KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Energy filtering in nanowire transistors with a geometric superlattice

Arnout Beckers

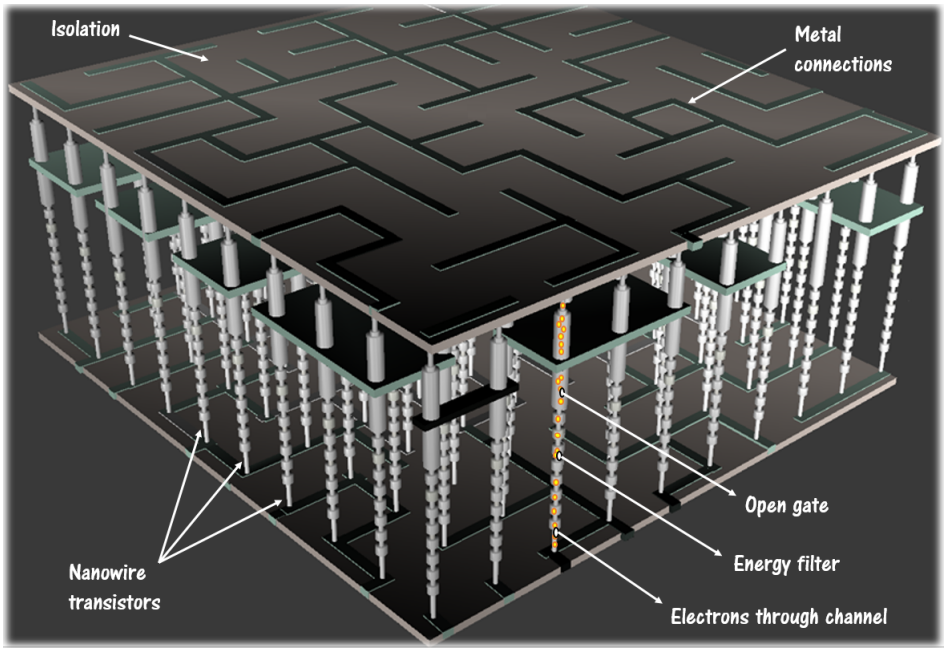
Thesis submitted for the degree of Master of Science in Nanoscience and Nanotechnology, option Nanoelectronic Design

Thesis supervisor:
Prof. dr. ir. Bart Sorée

Assessors:
Prof. dr. ir. Marc Heyns
Prof. dr. Chris Van Haesendonck

Mentor:
Ir. Maarten Thewissen

Academic year 2015 – 2016



Impression of a chip with energy filtering nanowire transistors.

Preface

I would like to express my sincere gratitude to my supervisor Maarten Thewissen for his guidance during the semester at IMEC as well as his remote guidance during my semester abroad. Maarten, I thank you for teaching me many things during the year, ranging from quantum physical insights to useful programming skills.

I would like to thank my promotor Bart Sorée for his feedback on my work and for giving me the opportunity to work remotely on the first part of my master thesis.

I am grateful to the members of the Physics, Modelling and Simulation-group at IMEC and other PhD students for their feedback and questions during my presentations. I thank Maarten Van de Put and Devin for their explanation on the whiteboard related to my thesis work.

I thank my neighbours at my work spot at IMEC, with a special thanks to Marta and Chidharth for their enthusiasm and interest in my simulation results. Marta and Chidharth, I have enjoyed explaining my thesis work to you and I hope this thesis text can do a better job in explaining everything you wanted to know.

Finally, I would like to thank my family and friends for their encouragements and support.

Arnout Beckers

Contents

Abstract

Energy leakage is nowadays becoming a fundamental problem in the chip's energy household. Energy leakage is especially a problem in chips targeted for devices with a long standby time, such as future Internet of Things (IoT) devices. Future IoT devices point to 'always on, always connected'-devices which will consume a lot of leakage energy. Energy leakage in chips is detrimental for the IoT technology because of three reasons. First, energy leakage drains the limited energy supply of the IoT chips, probably in the form of a battery or with energy harvesting. Second, energy leakage results in unwanted heat dissipation, which can be a potential problem in IoT chips worn close to the human skin or incorporated in food packages. Third, energy leakage in mass produced IoT chips is likely to result in a global waste of energy. Furthermore, to counteract the issue with privacy and security associated with IoT, a recent trend is under research to compute more of the raw data locally in the chip, instead of sending the data directly to the cloud for computation. As a consequence, the performance of the IoT chip in executing computational tasks needs to meet a certain minimum performance without deteriorating the power consumption. Steep slope transistors in IoT chips allow to lower the power consumption while keeping the required performance.

In this thesis we investigate the nanowire transistor with a geometric superlattice as a viable steep slope transistor concept. A geometric superlattice is any kind of periodic feature along the transport direction of the nanowire. A geometric superlattice in the nanowire transistor has the potential to reduce the energy leakage by filtering out the high energy electrons causing the leakage current in the chip. We model a nanowire transistor with a geometric superlattice using a continuous quantum approach in three dimensions by extending the quantum transmitting boundary method to a general three-dimensional device geometry. We proof the existence of minibands and minibandgaps in the transmission spectrum of the nanowire with a geometric superlattice. By changing the periodic features of the geometric superlattice in the simulations, we show the possibility of engineering the minibands and minibandgaps in the transmission spectrum such as to filter out the high energy electrons and to lower the leakage current. We propose the Si nanostrip geometric superlatticeFET as a steep slope transistor concept with close to ideal energy filtering characteristics and a switching slope of sub-60 mV per decade.

Keywords - Steep slope transistors, geometric superlattice, nanowires, passive

power consumption, energy filtering, QTBM, ballistic, Internet of Things.

List of Figures

List of Abbreviations and Symbols

Abbreviations

BTBT	Band-to-band tunneling
CCS	Crystal Coordinate System
DCS	Device Coordinate System
DFT	Density Functional Theory
DOS	Density Of States
EBL	Electron Beam Lithography
ECS	Ellipsoidal Coordinate System
VCS	Valley Coordinate System
EMA	Effective Mass Approximation
EVP	Eigenvalue Problem
IoT	Internet of Things
LCS	Lead Coordinate System
MEMS	Micro-Electromechanical Systems
MOSFET	Metal Oxide Semiconductor Field-Effect Transistor
NEGF	Non-Equilibrium Green's Functions
PDE	Partial Differential Equation
QTBM	Quantum Transmitting Boundary Method
TFET	Tunnel Field-Effect Transistor
1DEG	1-Dimensional Electron Gas

Symbols

a	complex amplitude of incoming wave
b	complex amplitude of outgoing wave
D	drain
E	energy
e	fundamental electron charge
G	gate
h	Planck's constant
$\hbar$	reduced Planck's constant
I	current
I_{on}	on-state current
I_{leak}	leakage current
i	complex
l	lead (number)
l'	injection lead (number)
m	subband number
m'	injection subband number
m_e	fundamental electron mass
m^*	effective electron mass
M^*	effective electron mass tensor
S	source
SS	subthreshold slope
ξ	local x-coordinate axis in lead
η	local y-coordinate axis in lead
ζ	local z-coordinate axis in lead
ψ	wavefunction
ϕ	basis function
Ω_i	region i
$\partial\Omega_i$	boundary of region i

Chapter 1

Introduction

1.1 Problem statement

1.1.1 Electronic society

The use of electronic devices has become mainstream in today's society. Electronic devices largely determine the way we work, communicate and obtain information. Over a few decades, electronic devices have been gaining steadily in functionality and portability. This increased functionality and portability is clearly visible with the laptops, smartphones and tablets which most of us use on a daily basis. However, these examples only include the consumer electronics. A larger part of the electronics is hidden away in objects like cars, traffic lights, medical appliances or industrial sensors. The hidden electronics enable a variety of applications to run smoothly and efficiently. An even wider variety of applications, closer to people's every day lives, can be envisioned when more every day objects become equipped with hidden electronics and connected to the Internet. This will enable a new generation in information and communication technology: the Internet of Things.

1.1.2 Internet of Things

In a society with Internet of Things (IoT), the hidden electronics will become even more ubiquitous than today. The hidden electronics will come in clothing, buildings, autonomous cars, food packages etc. The hidden electronics will sense the surroundings and extract raw data from it. The raw data can then be communicated over the Internet to other hidden electronics, to consumer electronics or to the cloud, depending on the application. By doing computations on the raw data, the raw data is then transformed into valid pieces of information useful for monitoring health[23] improving energy efficiency in buildings[24], monitoring traffic[49] or reducing waste in the food supply chain[44]. The leading vendor of embedded electronics ARM expects 30 billion of connected IoT devices by 2020[2]. An abundance in connected devices is a vital asset for the IoT technology, as a lot of raw data from different places is needed to reconstruct valid pieces of information. In a society with IoT, the more hidden electronics, the merrier.

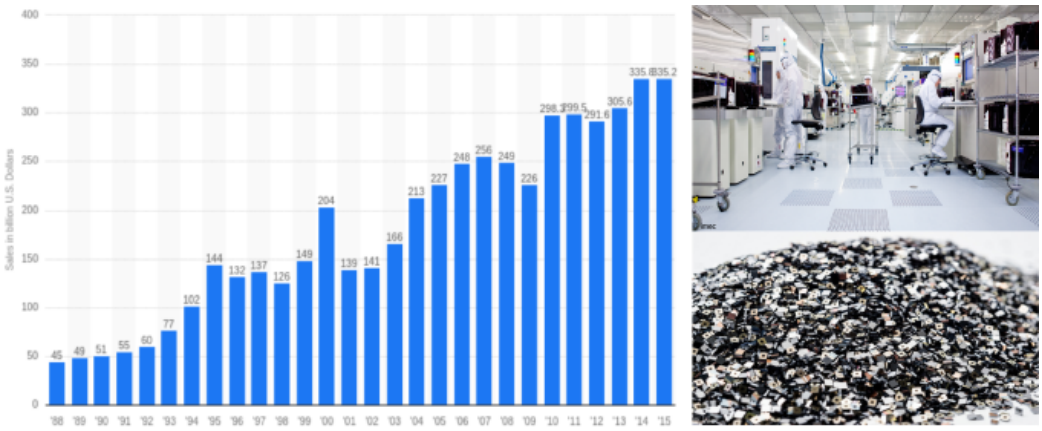


Figure 1.1: Left: Chip sales in billion US dollars from 1988 until 2015[42]. Top right: Processing of chips in a cleanroom on a chain of expensive tools ©imec. Bottom right: A pile of produced chips ©imec.

1.1.3 Mass production of chips

At the heart of all the hidden electronics and consumer electronics are the chips. Chips sales have risen to a 300 billion dollar market in 2015, as shown in figure 1.1. However, to connect clothes, cars, buildings etc. in a society with IoT, the production and sales of chips should keep increasing in the coming years. To enable a society with IoT, chips should become a real commodity, even more than today. However, chips are produced in cleanrooms with highly automated processes on a chain of very expensive tools. To produce a given amount of chips on a wafer, it takes around three to four months for the wafer to run through all the process steps. The throughput of chips at a given cost and time is limited by the number of chips which fit on the wafer. To increase chip throughput at a fixed wafer size, a solution is to make the chips on the wafer smaller. To retain the same functionality with smaller chips, the features on the chip have to become smaller. As a result, the transistors on the chip have scaled to 14 nanometer in 2015[3], enabling a higher transistor density on smaller chips. Today, researchers already explore the 10 nanometer node (and beyond) to take the mass production of chips to the next level and make an IoT technology possible.

1.1.4 Energy crisis

Current chips consume too much energy. Chips consume their available energy in two ways: actively, due to the execution of computational tasks, and passively, due to energy leakage. First, energy is consumed actively because the execution of computational tasks involves switching of the transistors on the chip. This switching of transistors is an irreversible process. Every time one of the billion transistors on the chip switches, a small amount of energy is dissipated to the surroundings. Until recently, the active energy consumption was the dominant factor in the energy

consumption of the chip. However, energy is also consumed passively from the moment the chip is switched on. During standby time, when the chip is waiting for the next computational task, this energy consumption is referred to as an energy leakage, because the chip is wasting energy while not fulfilling any computational tasks. The energy leakage originates from small currents flowing through the transistors in the off-state. Today's transistors are non-ideal switches and therefore do not fully block the current in the off-state, allowing small leakage currents to flow. By scaling the transistors down to nanometer lengths, it becomes increasingly difficult to block the leakage current and the energy leakage increases. As a consequence, the energy leakage is nowadays becoming a fundamental problem in the chip's energy consumption. Energy leakage is especially a problem in chips targeted for devices with a long standby time, such as IoT devices. IoT devices are targeted to be 'always on, always connected'-devices and will consume a lot of leakage energy.

Energy leakage in the chips is detrimental for the IoT technology because of three reasons: first, energy leakage drains the limited energy supply of the IoT chip, second, energy leakage results in unwanted heat dissipation, and third, energy leakage in masses of IoT chips is likely to result in a global waste of energy. First, an IoT chip will be fueled by a limited energy supply, probably in the form of a battery. To attain good operation in a general IoT application, the batteries in the connected IoT devices have to last very long. However, energy leakage will unnecessarily drain the batteries of the IoT devices in standby mode and decrease the lifetime of the IoT application. In the near future, the IoT devices may also be fueled by energy harvested from the surroundings (e.g. sunlight, vibrations)[29] instead of relying on a battery for their energy supply. In this case, reducing the energy leakage is even more crucial because the energy supply may be smaller and less reliable. Second, energy leakage dissipates heat to the surroundings of the IoT chip. This dissipated heat can pose problems in IoT applications where, for instance, the IoT chip is worn in clothes, close to the human skin or incorporated in food packages. In these example applications excessive heat is a problem and cooling the IoT chip is not a directly available option. Third, using masses of IoT chips in the world in standby mode will consume a lot of leakage energy and is likely to result in a global energy waste problem. In an IoT society this excessive energy consumption may represent a major global challenge. If researchers want to roll out an IoT technology in the near future with chips based on transistors scaled into the nanometer regime, the energy leakage has to be reduced.

1.2 Proposed solution

Chips are optimized for different metrics. The most important metrics are energy consumption, performance and cost. In designing a chip, all three metrics need to be taken into account. Depending on the application, one metric is given more weight than the other, but they all need to be considered. For instance, making the transistors for chips in a older (and longer) transistor technology will reduce the

leakage energy and the cost, but will at the same time deteriorate the performance of the chip. This is also the case with an IoT chip. To counteract the issue with privacy and security associated with IoT, there is a recent trend under research[20] to compute more of the raw data locally in the chip, instead of sending the data directly to the cloud for computation. As a consequence, the performance of the IoT chip in executing computational tasks needs to meet a certain minimum performance. In addition, the optimization to the different metrics (energy consumption, performance and cost) is done on different levels of abstraction. The main levels of abstraction are system level, circuit level and transistor level.

In this thesis we propose a solution for the energy leakage on transistor level and the primary metric we consider is energy consumption. Performance and cost are considered secondary metrics which are optimized when the energy consumption target is met. The proposed solution includes using nanowire transistors with a geometric superlattice for energy filtering. In the next sections the solution itself and the choice for this solution are explained in more detail. Nanowire transistors, energy leakage, energy filtering and superlattices are explained consecutively.

1.2.1 Nanowire transistors

Nanowire transistors are a promising candidate for transistors in future chips. Nanowires are typically made of Si or III-V materials in a horizontal or vertical configuration. The horizontal configuration is typically made using a top-down etching technique, while the vertical configuration is typically made with a bottom-up assembly technique[43]. Horizontal nanowire transistors are targeted in the roadmap for introduction in 2018 and vertical nanowire transistors for 2020[21]. Nanowire transistors can enable a higher transistor density on the chip, especially in the vertical configuration. However, vertical nanowire transistors will need a paradigm shift in the process flow, because today's process flow is established for chips with planar transistor architectures.

Nanowire transistors have several advantages over today's 14 nanometer planar transistors. First, nanowires have purely one-dimensional transport properties because of the motion quantization in the transverse direction. The electrons in the nanowire channel form a 1D-electron gas with a lot of available states along the nanowire and only a few on the transverse direction. Second, a better electrostatic gate control over the channel can be achieved because the gate can be placed all-around the nanowire. Third, because the nanowire's cross-sectional dimension is typically below 10 nanometer, the nanowires are small enough to become fully depleted by a gate bias. Hence, the junctionless transistor concept becomes possible[34]. The junctionless transistor concept removes the necessity to make very abrupt junctions in the source and the drain, simplifying the fabrication process and lowering the thermal budget. Additionally, in contrast with an inversion layer transistor concept, electron transport happens in the bulk of the device instead of at the interface. This allows a lower surface scattering and less degradation of mobility when the gate

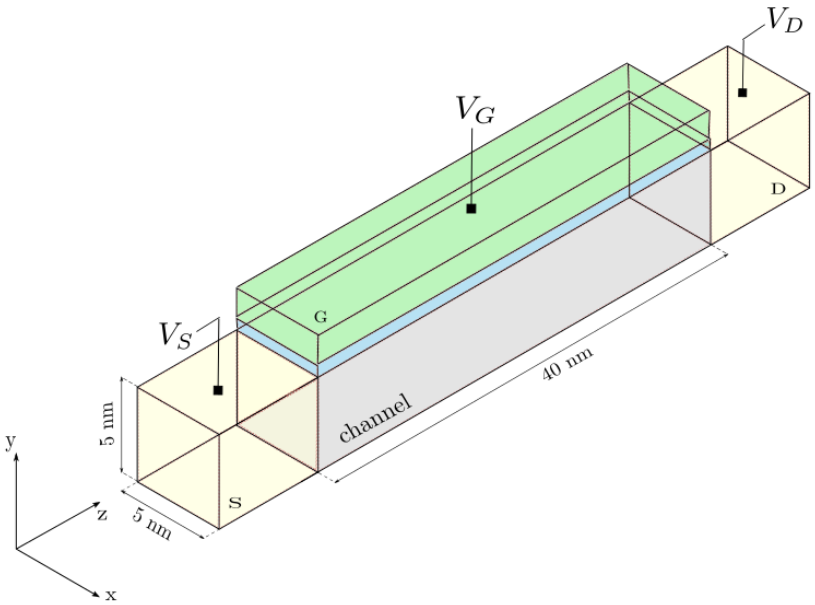


Figure 1.2: Junctionless nanowire transistor in horizontal configuration and with square cross-section. The cross-sectional dimensions of the nanowire are 5 nm by 5nm and the channel length is 40 nm. The z-coordinate axis is placed along the nanowire channel.

voltage is increased. The main disadvantages of the junctionless transistor concept are the variability in doping and the contact resistances at the source and drain. The doping concentration in nanowires can be upper limited[1] depending on the diameter to achieve a positive threshold voltage[13].

1.2.2 Energy leakage in nanowire transistors

Figure 1.2 shows a junctionless nanowire transistor in horizontal configuration. The nanowire channel measures 5 nm by 5 nm by 40 nm and the source, the drain and the gate are denoted with S, D and G respectively. The power consumption of the nanowire transistor can be estimated using the global power equation:

$$P = \underbrace{\alpha C V_{dd}^2 f}_{P_{active}} + \underbrace{I_{leak} V_{dd}}_{P_{passive}} \tag{1.1}$$

The power consumption in the nanowire transistor consists of the active power consumption P_{active} and the passive power consumption $P_{passive}$. The active power consumption is proportional to the activity level α, the capacitance C, the supply voltage V_{dd} and the operational frequency f. The passive power consumption is

[1] For a threshold voltage of $V_{th} = 0.3\ V$ and a diameter of 10 nm the doping concentration cannot exceed 10^{19} cm^{-3}[13].

proportional to the leakage current I_{leak} through the nanowire transistor in the off-state and the supply voltage V_{dd}. For IoT chips, the passive power consumption dominates because the IoT chips are in standby most of the time and the activity level α is low. Therefore, the optimization is done for low passive power consumption. To lower passive power consumption on transistor level, we can either lower the supply voltage V_{dd} or lower the leakage current I_{leak}.

Lowering the supply voltage V_{dd} is the first option to reduce passive power consumption on transistor level. A lower supply voltage is beneficial for reducing active as well as passive power consumption. The supply voltage in today's 14 nm planar transistor chips is 0.7 V. For nanowire transistors, the roadmap predicts a scaling of the supply voltage below 0.5 V[21]. However, lowering the supply voltage can decrease performance. A certain overdrive $V_{dd} - V_{th}$ with the threshold voltage V_{th} is necessary to sustain performance[26].

Lowering the leakage current I_{leak} is the second option to reduce passive power consumption on transistor level. Lowering the leakage current is more difficult than lowering the supply voltage, because the leakage current is inherently linked with the carrier injection mechanism of the transistor in the off-state. In this thesis we only consider n-type junctionless nanowire transistors and therefore the carriers are electrons. The electron injection mechanism in the nanowire transistor in the off-state is the thermal injection mechanism. The energies of the electrons in the source are statistically distributed according to Fermi-Dirac statistics. At room temperature, some of the electrons have sufficient energy to overcome the potential energy barrier and are injected into the channel. This is referred to as *thermal injection* and is qualitatively represented in figure 1.3. The high energy electrons in the source are responsible for the leakage current and thus the passive power consumption in the nanowire transistor.

Figure 1.3 shows the junctionless nanowire transistor with a qualitative energy band diagram for the first subband of the conduction band. The Fermi-Dirac distribution is also shown and gives the equilibrium distribution of the electron energies in the source at room temperature. The part of the exponential tail of the Fermi-Dirac distribution above the potential energy barrier corresponds to the *hot* electron density, as shown in red. The hot electrons have enough energy to surpass the potential energy barrier and can contribute to the leakage current, increasing passive power consumption.

Most of the IoT chips will be operated at room temperature. At room temperature, the Fermi-Dirac distribution is as shown in figure 1.3. However, the IoT chip can heat up due to active power consumption when executing computational tasks. At higher temperature, the distribution of electrons in the source changes to a distribution with more electrons occupying sufficiently high energies to surpass the potential energy barrier. As a result, the leakage current and the passive power consumption increase with higher temperature. IoT chips operated in high-temperature

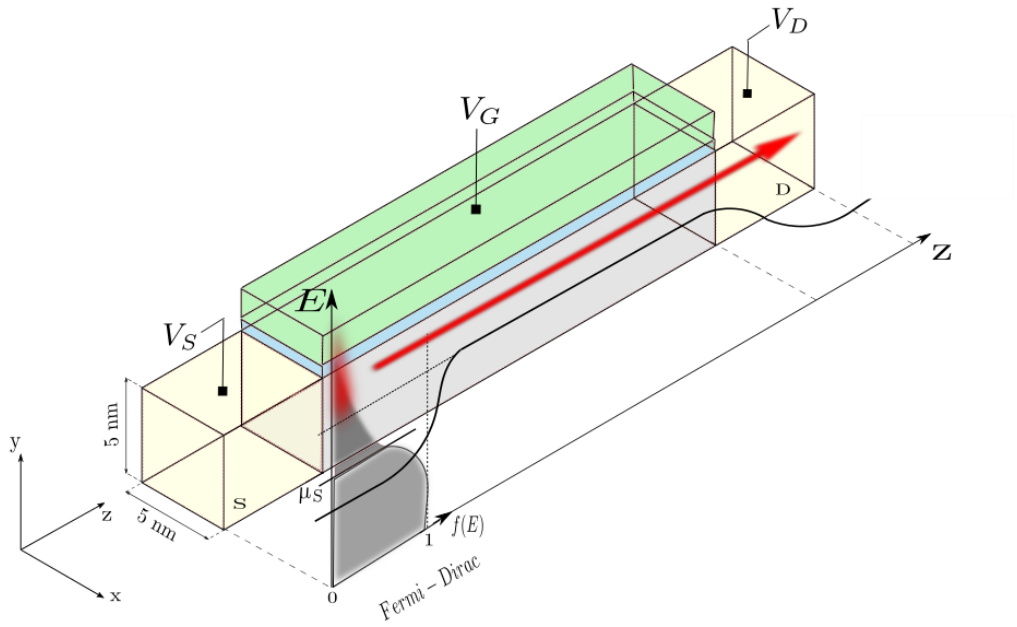


Figure 1.3: Junctionless nanowire transistor in the off-state with a qualitative energy band diagram of the first subband of the conduction band. The figure describes thermal injection of electrons in the source into the channel. Electrons with energy higher than the gate potential energy barrier can surpass the barrier and contribute to the leakage current.

environments, such as IoT sensors in industrial buildings, will have the same issue of increased passive power consumption. Energy filtering, discussed next, allows to filter out the high energy electrons in the source of the nanowire transistor to decrease the leakage current and the passive power consumption.

1.2.3 Energy filtering

Energy filtering is the concept of blocking the high energy electrons injected into the channel to attain a lower leakage current. Energy filtering can be achieved in three currently known ways: confinement, band-to-band tunneling or resonant tunneling. The two main energy filtering devices under research at the time of writing are tunnel field-effect transistors (tunnelFETs) and superlattice field effect transistors (superlatticeFETs). TunnelFETs use band-to-band tunneling to achieve energy filtering, while superlatticeFETs use resonant tunneling. In this thesis we focus on superlatticeFETs using nanowires. The nanowire superlatticeFET is shown in horizontal configuration and with material superlattice in figure 1.4.

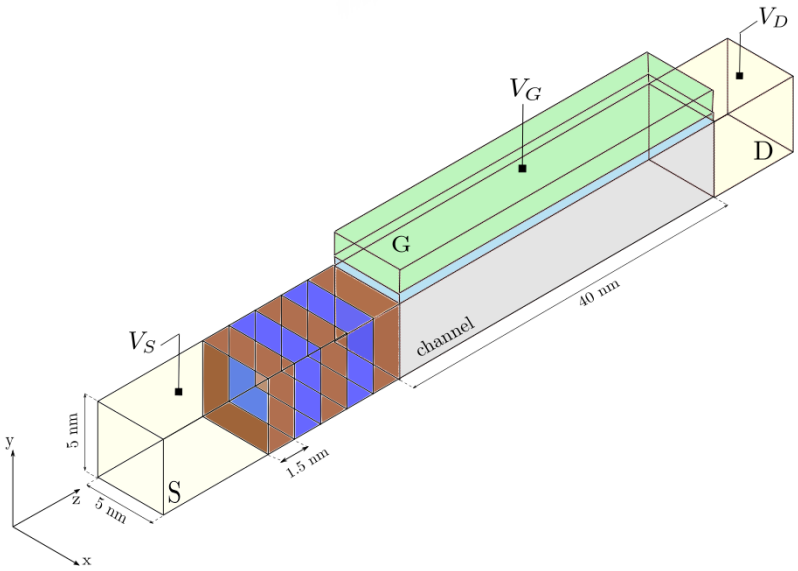


Figure 1.4: The nanowire superlatticeFET with a material superlattice in a horizontal configuration. The superlattice is made with a sequence of two alternating materials, shown in blue and brown colors. The superlattice allows to filter out the high energy electrons in the source.

Superlattice

The superlattice is the sequence of two alternating materials at the source[2]. The purpose of the superlattice is to provide an extra periodic potential profile along the nanowire, on top of the periodic lattice potential, hence the name 'super'-'lattice'. In a semiconductor material the periodicity of the crystal atoms is responsible for the formation of bandgaps. By applying an extra periodic potential profile along the transport direction of the nanowire with a spacing much bigger than the lattice constant, minibandgaps can be formed in the energy spectrum of the nanowire. The minibandgaps can be placed at energies corresponding to the energies of the hot electrons in the source. The minibandgaps will then block the hot electrons and avoid their contribution to the leakage current, as shown in figure 1.5. Therefore, the superlattice allows to act as an *energy filter* to filter out the hot electrons in the source.

Superlattice with ideal energy filtering capacity

The placement and width of the minibands and miniband gaps can be varied by engineering the superlattice structure, for instance by changing the material composition of the superlattice, the width of the periods etc. This allows to search

[2] A sequence of two alternating materials is one of the ways of making a superlattice, as patented by M. Björk et al[9] and researched by E. Gnani et al[12, 17, 14, 15].

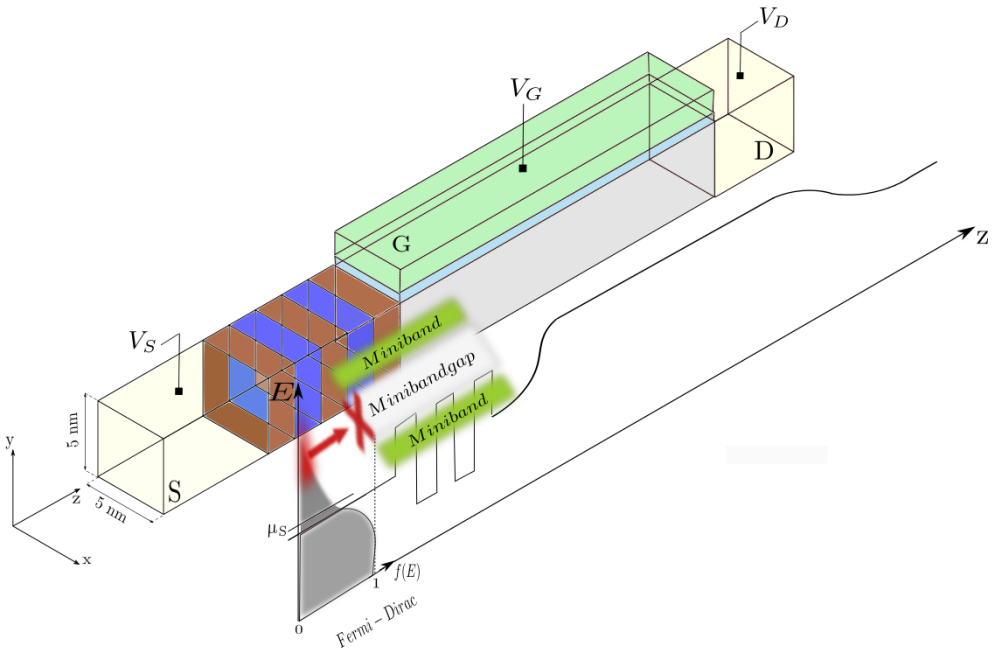


Figure 1.5: Nanowire superlatticeFET with material superlattice and a qualitative energy band diagram of the first subband of the conduction band. The figure shows the first two minibands and the first minibandgap in the energy spectrum of the superlattice. The minibandgaps allow to block (part of) the high energy electrons in the source. Ideally, the first minibandgap extends to higher energies to block the full exponential tail of the Fermi-Dirac distribution.

for the superlattice with ideal energy filtering capacity. The ideal energy filter has the following characteristics in its miniband structure. First, the miniband structure has a decent first miniband with a width around 0.1-0.3 eV to achieve a sufficient on-state current and sufficient performance. The width of 0.1-0.3 eV of the first miniband follows from the energy range which can be blocked by the gate potential energy barrier and corresponds to the necessary supply voltage V_{dd} (around 0.1-0.3 eV for nanowire transistors[21]). Second, the first minibandgap needs to be as big as the extension of the tail of the Fermi-Dirac distribution for blocking preferably all the high energy electrons. This allows a low leakage current and low passive power consumption. Third, a steep turn-on of the transistor is necessary, which translates in a steep transition between the first miniband and the first minibandgap. Ideally, the band edge between the first miniband and first miniband gap is infinitely sharp to go from zero to many electrons injected into the channel by lowering the potential energy barrier only over a small energy range[50]. This will allow for a steep rise in current for a small change in gate voltage. In the next paragraph we discuss another kind of superlattice which will allow for more variety in the superlattice to search for the superlattice with ideal energy filtering capacity.

