TraNaS OpenSuite Guide

version 1.02 (release) January 18, 2019

open software for
quantum nanoscale modeling

DFTB$^+$ eXTended version for
model and atomistic quantum transport at nanoscale
many-body nonequilibrium phenomena
material & device modeling

tranas.org/opensuite

detailed description of input and output is given in the DFTB$^+$XT USER MANUAL

this guide focuses on quantum transport
please go to dftbplus.org for additional DFTB$^+$ documentation

Developers and contribution

Developers

Main developers of DFTB$^+$XT

- Dmitry A. Ryndyk (University of Bremen and TU Dresden, Germany)

Main developers of DFTB$^+$

- Bálint Aradi (University of Bremen, Germany)
- Ben Hourahine (University of Strathclyde, UK)

Main developers of the transport part of DFTB$^+$ (DFTB+NEGF)

- Alessandro Pecchia (University of Rome "Tor Vergata", Italy)
- Gabriele Penazzi (formerly University of Bremen Germany (till 2016), now QuantumWise A/S, Denmark)

Full list of authors is in the *AUTHORS.rst* file.

Contribution to this Guide

The Guide is prepared by **D.A. Ryndyk**.

The Chapter 1 “Transport calculations (introduction)” is partially derived from the **“DFTB+NEGF recipes” © B. Aradi and A. Pecchia**.

The “Tutorial on 2D Carbon Materials” (Part II) is derived from the **“DFTB+ Tutorial on 2D Carbon Materials, Release 2014.11” © B. Aradi, G. Penazzi and B. Hourahine**.

Contents

Developers and contribution **i**

I Overveiw and recipes 1

1 Transport calculations (introduction) **3**
- 1.1 Specifying the geometry . . . 3
  - 1.1.1 Partitioning and atom numbering for a system with 2 electrodes . . . 3
  - 1.1.2 Supercell structures . . . 4
  - 1.1.3 Partitioning the system with N electrodes . . . 5
  - 1.1.4 Geometry input block . . . 5
  - 1.1.5 Transport input block . . . 5
- 1.2 Specifying the Hamiltonian . . . 6
  - 1.2.1 Hamiltonian input block . . . 6
  - 1.2.2 Green function solver . . . 7
  - 1.2.3 Poisson solver options . . . 8
  - 1.2.4 Model calculations without geometry . . . 10
- 1.3 Precalculation of electrodes . . . 11
- 1.4 Calculation of transmission and density of states . . . 11
  - 1.4.1 Analysis input block . . . 11

II Tutorial on 2D Carbon Materials 13

2 Introduction **15**
- 2.1 System environment . . . 15
- 2.2 General notes . . . 16
- 2.3 Creating of the directory with inputs . . . 16

3 Electronic structure of 2D carbon materials **17**
- 3.1 Perfect graphene . . . 17
  - 3.1.1 Geometry, density of state . . . 17
  - 3.1.2 Band structure . . . 21
- 3.2 Zigzag nanoribbon . . . 23
  - 3.2.1 Calculting the density and DOS . . . 23
  - 3.2.2 Band structure . . . 25
- 3.3 Armchair nanoribbon with defects . . . 26

3.3.1 Perfect armchair nanoribbon 27
3.3.2 Armchair nanoribbon with vacancy 28

4 Electron transport calculations in armchair nanoribbons **33**
4.1 Non-SCC Pristine armchair nanoribbon 33
4.1.1 Preparing the structure 33
4.1.2 Transmission and density of states 37
4.2 Non-SCC armchair nanoribbon with vacancy (A) 41
4.2.1 Transmission and Density of States 41
4.3 Non-SCC armchair nanoribbon with vacancy (B) 45
4.3.1 Transmission and Density of States 45
4.4 SCC Pristine armchair nanoribbon 47
4.4.1 Contact calculation 47
4.4.2 Transmission and Density of States 49
4.5 SCC armchair nanoribbon with vacancy (A) 54
4.5.1 Transmission and Density of States 55
4.6 SCC armchair nanoribbon with vacancy (B) 58

III Examples **61**

IV Developer guide **63**

5 TraNaS library **65**
5.1 TTraNaS type 65

V Appendices **67**

License **69**

Bibliography **71**

Part I

Overveiw and recipes

CHAPTER 1

Transport calculations (introduction)

In this chapter one can find a brief overview of the DFTB$^+$XT features for electronic structure and quantum transport calculations. The detailed description of input and output is given in the USER MANUAL. The step-by-step calculation of the electronic structure and basic transport properties can be done using the Tutorial on 2D Carbon Materials (Part II).

The following blocks must present in the input file for transport calculations: Geometry, Transport, Hamiltonian, Analysis. We discuss below the main ideas of the input and also some additional python scripts for different types of calculations. This overview is an introduction and does not describe all input keywords, which are listed and explained in the USER MANUAL.

1.1 Specifying the geometry

1.1.1 Partitioning and atom numbering for a system with 2 electrodes

In the simplest case, transport calculations can be performed on an atomistic structure comprising two semi-infinite electrodes. The electrodes are usually periodic systems being terminated on one end by the central region (extended molecule, device). In order to carry out a transport calculation with DFTB$^+$XT, the system must be carefully partitioned by the user and the structure must contain (Fig. 1.1):

- The extended molecule (central region, device);
- Two Principal Layers (see below) of the first contact;
- Two Principal Layers of the second contact.

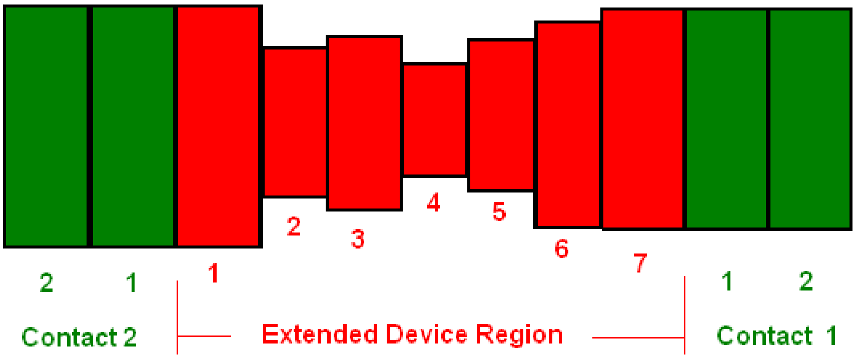

Fig. 1.1: Subdivision of a 2-terminal structure into principal layers (PL), two for each contact and an arbitrary number within the extended molecule.

The extended molecule contains the atomistic device itself plus those parts of the attached electrodes, which are directly influenced by the presence of the device. Each electrode, on the other hand, contains those parts of the physical electrode, which are far enough from the device and are not affected by it. Examples below will better clarify this point.

The most important concept to bear in mind is that of principal layers (PLs). These are defined as contiguous groups of atoms that have interaction only with atoms of adjacent PLs. In practice these layers must contain a sufficient number of atoms in order to ensure that Hamiltonian and overlap interactions with second neighbouring PLs vanish or can be considered negligible. The subdivision into PLs is essential for the definition of the two contacts and becomes useful in the computation of all Green functions that exploit a recursive algorithm [1].

As shown in the figure 1.1, the extended molecule can contain an arbitrary number of PLs ($\geqslant 1$), but the layers must follow a sequential ordering.

The ordering of the PLs follows directly from the ordering of the atoms in the DFTB$^+$ structure.[1]

Typically it is convenient to create the structure and then sort the atom along the transport coordinate, before partitioning the system. In other cases, when nanowires are constructed it is convenient to repeat a PL unit for the desired length of the extended molecule.

The (perfect) contacts must be defined by two principal layers. Differently from the layers of the extended molecule, those defining the contacts must follow additional rules:

- The two principal layers of a given contact must be identical;
- They must be rigidly shifted images of each other;
- The first PL must be the closest to the device region.

In order to ensure the above prescriptions, the numbering of the atoms must follow a precise ordering.

The atoms of the central region must be specified first, before the atoms of the contacts (see example below in Fig. 1.2). The atoms of the contacts follow the central region. The atoms of each contact must be specified continously. (You specify either all atoms of the left contact first, and then those of the right one, or vice versa.)

The numbering inside the first principal layer defining each contact is arbitrary, but the same numbering must be applied to the second PL. Thus, the indices of the corresponding atoms in the two contact PLs must differ by the same value, and the coordinates must differ by the same translation vector. These prescriptions are checked by the code and an error message is issued if the two PLs do not conform. It is important to remember to place the first PL closer to the device, i.e. the principal layer closer to the device must contain the atoms with lower indices.

1.1.2 Supercell structures

The code can compute transport on structures that have a periodicity in the transverse directions (with respect to transport). In this case the structure must be defined as a supercell and the rules listed above apply to each cell. The real-space Poisson solver of DFTB$^+$ limits the supercell lattices to orthorombic types (all angles between supercell vectors must be 90 degrees). In fact the supercell is always defined 3-dimensional, and the user should ensure that the dummy lattice vector along the transport direction is long enough to avoid superpositions between images.

---

[1] Note, however, the case of model calculations without geometry Sec. 1.2.4.

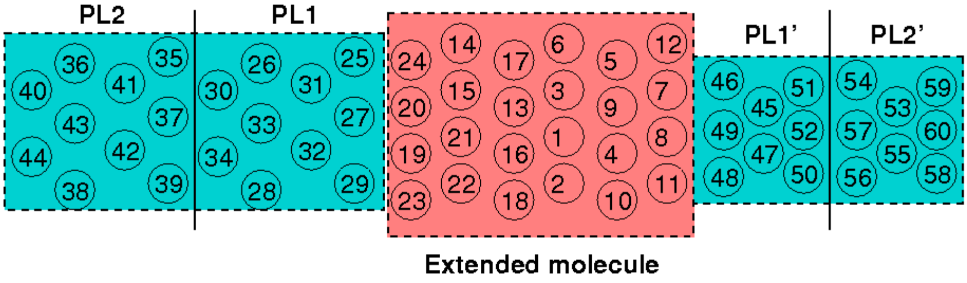
PL2
PL1
PL1'
PL2'
Extended molecule
36 35 40 41 37 43 44 42 38 39
26 25 30 31 27 33 34 32 28 29
24 14 17 6 5 12 20 15 13 3 9 7 19 21 16 1 4 8 23 22 18 2 10 11
46 51 45 49 52 47 48 50
54 59 53 57 60 55 56 58

Fig. 1.2: Example for numbering the atoms. Atoms of the device have the lowest indices, followed by the atoms of each contact, respectively. The two principal layers of both contacts are shifted images of each other.

The current version of DFTB$^+$ only supports one supercell definition for the entire system, including central region and contacts. It may be redundant to observe that in this case the two contacts must be of the same periodicity.

1.1.3 Partitioning the system with N electrodes

DFTB$^+$XT allows also multi-terminal calculations. The rules for the atomic sequence are the same: first must be the device atoms, then the electrodes represented by two PLs each, and the numbers of the PL close to the device should be smaller than the numbers of the second PL.

In the case when the device region consists of many PLs, it is important that every electrode is connected to only one of PLs, but any number can be connected to one PL. The trivial example is the the case with only one PL in the central region and two electrodes, in this case both electrodes are connected to the same PL. This is illustrated in Fig. 1.3.

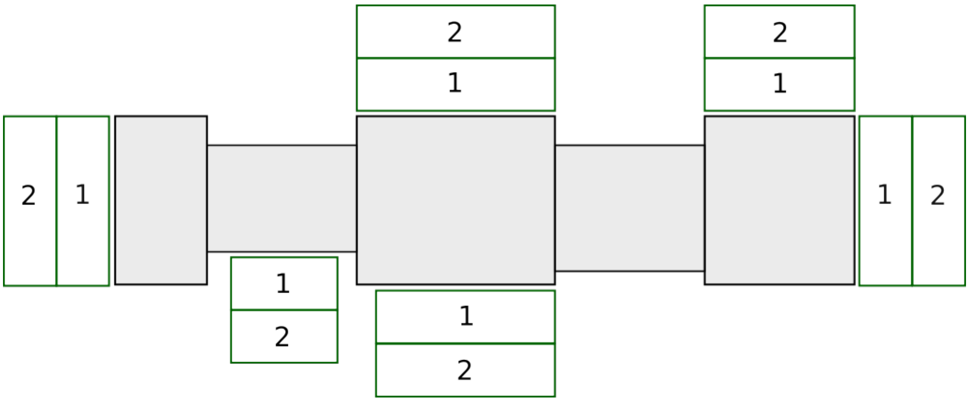
2
1
2
1
2
1
1
2
1
2
1
2

Fig. 1.3: Subdivision of a N-terminal structure into principal layers (PL), two for each contact and an arbitrary number within the extended molecule.

1.1.4 Geometry input block

The atomic structure is defined in the Geometry block (see the USER MANUAL for the description). The example of geometry definition is given in the tutorial, see from Sec. 3.1.1.

1.1.5 Transport input block

The geometry must be additionally defined in the input Transport block as specified in the following example:

```
Transport {
  Device {
    AtomRange = 1 8
  }
  Contact {
    Id = "source"
    AtomRange = 9 24
    FermiLevel [eV] = -8.4123
    Potential = 0.0
  }
  Contact {
    Id = "drain"
    AtomRange = 25 40
    FermiLevel [eV] = -8.4123
    Potential = 1.0
  }
  Task = UploadContacts
}
```

The Fermi level should be precalculated in the electrodes as discussed in Sec. 1.3 and in the tutorial in more detail. There are also many other parameters of the electrodes.

1.2 Specifying the Hamiltonian

1.2.1 Hamiltonian input block

The Hamiltonian block for Non-SCC DFTB calculations looks like

```
Hamiltonian = DFTB {
  SCC = No
  MaxAngularMomentum = {
    C = "p"
    H = "s"
  }
  SlaterKosterFiles = Type2FileNames {
    Prefix = "../../../sk/"
    Separator = "-"
    Suffix = ".skf"
  }
  Eigensolver = TransportOnly{}
}
```

Note that transport calculation can be started immediately in this case, but precalculations of electrodes may be done to determine the Fermi levels.

In the case of SCC DFTB calculation, however, the self-consistent density of electrons should be calculated at zero or finite voltage before the calculation of the transmission or density of states. The Hamiltonian block should include new options, the main are

```
Hamiltonian = DFTB {
  SCC = Yes
  SCCTolerance = 1e-6
  ReadInitialCharges = No
  Electrostatics = Poisson {
    Poissonbox [Angstrom] = 40.0 30.0 30.0
    MinimalGrid [Angstrom] = 0.5 0.5 0.5
    SavePotential = Yes
  }
  Eigensolver = GreensFunction{}
  Mixer = Broyden{}
    MixingParameter = 0.02
  }
  ...
}
```

Note change of the Eigensolver method and new Electrostatics subblock, required for a self-consistent calculation of electron densities and electric field.

1.2.2 Green function solver

If you calculate electron transport properties using the Green function formalism, you need to determine the electron density in your system with respect to certain electrostatic boundary conditions for your system. For that, you must use the GreensFunction{} as eigensolver, which is capable to deliver the appropriate density. Please note, that this technique does not deliver any eigenvalues, it does not solve the eigenproblem (no eigenvectors are computed).

You can instruct the code to use the Green function solver by setting Eigensolver in the block Hamiltonian = DFTB{} to GreensFunction{}:

```
Hamiltonian = DFTB {
  :
  Eigensolver = GreensFunction{}
  :
}
```

Let us discuss the most important parameters to be set in GreensFunction{}:

```
Eigensolver = GreensFunction {
  FirstLayerAtoms =  1   61   121   181   241   301   361   421   481   541
  Delta [eV] = 1e-4
  ContourPoints = 20 20
  LowestEnergy [eV] = -60.0
  EnclosedPoles = 0
}
```

FirstLayerAtoms is used to specify the PLs in the central region in the same way as described in Device subblock. Can be specified only if no Transport block exists. As the name of the keyword suggests the layers are defined by specifying the first atom of each layer. In this case 10 PLs have been defined. Note that the contacts are not included here, as they are specified in the Transport block.

Delta defines a small positive imaginary number used in the computation of the GFs.

ContourPoints is used to specify the number of quadrature points in the contour integration (see also the USER MANUAL for a description of the complex contour).

LowestEnergy is the initial energy from which the integration starts. It should be low enough to ensure that all the electronic states are correctly included in the integration. The default is -2.0 Hartree.

EnclosedPoles is set to 0 when T=0. For T>0 few poles (usually 3) needs to be included within the contour.

1.2.3 Poisson solver options

For the transport calculation the GammaFunctional solver (default) is substituted with the Poisson solver. The Poisson{} method is used to define the size of the Poisson domain. The Poisson equation is solved via a real-space multigrid solver that employs finite-differences for discretization on a finite box with a regular grid (structured mesh). The charge density on the right-hand-side is constructed exactly as in standard gamma-functional of DFTB, namely expanding the charge density into spherical s-like atomic densities weighted by atomic Mulliken charges.

```
Electrostatics = Poisson {
  Poissonbox [Angstrom] = 30.0 30.0 30.0
  MinimalGrid [Angstrom] = 0.4 0.4 0.4
  AtomDensityTolerance = 1e-6
  BuildBulkPotential = Yes
  SavePotential = Yes
  PoissonAccuracy = 1e-7
}
```

The method Poisson{} is used to define the size of the Poisson domain. The Poisson equation is solved via a real-space multigrid solver that employs finite-differences for discretization on a finite box with a regular grid (structured mesh). The charge density on the right-hand-side is constructed exactly as in standard gamma-functional of DFTB, namely expanding the charge density into spherical s-like atomic densities weighted by atomic Mulliken charges.

The equation is solved by imposing the following boundary conditions (BC):

1. Dirichelet or mixed BC on the faces containing contacts.
2. Neumann BC on the remaining faces.

In the example above a box of 30x30 Angstrom in the x- and y- directions (orthogonal to the transport direction z) have been chosen. This size should be sufficient in the current case for obtaining a converged result. A box too small may introduce artificial size effects. The specification of the box length along the z-direction is a dummy number ignored by the code as the box-length in this direction is constrained by the position of the contacts and is internally adjusted.

