoissonOuterIts = 25                      # Comment = "Max. outer␣
↪Poisson iterations where G^R is recalculated."
```

`MaxPoissonOuterIts` is an important number. Make sure that this number is not exceeded. If it is exceeded, the calculation did not converge! In the *.log file*, the following information is written to inform about the number of Poisson iterations: `Poisson iteration 30`

```
MaxPoissonInnerIts      = 15                  # Comment = "Max. inner␣
↪Poisson iterations where the density predictor is used."
```

(continues on next page)

(continued from previous page)

```
MaxPoissonDensityNorm   = 1e9  # Unit = 1/cm^3 # Comment = "Max. absolute␣
↪density   deviation to abort Poisson iterations."
MaxPoissonCorrectorNorm = 1e-4 # Unit = V      # Comment = "Max. absolute␣
↪potential deviation to abort Poisson iterations."
```

Now we define the core properties of the MSB method.

```
Core{
   BallisticCalculation = no   # Comment = "yes or no"  # default is no
```

In a ballistic calculation, where scattering is not taken into account, the following parameters are set to zero:

- `Probes`
- `ScatteringStrengthConst`
- `ScatteringStrengthBP`
- `ScatteringStrengthMSB`

```
PotentialInit = bulk
```

- `bulk` (default): The initial guess of the electrostatic potential assumes a bulk semiconductor where the resulting density is neutral at the grid point 0 or n, respectively.
- `zero`: The initial guess of the electrostatic potential is zero.

```
ProbeMode = direct  # Comment = "Specify method to calculate current␣
↪conservation. Novel (more stable) method is 'direct'.␣
↪[direct|iterative]"
```

Note that `direct` and `iterative` can lead to different results. They are not equivalent.

- The `iterative` model is explained in Section 5.2 in the PhD thesis of Peter Greck. Here, for each probe, *virtual chemical potentials* $\mu_{\mathrm{p}}(z)$ are calculated. The units are `[eV]`. Therefore, for each probe a virtual chemical potential exists which leads to a Fermi distribution of each probe $f(E, \mu_{\mathrm{p}}(z))$. This virtual chemical potential μ refers to the total probe, i.e. AC + LO.
- The `direct` model is explained in Section 7.2 in the PhD thesis of Peter Greck. Here one assumes that the distribution function $f_{\mathrm{p}}(z)$ of each probe is a linear combination of the source and drain distributions, f_{S} and f_{D}, respectively, where S means *Source*, and D means *Drain*.

  $$f_{\mathrm{p}}(E) = c_{\mathrm{p}} f_{\mathrm{S}}(E) + (1 - c_{\mathrm{p}}) f_{\mathrm{D}}(E)$$

  The coefficients c_{p} for each probe are in the interval [0,1] and are dimensionless. Here, a linear system of equations has to be solved to obtain the *coefficients* c_{p}. In contrast to the `iterative` model, the virtual chemical potentials μ_{p} for the probes cannot be extracted in this case. For the results presented in the PhD thesis of P. Greck, always the `direct` model was used.

```
VoltageMode = drop   # Comment = "Specify handling of applied bias␣
↪voltage. Drop-mode ensures that the specified voltage drops along the␣
↪device. Flat-mode uses Neumann boundary conditions that do not␣
↪guarantee a complete voltage drop. [drop|flat]"
```

The total scattering strength that is used for the calculation is the sum of the following three products, i.e. total = Model #1 + Model #2 + Model #3:

`` Probes =`` * `ScatteringStrengthMSB` +

`` Probes =`` * `ScatteringStrengthBP` + (This term is usually zero.) ==> These are the "normal" Büttiker probes (see *[DattaETMS1995]*).

`` Probes =`` * `ScatteringStrengthConst` + (This term is usually zero.) ==> These are the "oldest" type of Büttiker probes (see *[DattaETMS1995]*).

We recommed to only define `ScatteringStrengthMSB` and set `ScatteringStrengthBP` and `ScatteringStrengthConst` to zero.

Model #1 for scattering

Here, we can define a scattering strength for both LO, and AC phonon scattering.

```
ScatteringStrengthMSB    = 1.0  # Comment = "Novel scattering via MSB.␣
↪Scattering strength is calculated from material parameters. This␣
↪parameter tunes the scattering rates from 0.0=disabled over 1.0=normal␣
↪to X=amplified."
ScatteringStrengthMSB_AC = 1.0
ScatteringStrengthMSB_LO = 1.0
```

Model #2 for scattering

These are the *normal* Büttiker probes (see *[DattaETMS1995]*).

```
ScatteringStrengthConst = 0.0   # Unit = eV   # Comment = "Additional␣
↪constant scattering strength for every probe. Typical values are 0.001-
↪0.01 and zero disables this scattering mechanism."
```

Model #3 for scattering

These are the *oldest* type of Büttiker probes (see *[DattaETMS1995]*).

```
   ScatteringStrengthBP    = 0.0   # Unit = eV    # Comment =
↪"Additional scattering via Büttiker-Probes. Typical values are 0.001-0.
↪01."
} # End Core
```

Gain

```
   Gain{
      Cluster             = Center               # Comment = "Specfiy the␣
↪cluster where optical gain should be calculated."
      PhotonEnergyInitial = 3e-3    # Unit = eV  # Comment = "Min. photon␣
↪energy for gain calculation."
      PhotonEnergyFinal   = 30e-3   # Unit = eV  # Comment = "Max. photon␣
↪energy for gain claculation."
      PhotonEnergySteps   = 100     # Unit = ""  # Comment = "Number of␣
↪photon energies for gain calculation."
   }
} # End MSB_1Band
```

The more `PhotonEnergySteps` are used, the more ragged the gain curve gets. Decreasing this number, smoother gain curves can be obtained. This is normal. If the photon energy grid spacing is too small, one tries to resolve energies that are not resolved within the calculated Green's functions that are used in the gain calcuations, i.e. one tries to calculate several photon energies within one discrete energy step of the Green's function. An obvious solution would be to increase the energy grid resolution of the Green's functions. However, it is recommended to rather prefer a course photon energy grid as a high photon energy grid resolution does not lead to significantly more insight.

Hints

- One can disable the scattering within the barriers. This can reduce computational time with very minor influence on overall results.

```
Layer{
   Probes = 0.0   # disabled
```

(continues on next page)

(continued from previous page)

```
}

Layer{
   Probes = 1.0    # enabled
}
```

Important: If there are no probes inside the barrier, the iterative calculation of the probes is much more stable while hardly changing the physics because the occupation in the barriers is basically zero. Moreover, this avoids the numerical explosion of the linear system of equations through a division by zero which leads to `NaN` (*not a number*) in the `.log` file.

- How can I do a ballistic calculation? There are three ways to achieve this.

  a) Use

  ```
  Core{
     BallisticCalculation = yes    # Comment = "yes or no"
  }
  ```

  b) Set all `` Probes =`` to zero, i.e. `0.0`.

  c) Set all of the following parameters to zero: `ScatteringStrengthMSB`, `ScatteringStrengthBP` and `ScatteringStrengthConst`, i.e. `0.0`.

- For a quantum cascade laser simulation it can make sense to simulate 5 or 7 periods and take the middle period as the one where one extracts the results like local density of states, electron density, current density, gain, ...
- If one simulates only *one* period of a QCL, then one should add a *doping layer* at the left and right boundary (e.g. of width 4.5 nm). This should not be necessary if 5 periods or more are simulated. In this case, doping could be omitted.

9.4 Material database

The material parameters for zinc blende and wurtzite materials that are used by *nextnano.MSB* are stored in a file called `materials.negf`. The user can add further materials if needed.

If you run *nextnano.MSB* via The *nextnanomat* GUI, you can choose to read in a customized material database as follows:

nextnanomat ==> Tools ==> Options ==> Material database

In the database there are entries for binary compounds like GaAs, AlAs, InP, ..., as well as for ternary compounds like AlGaAs, InGaAs, ...

9.4.1 Elements and binary compounds

Note that the material database contains parameters that are not used by *nextnano.MSB*. This is because we unify the database with *nextnano.NEGF*.

```
# binary compound
    Material{
        Name = GaAs
        CrystalStructure = Zincblende

        ConductionBandOffset = 2.979       # Unit = eV
        ValenceBandOffset = 1.346          # Unit = eV
```

(continues on next page)

(continued from previous page)

```
        BandGap = 1.519                     # Unit = eV
        BandGapAlpha = 0.5405e-3            # Unit = eV/K
        BandGapBeta = 204                   # Unit = K

        ElectronMass = 0.067                # Unit = m0

        EpsStatic = 12.93
        EpsOptic = 10.89

        DeformationPotential = -9.36        # Unit = eV
        ValenceDefPotHydro = 1.16           # Unit = eV
        ValenceDefPotUniaxial_b = -2.0      # Unit = eV
        ValenceDefPotShear_d = -4.8         # Unit = eV

        MaterialDensity = 5.316e3           # Unit = kg/m^3
        VelocityOfSound = 4.73e3            # Unit = m/s

        LOPhononEnergy = 35e-3              # Unit = eV
        LOPhononWidth = 3e-3                # Unit = eV
        TOPhononEnergy = 33.86e-3           # Unit = eV
        AcousticPhononEnergy = 5e-3         # Unit = eV

        S = -2.88
        Ep = 28.8                           # Unit = eV
        kp_8_bands{
            B = 7.979
            L = 2.73984
            M = -3.86
            N = 1.37984
            kappa = -1.74
        }
        DeltaSO = 0.341 # Unit = eV

        Lattice_a = 0.565325                # Unit = nm
        Lattice_a_expansion = 3.88e-5

        Elastic_c11 = 122.1                 # Unit = GPa
        Elastic_c12 = 56.6                  # Unit = GPa
        Elastic_c44 = 60.0                  # Unit = GPa
        Piezo_e14 = -0.160                  # Unit = C/m^2
    }
```

ConductionBandOffset

type
double

unit
[eV]

Energy value that defines the position of the conduction band edges on an absolute energy scale. The zero point of energy is arbitrary. It can be used to define a *conduction band offset* between two different materials.

ValenceBandOffset

type
double

unit
`[eV]`

Energy value that defines the position of the average valence band edge energy $E_{\text{v,av}}$ on an absolute energy scale. The zero point of energy is arbitrary. It can be used to define a *valence band offset* between two different materials.

average valence band edge energy: $E_{\text{v,av}} = (E_{\text{hh}} + E_{\text{lh}} + E_{\text{so}})/3$

BandGap

type
double

unit
`[eV]`

Band gap at the Γ point given for temperature of $T = 0\,\text{K}$. The code automatically calculates the temperature dependent band gap using the *Varshni formula*. If the band gap is specified here for another temperature, the *Varshni parameters* `BandGapAlpha` and `BandGapBeta` should be set to zero.

BandGapAlpha

type
double

unit
`[eV/K]`

Varshni parameter α to allow for temperature dependent band gap.

BandGapBeta

type
double

unit
`[K]`

Varshni parameter β to allow for temperature dependent band gap.

Note: `BandGap`, `BandGapAlpha`, `BandGapBeta` are not used inside the calculation. They are just needed to output the valence band edge (which is not used either).

ElectronMass

type
double

unit
[m0]

Isotropic effective electron mass of the Γ conduction band.

EpsStatic

type
double

unit
[]

Static dielectric constant, *low* frequency dielectric constant ε_0

EpsOptic

type
double

unit
[]

Optical dielectric constant, *high* frequency dielectric constant :math:*varepsilon_ infty*

LOPhononEnergy

type
double

unit
[eV]

Longitudinal optical (LO) phonon energy E_{OP}.

This parameter must not be set to zero as there will be a divison by zero in this case, see p. 44 of PhD thesis of Peter Greck:

$N_{\mathrm{OP}} = \frac{1}{\exp(E_{\mathrm{OP}}/(k_{\mathrm{B}}T))-1} ... = 1/(1-1) = \mathrm{NaN}$ (*not a number*)

N_{OP} is the phonon distribution and a prefactor of the equation (eq. (7.5)) where the LO phonon scattering strength is calculated, i.e. if $N_{\mathrm{OP}} \ll 1$, then the LO phonon scattering is rather small.

$E_{\mathrm{OP}} = 35$ meV in GaAs

N_{OP} for GaAs	T [K]
$8 \cdot 10^{-45}$	4
$1.3 \cdot 10^{-6}$	30
0.000297	50
0.017524	100
0.151055	200
0.348148	300

LOPhononWidth

type
double

unit
`[eV]`

This is a numerical value that avoids reducing the coupling strength to a δ function:

$E + E_{\mathrm{OP}} \rightarrow E + E_{\mathrm{OP}} \pm \Delta E/2$, where $\Delta E =$ `LOPhononWidth`.

Note: The following 4 variables are only relevant for acoustic phonon scattering.

DeformationPotential

type
double

unit
`[eV]`

Scalar deformation potential

It is used for acoustic phonon scattering.

MaterialDensity

type
double

unit
`[kg/m^3]`

Material density or mass density

VelocityOfSound

type
double

unit
`[m/s]`

Sound velocity

AcousticPhononEnergy

type
double

unit
`[eV]`

Acoustic phonon energy

Note: The following 5 variables are only relevant for strain calculations.

Lattice_a

type
double

unit
[nm]

Lattice constant a

Elastic_c11

type
double

unit
[GPa]

Elastic constant c_{11}

Elastic_c12

type
double

unit
[GPa]

Elastic constant c_{12}

Elastic_c44

type
double

unit
[GPa]

Elastic constant c_{44}

Piezo_e14

type
double

unit
[C/m^2]

Piezoelectric constant e_{14}

9.4.2 Ternary compounds

For *ternary* compounds like $Al_xGa_{1-x}As$, we have to specify bowing parameters. The material parameters in many ternary alloys ($A_xB_{1-x}C$ or CA_xB_{1-x}) can be approximated in the form of the usual quadratic function

$T_{ABC} = xB_{AC} + (1-x)B_{BC} - x(1-x)C_{ABC}$

where C_{ABC} is the *bowing parameter*.

```
# ternary compound
    Material{
        Name = "In(x)Ga(1-x)As"
        Binary1 = "InAs(x)"
        Binary2 = "GaAs(1-x)"
        CrystalStructure = Zincblende
        ConductionBandOffset = 0                # Unit = eV
        ValenceBandOffset = -0.38               # Unit = eV
        BandGap = 0.477                         # Unit = eV
        BandGapAlpha = 0                        # Unit = eV/K
        BandGapBeta = 0                         # Unit = K
        DeltaSO = 0.15                          # Unit = eV
        ElectronMass = 0.0091                   # Unit = m0
        EpsStatic = 0
        EpsOptic = 0
        S = 3.54                                # Unit = eV
        Ep = -1.48                              # Unit = eV
        kp_8_bands{
            B = 0.0
            L = 0.0
            M = -1.140907266
            N = 0.0
        }

        DeformationPotential = 2.61             # Unit = eV
        ValenceDefPotHydro = 0                  # Unit = eV
        ValenceDefPotUniaxial_b = 0             # Unit = eV
        ValenceDefPotShear_d = 0                # Unit = eV
        MaterialDensity = 0                     # Unit = kg/m^3
        VelocityOfSound = 0                     # Unit = m/s
        LOPhononEnergy = 0                      # Unit = eV
        LOPhononWidth = 0                       # Unit = eV
        AcousticPhononEnergy = 0                # Unit = eV
        Lattice_a = 0                           # Unit = nm
        Elastic_c11 = 0                         # Unit = GPa
        Elastic_c12 = 0                         # Unit = GPa
        Elastic_c44 = 0                         # Unit = GPa
        Piezo_e14 = 0                           # Unit = C/m^2
    }
```

Note: Currently, the Varshni parameters α (`BandGapAlpha`) and β (`BandGapBeta`) are interpolated. It is better and more meaningful to interpolate the band gap instead.

9.4.3 Quaternary compounds

Quaternary compounds like $Al_xIn_yGa_{1-x-y}N$ are implemented as:

```
# quaternary compounds
    Material{
        Name = "Al(x)In(y)Ga(1-x-y)N"
        CrystalStructure = Wurtzite
        Alloy = "AlN(x);InN(y);GaN(1-x-y)"
        Binary1 = "AlN(x)"
        Binary2 = "InN(y)"
        Binary3 = "GaN(1-x-y)"
        Ternary_xy = "Al(x)In(1-x)N"
        Ternary_x = "Al(x)Ga(1-x)N"
        Ternary_y = "In(x)Ga(1-x)N"
        ConductionBandOffset = 0            # Unit = eV
        ValenceBandOffset = 0               # Unit = eV
    }
```

Final remark

It is recommended to use *position dependent* material parameters, i.e. for parameters like LO phonon energy, deformation potential, sound velocity, material density and acoustic phonon energy. Obviously, the Büttiker probes $B(x)$ depend on position. But in fact, the parameters for the wells are the most important ones. The parameters in the barriers only have a minor influence. One can include them in the calculation but the Büttiker probes in the barriers should not have any significant influence on the final result.

9.5 Simulation output

Output files of the simulation

Note: The output of a simulation can easily exceed 1 GB. Please make sure you have enough disk space available.

All files that have the file extension

- `.dat` can be plotted with *nextnanomat* or any other visualization software like e.g. Origin.
- `.gnu.plt` can be plotted with Gnuplot.
- `.fld` can be plotted with *nextnanomat* or AVS/Express.
- `.vtr` can be plotted with *nextnanomat*, ParaView or VisIt.

Note: Recommendation: Please install Gnuplot. It is then very convenient to plot the results of the *nextnano.MSB* calculations. Within *nextnanomat*, one can plot the band profile together with other data using *Keep current graph as overlay*.

9.5.1 Input

In this folder, all material and input parameters are contained.

Material parameters

- `BandEdge_conduction_input.dat`

  This is the conduction band edge profile that has been specified in the *input file*.

  `BandEdge_conduction_adjusted.dat`

  This is the conduction band edge profile that has been used in the *simulation*.

  The suffix `_adjusted` indicates that the well and barrier widths, as well as heights, had been adjusted automatically by the program.

  Why do the band edges for `*_input.dat` and `*_adjusted.dat` differ? See `AdjustBandedge` in *Input file* documentation for more information.

  conduction band edge in units of `[eV]`

  `Position [nm] Conduction Band Edge [eV]`

- `EffectiveMass.dat`

  electron effective mass in units of `[m0]`

  `Position [nm] Effective Mass [m0]`

- `EpsStatic.dat`

  static dielectric constant in units of `[]`

  `Position [nm] Relative Static Permittivity []`

- `EpsOptic.dat`

  optical dielectric constant in units of `[]`

  `Position [nm] Relative Optical Permittivity []`

- `MaterialDensity.dat`

  material density or mass density in units of `[kg/m^3]`

  `Position [nm] Material Density [kg/m^3]`

- `PhononEnergy_acoustic.dat`

  acoustic phonon energy in units of `[eV]`

  `Position [nm] Acoustic Phonon Energy [eV]`

- `PhononEnergy_LO.dat`

  longitudinal optical phonon energy (LO phonon energy) in units of `[eV]`

  `Position [nm] LO Phonon Energy [eV]`

- `PhononEnergy_LO_width.dat`

  width of longitudinal optical phonon energy (LO phonon energy) in units of `[eV]`

  For an explanation, see *Material database*.

  `Position [nm] LO Phonon Energy Width [eV]`

- `VelocityOfSound.dat`

  sound velocity in units of `[m/s]`

  `Position [nm] Velocity of Sound [m/s]`

Input parameters

- `AlloyContent.dat`

  alloy profile in units of `[]`

  `Position [nm] Alloy Content []`

- `DopingConcentration.dat`

  doping concentration in units of `[cm^-3]`.

  It is assumed that all dopants are ionized (*fully ionized*). An ionization model is not included.

  `Position [nm] Doping Concentration [1/cm3]`

- `ProbeValues.dat`

  profile of the Büttiker probes in units of `[]`, value is between `0` and `1`

  `Position [nm] Probe Values []`

9.5.2 Output

Energy profile

- `BandEdge_conduction.dat`

  conduction band edge in units of `[eV]`

  `Position [nm] Conduction Band Profile [eV]`

- `ElectrostaticPotential.dat`

  electrostatic potential in units of `[V]`

  `Position [nm] Electrostatic Potential [V]`

- `ElectricField.dat`

  electric field in units of `[kV/cm]`

  `Position [nm] Electric Field [kV/cm]`

Eigenstates

The data contained in this folder is not used inside the actual MSB algorithm. It is merely a post-processing feature. Once the self-consistently calculated conducton band edge,

$$E_{\mathrm{c}}(x) = E_{\mathrm{c},0}(x) - e\phi(x)$$

is known, the eigenenergies E_i and wave functions $\psi_i(x)$ of the single-band Schrödinger equation are calculated.

$\phi(x)$ is the electrostatic potential which is the solution of the Poisson equation.

$\mathbf{H}\psi(x) = E\psi(x)$

The Schrödinger equation is solved three times, i.e. with three different boundary conditions

- *periodic*: $\psi(x = 0) = \psi(x = L)$
- *Dirichlet*: $\psi(x = 0) = \psi(x = L) = 0$, and
- *Neumann*: $\frac{\mathrm{d}\psi}{\mathrm{d}x} = 0$ at the left ($x = 0$) and right ($x = L$) boundary.

There are files for the

- amplitudes $\psi_i(x)$ in units of `[nm^-1/2]`

  `Amplitudes_Dirichlet.dat` / `*_Neumann.dat` / ``*_Periodic.dat`

- amplitudes $\psi_i(x)$ *shifted* by their eigenenergies E_i

  `Amplitudes_shift_Dirichlet.dat` / `*_Neumann.dat` / `*_Periodic.dat`

- probability densities $\psi_i^2(x)$ in units of `[nm^-1]`

  `Probabilities_Dirichlet.dat` / `*_Neumann.dat` / `*_Periodic.dat`

- probability densities $\psi_i^2(x)$ *shifted* by their eigenenergies E_i

  `Probabilities_shift_Dirichlet.dat` / `*_Neumann.dat` / `*_Periodic.dat`

- eigenvalues E_i in units of `[eV]`

  `Eigenvalues_Dirichlet.dat` / `*_Neumann.dat` / `*_Periodic.dat`

CarrierDensity

- The position and energy resolved electron density $n(z, E)$ is contained in this file:

  `CarrierDensity_energy_resolved.avs.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

- The electron density $n(z)$ is contained in this file:

  `CarrierDensity.dat` / `*.gnu.plt`

  `Position [nm] Density [1/cm^3]`

DOS

- `DOS_position_resolved.avs.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  The position and energy resolved local density of states (LDOS) $\rho(z, E)$ in units of `[eV-1 nm-1]`.

- `DOS.dat` / `*.gnu.plt`

  `Energy [eV] DOS [1/eV]`

  The density of states (DOS) $n(E)$.

The density of states is the sum of the DOS due to source, drain and Büttiker probes, i.e.

`DOS = DOS_Source + DOS_Drain + DOS_Probes.`

- `DOS_Probes_position_resolved.avs.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  The position and energy resolved density of states (LDOS) $\rho(z, E)$ due to the Büttiker probes only in units of `[eV-1 nm-1]`.

  This DOS is induced by scattering events. Like the lead-connected DOS enters the device through the source or drain contacts, respectively, the probe DOS is due to scattering.

  Here we plot the LDOS for the probes, i.e. all probes are summed up, and the LDOS of the probes is determined by the self-energies of the probes. A probe has the scattering strength $B = B_{\text{AC}} + B_{\text{LO}}$.

  From this plot one cannot see if the DOS is due to LO or AC scattering events as both scattering potentials are added to obtain B.

  In fact, as one considers the probes for each grid point individually, one could print out the LDOS for each grid point. So each probe grid point produces a probe spectral function $A_{\text{p}}(z, E)$, e.g. the probe at grid point 5 produces the *grid point 5 connected local density of states* which is nonzero not only on grid point #5 but everywhere.

  Each probe has its own chemical potential μ, e.g. the probe at grid point 5 has μ_5. Then the LDOS of probe 5 $\rho_5(z, E)$ is occupied everywhere with this chemical potential μ_5. In our algorithm, we only have one probe at each grid point having the combined scattering potential $B = B_{\text{AC}} + B_{\text{LO}}$. In principle, each grid point could have 2 probes, one for AC and one for LO phonon scattering. However, this is not the case in our algorithm so far.

- `DOS_Probes.dat` / `*.gnu.plt`

  `Energy [eV] DOS [1/eV]`

  The density of states (DOS) $n(E)$ due to the *Büttiker probes* only (probe-connected DOS).

- `DOS_Lead_Source_position_resolved.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  `DOS_Lead_Drain_position_resolved.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  The position and energy resolved local density of states (LDOS) $\rho(z, E)$ due to the *source* and *drain* contact only in units of `[eV-1 nm-1]`.

- `DOS_Lead_Source.dat` / `*.gnu.plt`

  `DOS_Lead_Drain.dat` / `*.gnu.plt`

  `Energy [eV] DOS [1/eV]`

  The density of states (DOS) $n(E)$ due to the *source* and *drain* contact only (lead-connected DOS).

- `DOS_Leads_position_resolved.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  The position and energy resolved local density of states (LDOS) $\rho(z, E)$ due to the *drain* and *source* contacts in units of `[eV-1 nm-1]`.

  This corresponds to the sum of

  `DOS_Lead_Source_position_resolved.fld` + `DOS_Lead_Drain_position_resolved.fld`.

- `DOS_Leads.dat` / `*.gnu.plt`

  `Energy [eV] DOS [1/eV]`

  The density of states (DOS) $n(E)$ due to the *drain* and *source* contacts (lead-connected DOS). This corresponds to the sum of

  `DOS_Lead_Source.dat` + `DOS_Lead_Drain.dat`.

Probes

- `ProbeLevels.dat`

  This output depends on the probe model used: `ProbeMode`

  a) `ProbeMode = iterative # Comment="Specify method to calculate current conservation."`

  local Büttiker probe *virtual chemical potentials* μ_p `[eV]` related to the occupation of the probes

  `Position [nm] Local Probe Levels [eV]`

  For zero applied bias, the local probe levels are `0 [eV]` which is the same value as the chemical potentials of the source and drain contacts as there is no current flowing. The probe levels indicate the occupation of the scattering states.

  b) `ProbeMode = direct # Comment="Specify method to calculate current conservation."`

  local Büttiker probe *coefficients* c_p `[]` (dimensionless)

  `Position [nm] Local Probe Levels (% of Drain) [0..1]`

  Here, the units are dimensionless and the numbers are between `0` and `1`.

  `0` means 100 % occupation of the probes by the *source* contact. `1` means 100 % occupation of the probes by the *drain* contact.

  For zero applied bias, the local probe levels are `0.5`, i.e. 50 % occupation due to source and 50 % due to drain contact.

  See also the comments on `ProbeMode` in the documentation of the *Input file*.

There is only one $B(z, E)$ for which current conservation holds. Once this quantity has been calculated, one cannot distinguish any more between optical and acoustic phonon scattering.

If the command line argument `-debug 1` is provided, additional output is written to this folder.

- `NumericalPrefactor_MSB_AC.dat`

  `NumericalPrefactor_MSB_LO.dat`

  The numerical prefactors for the MSB scattering potentials for acoustic phonon (AC) and LO phonon scattering are given in units of `[...]`. (*Add correct units here.*)

  - For LO, the prefactor is given in eq. (7.9) of the PhD thesis of P. Greck. It reads:

    $B_{\mathrm{OP}} \sim \frac{e^2 \zeta E_{\mathrm{LO}}}{32\pi\varepsilon_0} \left(\varepsilon_{\mathrm{optic}}^{-1} - \varepsilon_{\mathrm{static}}^{-1}\right)$

  - For AC, the prefactor is given after eq. (7.8) of the PhD thesis of P. Greck. It reads:

    $B_{\mathrm{AP}} \sim \frac{V_{\mathrm{D}}^2 k_{\mathrm{B}} T}{8\pi\rho_{\mathrm{M}} v_{\mathrm{s}}^2 E_{\mathrm{AP}}}$

  The prefactors are independent of applied bias voltage.

- `ScatteringPotential_MSB_AC.dat`

  `ScatteringPotential_MSB_LO.dat`

  The scattering potentials for MSB for acoustic phonon (AC) and LO phonon scattering are given in units of `[...]`. *It is not* `[nm]` *as written in the output file.*

  The scattering potential for LO phonons B_{OP} is given in eq. (7.9) of the PhD thesis of P. Greck.

  The scattering potential for acoustic phonons B_{AP} is given after eq. (7.8) of the PhD thesis of P. Greck.

- `ScatteringPotential_MSB_AC_position_resolved.dat`

  `ScatteringPotential_MSB_LO_position_resolved.dat`

  The position resolved scattering potentials for MSB for acoustic phonon (AC) and LO phonon scattering are given in arbitrary units.

Gain

- `gain_energy_resolved.avs.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  The position and energy resolved optical gain $g(z, E)$ in units of `[eV-1 cm-1]`.

  Here, energy E is the photon energy.

- `gain_frequency_resolved.avs.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  The position and frequency resolved optical gain $g(z, \nu)$ in units of `[THz-1 cm-1]`.

  Here, frequency ν is the photon frequency.

- `gain_wavelength_resolved.avs.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  The position and wavelength resolved optical gain $g(z, \lambda)$ in units of `[µm-1 cm-1]`.

  Here, wavelength λ is the photon wavelength.

- `gain_energy.dat` / `*.gnu.plt`

  The optical gain as a function of photon energy $g(E)$ in units of `[cm^-1]`.

  `Photon Energy [eV] Optical Gain [1/cm]`

- `gain_frequency.dat` / `*.gnu.plt`

  The optical gain as a function of frequency $g(\nu)$ in units of `[cm^-1]`.

  `Photon Frequency [THz] Optical Gain [1/cm]`

- `gain_wavelength.dat` / `*.gnu.plt`

  The optical gain spectrum as a function of photon wavelength $g(\lambda)$ in units of `[cm^-1]`.

  `Photon wavelength [µm] Optical Gain [1/cm]`

Negative values of the *optical gain spectrum* correspond to *optical absorption spectrum*.

Gain-voltage characteristics

- `GainMaxFrequency-Voltage.dat` / `*.gnu.plt`

  `Source [V] Drain [V] Frequency of Max. Gain [THz]`

  `0 0 2.41798940e-001`

  This file shows the frequency of the maximum value of the gain as a function of voltage. The first two columns contain the source and drain voltages. The third column is the frequency of the maximum gain at this voltage.

- `GainMaxFrequency-Voltage_Source.dat`

  `GainMaxFrequency-Voltage__Drain.dat`

  These files contain the same as discussed above but here only the source or drain voltages are contained, respectively, i.e. only one column for the voltages instead of two. It is easier to plot the data from one of these files compared to `GainMaxFrequency-Voltage.dat`.

- `GainMax-Voltage.dat` / `*.gnu.plt`

  `Source [V] Drain [V] Max. Gain [1/cm]`

  `0 0 -1.46451103e+000`

  This file shows the maximum value of the gain as a function of voltage in units of `[cm^-1]`. The first two columns contain the source and drain voltages. The third column is the maximum gain at this voltage. From this file, one can extract the voltage for threshold of gain.

- `GainMax-Voltage_Source.dat`

  `GainMax-Voltage__Drain.dat`

  These files contain the same as discussed above but here only the source or drain voltages are contained, respectively, i.e. only one column for the voltages instead of two. It is easier to plot the data from one of these files compared to `GainMax-Voltage.dat`.

Transmission

- `Transmission.dat`

  Transmission $T(E)$ in units of `[eV]`

  `Energy [eV] Transmission (Source->Drain)`

  Does the transmission have a meaning in the actual calculation? Yes, it adds the ballistic part, i.e. the tunneling from source to drain to the current (compare with Landauer formula (*insert reference*)), see thesis page 65ff in PhD thesis of Peter Greck (*check this*).

  It has been calculated from the self-consistently obtained conduction band profile. The transmission function is only the coherent ballistic contribution to the current, i.e. the current that goes directly from source to drain. The meaning of this output should be interpreted with care. There is also a noncoherent contribution to the current.

If one does a ballistic calculation then the total current is based on this transmission function (see Landauer formula).

Current density

- `current_density_energy_resolved.avs.fld` (or the corresponding `*.gnu.plt` / `*.dat` file)

  position and energy resolved current density $j(z, E)$

- `current_density.dat / *.gnu.plt`

  current density $j(z)$

  `Position [nm] Current Density [A/cm^2]`

Current-voltage characteristics (I-V curve)

- `Current-Voltage.dat / *.gnu.plt`

  `Source [V] Drain [V] Current [A/cm^2]`

  `0 0 0.00000000e+000`

  This file contains the current through the device (current-voltage or I-V characteristics). The first two columns contain the source and drain voltages. The third column is the current density.

- `Current-Voltage_Source.dat`

  `Current-Voltage_Drain.dat`

  These files contain the same as discussed above but here only the source or drain voltages are contained, respectively, i.e. only one column for the voltages instead of two. It is easier to plot the I-V characteristics from one of these files compared to `Current-Voltage.dat`.

9.6 Log file

Log file of the simulation (`*.log`)