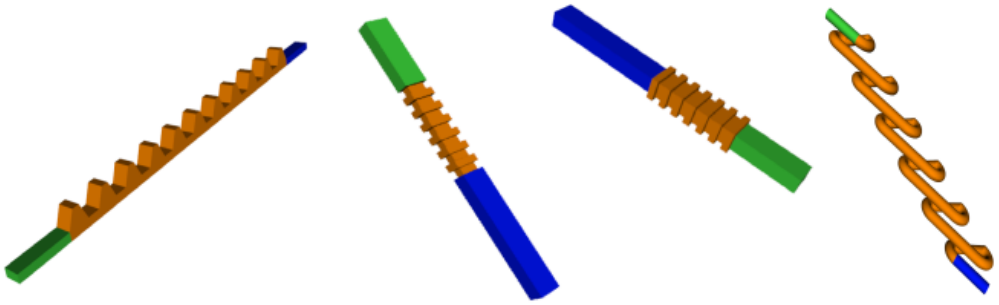


Figure 1.6: Example geometric superlattices including, from left to right: a periodicity of triangular bumps, indents, fins, spirals in the nanowire. The orange regions denote the superlattice. The blue and green regions denote the source and drain side of the nanowire superlatticeFET.

Geometric superlattice

Instead of making the superlattice with a sequence of two alternating materials as claimed by Björk et al[9] and further investigated by Gnani et al[12, 17, 14, 15], another possibility is to apply periodic features in the geometry of the nanowire. We refer to this kind of superlattice as a *geometric superlattice*. In this thesis we consider nanowire superlatticeFETs with a geometric superlattice. To the best of the author's knowledge, the nanowire superlatticeFET with geometric superlattice has not previously been discussed in literature.

The geometric superlattice has the following advantages over the superlattice with a sequence of two alternating materials. First, a geometric superlattice allows for a wider variety of possible superlattices than are possible with a material sequence. One can think of various geometric superlattices, such as constrictions in the nanowire, indentations, a sequence of fins, triangles etc. Some example geometric superlattices are shown in figure 1.6. A wide variety of possible superlattices is advantageous in the search for the superlattice with ideal energy filtering capacity. Second, in contrast with the material superlattice, a geometric superlattice can be made out of only one material, preferably Si. Using Si for the superlattice can result in lower defects and less chance the energy filtering capacity of the superlattice is obstructed by introducing states in the minibandgaps[50]. If the gate would be placed on top of the superlattice, silicon's native oxide SiO_2 should result in less interface traps and allow to keep a sharp band edge for steep turn-on. To boost the on-state current of the tunnelFET a transition was made from Si to III-V materials. However, for superlatticeFET there is no direct need to transition to III-V materials because the on-state current is expected to be higher due to resonant tunneling instead of band-to-band tunneling. Third, doping small structures is a fundamental problem

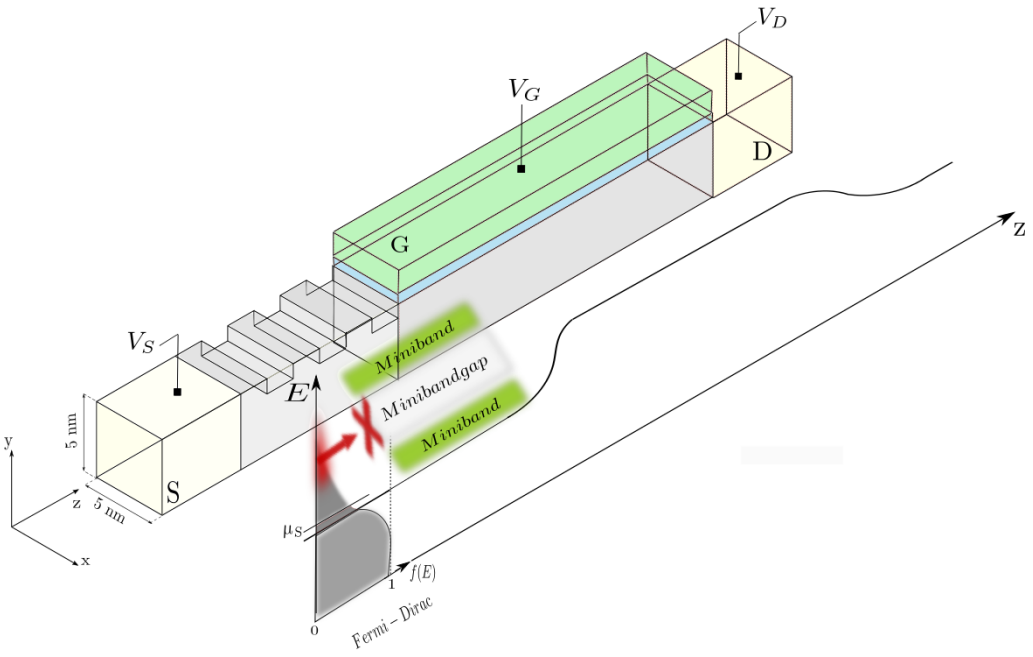


Figure 1.7: Qualitative figure of the nanowire superlatticeFET with a geometric superlattice made with periodic top indentations at the source. A qualitative energy band diagram of the first subband of the conduction band is shown in the transport direction of the nanowire. A transmission spectrum with a possible miniband structure for the geometric superlattice is shown on top of the energy band diagram at the superlattice. The minibandgaps allow to block (part of) the high energy electrons in the source. Ideally, the first minibandgap extends to higher energies to block the full exponential tail of the Fermi-Dirac distribution.

because of the variability of the dopants. To dope the separate materials in a material superlattice, a very high thermal budget is needed to keep the boundaries of the doped regions very sharp. The geometric superlattice does not have an alternating material sequence which makes doping less an issue, although the variability issue of the dopants in the nanowire remains.

The main disadvantage of a geometric superlattice is its fabrication. At the time of writing a geometric superlattice with periodic features with lateral dimensions of a few nanometers is either difficult, impossible or very expensive to make depending on the periodic features. Electron beam lithography should allow for a 2 to 3 nm precision in the lateral dimension[31]. However, making a full periodic superlattice can represent a major challenge in terms of fabrication and reliability at the time of writing.

1.3 Research objectives and methodology

In this section the research objectives are stated and the methodology to attain these research objectives is explained. The research objectives are translated into research questions to be answered in the different phases of the thesis work. The methodology is explained based on a flow chart covering the whole thesis work.

1.3.1 Research objectives

The first research objective is to gain insight in the various low power transistor concepts currently under research. We investigate the place of the nanowire superlatticeFET with a geometric superlattice among the other low power transistor concepts. We explore the available methods to model the nanowire superlatticeFET with a geometric superlattice and choose a suitable method.

The second research objective is to find and set up a suitable theoretical model for the energy filtering in the nanowire superlatticeFET with a geometric superlattice. The purpose of the model is to show the existence of energy filtering in 3D geometric superlattices. The model should allow for quick simulation of nanowire superlatticeFETs with various geometric superlattices and show their energy filtering capacity.

The third research objective is to first show the existence of energy filtering in 3D geometric superlattices and then, if possible, show the tunability of the energy filtering by varying the geometric superlattice. We explain the influence of the different periodic parameters of the geometric superlattice on the energy filtering. We try to find a geometric superlattice with ideal energy filtering capacity by simulating various kinds of geometric superlattices with different periodic parameters.

The fourth research objective is to decrease the relative importance of the geometric superlatticeFET's main disadvantage: its fabricational difficulty and cost of fabrication. For one specific geometric superlattice, we try to lower the fabricational difficulty and cost by lowering the number of periods in the geometric superlattice. We check the impact of a lower number of periods on the energy filtering capacity of the geometric superlattice. We calculate the turn-on characteristics of a nanowire superlatticeFET with a geometric superlattice. From the turn-on characteristics, we deduct the leakage current, the passive power consumption, the switching slope and the on-state current.

The research objectives can be translated to the following research questions:

- Why the superlatticeFET as low power transistor concept?
- How to model and simulate the superlatticeFET? Which method to use?
- How to model energy filtering in 3D?
- Is energy filtering possible with a geometric superlattice?

- How to vary the geometric superlattice to tune energy filtering?
- Is there a geometric superlattice with ideal energy filtering capacity?
- Can we lower the cost and fabricational difficulty of the geometric superlattice? Does it impact the energy filtering capacity?
- How is the geometric superlatticeFET characterized in terms of leakage current, switching slope and on-state current?

Thesis statement

Show energy filtering in 3D and for a periodicity in the geometry, instead of a sequence of two alternating materials:

- Find and set up a suitable theoretical model for energy filtering in 3D, which allows to simulate the transmission spectra of different geometric superlattices to place minibands and minibandgaps at relevant energies for supply voltages around 0.1 eV to 0.3 eV.
- Optimize for a specific geometric superlattice with ideal energy filtering capacity and check if this geometric superlattice can be simplified to decrease the fabricational difficulty of the superlattice.
- Characterize the nanowire superlatticeFET with a chosen geometric superlattice in terms of leakage current, passive power consumption, switching slope and on-state current.

1.3.2 Methodology

The methodology used to find answers to the posed research questions is explained in this section. The methodology is summarized in the flow chart in figure 1.8. The flow chart gives an overview of the sequence of phases in this thesis work. The thesis work is divided in a literature study and three phases of research: modelling energy filtering with a geometric superlattice (phase I), simulating energy filtering with a geometric superlattice (phase II) and investigating a Si nanostrip geometric superlatticeFET (phase III).

Literature study

The first two research questions are answered by doing a literature study. The result from the literature study is the insight in the place of the nanowire superlatticeFET with a geometric superlattice among the other low power transistor concepts currently under research. We validate a continuous quantum approach in 3D, which we will use in phase I for modelling the energy filtering in a 3D geometric superlattice.

Phase I: Modelling energy filtering with a geometric superlattice

In phase I of the research we answer the research question how we should model energy filtering in 3D by setting up a model using a continuous quantum approach in 3D. This approach was chosen based on the literature study. The continuous quantum approach in 3D involves solving a 3D Schrödinger boundary value problem with open boundaries. The open boundaries are necessary to model the current flow through the nanowire. The quantum transmitting boundary method (QTB method) is a suitable method to take the open boundaries into account in a continuum approach. The QTB method is derived in literature in 2D[27], but not in 3D. Therefore, we first derive the QTB method in 3D and to make the method generally applicable, we do the derivation for a general 3D device geometry and for isotropic effective mass as well as anisotropic effective mass. We rewrite the 3D Schrödinger equation in a form suitable for numerical solution with the finite element method. We derive expressions for the transmission coefficients to plot transmission spectra in phase II and state expressions for the electron charge density and current using a ballistic approach.

Phase II: Simulating energy filtering with a geometric superlattice

In phase II of the research we start with simulating straight nanowires without a geometric superlattice as a check for the model we set up in phase I. A straight nanowire without a geometric superlattice should result in a transmission spectrum with full transmission at all available energies starting from the injected subband energy. Thereafter, we look for the existence of energy filtering with a 3D geometric superlattice by plotting the transmission spectrum for a geometric superlattice with indentations. In this case, the transmission spectrum should have clear drops in transmission over a range of energies, corresponding to minibandgaps. If there's energy filtering, we vary the periodic parameters of the geometric superlattice to understand their influence on the energy filtering and find an ideal energy filter.

Phase III: Investigating the Si nanostrip geometric superlatticeFET

In phase III we choose one geometric superlattice with the best obtained energy filtering capacity in phase II. For this geometric superlattice, we lower the number of periods and check if the energy filtering capacity decreases. A lower number of periods in the geometric superlattice puts less pressure on the fabrication and variability of the periodic nanometer sized features in the geometric superlattice. We plot the turn-on characteristics of a Si nanostrip geometric superlatticeFET. From the turn-on characteristics we derive the leakage current, the passive power consumption, the on-state current and the switching slope.

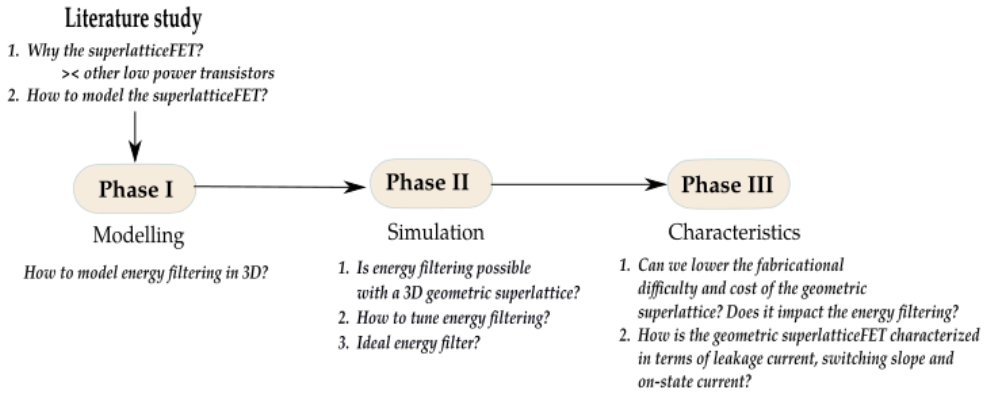


Figure 1.8: Schematic structure of the research

Chapter 2

Literature study

"Why the superlatticeFET as low power transistor?"

"How to model and simulate the superlatticeFET?"

The goal of the literature study is to develop an overview of the possible low power transistor concepts currently under research and gain insight in the place of the nanowire superlatticeFET among other steep slope transistor concepts described in literature. We also investigate the methods used to model and simulate low power transistor concepts. The result of the literature study is a validation of the superlatticeFET as a viable steep slope transistor concept and a chosen method which will be used in the rest of the thesis work to model and simulate nanowire superlatticeFETs with a geometric superlattice.

2.1 Low power transistor concepts

For a low-power transistor, we are especially interested in the turn-on characteristic. From the turn-on characteristic we can derive valuable metrics such as the leakage current, the passive power consumption, the inverse subthreshold slope and the on-state current. The turn-on characteristic of a transistor is the curve of the drain-source current in function of the gate voltage plotted on a logarithmic scale, i.e. $\log I_{DS}$ vs. V_{GS}. The figure of merit for a low power or *steep slope* transistor is the inverse subthreshold slope SS which can be derived from the turn-on characteristic. The inverse subthreshold slope SS is the inverse slope of the $logI_{DS}$ V_{GS} curve in the subthreshold regime. In literature, it is common to work with the inverse subthreshold slope, i.e. $\frac{\partial V_{GS}}{\partial log(I_{DS})}$ in units of mV per decade current. A steep slope of the turn-on characteristic then corresponds to a low value in mV per decade of current. The turn-on characteristics of a conventional planar MOSFET transistor, an ideal transistor and a low power transistor alternative are plotted in figure 2.1.

The inverse subthreshold slope $SS = \frac{\partial V_{GS}}{\partial log(I_{DS})}$ gives the change in gate voltage needed to realise one decade change in current through the transistor. The ideal

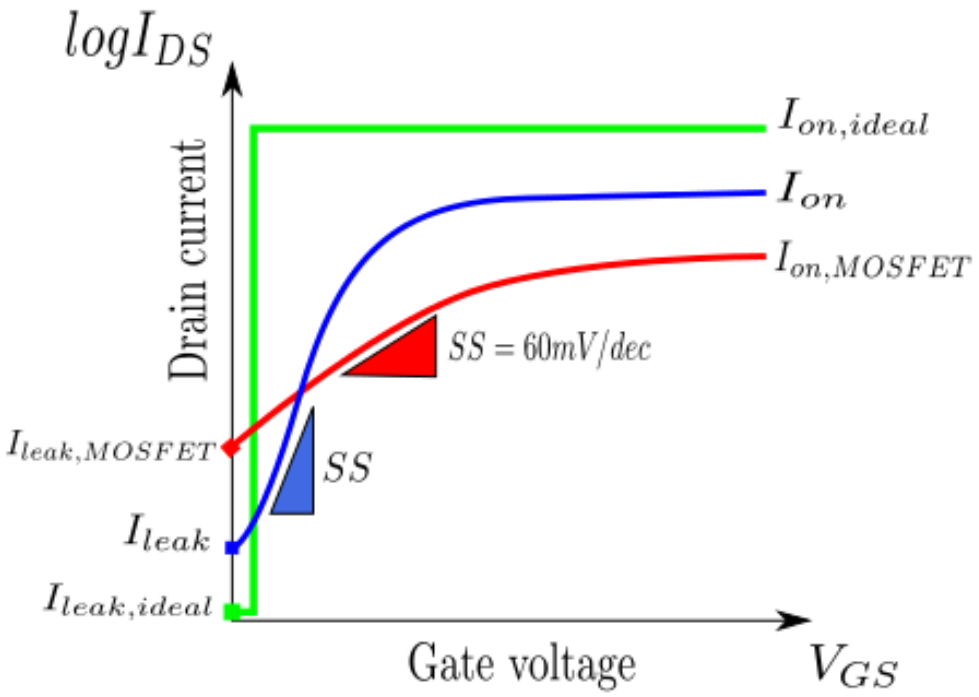


Figure 2.1: Turn-on characteristics of an ideal transistor (green), the planar MOSFET transistor (red) and a low power transistor alternative (blue). The planar MOSFET transistor has a fixed inverse subthreshold slope of 60 mV/dec while the ideal transistor has a turn-on characteristic which rises infinitely fast with gate voltage.

turn-on characteristic is infinitely steep resulting in an abrupt change between the on- and the off-state of the transistor. In figure 2.1 the green line shows the ideal SS. Other important metrics of an ideal steep slope transistor are a high on-state current for good performance and a low leakage current for decreasing the passive power consumption. A steep slope is advantageous because it allows to lower the supply voltage V_{dd} below 0.5 V and thus decreasing the active power consumption as well as the passive power consumption. With a lower limited 60 mV per decade slope for planar MOSFET transistors lowering the V_{dd} below 0.5 V is not possible because the current cannot rise enough order of magnitudes before reaching 0.5 V due to the smaller slope, resulting in a too small difference between on-state current and leakage current.

Today's planar MOSFET transistors usually have a subthreshold slope of around 70 to 120 mV per decade. The ideal 60 mV per decade subthreshold slope arises from thermodynamic considerations of the carrier injection mechanism. The carrier injection mechanism in planar MOSFETs is thermal injection of electrons from the source into the channel. The electron energy distribution of the electrons in the

source in equilibrium is described by Fermi-Dirac statistics in function of temperature. In the thermal injection mechanism, the electrons with the highest thermal energy in the source are injected into the channel. How fast the current can rise in the MOSFET transistor is thus fundamentally limited by the thermally injected electrons surpassing a modulated gate potential energy barrier. This limits the achievable inverse subthreshold slope SS. To obtain a steeper slope transistor, it is necessary to change the carrier injection mechanism. Another carrier injection mechanism is quantum mechanical tunneling. In quantum mechanical tunneling not only the electrons with highest thermal energy are injected into the channel, but also lower energetic electrons can contribute to the current. Low power transistor concepts which change the carrier injection mechanism to obtain a steep slope fall in the category of the energy filtering devices. The two main examples of energy filtering devices are the tunnelFETs (TFETs) and the superlatticeFETs. The tunnelFET changes the carrier injection mechanism to band-to-band-tunneling (BTBT), while the superlatticeFET changes the carrier injection mechanism to resonant tunneling. Considering the formula for the subthreshold slope SS, we note that changing the carrier injection mechanism is not the only way to influence the subthreshold slope.

$$SS = \frac{\partial V_{GS}}{\partial log\,(I_{DS})} = \underbrace{\frac{\partial V_{GS}}{\partial \phi_S}}_{a} \underbrace{\frac{\partial \phi_S}{\partial log\,(I_{DS})}}_{b} \tag{2.1}$$

where ϕ_S is the surface potential in the channel. Changing the carrier injection mechanism corresponds to changing the factor b in equation 2.1. The other possibility to change the inverse subthreshold slope SS is to vary factor a which corresponds to amplifying the non-linearity in the transistor using a positive feedback mechanism[18, 22].

The steep slope transistors currently under research can be divided into two categories: the transistors which use a change in carrier injection mechanism to achieve a steep slope and those which use an amplification mechanism inside the transistor to achieve a steep slope. The former category includes tunnelFETs and superlatticeFETs, while the latter category includes the iMOSFET (impact ionization MOSFET), the FeFET (Ferroelectric FET) and the SG-MOSFET (suspended gate MOSFET).

2.1.1 Energy filtering based steep slope transistors

Energy filtering is the concept of filtering out the high energy electrons injected into the channel. The electrons can be filtered out by changing the DOS at the source by means of a superlattice (the concept applied in superlatticeFETs) or by changing the band-to-band-tunneling probability (the concept applied in TunnelFETs). The TunnelFETs and superlatticeFETs are discussed next.

TunnelFET

The tunnelFET was first proposed by Quinn et al in 1978[36, 22]. TunnelFETs use the band-to-band tunneling mechanism to obtain energy filtering and achieve a steep

slope. The band-to-band tunneling mechanism (BTBT) governs electrons tunneling between energy bands in a highly doped p-n-junction[22]. The placement of the energy bands can be changed abruptly with the gate potential which allows for a steep slope. The main drawback of TunnelFETs is the low on-state current due to band-to-band tunneling. The lower on-state current results in a performance issue for TunnelFETs. Many performance boosters have been investigated for tunnelFETs to attain a better on-state current, without compromising on the leakage current and the subthreshold slope. The performance boosters include i.a. high-k gate dielectrics, higher source doping, a double gate or the transition from Si to III-V materials[22, 45]. III-V materials can increase performance because of their low effective mass and allow for a wide variety in band engineering possibilities to increase BTBT[46]. A disadvantage of the III-V materials is the increased chance for defects compared to Si, which can lead to trap-assisted tunneling (TAT) which degrades the operation of the tunnelFET[47].

SuperlatticeFET

The research done on superlatticeFETS is far behind the research done on TunnelFETS so far. The nanowire superlatticeFET was first proposed in 2008 by M. Björk et al. in an IBM-patent[9]. From 2010 onward extensive research has been done on the nanowire superlatticeFET by the group of E. Gnani in Bologna[12, 17, 14, 15]. Gnani et al. achieved a theoretical inverse subthreshold slope of 13 mV/dec for a cylindrical nanowire with InGaAs-InAlAs material superlattice and an on-state current of 4.5 mA/μm at a supply voltage of 0.4 V[15]. The superlatticeFET changes the carrier injection mechanism to resonant tunneling. Electrons tunnel resonantly through a superlattice placed at the source extension. At some energies, corresponding to minibands, electrons can tunnel from the source to the entrance of the channel (called the virtual source by Gnani). At other energies, corresponding to minibandgaps, the electrons are blocked from tunneling to the entrance of the channel. By carefully designing the miniband structure of the superlattice, the superlattice can block the high energy electrons in the source and act as an energy-filter to achieve a low leakage current. At the virtual source, the dominant carrier injection mechanism in the off-state is source-to-drain tunneling, allowing for a lower leakage current than the thermal injection mechanism. In the next paragraph the formation of minibandgaps in the energy spectrum is examined more closely.

Formation of minibandgaps

The formation of minibandgaps in the transmission spectrum of the nanowire can be understood from the wave nature of the electrons travelling through the superlattice. Using the wave nature of the electrons, two arguments can be developed to understand the formation of minibandgaps. The first argument is an argument from a wave interference perspective, the second argument is an argument from a tight binding perspective.

The first argument explains the formation of minibandgaps by the interference of electron waves. The electron waves undergo Bragg reflection due to the periodic superlattice potential. An incoming electron wave can be reflected by one of the periods of the superlattice and interfere constructively or destructively. Depending on the energy of the electron, the wavelength of the electron varies, which influences the tendency for constructive or destructive interference. Depending on where the constructive and destructive interferences happen in the nanowire superlattice and on which energies, the interferences can result in minibands or minibandgaps in the transmission spectrum of the superlattice nanowire.

The second argument explains the formation of minibandgaps by starting from a tight binding perspective. In a tight binding perspective, the periods of the superlattice are spaced infinitely far apart and the electrons are tightly bound to the separate potential wells. With an infinite spacing between the potential wells, the wavefunctions from the separate wells do not overlap. When we bring the periods of the superlattice closer to eachother, the wavefunctions in the separate wells start to overlap. The linear combinations of overlapping wavefunctions are also solutions of the Schrödinger equation, but yield different energy levels. The energy levels thus start to broaden and form minibands. The energy spacing between the minibands where no available energy levels are present form the minibandgaps.

2.1.2 Amplification based steep slope transistors

Amplification based steep slope transistors use positive feedback in the turn-on mechanism to achieve a steep slope. The steep slope transistors which use this mechanism to achieve a steep slope include the iMOSFET (impact ionization MOSFET), the FeFET (ferroelectric FET) and the SG-MOSFET (suspended gate MOSFET).

2.1.3 iMOSFET

The iMOSFET or impact ionization MOSFET was developed by the group of J.Plummer at Stanford in 2002[18]. Impact ionization of electrons is a mechanism that changes abruptly with the energy of the electrons, which is used in the iMOSFET to achieve a steep slope. Impact ionization starts only at a high breakdown voltage V_{BD}. The iMOSFET thus requires a drain voltage which is higher than V_{BD}. The breakdown voltage V_{BD} is at least $1.5\frac{E_g}{e}$ with E_g the bandgap of the semiconductor material and e the fundamental electron charge[12]. This limits the scope of the iMOSFET for use in circuits, as the stacking of iMOSFET will result in a very high supply voltage needed to keep every iMOSFET in the stack at the breakdown voltage.

2.1.4 SG-MOSFET

The suspended gate MOSFET (SG-MOSFET), also called the nano-electromechanical FET (NEMFET) by Kam et al[25], combines MEMS (Micro-Electromechanical

Systems) functionality with transistor functionality to achieve a steep slope. In the off-state of the transistor a nanometer-sized cantilever makes contact with the gate dielectric on top of the channel. When the cantilever makes contact with the gate dielectric, the channel is fully depleted and no current flows. When the voltage is increased on the cantilever, the depletion width decreases and a current starts to flow. At a certain voltage the electrostatic force on the cantilever will equal the spring force and the cantilever will snap back from the gate dielectric. This abrupt snapping back of the cantilever is responsible for the steep slope of the SG-MOSFET. In addition, a complete elimination of the gate leakage current is possible when the transistor is surrounded by a vaccuum and hence, the gap between cantilever and gate dielectric in the on-state is filled with vacuum. However, with the introduction of high-k gate dielectrics, the leakage current problem has shifted away from gate leakage toward drain-source leakage as the major problem. Subthreshold slopes of $< 2mV$ per decade were achieved experimentally with the SG-MOSFET by Ionescu et al[4] in Lausanne. However, the SG-MOSFET is not free of issues either. Scaling of the cantilever to nanometer dimensions can be an issue, as well as stiction of the cantilever to the gate dielectric and fatigue of the mechanical cantilever.

2.1.5 FeFET

The FeFET or ferroelectric FET was proposed by S. Datta at Purdue university as steep slope transistor concept[39]. The FeFET uses a positive feedback mechanism in the gate voltage attributed to the negative capacitance of the gate ferroelectric to achieve a steep slope. The main disadvantage of the ferroelectric FET is the replacement of the gate oxide with a ferroelectric capacitor. Ferroelectrics have just like the SG-MOSFETS issues with scalability and fatigue.

2.1.6 Conclusion

In conclusion, we state that the available low power transistor concepts with steep turn-on characteristic are limited and each have their own major disadvantage. Especially the amplification based steep slope transistors have serious drawbacks which inhibit further investigation and implementation of these steep slope transistor concepts for now. As regards the energy filtering devices, extensive research has already been done on tunnelFETs, while the research attention for superlatticeFETs is very low in comparison with tunnelFETs. Hence, superlatticeFETs represent an opportunity for further research. In addition, the research done on superlatticeFETs is mainly limited to nanowires with a material superlattice[12, 17, 14, 15]. To the best of the author's knowledge, no research has already been published on superlatticeFETs with a geometric superlattice.

2.2 Choice of the modelling and simulation method

In this section we validate the need for modelling and simulations of superlatticeFETs and choose one particular method which will be used to model and simulate nanowire

superlatticeFETs with a geometric superlattice.

2.2.1 Modelling and simulation

Running simulations before actual production of a test chip has become indispensable nowadays in semiconductor industry. It takes around two to three months to run a wafer through all the necessary process steps. Simulating new implementations on the chip beforehand has therefore several advantages: first, it allows to decrease the number of actual experiments needed in the lab, second, it can give insight in phenomena which cannot be measured experimentally, and third, it allows to test the viability of hypothetical devices which cannot be made yet or at high cost. Nanowire superlatticeFETs belong to the latter case. Superlattices with periodic features with lateral dimensions of a few nanometers are either difficult, impossible or very expensive to make depending on the periodic features. Electron beam lithography should allow for a 2 to 3 nm precision in the lateral dimension[31]. However, making a full periodic superlattice can represent a major challenge in terms of fabrication and reliability at the time of writing. Hence, probing the energy filtering experimentally on a fabricated transistor is not possible yet, validating the need for a proper model and simulation method for superlatticeFETs and superlatticeFETs with geometric superlattice in particular.

In the following paragraphs we explain the different needs which a proper model and simulation method for superlatticeFETs with geometric superlattice should fulfill. We validate the choice for a **three-dimensional (3D) continuous quantum approach** and clarify the exact methods used in this approach.

2.2.2 Dimensionality of the method

Gnani et al model the nanowire superlatticeFET with material superlattice in two dimensions by taking advantage of the rotational invariance[15]. However, in a geometric superlattice the periodic features in the geometric superlattice are very important for the energy filtering of the superlattice and are ideally modelled in 3D. A 3D model for the geometric superlattice also allows for more freedom in shaping its periodic parameters. The effects of various 3D periodic features can then be examined. However, using a 3D model may complicate the mathematical derivation of the theoretical model and increase computational time during simulation compared to lower dimensional models.