The meaning of the additional options are the following:

MinimalGrid = 0.4 0.4 0.4 is used to specify that the grid must be spaced less than 0.4 in all dimensions. The actual grid is adjusted internally since the number of grid points in every direction must be a power of 2 (exactly $N = 2^n + 1$).

AtomDensityTolerance = 1e-6 is used to specify, where the exponential decaying spherical s-like atomic charge densities should be cut off. Specifying a certain value here makes sure that all atoms contributing a density higher than the given value are considered when calculating the amount of charges in a certain point (default: 1e-5). The appropriate cutoff radius is calculated automatically by the code (and reported in the output). It is determined by finding the cutoff distance for each atom (α), where the s-like charge density

$$n_\alpha(r) = \frac{\tau_\alpha^3}{8\pi} e^{-\tau_\alpha r}$$

becomes smaller than the given tolerance. Then the maximal cutoff found for all atom types in the system is used. The quantity $\tau_\alpha = \frac{16}{5} U_\alpha$ is the relationship between the extintion coefficient and the Hubbard parameter in atomic units. See Ref. [2].

In order to have a consistent calculation, the determined cutoff length must be smaller than the width of the principal layers when doing a calculation with contacts. The program will check for this criterion and stop if it is not fulfilled. In this example we set it exactly to the default value, only for clarification purpose.

CutoffCheck If set to No, the code omits the check whether the cutoff for the atomic densities (either determined by AtomDensityTolerance or directly set by AtomDensityCutoff) is larger than the widths of the principal layers in the contacts. Please note that a cutoff bigger than the width of any contact PL results in inconsistent calculation, so it is highly discouraged to turn this check off, unless you exactly know what you are doing. (for experts only)

BuildBulkPotential = Yes is used to specify that at the device/contact interfaces the bulk potential must be imposed as BC. The bulk potential is computed for an ideal contact (infinite wire). Here we should remind that a key assumption in transport calculations is that the contacts are in equilibrium and that the device/contact interfaces are sufficiently deep inside such that bulk conditions are recovered. This means that the charge density and potential at this interface should smoothly join with the bulk values. Setting this flag to Yes is important whenever there is a charge redistribution within the contact atoms that has an effect on the bulk potential, like in structures of heteronuclear species (e.g. SiC, GaAs, ZnO, etc.). In the case of a SiNW it is important because of the charge redistribution between silicon and hydrogen atoms.

The bulk potential is computed by solving a Poisson problem for the contact PLs using a box with the same size and grid as specified for the central region along the x and y directions. In this way the result of the calculation is imposed on the device box without the need of interpolations.

The problem is solved by imposing periodic boundary conditions (PBC) along the z-direction and Dirichelet BC on the other four sides (V=0). Note that this is not exactly consistent with the contact calculation performed on the first step, in which it was assumed a periodic structure with a large contact separation (1000 a.u.). However the two calculations produce approximately the same result. In principles the bulk potential could be computed exactly in the same way, by imposing a supercell with a large periodicity on the (x, y) directions. The problem here is of technical nature because it is impractical to solve the Poisson equation on such a huge box (it would require a box with at least 1025x1025x257 grid points and a memory consumption of about 6.5 Gb). Secondly the Poisson equation generates a singular matrix when PBC are imposed on all sides of the box. A possibility (which is automatically switched on in supercells) is to compute the bulk potentials with an Ewald summation technique, but turns out to be quite time-consuming.

In order to achieve better consistency with the contact calculations it is possible to compute the contact Hamiltonians using the Poisson solver rather than the usual GammaFunctional. This option can be set by specifying:

Electrostatics = Poisson

DFTB$^+$ recognizes if 1D wires are computed and imposes in this special case the same BC as in the bulk calculation, namely PBC along z and Dirichelet on the other four sides. This procedure usually makes the equilibrium transmission function closer to the ideal conductance steps, since contact and device Hamiltonians are computed on an equal footing.

1.2.4 Model calculations without geometry

Starting from the version DFTB^{+}XT 1.01 the calculations with external model tight-binding Hamiltonians are possible. In particular "without geometry", it means that the coordinates are not used and are not required. It must be announced in the input file as

```
Geometry = NoGeometry{}
```

In the Transport block it is only one important addition: ContactPLs describes the indices of PLs to which the electrodes are coupled:

```
Transport {
  Device {
    AtomRange = 1 37
    FirstLayerAtoms =  1 9 19
    ContactPLs = 1 3
  }
}
```

Otherwise, it is described the geometry in the same way as before with the same assumptions about the order of tight-binding states (which are assumed to be "atoms" now). One atom corresponds to one electronic state. As a life hack, one can use the geometry from only hydrogen atoms (with only one electronic state) to create the real space tight-binding geometry, for example to include the self-consistent electric fields.

The Analysis block is also unchanged.

Most important is the change in the Hamiltonian block. It looks like

```
Hamiltonian = Model {
  NumStates = 44
  HamiltonianMatrix [eV] = {<<< "filename"}
  SpinDegeneracy = Yes
}
```

The new method is Model{}, it includes the required parameter NumStates, which gives the full number of states including the central region and the electrodes. The HamiltonianMatrix can be used to change the Hamiltonian and the energy units. The default filename is H.mtr. The Hamiltonian in the file should be saved as an array of real numbers. Or it can be inserted explicitly instead of <<< "filename". Finally, SpinDegeneracy = Yes says that it can be two electrons in every state and the same Hamiltonian is assumed for spin-up and spin-down (in the simplest case just to multiply the DOS and transmission by 2).

The Hamiltonian should be ordered in the right way to be consistent with the ordering of sites (atoms) in the Transport block! First the central region, then electrodes one by one, every from two blocks. For two electrodes it looks like

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}_C & \boldsymbol{V}_{CL} & 0 & \boldsymbol{V}_{CR} & 0 \\ \boldsymbol{V}_{CL}^{\dagger} & \boldsymbol{H}_{L0} & \boldsymbol{H}_{L1} & 0 & 0 \\ 0 & \boldsymbol{H}_{L1}^{\dagger} & \boldsymbol{H}_{L0} & 0 & 0 \\ \boldsymbol{V}_{CR}^{\dagger} & 0 & 0 & \boldsymbol{H}_{R0} & \boldsymbol{H}_{R1} \\ 0 & 0 & 0 & \boldsymbol{H}_{R1}^{\dagger} & \boldsymbol{H}_{R0} \end{pmatrix} \tag{1.1}$$

At the moment users should care about the right ordering.

1.3 Precalculation of electrodes

A transport calculation requires the execution of preliminary tasks for the computation of the contact Hamiltonians. The contacts are assumed ideal and having the properties of a 'bulk' material. For this reason a special calculation must be performed in advance and results are stored on files for subsequent uploading.

The Transport block contains a Task block, which describes the action to be carried out. The following tasks can be executed:

`Task = UploadContacts{}`

for transport calculations, or

`Task = ContactHamiltonian{}`

to calculate the Hamiltonian for a given contact. The name of the contact must be specified via the Id tag. Only one contact Hamiltonian can be calculated at one go, so that you have to run DFTB$^+$ for every contact separately.

The example below demonstrates the contact calculation for the contact called "source":

```
Transport {
  Device {
    AtomRange = 1 24
  }
  Contact {
    Id = "source"
    AtomRange = 25 44
  }
  Contact {
    Id = "drain"
    AtomRange = 45 58
  }
  Task = ContactHamiltonian {
    ContactId = "source"
  }
}
```

1.4 Calculation of transmission and density of states

Using the electrode Hamiltonians produced with Task = ContactHamiltonian{}, the surface Green functions for the contacts can be calculated, and open boundary conditions are applied. To calculate the electronic transport through the device, defined in Geometry, Transport and Hamiltonian blocks, one should define the additional parameters in the Analysis block.

1.4.1 Analysis input block

In the simplest case it looks like

```
Analysis{
  TransmissionAndDOS{
    EnergyRange [eV] = -5.0 -3.0
    EnergyStep [eV] = 0.02
    Region = {
      Atoms = 1:37
    }
  }
}
```

EnergyRange and EnergyStep determine the energy range over which the transmission function and local density of states are computed and the energy step.

Region defines atomic ranges or orbitals where the local density of states is calculated projected. The definition in the block follow the same syntax as a DFTB$^+$ calculation without transport.

See the USER MANUAL for details. The examples of transport calculations are given in the tutorial, see Chapter 4 Electron transport calculations in armchair nanoribbons.

Part II

Tutorial on 2D Carbon Materials

CHAPTER 2

Introduction

This tutorial is aimed at Master and PhD students with background knowledge of the theory of electronic structure and quantum transport at nanoscale. Minimal knowledge of Unix/Linux is also required. Knowledge of the DFTB+XT code itself is not necessary, but familiarity with the basic ideas behind the Density Functional Theory (DFT), the Density Functional Tight Binding (DFTB) method, the Landauer method and the Green function technique of the quantum transport theory could be helpful.

2.1 System environment

The easiest way to use this tutorial is to download the TraNaS OpenSuite LiveDVD ISO image from the tranas.org/opensuite site, which contains a preconfigured x86_64/Linux system with all the necessary tools for the tutorial. One can directly boot the ISO image within a virtual machine (e.g. *VirtualBox*). Alternatively, one can create a bootable USB-drive (e.g. with *usb-creator* under Linux or *Universal USB Installer* under Windows) or DVD, which can then be used to boot up an X86_64 machine with the live system.

In case you want to try the tutorial outside of the live system, you will need the following tools to be installed:

- *dftb+* – DFTB+XT binary (version 1.02 or higher).
- *waveplot* – Command line tool to create volumetric data files representing electron densities and wavefunctions. (included into the DFTB+XT package and is installed together with *dftb+*)
- *nano* – A simple graphical text editor. You can alternatively use your preferred editor instead.
- *gen2xyz*, *xyz2gen*, *dp_dos*, *dp_bands*, *repeatgen*, *makecube* – Various conversion scripts from the *dptools* package. (included into the DFTB+XT package, *tools/dptools/*, but should be installed separately)
- *jmol* – Graphical molecular visualisation tool. You can alternatively use any molecular visualiser able to plot volumetric data and isosurfaces.
- *buildwire* – Can be used to build a 1D geometry with the right ordering (included into the DFTB+XT package, *tools/transport/*, but should be installed separately).
- *paraview* – Software to visualise *.vtk* or *.cube* files.
- *plotxy[.py]* – Command line wrapper around the matplotlib python library (included into the DFTB+XT package, *tools/misc/*).

2.2 General notes

- The working directory of each of the tutorial examples is indicated at the beginning of its corresponding section. Please change to that directory and execute the specified commands from within that directory.
- It is impossible to describe all options accepted by DFTB$^+$XT within the tutorial. The detailed description of input and output is given in the USER MANUAL. Always make sure, that you understand the input file for DFTB$^+$ (*dftb_in.hsd*). If you find any unexplained options, consult the DFTB$^+$XT documentation page and the DFTB$^+$ documentation page for details.
- In order to save you some typing, many of the necessary commands have been already collected into small scripts. Please have a look at the content of these scripts before executing them, making sure you understand why those commands must be executed in that order to obtain the necessary results.
- The TraNaS OpenSuite LiveDVD version of the DFTB$^+$XT code is free for use and full functional. It can be used to install the standard Debian GNU/Linux operating system.

2.3 Creating of the directory with inputs

Please download the file *tutorial_input.zip* from the DFTB$^+$XT site or copy it from the *doc/dftb+/tutorial/* folder of the package. Then decompress it with:

```
unzip tutorial_input.zip
```

in some directory. The directory tree with input files and some additional files to speed up the calculations is created.

If use the TraNaS OpenSuite LiveDVD, the tree with input files is placed in */home/user/Tutorial*, where one can also find the *guide.pdf* and *manual.pdf* files.

Note, that if you first install the Debian GNU/Linux from LiveDVD to the hard disk, the tutorial materials will be still in */home/user/Tutorial* and you should use the root access to copy it to your actual home directory and change access rights. The same should be done for */home/user/bin* with executable files and, finally, *export PATH=$PATH:$HOME/bin* to make short commands available.

CHAPTER 3

Electronic structure of 2D carbon materials

Enter the directory *elect/*. All directories given in this part of the tutorial are subdirectories of the *elect/* directory.

3.1 Perfect graphene

First we will investigate some of the basic properties of the 2D graphene structure.

3.1.1 Geometry, density of state

[Working directory: *elect/graphene/latopt/*]

Preparing the input

Graphene has a hexagonal lattice with two C atoms in its primitive unit cell, which is specified in the supplied GEN-formatted geometry file. Open the file *geo.gen* in a text editor

```
nano geo.gen
```

You should see the following content:

```
2  S
C
 1 1    0.1427557522E+01    0.0000000000E-00    0.0000000000E-00
 2 1   -0.1427557522E+01    0.0000000000E-00    0.0000000000E-00
 0.0000000000E+00    0.0000000000E+00    0.0000000000E+00
 0.2141036415E+01   -0.1236340643E+01    0.0000000000E-00
 0.2141036415E+01    0.1236340643E+01    0.0000000000E-00
 0.0000000000E-00    0.0000000000E-00    0.5000000000E+02
```

The format of this GEN file is the following:

- The first line contains the number of atoms (2) and the boundary condition type (`S` for a periodic crystal).
- The second line lists all atomic elements present in the system separated by white space (`C` only in this example).
- Then a squence of lines follow, one for every atom in the system, each starting with a dummy integer (its sequential number in the structure), the type of the atom according to the list of elements in the second line of the file (`1` for carbon in this example), and finally the cartesian coordinates of the atom in angstroms.

- Since the structure is periodic, appropriate information for this boundary condition must be provided after the atomic coordinates. For a GEN file of type `S`, this is the cartesian coordinates of the origin followed by the 3 cartesian lattice vectors (one per line). DFTB$^+$ uses three dimensional periodic boundary conditions. In order to separate the graphene sheets from each other and to prevent interaction between them, the third lattice vector, which is orthogonal to the plane of graphene, has been chosen to have a length of 50 angstroms.

Before running the code, you should check, whether the specified unit cell, when repeated along the lattice vectors, indeed results in a proper graphene structure. To repeat the geometry along the first and second lattice vectors a few times (the *repeatgen* script), convert it to XYZ-format (the *gen2xyz* script) and visualize it:

```
repeatgen geo.gen 4 4 1 > geo.441.gen
gen2xyz geo.441.gen
jmol geo.441.xyz &
```

You should then see a graphene sheet displayed, similar to Figure *4x4x1 graphene supercell* (page 18).

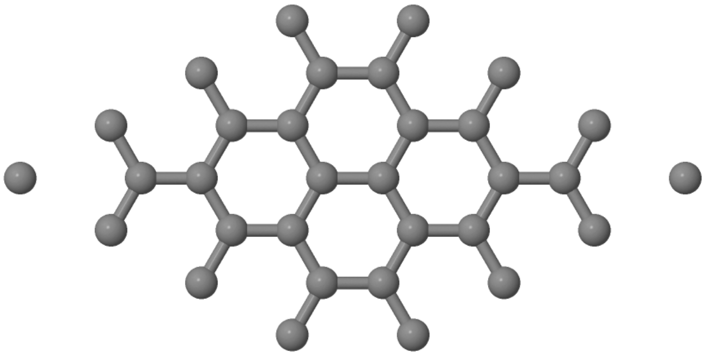

Fig. 3.1: 4x4x1 graphene supercell

Now open the DFTB$^+$ control file *dftb_in.hsd*.

```
nano dftb_in.hsd
```

You should see the following options within it:

- First we include the GEN-formatted geometry file, *geo.gen*, using the inclusion operator (<<<):

```
     Geometry = GenFormat {
        <<< "geo.gen"
     }
```

- Then we specify the `ConjugateGradient` driver to optimize the geometry and also the lattice vectors. Since neither the angle between the lattice vectors nor their relative lengths should change during optimization, we carry out an isotropic lattice optimization:

```
Driver = ConjugateGradient {
  LatticeOpt = Yes
  Isotropic = Yes
}
```

- Then the details of the DFTB hamiltonian follow:

```
Hamiltonian = DFTB {
```

- Within this block, we first specify the location of the parametrization files (the Slater-Koster files) and provide additional information about the highest angular momentum for each element (this information is not yet stored in the Slater-Koster-files):

```
MaxAngularMomentum {
  C = "p"
}
SlaterKosterFiles = Type2FileNames {
  Prefix = "../../../sk/"
  Separator = "-"
  Suffix = ".skf"
}
```

  Please note, that the highest angular momentum is **not a free parameter** to be changed, but it must correspond to the value given in the documentation section of the correspoding homonuclear Slater-Koster-files (e.g. see the *C-C.skf* file for carbon).

- We use the self-consistent charge approach (SCC-DFTB), enabling charge transfer between the atoms:

```
SCC = Yes
```

- As graphene is metallic we smear the filling function to achieve better SCC-convergence:

```
Filling = Fermi {
  Temperature [Kelvin] = 100
}
```

- For the Brillouin-zone sampling we set our k-points according to the 48 x 48 x 1 Monkhorst-Pack sampling scheme. This contains those k-points which would be folded onto the k-point (0.5, 0.5, 0.0) of an enlarged supercell consisting of the primitive unit cell repeated by (48, 0, 0), (0, 48, 0) and (0, 0, 1). This can be easily specified with the `SupercellFolding` option, where one defines those supercell vectors followed by the target k-point.

```
KPointsAndWeights = SuperCellFolding {
  48 0 0
  0 48 0
  0 0 1
  0.5 0.5 0.0
}
```

- We also want to do some additional analysis by evaluating the contributions of the *s*- and *p*-shells to the density of states (DOS). Accordingly, we instruct DFTB^{+} in the `Analysis` block to calculate the contribution of all C atoms to the DOS in a shell-wise manner (s and p) and store the shell-contributions in files starting with a prefix of *pdos.C*:

```
Analysis {
  ProjectStates {
    Region {
            Atoms = C
            ShellResolved = Yes
            Label = "pdos.C"
          }
        }
      }
```

Running the code

When you run DFTB$^+$, you should always save its output into a file for later inspection. We suggest using a construction like this (output is saved into the file *output*):

```
dftb+ | tee output
```

You will see that DFTB$^+$ optimizies the geometry of graphene by changing the lattice vectors and ion coordinates to locally minimise the total energy. As the starting geometry is quite close to the optimum one, the calculation should finish almost immediately.