```
nextnano.MSB v1.0.1(B9)
*********************************************************************
Wednesday, 2014-11-05, 14:22:02.
*********************************************************************
Parsing command line
License: E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release beta␣
↪9 - my\License\License.xml
Filename: E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period\THz_QCL_
↪NovelDesignPeterGreck_1period.xml
Output: E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period
Materials: E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release␣
↪beta 9 - my\nextnano.MSB\x64\..\Materials.xml
Materials: E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release␣
↪beta 9 - my\nextnano.MSB\Materials_THz_QCL_NovelDesignPeterGreck_no_
↪Varshni.xml
*********************************************************************
Checking license "E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB -␣
↪Release beta 9 - my\License\License.xml"
License info
Name: Stefan Birner
Email: abc@def.com
Expiration: Wednesday, 2014-12-31
Features
    MSB-1Band, OutputAvs, OutputGnuPlt.
*********************************************************************
```

(continues on next page)

(continued from previous page)

```
Hardware concurrency: 8 <== Returns the number of hardware thread contexts.
************************************************************
Loading "E:\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release beta 9 - my\
↪nextnano.MSB\Materials_THz_QCL_NovelDesignPeterGreck_no_Varshni.xml"
Database info
    Materials: 12 [#]
    Names: GaAsAlAs, InAs, InSb, GaSb,
        AlGaAs, AlGaAs_bowing_high, AlGaAs_bowing_low, InGaAs,
        GaAsSb, InAlAs, InAsSb
************************************************************
Loading "E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period\THz_QCL_
↪NovelDesignPeterGreck_1period.xml"
Header info
```

Here the information written in the input file within `Header` is printed.

```
Header{
    ...
}
```

Information on variables and sweeps, such as temperatue sweep `T = 100,250`.

```
Variables info
    QWDoping = 3e16
    QW_width_1 = 7.35
    Barrier_width_high = 2.1
    BarrierHigh = 0.27
    T = 100,250
************************************************************
```

Information on parameter sweep, such as temperature `T = 100`.

```
Parameter sweep step 1 of 2
Variables info
    QWDoping = 3e+016
    QW_width_1 = 7.35
    Barrier_width_high = 2.1
    BarrierHigh = 0.27
    T = 100
Output info
    Directory: E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period
Device info
    Temperature: 100 [K]
    Grid info
    Spacing: 0.5 [nm]
```

The following information on the layers is only printed if `-debug 1` is specified as command line argument.

```
Layer info
Width: 4.5 [nm]
Doping: 3e-005 [nm^-3]
AdjustBandedge: yes [yes|no]
Probes: 100 [%]
Material info
    Base: GaAs
Layer info
Width: 2.25 [nm]
```

(continues on next page)

(continued from previous page)

```
Doping: 0 [nm^-3]
AdjustBandedge: yes [yes|no]
Probes: 100 [%]
Material info
    Base: AlGaAs
    AlloyX: 27 [%]
Layer info
Width: 7.25 [nm]
Doping: 0 [nm^-3]
AdjustBandedge: yes [yes|no]
Probes: 100 [%]
Material info
    Base: GaAs
```

Information on the `Source` and `Drain` leads.

```
    Lead info
    Name: Source
    Voltage info
        Value: 0 [V]
    Lead info
    Name: Drain
    Voltage info
        Initial: 0 [V]
        Delta: 0.036 [V]
        Steps: 2 [#]
        Cluster: Center
Initialize Device
```

The following information is only printed if `-debug 1` is specified as command line argument.

```
Layer
Width: 4.5 [nm]
Nodes: 18 => Effective width 4.5nm (Error=0%)
Material GaAs
    Properties
    Mass: 0.067 [m0]
    Bandedge:
        Offset: -1.519 [eV]
        Gap: 1.519 [eV]
        Alpha: 0 [eV/K]
        Beta: 0 [K]
    EpsStatic: 12.93 [eps_0]
Layer
Width: 2.25 [nm]
```

Automatic adjustment of layer width and height.

```
Nodes: 9 => Effective width 2.25nm (Error=0%)
```

`Error` is related to `AdjustBandedge` feature. Here, grid spacing is 0.25 `[nm]` which is optimal. Therefore the `Error` is `0%`.

```
Nodes: 5 => Effective width 2.5nm (Error=11.1111%)
```

Here, grid spacing is 0.50 `[nm]` which is too coarse to resolve the layer width. Therefore the `Error` is larger than `0%`.

```
Material AlGaAs
    Alloy: 27% of AlAs and 73% of GaAs
    Properties
    Mass: 0.08941 [m0]
    Bandedge:
        Offset: -1.6621 [eV]
        Gap: 1.90092 [eV]
        Alpha: 0 [eV/K]
        Beta: 0 [K]
    EpsStatic: 12.1562 [eps_0]
```

The following information is only printed if `-debug 1` is specified as command line argument.

```
Material AlGaAs_bowing_high
Alloy: 27% of AlAs and 73% of GaAs
Properties
Mass: 0.08941 [m0]
Bandedge:
Offset: -1.6621 [eV]
Gap: 1.90092 [eV]
Alpha: 0 [eV/K]
Beta: 0 [K]
EpsStatic: 12.1562 [eps_0]
Layer
Width: 7.25 [nm]
Nodes: 15 => Effective width 7.5nm (Error=3.44828%)
```

Log file continues…

```
    Building parameter grids
    Total nodes: 98 [#]
    Building cluster maps
        Center: 28 -> 65 [#]
        QCL: 28 -> 65 [#]
    Lead "Source"
    Voltage is fixed at 0 V
    Lead "Drain"
    Cluster factor: 2.57895
    Voltage from 0 mV to 92.8421 mV in 2 steps
Kernel info
    MaxGreenIts: 25
    MaxGreenDelta: 3e-07
    MaxProbeIts: 15
    MaxProbeNorm: 1e-009 [A/cm^2]
    MaxPoissonOuterIts: 25
    MaxPoissonInnerIts: 15
    MaxPoissonDensityNorm: 1e+009 [1/cm^3]
    MaxPoissonCorrectorNorm: 0.0001 [V]
    Core info
    IterativeProbes: 0
    Flatband: 0
    Scattering strength
        Const: 0 [eV]
        BP: 0 [eV]
        MSB: 1 []
    Gain info
    PhotonEnergyInitial: 0.001 [eV]
```

(continues on next page)

(continued from previous page)

```
    PhotonEnergyFinal: 0.03 [eV]
    PhotonEnergySteps: 117 [#]
    Cluster: Center
Kernel
    Building Hamiltonian
Calculating
```

Voltage sweep step starts and MSB self-consistency cycle.

```
Voltage sweep step 1 of 2
    Voltage info
    Source: 0 [V]
    Drain: 0 [V]

    MSB self-consistency cycle
    it= 1, norm2=2.478388e+003 [%]
           normI=6.779549e+001 [%]
    it= 2, norm2=3.960580e+002 [%]
           normI=2.362588e+001 [%]
    ...
    it= 5, norm2=7.928989e-001 [%]
           normI=3.995536e-002 [%]
    Calculating Green's functions
    Poisson iteration 0
    it= 0, norm2=1.836019e+017 [cm^-3]
           normI=2.943621e+016 [cm^-3]
           charge=17 [%]
    it= 1, norm2=1.511046e+018 [cm^-3]
           normI=3.661754e+017 [cm^-3]
           charge=1024 [%]
           ...
    it= 7, norm2=1.771342e+007 [cm^-3]
           normI=4.557251e+006 [cm^-3]
           charge=100 [%]
    Corrector: norm2=1.661272e-001 [V]
               normI=1.766968e-002 [V]
    Writing output
    Device
    Density of States
    Transmissions
    Probes
    Carrier Density
    Current Density
    Consistency checks
    DOS: 1.032193e-013 [1/eV/nm]
```

This DOS value is `MAX( ABS(DOS_total - DOS_Source - DOS_Drain - DOS_Probes) )`. It must be (numerically) zero.

```
    Current: 0.000000e+000 [A/cm^2]
    Current: nrm2=1.311401e-011 [A/cm^2]
    Current: nrmi=8.209396e-012 [A/cm^2]
    Current: diff=1.244363e-013 [A/cm^2]

    MSB self-consistency cycle
     it= 1, norm2=4.457296e+002 [%]
            normI=2.933419e+001 [%]
```

(continues on next page)

(continued from previous page)

```
    ...
    it= 4, norm2=1.054857e-001 [%]
           normI=6.648939e-003 [%]
    Calculating Green's functions
    Poisson iteration 1
    it= 0, norm2=1.920755e+015 [cm^-3]
           normI=7.899763e+014 [cm^-3]
           charge=100 [%]
    ...
    it= 2, norm2=2.411963e+005 [cm^-3]
           normI=1.151118e+005 [cm^-3]
           charge=100 [%]
    Corrector: norm2=1.587036e-004 [V]
               normI=3.504813e-005 [V]
    Calculating optical gain
    Setup gain calculation
    Calculate Green's functions
    Calculate gain for photon energies
    Write output
    Writing output
    Device
    Density of States
    Transmissions
    Probes
    Carrier Density
    Current Density
    Consistency checks
    DOS: -4.641658e-007 [1/eV/nm]
    Current: 0.000000e+000 [A/cm^2]
    Current: nrm2=5.709931e-011 [A/cm^2]
    Current: nrmi=3.783379e-011 [A/cm^2]
    Current: diff=1.383798e-012 [A/cm^2]
Duration of Voltage Step
    Duration in seconds: 16
    Duration in hours, minutes, seconds: 0h, 0', 16''
Voltage sweep step 2 of 2
    Voltage info
    Source: 0 [V]
    Drain: 0.0928421 [V]
    MSB self-consistency cycle
    ...
    Voltage drop=100 [%]
    Writing output
    Device
    ...
    it= 3, norm2=3.391875e+005 [cm^-3]
           normI=9.720889e+004 [cm^-3]
           charge=100 [%]
    Corrector: norm2=6.652502e-003 [V]
               normI=9.127917e-004 [V]
    Voltage drop=105 [%]
    ...
    Calculating Green's functions
    Probe calculation
```

Second Poisson iteration starts. It is related to `MaxPoissonOuterIts`. Make sure that this number is not exceeded!

```
    Poisson iteration 2
    it= 0, norm2=5.254027e+015 [cm^-3]
    ...
    it= 2, norm2=3.598211e+005 [cm^-3]
            normI=1.012272e+005 [cm^-3]
            charge=100 [%]
    Corrector: norm2=7.631612e-005 [V]
                normI=8.710143e-006 [V]
    Voltage drop=100 [%]
    Calculating optical gain
    Setup gain calculation
    Calculate Green's functions
    Calculate gain for photon energies
    Write output
    Writing output
    Device
    Density of States
    Transmissions
    Probes
    Carrier Density
    Current Density
Consistency checks
    DOS: -7.571542e-007 [1/eV/nm]
    Current: 3.251042e+003 [A/cm^2]
    Current: nrm2=2.033016e-007 [A/cm^2]
    Current: nrmi=4.489969e-008 [A/cm^2]
    Current: diff=4.750423e-008 [A/cm^2]
Duration of Voltage Step
    Duration in seconds: 31
    Duration in hours, minutes, seconds: 0h, 0', 31''
Duration of Parameter Sweep Step
Duration in seconds: 47
Duration in hours, minutes, seconds: 0h, 0', 47''
****************************************************************
```

`Parameter sweep step` for `T = 250 [K]`.

```
Parameter sweep step 2 of 2
Variables info
    QWDoping = 3e+016
    ...
    T = 250
Write output
    Writing output
    Device
    Density of States
    Transmissions
    Probes
    Carrier Density
    Current Density
Consistency checks
    DOS: -1.721273e-006 [1/eV/nm]
    Current: 3.285627e+003 [A/cm^2]
    Current: nrm2=5.173793e-009 [A/cm^2]
    Current: nrmi=1.212275e-009 [A/cm^2]
    Current: diff=1.152515e-009 [A/cm^2]
```

Information on simulation time for `Voltage Step`, `Parameter Sweep Step` and `Main Program`.

```
Duration of Voltage Step
Duration in seconds: 34
Duration in hours, minutes, seconds: 0h, 0', 34''
Duration of Parameter Sweep Step
Duration in seconds: 50
Duration in hours, minutes, seconds: 0h, 0', 50''

************************************************************
Wednesday, 2014-11-05, 14:23:39.
************************************************************
Duration of Main Program
Duration in seconds: 97
Duration in hours, minutes, seconds: 0h, 1', 37''
```

9.7 Tutorials

9.7.1 Transmission coefficient of a double barrier structure

This tutorial is based on the following publication: *[BirnerCBR2009]*

Input files:

- *1D_Transmission_DoubleBarrier_CBR_paper.in* (input file for *nextnano*[3] code)
- *1D_Transmission_DoubleBarrier_CBR_paper_MSB.negf* (input file for *nextnano.MSB*)
- *Materials_1D_Transmission_DoubleBarrier_CBR_paper_MSB.negf* (material database for *nextnano.MSB*)

This example input file demonstrates how to calculate the transmission coefficient.

We use the same structure as outlined in Section 4.4 of *[BirnerCBR2009]* and compare the results obtained with the MSB method to the results obtained with the *CBR* method.

Our structure consists of

GaAs | **AlGaAs** | GaAs | **AlGaAs** | GaAs

where the barrier material is indicated in bold.

The following figure shows the conduction band edge profile and the probability density of this quasi-bound resonant state for the case of 10 nm barrier widths calculated with *nextnano*[3].

The barrier height is set to 0.1 eV.

The effective mass is set to 0.067 m_0 for both materials.

The widths of the AlGaAs barriers is varied between 2 nm, 4 nm and 6 nm. The *nextnanomat* feature *Template* or nextnanopy Sweep can be used to sweep the variables before execution.