2.2.3 Semiclassical vs. quantum approach

Most commercial software packages use the semiclassical approach. An example of a semiclassical software package is Sentaurus Device (S-Device) from Synopsys. S-Device can model 3D devices but does not yet support the simulation of resonant tunneling devices. Using a fully semiclassical approach for the modelling of the nanowire superlatticeFET with geometric superlattice is not desirable. Resonant

tunneling, the operation principle of the nanowire superlatticeFET, is a quantum-mechanical phenomenon which would be neglected in a fully semiclassical approach. However, using a semiclassical approach with quantum corrections can be a possibility.

Semiclassical methods with quantum corrections

The semiclassical approach with quantum corrections could somehow account for the resonant tunneling by implementing it as an extension of the semiclassical approach. However, the semiclassical approach derives the charge densities and the currents from the drift-diffusion equations. Drift-diffusion is not the transport mechanism governing transport in the superlattice and it is questionable if one can fill this error by including some quantum correction. Therefore, we choose to set up the model starting from a fully quantum approach.

Fully quantum methods

A quantum method involves solving the Schrödinger equation and in our case, the 3D Schrödinger equation, because we chose to model the nanowire superlatticeFET in 3D in 2.2.2. In the category of the quantum methods we further distinguish the continuous quantum methods and the atomistic quantum methods.

2.2.4 Continuous vs. atomistic quantum methods

A continuous quantum method models the superlatticeFET from a top-down perspective, while an atomistic quantum method models the nanowire superlatticeFET from a bottom-up perspective. Which method to use depends on the size of the modelled device concept. For the nanowire superlatticeFET, this is a delicate question because the size of the nanowire (several tens of nanometers) is quite small to be modelled by a continuous method and quite big to be modelled by an atomistic method. An atomistic method for the nanowire superlatticeFET can become computationally very expensive, while a continuous method can become inaccurate. The choice for a continuous or atomistic approach is not ready-made. Therefore, we consider both the continuous quantum methods as the atomistic quantum methods as possible candidates for modelling and simulation of the nanowire superlatticeFET.

Atomistic quantum methods

An atomistic approach is in general more computationally expensive than a continuous approach. The computational efficiency will be low for larger structures such as nanowires with still hundreds of atoms. Potential atomistic quantum methods are solving the Schrödinger equation in a tight binding approximation or using the density functional theory (DFT) to obtain the electronic structure. For the transport calculations, possible approaches are assuming ballistic transport or using the non-equilibrium Green's function formalism (NEGF).

Solving the Schrödinger equation in a **tight binding** approach involves making

linear combinations of the single atom wavefunctions of the atoms in the system. In this method the computational load rapidly increases for many atoms. Usually a nearest-neighbour assumption is made to lower the computational load. In the nearest-neighbour assumption only linear combinations are made of the wavefunctions of neighbouring atoms.

The **density functional theory (DFT)** is widely used to describe the electronic structure of atoms, molecules or even superconductors[40, 6]. A possible commercial software package which includes DFT is Atomistix Toolkit of Quantumwise. Density functional theory was first proposed in 1964 by P. Hohenberg and W. Kohn as a method to solve the Schrödinger equation for an interacting electron gas[19]. The ground state of the interacting electron gas is written as a functional of the charge density, which is then minimized in terms of energy to obtain the correct ground state energy. DFT is not used often for semiconductor materials, because DFT does not allow for an accurate description of the band gap in semiconductors[48].

The **non-equilibrium Green's function (NEGF)** formalism uses a Green's function at a point in the nanowire to calculate charge densities and currents in the whole nanowire. Possible commercial software packages which use NEGF are Atomistix Toolkit of Quantumwise and NEMO or OMEN of Purdue University. NEMO-3D models the electronic structure of structures of tens of nanometers long with tight binding and uses NEGF for the transport calculations[38]. However, the NEGF formalism is computationally very expensive, especially for 3D structures[33]. A more practical approach to calculating the charges and currents is to assume **ballistic transport**, which assumes the distribution of the electrons in the channel to keep the same equilibium distribution as in the source because no scattering is assumed in the channel.

The above atomistic quantum methods are computationally heavy. In our case, we do not require the most complex method from the start. Rather, in a first approximation, a simpler method may suffice to prove energy filtering in superlatticeFETs with geometric superlattice. Additionally, a simpler method will allow a higher turn-over rate of simulation results. A high turn-over rate of simulation results is desirable to test the energy filtering capacity of many geometric superlattices at reasonable computational time. The continuous quantum methods are computationally less demanding, but they should remain accurate.

Continuous quantum methods

A continuous method can become inaccurate for smaller structures. A continuous quantum method usually starts with the *effective mass theorem* to simplify the direct solution of the Schrödinger equation. The effective mass theorem converts the Schrödinger equation into an effective mass Schrödinger equation by leaving out the crystal potential contribution in the Schrödinger equation and changing the fundamental electron mass of the electron to an effective mass m^*. The effective

mass is derived from the bulk band structure of the semiconductor material, which becomes more and more questionable for structures away from the bulk configuration. The effective mass Schrödinger equation is stated in equation 2.2.

$$-\frac{\hbar^2}{2m^*}\nabla^2\psi + V\psi = E\psi \tag{2.2}$$

The effective mass theorem is advantageous because the crystal potential changes on the order of the lattice constant, which would require a mesh point at each lattice site for accurate computation of the wavefunction. The obtained solutions from the effective mass Schrödinger equation will be smooth envelope functions of the rapidly oscillating solutions of the Schrödinger equation and require less mesh points for accurate computation.

The bulk band structure used to derive the effective mass m^* in equation 2.2 is obtained from either the bulk conduction band or the bulk valence band. In this thesis we consider n-type junctionless nanowire transistors and hence, we use the bulk conduction band. The bulk conduction band is further approximated as a parabolic band by the *effective mass approximation* (EMA). A single band EMA is possible for superlatticeFETs in a first approximation because superlatticeFETs do not have interband transition as a working principle in contrast with tunnelFETs[11]. However, in the case of p-type superlatticeFETs, a multiband model should be used for the valence band, such as the $\mathbf{k}\cdot\mathbf{p}$ model. Under confinement the light hole, the heavy hole and the split off band of the valence band mix to form new subbands[32]. Therefore, it is expected that the hole states are very sensitive to variations in the orientation and cross-sectional structure of the nanowires[32]. Varying the cross-sectional structure is exactly the purpose of the geometric superlattice.

In general, the effective mass is an anisotropic property of the material and needs to be modelled by an effective mass tensor M^* instead of a scalar m^*. The Schrödinger equation in equation 2.2 then changes[1] to:

$$-\frac{\hbar^2}{2}\nabla\cdot\left(\frac{1}{M^*}\nabla\right)\psi + V\psi = E\psi \tag{2.3}$$

or written into components:

$$-\frac{\hbar^2}{2}\frac{\partial}{\partial x_i}\frac{1}{m^*_{i,j}}\frac{\partial}{\partial x_j}\psi + V\psi = E\psi \tag{2.4}$$

The effective mass tensor takes into account the different effective masses depending on the orientation of the crystal relative to the nanowire orientation. This is especially important for Si nanowires, because Si has multiple valleys and anisotropy of the effective mass in a valley. In the next two paragraphs we explain further the EMA and the effective mass tensor M^* in function of the crystal orientation.

[1] The full explanation of the form of the anisotropic effective mass Schrödinger equation is given in appendix A.2.

2.2.5 Effective mass approximation

Given a band structure or *dispersion relation* $E(\mathbf{k})$ from literature, the m^* in the effective mass Schrödinger equation can be computed as follows:

$$m^*_{ij} = \hbar^2 \left(\frac{\partial^2 E}{\partial k_i \partial k_j} \right)^{-1} \tag{2.5}$$

with $i, j = x, y, z$ denote the $\boldsymbol{k}_x$, $\boldsymbol{k}_y$ or $\boldsymbol{k}_z$ unit vectors in the band structure. The effective mass m^*_{ij} is inversely proportional to the curvature of the bands. For the three $\boldsymbol{k}_x$, $\boldsymbol{k}_y$ and $\boldsymbol{k}_z$ directions there are in general 6 different m^*_{ij} (symmetric tensor) and the effective mass is in fact a 3×3 tensor M^*.

$$M^* = \begin{bmatrix} m^*_{xx} & m^*_{xy} & m^*_{xz} \\ m^*_{yx} & m^*_{yy} & m^*_{yz} \\ m^*_{zx} & m^*_{zy} & m^*_{zz} \end{bmatrix} \tag{2.6}$$

If we approximate the bands at a certain (k_x, k_y, k_z)-point with spherical bands, the curvature of the bands is in every direction the same or *isotropic*. This approximation is called the *effective mass approximation* and it simplifies the effective mass tensor M^* to a scalar effective mass m^*:

$$M^* = \begin{bmatrix} m^* & 0 & 0 \\ 0 & m^* & 0 \\ 0 & 0 & m^* \end{bmatrix} \Rightarrow M^* = m^* \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = m^* I = m^* \tag{2.7}$$

2.2.6 Effective mass tensor M^* in function of the crystal orientation

The 3×3 anisotropic effective mass tensor M^* can be written in a simple form by using a coordinate system defined locally in the the band structure. However, we need an M^* which is defined in the device coordinate system (DCS). Suitable transformation matrices are needed to transform the effective mass tensor M^* expressed locally in the band structure to the DCS.

First we express M^* in a local coordinate system in the band structure. For bulk Si for instance, the constant energy surfaces are ellipsoids around the 6 conduction band minima at the Δ-points:

$$E(\boldsymbol{k}) = \frac{\hbar^2 k_L^2}{2m^*_L} + \frac{\hbar^2 \left(k^2_{T1} + k^2_{T2} \right)}{2m^*_T} \tag{2.8}$$

The longitudinal k_L vector and the two transverse k_{T1} and k_{T2} vectors are the unit vectors of the local ellipsoidal coordinate system (ECS) unique to each of the 6 ellipsoids. Equation 2.8 can be written in matrix notation:

$$E(\boldsymbol{k}) = \frac{\hbar^2}{2} \begin{bmatrix} k_L & k_{T1} & k_{T2} \end{bmatrix} \begin{bmatrix} \frac{1}{m^*_L} & 0 & 0 \\ 0 & \frac{1}{m^*_{T1}} & 0 \\ 0 & 0 & \frac{1}{m^*_{T2}} \end{bmatrix} \begin{bmatrix} k_L \\ k_{T1} \\ k_{T2} \end{bmatrix} \tag{2.9}$$

where the longitudinal effective mass m_L^* is 0.91 and the transverse effective mass m_T^* 0.19 for Si. Equation 2.9 can be written in a more compact form:

$$E(\boldsymbol{k}) = \frac{\hbar^2}{2}\boldsymbol{k}_E^T(M_E^*)^{-1}\boldsymbol{k}_E \tag{2.10}$$

where M_E^* is the effective mass tensor expressed in the ellipsoidal coordinate system (ECS) of which the tensor components are known. In general, the ellipsoidal coordinate system (ECS), the crystal coordinate system (CCS) and the device coordinate system (DCS) do not coincide. A vector expressed in the ECS can be transformed to the DCS by using two transformation matrices $T_{E\leftarrow C}$ and $T_{C\leftarrow D}$:

$$\boldsymbol{k}_E = T_{E\leftarrow C}T_{C\leftarrow D}\boldsymbol{k}_D \tag{2.11}$$

Inserting equation 2.11 into equation 2.10, results in:

$$E(\boldsymbol{k}) = \frac{\hbar^2}{2}\boldsymbol{k}_D^T\left(T_{E\leftarrow C}T_{C\leftarrow D}\right)^T(M_E^*)^{-1}T_{E\leftarrow C}T_{C\leftarrow D}\boldsymbol{k}_D \tag{2.12}$$

By comparing equation 2.12 with equation 2.10, we find the inverse effective mass tensor $(M^*)^{-1}$ expressed in the DCS:

$$(M_D^*)^{-1} = \left(T_{E\leftarrow C}T_{C\leftarrow D}\right)^T(M_E^*)^{-1}T_{E\leftarrow C}T_{C\leftarrow D} \tag{2.13}$$

By using suitable transformation matrices for $T_{C\leftarrow D}$ and $T_{E\leftarrow C}$, we can evaluate the effective mass tensor M^* in the DCS for the 6 different valleys in Si and the commonly used wafer orientations (100), (110) and (111).

2.2.7 Continuous quantum approach in 3D

In summary, we choose to do a continuous quantum approach in 3D. To obtain the electronic structure, we solve the Schrödinger equation in the effective mass approach for isotropic effective mass as well as anisotropic effective mass using the quantum transmitting boundary method (QTB method) for the open boundaries. The QTB method was proposed for two dimensions by C.S. Lent and D.J. Kirkner[27]. To use the QTB method in 3D, we will first derive the QTB method in 3D and for isotropic as well as anisotropic effective mass. To calculate charges and currents in the device, we assume ballistic transport. For the direct solution of the Schrödinger equation Gnani[15] uses the subband decomposition method as proposed by E. Polizzi[35]. The subband decomposition method is an approximation made to increase the numerical efficiency of the QTB method. In the subband decomposition method the solutions of the wavefunctions are decomposed using eigenfunctions on different cross-sections along the nanowire. The eigenfunctions on the cross-sections form a basis-set from which the wavefunctions in the system can be computed, as described in equation 2.14:

$$\psi(x,y,z) = \sum_i^N a_i(z)\chi_i(x,y,z') \tag{2.14}$$

with N the number of z-positions where the cross-sectional wavefunctions χ are computed. The coefficients $a_i(z)$ account for the z-directionality of the solution. However, the general QTB method suits our case best, because the subband decomposition simplification does not handle abrupt variations on the cross-section inbetween two different cross-sections very well. Abrupt variations on the cross-section are very important in the potential energy filtering behaviour of the geometric superlattice. The QTB method is explained in full in the next chapter when we set up the model for the nanowire superlatticeFET with a geometric superlattice by extending the QTB method to 3D and (an)isotropic effective masses.

2.3 Conclusion

The first research objective was to gain insight in the various low power transistor concepts currently under research. We found that the available steep slope transistor concepts are limited and each have their own major disadvantage. Especially the amplification based steep slope transistors have serious drawbacks which inhibit further investigation and implementation of these steep slope transistor concepts for now. As regards the energy filtering devices, extensive research has already been done on tunnelFETs, while the research attention for superlatticeFETs is very low in comparison with tunnelFETs. Hence, superlatticeFETs represent an opportunity for further research. In addition, the research done on superlatticeFETs is mainly limited to nanowires with a material superlattice. Therefore, in this thesis we simulate nanowire superlatticeFETs with a different superlattice, the geometric superlattice. We have chosen for a 3D continuous quantum approach to simulate the nanowire superlatticeFETs with a geometric superlattice. The chosen 3D continuous quantum approach involves the direct solution of the 3D effective mass Schrödinger equation using the QTB method for the open boundaries and assuming ballistic transport.

Chapter 3

Phase I: Modelling energy filtering with a geometric superlattice

"How to model energy filtering in 3D?"

The goal of this chapter is to set up a model for energy filtering in three-dimensional (3D) geometric superlattices. In the previous chapter we motivated the choice for a continuous quantum approach in 3D. To model energy filtering, we want to find transmission spectra for a geometric superlattice. The transmission spectra give the transmission for a range of energies and hence, allow to visualize the minibands and minibandgaps. The transmission spectra are expected to vary for different 3D geometric superlattices, which would allow to tune the energy filtering properties. Therefore, it is important to set up the theoretical model for a device geometry as general as possible.

3.1 Continuous quantum approach in 3D

In this section we find the electronic structure using a continuous quantum approach in 3D. Finding the electronic structure will allow us to visualize probabilities for the electrons in the device and calculate transmission coefficients for the transmission spectra. To find the electronic structure we solve the 3D open boundary Schrödinger equation in the effective mass approximation for the conduction band using Cartesian coordinates. We extend the quantum transmitting boundary method (QTBM) as proposed by C.S. Lent and D.J. Kirkner [27] to 3D problems with anisotropic effective mass. For the calculation of charges and currents in the device, we assume ballistic transport and derive expressions for the charges and currents.

3.1.1 General strategy

The model should be valid for any kind of geometry to allow for the modelling of a wide range of geometric superlattices. The system shown in figure 3.1 consists of a generally shaped 3D domain called Ω_0 with infinite leads Ω_l with l the lead index. The boundaries of the Ω domains are denoted by $\partial\Omega$. The system has a general number of open boundaries D_l where electrons can flow in and out. The rest of the system boundary $\partial\Omega - \sum D_l$ is a closed boundary. Equation 3.1 states the 3D Schrödinger equation[1] in the effective mass approximation for electrons in the conduction band:

$$-\frac{\hbar^2}{2}\nabla\cdot\left(\frac{1}{M^*}\nabla\right)\Psi(\mathbf{r})+V(\mathbf{r})\Psi(\mathbf{r})=E\Psi(\mathbf{r}) \tag{3.1}$$

Solving the partial differential equation[2] (PDE) in 3.1 is non-trivial. Two problems arise when trying to find the states $\Psi(\mathbf{r})$ for 3.1. The first problem is a problem with calculating the potential $V(\mathbf{r})$. The second problem is a problem with defining the boundary conditions for the PDE. These two problems are discussed next.

The potential $V(\mathbf{r})$ in the Schrödinger equation can be obtained from the Poisson equation. The Poisson equation is another PDE entering the problem. Equation 3.2 gives the Poisson equation.

$$-\nabla\cdot(\epsilon\nabla)\,V(\mathbf{r})=\rho \tag{3.2}$$

In the Poisson equation ϵ is the permittivity of the material and ρ the electron charge density. The charge density ρ is given by $\rho = e(n + N_D^+)$ with e the fundamental electron charge, n the electron concentration and N_D^+ the ionized donor doping concentration. We assume the dopants to be fully ionized, i.e. $N_D = N_D^+$. The ionized donor doping concentration N_D^+ is given in the problem statement. The electron charge density n depends on the solutions $\Psi(\mathbf{r})$ of the Schrödinger equation. The Schrödinger equation and the Poisson equation are thus mutually dependent and cannot be solved separately. To solve the Schrödinger equation, we need the potential $V(\mathbf{r})$ from the Poisson equation, but to solve the Poisson equation we need the $\Psi(\mathbf{r})$ from the Schrödinger equation. To find self-consistent solutions for the potential and the charge, the usual approach is to set up a self-consistent Schrödinger-Poisson loop. In chapters 4 and 5, the simulations are run non-self-consistently in a first approximation.

3.1.2 Boundary conditions for the Schrödinger equation: QTB method

A PDE only has uniquely defined solutions if stated as a full boundary value problem with boundary conditions[51]. For the problem geometry boundary conditions

[1] The exact order of the operators in the kinetic energy term follows from the hermiticity of the Hamiltonian (see appendix)

[2] The exact order of the operators in the kinetic energy term $-\frac{\hbar^2}{2}\nabla\cdot\left(\frac{1}{M^*}\nabla\right)$ follows from the fact that the Hamiltonian needs to be Hermitian. The full explanation is given in A.2.

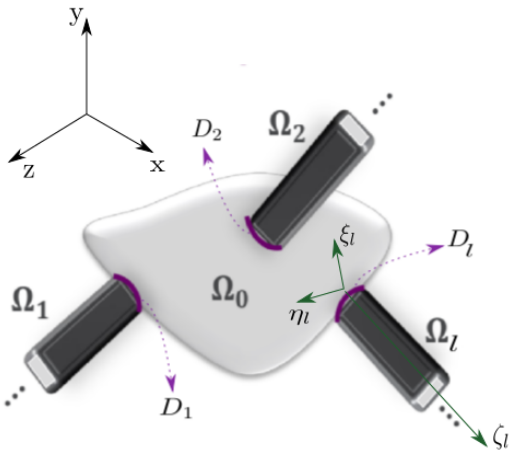


Figure 3.1: General 3D problem geometry with leads extruding from the open boundaries D_l of the device region Ω_0. The global xyz-coordinate system is shown as well as one of the $\xi_l\eta_l\zeta_l$ lead coordinate systems. For simplicity only three leads are shown, although the derivation is valid for any number of leads.

need to be defined on the boundary $\partial\Omega$. The boundary $\partial\Omega$ consists of the open boundaries D and the closed boundary $\partial\Omega - \sum D$. On the closed boundary the wavefunction is assumed to take a constant value. This is a Dirichlet boundary condition $\Psi|_{\partial\Omega-\sum D} = c$ with c a constant. Stating the boundary conditions on the open boundaries D is more difficult. The wavefunction is not constant on the open boundaries, because we have current carrying states (travelling waves) passing through these boundaries. We need to find an expression for the wavefunction Ψ on the open boundaries. A possible approach is to extend the problem geometry at the open boundaries. In the extensions we then solve the corresponding Schrödinger equations for the wavefunctions and assume continuity with the wavefunctions in the original domain. However, this does not solve the problem. We cannot solve the Schrödinger equations in the extensions because then again we need to define an open boundary condition on the far end of the extension.

A possible method which allows to find suitable boundary conditions for the open boundaries is the *quantum transmitting boundary method* (QTB method). The QTB method starts with extending the problem geometry at the open boundaries with *leads*. The leads are extrusions of the open boundaries with infinite length. The infinite length removes the necessity to define open boundary conditions on the far end of the extension. The problem geometry with leads is shown in figure 3.1. The state Ψ now runs over the whole system, i.e. the original domain plus the leads. In figure 3.1 the original domain is denoted by Ω_0 and called the *device region*. A global coordinate system (x, y, z) is defined for the device region. The leads are denoted by Ω_l with l the lead index. A local coordinate system (ξ, η, ζ) is defined in each lead with the ζ-axis along the lead and the origin of the axis system on D_l.

Additionally, for infinite leads, the potential change of $V_l(\xi, \eta, \zeta)$ in the direction along the lead can be spread over an infinite amount of distance and hence there is no potential change in the ζ-direction; $V_l(\xi, \eta, \zeta)$ can be assumed independent of ζ, i.e. $V_l(\xi, \eta)$. This is crucial for the QTB method, because it allows an analytical solution of the Schrödinger equations in the leads. The analytical solutions for the wavefunctions in the leads can then be used to state boundary conditions on the open boundaries D using the continuity conditions. To apply the QTB method to the problem at hand, we first have to extend this method to three dimensions.

The quantum transmitting boundary method (QTB method) was first suggested in 1990 by C.S. Lent and D.J. Kirkner for two-dimensional problems[27]. In the next two sections we extend the QTB method to three dimensions (3D), first for an isotropic effective mass (section 3.2) and then for an anisotropic effective mass (section 3.3). The QTB method includes the following two steps:

1. Solve analytically the Schrödinger equations for the wavefunctions in the leads
2. Use the analytical solutions for $\psi_l(\xi, \eta, \zeta)$ to obtain the *quantum transmitting boundary conditions (QTBCs)* for the open boundaries of the device region

Applying the QTBCs to the device region Ω_0 results in a fully defined problem for ψ_0 in the device region. After we found the solutions for ψ_0 in Ω_0, we can combine them with the solutions for ψ_l in $\sum_l \Omega_l$ and obtain the state Ψ in the whole system $\Omega = \Omega_0 + \sum_l \Omega_l$.

3.2 Quantum transmitting boundary conditions in 3D with isotropic effective mass

In this section we derive the QTB method for a Schrödinger equation with a scalar effective mass m^*. Because the effective mass is modelled with a scalar value, the effective mass is in this model an isotropic property of the material. Example materials with isotropic effective masses include most of the III-V materials, such as GaAs.

3.2.1 Step 1: Find analytical solutions of Schrödinger equations in the leads

The first step in the QTB method is to solve analytically the Schrödinger equations for the wavefunctions $\psi_l(\xi, \eta, \zeta)$ in the leads:

$$\left(-\frac{\hbar^2}{2m_l^*}\nabla^2 + V_l(\xi, \eta, \zeta)\right)\psi_l(\xi, \eta, \zeta) = E\psi_l(\xi, \eta, \zeta) \tag{3.3}$$

Equation 3.3 denotes a set of L partial differential equations in $\psi_l(\xi, \eta, \zeta)$ where L is the total number of leads and l the lead index. E is the total amount of energy in

the system. $V_l(\xi,\eta,\zeta)$ is the externally applied potential on the lead. In the QTB method, $V_l(\xi,\eta,\zeta)$ is assumed to be independent of ζ, i.e. $V_l(\xi,\eta)$. The potential does not vary along the lead, only on the cross-section. This assumption is a crucial part of the QTB method, because it allows separation of variables of the partial differential equation such as to obtain an analytical solution for the partial differential equation. Writing the Laplacian operator ∇^2 in its full Cartesian form in the lead coordinate system (ξ,η,ζ), yields:

$$\left(-\frac{\hbar^2}{2m_l^*}\left(\frac{\partial^2}{\partial\xi^2}+\frac{\partial^2}{\partial\eta^2}+\frac{\partial^2}{\partial\zeta^2}\right)+V_l(\xi,\eta)\right)\psi_l(\xi,\eta,\zeta)=E\psi_l(\xi,\eta,\zeta) \tag{3.4}$$

In the separation of variables technique $\psi_l(\xi,\eta,\zeta)$ is separated into a cross-sectional contribution $\chi_l(\xi,\eta)$ and a longitudinal contribution $\zeta_l(\zeta)$, i.e. $\psi_l(\xi,\eta,\zeta)=\chi_l(\xi,\eta)\zeta_l(\zeta)$. Inserting this in equation 3.4 and working out, we obtain:

$$-\frac{\hbar^2}{2m_l^*}\frac{1}{\chi_l(\xi,\eta)}\left(\frac{\partial^2}{\partial\xi^2}+\frac{\partial^2}{\partial\eta^2}\right)\chi_l(\xi,\eta)+V_l(\xi,\eta)-E=\frac{\hbar^2}{2m_l^*}\frac{1}{\zeta_l(\zeta)}\frac{\partial^2\zeta_l(\zeta)}{\partial\zeta^2} \tag{3.5}$$

The left side of equation 3.5 is only dependent on ξ and η, while the right side is only dependent on ζ. As a consequence, both sides of the equation are equal to a constant. This constant is given an arbitrary value of $-\frac{\hbar^2k_l^2}{2m_l^*}$ for later ease. Equation 3.5 then separates into two problems, the longitudinal problem and the transverse problem:

$$\begin{cases} \frac{\hbar^2}{2m_l^*}\frac{1}{\zeta_l(\zeta)}\frac{\partial^2\zeta_l(\zeta)}{\partial\zeta^2}=-\frac{\hbar^2k_l^2}{2m_l^*} & \textbf{(Longitudinal problem)} \\ -\frac{\hbar^2}{2m_l^*}\frac{1}{\chi_l(\xi,\eta)}\left(\frac{\partial^2}{\partial\xi^2}+\frac{\partial^2}{\partial\eta^2}\right)\chi_l(\xi,\eta)+V_l(\xi,\eta)-E=-\frac{\hbar^2k_l^2}{2m_l^*} & \textbf{(Transverse problem)} \end{cases}$$

The longitudinal and the transverse problem are now solved separately to find solutions for $\zeta_l(\zeta)$ and $\chi_l(\xi,\eta)$ respectively. The obtained solutions for $\zeta_l(\zeta)$ and $\chi_l(\xi,\eta)$ are then used to reconstruct the solution for $\psi_l(\xi,\eta,\zeta)$.

Longitudinal problem

The longitudinal problem can be written as an ordinary differential equation in $\zeta_l(\zeta)$:

$$\frac{\partial^2\zeta_l(\zeta)}{\partial\zeta^2}+(k_l)^2\zeta_l(\zeta)=0 \tag{3.6}$$

This differential equation has the solutions $\zeta_l(\zeta)=c_le^{\pm ik_\zeta^l\zeta}$ where c_l is a constant. The solutions are plane waves travelling in lead l along the ζ-axis. Because the ζ-axis was chosen in the direction outward from the device region, the solutions $\zeta_l(\zeta)=e^{-ik_\zeta^l\zeta}$ are incoming plane waves and the solutions $\zeta_l(\zeta)=e^{ik_\zeta^l\zeta}$ are outgoing plane waves. The constant k_l acquires the physical meaning of a wavevector of a plane wave along ζ. An index ζ was added to k_l to denote the ζ-direction of the wavevector.

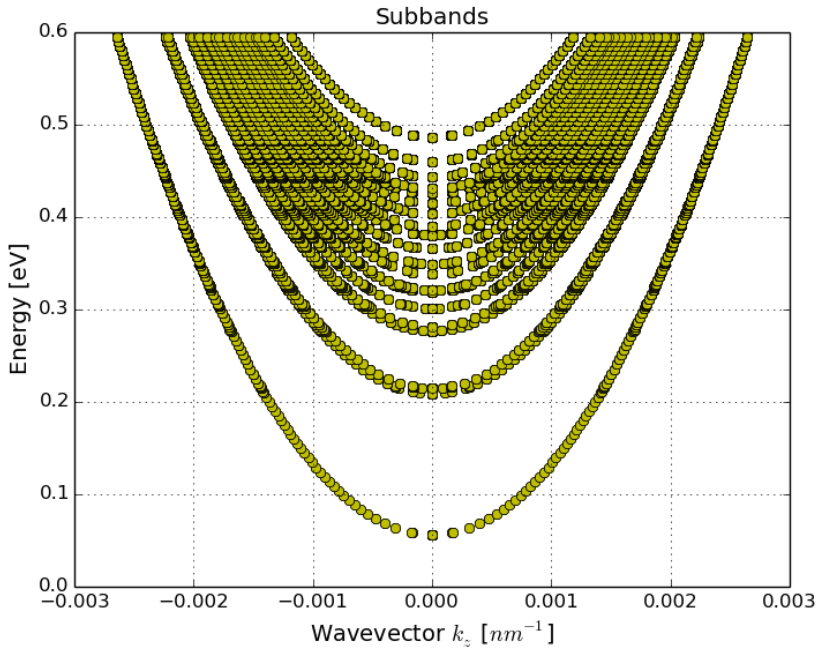


Figure 3.2: Dispersion relation showing subbands for the isotropic case.

Transverse problem

The transverse problem can be written as a Hamiltonian eigenvalue problem in $\chi_l(\xi,\eta)$:

$$\left(-\frac{\hbar^2}{2m_l^*}\left(\frac{\partial^2}{\partial\xi^2}+\frac{\partial^2}{\partial\eta^2}\right)+V_l(\xi,\eta)\right)\chi_l(\xi,\eta)=\mathcal{E}_l\chi_l(\xi,\eta) \tag{3.7}$$

with $\mathcal{E}_l=\left(E-\frac{\hbar^2(k_\zeta^l)^2}{2m_l^*}\right)$.

The solutions of this eigenvalue problem are the eigenenergies $\mathcal{E}_m^l$ and corresponding eigenfunctions $\chi_m^l(\xi,\eta)$. The quantum number m numbers the eigenenergies from zero to infinity. In practice, the eigenvalue problem is solved numerically for a predefined set of eigenenergies and corresponding eigenfunctions.