Apart from the saved output file (*output*), you will find several other new files created by the code:

dftb_ pin.hsd **Contains the parsed user input with all the default** settings for options which have not been explicitely set by the user. You should have look at it if you are unsure whether the defaults DFTB$^+$ used for your calculation are appropriate, or if you want to know which other options you can use to adjust your calculation.

detailed.out **Contains detailed information about the calculated** physical quantities (energies, forces, eigenlevels, fillings, charges, etc.) obtained in the last SCC cycle performed.

band.out **Eigenvalues (in eV) and fillings for each k-point and spin** channel.

charges.bin **Charges of the atoms at the last iteration, stored in** binary format. You can use this file to restart a calculation with those atomic charges.

geo_ end.xyz, ***geo_ end.gen*** **Final geometry in both XYZ and GEN** formats.

pdos.C.1.out, ***pdos.C.2.out*** **Output files containing the projected** density of states for the first and second angular shells of carbon (in this case the *2s* and *2p* shells). Their format is similar to *band.out*.

Analysing results

The very first thing you should check is whether your calculation has converged at all to a relaxed geometry. The last line of the *output* file contains the appropriate message:

```
Geometry converged
```

This means that the program stopped because the forces on the atoms which are allowed to move (all of them in this example) were less than a given tolerance (specified in the option `MaxForceComponent`, which defaults to 1e-4 atomic units) and not instead because the maximal number of geometry optimization steps have been executed (option `MaxSteps`, default 200).

You should visualize the resulting structure using Jmol (or any other molecular visualization tool). You should probably repeat the geometry again to get a better idea how it looks like, as we did for the starting structure above. The distance between the C atoms should be very similar to those in the initial structure.

In order to visualize the density of states and the partial density of states, you should convert the corresponding human readable files (with prefix *.out*) to XY-format data

```
dp_dos band.out dos.dat
dp_dos -w pdos.C.1.out pdos.C.1.dat
dp_dos -w pdos.C.2.out pdos.C.2.dat
```

Please note the flag `-w`, which is mandatory when converting *partial* density of states data for plotting. You can obtain more information about various flags for dp_dos by issuing:

```
dp_dos -h
```

You can visualize the DOS and the PDOS for the *s*- and *p*-shells of carbon in one picture using the *plotxy* tool, which is a simple command line wrapper around the matplotlib python library (issue the command `plotxy -h` for help):

```
plotxy --xlabel "Energy [eV]" --ylabel "DOS" dos.dat pdos.C.1.dat pdos.C.2.dat &
```

You can use also any other program (gnuplot, xmgrace) which can visualize XY-data. You should see something similar to Figure *DOS and PDOS of graphene* (page 21).

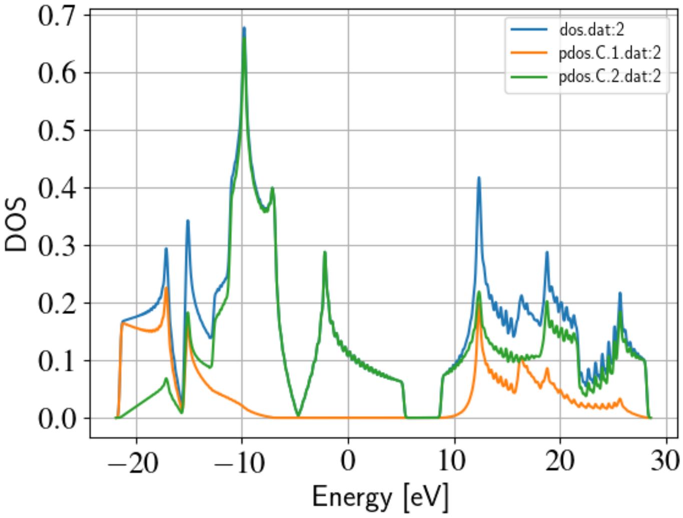

Fig. 3.2: DOS and PDOS of graphene

The position of the Fermi level (at -4.67 eV) can be read out from the *detailed.out* file, either directly or by using an appropriate *grep* command:

```
grep "Fermi level" detailed.out
```

As expected for graphene, the DOS vanishes at the Fermi-level. Around the Fermi-level, all states are composed of the *p*-orbitals of the carbons, the *s*-orbitals only contribute to energeticaly much lower and much higher states. Also, one can observe the van-Hove-singularties. The `wiggles` at around 0 eV and at higher energy are artifacts. Using more k-points for the Brillouin-zone sampling or using a slightly wider broadening function in *dp_dos* would smooth them out.

3.1.2 Band structure

[Working directory: *elect/graphene/bands/*]

Band structure calculations in DFTB (as in DFT) always consist of two steps:

1. Calculating an accurate ground state charge density by using a high quality k-point sampling.
2. Determining the eigenvalues at the desired k-points of the band structure, using the density obtained in the previous step. The density is not changed during this step of the band structure calculation.

Step 1 you just have executed, so you can copy the final geometry and the data file containing the converged charges from that calculation into your current working directory:

```
cp ../latopt/geo_end.gen .
cp ../latopt/charges.bin .
```

Have a look on the *dftb_in.hsd* file for the band structure calculation. It differs from the previous one only in a few aspects:

- We use the end geometry of the previous calculation as geometry:

```
    Geometry = GenFormat {
      <<< "geo_end.gen"
    }
```

- We need static calculation only (no atoms should be moved), therefore, no driver block has been specified.
- The k-points are specified along specific high symmetry lines of the Brillouin-zone (K-Gamma-M-K):

```
    KPointsAndWeights = KLines {
       1    0.33333333  0.66666666 0.0    # K
      20    0.0  0.0  0.0                 # Gamma
      20    0.5  0.0  0.0                 # M
      10    0.33333333  0.66666666 0.0    # K
    }
```

- We initialize the calculation with the charges stored during the previous run:

```
    ReadInitialCharges = Yes
```

- We do not want to change the charges during the calculation, therefore, we set the maximum number of SCC cycles to one:

```
    MaxSCCIterations = 1
```

Let's run the code and convert the band structure output to XY-format:

```
dftb+ | tee output
dp_bands band.out band
```

The dp_bands tool extracts the band structure from the file *band.out* and stores it in the file *band_tot.dat*. For spin polarized systems, the name of the output file would be different. Use:

```
dp_bands -h
```

to get help information about the arguments and the possible options for dp_bands.

In order to investigate the band structure we first look up the position of the Fermi level in the previous calculation performed with the accurate k-sampling

```
grep "Fermi level" ../latopt/detailed.out
```

which yields -4.67 eV, and then visualize the band structure by invoking

```
plotxy -L --xlabel "K points" --ylabel "Energy [eV]" band_tot.dat &
```

This results in the band structure as shown in Figure *Band structure of graphene* (page 23).

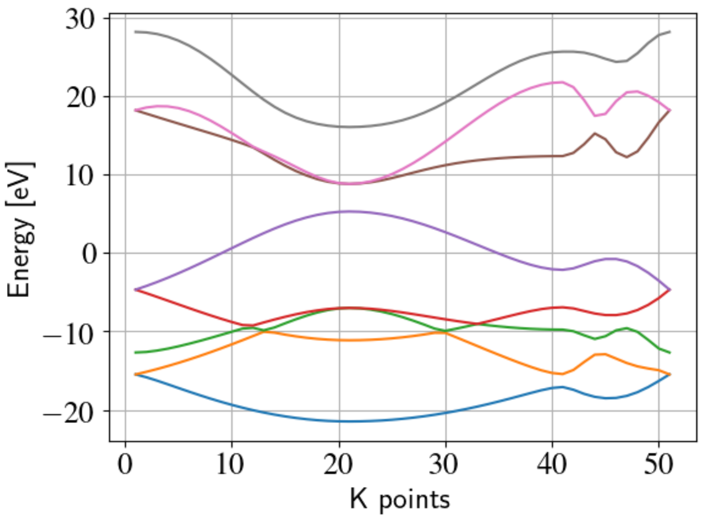

Fig. 3.3: Band structure of graphene

You can see the linear dispersion relations around the point K in the Brillouin-zone (k-points 0 and 51 in our circuit) which is a very typical characteristic of graphene.

3.2 Zigzag nanoribbon

Next we will study some properties of a hydrogen saturated carbon zigzag nanoribbon.

3.2.1 Calculting the density and DOS

[Working directory: *elect/zigzag/density/*]

The initial geometry for the zigzag nanoribbon contains one chain of the structure, repeated periodically along the z-direction. The lattice vectors orthogonal to the periodicity (along the x- and y- axis) are set to be long enough to avoid any interaction between the repeated images.

First convert the GEN-file to XYZ-format and visualize it:

```
gen2xyz geo.gen
jmol geo.xyz &
```

Similar to the case of perfect graphene, you should check first the initial geometry by repeating it along the periodic axis (the third lattice vector in this example) and visualize it. The necessary steps are collected in the file *checkgeo.sh*. Please have a look at its content to understand what will happen, and then issue

```
sh checkgeo.sh
```

to obtain the molecule shown in Figure *Section of an H-saturated zigzag nanoribbon* (page 24).

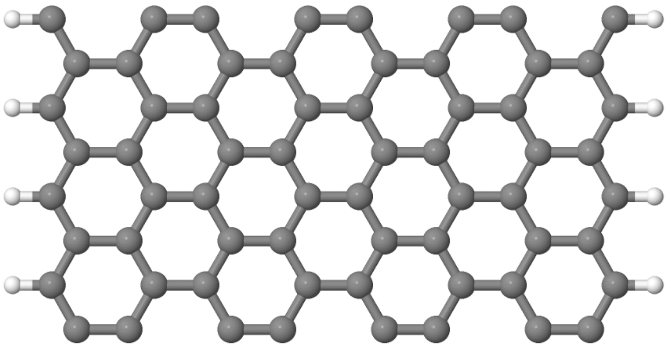

Fig. 3.4: Section of an H-saturated zigzag nanoribbon

The control file *dftb_in.hsd* is similar to the previous examples, with a few differences only:

- We use the 1 x 1 x 24 Monkhorst-Pack k-point set to sample the Brillouin-zone, since the ribbon is only periodic along the direction of the third lattice vector. The two other lattice vectors have been choosen to be long enough to avoid interaction between the artificially repeated ribons.:

```
KPointsAndWeights = SupercellFolding {
  1 0 0
  0 1 0
  0 0 24
  0.0 0.0 0.5
}
```

- In order to analyze, which atoms contribute to the states around the Fermi-level, we create four projection regions containing the saturating H-atoms, the C atoms in the outermost layer of the ribbon, the C atoms in the second outermost layer and finally the C atoms in the thirds outermost layer, respectively. Since the ribbon is mirror symmetric, we include the corresponding atoms on both sides in each projection region:

```
ProjectStates {

  # The terminating H atoms on the ribbon edges
  Region {
    Atoms = H
    Label = "pdos.H"
  }

  # The surface C atoms
  Region {
    Atoms  = 2 17
    Label = "pdos.C1"
  }

  # The next row of C atoms further inside
  Region {
      Atoms = 3 16
      Label = "pdos.C2"
    }

    # Some more 'bulk-like' C atoms even deeper
    Region {
      Atoms = 4 15
      Label = "pdos.C3"
    }
  }
```

You can run the program and convert the output files by issuing:

```
sh run.sh
```

When the program has finished, look up the Fermi-level and visualize the DOS and PDOS contributions. The necessary commands are collected in *showdos.sh*:

```
sh showdos.sh
```

When you zoom into the area around the Fermi level (-4.57 eV), you should obtain something like Figure *DOS of the zigzag nanoribbon around the Fermi energy* (page 25).

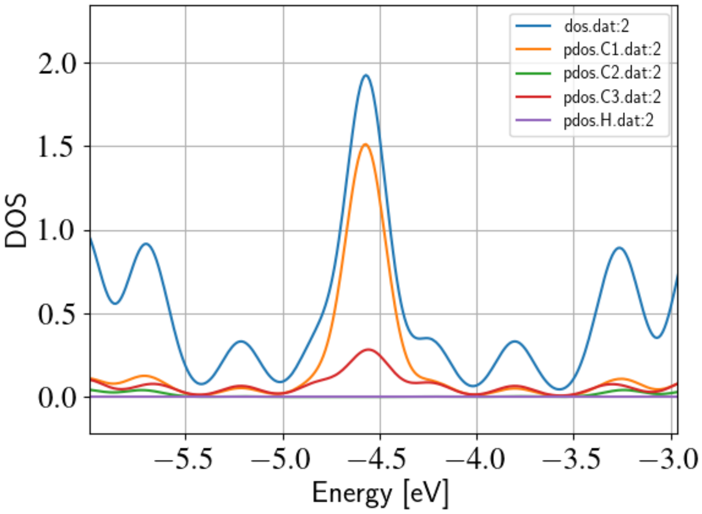

Fig. 3.5: DOS of the zigzag nanoribbon around the Fermi energy

You can see that the structure is clearly metallic (displaying a non-zero density of states at the Fermi energy). The states around the Fermi-level are composed of the orbitals of the C atoms in the outermost and the third outermost layer of the ribbon. There is no contribution from the C atom in the layer in between or from the H atoms to the Fermi level.

3.2.2 Band structure

[Working directory: *elect/zigzag/bands/*]

Now let's calculate the band structure of the zigzag nanoribbon. The commands are in the script *run.sh*, so just issue:

```
sh run.sh
```

You will see DFTB$^+$ finishing with an error message

```
WARNING!
-> SCC is NOT converged, maximal SCC iterations exceeded
```

Normally, it would mean that DFTB$^+$ did not manage to find a self consistent charge distribution for its last geometry. In our case, however, it is not an error, but the desired behaviour. We have specified in *dftb_in.hsd* the options

```
ReadInitialCharges = Yes
MaxSCCIterations = 1
```

requiring the program to stop after one SCC iteration. The charges are at this point not self consistent with respect to the k-point set used for sampling the band structure calculation. However, k-points along high symmetry lines of the Brillouin-zone, as used to obtain the band structures, usually represent a poor sampling. Therefore the a converged density obtained with an accurate k-sampling should be used to obtain the eigenlevels, and no self consistency is needed.

To look up the Fermi-level and plot the band structure use the commands in *showbands.sh*:

```
sh showbands.sh
```

You should obtain a band structure similar to Figure *Band structure of the zigzag nanoribbon* (page 26). Again, one can see, that there are states around the Fermi-energy, so the nanoribbon is metallic.

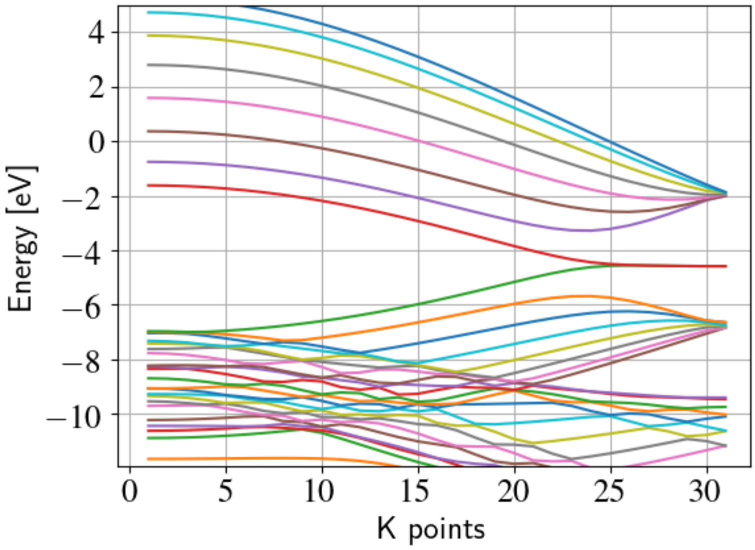

Fig. 3.6: Band structure of the zigzag nanoribbon

3.3 Armchair nanoribbon with defects

We now investigate a hydrogen saturated armchair carbon nanoribbon, examining both the perfect ribbon and two defective structures, each with a vacancy at a different position in the ribbon. In order to keep the tutorial short, we will not relax the vacancies, but will only remove one atom from the perfect structure.

3.3.1 Perfect armchair nanoribbon

Total energy and density of state

[Working directory: *elect/armchair/perfect_density/*]

The steps to calculate the DOS of the perfect H-saturated armchair nanoribbon are the same as for the zigzag case. First check the geometry with the help of repeated supercells:

```
sh checkgeo.sh
```

You will see a repeated image of the perfect armchair nanoribbon unit cell (Figure *Perfect armchair nanoribbon unit cell* (page 27)).

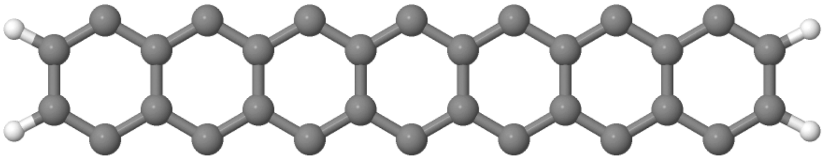

Fig. 3.7: Perfect armchair nanoribbon unit cell

The edge of the ribbon is visually different from the zigzag case. As it turns out, this also has some physical consequences. Let's calculate the electronic density and extract the density of states:

```
sh run.sh
```

If you look up the calculated Fermi-level and then visualize the DOS

```
sh showdos.sh
```

you can immediately see (Figure *DOS of the perfect armchair nanoribbon* (page 28)) that there are no states around the Fermi-energy (-4.4 eV), i.e. the investigated armchair nanoribbon is non-metallic.