```
$BarrierThickness = 2      # (DisplayUnit:nm) barrier thicknesses␣
↪(ListOfValues:2,4,10)
```

This simply generates multiple input files, which can be run in parallel (see *Options: Simulation*).

In the XML format, the iterator feature was used:

```
<Iterator>
 <Name Comment="barrier thicknesses">BarrierThickness</Name>
 <Values Unit="nm">2,4,10</Values>
</Iterator>
```

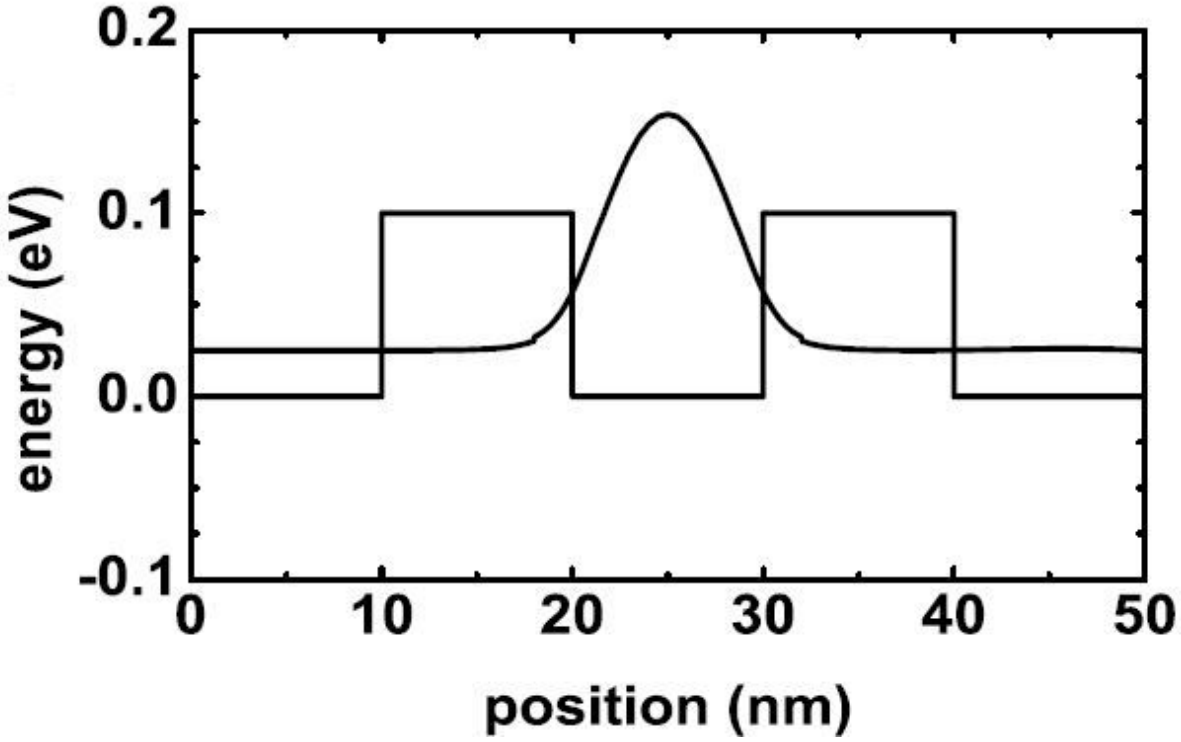


Figure 9.7.1.1: Conduction band edge and ground state probability density of the double barrier structure

In this case, single program execution simulates all the cases.

The following figure shows the calculated transmission coefficient $T(E)$ as function of energy for a double barrier structure with varying barrier widths of 2 nm, 4 nm and 10 nm (barrier separation 10 nm). At 25 meV there is a peak where the double barrier becomes transparent, i.e. $T(E) = 1$. This is exactly the energy that matches the resonant state in the well.

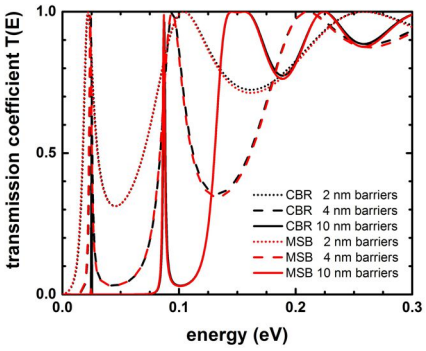


Figure 9.7.1.2: Transmission function for double barriers of different barrier thicknesses

The MSB results are very similar to the CBR results (see also Fig. 1 in *[BirnerCBR2009]*). The conduction band edge minimum is set to 0 eV. Note that the first peak of the 10 nm barrier structure is not well resolved for the MSB calculation in contrast to the CBR calculation. The reason is that the associated confined energy level is very sharp for the 10 nm structure and couples hardly to the source and drain leads. Therefore, the resolution of the energy grid plays an important role in order to resolve the energy peak.

The following figure shows the local density of states (LDOS), $\rho(z, E_z)$, for the double barrier structure with a barrier thickness of 2 nm. The conduction band edge is indicated by the white line. The barrier height is 0.1 eV. The results look very similar to Fig. 2 in `[BirnerCBR2009]` calculated with the CBR method.

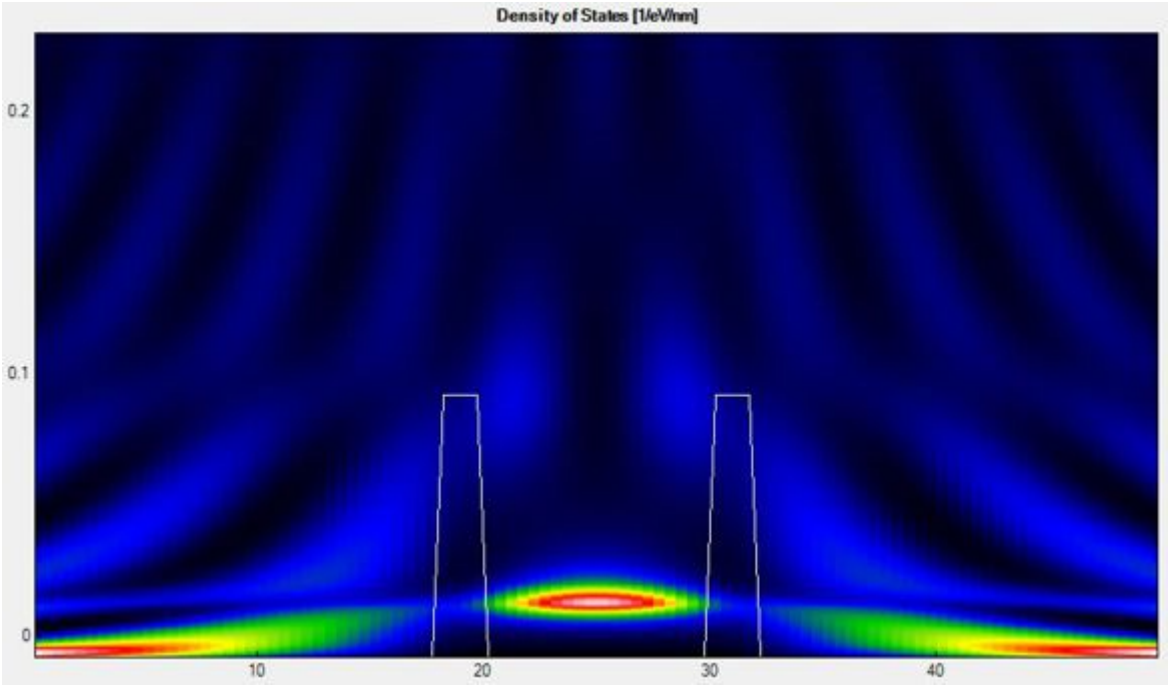


Figure 9.7.1.3: Local Density of states (LDOS) for the double barrier with 2 nm barrier thickness

The resonant state inside the double barrier is very broad with respect to energy because it couples strongly to the leads at the left and right boundaries. This is in contrast to the situation for the 10 nm barriers (not shown) where due to the large barrier widths the resonant state is quasi-bound, i.e. with a very sharp and high density of states at the resonance energy because of the very weak coupling to the contacts. Red (blue) color indicates high (low) density of states.

The following figure shows the calculated density of states $n(E)$ for the double barrier structures.

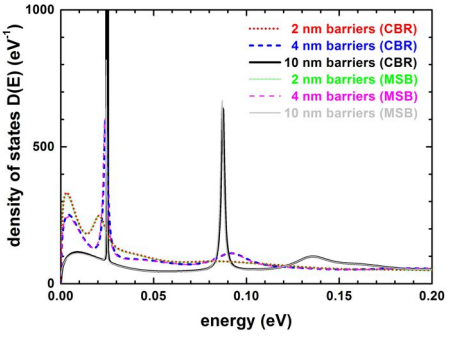


Figure 9.7.1.4: Density of states (DOS) for double barriers of different barrier thicknesses

The first peak in the DOS for the 10 nm barrier structure differs substantially from the other two structures because it is extremely sharp and high. The second peak in the DOS at 87 meV due to the second confined well state is only visible for the 10 nm structure. This is consistent to the transmission coefficient (see Fig. above) which shows a sharp maximum only for the 10 nm structure at this energy.

Further comments regarding the MSB input file

In order to avoid solving the Poisson equation for this example we use the following parameters for the number of Poisson iterations.

```
MaxPoissonOuterIts = 1      # Comment = Max. outer Poisson iterations where␣
↪G^R is recalculated.
```

(continues on next page)

(continued from previous page)

```
MaxPoissonInnerIts = 0      # Comment = Max. inner Poisson iterations where␣
↪the density predictor is used.
```

In the database we have adjusted the material parameters so that we obtain a barrier height of 0.1 eV and a constant effective mass of 0.067 m_0 for both materials.

Please help us to improve our tutorial. If you have any questions or comments, create a ticket nextnano Help Center.

9.7.2 Ballistic current calculation of a GaAs nin resistor

This tutorial is based on the following publications: *[BirnerCBR2009]*, *[GreckPhD2012]*

The following input files were used:

- Input files for *nextnano*[3]
  - `1D_nin_symmetric_PhDthesis_PeterGreck_cl_zero_bias.in`

    classical calculation at zero bias, Fig. 8.1(a) of *[GreckPhD2012]*
  - `1D_nin_symmetric_PhDthesis_PeterGreck.in`

    CBR calculation of Fig. 8.1 and Fig. 8.2 of *[GreckPhD2012]*
  - `1D_nin_symmetric_CBRpaper.in`

    CBR calculation of Chapter 8.5 of *[BirnerCBR2009]*, symmetric doping profile
  - `1D_nin_asymmetric_CBRpaper.in`

    CBR calculation of Chapter 8.5 of *[BirnerCBR2009]*, asymmetric doping profile
- Input files for *nextnano.MSB*
  - `nin_resistor_PhDthesis_PeterGreck.xml`

    MSB calculation of Fig. 8.1 and Fig. 8.2 of *[GreckPhD2012]*
  - `nin_resistor_CBRpaper_symmetric.xml`

    MSB calculation of Chapter 8.5 of *[BirnerCBR2009]*, symmetric doping profile
  - `nin_resistor_CBRpaper_asymmetric.xml`

    MSB calculation of Chapter 8.5 of *[BirnerCBR2009]*, asymmetric doping profile

These example input files demonstrate how to calculate the ballistic current on a GaAs nin resistor. We use the same structure as outlined in Section 8.1 of *[GreckPhD2012]* and section 8.5 of *[BirnerCBR2009]* and compare the results obtained with the MSB method to the results obtained with the *CBR* method.

Example 1

The following figure shows the electron density and the conduction band edge profile for a 50 nm GaAs nin structure.

The following input files can be used to reproduce these results.

- `1D_nin_symmetric_PhDthesis_PeterGreck_cl_zero_bias.in` (input file for *nextnano*[3] code)

  classical calculation at zero bias, Fig. 8.1(a) of *[GreckPhD2012]*
- `1D_nin_symmetric_PhDthesis_PeterGreck.in` (input file for *nextnano*[3] code)

  CBR calculation of Fig. 8.1 and Fig. 8.2 of *[GreckPhD2012]*

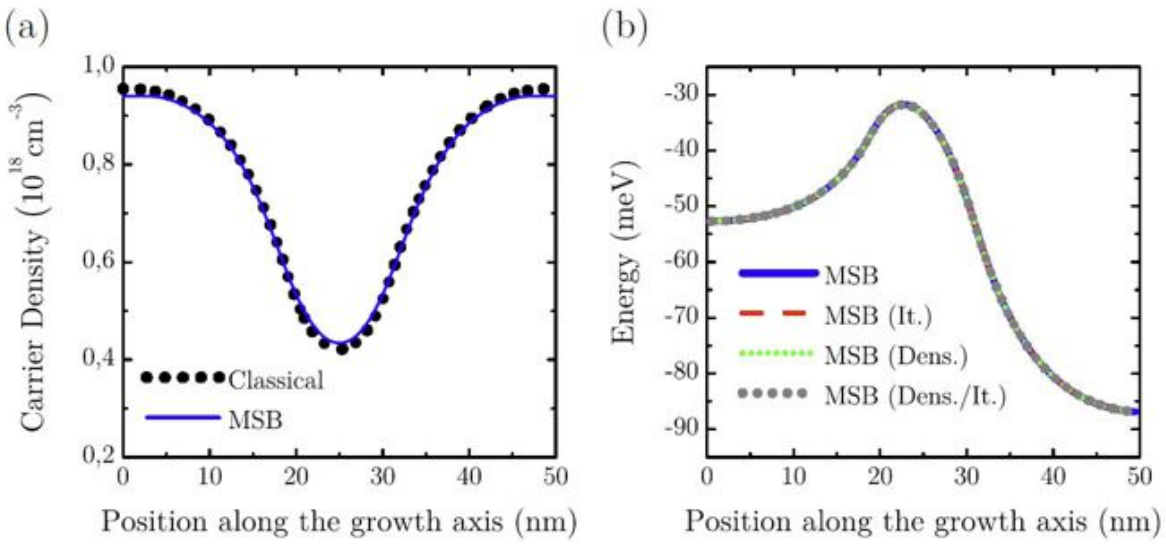


Figure 8.1: (a) Comparison of the calculated carrier density for an MSB calculation (solid blue line) and a semi-classical calculation (dotted black line) for a 50 nm GaAs *nin* resistor at zero bias voltage and 300 K. (b) Calculated conduction band edge profiles for the GaAs resistor for an applied bias voltage of 50 mV. All profiles are calculated with the MSB method but with different approximations and probe types as described in the text. Note that all four profiles almost perfectly match.

- `nin_resistor_PhDthesis_PeterGreck.xml` (input file *nextnano.MSB*)

  MSB calculation of Fig. 8.1 and Fig. 8.2 of *[GreckPhD2012]*

The MSB results are very similar to the CBR results (not shown). Note that the CBR method uses an adaptive energy grid

```
adaptive-energy-grid = adaptive-exponential
```

to improve convergence while the MSB method uses a constant energy grid spacing.

The following results of the CBR input files are consistent to the MSB results for both, zero bias and nonzero bias. This has been checked.

- conduction band edges
- electric field
- electron density and energy resolved electron density
- DOS, position resolved DOS, lead resolved DOS (although the numbers are slightly different but the energy grid is also slightly different, i.e. nonuniform (CBR) vs. uniform (MSB) energy grid)
- Transmission
- IV curve

Example 2

The following input files can be used to reproduce the results of section 8.5 of *[BirnerCBR2009]*.

- `1D_nin_symmetric_CBRpaper.in` (input file for *nextnano*[3] code)

  CBR calculation of Chapter 8.5 of *[BirnerCBR2009]*, symmetric doping profile

- `1D_nin_asymmetric_CBRpaper.in` (input file for *nextnano*[3] code)

  CBR calculation of Chapter 8.5 of *[BirnerCBR2009]*, asymmetric doping profile

- `nin_resistor_CBRpaper_symmetric.xml` (input file for *nextnano.MSB*)

  MSB calculation of Chapter 8.5 of *[BirnerCBR2009]*, symmetric doping profile

- `nin_resistor_CBRpaper_asymmetric.xml` (input file for *nextnano.MSB*)

  MSB calculation of Chapter 8.5 of *[BirnerCBR2009]*, asymmetric doping profile

Again, we obtained very good agreement between the CBR and the MSB algorithm.

Important comment

Here, we had to use more energy grid points compared to the CBR code (where we used `300`) in order to obtain convergence.

```
Nodes = 501                                    # Comment = Number of energy␣
↪grid points.
```

We noticed this because the number of outer Poisson iterations exceeded its maximum as can be seen in the *.log file*

```
Poisson iteration 60
```

The number of outer Poisson iterations should be increased as follows.

```
MaxPoissonOuterIts = 60    # Comment = Max. outer Poisson iterations where G^
↪R is recalculated.
```

Further comments regarding the MSB input file

In order to calculate the current ballistically we switched of scattering by using the following flag.

```
BallisticCalculation = yes
```

Please help us to improve our tutorial. If you have any questions or comments, create a ticket nextnano Help Center.

9.7.3 Resonant tunneling diode (RTD)

This tutorial is based on the following publications: *[GreckPhD2012]*

The following input files were used:

- *RTD_PhDthesis_PeterGreck_ballistic.negf* (MSB calculation of Fig. 8.2 of *[GreckPhD2012]*)
- *RTD_PhDthesis_PeterGreck.negf* (MSB calculation of Fig. 8.2 of *[GreckPhD2012]*)

These example input files demonstrate how to calculate the current in a *Resonant Tunneling Diode* (RTD). In RTD, quantum mechanical effects are essential. We use the same structure as outlined in Section 8.2 of *[GreckPhD2012]*.

We perform two simulations:

- *RTD_PhDthesis_PeterGreck_ballistic.negf* - calculation without scattering (ballistic calculation)
- *RTD_PhDthesis_PeterGreck.negf* - calculation including scattering

The RTD structure consists of 48 nm GaAs.

In the central region, there are two 4 nm wide AlGaAs barriers enclosing a 6 nm wide GaAs well. These two AlGaAs barriers are separated by a 4 nm intrinsic region.

Outside the barriers, the structure is doped with a concentration of 10^{17} cm^{-3}.

The barrier height was taken to be 0.156 meV.

The temperature is set to 150 K.

The grid spacing is 1.0 nm.

Further material parameters used are:

- effective mass: 0.067 (GaAs) and 0.0795 ($Al_{0.15}Ga_{0.85}As$)
- static dielectric constant: 10.89 (GaAs) and 10.4808 ($Al_{0.15}Ga_{0.85}As$)
- LO phonon energy: 0.035 eV (GaAs) and 0.0373 ($Al_{0.15}Ga_{0.85}As$)

The following figure shows the calculated current-voltage characteristics. It corresponds to Fig. 8.2 of *[Greck-PhD2012]*.

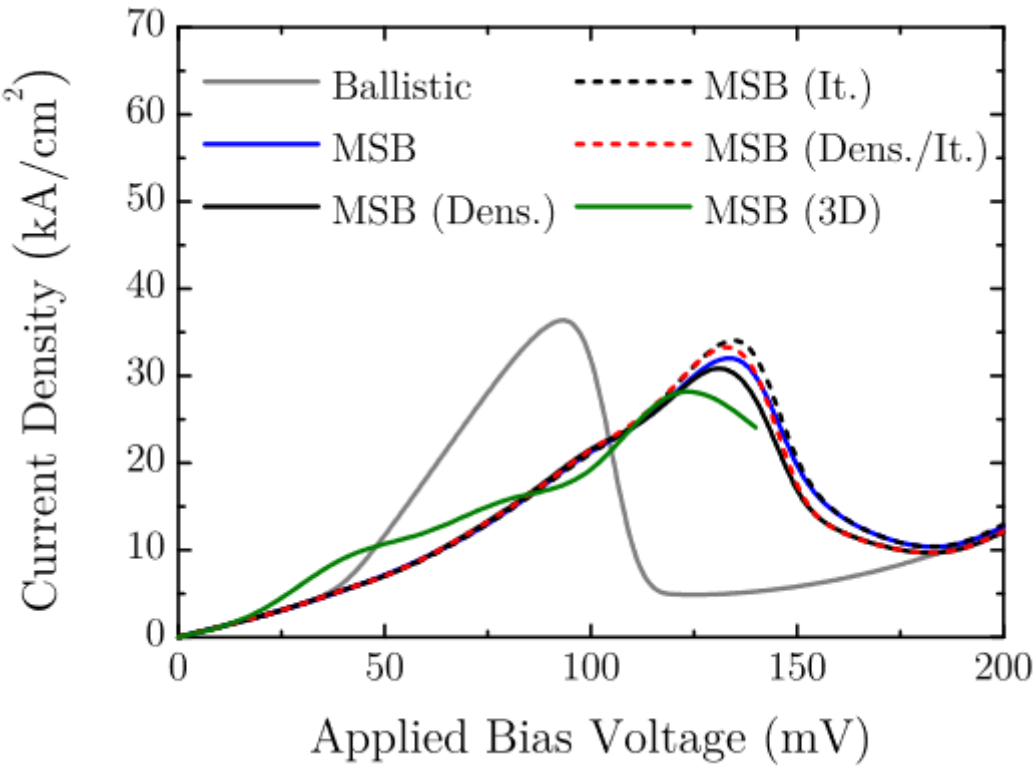


Figure 9.7.3.1: Fig. 8.2 of PhD thesis of Peter Greck

The relevant lines are *gray* (Ballistic) for the ballistic calculation and *blue* (MSB) for the calculation including scattering. The peak current density of the ballistic calculation is reached at about 100 mV where the chemical potential of the source contact is aligned with the lowest quantum well state. However, this result is physically wrong. This is due to the fact that a ballistic calculation neglects any kind of incoherent scattering. In a realistic device the carriers do not ballistically reach the RTD. Instead, a bound state is formed by the triangular shaped potential in front of the RTD and inelastic scattering processes lead to a capture of carriers within this state. Since the energy of this bound state is lower than the chemical potential of the source contact, the alignment with the RTD resonance is realized at higher bias voltages. For the calculated RTD, the peak currenty density is shifted from 100 mV to 140 mV.

Results

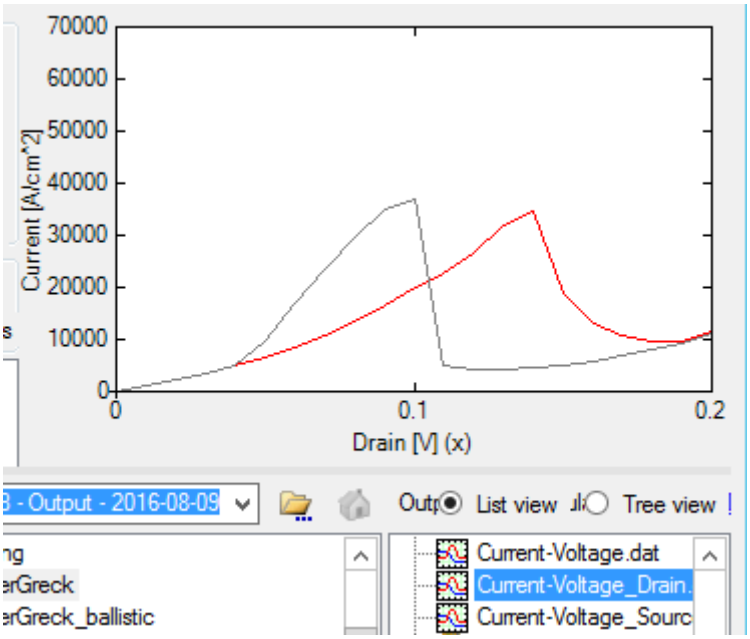


Figure 9.7.3.2: Calculated current-voltage characteristics of the ballistic calculation (gray) and the calculation including scattering (red). The peak current density is at 100 mV for the ballistic calculation but is shifted to 140 mV for the calculation that includes scattering. The file that is plotted is called `Current-Voltage_Drain.dat`.

The following figures shows the conduction band edges (`BandProfile/BandEdge_conduction.dat`) together with the

- local density of states $\rho(x, E)$ (`DOS/DOS_position_resolved.avs.fld`)
- electron density $n(x, E)$ (`CarrierDensity/CarrierDensity_energy_resolved.avs.fld`)
- current density $j(x, E)$ (`CurrentDensity/CurrentDensity_energy_resolved.avs.fld`)

at the peak current densities of 100 mV (left figures, ballistic calculation) and 140 mV (right figures, calculation including scattering). Note that the peak current density of the ballistic calculation (100 mV) deviates from the peak current density (140 mV) of the calculation including scattering.

Local density of states $\rho(x, E)$ plots:

Electron density $n(x, E)$ plots:

Current density $j(x, E)$ plots:

Further comments regarding the MSB input file

In order to calculate the current ballistically, we switched off the scattering by the following flag.