The dispersion relation $E=\mathcal{E}_m^l+\frac{\hbar^2(k_\zeta^{l,m})^2}{2m_l^*}$ is plotted in figure 3.2. Due to the quadratic dependence of E on $k_\zeta^{l,m}$ and the discreteness of $\mathcal{E}_m^l$, the plot shows a set of nested parabolas. Each parabola represents a subband of the conduction band and has a minimum value at $\mathcal{E}_m^l$. Extracting the wavevector $k_\zeta^{l,m}$ from the dispersion relation, gives:

$$k_\zeta^{l,m}=\pm\frac{\sqrt{2m_l^*(E-\mathcal{E}_m^l)}}{\hbar} \tag{3.8}$$

The wavevector is in general a complex value, because the total amount of energy E in the system can be lower than a subband energy $\mathcal{E}_m^l$.

Reconstructing the solutions in the leads

The solutions $\psi_l(\xi,\eta,\zeta)$ in the leads are now reconstructed using the previous substitution $\psi_l(\xi,\eta,\zeta) = \chi_m^l(\xi,\eta)\zeta_l(\zeta)$. All solutions are taken into account by summing over all subbands for both the incoming as the outgoing plane waves.

$$\psi_l(\xi,\eta,\zeta) = \sum_{m=0}^{\infty} \left(a_m^l e^{-ik_\zeta^{l,m}\zeta} + b_m^l e^{ik_\zeta^{l,m}\zeta} \right) \chi_m^l(\xi,\eta) \tag{3.9}$$

With equation 3.9 we have found the analytical solution for the wavefunctions in the leads. Each a_m^l coefficient denotes the complex amplitude of the incoming plane wave in lead l and subband m. Each b_m^l coefficient denotes the complex amplitude of the outgoing plane wave in lead l and subband m. The a_m^l-coefficients can be found by normalizing the solutions, while the b_m^l-coefficients are unknowns which are part of the final solution of the QTB-method. In the next steps, the solutions in the leads are first normalized and then used to derive the boundary conditions on the open boundaries of the device region.

3.2.2 Step 2: Obtain quantum transmitting boundary conditions

In the last step of the QTB method we derive the quantum transmitting boundary conditions (QTBCs). The QTBCs are suitable boundary conditions on the open boundaries of the device region. If the QTBcs are defined, then all the boundary conditions will be defined on the device region. Considering that the final state is a continuous wave which extends over the device region as well as the leads, one has to guarantee that the wave makes continuous transitions over the open boundaries D_l. In mathematical terms this translates into the continuity conditions for the wavefunctions on both sides of the boundary ψ_0 and ψ_l:

$$\begin{cases} \psi_0|_{D_l} = \psi_l|_{D_l} \\ \nabla\psi_0 \cdot \hat{\mathbf{n}}_{D_l}|_{D_l} = \nabla\psi_l \cdot \hat{\mathbf{n}}_{D_l}|_{D_l} \end{cases}$$

The second continuity condition can be rewritten as:

$$\nabla\psi_0 \cdot \hat{\mathbf{n}}_{D_l}|_{D_l} = \left(\frac{\partial\psi_l}{\partial\xi}, \frac{\partial\psi_l}{\partial\eta}, \frac{\partial\psi_l}{\partial\zeta} \right) \cdot (0,0,1) \Big|_{D_l} \tag{3.10a}$$

$$= \frac{\partial\psi_l}{\partial\zeta}\Big|_{D_l} \tag{3.10b}$$

In equation 3.10b we can use the analytical solution for $\psi_l(\xi,\eta,\zeta)$ found in equation 3.9. By taking into account the orthonormality of the eigenfunctions $\chi_m^l(\xi,\eta)$, we can find[3] an expression for the b_m^l coefficients in terms of the known a_m^l coefficients

[3] The derivation of the expression for b_m^l is given in appendix A.1.

and the wavefunctions in the leads:

$$b_m^l = \int_{D_l} \chi_m^l(\xi,\eta)\psi_l(\xi,\eta,\zeta=0)dS - a_m^l \tag{3.11}$$

Substituting 3.11 and applying the first continuity condition, results in:

$$\nabla\psi_0 \cdot \hat{\mathbf{n}}_{D_l}|_{D_l} = \sum_{m=0}^{\infty} ik_\zeta^{l,m}\chi_m^l(\xi,\eta)\left(-2a_m^l + \iint_{D_l}\chi_m^l(\xi,\eta)\psi_0(\xi,\eta,\zeta=0)dS\right) \tag{3.12}$$

With equation 3.12 we have obtained the QTBCs for the open boundaries D_l of the device region in the isotropic case of the effective mass. The QTBCs are Robin boundary conditions which relate the normal derivative of the wavefunction $\psi_0(x,y,z)$ to the values of the wavefunction $\psi_0(x,y,z)$ along the boundary.

3.3 Quantum transmitting boundary conditions in 3D with anisotropic effective mass

In this section the effective mass is not considered anymore as an isotropic property of the material. We will derive the QTB method for a Schrödinger equation with an effective mass tensor M^*. The scalar value m^* is replaced with a rank 2, 3×3 tensor M^*, which takes into account the different effective masses in three directions. Example materials with anisotropic effective mass include some of the well-established semiconductor group IV-materials, such as Si and Ge.

problem we which to solve is the 3D Schrödinger equation obtain solutions for the three-dimensional wavefunction $\psi(x,y,z)$ in the device region Ω. As in the isotropic case, the QTB method includes the following two steps:

1. Solve analytically the Schrödinger equations for the wavefunctions in the leads
2. Use the analytical solutions for $\psi_l(\xi,\eta,\zeta)$ to obtain the *quantum transmitting boundary conditions (QTBCs)* for the open boundaries of the device region

Applying the QTBCs to the device region Ω_0 results in a fully defined problem for ψ_0 in the device region. The final state Ψ in the whole system is a continuous wave which extends over the whole region Ω (device region as well as the leads).

3.3.1 Step 1: Find analytical solutions of Schrödinger equations in the leads

The first step in the QTB method is to solve analytically the Schrödinger equations for the wavefunctions $\psi_l(\xi,\eta,\zeta)$ in the leads:

$$-\frac{\hbar^2}{2}\nabla\cdot\left(\frac{1}{M_l^*}\nabla\right)\psi_l(\xi,\eta,\zeta) + V_l(\xi,\eta,\zeta)\psi_l(\xi,\eta,\zeta) = E\psi_l(\xi,\eta,\zeta) \tag{3.13}$$

Equation 3.3 denotes a set of L partial differential equations in $\psi_l(\xi,\eta,\zeta)$ where L is the total number of leads and l the lead index. E is the total amount of energy in the lead. $V^l_{ext}(\xi,\eta,\zeta)$ is the externally applied potential on the lead. In the QTB method, $V_l(\xi,\eta,\zeta)$ is assumed to be independent of ζ, i.e. $V_l(\xi,\eta)$. The potential does not vary along the lead, only on the cross-section. This assumption is a crucial part of the QTB method, because it allows to separate the partial differential equation into variables and to obtain an analytical solution for the partial differential equation. Writing $\nabla\cdot\left(\frac{1}{M_l^*}\nabla\right)$ in its full Cartesian form in the lead coordinate system (ξ,η,ζ), yields:

$$-\frac{\hbar^2}{2}\begin{bmatrix}\frac{\partial}{\partial\xi}\\ \frac{\partial}{\partial\eta}\\ \frac{\partial}{\partial\zeta}\end{bmatrix}\cdot\left(\begin{bmatrix}\frac{1}{m^*_{l,\xi\xi}} & \frac{1}{m^*_{l,\xi\eta}} & \frac{1}{m^*_{l,\xi\zeta}}\\ \frac{1}{m^*_{l,\eta\xi}} & \frac{1}{m^*_{l,\eta\eta}} & \frac{1}{m^*_{l,\eta\zeta}}\\ \frac{1}{m^*_{l,\zeta\xi}} & \frac{1}{m^*_{l,\zeta\eta}} & \frac{1}{m^*_{l,\zeta\zeta}}\end{bmatrix}\begin{bmatrix}\frac{\partial}{\partial\xi}\\ \frac{\partial}{\partial\eta}\\ \frac{\partial}{\partial\zeta}\end{bmatrix}\right)\psi_l(\xi,\eta,\zeta)+V_l(\xi,\eta)\psi_l(\xi,\eta,\zeta)=E\psi_l(\xi,\eta,\zeta) \tag{3.14}$$

In the separation of variables technique $\psi_l(\xi,\eta,\zeta)$ is separated into a cross-sectional contribution $\chi_l(\xi,\eta)$ and a longitudinal contribution $\zeta_l(\zeta)$, i.e. $\psi_l(\xi,\eta,\zeta)=\chi_l(\xi,\zeta)\zeta_l(\zeta)$. However, if we do this separation into the variables (ξ,η) and ζ, the tensor terms $\xi\eta$, $\eta\zeta$, $\zeta\xi$ and $\zeta\eta$ cause trouble, because they are not separable[4]. This issue can be circumvented by assuming the tensor takes a separable form:

$$\frac{1}{M_l^*}=\begin{bmatrix}\frac{1}{m^*_{l,\xi\xi}} & \frac{1}{m^*_{l,\xi\eta}} & 0\\ \frac{1}{m^*_{l,\eta\xi}} & \frac{1}{m^*_{l,\eta\eta}} & 0\\ 0 & 0 & \frac{1}{m^*_{l,\zeta\zeta}}\end{bmatrix} \tag{3.15}$$

In 3.15 we imposed a restriction on which form the effective mass tensor in the lead coordinate system can take. In the model, this has an influence on the orientation of the leads with respect to crystal coordinate system. We can only model systems with leads which have a ζ-axis lying along one of the axes of the crystal coordinate system. The leads can be turned around their ζ-axis. With this separable form, equation 3.5 separates into two problems, the longitudinal problem and the transverse problem:

$$\begin{cases}\frac{\hbar^2}{2}\frac{1}{\zeta_l(\zeta)}\frac{\partial}{\partial\zeta}\left(\frac{1}{m^*_{l,\zeta\zeta}}\frac{\partial}{\partial\zeta}\right)\zeta_l(\zeta)=-\frac{\hbar^2k_l^2}{2m^*_{l,\zeta\zeta}} & \textbf{(Longitudinal problem)}\\ -\frac{\hbar^2}{2}\frac{1}{\chi_l(\xi,\eta)}\begin{bmatrix}\frac{\partial}{\partial\xi}\\ \frac{\partial}{\partial\eta}\end{bmatrix}\cdot\left(\begin{bmatrix}\frac{1}{m^*_{l,\xi\xi}} & \frac{1}{m^*_{l,\xi\eta}}\\ \frac{1}{m^*_{l,\eta\xi}} & \frac{1}{m^*_{l,\eta\eta}}\end{bmatrix}\begin{bmatrix}\frac{\partial}{\partial\xi}\\ \frac{\partial}{\partial\eta}\end{bmatrix}\right)\chi_l(\xi,\eta)+V_l(\xi,\eta)-E=-\frac{\hbar^2k_l^2}{2m^*_{l,\zeta\zeta}}\\ \textbf{(Transverse problem)}\end{cases}$$

Longitudinal problem

The longitudinal problem can be written as an ordinary differential equation in $\zeta_l(\zeta)$:

$$\frac{\partial^2\zeta_l(\zeta)}{\partial\zeta^2}+(k_l)^2\zeta_l(\zeta)=0 \tag{3.16}$$

[4]The derivation is given in appendix A.

This differential equation has the solutions $\zeta_l(\zeta) = c_l e^{\pm i k_\zeta^l \zeta}$ where c_l is a constant. The solutions are plane waves travelling in lead l along the ζ-axis. Because the ζ-axis was chosen in the direction outward from the device region, the solutions $\zeta_l(\zeta) = e^{-ik_\zeta^l \zeta}$ are incoming plane waves and the solutions $\zeta_l(\zeta) = e^{ik_\zeta^l \zeta}$ are outgoing plane waves. The constant k_l acquires the physical meaning of a wavevector of a plane wave along ζ. An index ζ was added to k_l to denote the ζ-direction of the wavevector.

Transverse problem

The transverse problem can be written as a Hamiltonian eigenvalue problem in $\chi_l(\xi, \eta)$:

$$-\frac{\hbar^2}{2}\begin{bmatrix}\frac{\partial}{\partial\xi}\\ \frac{\partial}{\partial\eta}\end{bmatrix}\cdot\left(\begin{bmatrix}\frac{1}{m^*_{l,\xi\xi}} & \frac{1}{m^*_{l,\xi\eta}}\\ \frac{1}{m^*_{l,\eta\xi}} & \frac{1}{m^*_{l,\eta\eta}}\end{bmatrix}\begin{bmatrix}\frac{\partial}{\partial\xi}\\ \frac{\partial}{\partial\eta}\end{bmatrix}\right)\chi_l(\xi,\eta) + V_l(\xi,\eta)\chi_l(\xi,\eta) = \mathcal{E}_l\chi_l(\xi,\eta) \quad (3.17)$$

with $\mathcal{E}_l = \left(E - \frac{\hbar^2 (k_\zeta^l)^2}{2m^*_{l,\zeta\zeta}}\right)$.

The solutions of this eigenvalue problem are the eigenenergies $\mathcal{E}_m^l$ and corresponding eigenfunctions $\chi_m^l(\xi, \eta)$. The quantum number m numbers the eigenenergies from zero to infinity. In practice, the eigenvalue problem is solved numerically for a predefined set of eigenenergies and corresponding eigenfunctions.

The dispersion relation $E = \mathcal{E}_m^l + \frac{\hbar^2 (k_\zeta^{l,m})^2}{2m^*_{l,\zeta\zeta}}$ is plotted in figure 3.3. Due to the quadratic dependence of E on $k_\zeta^{l,m}$ and the discreteness of $\mathcal{E}_m^l$, the plot shows a set of nested parabolas. Each parabola represents a subband of the conduction band and has a minimum value at $\mathcal{E}_m^l$. Extracting the wavevector $k_\zeta^{l,m}$ from the dispersion relation, gives:

$$k_\zeta^{l,m} = \pm\frac{\sqrt{2m^*_{l,\zeta\zeta}(E - \mathcal{E}_m^l)}}{\hbar} \quad (3.18)$$

The wavevector is in general a complex value, because the total amount of energy E in the system can be lower than a subband energy $\mathcal{E}_m^l$.

Reconstructing the solutions in the leads

The solutions $\psi_l(\xi, \eta, \zeta)$ in the leads are now reconstructed using the previous substitution $\psi_l(\xi, \eta, \zeta) = \chi_m^l(\xi, \eta)\zeta_l(\zeta)$. All solutions are taken into account by summing over all subbands for both the incoming as the outgoing plane waves.

$$\psi_l(\xi,\eta,\zeta) = \sum_{m=0}^{\infty}\left(a_m^l e^{-ik_\zeta^{l,m}\zeta} + b_m^l e^{ik_\zeta^{l,m}\zeta}\right)\chi_m^l(\xi,\eta) \quad (3.19)$$

Each a_m^l coefficient denotes the complex amplitude of the incoming plane wave in lead l and subband m. Each b_m^l coefficient denotes the complex amplitude of the

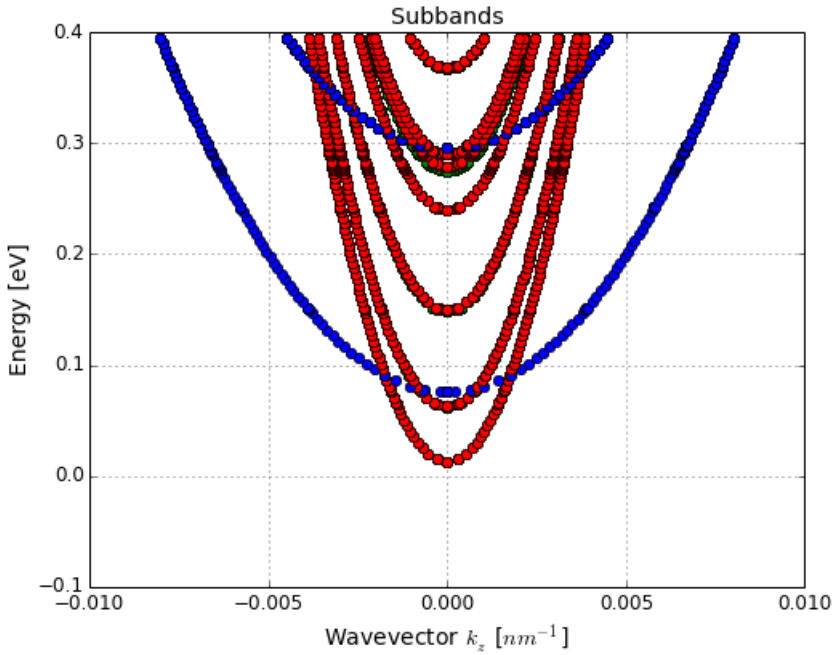


Figure 3.3: Subbands in the anisotropic case for a 5 nm by 5 nm nanowire. Green: 100-valley, red: 010-valley, blue: 001-valley. The green subbands of the 100-valley are not clearly visible because they are degenerate with the 010-valley. The 001-valley subbands are less steep because the 001-valley has a high effective mass in the ζ-direction. The 001-valley subbands lie also higher because the effective mass in the ξ- and η-direction is small for the 001-valley.

outgoing plane wave in lead l and subband m. With equation 3.19 we have found the analytical solution for the wavefunctions in the leads. The analytical solutions in the leads look the same as for the isotropic case, however the numerical solutions of the eigenfunctions $\chi_m^l(\xi, \eta)$ can take a different form because the eigenvalue problem changed compared to the isotropic case.

3.3.2 Step 2: Obtain quantum transmitting boundary conditions

In this step we derive suitable boundary conditions on the open boundaries of the device region. Then all the boundary conditions will be defined for the device region and the problem in the device region can be solved. In the QTBM paper of C.S. Lent and D.J. Kirkner [27] the sought boundary conditions on the open boundaries are called *quantum transmitting boundary conditions*. Considering that the final state is a continuous wave which extends over the device region as well as the leads, one has to guarantee that the wave makes continuous transitions over the open boundaries D_l. In mathematical terms this translates into the continuity conditions for the

wavefunctions on both sides of the boundary ψ_0 and ψ_l:

$$\begin{cases} \psi_0\Big|_{D_l} = \psi_l\Big|_{D_l} \\ \left(\frac{1}{M_0^*}\nabla\psi_0\right)\cdot\hat{\mathbf{n}}_{D_l}\Big|_{D_l} = \left(\frac{1}{M_l^*}\nabla\psi_l\right)\cdot\hat{\mathbf{n}}_{D_l}\Big|_{D_l} \end{cases}$$

The continuity conditions in the anisotropic case are different from the continuity conditions in the isotropic case (equation 3.10b), because we started from a different Schrödinger equation with the effective mass tensor inside of the divergence. In general, the effective mass tensor M_l^* in the leads is different from the effective mass tensor in the device region M_0^*. The second continuity condition can be rewritten as:

$$\left(\frac{1}{M_0^*}\nabla\psi_0\right)\cdot\hat{\mathbf{n}}_{D_l}\Big|_{D_l} = \left(\begin{bmatrix} \frac{1}{m^*_{l,\xi\xi}} & \frac{1}{m^*_{l,\xi\eta}} & 0 \\ \frac{1}{m^*_{l,\eta\xi}} & \frac{1}{m^*_{l,\eta\eta}} & 0 \\ 0 & 0 & \frac{1}{m^*_{l,\zeta\zeta}} \end{bmatrix}\begin{bmatrix} \frac{\partial\psi_l}{\partial\xi} \\ \frac{\partial\psi_l}{\partial\eta} \\ \frac{\partial\psi_l}{\partial\zeta} \end{bmatrix}\right)\cdot\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\Bigg|_{D_l} \tag{3.20a}$$

$$= \frac{1}{m^*_{l,\zeta\zeta}}\frac{\partial\psi_l}{\partial\zeta}\Big|_{D_l} \tag{3.20b}$$

In equation 3.20a the tensor in the lead coordinate system satisfies the assumption made in 3.15. In equation 3.20b we can substitute the analytical solution for $\psi_l(\xi,\eta,\zeta)$ found in equation 3.9. By taking into account the orthonormality of the eigenfunctions $\chi_m^l(\xi,\eta)$, we can find[5] an expression for the b_m^l coefficients:

$$b_m^l = \int_{D_l}\chi_m^l(\xi,\eta)\psi_l(\xi,\eta,\zeta=0)dS - a_m^l \tag{3.21}$$

Applying the first continuity condition in the integrand of equation 3.21, results in an expression for b_m^l only in ψ_0:

$$b_m^l = \int_{D_l}\chi_m^l(\xi,\eta)\psi_0(\xi,\eta,\zeta=0)dS - a_m^l \tag{3.22}$$

Taking expression 3.22 into account, results in:

$$\left(\frac{1}{M_0^*}\nabla\psi_0\right)\cdot\hat{\mathbf{n}}_{D_l}\Big|_{D_l} = \frac{1}{m^*_{l,\zeta\zeta}}\sum_{m=0}^{\infty} ik_\zeta^{l,m}\left(-2a_m^l + \int_{D_l}\chi_m^l(\xi,\eta)\psi_0(\xi,\eta,\zeta=0)dS\right)\chi_m^l(\xi,\eta) \tag{3.23}$$

In the next section we reformulate the problem such that the problem in the device region can be solved numerically for ψ_0 which will, together with the solutions ψ_l in the leads result in a solution of the total state Ψ over the system domain Ω. A numerical solution is necessary because the Schrödinger equation in the device region is in general not analytically solvable. The potential on the device region does not have an analytical form.

[5] The derivation of the expression for b_m^l is given in appendix A.1

3.4 Finite element method

The finite element method is a method to solve boundary value problems numerically. The boundary value problem is a stationary partial differential equation (PDE) with specified boundary conditions on a given geometry. We rely on a numerical method because the boundary value problem cannot be solved analytically to find the exact solution. The finite element method allows to find an approximate solution of the boundary value problem by dividing the system geometry in a finite set of elements. The problem is thus divided into smaller, easier to solve problems. These smaller problems are then solved together in a consistent way to find the the approximate solution on the whole system. The finite element method is suitable in this case because the method allows to handle complex geometries (by splitting the geometry in smaller parts) and can capture different properties in different domains of the system.

3.4.1 Variational boundary value problem

The finite element method is a special case of the Galerkin method from variational calculus. The first step in the finite element method is thus to transform the partial differential equation into a variational problem. Variational problems are defined in terms of trial and test functions. The variational formulation is obtained by first multiplying the *residual* of the PDE with a test function which satisfies the Dirichlet boundary conditions of the boundary value problem, and then integrating over the system domain[30]. Equation 3.24 restates the Schrödinger equation for ψ_0 in the device region:

$$-\frac{\hbar^2}{2}\nabla\cdot\left(\frac{1}{M_0^*}\nabla\right)\psi_0 + V_0\psi_0 = E\psi_0 \tag{3.24}$$

The residual R of this Schrödinger equation is given by:

$$R(\psi_0) = \left(-\frac{\hbar^2}{2}\nabla\cdot\left(\frac{1}{M_0^*}\nabla\right) + V_0 - E\right)\psi_0 = 0 \tag{3.25}$$

Multiplying the residual $R(\psi_0)$ with a test function $\overline{\psi}$ and then integrating over the system domain Ω_0 results in:

$$\int_{\Omega_0}\overline{\psi}\left(-\frac{\hbar^2}{2}\nabla\cdot\left(\frac{1}{M_0^*}\nabla\right) + V_0 - E\right)\psi_0 d\Omega = 0 \tag{3.26}$$

In equation 3.26 we have rewritten the Schrödinger equation in the device region in variational formulation. The variational formulation states that the residual $R(\psi_0)$ should be orthogonal to the test function $\overline{\psi}$. The trial functions ψ_0 are defined by the following function space V, referred to as the *trial space*:

$$V = \left\{\psi_0 \in F(\Omega_0) : \psi_0 = c\Big|_{\partial\Omega_0 - \sum_l D_l}\right\} \tag{3.27}$$

with c a constant value defined on the Dirichlet boundary $\partial\Omega_0 - \sum_l D_l$ and $F(\Omega)$ a function space defined on Ω_0. The test functions $\overline{\psi}$ are defined by the following function space $\hat{V}$, referred to as the *test space*:

$$\hat{V} = \left\{ \overline{\psi} \in F(\Omega_0) : \overline{\psi} = 0 \Big|_{\partial\Omega_0 - \sum_l D_l} \right\} \tag{3.28}$$

The hat on $\hat{V}$ denotes that the test space is built from the same basis functions as the trial space, except the values on the (Dirichlet) boundaries may be different. The variational boundary value problem can now be stated as follows:

Find $\psi_0 \in V$, such that:

$$\int_{\Omega_0} \overline{\psi} \left(-\frac{\hbar^2}{2} \nabla \cdot \left(\frac{1}{M_0^*} \nabla \right) + V_0 - E \right) \psi_0 d\Omega = 0 \tag{3.29}$$

$\forall\ \overline{\psi} \in \hat{V}$.

However, this variational boundary value problem is incomplete. The problem only incorporates the Dirichlet boundary conditions and not the Robin boundary conditions. The Dirichlet boundary conditions appear in 3.27 as part of the definition of the trial space V and are therefore *essential* boundary conditions. To incorporate the Robin boundary conditions in the variational problem statement, we bring equation 3.29 in weak formulation.

Weak formulation

The variational formulation and the underlying PDE require solutions with continuous derivatives. By rewriting the variatonal problem as a *weak* variational problem, the problem statement will also allow solutions with discontinuous derivatives[28]. This is important because the piecewise solution over the elements will in general not have a continuous derivative. The name weak formulation originates from the weaker continuity requirement for the solutions of the variational problem. We can transform the obtained variational formulation of the Schrödinger equation 3.26 in a weak variational formulation by using integration by parts on the first integral. In three dimensions integration by parts is equivalent to using Green's first identity.

$$\frac{\hbar^2}{2} \int_{\Omega_0} \nabla\overline{\psi} \cdot \left(\frac{1}{M_0^*} \nabla\psi_0 \right) d\Omega + \int_{\Omega_0} \overline{\psi}(V_0 - E)\psi_0 d\Omega = \frac{\hbar^2}{2} \oint_{\partial\Omega_0} \overline{\psi} \left(\frac{1}{M_0^*} \nabla\psi_0 \right) \cdot \hat{\mathbf{n}}_{\partial\Omega} dS \tag{3.30}$$

In equation 3.30 we have obtained the weak form of the Schrödinger equation in the device region. The test function $\overline{\psi}$ is assumed to be zero on the Dirichlet boundary $(\partial\Omega_0 - \sum_l D_l)$. As a consequence, the integral on the right hand side of equation 3.30 is only nonzero on the Robin boundaries $\sum_{D_l}$:

$$\frac{\hbar^2}{2} \int_{\Omega_0} \nabla\overline{\psi} \cdot \left(\frac{1}{M_0^*} \nabla\psi_0 \right) d\Omega + \int_{\Omega_0} \overline{\psi}(V_0 - E)\psi_0 d\Omega = \frac{\hbar^2}{2} \sum_l \oint_{D_l} \overline{\psi} \left(\frac{1}{M_0^*} \nabla\psi_0 \right) \cdot \hat{\mathbf{n}}_{D_l} dS \tag{3.31}$$

In equation 3.31 we find the Robin boundary conditions on the right hand side of the equations. The term in the integrand can be replaced with the QTBCs found in equation to find a fully defined problem for ψ_0. The Robin boundary condition are, in contrast with the Dirichlet boundary conditions, *natural* boundary conditions because they enter in the variational problem statement itself, instead of the definition of the trial space. As a consequence, solving the PDE will always result in a solution where the Dirichlet boundary condition is satisfied, but the solution of the BVP will only try to fulfill the Robin boundary conditions as good as possible. The variational boundary value problem can now be stated as follows:

Find $\psi_0 \in V$, such that:

$$\frac{\hbar^2}{2}\int_{\Omega_0} \nabla\overline{\psi}\cdot\left(\frac{1}{M_0^*}\nabla\psi_0\right)d\Omega + \int_{\Omega_0}\overline{\psi}(V_0-E)\psi_0 d\Omega = \frac{\hbar^2}{2}\sum_l \oint_{D_l}\overline{\psi}\left(\frac{1}{M_0^*}\nabla\psi_0\right)\cdot\hat{\mathbf{n}}_{D_l}dS \tag{3.32}$$

$\forall\ \overline{\psi} \in \hat{V}$ and with the Robin or quantum transmitting boundary conditions given by 3.23.

We have applied three steps to make the PDE finite element method-ready: first, we multiplied the PDE with a test function, second, we integrated the resulting PDE over the system Ω and third, we performed an integration by parts (using Green's first identity). This whole transformation from PDE to weak formulation is done manually on paper.

3.4.2 Discretization

The continuous variational problem in equation 3.34 needs to be discretized to solve the problem with a computer. If ψ_0 is the exact solution of the PDE, equation 3.34 will be zero for every test function $\overline{\psi}$. If ψ_0 is an approximate (numerical) solution of the PDE, equation 3.26 is not zero anymore for every test function $\overline{\psi}$. We require that the residual should be orthogonal to the test functions in a test space $\hat{V}_h$. The test space V_h is a function space of basis functions ϕ defined on elements of the domain Ω. The index h refers to the element size of one element. The set of basis functions $\{\phi_i\}$ for V_h is defined as:

$$\phi_i(\mathbf{r}_j) = \begin{cases} 1, \text{ if } i = j, \\ 0, \text{ if } i \neq j \end{cases}$$

The trial functions are part of the function space V_h with the same basis functions. We can thus write the possible solution as a linear combination of the basis functions ϕ_i:

$$\psi_{0h}(\mathbf{r}) = \gamma_1\phi_1(\mathbf{r}) + \gamma_2\phi_2(\mathbf{r}) + ... + \gamma_N\phi_3(\mathbf{r}) \tag{3.33}$$

The constants γ_i, $i = 1, 2, ..., N$ are unknowns to be determined. We can substitute the expression 3.33 for ψ_0 in the weak formulation found in 3.34. We do this for N different choices of test functions. The resulting equations are linear because

the Schrödinger equation is linear. By choosing the N different choices of test functions to be the N basis functions used in 3.33, we obtain a system of N algebraic equations $\mathbf{A}\gamma = \mathbf{b}$ which can be solved for the unknown constants γ_i. With the constants γ_i known, we can then evaluate the approximate solution ψ_{0h} using 3.33.