Band structure

[Working directory: *elect/armchair/perfect_bands*]

Let's have a quick look at the band structure of the armchair H-saturated ribbon. The steps are the same as for the zigzag case, so just issue:

```
sh run.sh
sh showbands.sh
```

You should obtain a band structure like in Figure *The band structure of the perfect hydrogen passivated armchair nanoribbon. The Fermi energy is at -4.4 eV.* (page 28). You can read off the position of the band edges, when you zoom into the energy region around the gap: The valence band edge and the conduction band edge are in the Gamma point at -4.7 and -4.2 eV,

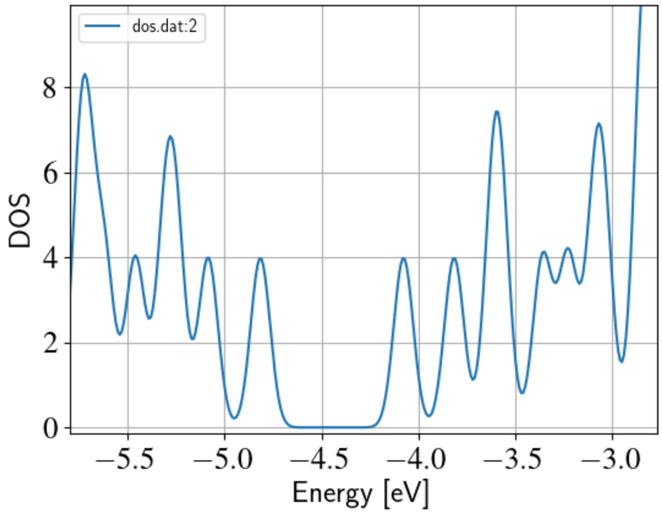

Fig. 3.8: DOS of the perfect armchair nanoribbon

respectively. You can also easily extract this information from the *band.out* file, when you look where to occupation goes from nearly 2.0 to nearly 0.0 in the first k-point (the Gamma point).

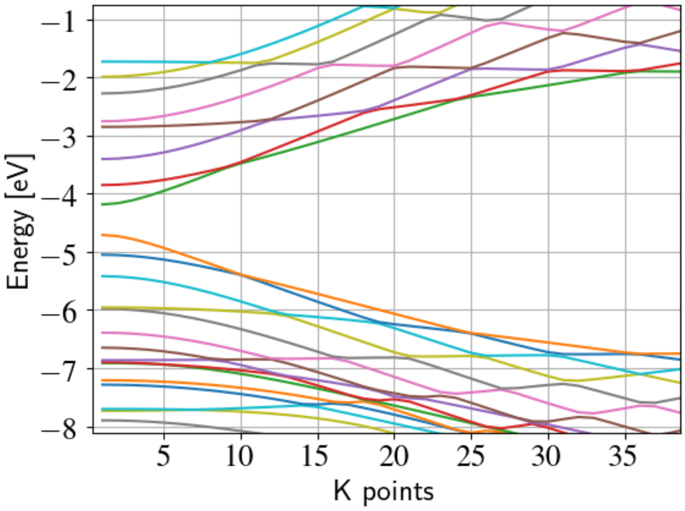

Fig. 3.9: The band structure of the perfect hydrogen passivated armchair nanoribbon. The Fermi energy is at -4.4 eV.

3.3.2 Armchair nanoribbon with vacancy

Density and DOS

[Working directory: *elect/armchair/*]

As next, we should investigate two armchair nanoribbons with a vacancy in each. The inputs can be found in the subdirectories *elect/armchair/vacancy1_density* and *elect/armchair/vacancy2_ density* and you can visualize both with the command

```
sh showgeom_v12.sh
```

As you can see on Figures *Armchair nanoribbon with vacancy (structure 1)* (page 29) and *Armchair nanoribbon with vacancy (structure 2)* (page 29), the vacancy is in the two cases on different sublattices.

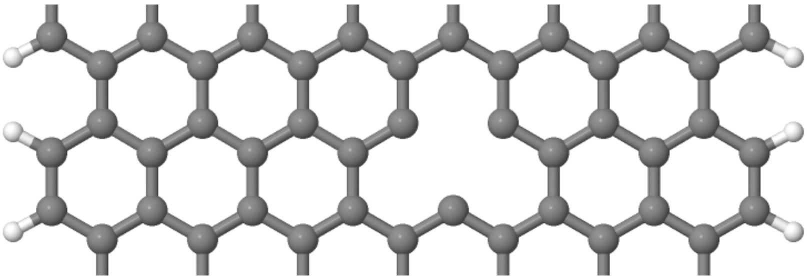

Fig. 3.10: Armchair nanoribbon with vacancy (structure 1)

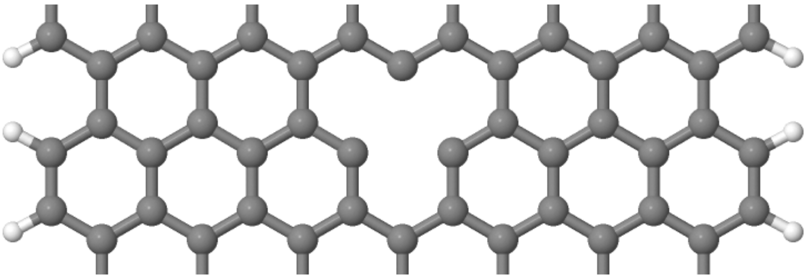

Fig. 3.11: Armchair nanoribbon with vacancy (structure 2)

The two vacancies (structures 1 and 2) are located on different sublattices. Since the geometries are periodic along the z-direction, the defects are also repeated. As we would like to calculate a single vacancy, we have to make our unit cell for the defect calculation large enough to avoid significant defect-defect interactions. In this case, the defective cells contain twelve unit cells.

In order to calculate the electron density of both vacancies, issue:

```
sh run_v12.sh
```

This will take slightly longer than the previous calculations, since each system contains more than four hundred atoms.

We want to analyse the density of states of the two different vacancies, together with that of the defect-free system. The commands necessary to extract the DOS of all three configurations and show them in one figure have been stored in the script `showdos_perf_v12.sh`. Execute it

```
sh showdos_perf_v12.sh
```

to obtain a figure like Figure *The DOS of the perfect nanoribbon is indicated by solid blue line, the DOS of the nanoribbons with vacancies with green and red lines, respectively.* (page 30).

As you can see, in contrast to the zigzag nanoribbon, the perfect armchair nanoribbon is insulating as it has no states around the Fermi-energy (-4.45 eV). The structures with vacancies, on the

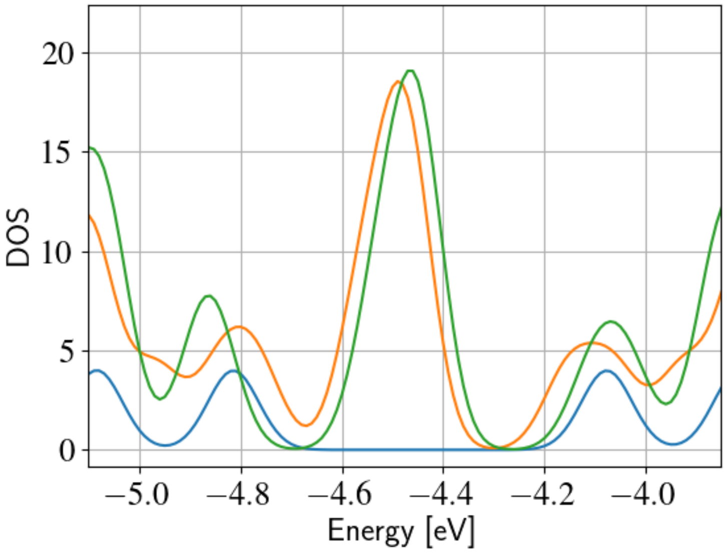

Fig. 3.12: The DOS of the perfect nanoribbon is indicated by solid blue line, the DOS of the nanoribbons with vacancies with green and red lines, respectively.

other hand, introduce dangling (unsaturated) bonds, leading to unoccupied states around the Fermi-energy. We can also see, that the defects affect the band edges, which are shifted with respect to their position in the perfect structure. It also seems that the valence band edge is more affected than the conduction band edge, and in the case of vacancy 2 (red line) the effect is significantly larger than for vacancy 1 (green line).

Vacancy formation energy

You should also be able to calculate the formation energies of the two vacancies. The formation energy E_{form} of the vacancy in our case can be calculated as

$$E_{\mathrm{form}} = (E_{\mathrm{vac}} + E_{\mathrm{C}}) - 12 \times E_{\mathrm{perf}}$$

where E_{vac} is the total energy of the nanoribbon with the vacancy present, E_{C} is the energy of a C-atom in its standard phase and E_{perf} is the energy of the perfect nanoribbon. Since the defective nanoribbons contain 12 unit cell of the perfect one, the energy of the perfect ribbon unit cell has to be multiplied by twelve. As a standard phase of carbon, we will take perfect graphene for simplicity. The energy of the C-atom in its standard phase is then obtained by dividing the total energy of the perfect graphene primitive unit cell by two. (Look up this energy from *detailed.out* in the directory *elect/graphene/density*.) By calculating the appropriate quantities you should obtain ~8.5 eV for the formation energy of both vacancies. This is quite a high value, but you should recall that the vacancies have not been structurally optimised, and their formation energies are therefore, significantly higher than for the relaxed configurations.

Defect levels

[Working directory: *elect/armchair/vacancy2_wf/*]

Finally we should identify the localised defect levels for vacancy 2 and plot the corresponding one-electron wavefunctions.

The vacancy was created by removing one C-atom, which had three first neighbors. Therefore, three *sp2* type dangling bonds remain in the lattice, which will then form some linear combinations to produce three defect levels, which may or may not be in the band gap. The DOS you have plotted before, indicates there are indeed defect levels in the gap, but due to the smearing it is hard to say how many they are.

We want to investigate the defect levels at the Gamma point, as this is where the perfect nanoribbon has its band edges. We will therefore do a quick Gamma-point only calculation for vacancy structure 2 using the density we obtained before. We will set up the input to write out also the eigenvectors (and some additional information) so that we can plot the defect levels with *waveplot* later. This needs the following additional settings in *dftb_in.hsd*:

```
Options {
  WriteEigenvectors = Yes
  WriteDetailedXML = Yes
  WriteDetailedOut = No
}
```

To just run the calculation

```
sh run.sh
```

and open the *band.out* file. You will see, that you have three levels (levels 742, 743 and 744 at energies of -4.51, -4.45 and -4.45 eV, respectively) which are between the energies of the band edge states of the perfect ribbon. We will visualize those three levels by using the *waveplot* tool.

Waveplot reads the eigenvectors produced by DFTB$^+$ and plots real space wave functions and densities. The input file *waveplot_in.hsd* can be used to control which levels and which region waveplot should visualize, and on what kind of grid. In the current example, we will project the real part of the wave functions for the levels 742, 743 and 744. In order to run Waveplot, enter:

```
waveplot | tee output.waveplot
```

The calculation could again take a few minutes. At the end, you should see three files with the *.cube* prefix, containing the volumetric information for the three selected one-electron wavefunctions.

We will use Jmol to visualize the various wave function components. Unfortunately, the visualization of iso-surfaces in Jmol needs some scripting. You can find the necessary commands in the files *show*.js*. You can either type in these commands in the Jmol console (which should be opened via the menu *File / Console...*) or pass it to Jmol using the *-s* option at start-up. For the case latter you will find prepared command to visualize the various orbitals in the files

```
sh showdeflev1.sh && sh showdeflev2.sh && sh showdeflev3.sh
```

Looking at the defect levels, you can see that the defect level lowest in energy (742) has a significant contribution on the atoms around the defect, but also a non-negligible delocalized part smeared over almost all atoms in the system. Apparently a localized defect level has hybridized with the delocalized valence band edge state, resulting in a mixture between localized and non-localized state. The other two defect levels, on the other hand, have wavefunctions which are well localized on the atoms around the vacancy site. Note that in accordance with the overall symmetry of the system, the defect levels are either symmetric or antisymmetric with respect to the mirror plane in the middle of the ribbon.

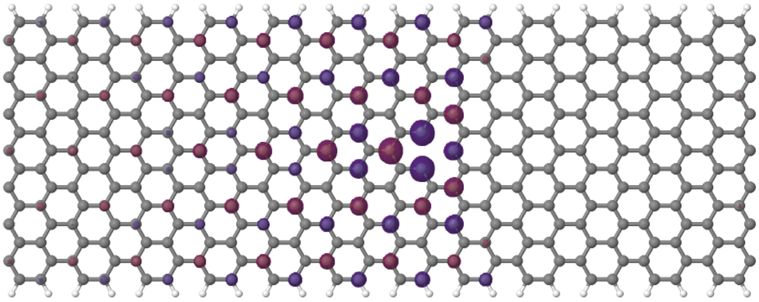

Fig. 3.13: Wave function of the lowest defect level of the hydrogen saturated armchair nanoribbon with a vacancy. Blue and red surfaces show indicate isosurfaces at +0.02 and -0.02 atomic units, respectively.

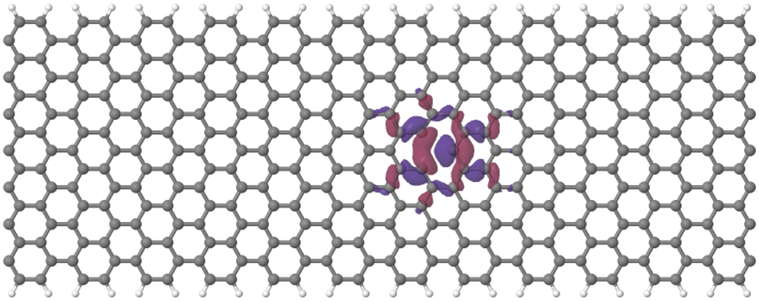

Fig. 3.14: Wave function of the second lowest defect level of the hydrogen saturated armchair nanoribbon with a vacancy.

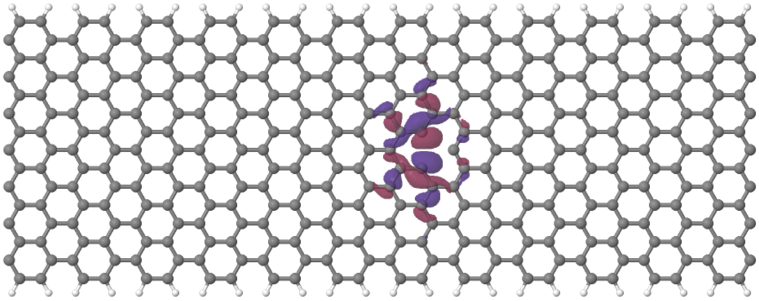

Fig. 3.15: Wave function of the highest defect level of the hydrogen saturated armchair nanoribbon with a vacancy.

CHAPTER 4

Electron transport calculations in armchair nanoribbons

In this sections of the tutorial we will learn how to set up self-consistent and non self-consistent simulations with open boundary conditions. We will then calculate the density of states and the transmission coefficients and then analyse the results in comparison with the previous periodic calculations.

If you have not done so yet, please download the file *tutorial_ input.zip* and decompress it by issuing:

```
unzip tutorial_input.zip
```

in some directory. Then enter the directory *transport/*. All directories given in this part of the tutorial are sub-directories of the *transport/* directory.

4.1 Non-SCC Pristine armchair nanoribbon

[Working directory: *transport/agr_ nonscc/ideal/*]

4.1.1 Preparing the structure

When we run a transport calculation with open boundary conditions, the geometric structure specified in the input needs to obey some rules. The system must consist of an extended device (or molecule) region, and two or more semi-infinite bulk contacts. The bulk contacts are described by providing two *principal layers* for each contact.

A *Principal Layer (PL)* is defined as a contiguous group of atoms that have finite interaction only with atoms belonging to adjacent PLs. In a sense, a PL is a generalisation of the idea of nearest neighbour atoms to the idea of nearest neighbour blocks. The PL partitioning in the electrodes is used by the code to retrieve a description of the bulk system. PLs may be defined, as we will see, in the extended device region to take advantage of the iterative Green function solver algorithm.

Additional information about the definition of PLs, contacts and extended device region can be found in the manual and in the on-line recipes.

In the case of an ideal one-dimensional system, all the PLs are identical. The system we start with is an infinite armchair graphene nanoribbon (AGR), therefore the partitioning into device and contact regions is somewhat arbitrary. We will therefore start from a structure file containing a single PL (*2cell_ 7.gen*), which has been previously relaxed. The PL can be converted to XYZ format by using the *gen2xyz* script and visualised with *Jmol*:

```
gen2xyz 2cell_7.gen
jmol 2cell_7.xyz
```

The structure is shown in Figure *Armchair nanoribbon principal layer (PL)* (page 34)

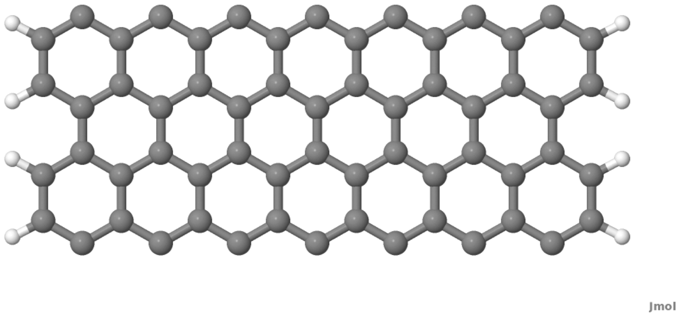

Fig. 4.1: Armchair nanoribbon principal layer (PL)

As you may notice, we did not take a single unit cell length as a PL, but rather two unit cells. This choice is dictated by the definition of the PL itself, as we want to avoid non-zero interactions between second-neighbour PLs. This is better explained by referring to Figure *Layer definition* (page 34). The red carbon atoms represent the closest atoms which would belong to non-nearest neighbour PLs, and these have a separation of 0.568 nm, as shown in Figure *Layer definition* (page 34). The carbon-carbon interaction is non zero up to a distance of 6 a.u., therefore the interaction between the two red atoms would be small, but non zero. Hence this is too small a separation for a one unit cell long section of nanoribbon to be used as the PL.