```
MSB_1Band{
   Core{
      BallisticCalculation = yes
   }
}
```

Please help us to improve our tutorial. If you have any questions or comments, create a ticket nextnano Help Center.

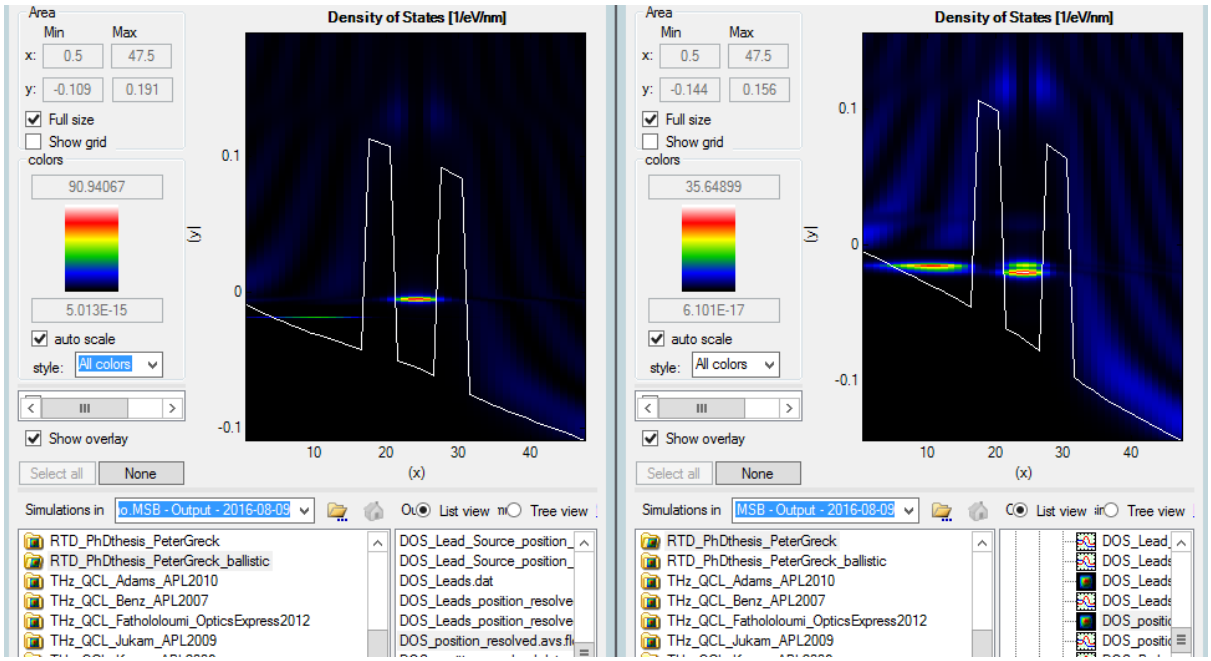


Figure 9.7.3.3: For the ballistic calculation (left figure), the quantum well state is aligned energetically with the chemical potential of the left contact (source). For the scattering calculation (right figure), the quantum well state is aligned energetically with the triangular quantum well state left of the barrier.

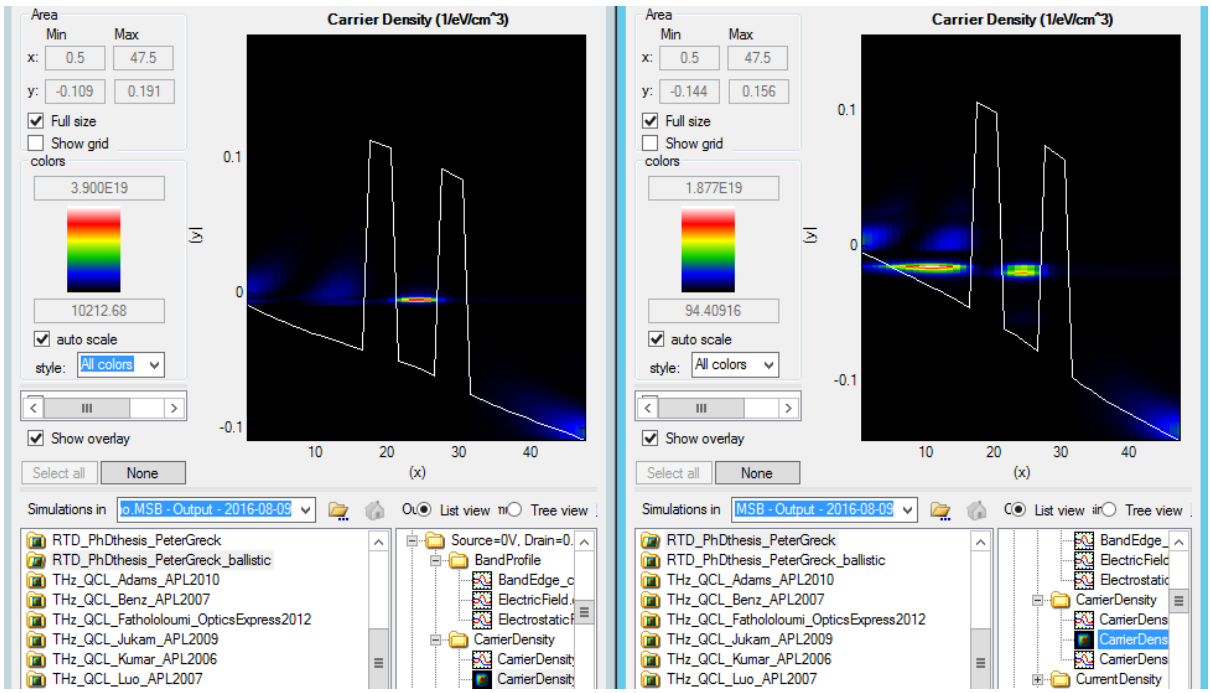


Figure 9.7.3.4: For the ballistic calculation (left figure), the quantum well state is aligned energetically with the chemical potential of the left contact (source). For the scattering calculation (right figure), the quantum well state is aligned energetically with the triangular quantum well state left of the barrier.

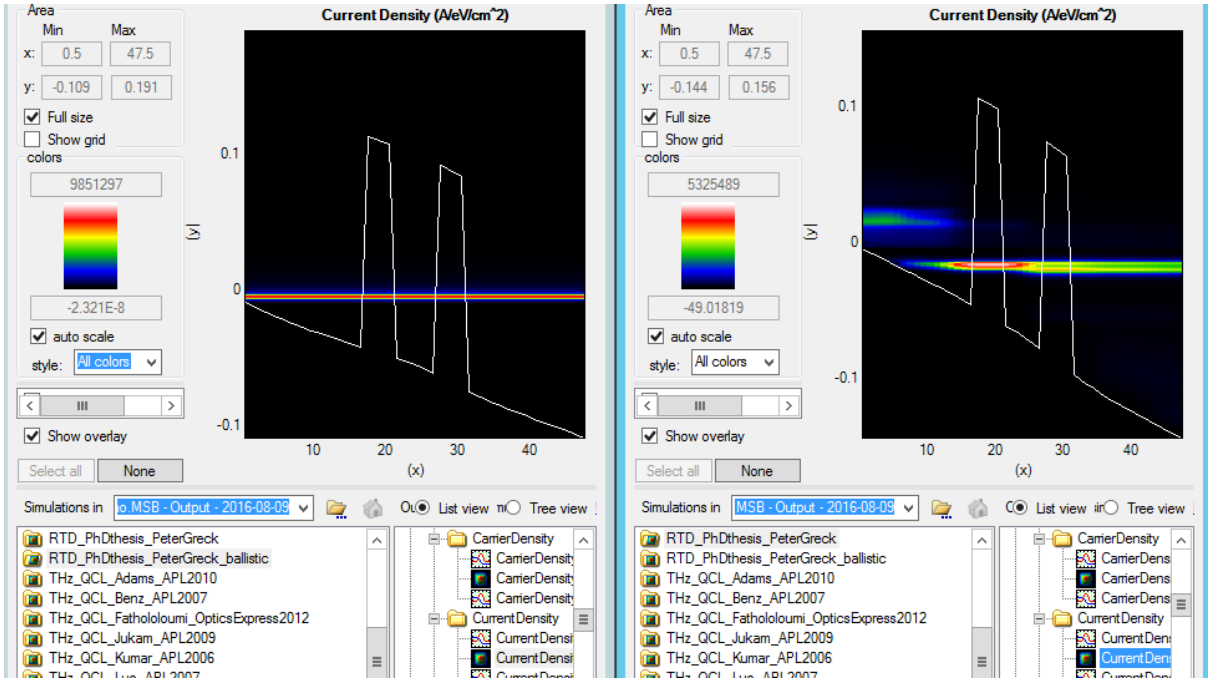


Figure 9.7.3.5: For the ballistic calculation (left figure), the quantum well state is aligned energetically with the chemical potential of the left contact (source). For the scattering calculation (right figure), the quantum well state is aligned energetically with the triangular quantum well state left of the barrier. On the right figure one can nicely see that the carriers that are injected from the left contact (source) scatter into the triangular quantum well state and then tunnel through the double barrier into the right contact (drain).

9.7.4 Quantum Well

This tutorial is based on the following *nextnano*[3] tutorial

- Applying the NEGF method to a quantum well

and on the following paper: *[KubisNEGF2005]*

The following input files were used:

- *QW_InGaAs_ballistic.negf* (input file for *nextnano.MSB*)
- *QW_InGaAs_scattering.negf* (input file for *nextnano.MSB*)
- *Materials_QW_InGaAs.negf* (material database for *nextnano.MSB*)
- *1DNEGF_InGaAs_QW_ballistic_CBR.in* (input file for *nextnano*[3], CBR method)
- *1DNEGF_InGaAs_QW_ballistic.in* (input file for *nextnano*[3], NEGF method)
- *1DNEGF_InGaAs_QW_scattering.in* (input file for *nextnano*[3], NEGF method)
- *1DNEGF_InGaAs_QW_scattering_bias.in* (input file for *nextnano*[3], NEGF method)

This example input file demonstrates how to calculate the current across a quantum well. It compares the results of a *ballistic* calculation to the results of a calculation including *scattering*. Here, input files for both *nextnano.MSB* and for *nextnano*[3] are provided so that the MSB algorithm can be benchmarked against the full NEGF algorithm as implemented by T. Kubis. We use the database file called *Materials_QW_InGaAs.negf* where we adjusted the default material parameters so that they match the publication of *[KubisNEGF2005]*.

Our structure consists of a 12 nm $In_{0.14}Ga_{0.86}As$ QW in the center surrounded on each side by GaAs barriers (of width 19 nm each).

GaAs | InGaAs | **GaAs**

where the barrier material is indicated in bold.

Further comments regarding the MSB input file

- We used 1000 energy grid points. In contrast, the NEGF simulation used much less.
- In the database we have adjusted the material parameters, e.g. for InGaAs we used the effective mass of GaAs, the static and optical dielectric constant of GaAs and the GaAs LO phonon energy for simplicity. We use a conduction band offset of 0.150 eV.

9.7.5 AlGaAs/GaAs THz QCL

Input file:

- *THz_QCL_Fathololoumi_OpticsExpress2012.xml* (old format)
- *THz_QCL_Fathololoumi_OpticsExpress2012.negf* (since 2022-04)

The tutorial is based on this paper: *[FathololoumiOE2012]*

This paper describes a THz quantum cascade laser (QCL) with a performance up to 199.5 K. The QCL layout is based on a resonant phonon three well design.

The input file in the XML format (old) uses constants (e.g. `Barrier` for the aluminum content of the barriers) and iterators for temperature sweep. In this case, single program execution simulates all the cases.

The input file in the .negf format (new) uses *nextnano++*-style variables starting with dollar sign (see *Input Syntax*). Then, the *nextnanomat* feature *Template* or nextnanopy Sweep can be used to sweep the variables before execution. This simply generates multiple input files, which can be run in parallel (see *Options: Simulation*).

Furthermore, in these input files the usage of so-called clusters is demonstrated. A cluster is a set of layers which is specified via `BeginCluster` and `EndCluster`, where the last layer is inclusive. The clusters are used to specify a certain voltage which is extended to the whole device and to define the region in which the optical gain is calculated. This is especially useful if more than one period is calculated.

The input file can be edited by the users to input the structure of a device and some further necessary input parameters. The details of the syntax of the input files is explained in the *input file documentation*.

Our structure consists of GaAs wells and $Al_{0.15}Ga_{0.85}As$ barriers of width

> *4.0 nm* | **4.3 nm** | 8.9 nm | **2.46 nm** | 8.15 | **4.1 nm** | 5.5 nm + *5.0 nm* + 5.5 nm = 16.0 nm | **4.3 nm** | *4.0 nm*

where the barrier material is indicated in **bold** and the doped region in *italics*. The doping concentration is $6 \cdot 10^{16}$ cm^{-3}.

The QCL structure consists of the actual QCL period which is embedded between two 4.0 nm wide GaAs regions that serve as the leads at the left and right boundary. These lead layers are also doped. The concentration is $1 \cdot 10^{16}$ cm^{-3}.

The QCL design consists of three quantum wells (GaAs) and three barriers (AlGaAs). On the right, just before the lead, we have to add an additional barrier arising from the next QCL period. The total length of the structure as specified above is 56.21 nm. As we are using a 0.5 nm grid, our simulated structure essentially has a length of 56.5 nm. The QCL is designed with respect to an electric field of -12.2 kV/cm. This corresponds to a bias voltage of 68.93 mV for our 56.5 nm structure.

Each layer of the structure is defined within the keyword <`Layer`> where the material name is given, its width and a doping concentration if applicable.

```
Layer{
   Thickness = 5.0                                    # Unit = nm
   Material{
       Base = GaAs
   }

   Probes = 1.0
```

(continues on next page)

(continued from previous page)

```
    Doping = 6e16                                   # Unit = 1/cm^3
}
```

An alloy like $Al_{0.15}Ga_{0.85}As$ is specifed as follows:

```
Material{
   Base = "Al(x)Ga(1-x)As"
   AlloyX = 0.15
}
```

The following figure shows the conduction band profile at -12.2 kV/cm.

Add figure here.

The gain has been calculated between 3 and 30 meV in the cluster called `Center`. The photon energy interval includes of 100 photon energies.

```
Gain{
    Cluster = Center                                    # Comment = Specfiy␣
↪the cluster where optical gain should be calculated.
    PhotonEnergyInitial = 1e-3              # Unit = eV             # Comment␣
↪= Min. photon energy for gain calculation.
    PhotonEnergyFinal = 30e-3               # Unit = eV             # Comment␣
↪= Max. photon energy for gain claculation.
    PhotonEnergySteps = 117                         # Unit = #              #␣
↪Comment = Number of photon energies for gain calculation.
}
```

The term `Center` refers to the `Center` entries in

```
BeginCluster = "Center, QCL"
```

and

```
EndCluster = "Center, QCL"
```

There is also a `Center` entry in the `Lead` called `Drain`. Here, the applied voltage drops over the layers that are inside the `Center` region.

a) Specify `Center`

```
 Lead{
    Name = Drain
    Voltage{
        Cluster = Center
    }
}
```

b) or `QCL`.

```
 Lead{
    Name = Drain
    Voltage{
        Cluster = QCL
    }
}
```

Both are possible because `BeginCluster` and `EndCluster` contain both labels.

In this tutorial, we have enabled, i.e. `1.0`, the scattering within the AlGaAs barriers. For each layer we have defined a variable with the name `BarrierProbes` which we set to `1.0`.

```
$BarrierProbes = 1.0                # Enable/disable scattering within␣
↪barriers. This can reduce computational time with minor influence on␣
↪overall results.
```

Consequently, for each `AlGaAs` barrier layer we have included the constant `BarrierProbes`.

```
Layer{
    ...
    Material{
        Base = "Al(x)Ga(1-x)As"
        AlloyX = $Barrier
    }
    Probes = $BarrierProbes
    ...
}
```

The simulation took

```
Duration in days, hours, minutes, seconds: 5h 50'34''
```

on an Intel Core i5-3470S (2.9 GHz) for a 0.5 nm grid spacing.

Please help us to improve our tutorial. If you have any questions or comments, create a ticket nextnano Help Center.

CHAPTER

TEN

NEXTNANOPY

The *nextnanopy* Python package is developed for *nextnano++*, *nextnano*^{3}, *nextnano.NEGF* and *nextnano.MSB* tools to automate nextnano GmbH simulations and analysis of the results. You can write a Python script to:

10.1 Overview

10.1.1 What is nextnanopy?

The *nextnanopy* Python package is developed for *nextnano++*, *nextnano*^{3}, *nextnano.NEGF* and *nextnano.MSB* to automate simulations and analysis of the results. You can write a Python script to:

- run multiple simulations (multidimensional parameter sweep, many input files, etc.)
- plot multiple figures, overlay data, save as image or PDF
- calculate further quantities using output data
- load polygons from a GDSII file for device geometry input

and anything else possible with the Python language. We maintain the *nextnanopy* Python package on GitHub as well as on the Python Package Index (PyPI) repository.

10.1.2 How do I install it?

Instructions are on our GitHub site.

Additional to the package installation, you need a valid nextnano GmbH license and a configuration file for *nextnanopy*, if you want to execute a nextnano GmbH product.

To easily use your settings and paths already stored in *nextnanomat*, you can export these settings into a *nextnanopy* config format. Documented on the following page: *Generate nextnanopy config file*.

10.1.3 Where to start?

You can start with the following tutorials: *Basic Tutorials*. We also provide sample scripts for *nextnano++*, *nextnano*^{3}, *nextnano.NEGF* and *nextnano.MSB*.

10.2 Tutorials

10.2.1 Basic Tutorials

The following links direct you to our tutorials on GitHub. There are Jupyter Notebook (.ipynb) and reStructuredText (.rst) versions. We recommend opening the Jupyter Notebook (.ipynb) files.

- Tutorial 0 - Set up the configuration
- Tutorial 1 - Execute an input file
- Tutorial 2 - Plotting data files
- Tutorial 3 - Load polygons from a GDSII file
- Tutorial 4 - Sweep to automate the execution
- Tutorial 5 - Navigation in output folder using DataFolder

In addition, we provide sample scripts to run and postprocess *nextnano++*, *nextnano*^{3}, *nextnano.NEGF* and *nextnano.MSB* simulations.

10.2.2 Data Visualization

10.3 Release Notes

Release notes for **nextnanopy** can be found on our GitHub site.

CHAPTER

ELEVEN

NEXTNANOEVO

Note: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

The *nextnanoevo* Python package is interfacing PyGMO and SciPy with *nextnano++*, *nextnano*^{3}, *nextnano.NEGF* and *nextnano.MSB* using *nextnanopy* to perform optimizations of designs modelled with the nextnano software.

11.1 Overview

Note: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

11.1.1 About

The *nextnanoevo* Python package is interfacing minimization and evolution algorithms with our other tools *nextnano++*, *nextnano*^{3}, and *nextnano.NEGF*. The package takes care of proper execution of simulations for optimization algorithms and definitions of problems to optimize based on standard nextnano GmbH outputs.

The user can choose between optimization algoritms from scipy.optimize and PyGMO:

- SciPy.optimize contains set of deterministic optimization algorithms to solve root and minimization problems. You can visit the documentation here.
- PyGMO is a Python library developed by F. Biscani and D. Izzo *[Biscani2020]*. It is an interface facilitating implementation of various evolutionary algorithms such as evolution strategies or genetic algorithms. It can be used to solve many kinds of single- and multi-objective problems.

We recommend using SciPy algorithms to solve simple single-objective problems, while PyGMO is more suited for multi-objective problems as well as for problems that have convergence issues with deterministic algorithms.

11.1.2 Purpose

For example, one want to study an LED with a goal to maximize quantum efficiency (QE). Let's assume that someone start with a design having the QE of 4 %. To improve it, one needs to provide the algorithm with information about which parts of the device can be modified (for instance certain lengths or mole fractions) and input the limits of these parameters which should depend on what is experimentally feasible. Then, the algorithm runs maximizing the quantum efficiency after which a new design yielding higher theoretical efficiency of 50 % is found.

11.1.3 Requirements

To run the *nextnanoevo* module, you need to:

- have a valid license for *nextnanoevo*
- install PyGMO (see GitHub)
- install nextnanoevo
- prepare configuration file for *nextnanopy* (see GitHub)

11.1.4 Installation

We recommend installing *nextnanoevo* with conda package manager. Installation without conda using different python distributions is possible if requirements above are met (but not tested).

Installation with conda

1. Install Anaconda or Miniconda

   Miniconda is a lightweight version that install only the minimum necessary liabraries).

2. Create and new conda environment with python 3.10

```
$ conda create -n nnevoEnv python=3.10
```

3. Activate newly created conda environment

```
$ conda activate nnevoEnv
```

4. Install pygmo (this step might take few minutes)

```
$ conda config --add channels conda-forge
$ conda config --set channel_priority strict
$ conda install pygmo
```

5. Install nextnanoevo into the activated environment with installation script

```
$ path/to/nextnanoevo/install.bat
```

11.1.5 License activation

The license is activated in the same way as other products, for more details visit *License activation page*

11.1.6 Config setup

In order to use nextnanoevo, at least 2 products should be set up in nextnanopy config.

- nextnanoevo itself (license option only)
- nextnano product you want to use it with. For example, for nextnano++ you need to set up license, exe, database and output directory

To set up nextnanoevo license, run the following script.

```
import nextnanopy as nn
nn.config.set('nextnanoevo', 'license', 'path\to\your\License_nnevo.lic')
nn.config.save()
```

Tutorial on how to set up other nextnano product can be found here - Tutorial 0 - Set up the configuration.

11.2 Definitions

Note: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

The optimization procedure is defined by vector of input variables, *metric* to be optimized (also called *fitness* for some algorithm), and *function* that converts input variabels to metric.

To be able to define these parameters of optimization procedure in nextnanoevo, 2 key instances should be created: *NextnanoIO* and *NextnanoMetricExtractor*.

11.2.1 nextnanoevo.IO

IO stands for nextnano input-output. This class defines:

- *input_file_path*: input file to run
- *input_variable_names*: variables to variate in the input file
- *target_output*: output datafiles that will be used

To create IO instance, use the following syntax

```
from nextnanoevo.IO import IO

input_file_path = r'\path\to\input_file.in'

varibles = ['variable_name1', 'variable_name2']
# example: variables = ['WellWidth', 'BarrierWidth']

output_files = [('relative', 'path', 'datafile1'),
                ('relative', 'path', 'datafile2')]

# the path of output files is relative to output directory
# example: output_files = [('Strain', 'strain_simulation.dat'),
```

(continues on next page)

(continued from previous page)

```
#                           ('bias_00000', 'Quantum', 'energy_spectrum_
↪quantum_region_Gamma_00000.dat')]

nn_io = IO(input_file_path, variables, output_files)
```

11.2.2 Metric

This class defines:

- *input_length*: number of datafiles that will be processed (should match with NextnanoIO)
- *output_length*: dimensionality of the metric vector
- *extraction_function*: how to extract the final metric vector from the datafiles (the function output should match with *output_length*)

Extraction function is defined by user. The extraction function will take as input list of nextnanopy datafiles. Visit this tutorial to get to know about nextnanopy datafiles Tutorial 2 - Plotting data files.

To create *Metric* instance, use the following syntax

```
from nextnanoevo.MetricExtractor import Metric
import numpy as np


def dummy_extraction_function(dfiles):
    """
    This is an example of simple extraction function that takes first value␣
↪of first variable from 2 datafiles.
    Reutrn the vector contatining sum and product of these values.
    """
    dfile1 = dfiles[0]
    dfile2 = dfiles[1]

    val1 = dfile1.variables[0].value[0]
    val2 = dfile1.variables[0].value[0]

    return np.array([val1+val2, val1*val2])


nn_metric = Metric(input_length=2, output_length=2, extraction_
↪function=dummy_extraction_function)

# input_length = 2, because the metric processes 2 datafiles
# output_length = 2, because it returns vector of length 2.
```

In some cases, the metric function can be dependent on the input variables. The workaround to achieve this is to use the output file that contains the input paramters (*variables_input.txt* in case of *nextnano++*). Lets have a look at the dummy example with metric dependent on one value from output file and one input variable.