The discrete variational boundary value problem can now be stated as follows:

Find $\psi_{0h} \in V_h \subset V$, such that:

$$\frac{\hbar^2}{2}\int_{\Omega_0} \nabla\overline{\psi}\cdot\left(\frac{1}{M_0^*}\nabla\psi_{0h}\right)d\Omega + \int_{\Omega_0}\overline{\psi}(V_0 - E)\psi_{0h}d\Omega = \frac{\hbar^2}{2}\sum_l \oint_{D_l}\overline{\psi}\left(\frac{1}{M_0^*}\nabla\psi_{0h}\right)\cdot\hat{\mathbf{n}}_{D_l}dS \tag{3.34}$$

$\forall\ \overline{\psi} \in \hat{V}_h \subset \hat{V}$ and with the Robin or quantum transmitting boundary conditions given by 3.23.

Dividing the system in a finite set of elements is referred to as *meshing* the domain. The elements of the mesh are connected to eachother with nodes. There are various types of elements and function spaces available in finite element libraries. The shape functions used previously are the piecewise linear hat functions ϕ_i. Other piecewise polynomial functions are also possible. The shape functions are defined on reference elements and mapped to the elements of the mesh. On the boundary $\partial\Omega_0$ the elements are normally triangular in shape (2D) and in the volume Ω_0 tetrahedral in shape (3D). A correct choice of the size of the elements is important. Decreasing

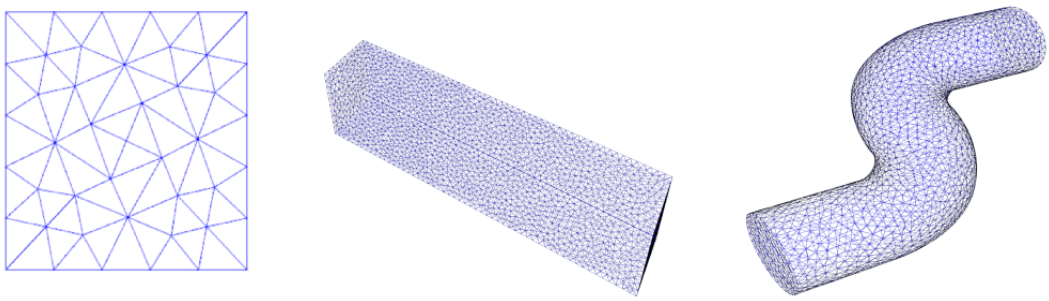


Figure 3.4: Example meshes

the size of the elements of the mesh will narrow the gap between the numerical solution and the exact solution, but will also increase computational time. The size of the elements is thus chosen just small enough to obtain an acceptable result for ψ_0 at reasonable computational time. By virtue of the effective mass theorem we used for the Schrödinger equation, the solution for ψ_0 does not change on the order of the lattice constant. As a result, a rougher mesh can be used to find a solution close to the exact solution. This is the practical advantage of the effective mass theorem.

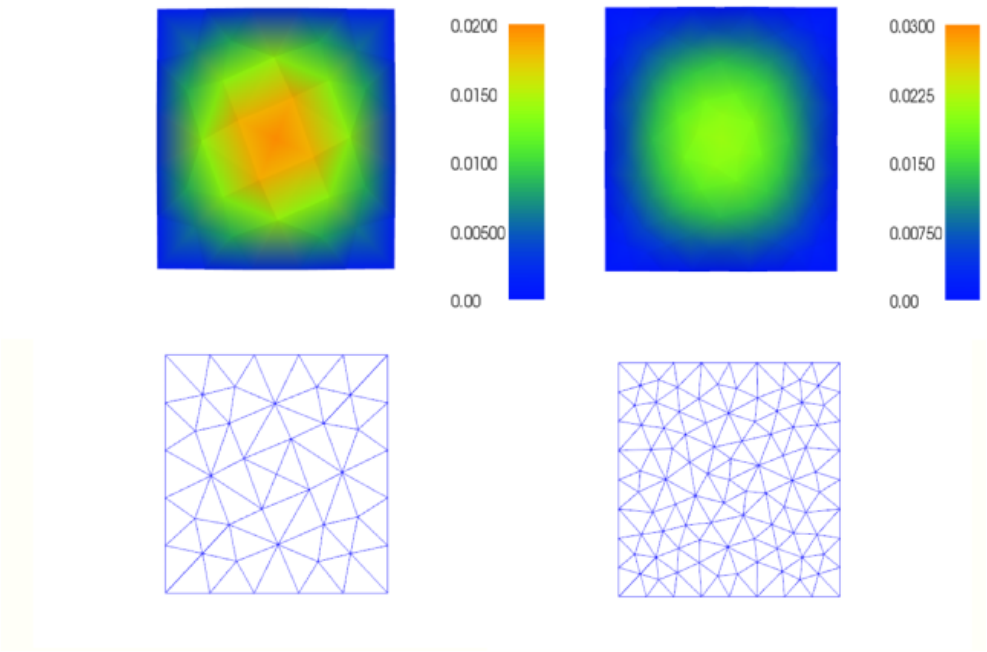


Figure 3.5: Figure showing the difference in obtained solution for a finer mesh and rougher mesh.

3.4.3 Solving the system of equations

As a rule of thumb, the system of equation $\mathbf{A}\gamma = \mathbf{b}$ will be well-conditioned if the elements have the same size everywhere on the mesh. If the system of equations is ill-conditioned, a small variation in the inputs ($\mathbf{A}$ or $\mathbf{b}$) will result in a big variation in the solution for γ and thus for ψ_0. The linear system $\mathbf{A}\gamma = \mathbf{b}$ can be solved using either a direct approach (e.g. LU solver) or an iterative approach (e.g. Krylov solver). It is important that the solver can handle sparsity of the linear system for efficient computation.

3.5 Reformulated problem

In this section we reformulate the problem as it is used to find the state Ψ in Ω. The state Ψ is spread over the leads as well as the device region. Using the QTB-method we solve for the wavefunctions ψ_l in the leads analytically and find QTBCs on the open boundaries of the device region. An expression for the wavevector k_ζ^l and 2D eigenvalue problems (EVP) for the wavefunctions $\chi_m^l(\xi, \eta)$ on the cross-section of the leads were also found from the analytical solution in the leads. Inserting the QTBCs in the weak formulation found in 3.34, we find the following discrete variational problem in the device region, to be solved with the finite element method.

Find $\psi_{0h} \in V_h \subset V$, such that:

$$\frac{\hbar^2}{2}\int_{\Omega_0} \nabla\overline{\psi}\cdot\left(\frac{1}{M_0^*}\nabla\psi_{0h}\right)d\Omega + \int_{\Omega_0}\overline{\psi}(V_0-E)\psi_{0h}d\Omega$$
$$= \frac{\hbar^2}{2}\sum_l \oint_{D_l}\overline{\psi}\frac{1}{m^*_{l,\zeta\zeta}}\sum_{m=0}^{\infty} ik_\zeta^{l,m}\left(-2a_m^l + \int_{D_l}\chi_m^l(\xi,\eta)\psi_{0h}(\xi,\eta,\zeta=0)dS\right)\chi_m^l(\xi,\eta)dS$$
$$\forall\ \overline{\psi}\in\hat{V}_h\subset\hat{V}.$$

The problem statement in 3.36 is a boundary value problem because the boundary conditions are part of the problem statement. The Dirichlet boundary conditions on the closed boundary is incorporated in the definition of the function spaces V_h and $\hat{V}_h$ (and are therefore essential boundary conditions), while the QTBCs on the open boudaries are incorporated on the right hand side of the variational problem statement (and are therefore natural boundary conditions).

In the discrete variational problem, the potential V_0 follows from the self-consistent solution of the Poisson equation. The incoming amplitudes a_m^l follow from the normalization of the total states Ψ, which will be done in section . The effective mass tensor M_0^* and the effective masses $m^*_{l,\zeta\zeta}$ are known from the effective masses of the material in the device and the leads respectively. In practice, the eigenfunctions $\chi_m^l(\xi,\eta)$ are found numerically by solving the 2D eigenvalue problem in the leads, although analytical solutions for $\chi_m^l(\xi,\eta)$ exist.

The reformulated problem states that we need to inject a superposition of all the subbands. In practice, we consider only electron waves injected through one lead l_inj and one subband m_inj at a time. The injection through one lead is validated by the device operation of the nanowire transistor where in normal device operation only electrons flow in from the source. The injection of only one subband at a time is merely done for practical purposes. Applying a superposition of incoming subbands will also result in a valid solution, but this solution will be a superposition of the solutions which would be obtained when applying the subbands separately. The incoming amplitudes a_m^l are therefore all zero, except from one l_{inj} and one m_{inj} at a time. This simplifies the reformulated problem to the following problem statement:

Find $\psi_{0h} \in V_h \subset V$, such that:

$$\int_{\Omega_0} \nabla\overline{\psi}\cdot\left(\frac{1}{M_0^*}\nabla\psi_{0h}\right)d\Omega + \frac{2}{\hbar^2}\int_{\Omega_0}\overline{\psi}(V_0-E)\psi_{0h}d\Omega$$
$$= -2a_{m\_inj}^{l\_inj}\frac{1}{m^*_{l\_inj,\zeta\zeta}}ik_\zeta^{l\_inj,m\_inj}\oint_{D_{l\_inj}}\overline{\psi}\chi_{m\_inj}^{l\_inj}(\xi,\eta)dS$$
$$+\sum_l\oint_{D_l}\overline{\psi}\frac{1}{m^*_{l,\zeta\zeta}}\sum_{m=0}^{\infty}ik_\zeta^{l,m}\left(\int_{D_l}\chi_m^l(\xi,\eta)\psi_{0h}(\xi,\eta,\zeta=0)dS\right)\chi_m^l(\xi,\eta)dS$$
$$\forall\ \overline{\psi}\in\hat{V}_h\subset\hat{V}.$$

The sum over the subbands indicates that we need to take into account all subbands from zero to infinity. In practice, we truncate the sum over the subbands to a specific amount of subbands. The more subbands we take into account, the better the solution but the longer the computational time.

3.6 Numerical solution with Python and FEniCS

To solve the discrete variational problem and find numerical solutions for ψ_0, we extend the quantum simulator IMQUS written by my supervisor Maarten Thewissen to include anisotropic effective masses for three-dimensional structures. The quantum simulator uses the following open-source software: Gmsh[10] to define the geometry and make the mesh, the finite element library FEniCS[7, 28] to solve all PDEs, the Python language (version 2.7)[37] for handling all input parameters and all calculations not related to finite elements, PyTables[8] for efficient storage of the calculated states in HDF5-format[41] and Paraview[5] for visualization and postprocessing of the data.

3.6.1 Reformulated problem for FEniCS

To solve the partial differential equations with the finite element method, the quantum simulator uses the finite element library FEniCS. However, at the time of writing FEniCS does not support complex numbers. We can work around this issue by rewriting the problem as two separate problems for the real and the complex part. We insert $\psi_0 = u_r + iu_i$, $k_\zeta^{l,m} = k_{\zeta,r}^{l,m} + ik_{\zeta,i}^{l,m}$ and $\overline{\psi} = v$ in the previously obtained weak formulation and separate the resulting equation in a real and complex part[6]. We have brought the solution directly in linear and bilinear forms as suitable for FEniCS:

$$a_{\mathbb{R}}(u,v) - L_{\mathbb{R}}(v) + i\left(a_{\mathbb{C}}(u,v) - L_{\mathbb{C}}(v)\right) = 0$$

$$\begin{cases} a_{\mathbb{R}}(u,v_1) - L_{\mathbb{R}}(v_1) = 0 \\ a_{\mathbb{C}}(u,v_2) - L_{\mathbb{C}}(v_2) = 0 \end{cases}$$

with:

$$a_{\mathbb{R}}(u,v) = \int_{\Omega_0} \nabla v \cdot \left(\frac{1}{M_0^*}\nabla u_r\right) d\Omega + \frac{2}{\hbar^2}\int_{\Omega_0} v\left(V_0 - E\right) u_r d\Omega$$

$$+\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,r}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_i dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$+\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,i}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_r dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$L_{\mathbb{R}}(v) = 2a_{m\_inj}^{l\_inj} \frac{1}{m^*_{l\_inj,\zeta\zeta}} k_{\zeta,i}^{l\_inj,m\_inj} \left(\oint_{D_{l\_inj}} v\chi_{m\_inj}^{l\_inj}(\xi,\eta) dS\right)$$

[6] The proof we do not need to split the test function $\overline{\psi}$ in a real and complex part as $\overline{\psi} = v_r + iv_i$ is given in appendix A.4.

$$a_{\mathbb{C}}(u,v) = \int_{\Omega_0} \nabla v \cdot \left(\frac{1}{M_0^*}\nabla u_i\right) d\Omega + \frac{2}{\hbar^2}\int_{\Omega_0} v\,(V_0 - E)\,u_i d\Omega$$

$$-\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,r}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_r dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$+\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,i}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_i dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$L_{\mathbb{C}}(v) = -2a_{m\_inj}^{l\_inj} \frac{1}{m^*_{l\_inj,\zeta\zeta}} k_{\zeta,r}^{l\_inj,m\_inj} \left(\oint_{D_{l\_inj}} v\chi_{m\_inj}^{l\_inj}(\xi,\eta) dS\right)$$

We only inject travelling waves. Therefore, the wavevector $k_\zeta^{l\_inj,m\_inj}$ is purely real. As a consequence, $L_{\mathbb{R}}(v)$ is zero and the problem can be simplified to:

$$a_{\mathbb{R}}(u,v) + i\,(a_{\mathbb{C}}(u,v) - L_{\mathbb{C}}(v)) = 0$$

$$\begin{cases} a_{\mathbb{R}}(u,v_1) = 0 \\ a_{\mathbb{C}}(u,v_2) - L_{\mathbb{C}}(v_2) = 0 \end{cases}$$

with:

$$a_{\mathbb{R}}(u,v) = \int_{\Omega_0} \nabla v \cdot \left(\frac{1}{M_0^*}\nabla u_r\right) d\Omega + \frac{2}{\hbar^2}\int_{\Omega_0} v\,(V_0 - E)\,u_r d\Omega$$

$$+\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,r}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_i dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$+\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,i}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_r dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$a_{\mathbb{C}}(u,v) = \int_{\Omega_0} \nabla v \cdot \left(\frac{1}{M_0^*}\nabla u_i\right) d\Omega + \frac{2}{\hbar^2}\int_{\Omega_0} v\,(V_0 - E)\,u_i d\Omega$$

$$-\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,r}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_r dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$+\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} k_{\zeta,i}^{l,m} \left(\int_{D_l} \chi_m^l(\xi,\eta) u_i dS\right)\left(\oint_{D_l} v\chi_m^l(\xi,\eta) dS\right)$$

$$L_{\mathbb{C}}(v) = -2a_{m\_inj}^{l\_inj} \frac{1}{m^*_{l\_inj,\zeta\zeta}} k_{\zeta,r}^{l\_inj,m\_inj} \left(\oint_{D_{l\_inj}} v\chi_{m\_inj}^{l\_inj}(\xi,\eta) dS\right)$$

3.6.2 Python code

The following code snippet contains the main loop to calculate the states in the simulations. The rest of the relevant Python code to which I made extensions is given in appendix B.

```
1  for e in initial_energies:
2      self.schroedinger.update_parameters(energy=e)
3      for l in self.device.descr.lead_boundrs:
4          max_sb_enrgy = max_enrgy
5          for v in self.schroedinger.valleys:
6              self.schroedinger.redefine_effective_mass_tensor_terms(v)
7              for sb in range(self.subbands[l][v]['number of']):
8                  self.schroedinger.update_parameters(inj_lead=l,
       rel_subband=sb)
9                  if self.subbands[l][v]['energies'][sb] < e <
       max_sb_enrgy:
10                     self.schroedinger.redefine_rhs(v)
11                     psi = self.schroedinger.solve(v)
12                     psi_real, psi_imag = psi.split(deepcopy=True)
13                     transm = self.schroedinger.find_transmissions(psi,
       l, sb, v)
14                     print "State nr.: ", nr, "Lead: ", l, "   Subband:
       ", sb, "   Subband energy: ", self.subbands[l][v]['energies'][sb], "
          Energy: ", e, "      Valley: ", v
15                     print "Transmission of this state is: " + str(
       transm)
16                     state['number'] = nr
17                     state['coeffs_real'] = psi_real.vector().array()
18                     state['coeffs_imag'] = psi_imag.vector().array()
19                     state['inj_lead'] = l
20                     state['rel_subband'] = sb
21                     state['inj_subband_energy'] = self.subbands[l][v]['
       energies'][sb]
22                     state['inj_subband'] = self.subbands[l][v]['indices
       '][sb]
23                     state['inj_lead_effective_mass_along_transport'] =
       self.schroedinger.lead_effective_masses_along_transport[l][v]
24                     state['energy'] = e
25                     state['transmission'] = transm
26                     state['degeneracy'] = 2
27                     state['refine_iter'] = 0
28                     state['valley'] = v
29                     state.append()
30                     nr += 1
```

In the main loop we run over all the energies, over all the leads, over all the valleys and over all the calculated subbands for each valley. One calculated state then corresponds to one injected subband of a particular valley in one of the leads for a certain energy in the total system. Each time the energy changes we update the Schrödinger equation and if the valley changes, we update the effective mass tensors. On line 11 we solve the Schrödinger equation for a combination of one total energy, one injection lead, one valley and one subband. On line 13 we calculate the transmission coefficients with the calculated states on line 11. Lines 16 to 30 store

the characteristics of the calculated state in memory and goes on to the next state.

3.7 Normalization

In this section the states Ψ are normalized, because proper wavefunctions are normalized. Because the leads are infinite, the device region can be ignored in the normalization process. Normalizing the longitudinal solutions in the leads allows to find the incoming amplitudes A_m^l. Normalizing the transverse solutions is necessary to find consistent solutions in the numerical solution of the variational problem for ψ_{0h}.

Delta-normalization of the final states Ψ

The incoming amplitudes A_m^l of the final states are expressed as:

$$|A| = \sqrt{\frac{m^*_{l,\zeta\zeta}}{2\pi\hbar^2 k_m^l}} = \sqrt{\frac{1}{L}}\sqrt{\frac{DOS(E)}{2}} \propto \sqrt{\frac{1}{\sqrt{E}}} \tag{3.37}$$

where we have used the 1D-DOS of the incoming states in terms of the wavevector k_m^l: $DOS(E) = \frac{Lm^*_{l,\zeta\zeta}}{\pi\hbar^2 k_m^l}$. The same result can be obtained with a normalization to a delta function. The density of states (DOS) is incorporated in the wavefunctions normalized to a delta function and will not have to be taken into account separately in the calculations of charge density and current.

Normalization of the transverse solutions $\chi_m^l(\xi,\eta)$

The transverse solutions $\chi_m^l(\xi,\eta)$ can be normalized with an ordinary normalization:

$$\oint_{D_l} C_m^l \chi_m^l(\xi,\eta)(C_m^l)^* \left[\chi_m^l(\xi,\eta)\right]^* dS = 1 \tag{3.38}$$

Equation 3.38 results in the normalization constant $C_m^l = \frac{1}{\sqrt{\|\chi_m^l(\xi,\eta)\|}}$ with $\|\chi_m^l(\xi,\eta)\|$ the norm of the eigenfunction which equals $\oint_{D_l} \chi_m^l(\xi,\eta)\left[\chi_m^l(\xi,\eta)\right]^* dS$.

In the previous section we found solutions for the wavefunctions in the leads and derived the QTBCs on the open boundaries of the device region. The problem is now fully defined to solve for the wavefunction Ψ in the whole system Ω. In the following sections we find expressions for the transmission coefficient, the electron charge density and the current in the general device geometry. These derivations are done directly for anisotropic effective mass. Similar derivations for isotropic effective mass can be found in appendix A.3.

3.8 Transmission spectrum

The transmission spectrum gives the transmission through the device region of one injected subband over a range of energies. The transmission spectrum can be

obtained by plotting the transmission coefficients $T(E)$ over a range of energies. The transmission coefficient $T(E)$ gives the transmission probability of the injected subband at this energy to any of the other leads. Vice versa, it is the probabibility of the injected subband not being reflected back into the injection lead. The geometry of the device region changes the transmission probability. In this section we derive an expression for the transmission coefficient $T(E)$.

The transmission coefficient $T(E)$ is given by the outgoing current in all the leads over the incoming current in the incoming lead:

$$T(E) = \frac{I_{out}}{I_{in}} \tag{3.39}$$

The current comes in through the injection lead l' and goes out through any of the other leads. The incoming current I_{in} and the outgoing current I_{out} can be written in terms of the incoming and outgoing probability currents j_{in} and j_{out} in the leads:

$$T(E) = \frac{\sum_l \oint_{D_l} j^l_{out,\zeta} dS}{\oint_{D_{l'}} j^{l'}_{in,\zeta} dS} \tag{3.40}$$

where $j^l_{out,\zeta}$ is the ζ-component of the outgoing probability current perpendicular to D_l in lead l and $j^{l'}_{in,\zeta}$ is the ζ-component of the incoming probability current perpendicular to $D_{l'}$ in the injection lead l'. The probability current $\mathbf{j}$ in the leads for anisotropic effective mass is defined as:

$$\mathbf{j}_l = \frac{\hbar}{2M^*_l i}\left(\psi^*_l \nabla\psi_l - \psi_l \nabla\psi^*_l\right) \tag{3.41}$$

We only need the ζ-component of the probability current vector $\mathbf{j}_l$. Writing out the tensor and vector terms:

$$\mathbf{j}_l = \frac{\hbar}{2i}\begin{bmatrix} \frac{1}{m^*_{l,\xi\xi}} & \frac{1}{m^*_{l,\xi\eta}} & 0 \\ \frac{1}{m^*_{l,\eta\xi}} & \frac{1}{m^*_{l,\eta\eta}} & 0 \\ 0 & 0 & \frac{1}{m^*_{l,\zeta\zeta}} \end{bmatrix}\left(\psi^*_l \begin{bmatrix} \frac{\partial\psi_l}{\partial\xi} \\ \frac{\partial\psi_l}{\partial\eta} \\ \frac{\partial\psi_l}{\partial\zeta} \end{bmatrix} - \psi_l \begin{bmatrix} \frac{\partial\psi^*_l}{\partial\xi} \\ \frac{\partial\psi^*_l}{\partial\eta} \\ \frac{\partial\psi^*_l}{\partial\zeta} \end{bmatrix}\right) \tag{3.42}$$

where we have assumed the separable form of the effective mass tensor in the leads. The ζ-component of the probability current is then given by $j^l_\zeta = \frac{\hbar}{2m^*_{l,\zeta\zeta}i}\left(\psi^*_l \frac{\partial\psi_l}{\partial\zeta} - \psi_l \frac{\partial\psi^*_l}{\partial\zeta}\right)$. The solutions for the wavefunction ψ_l in the leads found in equation 3.19 give the incoming and outgoing wavefunctions:

$$\begin{cases} \psi^{l'}_{in}(\xi,\eta,\zeta) = a^{l'}_{m'} e^{-ik^{l',m'}_\zeta \zeta}\chi^{l'}_{m'}(\xi,\eta) \\ \psi^l_{out}(\xi,\eta,\zeta) = \sum_{m=0}^{\infty} b^l_m e^{ik^{l,m}_\zeta \zeta}\chi^l_m(\xi,\eta) \end{cases}$$

where we have assumed an injection only through one lead l' and one subband m'. Using these wavefunctions in the expression for the probability current we find:

$$j^{l'}_{in,\zeta} = \frac{-\hbar}{2m^*_{l,\zeta\zeta}}\Big((a^{l'}_{m'})^* \chi^{l'}_{m'}(\xi,\eta) e^{ik^{l',m'}_\zeta \zeta} a^{l'}_{m'}\chi^{l'}_{m'}(\xi,\eta) k^{l',m'}_\zeta e^{-ik^{l',m'}_\zeta \zeta} + a^{l'}_{m'}\chi^{l'}_{m'}(\xi,\eta) e^{-ik^{l',m'}_\zeta \zeta}(a^{l'}_{m'})^*\chi^{l'}_{m'}(\xi,\eta) k^{l',m'}_\zeta e^{ik^{l',m'}_\zeta \zeta}\Big)$$

$$j^l_{out,\zeta} = \frac{\hbar}{2m^*_{l',\zeta\zeta}} \left(\sum_{m=0}^{\infty} (b^l_m)^* \chi^l_m(\xi,\eta) e^{-ik^{l,m}_\zeta \zeta} \sum_{m=0}^{\infty} b^l_m \chi^l_m(\xi,\eta) k^{l,m}_\zeta e^{ik^{l,m}_\zeta \zeta} + \sum_{m=0}^{\infty} b^l_m \chi^l_m(\xi,\eta) e^{ik^{l,m}_\zeta \zeta} \sum_{m=0}^{\infty} (b^l_m)^* \chi^l_m(\xi,\eta) k^{l,m}_\zeta e^{-ik^l_\zeta \zeta} \right)$$

The incoming probability current is negative because the ζ-axis is defined outward of the device region. The eigenfunctions on the boundary $\chi^l_m(\xi,\eta)$ are assumed real, because the phase factor can be incorporated in the complex amplitude a^l_m. We can then calculate the transmission coefficient $T(E)$ given in equation 3.40. Using the orthonormality of the subbands $\oint_{D_l} \chi^l_m(\xi,\eta)\chi^l_n(\xi,\eta)dS = \delta_{mn}$:

$$T(E) = \frac{\sum_l \frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} |b^l_m|^2 k^{l,m}_\zeta}{\frac{1}{m^*_{l',\zeta\zeta}} |a^{l'}_{m'}|^2 k^{l',m'}_\zeta} \tag{3.45}$$

Together with the expression for the b^l_m coefficients derived in A.1, we have found an expression to calculate the transmission coefficient $T(E)$. The transmission coefficient $T(E)$ can be calculated after the solutions for the wavefunctions ψ_{0h} have been found by solving the discrete variational problem for ψ_{0h} with the finite element method, as explained in section 3.4. The transmission coefficient $T_{l'\to l}(E)$ from the injection lead l' to a specific lead l, can be found by:

$$T_{l'\to l}(E) = \frac{\frac{1}{m^*_{l,\zeta\zeta}} \sum_{m=0}^{\infty} |b^l_m|^2 k^{l,m}_\zeta}{\frac{1}{m^*_{l',\zeta\zeta}} |a^{l'}_{m'}|^2 k^{l',m'}_\zeta} \tag{3.46}$$

3.9 Electron charge density and current

We assume ballistic transport which allows the Fermi-Dirac distribution of the injection lead $f^{l'}_{FD}(E)$ to be used for the calculation of charges and currents in the device. We sum over all injected subbands for all valleys and injection leads to calculate the charge and current. The electron charge density in the device can be calculated from the obtained wavefunctions $\psi^{l,v}_m$ with the following formula:

$$n_e = 2e \sum_l \sum_v \sum_m \int_{E^{l,v}_m}^{\infty} \left|\psi^{l,v}_m(E)\right|^2 f^l_{FD}(E)dE \tag{3.47}$$

where the factor 2 accounts for the spin degeneracy. The current in lead l' in the ballistic case can be calculated using the obtained transmission coefficients in the previous section:

$$I_{l'} = 2 \sum_l \sum_v \sum_m \int_{E^{l,v}_m}^{\infty} -e v^{l,v}_m(E) T^{l'\to l}_{m,v}(E) f^l_{FD}(E)dE \tag{3.48}$$

where the 2 accounts for spin degeneracy. Ballistic transport neglects the possible scattering in the device and will therefore represent the best case scenario for the calculated on-state current.

3.10 Conclusion

In this chapter we fulfilled the second research objective. We found and set up a suitable theoretical model for the energy filtering in the nanowire superlatticeFET with a geometric superlattice, by extending the QTB method to 3D and (an)isotropic effective mass for a general device geometry. In the derivation of the QTB-method for anisotropic effective mass, we had to make an assumption limiting the form which the effective mass tensor in the leads can take. We explained the finite element method which will be used in the next chapter to obtain numerical solutions for the wavefunctions using Python[37] and FEniCS[7, 28]. We derived expressions for the transmission coefficient, the electron charge density and the current in the ballistic case. In the next chapter we use the model to show the existence of energy filtering in 3D geometric superlattices. The model allows to simulate nanowire superlatticeFETs with various geometric superlattices and to check their energy filtering capacity by plotting the transmission spectra.

Chapter 4

Phase II: Simulating energy filtering with a geometric superlattice

"Is energy filtering possible with a 3D geometric superlattice?"

"How to vary the geometric superlattice to tune energy filtering? Is there an ideal energy filter?"

The goal of this chapter is to show that it is possible to obtain energy filtering with a 3D geometric superlattice. In addition, by simulating nanowires with various geometric superlattices, we show that it is possible to vary the energy filtering by changing the periodic parameters of the geometric superlattice. By varying the periodicity of the geometric superlattice, we will search for a transmission spectrum with ideal energy filtering capacity.

4.1 Simulation model

In figure 4.1 we convert the general model derived in chapter 3 into a model we will use for the simulation of nanowires with a geometric superlattice. For the simulations, the leads are assumed to lie along the [001]-crystal coordinate axis, such that the effective mass tensors in the leads M_l^* expressed in the lead coordinate system are separable, as was required in the derivation of the QTB method in chapter 3.