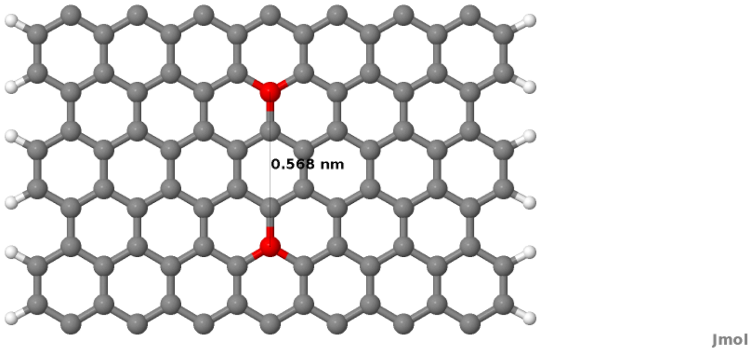

Fig. 4.2: Layer definition

As currently there is no way to damp out small interactions, the PL must contain two unit cells in this case, as shown in figure *Layer definition* (page 34). It follows that the correct definition of a PL depends both on the geometry of the system and the interaction cut-off distance.

After having defined a proper PL, we then build a structure consisting of a device region with 2 PLs and contacts at each end of this region, each with 2 PLs.

Note: For the pristine system, the equilibrium results should not depend on the length of the device region, as the represented system is an infinite ideal nanoribbon with discrete translational symmetry along the ribbon.

The input atomic structure must be defined according to a specific ordering: the device atoms come first, then each contact is specified, starting with the PL closer to the device region. For an ideal system defined by repetition of identical PLs, the tool *buildwire* (distributed with the code) can be used to build a 1D geometry with the right ordering.

When you type:

```
buildwire 2cell_7.gen
```

you will be asked to type the name of the file containing the input supercell (*2cell_ 7.gen*), direction and the number of principal layers in the device region (we will set this to be 2). *buildwire* will always give its output as a supercell structure, but in some cases we will need to manually modify this structure file so that it corresponds to the ordering explained in the previous paragraph.

The following output will be visualised:

```
 Insert PL .gen file name:
2cell_7.gen
 Insert direction (e.g. 0 0 1)
0 0 1
 Insert number of PLs in channel:
2
 structure built
 *iatc=
        137         272 0
        273         408 0
 *PLs=
    1    69;
```

The indexes **iatc** and **PLs** define the atoms belonging to the contacts, to the device region and to the PLs of the device region, and will be useful when we will write the input files. You should take a note of them (the iterative algorithm used in solving the Green function requires these values). A file *Ordered_ 2cell_ 7.gen* will have been created, and defined as a supercell format GEN file (`S`), which we will rename *device_ 7.gen* for the following:

```
mv Ordered_2cell_7.gen device_7.gen
```

We can better understand the ordering of the atomic indexes if we convert this structure to XYZ, open it with jmol and then change the colours of specific ranges of atoms by using the following syntax in the jmol console (for example, we select here the first contact and split it into two sub-ranges containing its first and second PLs):

```
> select atomno>136 && atomno<205
> color yellow
> select atomno>204 && atomno<273
> color red
```

In Figure *The PLs of contact 1* (page 36) a *Jmol* export of the structure is shown.

The yellow and red atoms represent the first and second PLs of the first contact. When you build a structure yourself, it is always a good idea to use a visualiser and verify that the atomic indices are consistent with the transport setup definitions.

The last step is to change the supercell definition in the gen structure file. From the point of view of an open boundary condition calculation, Supercell (`S`) and Cluster (`C`) have a slightly different

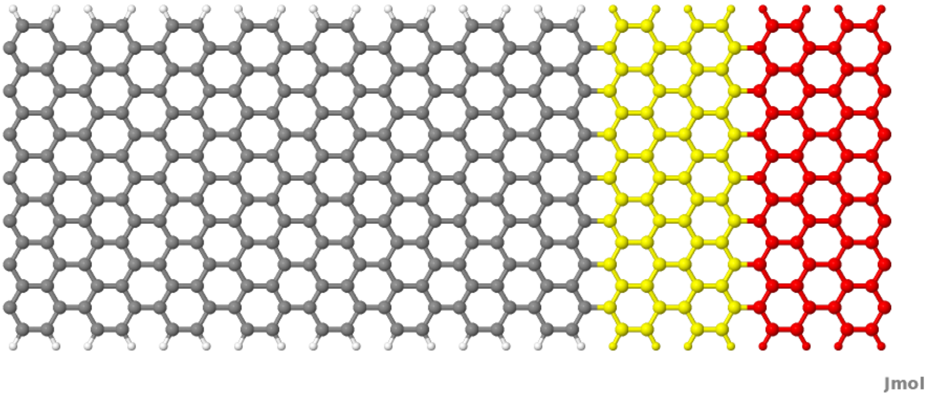

Fig. 4.3: The PLs of contact 1

meaning with respect to a canonical *dftb* calculation. By Supercell we mean any structure which is *periodic in any direction transverse to the transport direction*, while for cluster we mean any structure *not periodic in any direction transverse to transport.* It follows that purely 1D systems, like nanowires and nanoribbons, should be regarded as clusters (`C`). Therefore we edit the structure file *device_ 7.gen*, changing in the first line the `S` (supercell) to be `C` (cluster) and remove the last four lines, which would normally only be defined for periodic systems.

This is the file we get after running *buildwire*:

```
408  S
C    H
  1    1     37.831463060000    -20.000000000000      0.710000000000
  2    1     39.061219140000    -20.000000000000      1.420000000000
  3    1     39.061219140000    -20.000000000000      2.840000000000
  4    1     37.831463060000    -20.000000000000      3.550000000000
  5    1     35.371950920000    -20.000000000000      0.710000000000
  6    1     36.601706990000    -20.000000000000      1.420000000000
  7    1     36.601706990000    -20.000000000000      2.840000000000
  8    1     35.371950920000    -20.000000000000      3.550000000000
  ........
  65    2     20.880312110000    -20.000000000000    -11.870830122700
  66    2     20.880312110000    -20.000000000000     -9.429169877000
  67    2     40.025607920000    -20.000000000000    -11.870893735700
  68    2     40.025607920000    -20.000000000000     -9.429106264000
  0.000000000000000      0.000000000000000      0.000000000000000
  59.676097169999998      0.0000000000000000      0.0000000000000000
  0.0000000000000000     -40.000000000000000      0.0000000000000000
  0.0000000000000000      0.0000000000000000      51.119999999999997
```

Note that the numbering of atoms at the start of each line, as output by *buildwire* are sequential according to the numbering of the initial structure, not its global position in the output file.

The corrected definition for the 1D ribbon with open boundary conditions is then:

```
408  C
C    H
  1    1     37.831463060000    -20.000000000000      0.710000000000
  2    1     39.061219140000    -20.000000000000      1.420000000000
  3    1     39.061219140000    -20.000000000000      2.840000000000
  4    1     37.831463060000    -20.000000000000      3.550000000000
  5    1     35.371950920000    -20.000000000000      0.710000000000
  6    1     36.601706990000    -20.000000000000      1.420000000000
  7    1     36.601706990000    -20.000000000000      2.840000000000
  8    1     35.371950920000    -20.000000000000      3.550000000000
  ........
  65    2     20.880312110000    -20.000000000000    -11.870830122700
  66    2     20.880312110000    -20.000000000000     -9.429169877000
  67    2     40.025607920000    -20.000000000000    -11.870893735700
  68    2     40.025607920000    -20.000000000000     -9.429106264000
```

Now the file *device_7.gen* contains the correct structure, defined as a cluster and with the proper atom ordering. Next, we set up the input file for a transmission calculation.

4.1.2 Transmission and density of states

In the DFTB$^+$ input format, settings related to a transport calculation may be required to appear in separate sections of the *dftb_in.hsd* file, depending on the functionality they invoke. In the following we will set up the simplest open boundary condition calculation: a calculation of transmission coefficients according to the Landauer-Caroli formula, assuming a non-SCC DFTB hamiltonian. We will analyse and comment the different sections contained in the file *dftb_in.hsd*.

First, we have the specification of the geometry:

```
Geometry = GenFormat {
<<< 'device_7.gen'
}
```

This follows the same rule as in a regular DFTB$^+$ calculation, except for the fact that the structure should follow the specific partitioning structure explained in the previous section.

Whenever an open boundary system is defined, we have to specify a block named *Transport* which contains information on the system partitioning and additional information about the contacts to the device:

```
Transport {
  Device {
    AtomRange = 1 136
    FirstLayerAtoms =  1 69
  }
  Contact {
    Id = "source"
    AtomRange = 137 272
    FermiLevel [eV] = -4.7103
    potential [eV] = 0.0
  }
  Contact {
    Id = "drain"
    AtomRange = 273 408
    FermiLevel [eV] = -4.7103
    potential [eV] = 0.0
  }
}
```

Here we have used the indexes printed by *buildwire*. *Device* contains two fields: *AtomRange* specifies which atoms belong to the extended device region (1 to 136) and *FirstLayerAtoms* specify the starting index of the PLs in the device region. This field is optional, but if not specified the iterative algorithm will not be applied and the calculation will be slower, even though the result will be still correct. Then we have the definitions of the contacts. In this example we define a two terminal system, but in general N contacts are allowed. A contact is defined by an *Id* (mandatory), the range of atoms belonging to the contact specified in *AtomRange* (mandatory) and a *FermiLevel* (mandatory). The potential is set by default to 0.0, therefore need not be specified in this example. Note that according to equilibrium Green function theory, the Fermi level and the contact potential are not necessary to calculate the transmission curve, but are required to calculate the current via the Landauer formula, as they would determine the occupation distribution in the contacts.

Then we have the *Hamiltonian* block, which describes how the initial Hamiltonian and the SCC component, if any, will be calculated:

```
Hamiltonian = DFTB {
  SCC = No
  MaxAngularMomentum = {
    C = "p"
    H = "s"
  }

  SlaterKosterFiles = Type2FileNames {
    Prefix = "../../../sk/"  # To be substituted with the path to
                             # SK parameters on your local disk
    Separator = "-"
    Suffix = ".skf"
  }

  Eigensolver = TransportOnly{}

}
```

In this example we will calculate the transmission according to Caroli (referred by some authors as Fisher Lee) formula in a non-SCC approximation, i.e. the Hamiltonian is directly assembled from the Slater-Koster files and used *as is* to build the contact self energies and the extended device Green function. The definition of an eigensolver is not meaningful in an open boundary setup, as the system is instead solved by the Green function technique. Therefore we just use a keyword *TransportOnly* to indicate that we do not want to solve an Eigenvalue problem. The other fields are filled up in the same way as for a regular DFTB$^+$ calculation.

In general, in DFTB$^+$ an Eigensolver is regarded as a calculator which can provide charge density in the SCC cycle, therefore we will define a Green function based eigensolver later, but only for SCC calculations.

Note that as C-H bonds are present in the system, charge transfer should occur, hence the result will not be accurate at the non-SCC level. It is not *a-priori* trivial to predict whether this affects qualitatively or quantitatively the transmission. We will therefore later compare these results with an SCC calculation - at the moment we will stay at the level of a non-SCC calculation, because it is faster to execute and also allows us to use the simplest input file possible.

Finally, the implementation of the Landauer-Caroli formula is regarded as a post-processing operation and specified by the block *TransmissionAndDos* inside *Analysis*:

```
Analysis = {
    TransmissionAndDos {
      Verbosity = 101
      EnergyRange [eV] = -6.5  -3.0
      EnergyStep [eV] = 0.01
      Region = {
        Atoms = 1:136
      }
    }
}
```

TransmissionAndDos allows for the calculation of transmission coefficient, Local Density of States (LDOS) and current. A transmission is always calculated using the energy interval and energy step specified here. The LDOS is only calculated when sub-blocks *Region* are defined. *Region* can be used to select some specific subsets of atoms or orbitals, according to the syntax explained in the manual. In this example, we are specifying the whole extended device region (atoms 1 to 136). Note that the energy range of interest is not known a priori. Either you have a reference band structure calculation, therefore you know where the first sub-bands are (this is the correct way to do this), or you can run a quick calculation with a large energy step and on the basis of the transmission curve then refine the range of interest.

Now that the input file is complete, we have to complete one last step. During a transport run, DFTB$^+$ will look for two directories named *GS* and *contacts*. We have to create these directories in advance:

```
mkdir GS
mkdir contacts
```

We can then start the calculation:

```
dftb+ dftb_in.hsd | tee output
```

We can take advantage of parallelisation over the energy points in the calculation by running the code with *mpirun*:

```
mpirun -n 4 dftb+ dftb_in.hsd | tee output
```

where *4* should be substituted by the number of available nodes. Note that Green function method is parallelised over energy points, therefore a number of nodes larger than the energy grid will not improve performances and secondly that the memory consumption is proportional to the number of nodes used - this may be critical in shared memory systems with a small amount of memory per node.

When the calculation has finished, the transmission and density of states are saved to both the *detailed.out* file and to two separate *transmission.dat* and *localDOS.dat* files. These additional files both contain the energy points in the first column and the desired quantities as additional columns.

We can plot the transmission by using the *plotxy* script:

```
plotxy --xlabel 'Energy [eV]' --ylabel 'Transmission' -L transmission.dat &
```

The plot is shown in Figure *Non-SCC transmission through a pristine AGR* (page 40):

The ribbon is semiconducting, therefore we can see a zero transmission at energies corresponding to the band gap. As the system is ideal, outside of the band gap we can observe the characteristic conductance steps where the value of the transmission is 1.0 for every band which crosses a given energy. This is a normal signature of ideal 1D systems with translational invariance.

Similarly, we can visualise the density of states by typing (the x and y axis limits are chosen to focus on the first few sub-bands):

```
plotxy --xlabel 'Energy [eV]' --ylabel 'DOS [arbitrary units]' -L \
--xlimits -6.5 -3 --ylimit 0 1000 localDOS.dat &
```

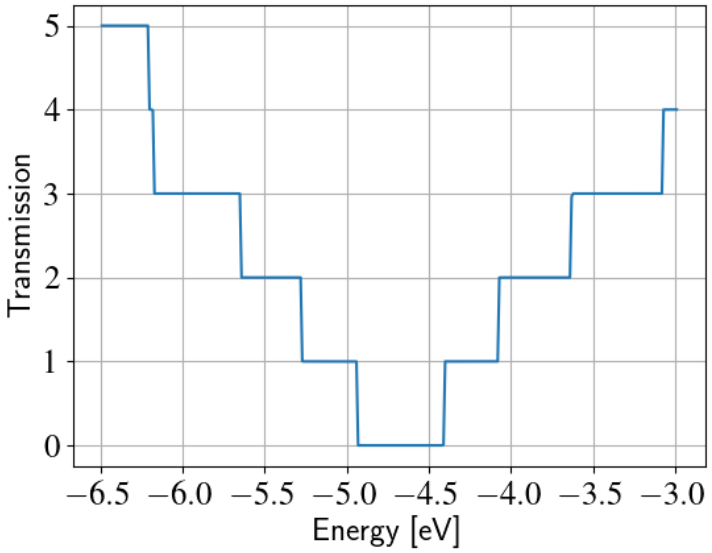

Fig. 4.4: Non-SCC transmission through a pristine AGR

The result is shown in Figure *Non-SCC density of states for a pristine AGR* (page 40):

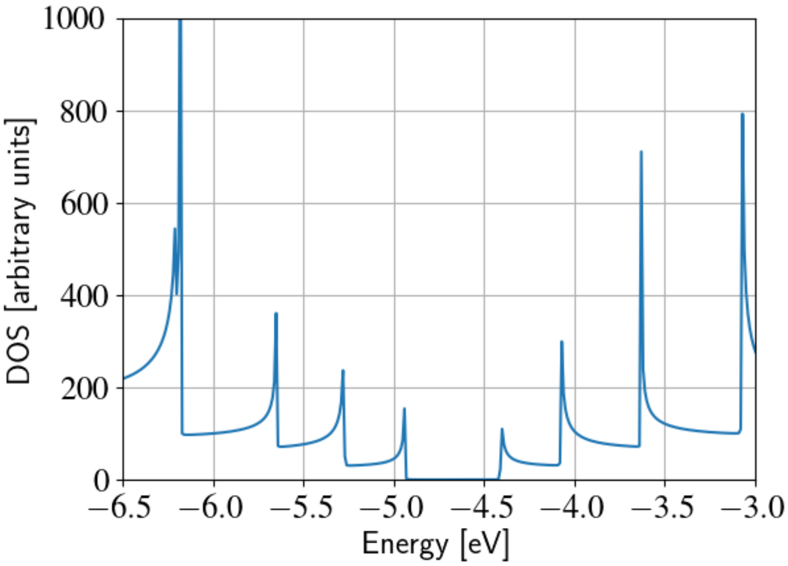

Fig. 4.5: Non-SCC density of states for a pristine AGR

You can plot the transmission or the density of states on a semi-logarithmic scale:

```
plotxy --xlabel 'Energy [eV]' --ylabel 'DOS [arbitrary units]' -L \
--xlimits -6.5 -3 --logscale y localDOS.dat &
```

If you do so, you will obtain the plot shown in Figure *Non-SCC density of states on logarithmic scale* (page 41).

The density of states in the band-gap is not zero, but decreases by several orders of magnitude.