```
from nextnanoevo.IO import IO
from nextnanoevo.MetricExtractor import Metric

# first, we define the IO instance with one normal output file and one␣
↪output file with input variables
input_file_path = r'\path\to\input_file.in'

varibles = ['variable_name1']
```

(continues on next page)

(continued from previous page)

```
output_files = [('relative', 'path', 'datafile'),
                ('variables_input.txt', )]

nn_io = IO(input_file_path, variables, output_files)

def dummy_extraction_function_with_input(dfiles):
    """
    Example of extraction function that takes first value of datafile and␣
↪input varibale and returns the ratio
    """
    dfile = dfiles[0]
    input_variables_dfile = dfiles[1]

    value = dfile.variables[0].value[0]
    input_value = input_variables_dfile.variables[0].value



    return np.array([value/input_value])

nn_metric = Metric(input_length=2, output_length=1, extraction_
↪function=dummy_extraction_function_with_input)

# input_length = 2, because the metric processes 2 datafiles
# output_length = 1, because it returns vector of length 1.
```

11.3 Tutorials

Note: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

11.3.1 PyGMO

Maximizing Envelope Overlaps for a Given Transition Energy

Attention: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

Header

Relevant files

- *T2SL_InAs-GaInSb_Grein_JAP_1995_1D_nnp.in* (the same as *here*)
- *InGaSb_InAs_script.py* (not available yet)

Scope

- design optimization
- envelope overlaps
- transition energies
- type-II superlattice

Important variables

- $L_InGaSb = thickness of the layer of In(x)Ga(1-x)Sb (nm)
- $L_InAs = thickness of the layer of InAs (nm)
- $X_InGaSb = indium content of the layer of In(x)Ga(1-x)Sb
- E_0 = fundamental transition energy
- S = overlap between the c1 and v1 wave function envelopes
- d = fitness (distance from the ideal solution)

Introduction

This tutorial briefly introduces concept of evolution strategies for semiconductor device optimization and discusses its application on an example of type-II superlattice from a tutorial *1D - InAs/In0.4Ga0.6Sb superlattice dispersion with 8-band k.p (type-II band alignment)* using *nextnanoevo package*.

Evolutionary algorithms

Evolutionary algorithms are algorithms inspired by biological processes such as selection, mutation and reproduction that are used to solve optimization problems. In this tutorial, we specifically use the so-called NSGA-II genetic algorithm (see more details below) to improve the design of a type-II superlattice semiconductor device. The objective is to obtain a fundamental transition energy of 0.15 eV and to maximize the envelope overlaps between of the first conduction band c1 and the first valence band v1 states. The algorithm begins with a template design and a set of initial parameters to be modified as well as ranges within which the parameters can be adjusted. The ranges of these parameters form the search space for the algorithm and should highly depend on experimental feasibility.

The template 'default' design constitutes the initial parent population. From this first generation, a few modifications (mutations) of the initial design are made to generate a new set of designs which forms the children population. Then, *nextnano++* is used to evaluate the fitness of each individual of this new population. The fitness function is defined based on the optimization objectives chosen by the user. The best individuals are selected according the fitness function and become the next parent population to then have children of their own. Again, the best designs are selected to repeat the process of mutation, reproduction, and selection over and over again until the optimal design is reached. The algorithm stops when the objectives have been reached or, in the case of more loosely-defined objectives (i.e. maximize the overlap), the user sets the number of iterations (or generations) before starting the algorithm.

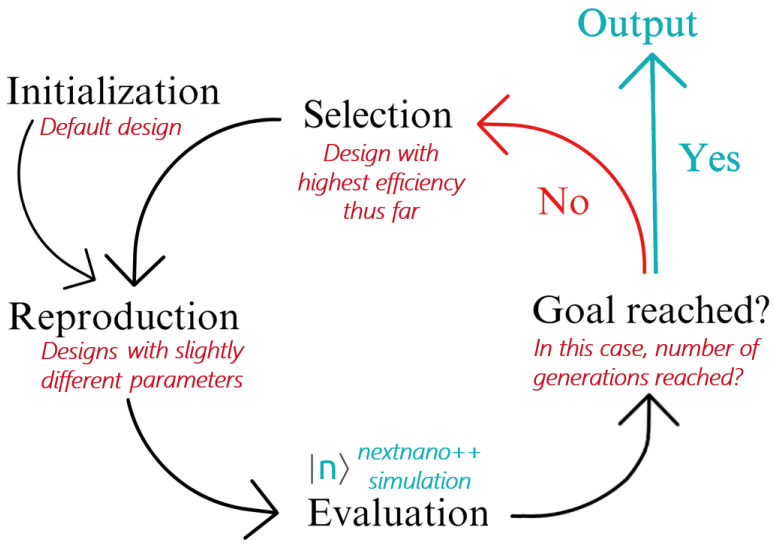


Figure 11.3.1.1: Diagram of the evolutionary algorithm combined with *nextnano++*.

Algorithm NSGA-II

There are several evolutionary algorithms that can be used to solve multi-objective problems but in this tutorial, NSGA-II (non-dominated sorting genetic algorithm) as developed by PyGMO was employed (for more information, visit the PyGMO documentation page). NSGA-II is a genetic algorithm that employs techniques such as non-dominated sorting and crowding distance to solve problems in a way that promotes elitism and diversity.

Non-dominated sorting is a technique used in multi-objective problems to identify the solutions that cannot be improved in any objective without worsening another objective, which can also be referred to as the Pareto-optimal solutions.

Crowding distance is a measure of how close a solution is to its neighbors. Solutions with a large distance to their neighbors are more likely to be selected for the next step in the algorithm than solutions with a small crowding distance to ensure the well-representation of the Pareto front in the next generation of solutions.

There are different parameters for the evolution process which can be tuned by the user, the most important one being the number of generations. The bigger it is, the longer the simulation will take but the better the results will possibly be. However, it is possible to reach the best result just after a couple of generation. For example, in this tutorial, the number of generations is set to 100, but the best design is obtained at the 50th generation. Or for example, one could also tune the crossover and mutation probabilities, that is the likelihood of two individuals combining their "genes" and the likelihood of a random change in an individual, respectively.

Parameters for optimization algorithm:

Variable	Value	Description
alg	'nsga2'	*name of the algorithm*
gen	[100]	*number of generations* - int or list of int
cr	0.95	*crossover probability* - float, [0,1[
eta_c	10	*distribution index for sbx crossover* - float, [0,100[
m	0.01	*mutation probability* - float, [0,1[
eta_m	50	*distribution index for polynomial mutation* - float, [0,100[
seed	876624	*seed for randomness* - int

Note: Note: the evolution process is random, so each run may give a different result. Our suggestion is to change

the seed (input a random number) for every run to promote randomness.

How to run the optimization

Requirements:

- Install the *nextnanoevo* module (see instructions *here*)
- download Python scripts *nextnanoevo.py* and *InGaSb_InAs_script.py*
- download *nextnano++* input file *T2SL_InAs-GaInSb_Grein_JAP_1995_1D_nnp.in*

Steps to follow:

1) Create a main folder containing the two Python scripts *nextnanoevo.py* and *InGaSb_InAs_script.py* and the nextnano input file *InAs_InGaSb.in* as well as an "output" folder.
2) Open *InGaSb_InAs_script.py* and copy the path to the *InAs_InGaSb.in* input file as well as the path to the output folder.
3) OPTIONAL: tune the parameters of the optimization such as the lower and upper boundaries of the parameters to optimize, the parameters of the default design or the number of generations.
4) Execute the script to run the optimization.
5) At the end of the run, the parameters and properties of the optimized design will be printed. This information will also be saved in a data file along with all the designs that have been simulated throughout the run. The data file can be found in the output folder, under "results". The simulation results of both the default and optimized designs are also saved in the output folder and are named in the following way: *[filename]_[L_InGaSb]_[L_InAs]_[X_InGaSb]*. For example, the results of the default design will be the following folder: *T2SL_InAs-GaInSb_Grein_JAP_1995_1D_nnp_1.5_3.98_0.4*.

Optimizing type-II superlattice

Defining optimization problem

Let's define optimal structure as the one characterized by a fundamental transition energy of 0.15 eV with the envelope overlaps between of the first conduction band c1 and the first valence band v1 states equal 1.

To run an evolution with *nextnanoevo*, a default design is provided by the user as well as parameters to optimize along with a search space, that is to say both lower and upper limits for each of these parameters.

For this device, the parameters to optimize were the thicknesses of the two layers of InGaSb and InAs as well as the indium content of the InGaSb layer. Our default design had the following parameters and properties:

Variable	Value	Description
$L_InGaSb	1.5 nm	Thickness of the layer of In(x)Ga(1-x)Sb
$L_InAs	3.98 nm	Thickness of the layer of InAs
$X_InGaSb	0.4	Indium content of the layer of In(x)Ga(1-x)Sb
E_0	0.10 eV	Fundamental transition energy
S	0.497	Overlap between the c1 and v1 wave function envelopes
d	0.677	Fitness of the solution / distance from the ideal solution

And the defined search space was:

Variable	Value
$L_InGaSb	[0.1, 10] nm
$L_InAs	[0.1, 10] nm
$X_InGaSb	[0.01, 1]

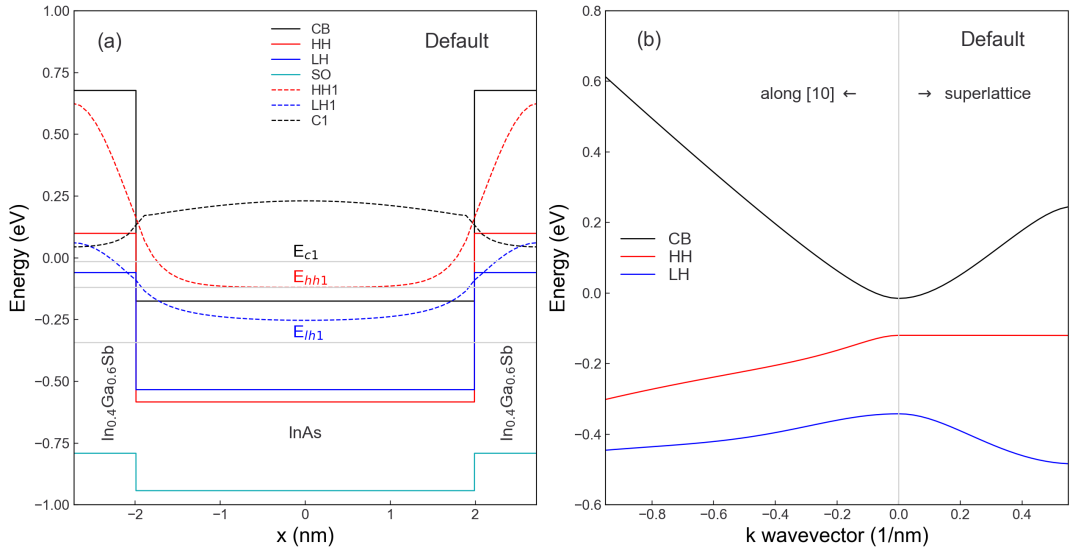


Figure 11.3.1.2: Band edges and energy levels (a) and energy dispersion of conduction, heavy hole and light hole bands along two directions in (k_x, k_y) space (b) of the default design.

To assess the performance of our candidate designs, we calculated the fundamental transition energy:

$$E_0 = |E_{c1} - E_{v1}|$$

and the overlap between the c1 and v1 wave function envelopes:

$$S \propto \sum_{\beta \in \{X,Y,Z\}} \int_L F^*_{v1,\beta} F_{v1,S} + \sum_{\alpha \in \{X,Y,Z\}} \int_L F_{v1,\alpha} F^*_{c1,S}$$

Then, the fitness of a candidate is calculated by its distance from the ideal solution, that is the solution with the desired energy and overlap (in our case 0.15 eV and 1, respectively):

$$d = \sqrt{(E_{obj.} - E_0)^2 + (S_{obj.} - S)^2}$$

During the optimization process, this distance should decrease as we reach our optimized design which is the closest to the ideal solution.

Optimized design

At the end of our optimization run, the best design we obtained had the following parameters and properties:

Variable	Value	Description
$L_InGaSb	1.47 nm	Thickness of the layer of In(x)Ga(1-x)Sb
$L_InAs	1.98 nm	Thickness of the layer of InAs
$X_InGaSb	0.75	Indium content of the layer of In(x)Ga(1-x)Sb
E_0	0.152 eV	Fundamental transition energy
S	0.723	Overlap between the c1 and v1 wave function envelopes
d	0.277	Fitness of the solution / distance from the ideal solution

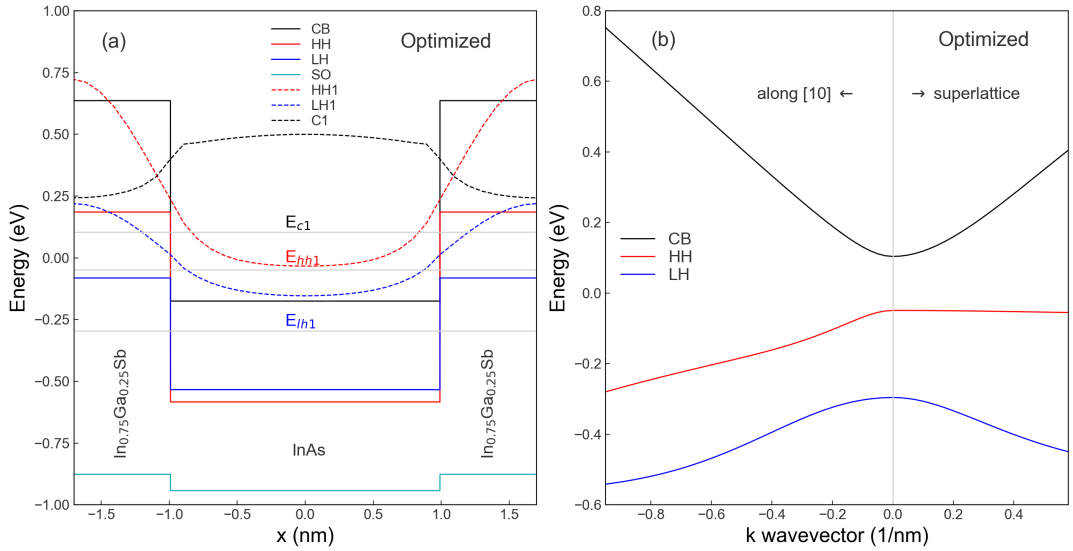


Figure 11.3.1.3: Band edges and energy levels (a) and energy dispersion of conduction, heavy hole and light hole bands along two directions in (k_x, k_y) space (b) of the optimized design.

Conclusion

We successfully obtained a design with the desired fundamental transition energy of 0.15 eV showing promise for applications in long-wave infrared detectors. The overlap of the wave function envelopes was also increased thus improving the overall performance of the device. The overall fitness of the design went from 0.677 to 0.277 by changing the lengths of the two layers by a few nanometers as well as the indium content in the InGaSb layer. This solution was the one that yielded the best trade-off between the objective transition energy and the highest overlap possible. However, other solutions in the Pareto front showed higher overlap values but slightly different transition energies than the objective or transition energies closer to the objective but lower overlap values.

Optimization of Mid-IR Quantum Cascade Laser Gain

Note: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

Header

Relevant files

- *MidIR_QCL_InGaAs_InAlAs_Bai_NaturePhot2010_nextnanoevo.negf*
- *MIdIR_QCL_example.py* (not available public yet)

Relevant output files:

- *330mV/Gain/SemiClassical_vs_Energy_[0.0,0.0,1.0].dat*

Scope

- design optimization
- QCL
- NEGF
- laser gain

Important variables

- `$thickness_scaling_factor` (scaling factor for all wells and barries in QCL heterostructure)

Intoroduction

This tutorial shows how to optimize the gain of a 1D Mid-IR Quantum Cascade Laser (QCL) using the The *nextnanoevo* Python package. The goal is to maximize the gain of the QCL by varying the thickness of the quantum wells and barriers. The gain is calculated using the *nextnano.NEGF*.

Starting point

The starting point is the design of a Mid-IR QCL with a peak gain at photon energy of 275 meV. The design consists of sequence of InGaAs wells and AlInAs barriers with applied 330 mV bias (*[Bai2010]*.)

The bandedges are shown in the figure below

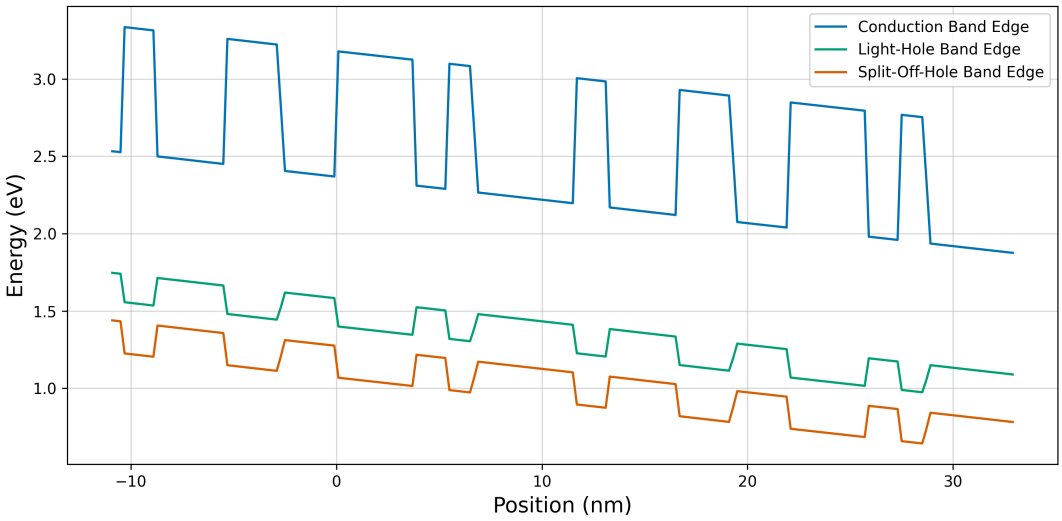


Figure 11.3.1.4: The band edges of the MidIR QCL following *[Bai2010]* design.

The simulated gain with the peak at 275 meV is shown in the figure below

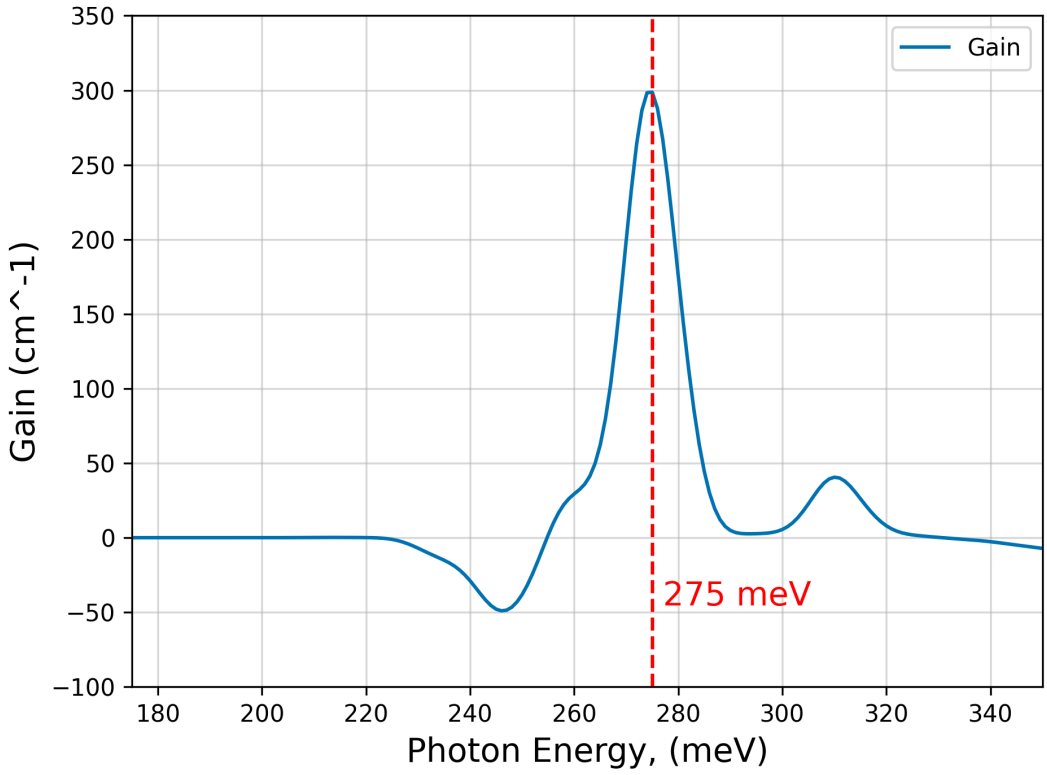


Figure 11.3.1.5: The gain of the MidIR QCL following *[Bai2010]* design.

Optimization objective

The goal is to maximize the gain of the QCL at 270 meV by varying the thickness of the quantum wells and barriers. All the wells and barriers will be scaled by the same factor.

Variables under optimization:

Name	Units	Initial value	Bounds
`$thickness_scaling_factor`	–	10.0	0.9-1.1

Target output:

Name	Units	Initial value	Target
Gain at 270 meV	cm^{-1}	199.49	max

Optimization script

At first, specify input file, variables, and relevant output files in the `IO` instance:

```
from nextnanoevo.IO import IO
output_files = [("330mV", "Gain", "SemiClassical_vs_Energy_[0.0,0.0,1.0].dat")]

nn_io = IO(
    input_file_path,
    variable_names=["thickness_scaling_factor"],
    target_output_paths=output_files,
)
```

Next, define the metric to get the gain at 270 meV.

```
def get_gain_at_270mev(df_list):
  df = df_list[0]
  energy = df.coords["Photon Energy"].value
  gain = df.variables["Gain"].value
  # get gain exactly at 270 meV
  mask = np.isclose(energy, 270, atol=0.001)
  gain_at_270mev = gain[mask]
  # minus sign to transform maximization to minimization
  return -gain_at_270mev
```

Define the Metric Metric class

```
from nextnanoevo import Metric
metric = Metric(input_length=1, output_length=1, extraction_function=get_gain_at_
↪270mev)
```

Create and run evolution

```
from nextnanoevo import Evolution
evolution = Evolution(nextnanoio=nn_io, metric=metric, bounds=([1.0], [1.1]), size=5)

evolution.run(gen=4, algorithm_kwargs={"m": 0.4, "cr": 0.95, "seed": 2023},␣
↪seed=12345)
```

And print the final solution and fitness

```
for index, (sol, fit) in enumerate(
    zip(evolution.pareto_solutions, evolution.pareto_fits)
):
    print(f"{index}. Solution {sol}: {fit}")
```

This example is created to showcase how to use nextnanoevo to optimize the devices with *nextnano.NEGF*. To decrease the time of the simulations, the bounds are set to 0.9-1.1, and the number of generations is set to 3.

Result of optimization:

$thickness_scaling_factor	Gain at 270 meV (cm^{-1})
1.0662883	324.4952

The resulting gain in the optimized QCL is shown in the figure below

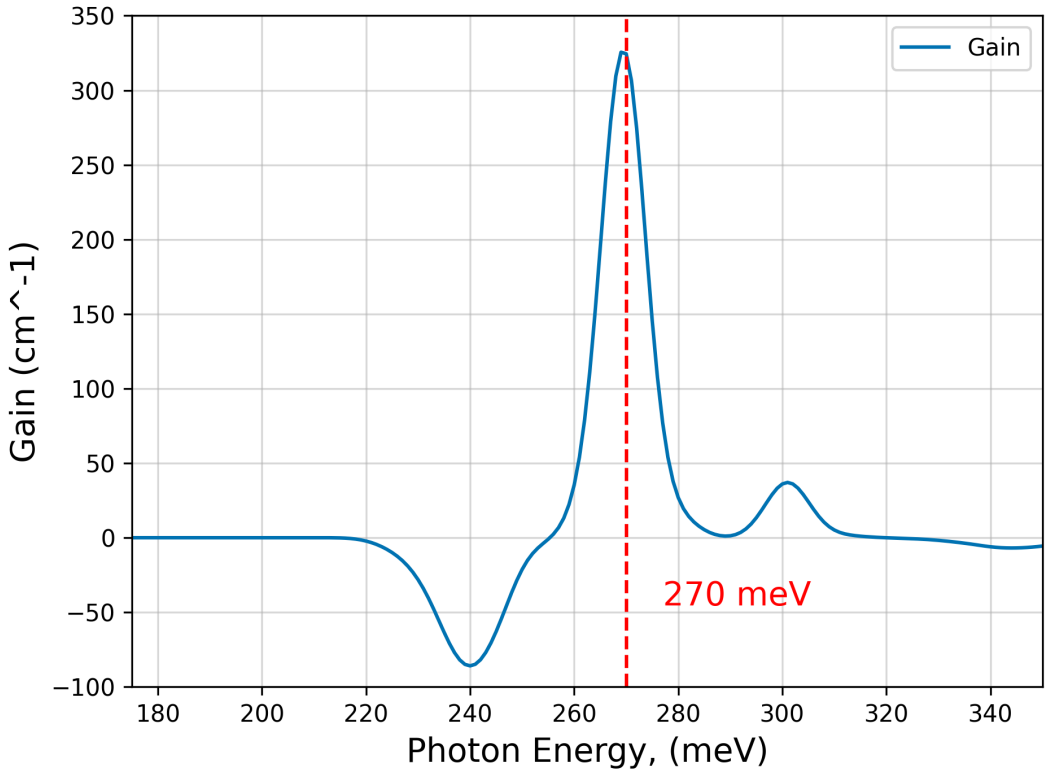


Figure 11.3.1.6: The gain of the MidIR QCL using optimized design.

Conclusion

This tutorial shows how to use the *nextnanoevo* to optimize the gain at the given photon energy of QCL. The optimized sttructure has a peak gain at the target wavelength after 5 generations of optimization.

11.3.2 SciPy

Optimizing well width for specific target tranisition energy in an infinite quantum well

Attention: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

Header

Relevant files

- *1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_sg_nnp.in* (the same as *here*.)
- *InfiniteWell_example.py* (not public yet, provided by request)

Relevant output files:

- *bias_00000/Quantum/energy_spectrum_quantum_region_Gamma_00000.dat*

Objective

The goal of this optimization is to determine the quantum well thickness that aligns the transition between the first and second electron states with a specified target energy.

Variables under optimization:

Name	Units	Initial value	Bounds
QuantumWellWidth	nm	10.0	–

Target output:

Name	Units	Initial value	Target
Transition energy	eV	0.1668	0.1

Scipy.optimize.root

In this example scipy.optimize.root root-solver is used. The algorithm searches for the root of the equation

$$f(x) - target = 0$$

where x represents the vector of input variables and f is the user-defined metric.

Optimization script

Initially, create a nextnanoevo.IO instance specifying the input file, optimization variables, and output files relevant for the metric.

```
from nextnanoevo.IO import IO
nn_io = NextnanoIO(input_file_path,
                   variable_names=['QuantumWellWidth'],
                   target_output_paths=[('bias_00000', 'Quantum', 'energy_spectrum_
↪quantum_region_Gamma_00000.dat')])
```

Next, define the metric function to extract the transition energy from the energy_spectrum datafile.

```
def first_transition_extraction_function(df_list):
    df_energy_spectrum = df_list[0]  # df_list will be a list containing one datafile
    energy_spectrum = df_energy_spectrum.variables['Energy'].value
    transition_energy = energy_spectrum[1] - energy_spectrum[0]  # transition energy␣
↪between the ground state and second state
    return np.array([transition_energy])
```

A nextnanoevo.Metric is created with the custom metric function. Extraction function takes single output file and converts it to single value, so both input length and output length equal 1.

```
from nextnanoevo.MetricExtractor import Metric
metric = Metric(input_length=1, output_length=1, extraction_function=first_transition_
↪extraction_function)
```

Create the Optimizer, and set the initial value for quantum well thickness and target transition energy.