We start with simulating Si nanowires without a geometric superlattice as a check for the model we set up in the previous chapter. A straight nanowire without a geometric superlattice should result in a transmission spectrum with full transmission at all energies starting from the injected subband energy. The simulation for a straight Si nanowire is done in section 4.2. In section 4.3 we simulate Si nanowires with various

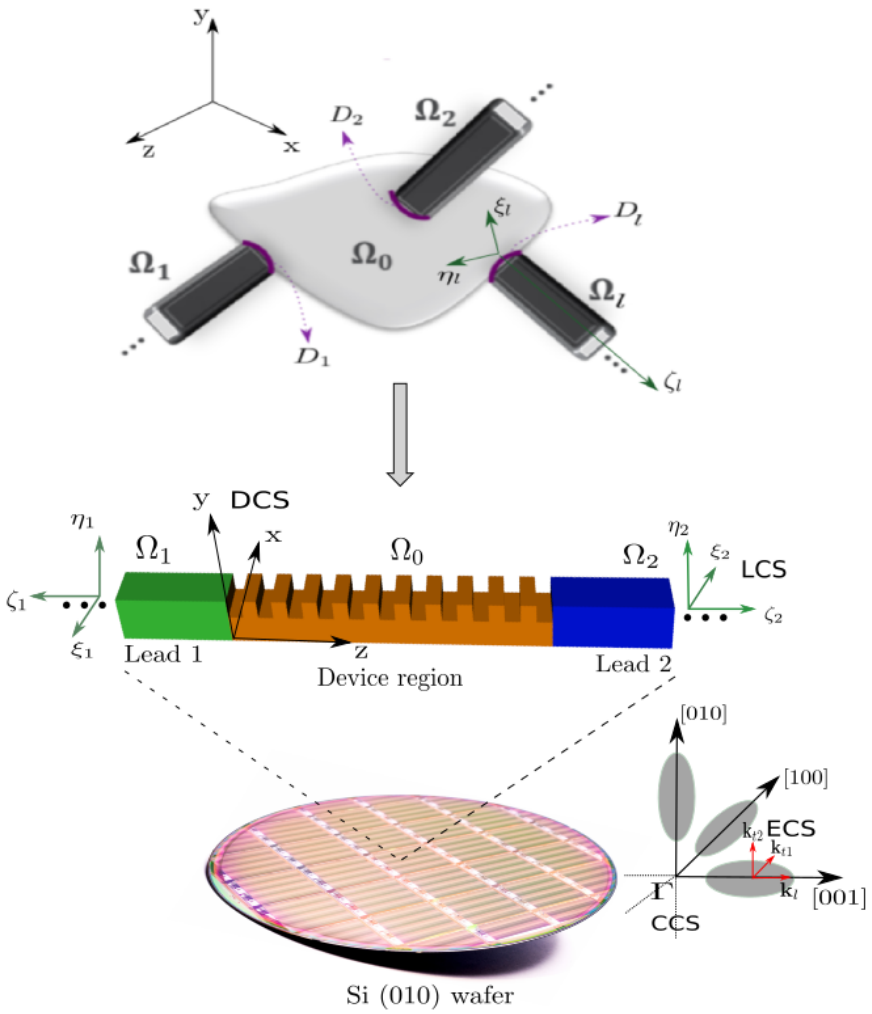


Figure 4.1: Conversion from the general model derived in chapter 3 (top) to a model for a Si nanowire with a geometric superlattice as used for the simulations (middle). As a consequence of the derivation of the 3D QTB method in chapter 2, the ζ_1 and ζ_2 axes along the leads can only lie along one of the crystal coordinate axes [100], [010] or [001]. In the simulations the leads lie along the [001]-direction. The derived 3D QTB method sets no limitation on the orientation of the device region Ω_0. The three equivalent valleys of Si in the crystal coordinate system are shown on the bottom right of the figure. DCS: device coordinate system, LCS: lead coordinate system, CCS: crystal coordinate system, ECS: ellipsoidal or valley coordinate system.

geometric superlattices to understand the influence of the periodic parameters and search for an ideal energy filtering superlattice.

4.2 Si nanowires without a geometric superlattice

4.2.1 Straight Si nanowire with square cross-section

In this section we simulate a straight Si nanowire with square cross-section. To plot the injected subbands in the Si nanowire, we solve the eigenvalue problem (EVP) on the lead boundaries of the device region given by equation 3.3.1. For Si, the EVP is different depending on the valley, because the effective mass tensor M_l^* changes. The solutions for the wavefunctions $\chi_m^l(\xi,\eta)$ on the boundary will therefore be different depending on the valley. In figure 4.2, 4.3 and 4.4 the solutions for $\chi_m^l(\xi,\eta)$ are plotted for the 100-valley, 010-valley and 001-valley respectively. For the 100-valley the wavefunctions with nodes in the [100]-direction (x-direction in figure 4.1 and horizontal in figure 4.2) are preferred at lower energy. For the 010-valley the wavefunctions with nodes in the [010]-direction (y-direction in figure 4.1 and vertical in figure 4.3) are preferred at lower energy. This can be understood from the analytical solution for the subband energy $\mathcal{E}_m^l$:

$$\mathcal{E}_m^l = \frac{n_x^2\hbar^2\pi^2}{2m_x^* l_x^2} + \frac{n_y^2\hbar^2\pi^2}{2m_y^* l_y^2} \tag{4.1}$$

The subband energy $\mathcal{E}_m^l$ is proportional to $\frac{n_x^2}{m_x^*}$ and $\frac{n_y^2}{m_y^*}$. The 100-valley, for instance, has a high effective mass ($0.98m_e$) along the x-direction and a low effective mass ($0.19m_e$) along the y-direction, as shown in figure 4.5. The solutions with a higher number of nodes in the x-direction (high n_x) can thus shift to lower $\mathcal{E}_m^l$ energies. The subbands are the same for the different valleys only the ordering is different. For the 001-valley, superpositions of the eigenfunctions come out, which are also valid solutions of the linear Schrödinger equation. The characteristics of the three equivalent valleys in Si are illustrated in figure 4.5 for subband $m = 1$.

100-valley subbands

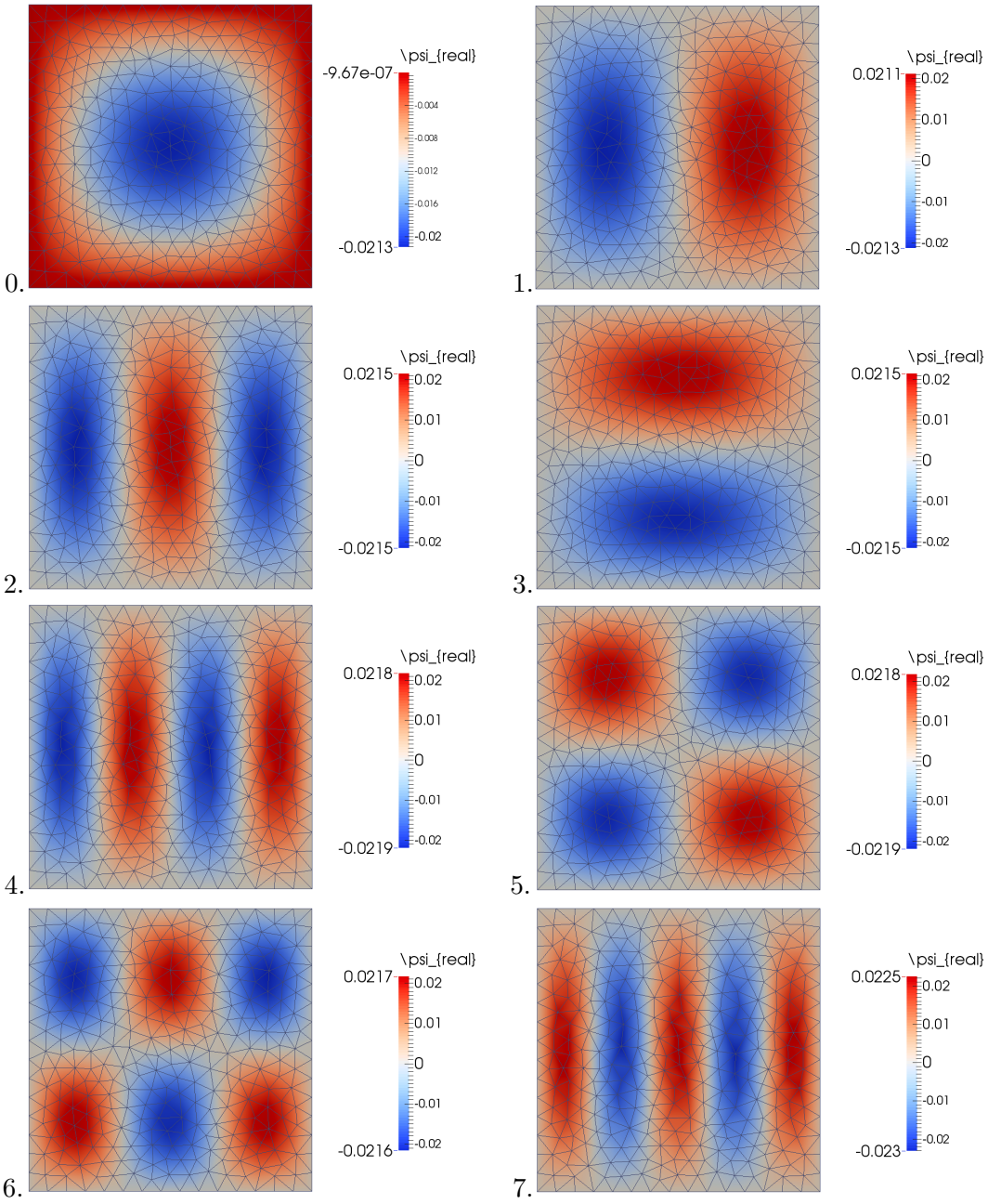


Figure 4.2: First 8 100-valley subbands $\chi_m^l(\xi, \eta)$ with m from 0 to 7 for a Si nanowire. The finite element mesh used to obtain the solution is also plotted.

010-valley subbands

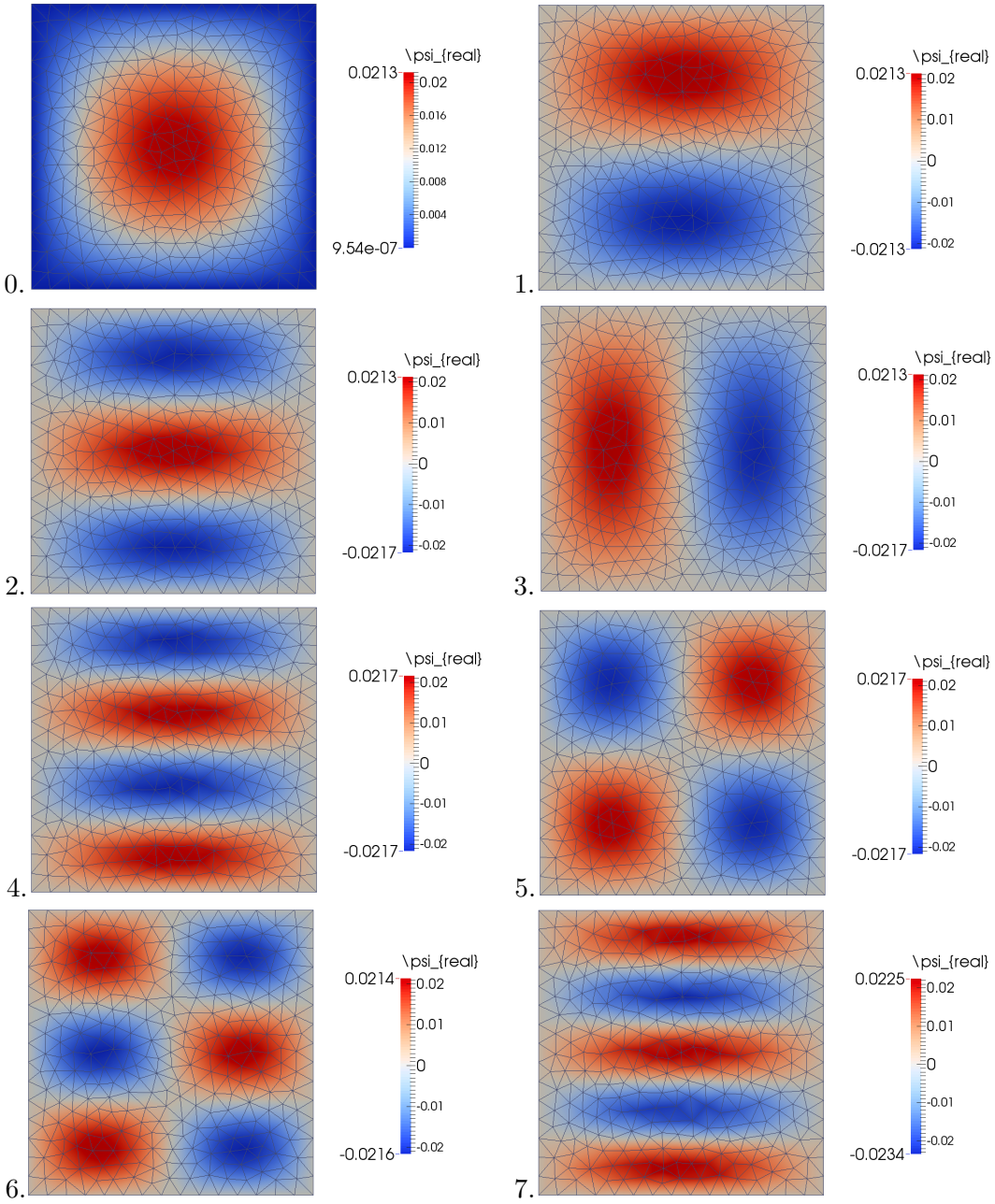


Figure 4.3: First 8 010-valley subbands $\chi^l_m(\xi, \eta)$ with m from 0 to 7 for a Si nanowire. The finite element mesh used to obtain the solution is also plotted.

001-valley subbands

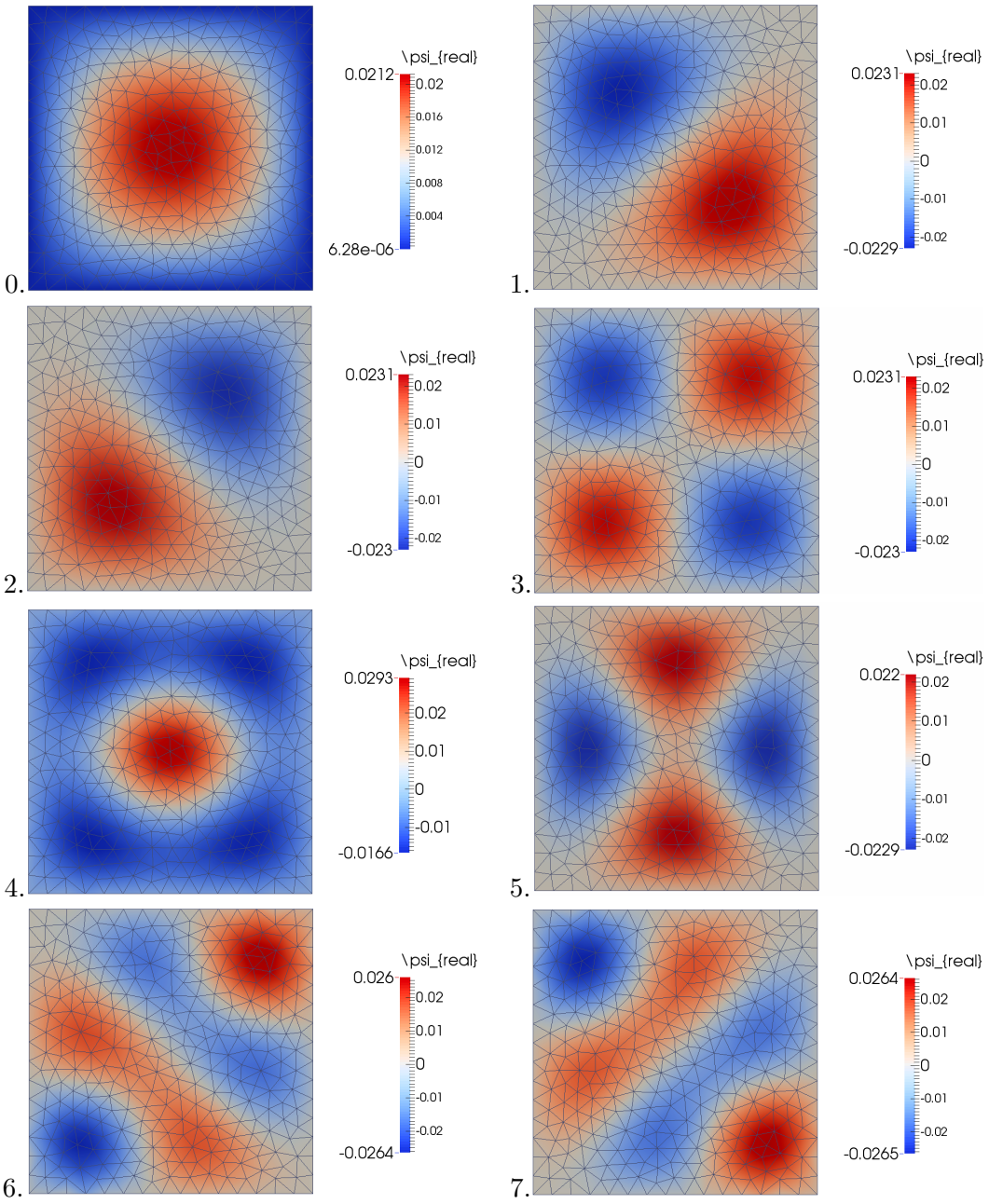


Figure 4.4: First 8 001-valley subbands $\chi_m^l(\xi, \eta)$ with m from 0 to 7 for a Si nanowire. The finite element mesh used to obtain the solution is also plotted.

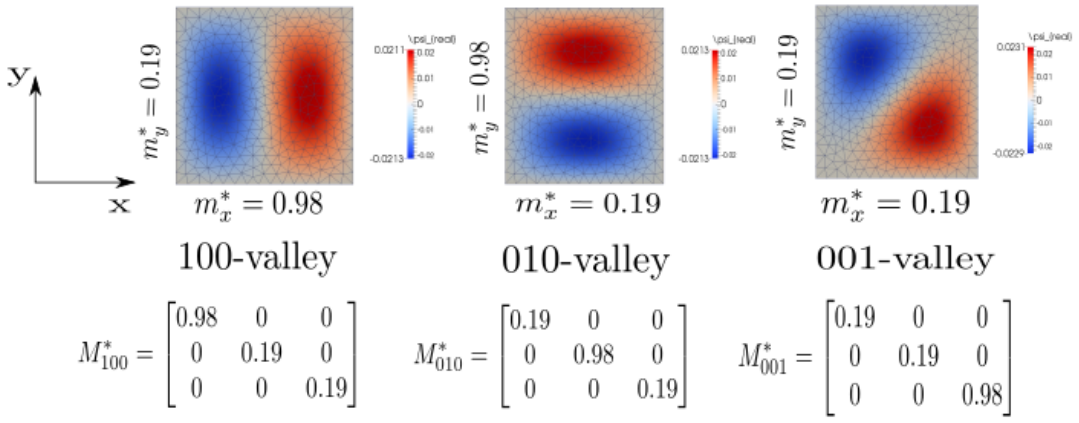


Figure 4.5: Characteristics of the three equivalent valleys in Si, illustrated with subband $m = 1$.

For a straight Si nanowire without a geometric superlattice as shown in figure 4.6 we expect transmission spectra which are 1 for each injected subband starting from the respective subband energy. Figure 4.7 shows the transmission spectra for the injected subbands of the three equivalent valleys of Si. The injected subbands are ordered from low to high subband energy, which mixes the subbands of the three valleys. The solid blue line in each transmission spectrum shows the subband energy. Because we simulated a nanowire with square cross-section, the resulting subband energies for the 100-valley and 010-valley are pairwise degenerate, as can be inferred from the subband energy equation in 4.1 with $l_x = l_y$. We note that only the first subband of the 001-valley is present, which can be explained by the confinement of the nanowire to 5 nm by 5 nm and the low effective mass of the 001-valley in both directions on the cross-section. The green area shows the 1D-DOS $\propto \frac{1}{\sqrt{E}}$ and the red area shows the Fermi-Dirac distribution. The dotted blue line is the Fermi-level in the source which is set to 0 eV. Only the injected subbands with an overlap between the transmission spectrum, the 1D-DOS and the Fermi-Dirac distribution can contribute to the current[1]. In figure 4.7 only the first 5 subbands contribute significantly to the current. The transmission spectra indeed show a transmission of 1 for each injected subband energy, which validates the solver to be used for the simulation of geometric superlattices. Only for higher injected subbands the transmission starts slightly above the subband energy before rising to 1, probably due to the use of a not sufficiently fine mesh on the cross-section because the error is most noticeable for eigenfunctions with many oscillations on one of the cross-sectional directions, i.e. injected subband 9 or 10.

Figures 4.8 to 4.11 show slices of the transmission through the straight Si nanowire for the injected subbands 0, 2, 10 and 11. The numbering of the injected subbands

[1] This follows from the derived formula of the ballistic current in section 3.9.

Figure 4.6: Si nanowire with a width of $w = 5$ nm and a height of $h = 5$ nm without a geometric superlattice.

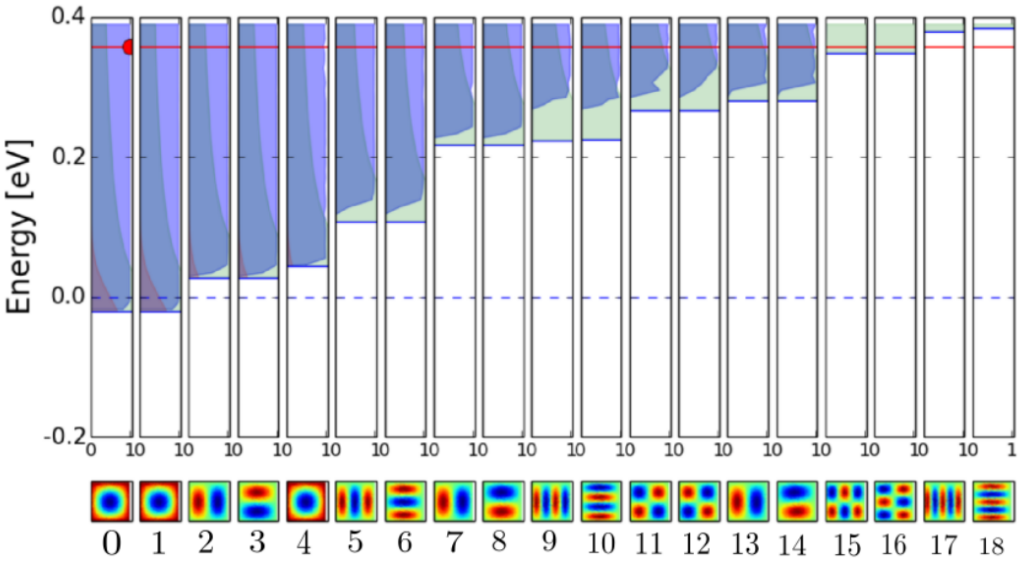


Figure 4.7: Transmission spectra for each injected subband in a 5 nm by 5 nm Si nanowire without a geometric superlattice as shown in 4.6. The injected subbands are ordered from low to high subband energy, which mixes the subbands of the three valleys. The solid blue line in each transmission spectrum shows the subband energy. The green area shows the 1D-DOS and the red area shows the Fermi-Dirac distribution. The dotted blue line is the Fermi-level in the source which is set to 0 eV. The donor doping concentration is $N_D = 10^{19}$ cm^{-3}.

is given underneath the transmission spectra in figure 4.7. In each of the four figures the left side shows the real part of the wavefunction plotted on the solid nanowire and the right side shows a slice through the nanowire to visualize the transmission of the wavefunction inside the nanowire. The slices were obtained using a filter from the data visualization package Paraview[5].

4.2.2 Straight Si nanowire with strong confinement in the height

Figure 4.12 shows a straight Si nanowire with strong confinement in the height down to 2 nm thickness. The width remains the same compared to the straight Si nanowire

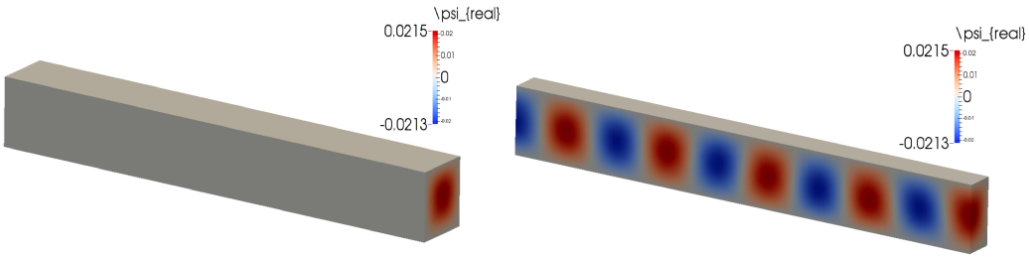


Figure 4.8: Left: real part of the wavefunction for injected subband 0, corresponding to the 100-valley. Right: vertical slice through the nanowire showing the transmission of the wavefunction through the nanowire.

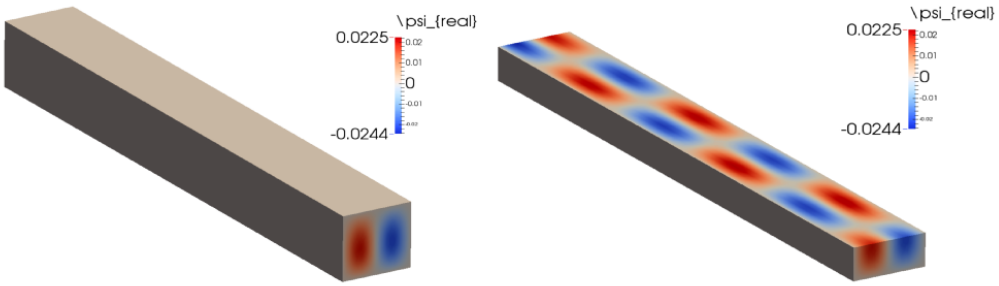


Figure 4.9: Left: real part of the wavefunction for injected subband 2, corresponding to the 100-valley. Right: horizontal slice through the nanowire showing the transmission of the wavefunction through the nanowire.

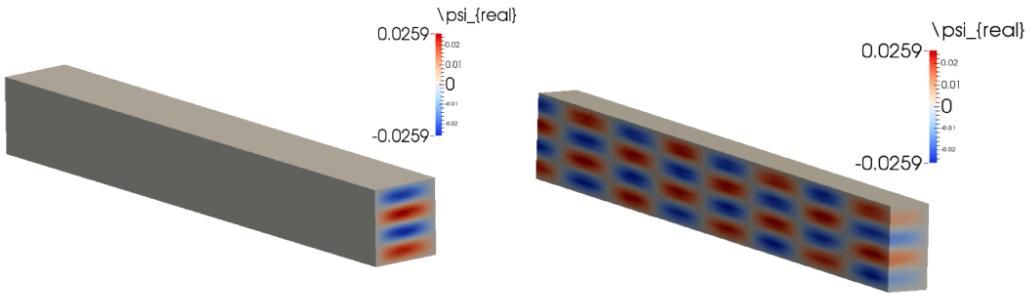


Figure 4.10: Left: real part of the wavefunction for injected subband 10, corresponding to the 010-valley. Right: vertical slice through the nanowire showing the transmission of the wavefunction through the nanowire.

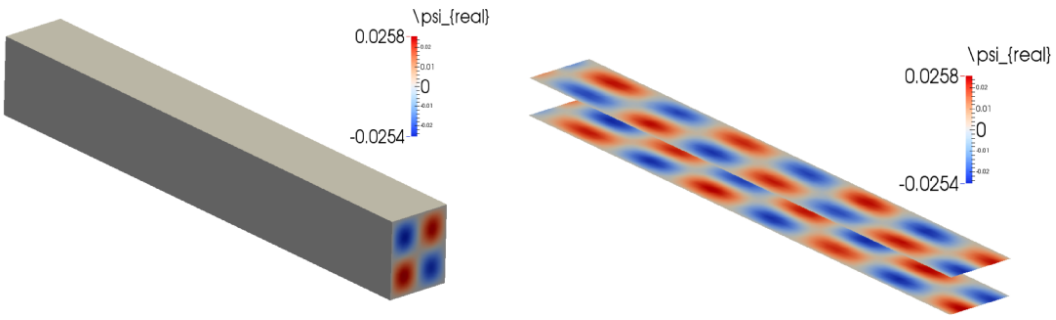


Figure 4.11: Left: real part of the wavefunction for injected subband 11, corresponding to the 100-valley. Right: two horizontal slices through the nanowire showing the transmission of the wavefunction through the nanowire.

in figure 4.6. We choose to confine in the height, because nanometer thick layers can be grown in the vertical dimension using molecular beam epitaxy. Figure 4.13 shows the transmission spectra for the injected subbands of the three equivalent valleys of Si in a nanostrip with height 2 nm and width 5 nm. The subband energies can be extracted from this simulation, although the simulation was run with a rougher finite element mesh, resulting in non-expected values for the transmission coefficient at higher energies. A rougher finite element mesh cannot handle fast oscillations of the wavefunctions very well at higher energies. The main conlusion of figure 4.13 is the isolation of the 010-valley at low energies due to the strong confinement in the height. The 100-valley and 001-valley have a low effective mass along the confinement direction (see figure 4.5) which shifts the 100-valley and 001-valley subband energies up and isolates the 010-valley subbands at lower energies. Confining to 2 nm in the lateral dimension would shift the 010-valley and 001-valley subbands upward and isolate the 100-valley. The isolation of one of the valleys of Si will prove useful in the search for the geometric superlattice with ideal energy filtering characteristics in the next section.

4.3 Si nanowires with a geometric superlattice

To obtain energy filtering we equip the straight Si nanowires with a geometric superlattice. The geometric superlattice can consist of any kind of periodic feature. In contrast with the transmission spectra of straight Si nanowires, we now expect dips to appear in the transmission spectra corresponding to minibandgaps. We start from a superlattice with 10 periods, which should result in minibands and minibandgaps if energy filtering is possible with a 3D geometric superlattice. Gnani et al. used a sequence of 10 barriers and wells in the material superlattice to achieve energy filtering[16].

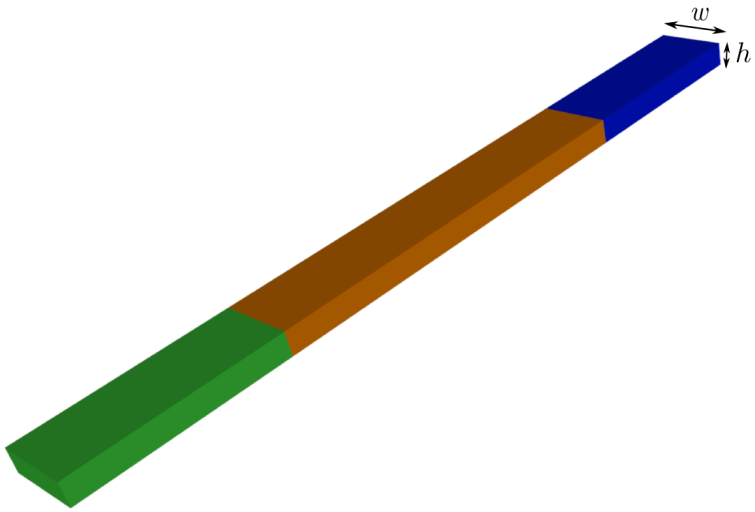


Figure 4.12: Si nanowire with a width of $w = 5$ nm and a strongly confined height of $h = 2$ nm without a geometric superlattice.