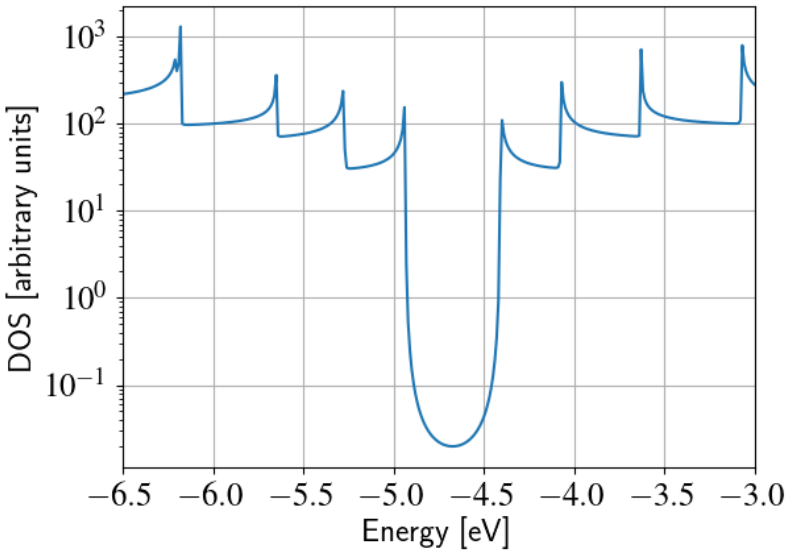

Fig. 4.6: Non-SCC density of states on logarithmic scale

This is a natural consequence of the quasi-particle nature of the Green function formalism: every state in the system has a finite broadening in energy.

4.2 Non-SCC armchair nanoribbon with vacancy (A)

[Working directory: *transport/agr_nonscc/vacancy1/*]

4.2.1 Transmission and Density of States

Now that we have a calculation of the reference pristine system, we will introduce a scattering centre by producing a vacancy in the system. In order to do so, we directly modify the structure file *device_7.gen* and the input file *dftb_in.hsd*. We remove atom number 48 from the structure file. Note that DFTB$^+$ ignores the indexes in the first column of the .gen file, therefore we do not need to adjust them. We have, however, to remember to change the total number of atoms in the first line from 408 to 407:

```
407  C
C    H
1    1     37.831463060000    -20.000000000000     0.710000000000
2    1     39.061219140000    -20.000000000000     1.420000000000
3    1     39.061219140000    -20.000000000000     2.840000000000
.....
46    1     32.912438770000    -20.000000000000     7.810000000000
47    1     30.452926620000    -20.000000000000     4.970000000000
49    1     31.682682700000    -20.000000000000     7.100000000000
50    1     30.452926620000    -20.000000000000     7.810000000000
...
```

The resulting structure should look like this:

We then also adjust the dftb_in.hsd file accordingly. As we have removed an atom, all the indexes in the transport block need to be adjusted properly. Note that we removed an atom in the first

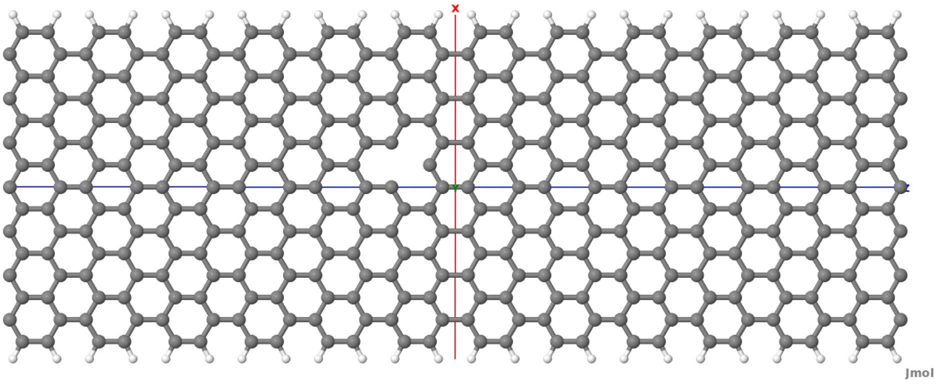

Fig. 4.7: Geometry with vacancy on sublattice A

PL of the extended device, therefore we also need to adjust the values of FirstLayerAtoms. The *Transport* block now reads:

```
Transport {
    Device {
      AtomRange = 1 135
      FirstLayerAtoms =  1 68
    }
    Contact {
      Id = "source"
      AtomRange = 136 271
      FermiLevel [eV] = -4.7103
      potential [eV] = 0.0
    }
    Contact {
      Id = "drain"
      AtomRange = 272 407
      FermiLevel [eV] = -4.7103
      potential [eV] = 0.0
    }
}
```

Compared to the pristine system, we have modified *AtomRange* in all the blocks and the values of *FirstLayerAtoms*.

After running the calculation, we can compare the transmission curve for this structure with a single vacancy and the pristine ribbon by using plotxy:

```
plotxy --xlabel 'Energy [eV]' --ylabel 'Transmission' -L --xlimits -6.5 -3 \
 ../ideal/transmission.dat transmission.dat &
```

Clearly, the presence of a vacancy introduces some finite scattering which reduce the transmission with respect to the ideal ribbon. In particular, the effect is quite small in the first conductance band while it is more visible in the first valence band and in higher bands. The reflection amplitude is increased near the band edges. This is expected in 1D systems, as near the band edges the density of states diverges (Van Hove singularities), hence the group velocity is lower, and it is known from semi-classical transport theory that the scattering probability is, when short range disorder is present, inversely proportional to the group velocity. The absence of resonant features in the transmission may point to the fact that the vacancy does not induce additional states in the conduction or valence bands. This can be verified by visualising the density of states, as in Figure *Non-SCC DOS for single vacancy in sublattice A (linear scale)* (page 43).

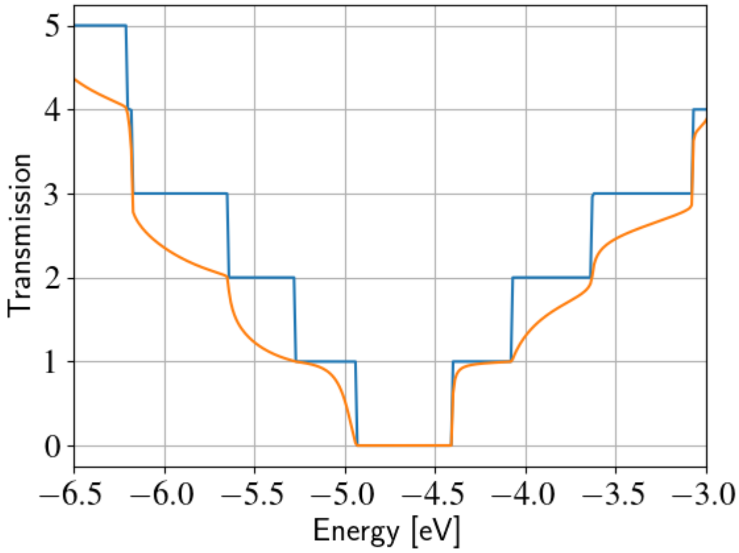

Fig. 4.8: Non-SCC transmission in pristine (blue) and single vacancy (red) ribbons

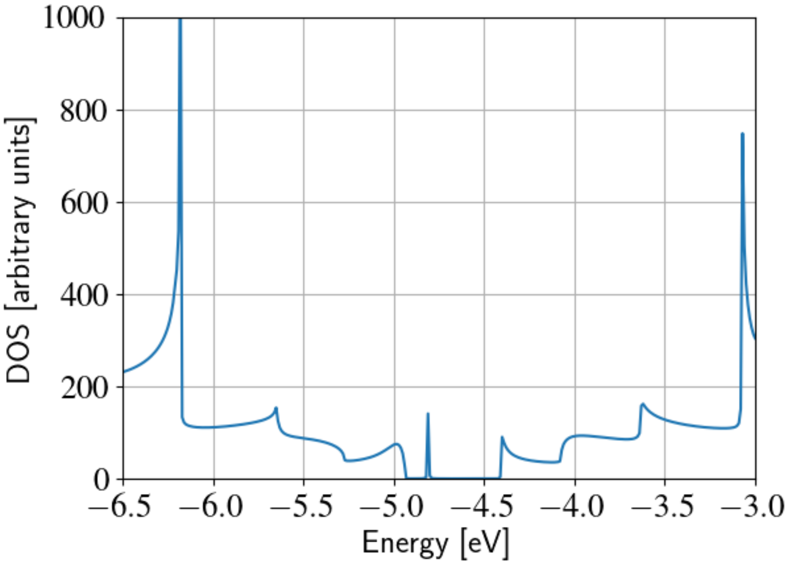

Fig. 4.9: Non-SCC DOS for single vacancy in sublattice A (linear scale)

The same density of states can be visualised on logarithmic scale as well, as in Figure *Non-SCC DOS for single vacancy on sublattice A (semilog scale)* (page 44).

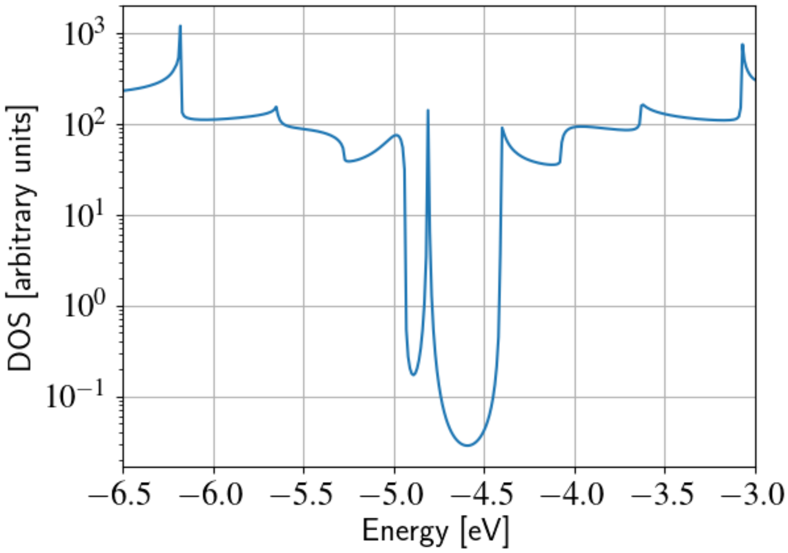

Fig. 4.10: Non-SCC DOS for single vacancy on sublattice A (semilog scale)

The vacancy is adding some close energy levels in the gap, as verified already from the *DFTB* calculation in the first part of the tutorial. The Van Hove singularities are partially suppressed as the system no longer possesses translational symmetry along the transport direction. Even in a simple non-SCC approximation, the qualitative picture is consistent with the previous SCC periodic calculation. We will now consider a vacancy sitting on the other sublattice (B) and try to understand whether the relative position of the vacancy is relevant or not by calculating once more the non-SCC transmission and density of states

4.3 Non-SCC armchair nanoribbon with vacancy (B)

[Working directory: *transport/agr_nonscc/vacancy2/*]

4.3.1 Transmission and Density of States

We will now consider a vacancy sitting on the other sublattice (B), i.e. we can take the structure file we used for the ideal ribbon and delete the atom number 47. The structure file is:

```
407  C
C    H
1    1     37.831463060000    -20.000000000000      0.710000000000
2    1     39.061219140000    -20.000000000000      1.420000000000
3    1     39.061219140000    -20.000000000000      2.840000000000
.....
46    1     32.912438770000    -20.000000000000      7.810000000000
48    1     31.682682700000    -20.000000000000      5.680000000000
49    1     31.682682700000    -20.000000000000      7.100000000000
50    1     30.452926620000    -20.000000000000      7.810000000000
.....
```

The *jmol* rendering of the geometry:

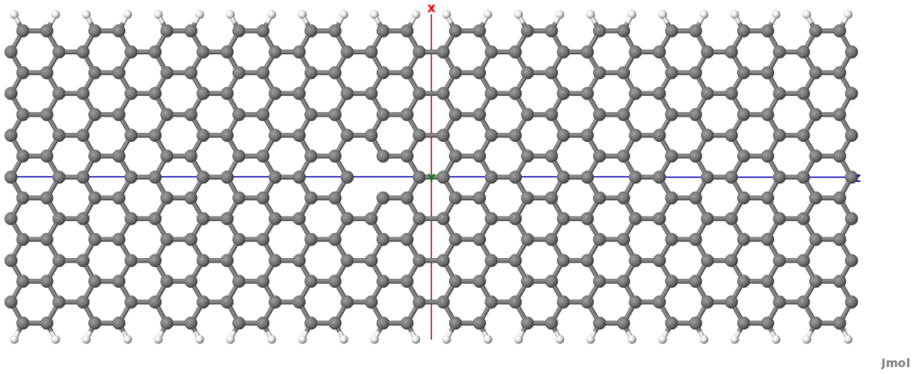

Fig. 4.11: Geometry with vacancy on sublattice B

Also in this case we remove an atom from the first PL of the extended device region, therefore the rest of the *dftb_in.hsd* input file is identical to the one we used for the vacancy on sublattice A. We can therefore just copy it and run the *dftb* calculation. The transmission is shown in Figure *Non-SCC transmission for vacancy B (green), pristine (blue) and vacancy A (red)* (page 46) (Transmission for vacancy on sublattice B in blue, transmission for vacancy on sublattice A in green and pristine system in green):

We can see a very strong suppression of transmission in the first sub-bands, especially in the first valence band. Again, the absence of resonances may be due by gap states. In fact, we can verify it by plotting the density of states, as shown in Figure *Non-SCC DOS for vacancy in sublattice B (semilog scale)* (page 46).

We can clearly see that the vacancy induces some nearly degenerate gap states, and that the density of states at higher energies is largely unaffected. It is known that the relative position of a scattering centre in a graphene nanoribbon with respect to different sub-lattices strongly affects its scattering properties, as is shown in these non-SCC calculation. Qualitatively, the picture is also consistent with periodic calculations, with the difference that we obtain directly information on the effect on transport properties via transmission function. This also ensures

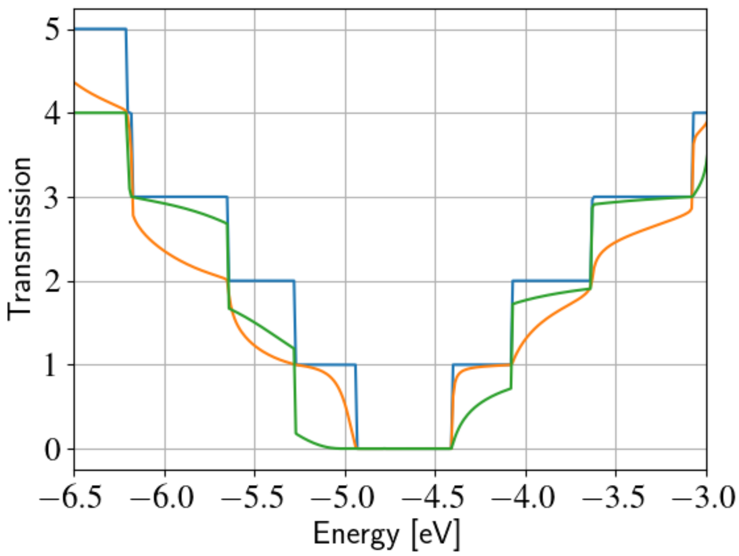

Fig. 4.12: Non-SCC transmission for vacancy B (green), pristine (blue) and vacancy A (red)

that we do not have to worry about choosing the right supercell or k-point sampling as the open boundary conditions represent exactly the infinite system with a single scattering centre. As already pointed out earlier, there is no warranty that a non-SCC calculation give the proper result in a system if relevant charge transfer is occurring, and in general it will not. Therefore in the next section we will repeat the same calculation by solving the SCC problem.

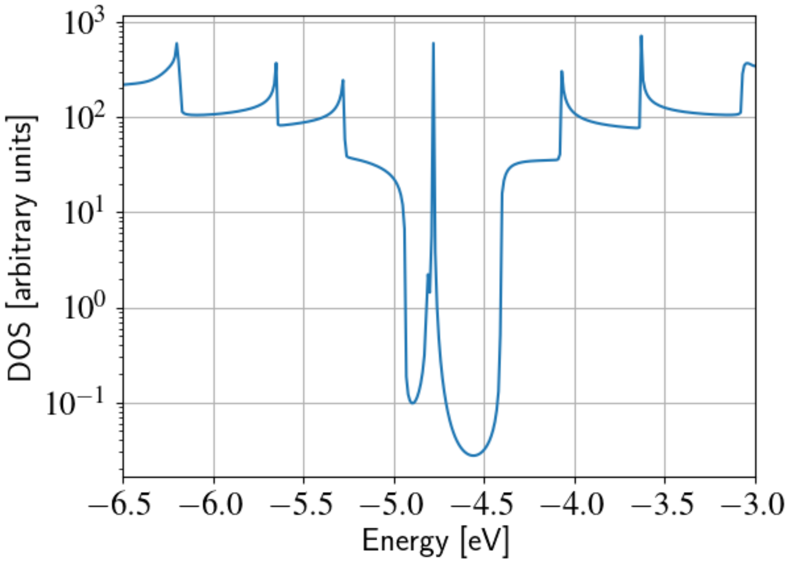

Fig. 4.13: Non-SCC DOS for single vacancy on sublattice B (semilog scale)

4.4 SCC Pristine armchair nanoribbon

A *DFTB* Hamiltonian is in general given by two terms:

$$H^{SCC} = H^0 + H^{shift}$$

Where the component H^{shift} is the self-consistent (SCC) correction. The SCC correction is in general needed whenever there is a finite charge transfer between atoms, i.e. whenever there are bonds between atoms with different chemical species or with different coordination numbers. In our case, we can expect a finite charge transfer between the C and H atoms at the edges, and an SCC component may be relevant. While in the previous sections, we have only considered the non-SCC component H^0, in the next sections we will compute the same calculation by including the correction given by the shifts H^{shifts}.