```
from nextnanoevo.OptimizerClass import Optimizer
optimizer = Optimizer(nextnanoio=nn_io, metric=metric, optimization_method='root')
# arguments of the optimization_method. For scipy optimize.root x0 is mandatory
```

(continues on next page)

(continued from previous page)

```
optimizer.set_optimization_parameters(x0=np.array([10]))
# set target
optimizer.set_target(np.array([0.1]))
```

Run the optimization. The details are recorded in the Simulation*.log file, which tracks each input file execution.

```
result = optimizer.run_optimization()
print(f"The optimal quantum well thickness is {result.x} nm")
```

Output:

```
The optimal quantum well thickness is [12.80762256] nm
```

Conclusion

In summary, this tutorial has demonstrated an approach to optimize a quantum well structure to match a specific transition energy using nextnanoevo. The process involved setting up the NextnanoIO instance, defining a custom metric function, utilizing the NextnanoMetricExtractor, and finally employing the NextnanoOptimizer for the actual optimization task.

Optimization of a 2DEG structure

Attention: The *nextnanoevo* Python package is under development. Its release is planned for 2024.

Header

Relevant files

- *Greg_Snider_MANUAL_1D_optimization_nnp.in* (the input file is almost the same as *here*, except of few adjustments to enable optimization)
- *2DEG_example.py* (not available public yet)

Relevant output files:

- *bias_00000/density_electron.dat*

Scope

- design optimization
- 2DEG
- electron density

Important variables

- `$quantum_well_width` (thickness of the GaAs quantum well)

Introduction

This tutorial demonstrates the use of *nextnanoevo* for optimization of a 2DEG design from the tutorial *— FREE — Schrödinger-Poisson - A comparison to the tutorial file of Greg Snider's code*.

Objective

The first state (subband) confined in the GaAs quantum well is forming 2-dimensional electron gas (2DEG) formed at one of interfaces due to electric field originated from the Schottky layer. Multiple parameters of the structure are impacting formation of the 2DEG, and one of them is the width of the quantum well containing the 2DEG. Therefore, one can aim at choosing the best width of the quantum well for a given design, to obtain 2DEG with the highest possible occupation of the first state (subband). In this tutorial we show, how to perform such optimization. We show how to maximize occupation of the first electron state by varying the thickness of the GaAs quantum-well layer in the range from 1 nm to 20 nm.

Variables under optimization:

Name	Units	Initial value	Bounds
`$quantum_well_width`	nm	15.0	1.0-20.0

Target output:

Name	Units	Initial value	Target
Occupation of the fisrt subband	cm^{-2}	0.1668	max

Scipy.optimize.minimize

In this example *scipy.optimize.minimize* algorithm is used. The algorithm searches for the minimum value in the provided variable space. In this tutorial only 1 variable is optimized, but the algorithm supports n-dimensional variable space.

> **Hint:** Every maximization problem can be formulated as minimization by changing the sign of the target value.

Optimization script

At first, specify input file, variables, and relevant output files in the `IO` instance:

```
from nextnanoevo.IO import IO
nn_io = IO(input_file_path,
       ['quantum_well_width'],
       [('bias_00000', 'Quantum', 'occupation_quantum_region_Gamma.dat')])
```

Next, define the metric function to extract the occupation of the first subband in the 2DEG region.

```
def max_first_state_occupation(df_list):
    gamma_first_state_occupation = df_list[0].variables[0].value[0]
```

(continues on next page)

(continued from previous page)

```
    # negative sign to use "minimize" method
    return -gamma_first_state_occupation
```

```
metric = Metric(input_length=1, output_length=1, extraction_function=max_first_state_
↪occupation)
```

Create the Optimizer, and set the initial value and bounds for the quantum well thickness.

```
from nextnanoevo.OptimizerClass import Optimizer
optimizer = Optimizer(nextnanoio=nn_io, metric=metric, optimization_method='minimize')

# Setting the scipy.optimize.minimize arguments
optimizer.set_optimization_parameters(x0=15, bounds=[(1, 20), ], method='Nelder-Mead')
```

Run the optimization. The details are recorded in the Simulation*.log file, which tracks each input file execution.

```
result = optimizer.run_optimization()
print(f"The optimal quantum well thickness is {result.x} nm")
```

Output:

```
The optimal quantum well thickness is [8.34375] nm
```

CHAPTER

TWELVE

HTCONDOR

You can use HTCondor to run the nextnano software on your local computer infrastructure ("on-premise"). Essentially, *nextnanomat* submits the job either locally or on the "HTCondor" cluster. In both cases, the results of the calculations are located on your local computer.

This feature is only supported with our new license system.

12.1 HTCondor on nextnanomat

The following shows a screenshot from *nextnanomat*. 6 computers are connected to the HTCondor pool called `e25nn`. 120 slots are configured, 44 are currently available. Computers 2, 3, 4 and 6 are selected to accept jobs. Computers 2 and 6 are currently not available as they are in use.

12.2 Recommended Installation Process

Download HTCondor installer from HTCondor.

1. On the webpage, click on `Download` and go to `Current Stable Release` of `UW Madison` (as of September 24 2020, HTCondor 8.8.10).
2. We recommend the file for Windows in `Native Packages`. The filenames look similar to this one:
   - `condor-8.8.10-513586-Windows-x64.msi` (Version 8.8.10)
3. Select the file, agree to the license agreement and download the `.msi` file. When you download it, you can optionally enter your name, email address and institution and subscribe to the HTCondor newsletter.

Install HTCondor.

1. Start installer
2. Click `Next` and then accept License Agreement
3. Then there are two options. There will be one special computer that manages all HTCondor jobs (Central Manager), and other computers as submit/execute nodes. If there is no Central Manager yet, we have to create a new pool.
   1. If you **are** on the Central Manager, choose `Create a new HTCondor Pool` and fill in the name of the Pool, e.g. `nextnanoHTCondorPool`. This is a unique name for your pool of machines.
   2. If you **are not** the Central Manager, choose `Join an existing HTCondor Pool` and fill in the hostname of the central manager, e.g. computername where `nextnanoHTCondorPool` has been created.
4. Tic `Submit jobs to HTCondorPool` and choose `Always run jobs and never suspend them.` (Alternative: If you do not want other people to run jobs on your machine at all, select `Do not run jobs on this machine` or if you do not want other people to run jobs on your machine while you are working, select `When keyboard has been idle for 15 minutes.`. You can of course modify these settings later.)

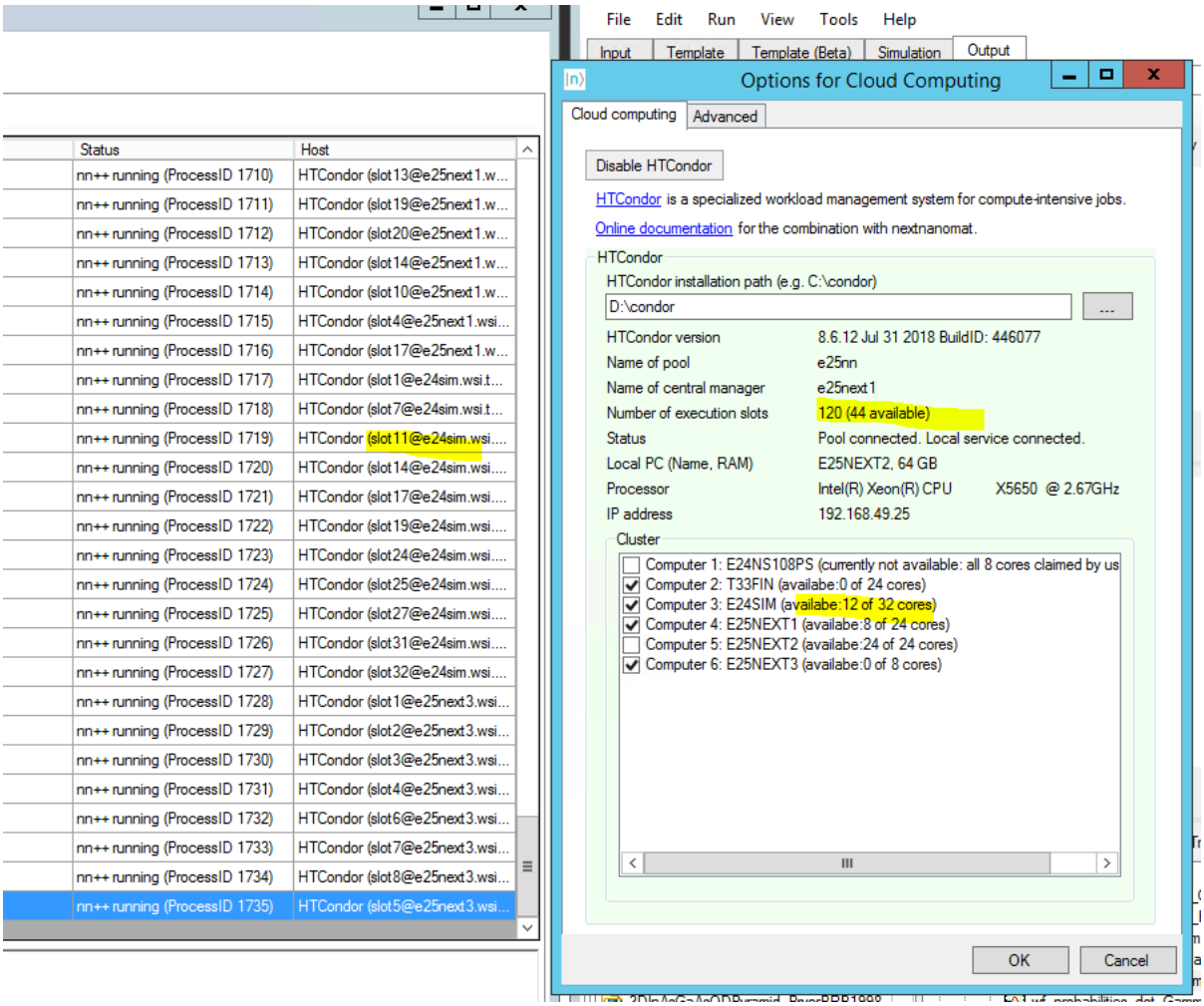


Figure 12.1.1: Screenshot taken from *nextnanomat* with integrated HTCondor feature.

5. Fill in your domain name (Example: Your Windows domain, e.g. `yourcompanyname.com` (without `www`).) All PCs of your network should get the same domain name, this does not necessarily have to be your Windows domain.
6. Hostname of SMTP Server and email address of administrator (not needed currently, leave it blank)
7. Path to Java Virtual Machine (not needed currently, leave it blank)
8. Host with Read access: *
9. Host with Write access: `$(CONDOR_HOST), $(IP_ADDRESS), *.yourdomainname.com, 192.168.178.*`, (**Replace** *.cs.wisc.edu with your domain name and **add** your local IP subnet e.g. 192.168.178.*). On Windows you can find your IP subnet by opening the Command Prompt `cmd.exe` and typing in `ipconfig`.
10. Host with Administrator access * (or `$(IP_ADDRESS)`)
11. Enable VM Universe `No`
12. Choose an installation directory and press next (e.g. `C:\condor\`). The directory `Program Files` is problematic due to write permissions, so we do not recommend using it.
13. Press `Install` and type in the Administrator password of your PC. (You need Administrator rights.)
14. Once installed, please restart the computer. Then your new pool or pool member should be up and running.

A few more setups

1. To be able to submit jobs from *nextnanomat* to HTCondor, you have to store your credentials once. Open a command shell and type the following command: `condor_store_cred add`
   - Enter your password and you are ready to submit your first HTCondor job.
   - If this does not work, try to enter `condor_store_cred add -debug` for more output information on the error.
2. Please make sure that *nextnanomat* has successfully found the HTCondor pool. In *nextnanomat* go to `Tools -> Options -> Cloud computing`. If everything is correctly set up, you will find the "HTCondor" section highlighted with green color, and the available computers show up in "Cluster". If this is not the case, maybe you have not installed HTCondor on the computer where you are running *nextnanomat*. Please also check that the HTCondor installation path is correctly set within *nextnanomat*, e.g. the default path `C:\condor` might not be the one where you installed HTCondor.

12.2.1 Summary of settings (Example)

```
Hostname (for HTCondor pool): computername.yourcompanyname.com
Policy: "Always run jobs"
Accounting domain: yourcompanyname.com
Read access: *
Write access: $(CONDOR_HOST), $(IP_ADDRESS), *.yourcompanyname.com, 192.168.178.*
Administrator: $(IP_ADDRESS)
```

12.2.2 Config file

You can find your HTCondor config settings in the file `C:\condor\condor_config`. Let's look at an example below.

- Your company is called `Simpson`.
- Your Windows domain is called `simpson.com`.
- Your HTCondor pool shall have the name `TheSimpsonsCondorPool`.
- The HTCondor host that manages the HTCondor jobs has the computer name `homer.simpson.com`.

- Your computer is called `lisa.simpson.com`.
- The computers in your network have the IP range `192.168.188.*`. (or `2001:db8:2042::*` in IPv6)

```
 RELEASE_DIR = C:\condor
 LOCAL_CONFIG_FILE = $(LOCAL_DIR)\condor_config.local
 REQUIRE_LOCAL_CONFIG_FILE = FALSE
 LOCAL_CONFIG_DIR = $(LOCAL_DIR)\config
 use SECURITY : HOST_BASED
#CONDOR_HOST: $(FULL_HOSTNAME)          # on computer called homer
 CONDOR_HOST: homer                      # on computer called lisa
 COLLECTOR_NAME = TheSimpsonsCondorPool  # only on computer called homer
#UID_DOMAIN =                           # empty if you do not have a domain
 UID_DOMAIN = simpson.com
 SOFT_UID_DOMAIN=TRUE             # entry is missing if you do not have a domain
 FILESYSTEM_DOMAIN = simpson.com  # entry is missing if you do not have a domain
 CONDOR_ADMIN =
 SMTP_SERVER =
 ALLOW_READ = *
 ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS), *.simpson.com, 192.168.188.*,␣
↪2001:db8:2042::*
 ALLOW_ADMINISTRATOR = $(IP_ADDRESS)
 use POLICY : ALWAYS_RUN_JOBS
#use POLICY : DESKTOP
 WANT_VACATE = FALSE
 WANT_SUSPEND = TRUE
#DAEMON_LIST = MASTER SCHEDD COLLECTOR NEGOTIATOR STARTD # on computer called homer
#DAEMON_LIST = MASTER SCHEDD STARTD KBDD                 # on computer called lisa if␣
↪keyboard idle 15 minutes option was chosen
 DAEMON_LIST = MASTER SCHEDD STARTD                      # on computer called lisa
```

12.2.3 Configuring a pool without a domain

It is also possible to set up a HTCondor pool with computers that are not on a domain by using the **NO_DNS** option in the compig file. This also provides an easy way to setup a mixed pool with Windows and Linux machines. Below are example comfigurations for central manager and submit/execute nodes for Windows and Linux, respectively. The user passwords on Windows machines need to be stored by `condor_store_cred add` command as above, on Linux machines no password is needed.

Central Manager on Windows:

```
RELEASE_DIR = C:\condor
LOCAL_DIR = $(RELEASE_DIR)
qLOCAL_CONFIG_FILE = $(LOCAL_DIR)\condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = FALSE

use SECURITY : HOST_BASED

CONDOR_HOST = $(FULL_HOSTNAME)
COLLECTOR_NAME = EXAMPLE_POOL

NO_DNS = True
DEFAULT_DOMAIN_NAME = MY_DOMAIN

ALLOW_CONFIG = $(IP_ADDRESS)
ALLOW_ADMINISTRATOR = $(CONDOR_HOST), $(IP_ADDRESS), condor_pool@(UID_DOMAIN),␣
↪EXAMPLE_POOL
```

(continues on next page)

(continued from previous page)

```
ALLOW_READ = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_WRITE = $(IP_ADDRESS)/8, $(CONDOR_HOST)

ALLOW_NEGOTIATOR = $(CONDOR_HOST), condor_pool@$(UID_DOMAIN)

ALLOW_ADVERTISE_MASTER = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_STARTD = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_SCHEDD = $(IP_ADDRESS)/8, $(CONDOR_HOST)

ALLOW_DAEMON = condor_pool@$(UID_DOMAIN), condor@$(UID_DOMAIN), $(IP_ADDRESS)
DAEMON_LIST = MASTER STARTD SCHEDD NEGOTIATOR COLLECTOR

use POLICY : ALWAYS_RUN_JOBS
WANT_VACATE = FALSE
WANT_SUSPEND = TRUE
```

Submit/Execute Node on Windows:

```
RELEASE_DIR = C:\condor
LOCAL_DIR = $(RELEASE_DIR)
LOCAL_CONFIG_FILE = $(LOCAL_DIR)\condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = FALSE

use SECURITY : HOST_BASED

CONDOR_HOST = 192.168.188.xy #substitute with the ip address of the central manager
COLLECTOR_HOST = $(CONDOR_HOST)

NO_DNS = True
DEFAULT_DOMAIN_NAME = MY_DOMAIN

ALLOW_CONFIG = $(IP_ADDRESS), $(CONDOR_HOST)
ALLOW_ADMINISTRATOR = $(CONDOR_HOST), $(IP_ADDRESS), condor_pool@(UID_DOMAIN),␣
↪EXAMPLE_POOL

ALLOW_READ = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_WRITE = $(IP_ADDRESS)/8, $(CONDOR_HOST)

ALLOW_ADVERTISE_MASTER = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_STARTD = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_SCHEDD = $(IP_ADDRESS)/8, $(CONDOR_HOST)

ALLOW_DAEMON = condor_pool@$(UID_DOMAIN), condor@$(UID_DOMAIN), $(IP_ADDRESS),
↪$(CONDOR_HOST)
DAEMON_LIST = MASTER STARTD SCHEDD

use POLICY : ALWAYS_RUN_JOBS
WANT_VACATE = FALSE
WANT_SUSPEND = TRUE
```

Central Manager on Linux:

```
RELEASE_DIR = /usr
LOCAL_DIR = /var
LOCAL_CONFIG_FILE = /etc/condor/condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = false
```

(continues on next page)

(continued from previous page)

```
LOCAL_CONFIG_DIR = /etc/condor/config.d

##  Pathnames
RUN      = $(LOCAL_DIR)/run/condor
LOG      = $(LOCAL_DIR)/log/condor
LOCK     = $(LOCAL_DIR)/lock/condor
SPOOL    = $(LOCAL_DIR)/spool/condor
EXECUTE = $(LOCAL_DIR)/lib/condor/execute
BIN      = $(RELEASE_DIR)/bin
LIB      = $(RELEASE_DIR)/lib/condor
INCLUDE = $(RELEASE_DIR)/include/condor
SBIN     = $(RELEASE_DIR)/sbin
LIBEXEC = $(RELEASE_DIR)/lib/condor/libexec
SHARE    = $(RELEASE_DIR)/share/condor
GANGLIA_LIB64_PATH = /lib,/usr/lib,/usr/local/lib
PROCD_ADDRESS = $(RUN)/procd_pipe

use SECURITY : HOST_BASED

CONDOR_HOST = $(FULL_HOSTNAME)
COLLECTOR_NAME = EXAMPLE_POOL

NO_DNS = True
DEFAULT_DOMAIN_NAME = MY_DOMAIN

ALLOW_CONFIG = $(IP_ADDRESS)
ALLOW_ADMINISTRATOR = $(IP_ADDRESS), $(CONDOR_HOST), condor_pool@(UID_DOMAIN),␣
↪EXAMPLE_POOL

ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS)/8
ALLOW_READ = $(CONDOR_HOST), $(IP_ADDRESS)/8

ALLOW_NEGOTIATOR = $(CONDOR_HOST), condor_pool@$(UID_DOMAIN)

ALLOW_ADVERTISE_MASTER = $(IP_ADDRESS)/8
ALLOW_ADVERTISE_STARTD = $(IP_ADDRESS)/8
ALLOW_ADVERTISE_SCHEDD = $(IP_ADDRESS)/8

ALLOW_DAEMON = condor_pool@$(UID_DOMAIN), condor@$(UID_DOMAIN), $(IP_ADDRESS)
DAEMON_LIST = MASTER, STARTD, SCHEDD, NEGOTIATOR, COLLECTOR

# Don't phone home
CONDOR_DEVELOPERS = NONE
CONDOR_DEVELOPERS_COLLECTOR = NONE

SSH_TO_JOB_SSHD_CONFIG_TEMPLATE = /etc/condor/condor_ssh_to_job_sshd_config_template
```

Submit/Execute Node on Linux:

```
RELEASE_DIR = /usr
LOCAL_DIR = /var
LOCAL_CONFIG_FILE = /etc/condor/condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = false
LOCAL_CONFIG_DIR = /etc/condor/config.d

##  Pathnames
```

(continues on next page)

(continued from previous page)

```
RUN      = $(LOCAL_DIR)/run/condor
LOG      = $(LOCAL_DIR)/log/condor
LOCK     = $(LOCAL_DIR)/lock/condor
SPOOL    = $(LOCAL_DIR)/spool/condor
EXECUTE = $(LOCAL_DIR)/lib/condor/execute
BIN      = $(RELEASE_DIR)/bin
LIB      = $(RELEASE_DIR)/lib/condor
INCLUDE = $(RELEASE_DIR)/include/condor
SBIN     = $(RELEASE_DIR)/sbin
LIBEXEC = $(RELEASE_DIR)/lib/condor/libexec
SHARE    = $(RELEASE_DIR)/share/condor
GANGLIA_LIB64_PATH = /lib,/usr/lib,/usr/local/lib
PROCD_ADDRESS = $(RUN)/procd_pipe

use SECURITY : HOST_BASED

CONDOR_HOST = 192.168.188.xy #subsitute with the ip address of the central manager
COLLECTOR_HOST = $(CONDOR_HOST)

NO_DNS = True
DEFAULT_DOMAIN_NAME = MY_DOMAIN

ALLOW_CONFIG = $(IP_ADDRESS)
ALLOW_ADMINISTRATOR = $(IP_ADDRESS), $(CONDOR_HOST), condor_pool@(UID_DOMAIN),␣
↪EXAMPLE_POOL

ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS)/8
ALLOW_READ = $(CONDOR_HOST), $(IP_ADDRESS)/8

ALLOW_ADVERTISE_MASTER = $(IP_ADDRESS)/8
ALLOW_ADVERTISE_STARTD = $(IP_ADDRESS)/8
ALLOW_ADVERTISE_SCHEDD = $(IP_ADDRESS)/8

ALLOW_DAEMON = condor_pool@$(UID_DOMAIN), condor@$(UID_DOMAIN), $(IP_ADDRESS)
DAEMON_LIST = MASTER, STARTD, SCHEDD

# Don't phone home
CONDOR_DEVELOPERS = NONE
CONDOR_DEVELOPERS_COLLECTOR = NONE

SSH_TO_JOB_SSHD_CONFIG_TEMPLATE = /etc/condor/condor_ssh_to_job_sshd_config_template
```

12.3 Submitting jobs to HTCondor pool with nextnanomat

Submit job

1. Add a job to the Batch list in the **Run** tab.
2. Click on the **Run in HTCondor Cluster** button (button with triangle and network).

Show information on HTCondor cluster

1. Click on **Show Additional Info for Cluster Simulation**.
2. Press the **Refresh** button on the right.
3. The results of the `condor_status` command are shown, i.e. the number of compute slots are displayed.

4. You can select another HTCondor command such as `condor_q` to show the status of your submitted jobs, i.e. select `condor_q`, and then press the **Refresh** button.

- You can type in any command in the line **System command:**, e.g. `dir`.
- The button **Open Documentation** opens the online documentation (this website).

Results of HTCondor simulations

- Once your HTCondor jobs are finished, the results are automatically copied back to your simulation output folder `<nextnano simulation output folder\<name of input file>\`.
- For debugging purposes regarding the HTCondor job, you can analyze the generated log file, `<input file name>.log`.

12.4 Useful HTCondor commands for the Command Prompt

- `condor_submit <filename>.sub` Submit a job to the pool.
- `condor_q` Shows current state of own jobs in the queue.
  - `condor_q -nobatch -global -allusers` Shows state of all jobs in the cluster. Of all users.
  - `condor_q -goodput -global -allusers` Shows state and occupied CPU of all jobs in the cluster.
  - `condor_q -allusers -global -analyze` Detailed information for every job in the cluster.
  - `condor_q -global -allusers -hold` Shows why jobs are in hold state.
- `condor_status` Shows state of all available resources.
- `condor_status -long` Shows state of all available resources and many other information.
- `condor_status -debug` Shows state of all available resources and some additional information, e.g. *WARNING: Saw slow DNS query, which may impact entire system: getaddrinfo(<Computername>) took 11.083566 seconds.*
- `condor_rm` Remove jobs from a queue:
  - `condor_rm -all` Removes all jobs from a queue.
  - `condor_rm <cluster>.<id>` Removes jobs on cluster <cluster> with id <id> (It seems `<cluster>.` can be omitted, and `id` is the `JOB_IDS` number.)
- `condor_release -all` If any jobs are in state hold, use this command to restart them.
- `condor_restart` Restart all HTCondor daemons/services after changes in config file.
- `condor_version` Returns the version number of HTCondor
- `condor_store_cred query` Returns info about the credentials stored for HTCondor jobs
- `condor_history` Lists the recently submitted jobs. If for a specific job `ID` the status has the value `ST=C`, then this job has been completed (`C`) successfully.
- `condor_status -master`: returns Name, HTCondor Version, CPU and Memory of central manager
- Open Command Prompt `cmd.exe` as Administrator. Type in: `net start condor`. This has the same effect as restarting your computer, i.e. the networking service `condor` is started. This is useful if you have changed your local `condor_config` file.

12.5 HTCondor Pool - Managing Slots

- Each PC runs one `condor_startd` daemon. By default, the `condor_startd` will automatically divide the machine into slots, placing one core in each slot. E.g. a 6-core computer with hyperthreading has 12 logical processors. Alternatively, the number of cores (or logical processors) can be distributed to the slots as follows.