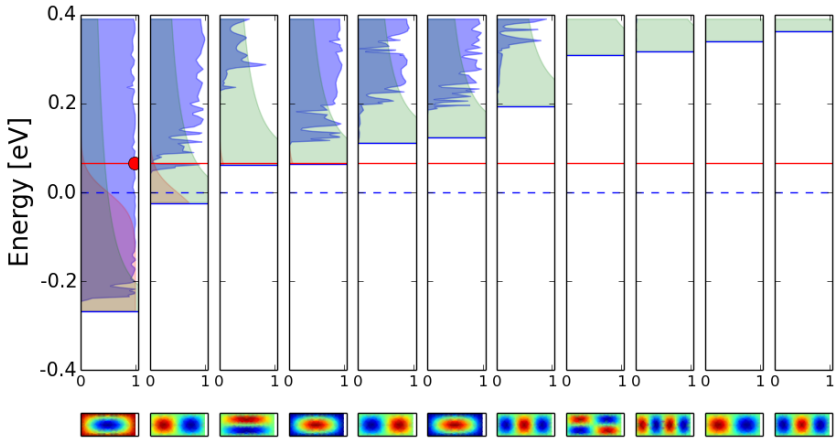


Figure 4.13: Transmission spectra for the injected subbands of each valley in a nanostrip with height 2 nm and width 5 nm. The solid blue line in each transmission spectrum shows the subband energy. The green area shows the 1D-DOS and the red area shows the Fermi-Dirac distribution. The dotted blue line is the Fermi-level in the source which is set to 0 eV. The donor doping concentration is $N_D = 10^{19}$ cm^{-3}.

Figure 4.14: Si nanowire of width $w = 5$ nm, height $h = 5$ nm and 10 periodic indents of depth $i = 1$ nm. The length of one indent is $il = 2$ nm and the unindented length is $ul = 2$ nm.

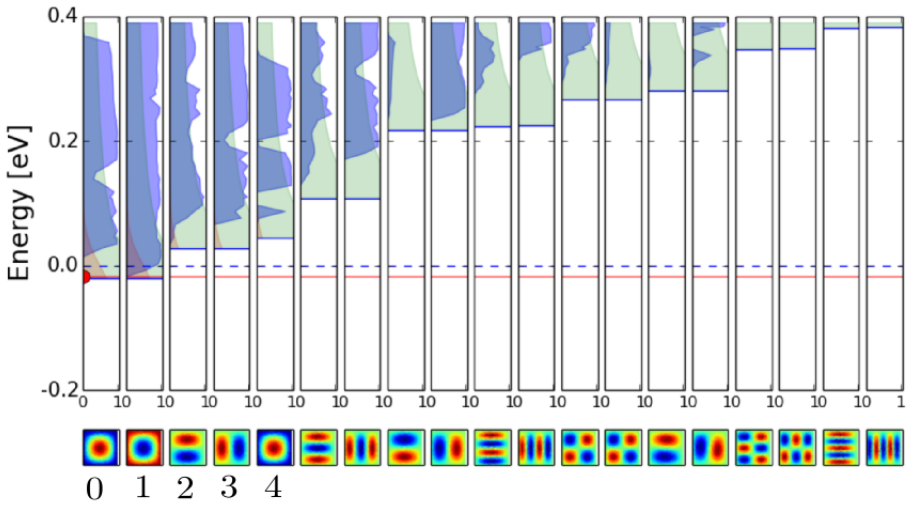


Figure 4.15: Transmission spectra for each injected subband in a Si nanowire with periodic indents of 1 nm shown in figure 4.14. Only the subbands which can contribute considerably to the current are numbered. The green area shows the 1D-DOS and the red area shows the Fermi-Dirac distribution. The dotted line is the Fermi-level in the source which is set to 0 eV. The donor doping concentration is $N_D = 10^{19}$ cm^{-3}.

4.3.1 Si nanowire with 1 nm periodic indents

Figure 4.14 shows a Si nanowire with 10 periodic indents of 1 nm depth. The nanowire cross-sectional dimensions are 5 nm by 5 nm. The length of one indent is 2 nm and the unindented length is 2 nm. Figure 4.15 shows the transmission spectra for each injected subband from low to high subband energy. The first 5 subbands which can contribute considerably to the current are numbered from 0 to 4. The transmission spectra in figure 4.15 show clear dips in the transmission over ranges of energy, which is a sign for the onset of minibandgaps. However a full minibandgap should have zero transmission over a range of energies. In the next paragraph we deepen the indents to 2 nm to enhance the periodicity and possibly the formation of the miniband structure.

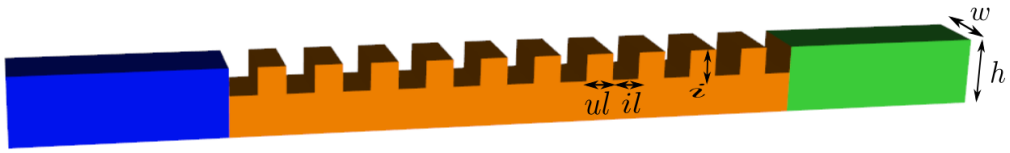


Figure 4.16: Si nanowire of width $w = 5$ nm, height $h = 2$ nm and indents of depth $i = 2$ nm. The length of one indent is $il = 2$ nm and the unindented length is $ul = 2$ nm.

4.3.2 Si nanowire with 2 nm periodic indents

Figure 4.16 shows a Si nanowire with 10 periodic indents of 2 nm depth. Only the indents have increased in depth compared to 4.3.1. All the other periodic parameters have remained the same. Figure 4.17 shows the transmission spectrum for each injected subband in the Si nanowire with periodic indents of 2 nm. From figure 4.17 we note that deeper indents of 2 nm enhance the minibandgaps to zero transmission compared to 4.3.1. At an energy in the the minibandgap of the injected subband 0 (red line with red dot in figure 4.17) there is no transmission of the wavefunction through the nanowire as shown by figure 4.18. Figures 4.17 and 4.18 proof that it is indeed possible to obtain a miniband structure and as a consequence, energy filtering with a 3D geometric superlattice.

Influence of the anisotropy of the effective mass

In figure 4.17 the first 5 subbands which can contribute considerably to the current are numbered from 0 to 4. For these first 5 subbands, we note that minibands and minibandgaps appear for the 100-valley subband (subband 0 and 3) and the 001-valley subband (subband 4). However, no minibandgaps appear for the 010-valley (subbands 1 and 2). The miniband structure (at energies lower than 0.4 eV is thus obtained for the valleys with a low effective mass along the indentation direction. We state that a low effective mass in the direction of the indent is beneficial for the formation of a miniband structure at low energies. This statement is qualitatively explained in figure 4.19 and supported with simulations in figures 4.20 and 4.21.

Figure 4.19 explains why minibands appear for injected subband 3 of the 100-valley and not for injected subband 2 of the 010-valley in figure 4.17. Figure 4.19 shows the injection of these two subbands in the Si nanowire with top indentations. Qualitatively we can understand that the indentations will have more influence on the 100-valley injected subband than on the 010-valley injected subband, because the indentations lie perpendicular to the wavefunction blobs for the 100-valley, while they lie in the direction of the wavefunction blobs for the 010-valley. This results in a miniband structure for the 100-valley and no miniband structure for the 010-valley in figure 4.17. Hence, the superlattice with indentations on top is a better energy

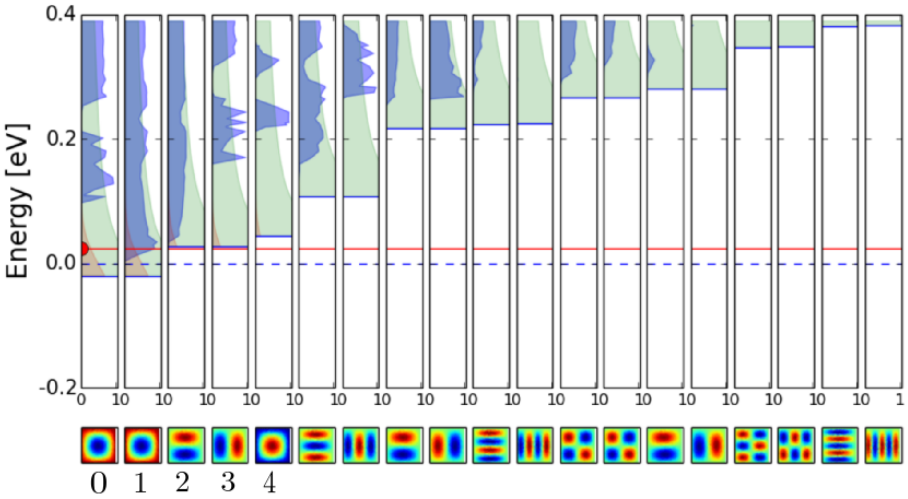


Figure 4.17: Transmission spectra for each injected subband in a Si nanowire with periodic indents of 2 nm shown in figure 4.16. Only the subbands which can contribute considerably to the current are numbered. The green area shows the 1D-DOS and the red area shows the Fermi-Dirac distribution. The dotted line is the Fermi-level in the source which is set to 0 eV. The donor doping concentration is $N_D = 10^{19}$ cm^{-3}

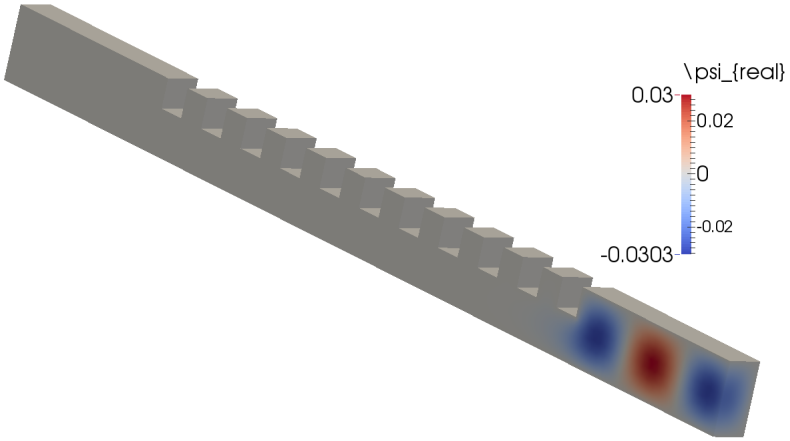


Figure 4.18: Wavefunction inside the Si nanowire with indents of 2 nm depth for injected subband 0 from figure 4.17. The energy corresponds to an energy in the minibandgap, as shown by the red line in figure 4.17. At this energy there is no transmission toward the end.

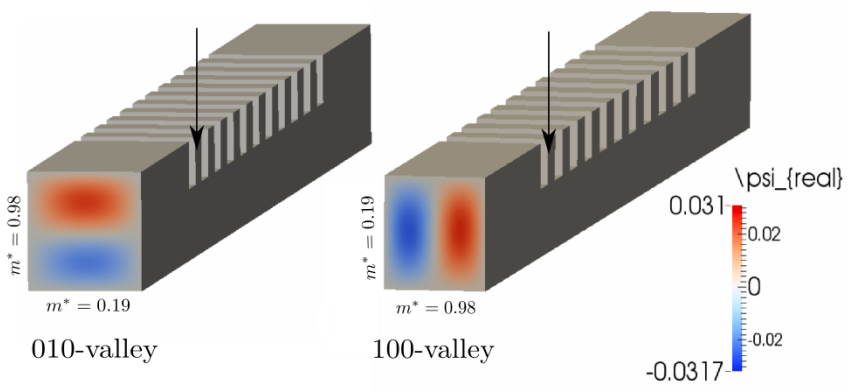


Figure 4.19: For this geometry, the periodic feature is placed along the direction with low effective mass for the 100-valley and along the high effective mass for the 010-valley. A top indentation is a better energy filtering feature for electrons in the 100-valley than for electrons in the 010-valley.

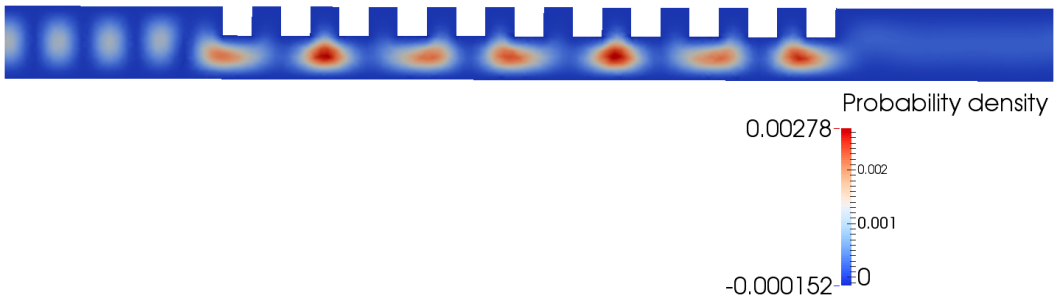


Figure 4.20: Probability density of injected subband 3 in figure 4.17. Injected subband 3 is a subband from the 100-valley with low effective mass along the indent direction. The probability density in the non-indented parts is non-zero. Qualitatively, the probability density blobs try to get into the non-indented parts of the geometric superlattice. As a consequence, the geometric superlattice has an influence over the probability density, which leads to the formation of a miniband structure for the 100-valley.

filter for the electrons in the 100-valley than for the electrons in the 010-valley. The energy filter can be tuned to obtain energy filtering for the 010-valley by making indentations in the direction where the 010-valley has a low effective mass, i.e. use a lateral indentation. Figures 4.20 and 4.21 show the probability density for injected subband 3 and 2 in figure 4.17 respectively.

We have shown the possibility of energy filtering in a 3D geometric superlattice. In the next sections we vary the periodic parameters further to understand how the minibands and minibandgaps are influenced by changing the periodicity of the superlattice. In the transmission spectra 4.17 for 2 nm indentations we note that the

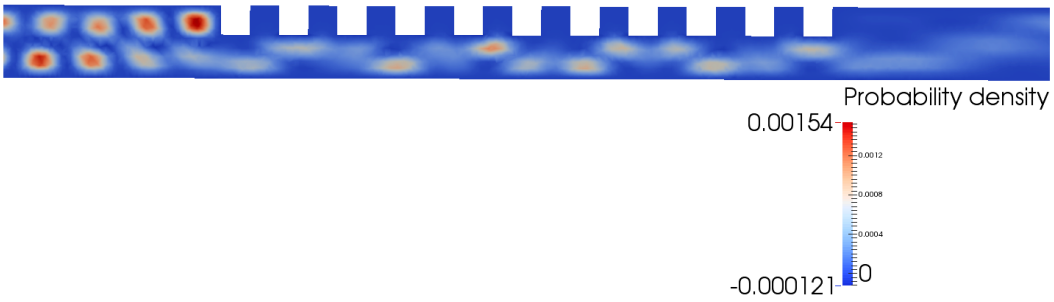


Figure 4.21: Probability density of injected subband 2 in figure 4.17. Injected subband 2 is a subband from the 010-valley with high effective mass along the indent direction. The probability density in the non-indented parts is zero, in contrast with the probability density of the 100-valley in figure 4.20. The probability density blobs are not influenced by the geometric superlattice, which results in the absence of a miniband structure for the 010-valley.

transmission probability for the subbands contributing to the current is lowered by a qualitative factor of $\frac{1}{2}$ compared to 1 nm indentations in 4.3.1. With 2 nm indentations, the indentations take away almost half the 5 nm nanowire height. Therefore, making fins instead of indents should result in more transmission probability and is considered in the next section.

4.3.3 Si nanowire with 2 nm periodic fins

Figure 4.22 shows a Si nanowire with 10 periodic fins of 2 nm height. All the other periodic parameters have remained the same. Figure 4.23 shows the transmission spectra for each injected subband in the case of 2 nm fins. We note that again the minibandgaps appear more clearly for the injected subbands of the 100-valley and less for the injected subbands of the 010-valley. A low effective mass in the direction of the periodic feature (in this case a fin) enhances the formation of a miniband structure at low energies. Figure 4.24 shows the transmission of the wavefunction inside the Si nanowire with fins for injected subband 0 at a miniband energy.

By comparing the transmission spectra for 2 nm fins in 4.23 with the transmission spectra for 2 nm indents in 4.17, the transmission probability in the first miniband is bigger for a fin periodic structure. A higher transmission probability in the first miniband will increase the on-state current and boost performance. However, fins of 2 nm have a degraded miniband structure compared to 2 nm indents which results in worse energy filtering capacity with a fin periodic structure. In the next paragraph we consider an additional fin on the right side of the nanowire which should result in minibandgaps for the 010-valley as well as for the 100-valley. We also increase the fin height in an attempt to enhance the periodic feature and deepen the miniband gaps. This is considered in the next paragraph.

Figure 4.22: Si nanowire of 5 nm by 5 nm with fins of 2 nm height. The length of one fin is 2 nm and the spacing between the fins is 2 nm.

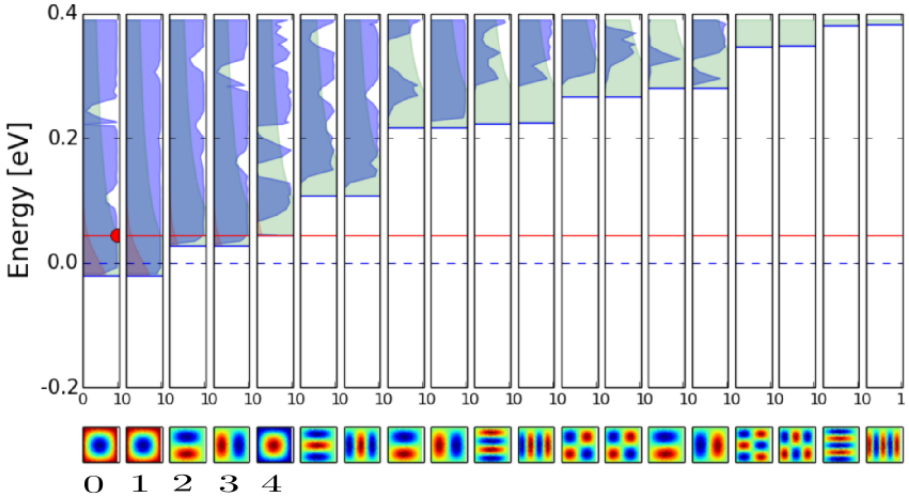


Figure 4.23: Transmission spectra for each injected subband in a Si nanowire with periodic fins of height 2 nm shown in figure 4.22. Only the subbands which can contribute considerably to the current are numbered. The green area shows the 1D-DOS and the red area shows the Fermi-Dirac distribution. The dotted line is the Fermi-level in the source which is set to 0 eV. The donor doping concentration is $N_D = 10^{19}$ cm^{-3}.

4.3.4 Si nanowire with fins on top and side

Figure 4.25 shows a Si nanowire with 10 periodic fins on top and side of the nanowire. Figure 4.26 shows the transmission spectra for fins with height of 2 nm. By comparing figure 4.26 with the transmission spectra for 2 nm fins only on top in 4.23, we note that the onsets of minibandgaps now also appear for the first subband of the 010-valley, although the onsets of the minibandgaps are not very big for the 100-valley and 010-valley. For the 001-valley (injected subband 4) however, with low effective mass on both cross-sectional directions, a pronounced minibandgap is obtained with fins on top and side in figure 4.26 and not with fins only on top in figure 4.23. Figure 4.27 shows that the minibandgaps cannot be enhanced by increasing the height of the fins to 4 nm and that the pronounced minibandgap for injected subband 4 is

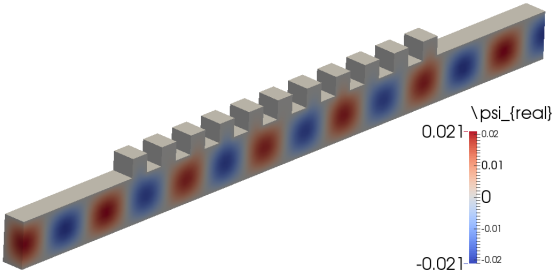


Figure 4.24: Wavefunction inside the Si nanowire with fins of 2 nm height for injected subband 0 from figure 4.23. The energy corresponds to an energy in a miniband, as shown by the red line in figure 4.23. At this energy there is transmission toward the end.

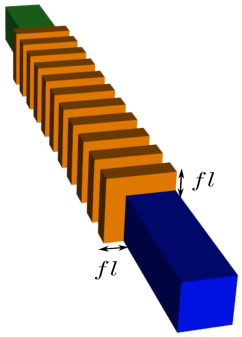


Figure 4.25: Si nanowire with fins of length $fl = 2$ nm on top and side.

gone.

Influence of varying the spacing between the periods

Until now, the indents and fins had a fixed spacing between the periodic feature and a given length of the periodic feature, i.e. 2 nm for both. Figures 4.28 and 4.29 show that the required spacing between the features depends on the wavelength of the wavefunctions in the geometric superlattice. At low energy in figure 4.28, the wavevector in the transport direction is small (see equation 3.18) and the oscillations in the probability density are too long to be influenced by the narrowly spaced fins. At high energy in 4.29, the wavevector in the transport direction is big and the oscillations in the probability density are short enough to be influenced by the fins. Ideally, in figure 4.28 the fins are longer such that also at low energies the fins influence the probability density and a miniband structure can be obtained at

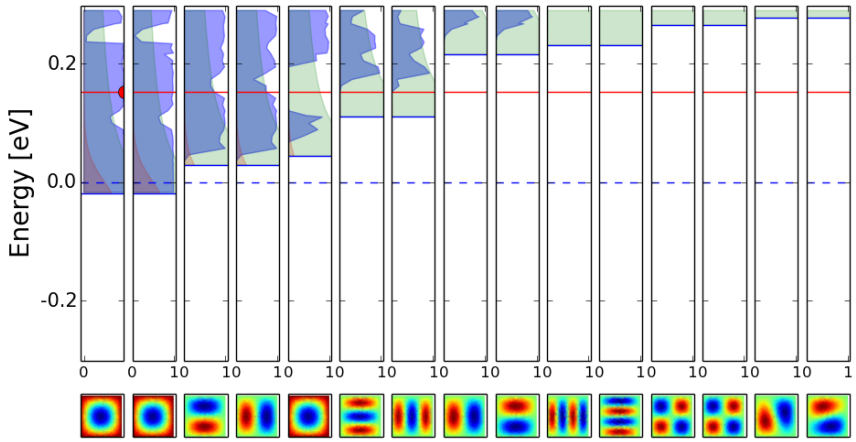


Figure 4.26: Transmission spectra for the injected subbands in a Si nanowire with fins on top and side of 2 nm height. The donor doping concentration is $N_D = 10^{19}$ cm^{-3}.

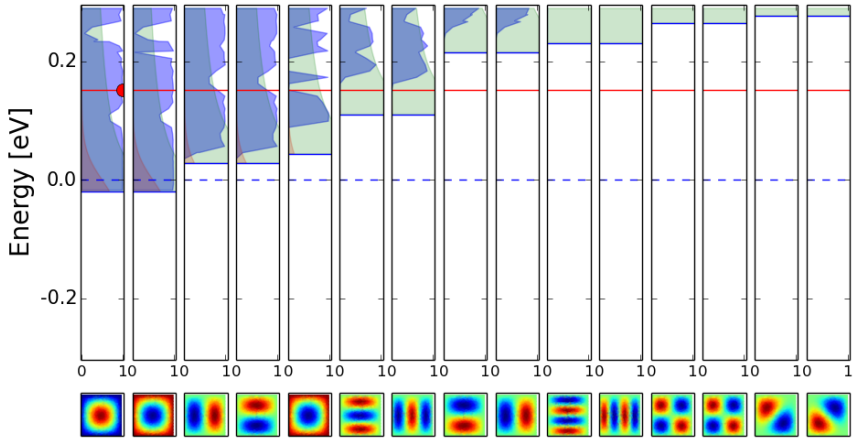


Figure 4.27: Transmission spectra for the injected subbands in a Si nanowire with fins on top and side of 4 nm height. The donor doping concentration is $N_D = 10^{19}$ cm^{-3}.

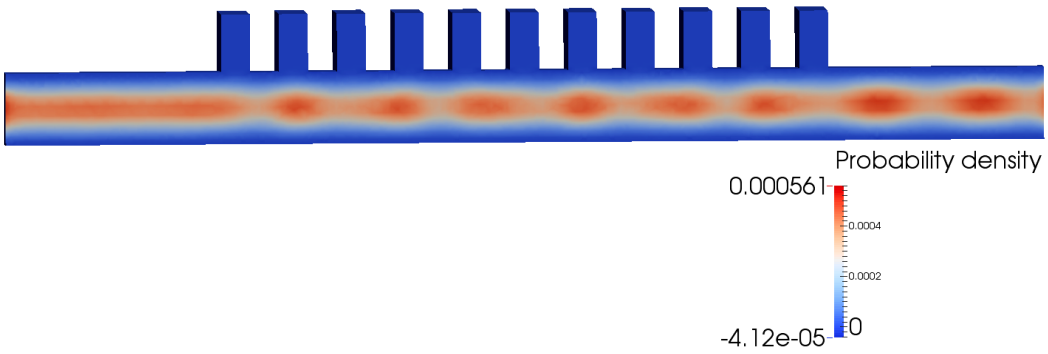


Figure 4.28: Slice of the probability density for injected subband 0 at lower energy.

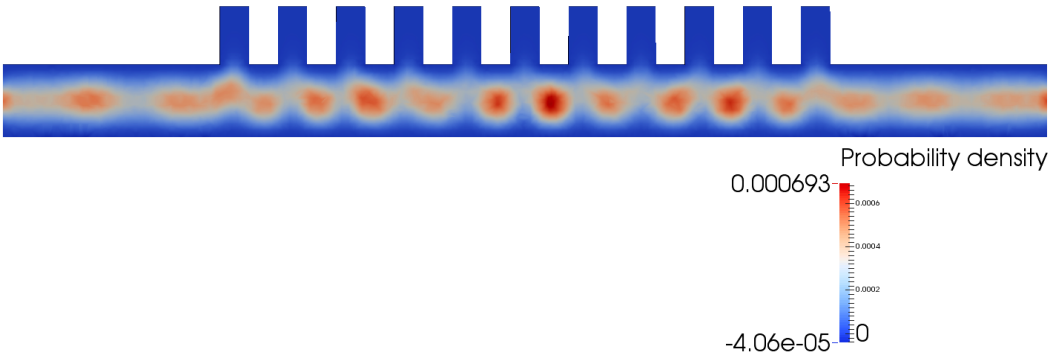


Figure 4.29: Slice of the probability density for injected subband 0 at higher energy.

low energy. In addition, from equation 3.18 it follows that a low effective mass in the transport direction results in a smaller wavevector and bigger oscillations of the probability density. A smaller effective mass in the transport direction results in the possibility to obtain interference with longer periodic features. Longer periodic features are advantageous from a fabrication point of view. To achieve an ideal transmission spectrum starting from 4.26, there are two problems to face. First, injected subband 0 and 1 in figure 4.26 do not fully reach zero transmission and do not extend over a big energy range. Second, subbands 2, 3 and 4 also contribute to the current and have transmission at the energies where injected subbands 0 and 1 have a minibandgap. Injected subbands 2, 3 and 4 thus deteriorate the energy filtering of subband 0 and 1. In the next section we propose a geometric superlattice with a close to ideal energy filtering transmission spectrum.

4.4 Ideal energy filter

We characterized the transmission spectrum for an ideal energy filter with three necessary characteristics. First, the ideal transmission spectrum has a decent first miniband to achieve a sufficient on-state current for minimum performance. The width of the first miniband is the energy range which can be blocked by the gate

potential energy barrier and corresponds to the supply voltage (around 0.1 V-0.4 V for nanowire transistors[21]). Second, the ideal transmission spectrum has a first minibandgap which is as big as possible to block the high energy electrons and achieve low passive power consumption. Third, the ideal transmission spectrum has a sharp band edge from first minibandgap to first miniband to achieve a steep SS.

4.4.1 Si nanostrip with side indentations

In figure 4.31 we show a transmission spectrum for the first injected subband which comes very close to the ideal transmission spectrum and is not deteriorated by the transmission spectra of higher subbands. The transmission spectra in figure 4.31 belong to a Si nanostrip with side indentations as shown in figure 4.30. In the next paragraph we explain why the Si nanostrip with side indentations comes close to an ideal energy filter.

The first important characteristic of the Si nanostrip with side indentations is the strong confinement in the height down to 2 nm. As explained and simulated in 4.13, a strong confinement along the height allows to isolate the 010-valley. In figure 4.31 we therefore only plotted the 010-valley. The 100-valley and 001-valley will lie at sufficiently high energies due to their low effective mass in the height direction. The potential transmission in the 100-valley and 001-valley can therefore not deteriorate the miniband structure of injected subband 0 of the 010-valley. The second important characteristic of the Si nanostrip is its side indentations along the direction where the effective mass of the electrons in the 010 valley is low. As explained and simulated in 4.19, a low effective mass along the direction of the indent enhances the formation of minibands and minibandgaps at low energies. A third important characteristic is the low effective mass in the transport direction due to the isolated 010-valley. A smaller effective mass in the transport direction results in the possibility to obtain interference with longer periodic features. Longer periodic features are advantageous from a fabrication point of view. The configuration of the Si nanostrip for which we achieved the close to ideal transmission spectrum in figure 4.31 has an indent length of 5 nm and an unindented length of 3 nm. Also note in figure 4.31 that we increased the doping concentration from $N_D = 10^{19}$ cm^{-3} to $N_D = 10^{20}$ cm^{-3} compared to the previous cases to place the Fermi-level inside the first miniband of the first injected subband of the 010-valley.

4.5 Conclusion

In this chapter we fulfilled the third research objective. We proved the existence of energy filtering in 3D geometric superlattices by simulating nanowires with 10 periodic indentations and showing the minibands and minibandgaps in the transmission spectra of the injected subbands. We showed the tunability of the energy filtering by deepening the indentations and varying the geometric superlattice to fins. Indentations allowed to obtain more pronounced minibandgaps for the lowest injected subbands, while fins only showed decent minibandgaps for the 001-valley.

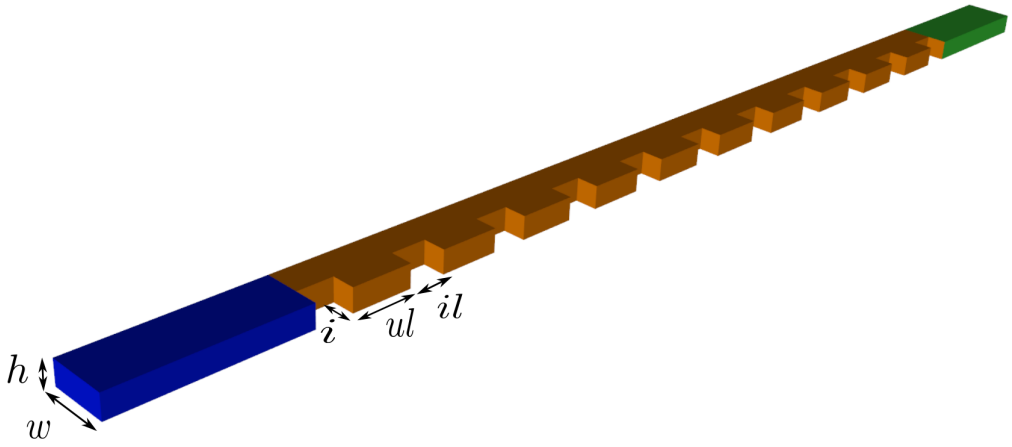


Figure 4.30: Si nanostrip with side indentations. The measurements of the periodic parameters corresponding to the transmission spectra in figure 4.31 are 10 periods, width $w = 5$ nm, height $h = 2$ nm, indent depth $i = 2$ nm, indent length $il = 3$ nm and unindented length $ul = 5$ nm.