Note that the equilibrium SCC problem can be tackled in two ways: we could apply the Landauer-Caroli to an SCC Hamiltonian taken, for example, from a periodic calculation (i.e. uploading the SCC component), or we can solve the problem as a full NEGF setup with 0 bias. The code flow is currently such that this second procedure has to be used (however, the first technique will be available in future release). Therefore we will need to learn to set the input related to two other components of the NEGF machinery: the real space Poisson solver and the Green function density matrix.

In this way we will introduce a first complete input file. It is important, from a didactic point of view, to be clear that as long as the applied bias is zero and we are interested in equilibrium properties, the two approaches are equivalent and the results are only valid in the limit of linear response.

4.4.1 Contact calculation

[Working directory: *transport/agr_scc/contacts/*]

In order to run an SCC transport calculation, the code needs some additional knowledge about the contact PLs. In particular, the SCC shifts and Mulliken charges have to be saved somewhere to enable consistency between the calculation of the self-energy and the calculation of the Poisson potential. To this end, we have to introduce an additional step in the procedure: the contact calculation.

The contact calculation is simply a *DFTB* periodic calculation for the contact PL. As such, not all the field defined in the transport are meaningful and the input file will of course look different. The *Geometry* block is identical:

```
Geometry = GenFormat {
<<< 'device_7.gen'
}
```

While the *Transport* block needs to be modified as follows:

```
Transport {
    Device {
      AtomRange = 1 136
    }
    Contact {
      Id = "source"
      AtomRange = 137 272
    }
    Contact {
      Id = "drain"
      AtomRange = 273 408
    }
  Task = ContactHamiltonian{
     ContactId = "source"
  }
}
```

We first notice the addition of an option *Task =ContactHamiltonian{...}*, which was previously absent. This block specifies that we intend to calculate the bulk contact SCC properties, and the field *ContactId* specifies which contact we want to calculate. The field *FirstLayerAtoms* in the *Device* block is absent (it does not make sense in a contact calculation) and so are the fields *FermiLevel* and *Potential* in the two *Contact* sections, as they are not meaningful during this step. In general, the philosophy of a *DFTB* input file is that if input fields that would be useless or contradictory are present, the code will halt with an error message.

The Hamiltonian block shows some differences, too:

```
Hamiltonian = DFTB {
  SCC = Yes
  SCCTolerance = 1e-6
  EwaldParameter = 0.1
  MaxAngularMomentum = {
    C ="p"
    H = "s"
  }

  SlaterKosterFiles = Type2FileNames {
  Prefix = "../../../sk/"
  Separator = "-"
  Suffix = ".skf"
}

  KpointsAndWeights = SupercellFolding{
   25 0 0
    0 1 0
    0 0 1
    0.0 0.0 0.0
  }
}
```

The flags *SCC=Yes* and *SCCTolerance=1e-6* enable the SCC calculation. A small tolerance in the contact calculation, and in general in transport calculation, helps to avoid artificial mismatches at device/contact boundaries. The parameter *EwaldParameter* needs to sometimes be set when using parallel calculations to reduce the size of the neighbour list. Typically, the code may complain about a too small parameter: in that case, setting a value of 0.1 is considered to be good practice. The other parameters are the usual ones, except for the *KPointsAndWeight*, which deserves special attention.

The bulk contact is of course a periodic structure, hence we need to specify a proper k-point sampling, as we would do in a regular periodic *DFTB* calculation. However, you should be careful about the way the lattice vector is internally defined. In the input system is a cluster (C), i.e. *it has no periodicity in direction transverse to the transport directions*, the lattice vector of the contact is internally reconstructed and assigned to be the first lattice vector, *regardless the spatial orientation of the structure*. This means that the *KPointsAndWeights* for a cluster system are always defined as above: a finite number of k-points along the first reciprocal vector (according to a 1D Monkhorst-Pack scheme) and a Gamma point sampling along the other two directions. The reason for this choice is that we do not want to assign a specific direction to the structures, i.e. at this level we do not assume in any way that the structure must be oriented along x,y or z direction.

Note also that as the contact information is used in the transport calculation, it is a good idea to use a dense k point sampling and a low SCC tolerance, in order to get a very well converged solution. The contact calculation will be usually much faster than the transport calculation, so this does not usually present a problem.

On the other hand, this rule regarding k-points does not apply to periodic transport calculations, as the periodicity along the transverse directions must also be preserved (refer to the following section for a periodic system example). We can run the calculation by typing:

```
dftb+ dftb_in.hsd |tee output
```

After running the calculation, we notice that a file *shiftcont_ source.dat* is generated. This file contains the information useful for the transport calculation (shifts and charges of a bulk contact). It is suggested you also keep a copy of the *detailed.out* for later reference. We can obtain the value of the Fermi energy, which we will later need, from *detailed.out* as -4.7103 eV.

We can now run the same calculation for the drain contact by just modifying the *Task* block:

```
Task = ContactHamiltonian{
     ContactId = "drain"
  }
```

The contact are identical, therefore we expect the same results, also with the same Fermi energy. We now have a file *shiftcont_ drain.out*, which is equivalent to *shiftcont_ drain.dat* apart from small numerical error. In fact, we could have simply copied the previous contact results into this file.

Now that the contact calculation is available, we can set up the transport calculation.

4.4.2 Transmission and Density of States

[Working directory: *transport/agr_ scc/ideal/*]

In order to calculate the transmission for the SCC system, we have to copy the files *shiftcont_ drain.dat* and *shiftcont_ source.dat* into the current directory:

```
cp ../contacts/shiftcont* .
```

Then, we have to specify some additional blocks with respect to a non-SCC calculation. We first look at the *Transport* block.:

```
Transport {
  Device {
    AtomRange = 1 136
    FirstLayerAtoms =  1 69
  }
  Contact {
    Id = "source"
    AtomRange = 137 272
    FermiLevel [eV] = -4.45
    potential [eV] = 0.0
  }
  Contact {
    Id = "drain"
    AtomRange = 273 408
    FermiLevel [eV] = -4.45
    potential [eV] = 0.0
  }
  Task = UploadContacts {
  }
}
```

The atom indices are of course the same, as the geometry of the system is not changed. This time though, we explicitly specified a *Task* block named *UploadContacts*, which declares that we are now running a full transport calculation. *Task=UploadContacts{}* is the default and does not take any additional parameters, therefore you can safely omit it.

Now that we are solving the full SCC scheme, we will allow for charge transfer between the open leads and the extended device region, therefore it is important to set a well-defined Fermi energy. While this does not make any difference in a non-SCC transmission calculation, it is crucial for the SCC calculation. A wrong or unphysical Fermi energy will lead to unphysical charge accumulation or depletion in the system.

To this end, you will have to pay some attention to the definition of the Fermi energy. As we are calculating a semiconductor system, the Fermi level should be in the energy gap. By calculating a band structure or by inspection of the eigenvalues in the file *detailed.out* you can verify that the value -4.7103 is on the edge of the conduction band. This can be explained as numerically the Fermi level is defined by filling the single particle states till the reference density is reached, therefore its position inside the gap of a semiconductor is arbitrary. Therefore, while in metallic system we may ensure consistency and use a well calculated Fermi level at some specific temperature during all our transport calculation, in the case of a semiconductor system we can manually set the Fermi level in the middle of the energy gap (for this system, roughly at -4.45 eV) and freely vary the temperature as long as the gap is larger than several times the value of kT.

We will see in the following that there are some ways to verify that the Fermi level is defined consistently, as this is often source of confusion. Note also that, differently from other codes, *dftb+* allows for different Fermi levels in different contacts, which can be useful when heterogeneous contacts are defined (for example, in a PN junction). In that case a built-in potential is internally added to ensure no current flow at equilibrium.

In the *Hamiltonian* block now an SCC calculation has to be specified:

```
Hamiltonian = DFTB {
  SCC = Yes
  SCCTolerance = 1e-6
  ReadInitialCharges = No
  MaxAngularMomentum = {
    C = "p"
    H = "s"
  }

  SlaterKosterFiles = Type2FileNames {
  Prefix = "../../../sk/"
  Separator = "-"
  Suffix = ".skf"
  }
  ....
```

MaxAngularMomentum and *SlaterKosterFiles* are not modified. But differently from the non-SCC calculation, we now need to specify a way to solve the Hartree potential and the charge density self-consistently. In a NEGF calculation, we use a real-space Poisson solver to calculate the potential, and a Green function integration method to calculate the density matrix:

```
...
Electrostatics = Poisson {
  Poissonbox [Angstrom] = 40.0 30.0 30.0
  MinimalGrid [Angstrom] = 0.5 0.5 0.5
  SavePotential = Yes
}

Eigensolver = GreensFunction {
  }

Mixer = Broyden {
   MixingParameter = 0.02
 }
}
...
```

The Poisson section contains the definition of the real space grid parameters. Note that differently from a normal *dftb+* calculation, simulating regions of vacuum is not for free, as the simulation domain must be spanned by the real space grid. The grid is always oriented along the orthogonal cartesian coordinate system. *Poissonbox* specifies the lateral length of the grid. The length along the transport direction is ignored as it is automatically determined by the code (in this case, z=30.0). The length along the transverse direction are relevant and *should be carefully set.* In order not to force unphysical boundary conditions, you may extend the grid at least 1 nm away. If a strong charge transfer is present, you may go for a larger grid according to your available computational resources. A poorly defined grid can lead to no convergence at all, to a very strange (and slow) convergence path or to unphysical results. *MinimalGrid* specifies the minimum step size for the multigrid algorithm. Values between 0.2 and 0.5 are usually good, where a lower value stands for higher precision. *SavePotential = Yes* will return a file containing the potential and charge density profile, for later reference. These files can be quite large, therefore the default is *No*.

The Eigensolver is now specified as *GreensFunction*. With this definition, we instruct the code not to solve an eigenvalue problem but rather to calculate the density matrix by integration of the Keldysh Green function. This block provides the SCC charge density with or without applied bias. The options define the integration path. Usually the default options are good enough in most cases and advanced users may refer to the manual and references therein.

The *Mixer* options is present in *dftb* calculations as well. Convergence is known to be critical in NEGF schemes. In that case, a lower *MixingParameter* value will help to avoid strong oscillation in the SCC iterations.

The last block is *Analysis*:

```
...
Analysis = {
  TransmissionAndDos {
    Verbosity = 101
    EnergyRange [eV] = -6.0  -3.0
    EnergyStep [eV] = 0.01
  }
}
```

This block is identical to the non-scc calculation as the same task is performed: calculation of transmission, current and DOS by using the Landauer-Caroli formula. The transmission will be of course be different due to the fact that the ground state charge density is now solution of the SCC Hamiltonian and we have slightly changed the energy range as the SCC component introduce a shift of the band-structure (try to compare the SCC and non-SCC transmission results when you are done). We can now run the calculation (after defining the directories GS and contacts):

```
mkdir GS
mkdir contacts
dftb+ dftb_in.hsd |tee output
```

or:

```
mpirun -n 4 dftb+ dftb_in.hsd |tee output
```

Where *-n 4* should be adapted to the number of available nodes. As transport calculations in *dftb+* are parallelised on energy points, a quantity larger than 40 (the default number of integration points at equilibrium) will not speed up the calculation of the density matrix.

An inspection of the file *detailed.out* reveals that we have additional information with respect to the non-SCC calculation, including a list of atomic charges and orbital population, as now the SCC density matrix has been calculated. The transmission is also saved as separate file, and is shown in Figure *SCC (red) and Non-SCC (blue) transmission through pristine AGR* (page 52).

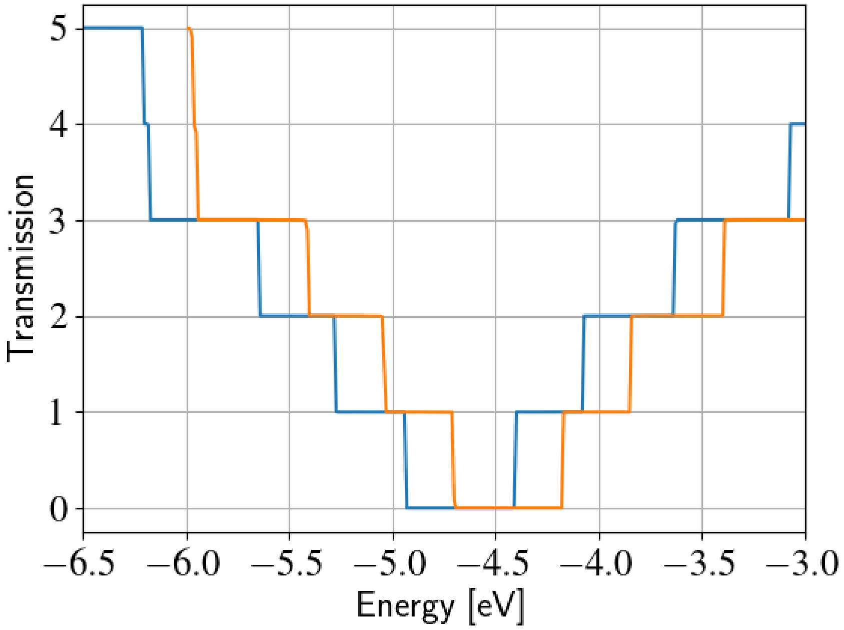

Fig. 4.14: SCC (red) and Non-SCC (blue) transmission through pristine AGR

As you'd expect, it still step-like as in the non-SCC calculation. This is correct, as we're calculating an ideal 1D system. The bandwidth (i.e., the steps width) may differ due to SCC contribution and the overall transmission is shifted. Note that while the non-SCC calculation is very robust, meaning that you will always get step-like transmission for a 1D system, in the SCC calculation a poor definition of the boundary conditions, of the bulk contact properties or of the additional *GreensFunction* and *Poisson* blocks may induce numerical artifacts and scattering barriers which should not be there. As a result, the transmission will not appear step-like but rather visibly smoothed out.

You can also verify the quality of the calculation by inspection of the potential and charge density profiles. In a pristine periodic system we would expect a periodic potential, without discontinuities at the boundary between extended device and electrodes. The information needed to construct the real space potential and charge density are contained in 5 files: *box3d.dat*, *Xvector.dat*, *Yvector,dat*, *Zvector.dat*, *potential.dat* and *charge_ density.dat*. The first 4 files contain the grid information, and the last two ones the list of potential and charge density values (following a row major order). Those information can be converted to any useful with some simple scripting, we provide an utility called *makecube* which can be used to convert them to Gaussian *cube* format or a more flexible *vtk* format. There's plenty of software to visualise *vtk* or *cube* files, but unluckily at present current choices of software which are effective at visualising real space grid data are weak at visualising atomistic structures, and vice versa. In the following we will use *paraview* and work with the *vtk* format. Paraview is freely available and is supplied with many gnu/linux distributions as a compiled package.

The *vtk* file can be obtained by simply running:

```
makecube potential.dat pot.vtk
```

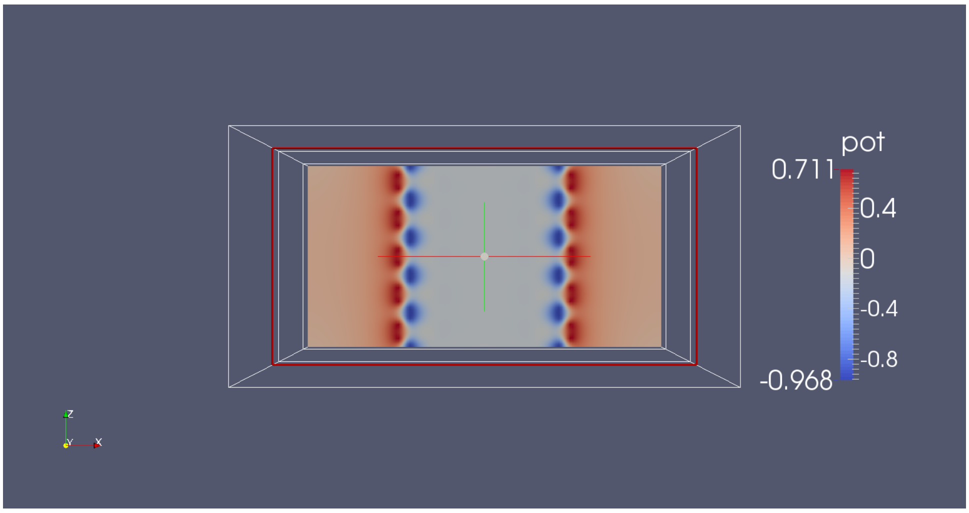

Fig. 4.15: Potential profile along the nanoribbon

An extensive explanation of *paraview* features is beyond the scope of this tutorial. Following some easy steps, you can produce the potential map shown in Figure *Potential profile along the nanoribbon* (page 53).

1. Open paraview and import the file *pot.vtk* from File->Open
2. Click on Properties -> Apply (Properties are usually visualised on the left side of the screen) and you should see the bounding box in the visualisation windows.
3. In the Pipeline browser select the file *pot.vtk* by clicking once on it, and then select the Clip filter from Filters -> Alphabetical (or from the filter toolbar).
4. In Properties, click on ‘Y Normal’ to produce a clip along the nanoribbon.
5. Click on Properties -> Apply.

The plot shown in Figure *Potential profile along the nanoribbon* (page 53) above is the self-consistent potential along the nanoribbon. We can see that the charge transfer between carbon and hydrogen at the edges results in a non-flat potential. At a first glance, the potential looks quite homogeneous, meaning that there are no clear discontinuities at the box boundary. This is important: being it a homogeneous ribbon, the potential should have the same periodicity as the lattice. We can verify this with a closer inspection by plotting a cut along a line. We apply the following steps:

1. We select *pot.vtk* in the Pipeline Browser and Filters->Alphabetical->Plot Over Line
2. From the Properties window, we select 'Z Axis' and click on 'Apply'

By following this procedure we obtain Figure *Potential profile along the nanoribbon* (page 54).