```
SLOT_TYPE_1 = cpus=4
SLOT_TYPE_2 = cpus=4
SLOT_TYPE_3 = cpus=2
SLOT_TYPE_4 = cpus=1
SLOT_TYPE_5 = cpus=1
SLOT_TYPE_1_PARTITIONABLE = TRUE
SLOT_TYPE_2_PARTITIONABLE = TRUE
SLOT_TYPE_3_PARTITIONABLE = TRUE
SLOT_TYPE_4_PARTITIONABLE = TRUE
SLOT_TYPE_5_PARTITIONABLE = TRUE
NUM_SLOTS_TYPE_1 = 1
NUM_SLOTS_TYPE_2 = 1
NUM_SLOTS_TYPE_3 = 1
NUM_SLOTS_TYPE_4 = 1
NUM_SLOTS_TYPE_5 = 1
```

- `PartitionableSlot`: For SMP (symmetric multiprocessing) machines, a boolean value identifying that this slot may be partitioned.
- `DynamicSlot`: For SMP machines that allow dynamic partitioning of a slot, this boolean value identifies that this dynamic slot may be partitioned.
- `SlotID`: For SMP machines, the integer that identifies the slot.
- A useful command might be: `condor_status -af Name TotalCpus DynamicSlot PartitionableSlot SlotID`. It returns the requested properties of each slot:
  - `Name`
  - `TotalCpus`
  - `DynamicSlot`
  - `PartitionableSlot`
  - `SlotID`

12.5.1 Dynamic slots

In our pool we have chosen dynamic partitioning which gives full flexibility. For instance, a quad-core CPU that is dynamically partitioned can accept

- 4 single-threaded jobs (`request_cpus = 1`)
- 2 jobs with 2 threads each (`request_cpus = 2`)
- 2 jobs of which one is single-threaded (`request_cpus = 1`) and the other uses 3 threads (`request_cpus = 3`)
- 1 job with 4 threads (`request_cpus = 4`).

```
#####################################################
# Dynamic partitioning
# We use HTCondors dynamic partitioning mechanism.
# Each PC has one partitionable whole machine slot.
```

(continues on next page)

(continued from previous page)

```
# (It seems that hyperthreading is not taken into account.)
##################################################
NUM_SLOTS = 1
NUM_SLOTS_TYPE_1 = 1
SLOT_TYPE_1 = 100%
SLOT_TYPE_1_PARTITIONABLE = true
SlotWeight = Cpus
```

12.6 Machine states

A machine is in any of the following 6 states. The most important one are `Owner`, `Unclaimed`, `Claimed`.

- `Backfill`: (not relevant for us)
- `Owner`: The machine is being used by the machine owner, and/or is not available to run HTCondor jobs. When the machine first starts up, it begins in this state.
- `Unclaimed`: The machine is available to run HTCondor jobs, but it is not currently doing so.
- `Matched`: The machine is available to run jobs, and it has been matched by the negotiator with a specific schedd. That schedd just has not yet claimed this machine. In this state, the machine is unavailable for further matches.
- `Claimed`: The machine has been claimed by a schedd.
- `Preempting`: The machine was claimed by a schedd, but is now preempting that claim for one of the following reasons.

```
- The owner of the machine came back
- Another user with higher priority has jobs waiting to run.
- Another request that this resource would rather serve was found.
```

12.7 Machine activities

Each machine state can have different activities. The machine state `Claimed` can have one out of these four activities.

- `Idle`:
- `Busy`:
- `Suspended`:
- `Retiring`:

12.8 Configuration options for the Central Manager computer

With this option in the `condor.config` file on the central manager, one can set a policy that the jobs are spread out over several machines rather than filling all slots of one computer before filling the slots of the other computers.

```
##------nn: SPREAD JOBS BREADTH-FIRST OVER SERVERS
##-- Jobs are "spread out" as much as possible,
##   so that each machine is running the fewest number of jobs.
NEGOTIATOR_PRE_JOB_RANK = isUndefined(RemoteOwner) * (- SlotId)
```

12.9 FAQ

Q: I submitted a job to HTCondor, but nothing happens. The *nextnanomat* GUI says "transmitted".

A: It could be that *nextnanomat* does not have read in all required settings. You can try to type in the command line `condor_restart`. Please make sure that you entered your credentials using `condor_store_cred add -debug`. You should then start *nextnanomat* again.

Q: I submitted a job to HTCondor, but the Batch line of *nextnanomat* is stuck with `preparing`. What is wrong?

A1: Did you store your credentials after the installation of HTCondor? If not, enter `condor_store_cred add` into the command prompt to add your password, see above (Recommended Installation Process).

A2: Did you change your password recently? If yes you have to reenter your credentials for HTCondor. Enter `condor_store_cred add` into the command prompt to add your password, see above (Recommended Installation Process). If this does not work, try to enter `condor_store_cred add -debug` for more output information on the error.

Q: I specified target machines in Tools - Options. Afterwards every submitted job to HTCondor is stuck with `transmitting`. What is wrong?

A: The value for `UID_DOMAIN` within the condor_config file needs to be the same for every computer of your cluster. (You can easily test it in a command prompt with `condor_status -af uiddomain`) If it's not the same value, no matching computer will be found and the job won't be transmitted successfully.

12.10 Problems with HTCondor

12.10.1 Error: communication error

If you receive the following error when you type in `condor_status`

```
C:\Users\"<your user name>">condor_status
Error: communication error
CEDAR:6001:Failed to connect to <123.456.789.123>
```

you can check whether the computer associated with this IP address is your HTCondor computer using the following command.

```
nslookup 123.456.789.123
```

It is also a good idea to type in

```
nslookup
```

This will return the name of the Default Server that resolves DNS names. If it is not the expected computer, you can open a Command Prompt as **Administrator** and type in `ipconfig /flushdns` to flush the DNS Resolver Cache.

```
C:\Users\"<your user name>">ipconfig /flushdns
```

If the DNS address cannot be resolved correctly it could be related to a VPN connection that has configured a different default server for Domain Name to IP address mapping. E.g. if your Windows Domain is called contoso.com (which is only visible within your own network and your own HTCondor pool) but your DNS is resolved to www.contoso.com (which might be outside your local HTCondor pool).

12.10.2 Error: `condor_store_cred add` failed with `Operation failed. Make sure your ALLOW_WRITE setting include this host.`

Solution: Edit `condor_config` file and add host, i.e. local computer name (here: nn-delta).

```
    ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS)
==> ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS), nn-delta
```

12.10.3 Error? Check the Log files

If you encounter any strange errors, you can find some hints in the history or Log files generated by HTCondor. You can find them here:

`C:\condor\spool`

- history

`C:\condor\log`

- CollectorLog
- MasterLog
- MatchLog
- NegotiatorLog
- ProcLog
- SchedLog
- ShadowLog
- SharedPortLog
- StarterLog
- StartLog

More details can be found here: Logging in HTCondor

12.11 Known bugs

- HTCondor < 8.9.5 works with all nextnano GmbH executables
- HTCondor >= 8.9.5 works with nextnano GmbH executables newer than 2020-Jan

12.12 Run your custom executable on HTCondor with nextnanomat

You can even run your own executable with *nextnanomat* locally or on HTCondor! We tested the following programs:

- `HelloWorld.exe`
- Quantum ESPRESSO (`pw.exe`)
- ABINIT (`abinit.exe`)

12.12.1 Input file identifier

An input file identifier is a special string in the input file that signals to *nextnanomat* whether the input file is an input file for the *nextnano++*, *nextnano*^{3}, *nextnano.NEGF* or *nextnano.MSB* tools, or for a custom executable.

12.12.2 Settings for Hello World (HW)

In *nextnanomat*, we need the following settings:

- Path to executable file: e.g. `D:\HW\HelloWorld.exe`
- Input file identifier: e.g. `HelloWorld`
- Working directory: Select 'Simulation output folder'
- HTCondor: Output folder and files (transfer_output_files = ...): `.`

Open input file `input_file_for_HelloWorld.in` (or any other input file that contains the string `HelloWorld`) and run the simulation either locally or on HTCondor.

12.12.3 Settings for Quantum ESPRESSO (QE)

Our folder structure is

- `D:\QE\inputfile\My_QE_inputfile.in` (QE input file)
- `D:\QE\input\pseudo\C.UPF` (pseudopotential file for atom species 'C' as specified in input file)
- `D:\QE\exe\pw.exe` (QE executable file)
- `D:\QE\exe\*.dll` (all dll files needed by pw.exe)
- `D:\QE\working_directory\QE_nextnanomat_HTCondor.bat` (batch file)

In *nextnanomat*, we need the following settings:

- Path to executable file: e.g. `D:\QE\working_directory\QE_nextnanomat_HTCondor.bat`
- Path to folder with additional files: `D:\QE\`
- Input file identifier: e.g. `&control`
- Working directory: Select 'Simulation output folder'
- HTCondor: Output folder and files (transfer_output_files = ...): `.`
- (Additional arguments passed to the executable: `$INPUTFILE`)

The batch file (`*.bat`) contains the following content:

```
.\exe\pw.exe -in .\inputfile\My_QE_inputfile.in
```

This means that relative to the working directory, `pw.exe` is started, and the specified input file is read in. In this input file, the following quantities are specified:

- `C.UPF`: name of pseudopotential file
- `./input/pseudo/`: path to pseudopotential file `C.UPF`

Open input file `My_QE_inputfile.in` and run the simulation either locally or on HTCondor.

Things that could be improved:

- Write all files into output folder created by *nextnanomat*. In particular, the folder output/ should be moved.
- `condor_exec.exe` is deleted (better: do not copy it back)
- all `*.dll` files should be deleted (better: do not copy them back)
- Don't copy back *.exe and *.dll files (both HTCondor and local)

12.12.4 Settings for ABINIT

Our folder structure is

- `D:\abinit\inputfile\t30.in` (ABINIT input file)
- `D:\abinit\input\*` (input files needed by ABINIT)
- `D:\abinit\exe\abinit.exe` (ABINIT executable file)
- `D:\abinit\exe\*.dll` (all dll files needed by abinit.exe)
- `D:\abinit\working_directory\abinit_nextnanomat.bat` (batch file)
- Path to executable file: e.g. `D:\abinit\working_directory\abinit_nextnanomat.bat`
- Path to folder with additional files: `D:\abinit\`
- Input file identifier: e.g. `acell`
- Working directory: Select 'Simulation output folder'
- HTCondor: Output folder and files (transfer_output_files = ...): `.`
- Additional arguments passed to the executable: (empty)

The batch file (`*.bat`) contains the following content:

```
.\exe\abinit.exe < .\input\ab_nextnanomat_HTCondor.files
```

This means that relative to the working directory, `abinit.exe` is started, and the specified input file is read in. In this input file, the following quantities are specified:

- `.\inputfile\t30.in`: name of input file
- `.\input\14si.pspnc`:

Open input file `t30.in` and run the simulation either locally or on HTCondor.

Notes

- `condor_exec.exe` is deleted (better: do not copy it back)
- all `*.dll` files should be deleted (better: do not copy them back)
- Don't copy back *.exe and *.dll files (both HTCondor and local)

CHAPTER

THIRTEEN

SUPPORT TICKET SYSTEM

We are using a support system on all our channels. You have plenty of possibilities to get in contact with us. Whether you prefer writing emails, using the widget on our website or the Help Center directly, all requests are collected centrally and will be processed by their number - fair and square. Note, unless you purchased a special support option only technical support is included in your license fee, i.e. **no** simulation requests or adapting your input files.

This support system also offers the possibility of giving feedback. Critique and suggestions are highly appreciated.

13.1 How to get the fastest support possible?

There are certain rules you can follow to faciliate the work of the support team and thus ensure fast and meaningful support.

1. **Create *single* and *new* tickets for each separate topic**
   a. Do not answer to resolved support tickets, if you have a new unrelated question. (There is a risk we won't get notified at all)
   b. Do not mix up questions about e.g. graphical user interface, syntax of the input file, simulation results, errors, billing etc.
2. **Give a meaningful subject - it will be easier to find the right person for your request**
   a. Don'ts: *I need help*
   b. Dos: *How can I plot multiple files superimposed?*
3. **Give us all the information we need**
   a. Describe what led to the observed behaviour
   b. Attach screenhots of error messages or input and log files
   c. Refer to this documentation to find the right files: *Preparing Support Request*
4. **Stay polite and respectful**
   a. Please sign your request with your name, so we can address you ;)
   b. Check for spelling and grammar - it's a hassle to guess the meaning of sentences

c. Support conversations can be a helpful and satisfactory process for both sides, it's in our hands to make them this way.

13.2 Channels

13.2.1 Help Center (recommended)

In the help center https://nextnano.atlassian.net/servicedesk you can raise a new request or view your current or closed support tickets. It is all kept together, media files and conversation on your request.

How to set up your free customer account

This only needs to be done **once**. If you encounter any problems, don't hesitate to contact us! Just click on the link `View your support ticket` in the first mail you receive, after creating a ticket (see Figure 13.2.1.1). You will be leaded through the process of setting up your help center account. Note, you don't need an account for the communication on your ticket, but you will need one to access files which are provided by our support team.

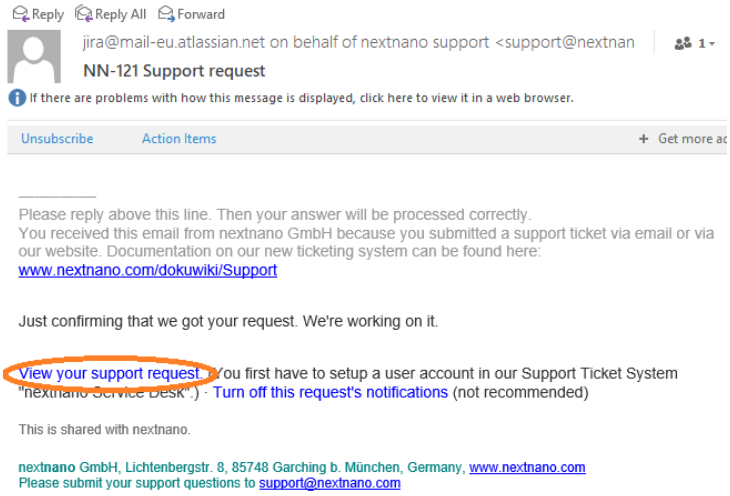


Figure 13.2.1.1: Email confirmation after creating a support request.

13.2.2 Widget

The easiest way, if you encounter a problem or do not find what you search for on our website - just click on the widget in the bottom right corner and fill out the form. The Widget is not suitable for elaborate support requests with multiple files, but rather quick questions. If you have a larger support request, use our help center.

13.2.3 Email

Emails to support [at] nextnano.com will automatically create a support ticket.

CHAPTER

FOURTEEN

PREPARING SUPPORT REQUEST

There are four major groups of issues that our users are typically facing. Below you can find guidelines about to prepare a support ticket for your problem.

14.1 nextnano simulation does not start

14.1.1 Possible error sources

If a nextnano GmbH simulation will not start, possible error sources could be

- the license
- the path to the nextnano GmbH executable
- the path to the output directory
- the path to the material database
- the path or syntax of the input file

So first check the *nextnanomat* settings ('Tools' -> 'Options') for

1. 'Simulation'
2. 'Material database'
3. 'Licenses'
4. 'View/Output'

Additionally, verify if your input file does exist and is not located on a shared drive. For help on the input file syntax, go to the product specific syntax documentation for either *nextnano*[3], *nextnano++*, *nextnano.NEGF* or *nextnano.MSB*.

14.1.2 Files to be included in support request

If you are unable to solve the error on your own, please provide the following files for your support request:

1. System snapshot for trouble shooting
2. Screenshot of error message, if there is any
3. Simulation log file, if there is any, including additional debug comments
4. Simulation input file

Have a look at *Preparing Support Request* if you do not know how generate/find these files.

14.2 nextnano simulation does not finish (errors in log file)

Note, if the error is connected to the syntax of the input file, please try to resolve the error on your own. You can find the product specific syntax documentation within the product sections of this documentation.

If an error message occurs during the simulation and you do not know how to resolve it on your own, add the following files to your support request:

1. System snapshot for trouble shooting
2. Screenshot of error message
3. Simulation log file, including additional debug comments
4. Simulation input file

Have a look at *Preparing Support Request* if you do not know how generate/find these files.

14.3 The nextnanomat GUI does not start

Sometimes the application will not start or crash unexpectedly. The Windows event log file can be helpful to troubleshoot errors. You can find them by opening the 'Event Viewer' application:

1. Search for 'event viewer' in the Windows search bar
2. Select and open the Event Viewer application
3. Navigate to 'Windows Logs' and select 'Application'
4. You will see a list of recently occured errors and warnings
5. Select the Application error which correlates with the time of the application crash
6. Copy the information within the 'General' tab
7. Paste the information into a text editor (e.g. Notepad++) and save as `.txt`
8. Additionally, you can take a screenshot of the Event Viewer and save as `.jpg` or `.png` file
9. Attach these files to your support request
10. Note, sometimes two error events belong to one application crash. Then proceed as described for both of them.

14.4 The nextnanomat GUI prompts error or exception message

If an error message pops up during the runtime of *nextnanomat*, please send the following files with your support request:

1. System snapshot for trouble shooting
2. Screenshot of error message

Have a look at *Preparing Support Request* if you do not know how generate/find these files.

Useful links:

- *Options: Simulation*
- *Options: Licenses*
- *Options: View/Output*
- *Options: Material database*
- *Where to find simulation LOG file*
- *How to add additional debug information to the LOG file*
- *Generate System Snapshot for Troubleshooting*
- *FAQ - Errors and Exceptions*

CHAPTER

FIFTEEN

FREQUENTLY ASKED QUESTIONS (FAQ)

15.1 FAQ - General

15.1.1 Copyright Statement

15.1.2 Are there any video tutorials available?

Yes, there are! Check out our Sway presentation and our YouTube channel.

15.1.3 Hardware requirements for nextnano? I want to buy a new computer. What shall I buy?

The nextnano software even runs on Laptops. Therefore, for most typical simulations, you don't have to buy a special computer. Still, you might have CPU-intensive calculations which require more horsepower. In this case, the following configurations or their equivalent will give excellent performance for about 1.000 € or less:

CPU+RAM
Intel i7-8700, 6 cores, 3.2 GHz (Coffee Lake) with 32 GB DDR4-2666 RAM, or Intel i7-7700, 4 cores, 3.6 GHz (Kaby Lake) with 32 GB DDR4-2400 RAM.

Note: These CPUs can be bought as ''boxed" (including a default cooler) or ''tray" (without default cooler). However, both CPUs are too noisy with the default cooler in the boxed variant when running simulations. We recommend buying a large additional CPU cooler.

Main board
Any compatible motherboard will work, please select according to your extensibility needs, required peripheral ports, etc. The integrated graphics on the mainboard usually suffices for office use and development. But be prepared to add a dedicated graphics card for CAD, multi-monitor setups and the likes.

Storage
E.g. a 500 GB SSD for the OS and programs, together with a 2 TB HD for simulation data.

Power supply
E.g. 500 W to allow future upgrades such as a dedicated graphics card.

Computer case
Select according to available space, future extensibility, and aesthetic desire. Silent cases are recommended for reducing noise. Please make sure that there is enough clearance to fit your CPU cooler and large additional components such as graphics cards inside.

Recommended BIOS settings (if available)

- CPU: Hyperthreading ⇒ Enabled (accelerates compilation)
- CPU: VT-d ⇒ Enabled (accelerates virtualization)
- CPU: Hardware prefetch ⇒ Enabled (accelerates linear memory accesses)
- **RAM: XMP ⇒ enabled (if disabled, RAM much slower than maximum supported speed,**
  e.g. 2166 instead of maximum supported 2666 (DDR4-2666))

(For the experts: If you are planning to have your PC assembled from components, please use tools such as prime95, Intel Extreme Tuning Utility, and/or Intel Linpack to test system stability and adequate cooling under extended heavy load.)

Example: Our latest computers have the following configurations

i7-8700 (purchased 2018-Oct: ~1.000 EUR, purchased 2018-Dec: ~900 EUR)

- CPU: Intel i7-8700, 6 cores, 3.2 GHz (Coffee Lake) (a large additional CPU cooler is recommended, see comments above)
- CPU cooler, e.g. be quiet! Pure Rock
- RAM: 32 GB (DDR4-2666), e.g. Corsair Vengeance
- Storage: 500 GB SDD + 3 TB HDD
- Motherboard with integrated graphics processing unit, e.g. ASRock Z370M-ITX/ac (CPU socket 1151)
- Power supply: e.g. be quiet! Pure Power 10-CM or 11, 500 W
- Computer case: Midi Tower, e.g. Zalman R1 or be quiet! Pure Base 600

i7-7700 (purchased 2018-Jan, ~900 EUR)

- CPU: Intel i7-7700, 4 cores, 3.6 GHz (Kaby Lake) (a large additional CPU cooler is recommended, see comments above)
- CPU cooler, e.g. be quiet! Pure Rock Slim
- RAM: 32 GB (DDR-2400), e.g. Corsair Vengeance
- Storage: 256 GB SDD + 2 TB HDD
- Motherboard with integrated graphics processing unit, e.g. ASRock Z270M-ITX/ac
- Power supply: e.g. Pure Power 10, 500 W ATX 2
- Computer case: Midi Tower, e.g. Zalman R1

15.1.4 How shall I cite the nextnano software in publications?

You can cite any of the following papers:

- nextnano: General Purpose 3-D Simulations

  S. Birner, T. Zibold, T. Andlauer, T. Kubis, M. Sabathil, A. Trellakis, P. Vogl

  IEEE Trans. Electron Dev. **54**, 2137 (2007)

- The 3D nanometer device project nextnano: Concepts, methods, results

  A. Trellakis, T. Zibold, T. Andlauer, S. Birner, R. K. Smith, R. Morschl, P. Vogl

  J. Comput. Electron. **5**, 285 (2006)

For simulations including electrolytes, you should cite:

- Theoretical model for the detection of charged proteins with a silicon-on-insulator sensor

  S. Birner, C. Uhl, M. Bayer, P. Vogl

  J. Phys.: Conf. Ser. **107**, 012002 (2008)

For simulations that use the Contact Block Reduction method (CBR) (ballistic transport), you should cite any of the following papers:

- Efficient method for the calculation of ballistic quantum transport

  D. Mamaluy, M. Sabathil, P. Vogl

  J. Appl. Phys. **93**, 4628 (2003)

- Ballistic quantum transport using the contact block reduction (CBR) method - An introduction

  S. Birner, C. Schindler, P. Greck, M. Sabathil, P. Vogl

  J. Comput. Electron. **8**, 267 (2009)

The *nextnano.MSB* tool: For simulations that use the multi-scattering Büttiker (MSB) probe model (NEGF), you should cite:

- Efficient method for the calculation of dissipative quantum transport in quantum cascade lasers

  P. Greck, S. Birner, B. Huber, P. Vogl

  Optics Express **23**, 6587

The *nextnano.NEGF* tool: For simulations that use the NEGF method, you should cite:

- Contrasting influence of charged impurities on transport and gain in terahertz quantum cascade lasers

  T. Grange

  Phys. Rev. B **92**, 241306(R) (2015)

For simulations that use the NEGF algorithm included in *nextnano*^{3}, you should cite any of these publications:

- Modeling techniques for quantum cascade lasers

  C. Jirauschek, T. Kubis

  Appl. Phys. Rev. **1**, 011307 (2014)

- Theory of non-equilibrium quantum transport and energy dissipation in terahertz quantum cascade lasers

  T. Kubis, C. Yeh, P. Vogl, A. Benz, G. Fasching, C. Deutsch

  Phys. Rev. B **79**, 195323 (2009)

There might be further papers in the literature that are more suited to be cited in certain cases.

15.2 FAQ - Licensing

15.2.1 Which types of licenses exist?

There are three types of licenses:

- University license
- Government institution license
- Company license

15.2.2 How many people can use the software simultaneously?

This depends on the type of license:

- A *University license* and a *Government institution license* are issued to a particular research group (e.g. a professor or group leader) and can be used by all group members simultaneously. The university and government institution licenses can be used on several computers simultaneously. A university license includes 10 computers, a government institution license includes 5 computers.
- A *Company license* applies to a single and named user or a single-PC.

The license is an annual license. After the license has expired, no further simulations can be done. Visualization of previous results of calculations is still possible.

If a user has a valid license, this license can also be installed on a private computer.

15.2.3 Which license do I need for Cluster computing?

Only *.lic* licenses can be used for cluster computing. Any computer that has the free HTCondor software installed can be connected to a licensed computer. There is **no** need to activate a license for the cluster computers or install nextnano GmbH on them. They can be used to execute a batch list of nextnano GmbH simulations.

15.2.4 Is there a possibility to evaluate the nextnano software before purchasing?

Yes, we offer evaluation licenses with feature restriction. You have to register with an institutional email address. Register now.

If you got granted an evaluation period for a product of the nextnano software, you will receive an `evaluation license key` by mail. Your evaluation license (free-of-charge) will be valid for 1 month starting on the day you receive the mail and can be activated on 1 computer. If you need a prolongation, please do not hesitate to contact us. Note, we reserve the right to not grant an evaluation license, for example if the same user already used an evaluation license in previous years.

15.2.5 I remember there have been *.txt* licenses before, what happened to them?

.txt has been the extension of our outdated license format. This license format has been abolished due to security risks and is not used anymore. The current format has the *.lic* extension and enables a wider range of features within the nextnano software.

15.2.6 License activation

I don't have a license (key)

Before being able to run any simulations with nextnano GmbH, you need a valid license. After your purchase order is complete you will receive a `license key` by mail. This key is necessary to activate your license. If you haven't purchased your license yet, please contact us.

I already purchased a license

For each PC where you want to use nextnano GmbH, you have to activate your license(s). For the online activation, you need an internet connection. There are also possibilities to activate your license without an internet connection. The license activation procedure is documented here: *License Activation*

Can I activate my license without using nextnanomat?

The license activation procedure via command prompt is documented here: *License activation via command line*.

I encounter an error during license activation

There are multiple possible error sources...

Connection error
It can help to retry a couple of minutes or an hour later.

We are currently using the port 8001 to establish a server connection. If this port is blocked by your IT department, a connection cannot be established. Either you can open the port or use the offline activation.

User cannot be found
Check for spelling errors in your license key. Also, make sure you are using the license key of this year and not your previous one. If it cannot be resolved, contact support.

No licenses have been saved
This means a connection to the server has been established and a valid user has been found. But still no license could be saved. It could be that the number of licenses exceeded, the xml reading process of the license failed or the directory cannot be accessed. Please contact support to figure out the source of this error.

If you are unable to resolve the error, please make a screenshot of the error message and contact nextnano GmbH support.

15.2.7 *Licensing.dll* cannot be found

This error prevents the license check within *nextnanomat*. It is not a critical error as local simulations can still be run without any problems, hence you can choose to ignore it. (Each product performs its own license check).

However the error can normally be resolved by installing the *visual studio tools C++ runtime environment*. This is needed for most applications based on C. Download page for visual C redistributable download. Choose the corresponding version matching your operation system from the Visual Studio 15/17/19/22 download section. Note, the name is misleading, but the package is a standalone to support runtime of apps. So Visual Studio does not have to be installed!

This solution is assuming the *.NET* framework is above version 4.7. In case of uncertainty it can be looked up within the *Troubleshooting Snapshot text file* which is documented here: *Generate System Snapshot for Troubleshooting*. If *.NET* framework is below 4.7 a more recent version should be installed.

15.2.8 License could not be verified

If you encounter this error when running a simulation, we recommend to install the *visual studio tools C++ runtime environment*. Download page for visual C redistributable download. Choose the corresponding version matching your operation system from the Visual Studio 15/17/19/22 download section.

If installing the runtime environment does not resolve the error, please contact support via *nextnano Help Center*.

15.3 FAQ - nextnanomat

15.3.1 How do I produce 1D slices through the 2D plots in the GUI?

1. Visualize horizontal and vertical slices:

   Menu `View => Show Horizontal and Vertical Slices` (or click on appropriate button)

2. Export horizontal or vertical slices:

   Right-click on 2D graph, `Export horizontal/vertical slice as data (.dat)` or `as image (.png)`.

15.3.2 Is there a way to produce a 1D (or 2D) plot of some result, for example the probability density with the conduction band edge superimposed?

15.3.3 In other words, can the GUI show multiple plots at once?

Yes, this is possible. Multiple output files can be shown at once, see *Overlay feature documentation*.

15.3.4 What is the difference between "List view" and "Tree view"?

In/For the Output tab of *nextnanomat*, you can switch between these options using: `Tools ==> Options ==> View ==> Output folder browser`

`List view` is the default and shows a list of the top level output folders. `Tree view` additionally offers the possibility of showing the subfolders within the output folder panel.

If you want to sort your output folders by date, `List view` is better suited.

15.3.5 How to prevent fonts from changing back to Calibli in the input tab?

Problem

As our text editor is based on **Rich Text Box** WinForms its behavior may be sensitive to the choice of your **Keybord Language**. Certain input languages for your keyboard, e.g., **English (Germany)**, may cause the editor to write in **Calibri** font instead of your selected editor font, which you specified in **Tools/Options/Editor**.

Note: Calibri is a default font of Windows.

Your desired font is restored upon saving the input file which updates the syntax highlighting.

Solutions

If you have other language settings already installed, you can consider switching between them to see if one of them is supported. If so, then you can use this setting to write codes in the input file with *nextnanomat*. You can do it using a shortcut **Start + Space**.

If you are lacking supported language settings for your keyboard, then go to **Settings / Time & Language / Language** and **Add a language**.

Some known supported language settings for keyboards are:

- English (United States)
- English (United Kingdom)
- German (Germany)
- Polish

Known not supported language settings for keyboards are:

- English (Germany)
- Japanese

15.4 FAQ - nextnano++ and nextnano^3

15.4.1 What is the difference between nextnano^3 and nextnano++?

The short answer is: Use *nextnano++*.

This is the tool where we put on most effort in improving it by adding new features. The *nextnano^3* tool is mainly distributed for historical reasons.

The *nextnano^3* tool is written in Fortran. It has been developed at the Walter Schottky Institute from 1999 to 2010. The *nextnano++* tool is written in C++. It has been developed at the Walter Schottky Institute from 2004 to 2008. The tools have been written by different people that were working at the Walter Schottky Institute of the Technische Universität München in the Theoretical Semiconductor Physics group of Prof. Peter Vogl.

Essentially, both tools cover the same physics and methods, namely

- the strain equation
- the Poisson equation

- the Schrödinger equation
- the drift-diffusion current equation.

Additionally, *nextnano*^{3} includes

- solar cells
- electrolytes
- graphene
- tight-binding (for bulk and one-dimensional superlattices)
- the self-consistent CBR method (1D, 2D and 3D)
- the NEGF method (1D only) which is particularly suited for quantum cascade lasers.

In contrast to *nextnano++*, *nextnano*^{3} is not able to

- treat the magnetic field within the $\mathbf{k} \cdot \mathbf{p}$ approach
- calculate the g tensor
- include quaternary materials.

The *nextnano++* tool is much faster for drift-diffusion calculations and for 2D/3D simulations. For instance, if you work on LEDs, MOSFETs or Quantum Dots, *nextnano++* is much better suited.

There are some applications where it is irrelevant which tool to use. In this case we recommend to use both. This has the advantage that the results of one tool can be compared to the results of the other one in order to gain more confidence in them. For some applications, one tool should be preferred. Please contact <support[at]nextnano.com> to find out which tool to choose for your particular application.

***nextnano*^{3} syntax**

- ''$" character for the keywords: `$regions ... $end_regions`
- ''%" character for the variables: `%QuantumWellWidth = 5.0`
- ''!" character for comments: `! This is a comment.` (`#` is supported by both nextnano GmbH and *nextnano*^{3} as a comment sign.)

***nextnano++* syntax**

- ''{}" brackets for the keywords: `structure{ ... }`
- ''$" character for the variables: `$QuantumWellWidth = 5.0``
- ''#" character for comments: `# This is a comment.` (`#` is also supported by *nextnano*^{3}.)

15.4.2 Can I convert nextnano^3 input files into nextnano++ input files?

Within *nextnanomat*, there is an experimental feature to convert a *nextnano*^{3} input file to a *nextnano++* input file. How to use this automatic conversion:

In the menu select `Tools ==> Convert nextnano`^{3} `input file to nextnano++`

When you use the automatic conversion of *nextnano*^{3} input file into *nextnano++*, you will find that it probably does not work completely.

If you save and run the *nextnano++* input file that has been converted and that has the suffix `_nnp.in`, very likely some errors appear indicating which line(s) to change. Then some manual adjustments are needed, but the rough structure should help a lot for the conversion.

15.4.3 How can I track how much memory is used during the simulations?

`Tools ==> Options ==> Expert settings ==> Include memory usage in log file (units: MB)`

15.4.4 Can I pass additional command line arguments to the executable?

Yes, this is possible. Go to `Tools ==> Options ==> Expert settings ==> Command line`

For *nextnano*[3], one could use for instance: `-database "D:\My folder\nextnano3\Syntax\my_database_nn3.in`

`-threads 4`

15.4.5 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 15.4.5.1). If one is using a coarse grid of only **10 nm**, the calculated values (squares in Figure 15.4.5.1) agree very well with the calculated values of the thin lines.

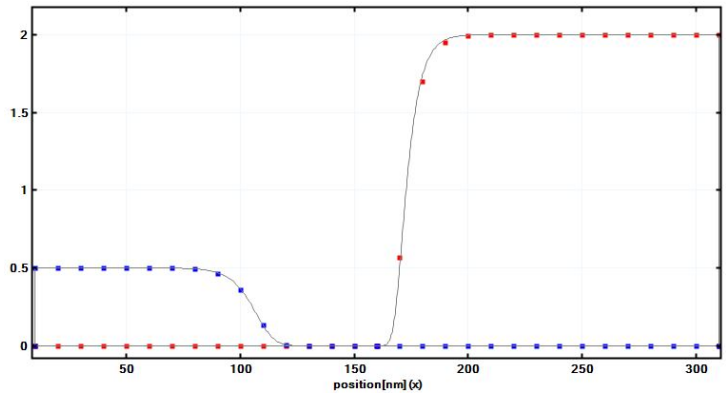


Figure 15.4.5.1: Hole (blue) and electron (red) densities of the p-n junction in units of $10^{18} cm^{-3}$. The gray lines are from simulations using a 1 nm grid spacing. The squares are from a simulation that uses only a 10 nm grid resolution. Note that the center coordinate of this plot is x=160 nm. 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 $O\left(N^2\right)$.

15.4.6 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.
- *nextnano³* - uses MKL (parallel version) The executables that are compiled with the Intel and NAG (64-bit) compilers use MKL (parallel version). The executables that are compiled with the GNU compiler (gfortran) and NAG (32-bit) use the nonparallelized version of the BLAS and LAPACK source codes available from netlib webpage. There is a *nextnano³* executable available that uses OpenMP parallelization for
  - 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 GmbH simulations are running in parallel on a quad-core CPU, i.e. 4 nextnano GmbH 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 GmbH executable compiled with the serial version of the Intel MKL could be faster.

In fact, it strongly depends on your nextnano GmbH 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 GmbH 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.

15.4.7 Dirichlet vs. Neumann boundary conditions

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*: $\frac{\mathrm{d}\psi}{\mathrm{d}x} =$ 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.

15.4.8 Quasi-Fermi level

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.

15.4.9 I don't 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.

Specific implementation *nextnano++*

See section `quantum{ region{ kp_8band{} } }` in *Multi-band models in quantum{ region{ } }*.

Specific implementation *nextnano³*

- See section *Choice of* $\mathbf{k} \cdot \mathbf{p}$ *parameters* in $numeric-control.
- See section $\mathbf{k} \cdot \mathbf{p}$ *parameters* in Which material parameters are used?.
- See section *Luttinger-parameters* in $binary-zb-default.

15.4.10 Can I add new materials to the database?

Sure!

Option 1

The material parameters are contained in ASCII text files. You can find them in the installation folder:

nextnano³ tool: `C:\Program Files (x86)\nextnano\<date>\nextnano++\Syntax\database_nnp.in`

nextnano++ tool: `C:\Program Files (x86)\nextnano\<date>\nextnano3\Syntax\database_nn3.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 and then you rename ''GaSb" to something like ''GaSb_test". Finally, you adjust the necessary material parameters that you need. In most cases, you don't 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++* (or *nextnano³*) database specifying the location within the Tools Options of *nextnanomat*.

`Tools => Options... => Material database => nextnano++/nextnano³ database file:`

Option 2

A quicker way is the following. You can overwrite certain material parameters in the input file rather than entirely defining new materials. For instance if you need ''HfO2", you could 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" a modified static dielectric constant and band edges.

Please note that we treat all materials to be either of the crystal structure

- zinc blende (including diamond type) or
- wurtzite.

Attention: More information on how to add materials

- to *nextnano++* can be found in *Material Database*
- to *nextnano³* can be found on How to add material parameters.

15.4.11 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.

15.5 FAQ - Errors and Exceptions

15.5.1 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 *How to add additional debug information to the LOG file*.

15.5.2 How to add additional debug information to the LOG file

Within the *nextnanomat* options *Options: Expert settings* some flags can be found which add additional information to the log file. Please check the following flags and re-run the simulation so the log file will include this information:

1. 'Include simulation time in log file'
2. 'Include memory usage in log file (units: MB)'
3. 'Include additional debug information in log file'

15.5.3 ERROR when loading input file "File format is not valid"

A beta feature in a recent Windows 10 version (1803) causes the following error message when starting *nextnanomat*:

You can disable this feature in the **Region's settings** of Windows (screenshot see Figure 15.5.3.2).

1. Select `Additional date, time, & regional setting`
2. Then `Change date, time, or number formats`
3. Go the the `Administrative Tab`
4. Click `Change system locale...`

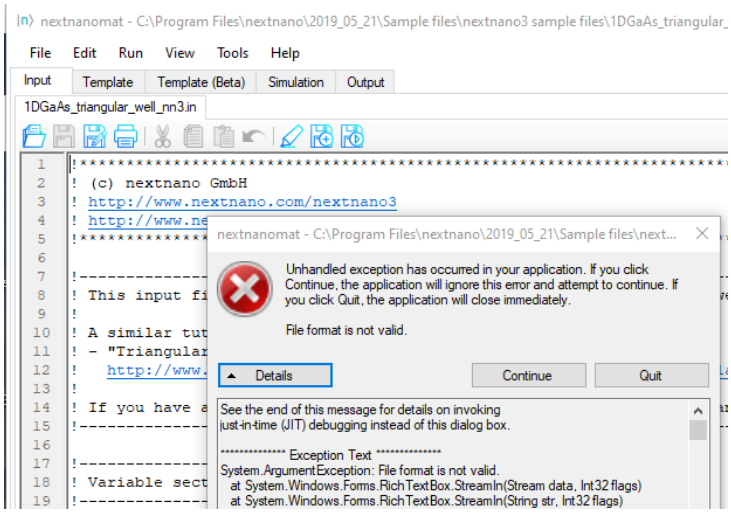


Figure 15.5.3.1: Input file format exception.

5. And finally you can un-select that Beta-Feature.

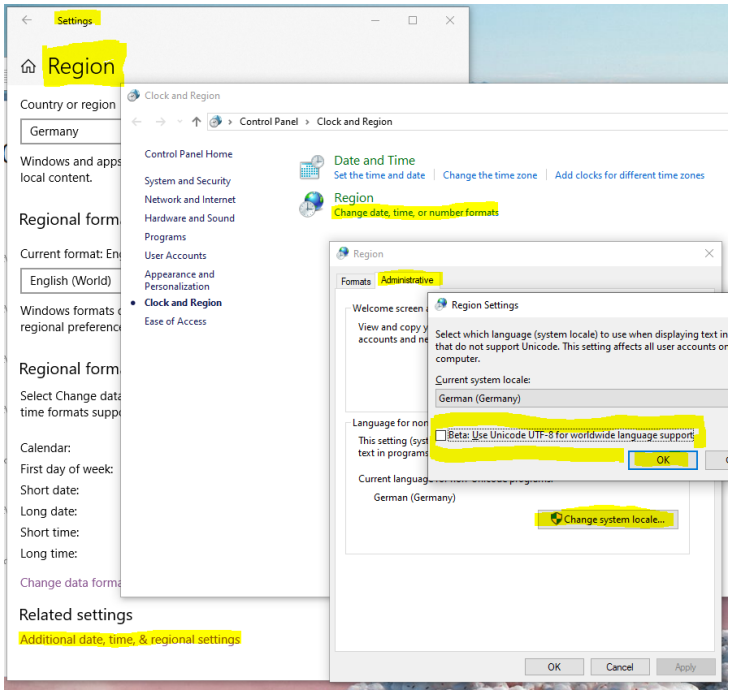


Figure 15.5.3.2: How to change Windows 10 region settings.

15.5.4 Error while starting simulation ("The specified executable is not a valid application for this OS platform.")

If you get this error, you are executing an executable that is not suited for this operating system. There are two possible reasons:

- You are executing a 64-bit version on a 32-bit operating system. Solution: Specify the location of the 32-bit *nextnano++* or *nextnano*[3] executables. Tools ==> Options... ==> Simulation ==> nextnano GmbH executable file ==> `32bit.exe`
- You are executing the Intel or Microsoft 64-bit executable that has not been compiled for your rather old CPU. Our 64-bit executables use more advanced CPU instructions compared to our

32-bit executables. Solution: Use the 32-bit *nextnano++* executable that has been compiled with the Microsoft compiler. Tools ==> Options... ==> Simulation ==> *nextnano++* executable file ==> `nextnano++_Microsoft_32bit.exe`

Sometimes the displayed error message does not contain any content apart from `ERROR`.

You can share information about your CPU with us using the menu `Help ==> Generate System Snapshot for Support` and then send us the generated file `nextnano_SystemSnapshot.txt`.

15.5.5 No Dirichlet points for Fermi levels found

Question

I get the error message:

```
ERROR: No dirichlet points for Fermi levels were found!
Terminating program !!
```

Comment: Meanwhile, this error message is different. Once an update has been released, change the FAQ here.

Answer

The program does not know at which energy the Fermi level should be located. A solution is this:

```
currents{
region{
    everywhere{}
    binary { name = "GaAs"  }
    contact{ name = "dummy" }
}
contacts{
    fermi{ name = "dummy" bias = 0.0 }
    # only needed to define reference energy
}
```

This defines the Fermi level to be at 0 `[eV]` in the whole device.

15.5.6 Quantum-Current-Poisson fails to converge

Question

My Quantum-Current-Poisson calculation fails to converge. How can I avoid this problem?

Answer

The most common errors are the following:

a) The maximum number of iterations has been chosen too small (*run{ }* ==> `outer_iteration{}` ==> `iterations`) and the calculation just needs some more iterations.

b) The number of electron or hole eigenstates has been chosen too small (e.g. *quantum{ }* ==> `region{}` ==> `Gamma{}` ==> `num_ev`).

Check the occupations in the files `wf_spectrum_"name quantum region"_"name quantum model".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 (= more possibilities to fill in charges).

c) The under-relaxation parameter has been chosen too large (*run{ }* ==> `outer_iteration{}` ==> `alpha_current`).

Check whether the solution oscillates (residuals do not drop continuously but increase in some iterations). Try to decrease the under-relaxation parameter in order to damp the oscillations.

d) The minimum charge density in the current equation has been chosen too small (*currents{ }* ==> `minimum_density`).

Try to increase the minimum charge density to improve conditioning of the current equation. In fact, the current equation is solved with the charge density = max(*real density*, `minimum_density`).

15.5.7 Import error - Nodes

If the simulation is terminated at the very beginning, when the files are imported, with the following message

```
error:(nodes number of coordinate 1) != (lines number in 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.

CHAPTER

SIXTEEN

BOOKS

- [Adachi2009]
  Proerties of Semiconductor Alloys: Group-IV, III-V and II-VI Semiconductors
  S. Adachi
  John Wiley & Sons, Inc., New York (2009)
- [ChuangOpto1995]
  Physics of Optoelectronic Devices
  S. L. Chuang
  John Wiley & Sons, Inc., New York (1995)
- [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, 2nd ed., John Wiley & Sons, Ltd (2005)
  DOI 10.1017/CBO9780511805776
- [NelsonPSC2003]
  The Physics of Solar Cells
  J. Nelson
  World Scientific Publishing (2003)
- [Landau Lifshitz]

L. D. Landau, E. M. Lifshitz,
Quantum Mechanics Non-Relativistic Theory
(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

SEVENTEEN

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

EIGHTEEN

JOURNAL PAPERS

- [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, ED-29, 292, (1982)
- [BahderPRB1990]
  Eight-band kp model of strained zinc blende crystals
  Thomas B. Bahder
  Phys. Rev. B, **41**, 11992 (1990)
- [Bai2010]
  Mid-infrared quantum cascade lasers operating in continuous wave above 300 K
  Y. Bai, S. Slivken, S. R. Darvish, W. Zhang, A. Evans, J. Nguyen, and M. Razeghi
  Nature Photonics **4**, 99 (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)
- [Beynon1996]
  Buffer solutions: The basics
  R. J. Beynon, and J. S. Easterby
  Oxford University Press (1996)
- [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 Transactions on Electron Devices, vol. ED-22, no. 11, pp. 1045-1047 (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 Vol. 142, No. 2, pp. 530-543 (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, vol. 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)
- [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, 383-389 (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 TRANSACTIONS ON ELECTRON DEVICES, VOL. 48, NO. 3, MARCH 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), 173-182 (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)
- [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)

- [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, Vol. 27, No. 5, pp. 519-524
- [KnoetigLaserPhotRev2022]
  Mitigating valence intersubband absorption in interband cascade lasers <https://doi.org/10.1002/lpor.202200156>
  H. Knoetig, J. Nauschuetz, N. Opacak, S. Hoefling, J. Koeth, R. Weih, B. Schwarz
  Laser Photonics Rev., 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.
- [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).
- [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)
- [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, pp. 3891-3894 (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 Vol. 74, No. 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, Vol. 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)
- [VurgaftmanNatComm2011]
  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, Vol. 44, No. 5, pp. 365-378
- [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

NINETEEN

OLD SITES

The sites on this list will be deleted in the near future.

19.1 Downloads

The new download site has been moved *here*.

19.2 Release Notes

Attention: This page is going to be closed soon. All subpages has been already moved.

19.2.1 nextnanomat — release notes

This page has been moved *here*

19.2.2 Release notes of nextnano++

This page has been moved *here*

19.2.3 Release notes of nextnano^3

This page has been moved *here*

19.2.4 nextnano.NEGF — release notes

This page has been moved *here*

19.2.5 nextnano.NEGF_classic — release notes

This page has been moved *here*

Release notes for **nextnanopy** can be found on our GitHub site.

19.3 Applications

This page will be closed soon, please visit *Tutorials* instead.

19.4 run{ }

New page about the `run{ }` keyword can be found *here*.

19.5 strain{ }

The new page can be found *strain{ }*

19.6 currents{}

The new site can be found here: *currents{ }*.

19.7 mobility_model

This keyword has been deprecated. See *electron_mobility{ }* and *hole_mobility{ }* for new keywords.

19.8 structure{ } - outputs

This site has been moved to:

- *output_region_index{ }*
- *output_material_index{ }*
- *output_user_index{ }*
- *output_contact_index{ }*
- *output_alloy_composition{ }*
- *output_impurities{ }*
- *output_generation{ }*
- *output_injection{ }*

19.9 structure{ region{} } - Repeating regions

This site has been moved to:

- *array_x{ }*
- *array_y{ }*
- *array_z{ }*
- *array2_x{ }*
- *array2_y{ }*

- *array2_z{ }*
- *repeat_profiles*

19.10 structure{ region{} } - boundary conditions

This page has been moved to *contact{ }*

19.11 structure{ region{ doping{ } } }

This page has been moved to *doping{ }*