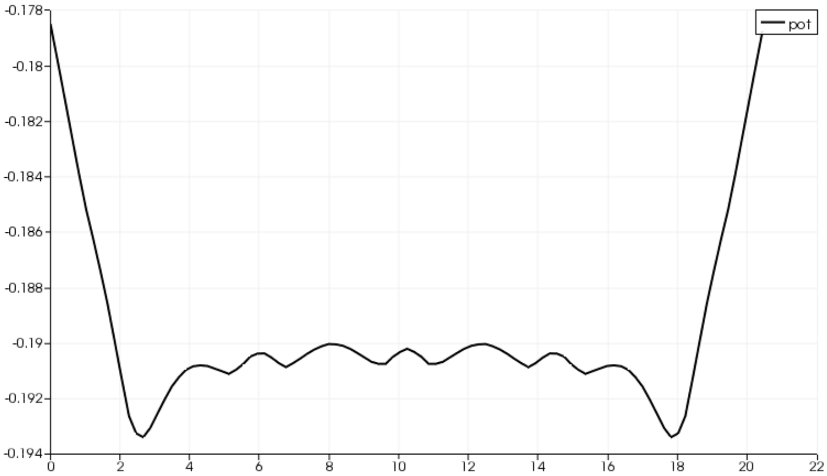

Fig. 4.16: Potential profile along the nanoribbon

As you can notice, there is a discontinuity at the interface. However, it is quite small (∼ 12 meV). Defining a 'perfect' interface between the bulk semi-infinite contacts and the device region is very difficult, especially in a semiconductor where no free charge can contribute to screen such an interface potential. A smaller tolerance in the self-consistent charge during the contact and the device calculation, a finer calculation of the Fermi level (in metallic systems) and a finer Poisson grid can decrease the discontinuity: you should be able to reach about 1 meV, but it is difficult to go below this value. However, as you can see in the transmission plot, as long as the discontinuity is this small, it hardly affects the transmission.

However, it is important for you to verify that the behaviour at the boundaries is reasonable. Otherwise, the extended region may be too small to allow to the relevant physical quantities (charge, potential) to relax to bulk values. Be aware that numerical errors are unavoidable, therefore it is important to understand their relevance and the impact on the results. In the transmission calculation we do not notice anything different because the energy step is close to the mismatch at the boundaries.

After running the calculation for the pristine system, we will introduce vacancies as we did in the non-SCC calculation. The results should be now directly comparable to the bulk periodic SCC *dftb* calculation.

4.5 SCC armchair nanoribbon with vacancy (A)

[Working directory: *transport/agr_scc/vacancy1/*]

We will now calculate the SCC transmission for the nanoribbon with a vacancy on the sublattice

A, using the same input structure set up for the non-SCC calculation. The contacts are identical to the pristine case, therefore in the following we will only modify the extended device calculation.

4.5.1 Transmission and Density of States

As previously done, the transport section must be modified in order to account for the different number of atoms in the extended device region:

```
Transport {
    Device {
      AtomRange = 1 135
      FirstLayerAtoms =  1 68
    }
    Contact {
      Id = "source"
      AtomRange = 136 271
      FermiLevel [eV] = -4.45
      potential [eV] = 0.0
    }
    Contact {
      Id = "drain"
      AtomRange = 272 407
      FermiLevel [eV] = -4.45
      potential [eV] = 0.0
    }
  Task = UploadContacts {
  }
}
```

We use the same Fermi level and the files *shiftcont_ source.dat* and *shiftcont_ drain.dat* as in the pristine system calculation, as the contacts are not modified.

The *Hamiltonian* block is also not modified, except for an additional finite temperature:

```
Hamiltonian = DFTB {
  SCC = Yes
  SCCTolerance = 1e-6
  ReadInitialCharges = No
  MaxAngularMomentum = {
    C = "p"
    H = "s"
  }
  SlaterKosterFiles = Type2FileNames {
  Prefix = "../../../sk/"
  Separator = "-"
  Suffix = ".skf"
  }
  Filling = Fermi {
    Temperature [Kelvin] = 150.0
  }
  Electrostatics = Poisson {
    Poissonbox [Angstrom] = 40.0 40.0 30.0
    MinimalGrid [Angstrom] = 0.5 0.5 0.5
    SavePotential = Yes
  }
  Eigensolver = GreensFunction {
    }
}
```

A finite temperature is used to provide a finite temperature broadening, useful if the vacancy induces partially filled gap states. In general, temperature broadening may improve convergence and dump oscillations in the SCC iterations.

The *Analysis* block is also similar, we add the DOS calculation to verify if we can identify a vacancy state:

```
Analysis = {
  TransmissionAndDos {
      Verbosity = 101
      EnergyRange [eV] = -6.0  -3.0
      EnergyStep [eV] = 0.025
      Region = {
        Atoms = 1:135
      }
  }
}
```

As usual, you can now create the *GS* and *contacts* directories, copy the *shiftcont_source.dat* and *shiftcont_drain.dat* in the current directory and run the calculation. The density of states and transmission are shown in Figure *SCC DOS for single vacancy on sublattice A (semilog scale)* (page 56) and *SCC transmission in pristine (blue) and single vacancy (red) ribbons* (page 57).

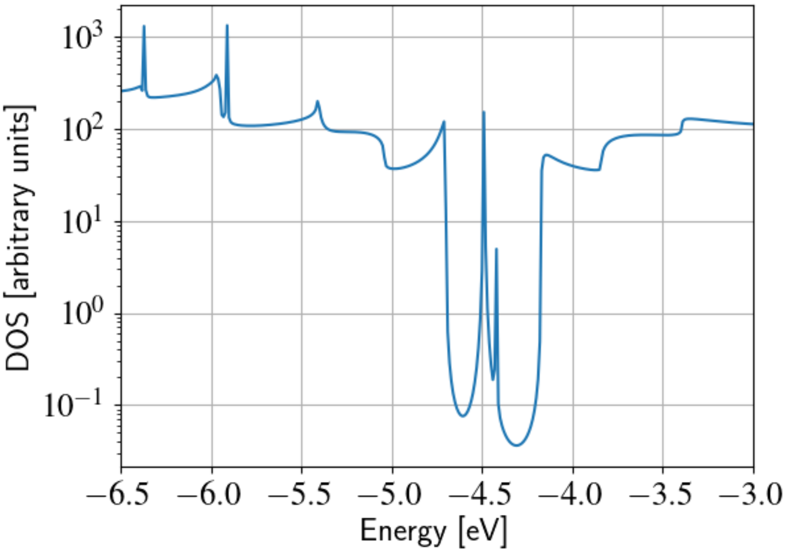

Fig. 4.17: SCC DOS for single vacancy on sublattice A (semilog scale)

The vacancy states are located in the energy gap, consistently with the periodic calculation, and that the transmission curve is qualitative similar to the non-scc calculation. The first conduction and valence band are weakly affected by the vacancy which does not act as a strong scatterer. There is no signature of resonances, as the additional levels are located in the gap.

Note also that we previously recommended the use of large extended regions and to verify that the potential and charge density are smooth at interfaces. As you can see in Figure *Potential profile for vacancy (A)* (page 57), the impurity is very close to the boundaries, resulting to a potential profile which varies significantly close in to the boundary. It is left to the reader to verify that the overall transmission does not change significantly if a longer extended region is considered.

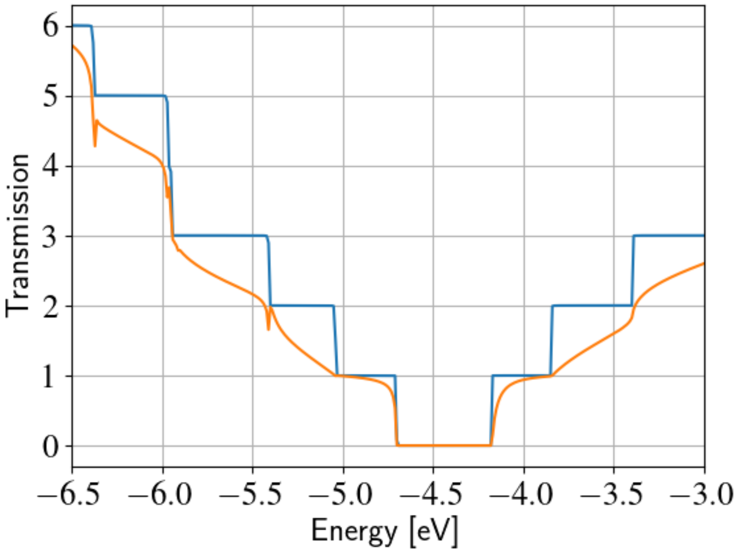

Fig. 4.18: SCC transmission in pristine (blue) and single vacancy (red) ribbons

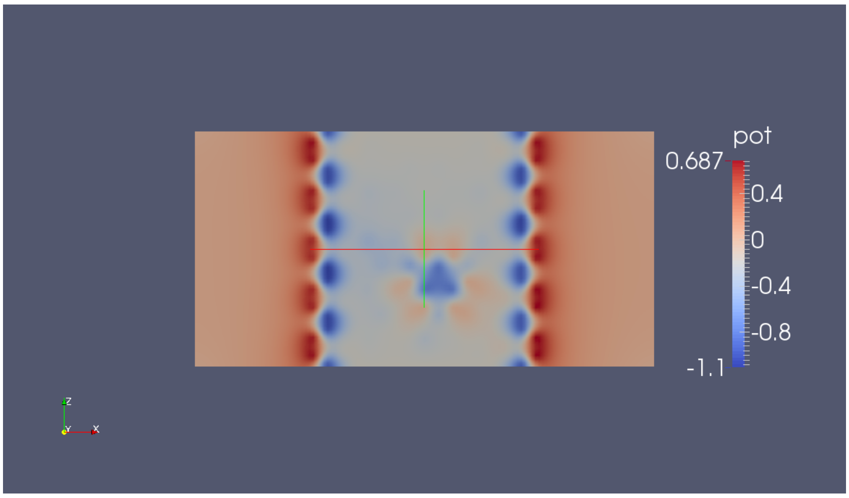

Fig. 4.19: Potential profile for vacancy (A)

4.6 SCC armchair nanoribbon with vacancy (B)

[Working directory: *transport/agr_scc/vacancy2/*]

We will now run the same calculation, but with the vacancy on the sublattice B. As in the non-SCC case, the only difference with the previous calculation is the location of the vacancy, therefore the input file is absolutely identical. The contacts are the same, therefore all we have to do is copy the *shiftcont_source.dat* and *shiftcont_drain.dat* files into the current directory and run the calculation.

The resulting transmission and density of states are shown in Figures *SCC DOS for single vacancy on sublattice B (semilog scale)* (page 58) and *SCC transmission for single vacancy on sublattice B* (page 59).

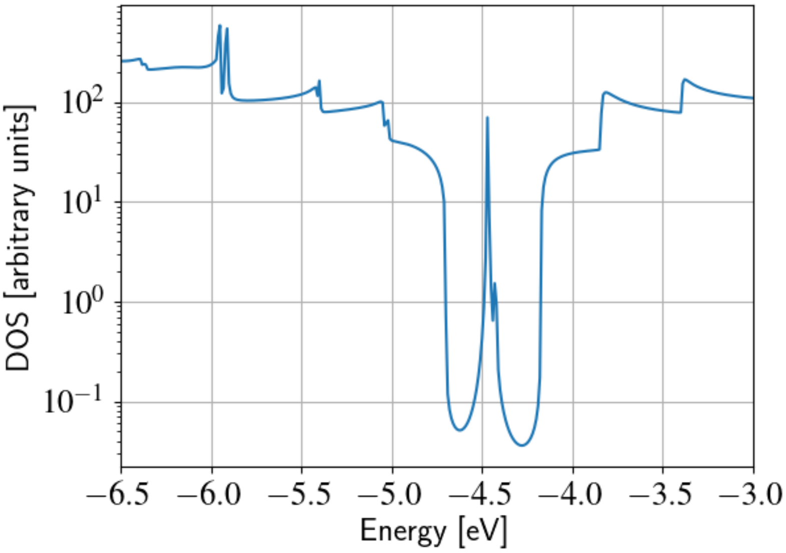

Fig. 4.20: SCC DOS for single vacancy on sublattice B (semilog scale)

We immediately notice that the Van Hove singularities are strongly suppressed and that the valence band is almost completely suppressed. Consistently with the picture obtained by periodic calculation, a quasi-bounded vacancy level hybridise with the valence band edge causing a strong back-scattering. A comparison between all the three cases (see Figure *SCC transmission for vacancy B (green), pristine (blue) and vacancy A (red)* (page 59)) shows that the scattering probability is deeply affected by the exact position of the vacancy. This is, in graphene nanoribbon, generally true for other kinds of short range scattering centres such as substitutional impurities. We can also notice that, in this particular case, the non-scc approximation is qualitatively consistent for two reasons: the vacancy level are not populated and the charge transfer at the edges is not critical as the edges contribute poorly to the transmission in an armchair ribbon.

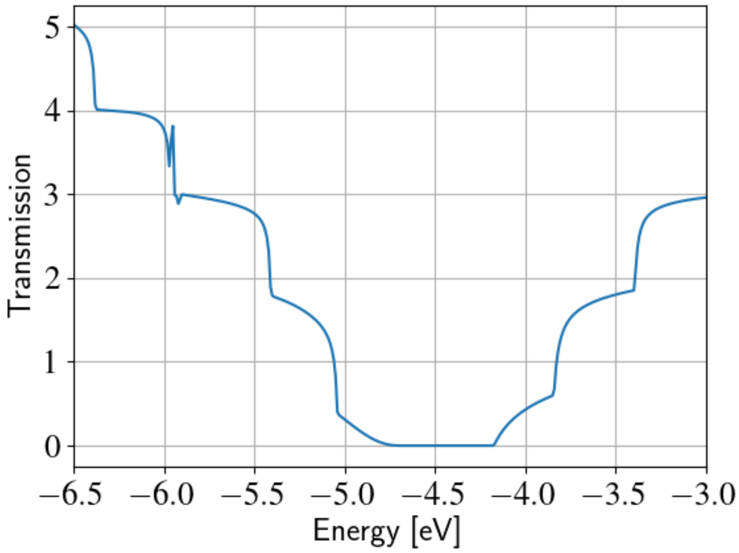

Fig. 4.21: SCC transmission for single vacancy on sublattice B

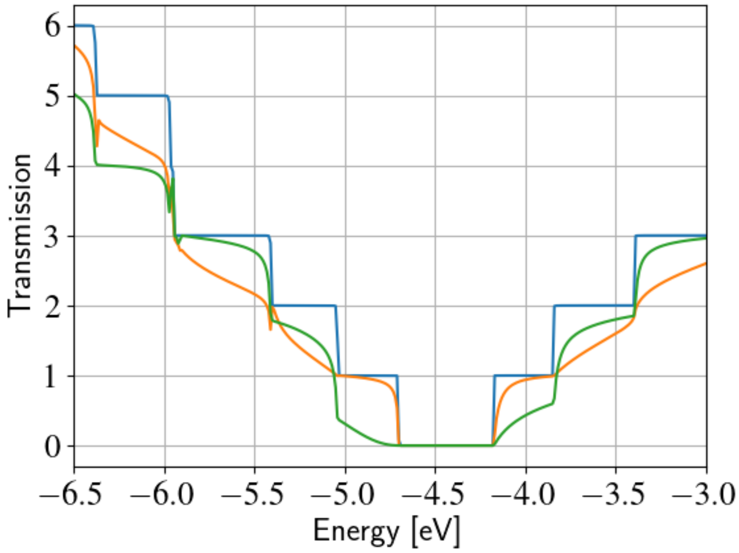

Fig. 4.22: SCC transmission for vacancy B (green), pristine (blue) and vacancy A (red)

Part III

Examples

Part IV

Developer guide

CHAPTER 5

TraNaS library

5.1 TTraNaS type

Here we describe the structure of the global container TTraNaS.

- Type(TTraNaSInput) :: input – Input container. Is not changed inside the library.
  - logical :: tPhotons = .false.
  - type(TPhotons) :: photons
    - integer :: NumberModes = 0
    - real, dimension(:), allocatable :: Frequencies
- Type(TNGF) :: ngf – Container for Green function methods.
  - type(TMBNGF) :: mbngf
    - logical :: tHartreeFock = .false.
    - logical :: tRPA = .false.
    - type(TStruct_info) :: str – System partitioning (as in negf%str).
    - type(z_DNS), dimension(:,:), allocatable :: SelfEnergyR_HF
    - type(z_CSR), dimension(:), allocatable :: GreenFunctionR
    - type(z_CSR), dimension(:), allocatable :: PolarizationOperatorR
    - real(dp) :: scc_tol = 1.0d-7 – SCC Tolerance.
    - procedure :: add_SelfEnergyR
    - procedure :: add_SelfEnergyR_HF
    - procedure :: get_SelfEnergyR_HF
    - procedure :: get_SelfEnergyR_RHF
  - type(TTDNGF) :: tdngf
- Type(TQME) :: qme
- Type(TNEGF) :: negf – global container of the LibNEGF library.

Part V

Appendices

License

This DFTB$^+$XT Guide is licensed under the Creative Common Attribution 4.0 International license.

You are free to:

- **Share** – copy and redistribute the material in any medium or format
- **Adapt** – remix, transform, and build upon the material

for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution You must give **appropriate credit**, provide a link to the license, and **indicate if changes were made**. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions – You may not apply legal terms or **technological measures** that legally restrict others from doing anything the license permits.

Bibliography

[1] A. Pecchia, G. Penazzi, L. Salvucci, and A. Di Carlo, "Non-equilibrium Green's functions in density functional tight binding: method and applications," New Journal of Physics **10,** 065022 (2008).

[2] M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, T. Frauenheim, S. Suhai, and G. Seifert, "Self-consistent-charge density-functional tight-binding method for simulations of complex materials," Phys. Rev. B **58,** 7260 (1998).