We found that the anisotropy of the effective mass in Si plays an important role in relation with the periodic features of the geometric superlattice and the presence of minibands and minibandgaps at low energies. We explained why the Si nanostrip with side indentations is a close to ideal energy filter. In the next section we propose the Si nanostrip geometric superlatticeFET and relax the fabrication of its geometric superlattice by lowering the number of periods to 5 without decreasing the energy filtering capacity.

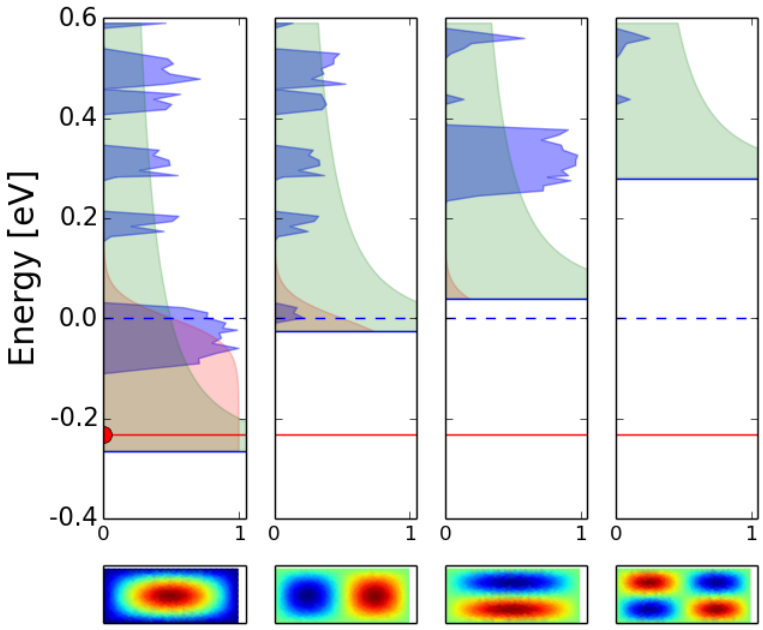


Figure 4.31: Transmission spectra for the injected subbands of the isolated 010-valley in a Si nanostrip with side indentations and parameters as shown in figure 4.30. Only the first injected subband of the 010-valley contributes significantly to the current. The green area shows the 1D-DOS and the red area shows the Fermi-Dirac distribution. The dotted line is the Fermi-level in the source which is set to 0 eV. The doping concentration in the Si nanostrip is $N_D = 10^{20}$ cm^{-3}.

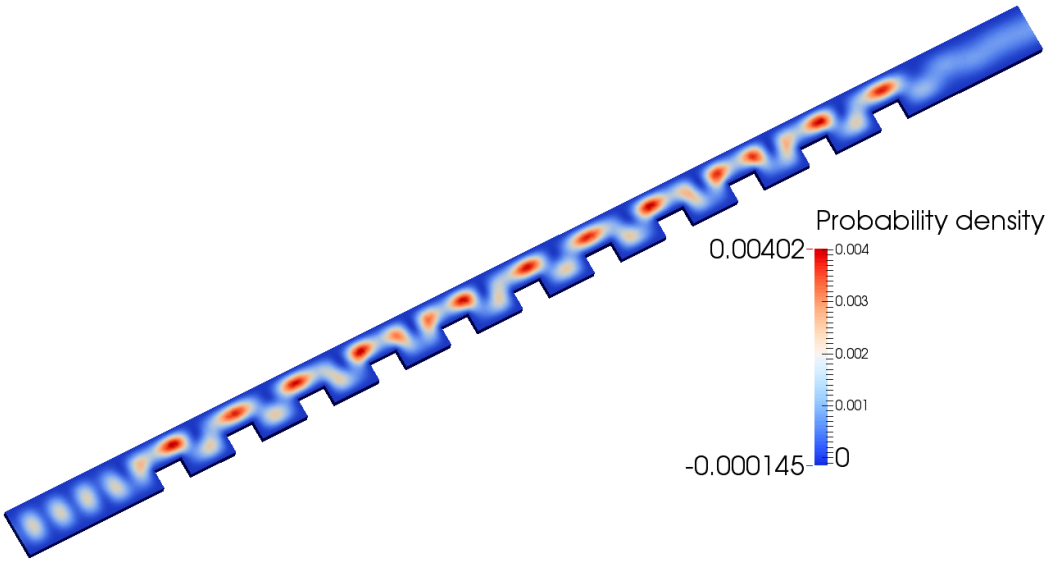


Figure 4.32: Probability density through the Si nanostrip with side indentations for the first injected subband of the 010-valley at an energy in the first miniband. In the miniband, a high probability density is present under the geometric superlattice

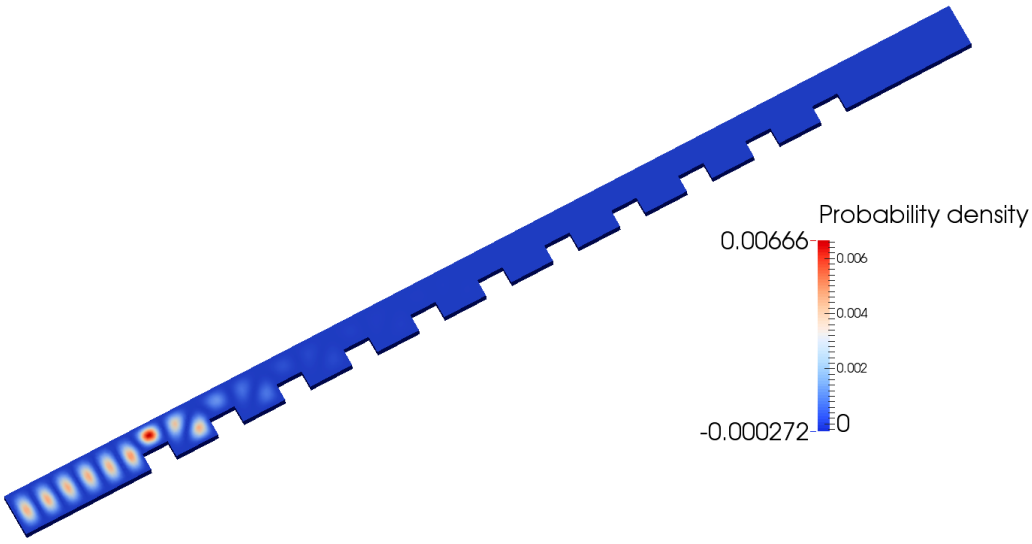


Figure 4.33: Probability density through the Si nanostrip with side indentations for the first injected subband of the 010-valley at an energy in the first minibandgap. In the minibandgap a high probability density is blocked by the geometric superlattice due to constructive interference at the source side.

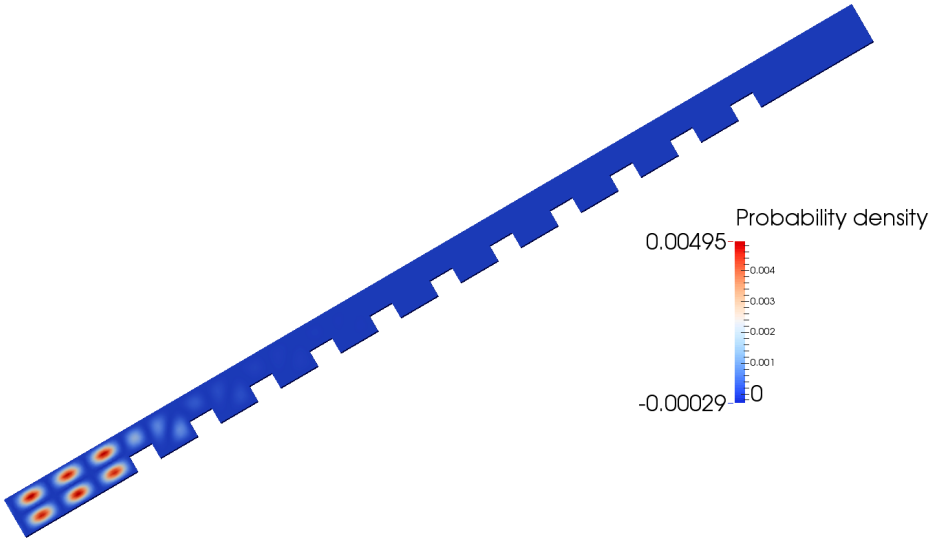


Figure 4.34: Probability density through the Si nanostrip with side indentations for the second injected subband of the 010-valley at an energy in a minibandgap.

Chapter 5

Phase III: Investigating the Si nanostrip geometric superlatticeFET

"Can we lower the fabricational difficulty of the Si nanostrip geometric superlatticeFET? Does it impact the energy filtering?"

"How is the geometric superlatticeFET characterized in terms of leakage current, switching slope and on-state current?"

In this chapter we investigate the characteristics of the Si nanostrip with side indentations as a superlatticeFET. The Si nanostrip with side indentations came out as the geometric superlattice with an energy filtering capacity closest to ideal of the simulated structures in chapter 4. In section 5.1 we lower the number of periods in the geometric superlattice of the nanostrip and find out if the energy filtering capacity becomes deteriorated by the decreased number of periods. A lower amount of periods in the geometric superlattice simplifies potential fabrication of the Si nanostrip geometric superlatticeFET and can lower the cost of fabrication. A lower number of periods is also beneficial in terms of the variability of the periodic nanometer sized features in the geometric superlattice. A mismatch in the geometry of one periodic feature may possibly result in a deteriorated miniband structure and shift the energy filtering capacity away from close to ideal. In section 5.2 we simulate the turn-on characteristics of the Si nanostrip geometric superlatticeFET. From the turn-on characteristics we derive the leakage current, the passive power consumption, the switching slope and the on-state current.

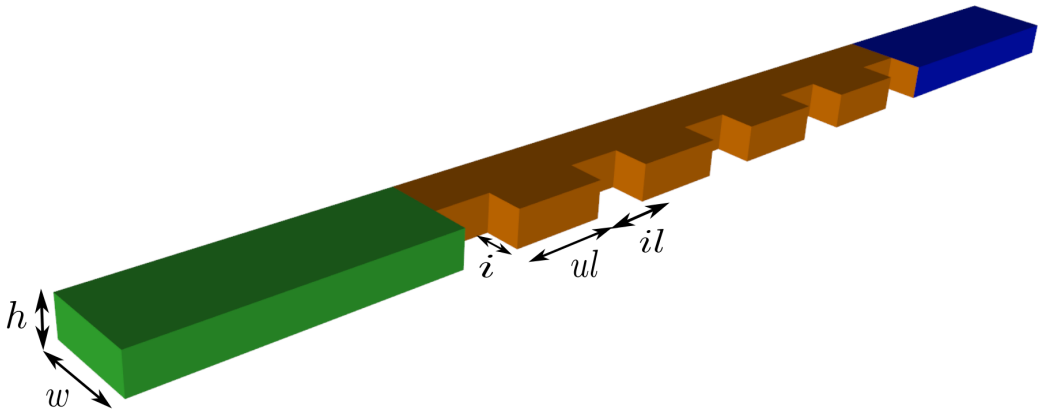


Figure 5.1: Si nanostrip with 5 side indentations. The measurements of the periodic parameters are width $w = 5$ nm, height $h = 2$ nm, indent depth $i = 2$ nm, indent length $il = 3$ nm and unindented length $ul = 5$ nm.

5.1 Lowering the number of periods in the geometric superlattice

In figure 5.1 we lowered the number of periods of the Si nanostrip with side indentations to 5. The transmission spectrum remains close to ideal because, first, the decent first miniband is still present for the lowest injected subband, guaranteeing a minimum on-state current and performance, second, the first minibandgap extends far enough to block the exponential Fermi-Dirac tail, resulting in a low leakage current, and third, the band edge between the first miniband and first minibandgap is still sharp. Figures 5.3 and 5.4 show the probability densities in the Si nanostrip for the first injected subband of the 010-valley at an energy in the first miniband and minibandgap respectively. Figure 5.5 shows the probability density in the Si nanostrip for the second injected subband of the 010-valley at an energy in a minibandgap.

5.2 Turn-on characteristics

In this section we simulate the turn-on characteristics of the Si nanostrip geometric superlatticeFET with 5 side indentations. Based on the turn-on characteristics it is possible to derive magnitudes for the inverse subthreshold slope, the leakage current, the passive power consumption and the on-state current. For a new transistor concept, circuit designers are interested in these metrics to predict the performance and power consumption of their circuits.

Figure 5.6 shows the Si nanostrip geometric superlatticeFET with 5 side indentations. The geometric superlattice is placed between the source and the gate. A drain-source potential V_{DS} of 0.1 V is applied at the drain. The gate potential is applied fully around the gate and is ramped from 0.0 V in the normally-on state

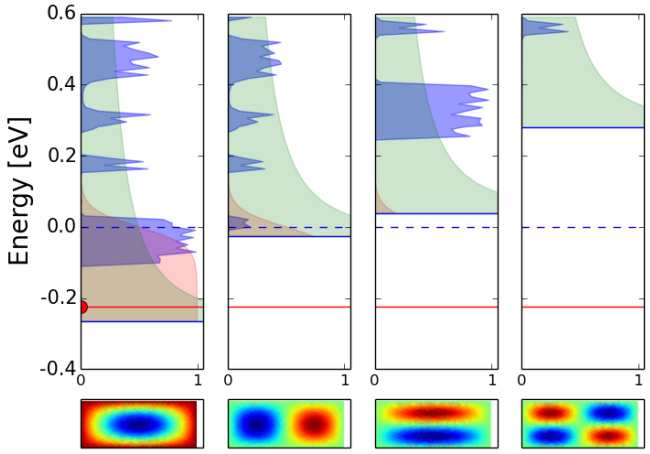


Figure 5.2: Transmission spectra for the injected subbands of the isolated 010-valley in the Si nanostrip with 5 side indentations shown in figure 5.1. The green area shows the 1D-DOS and the red area shows the Fermi-Dirac distribution. The dotted line is the Fermi-level in the source which is set to 0 eV. The donor doping concentration is $N_D = 10^{20}$ cm^{-3}.

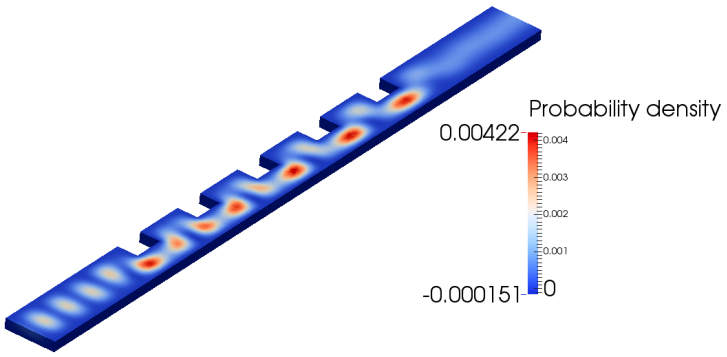


Figure 5.3: Probability density through the Si nanostrip with side indentations for the first injected subband of the 010-valley at an energy in the first miniband. In the miniband, a high probability density is present under the geometric superlattice.

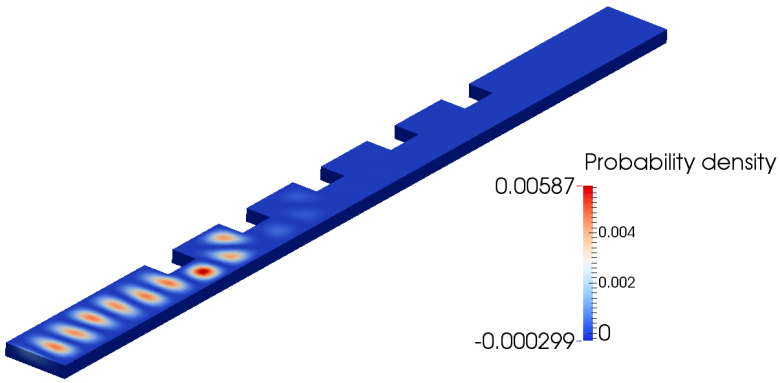


Figure 5.4: Probability density through the Si nanostrip with side indentations for the first injected subband of the 010-valley at an energy in the first minibandgap. In the minibandgap a high probability density is blocked by the geometric superlattice due to constructive interference at the source side.

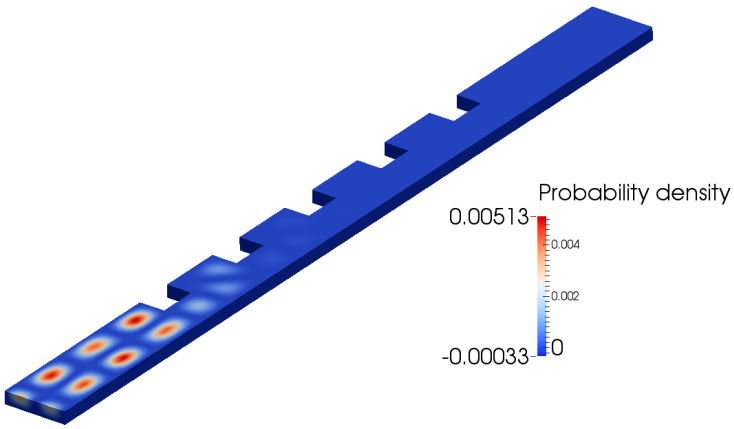


Figure 5.5: Probability density through the Si nanostrip with side indentations for the second injected subband of the 010-valley at an energy in a minibandgap.

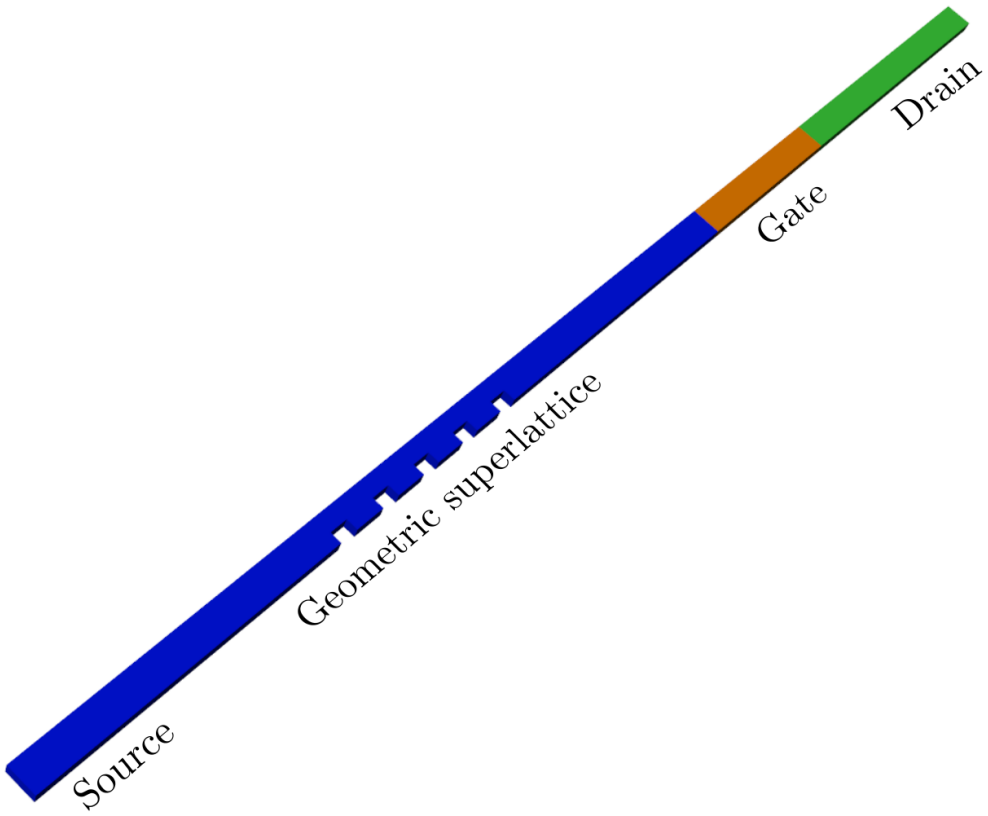


Figure 5.6: Si nanostrip geometric superlatticeFET with 5 side indentations. The blue region corresponds to the source, the orange region to the gate and the green region to the drain. The geometric superlattice consists of 5 side indentations and is placed between the source and the gate. The gate potential is applied fully around the gate.

to -0.5 V in the off-state. Figure 5.7 shows the obtained turn-on characteristics which plots the drain-source current $logI_{DS}$ in function of the applied gate-source potential V_{GS}. The turn-on curve has a sub-60 mV/dec inverse subthreshold slope, which validates the Si nanostrip geometric superlatticeFET as a steep slope transistor concept. A steeper slope than the fixed ideal 60 mV/dec slope of planar MOSFETs allows to lower the power consumption while keeping the required performance. In the on-state (at 0.0 V in figure 5.7), we obtain an on-state current in the order of 10^{-5} A. In the off-state (at -0.5 V in figure 5.7), we achieve a leakage current in the pA-regime. A leakage current in the order of 10^{-12} A at a drain-source voltage of 0.1 V allows for a passive power consumption of 10^{-13} W in the Si nanostrip geometric superlatticeFET. Figures 5.8 and 5.9 show the transmission spectra of the first injected subbands of the 010-valley in the on-state and the off-state respectively.

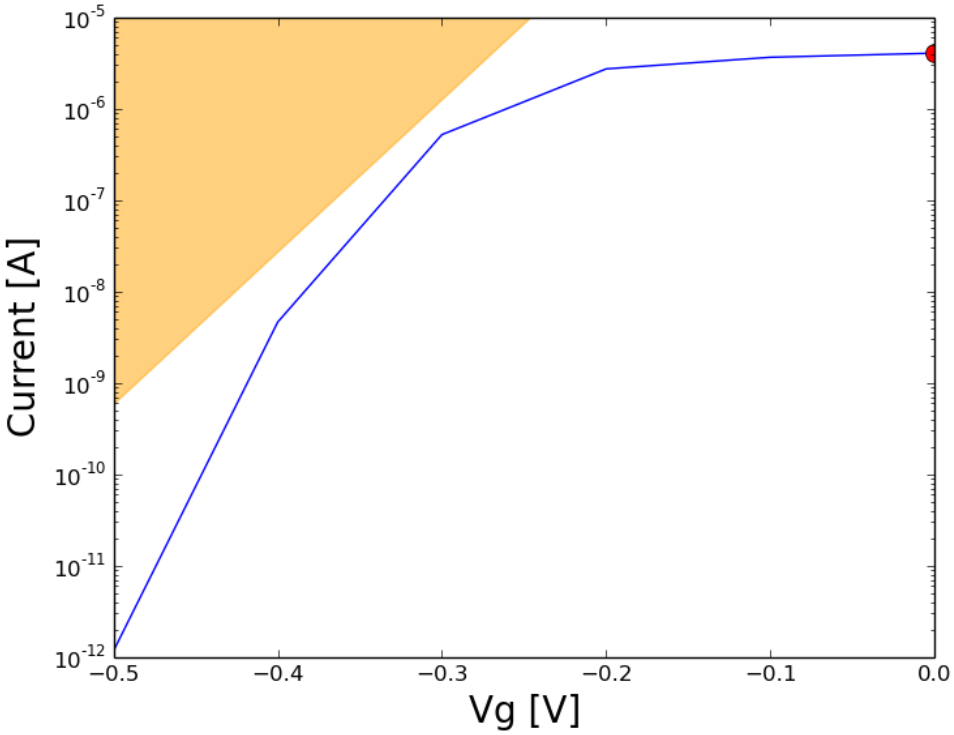


Figure 5.7: Turn-on characteristics $logI_{DS}$ in function of V_{GS}. The turn-on characteristics shows a sub-60 mV/dec inverse subthreshold slope, validating the Si nanostrip geometric superlatticeFET as a steep slope transistor concept. The yellow slope denotes 60 mV/dec. The on-state current is in the order of 10^{-5} A and the leakage current is in the pA-regime.

5.3 Conclusion

In this chapter we proposed the Si nanostrip geometric superlatticeFET with 5 periods and showed that the energy filtering capacity of the Si nanostrip geometric superlatticeFET is kept with a periodicity of 5 periods. A lower number of periods simplifies the potential fabrication of the Si nanostrip and can lower the fabricational cost. In addition, a lower number of periods is beneficial in terms of the variability of the periodic nanometer sized features in the geometric superlattice. We simulated the turn-on characteristics of the Si nanostrip geometric superlatticeFET. We achieved an inverse subthreshold slope of sub-60 mV/dec, which is below the fixed ideal slope of planar MOSFETs. The sub-60 behaviour validates the Si nanostrip geometric superlatticeFET as a steep slope transistor concept. The on-state current is in the order of 10^{-5} A and the leakage current in the pA-regime.

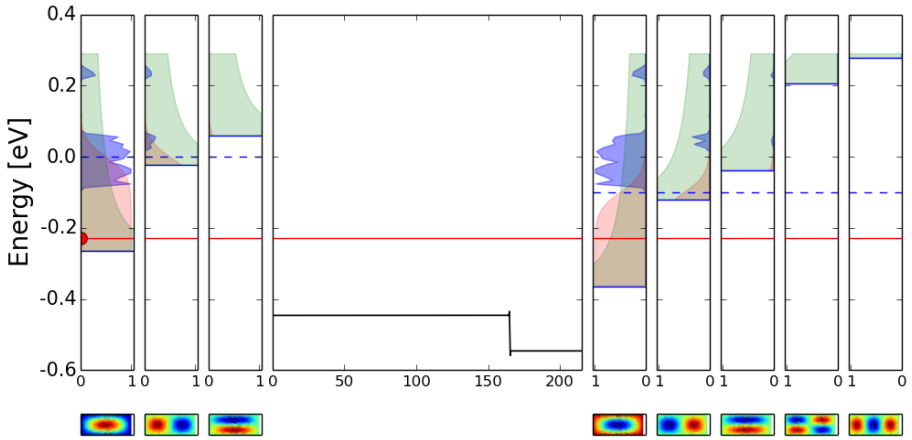


Figure 5.8: Transmission spectra of the first injected subbands of the 010-valley in the on-state. The gate potential barrier is flat at 0.0 V. A drain-source voltage V_{DS} is applied of 0.1 V.

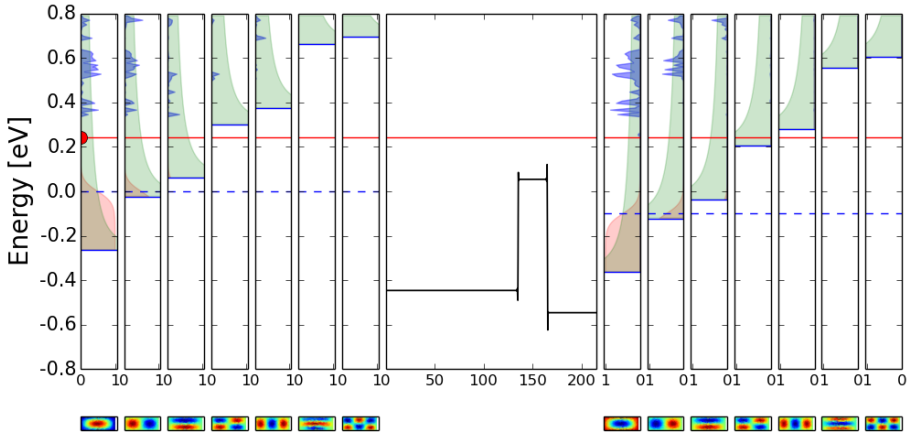


Figure 5.9: Transmission spectra of the first injected subbands of the 010-valley in the off-state. The gate potential barrier is raised by applying a potential of -0.5 V. A drain-source voltage V_{DS} is applied of 0.1 V.

Chapter 6

General conclusion and directions for future research

In the last chapter we present the main findings of the thesis research and propose directions for future research.

6.1 General conclusion

In this thesis we proposed a solution on transistor level for the energy leakage in chips. Reducing the energy leakage is especially important for chips targeted for low power applications such as future IoT devices. Steep slope transistors can reduce the power consumption of the chip, while keeping the necessary performance. From the literature study we found that the available steep slope transistor concepts are limited and each have their own major disadvantage. The research done on superlatticeFETs is mainly limited to nanowires with a material pair-superlattice. In this thesis we investigated Si nanowire superlatticeFETs with a geometric superlattice. We derived the QTB method in 3D, first with isotropic and then with anisotropic effective mass. The model with anisotropic effective mass can take into account the different valleys in the conduction band of Si. We proved the possibility of energy filtering in 3D geometric superlattices by simulating Si nanowires with 10 periodic indentations and showing the minibands and minibandgaps in the transmission spectra of the injected subbands. We showed the tunability of the energy filtering by deepening the indentations and varying the geometric superlattice to fins. Indentations allowed to obtain more pronounced minibandgaps for the lowest injected subbands, while fins only showed decent minibandgaps for the 001-valley. We found that the anisotropy of the effective mass in Si plays an important role in relation with the periodic features of the geometric superlattice and the presence of minibands and minibandgaps at low energies. We suggested the Si nanostrip with side indentations as a steep slope transistor concept with close to ideal energy filtering characteristics to reduce the energy leakage. The Si nanostrip geometric superlatticeFET achieves energy filtering with a 5 period-geometric superlattice and is made completely from Si. The current semiconductor industry is established for Si and Si should allow for

less defects in the fabrication of the geometric superlattice, which could possibly obstruct the energy filtering capacity of the geometric superlattice. We simulated the turn-on characteristics of the Si nanostrip geometric superlatticeFET. We achieved an inverse subthreshold slope of sub-60 mV/dec, which is below the fixed slope of planar MOSFETs. The sub-60 behaviour validates the Si nanostrip geometric superlatticeFET as a steep slope transistor concept. We achieved an on-state current in the order of 10^{-5} A and a leakage current in the pA-regime. The Si nanostrip geometric superlatticeFET has a smallest lateral feature of 3 nm. Electron beam lithography has the potential, although challenging, of fabricating structures with a resolution down to 2.2 nm[31], which opens up the possibility for a first experimental analysis of the Si nanostrip geometric superlatticeFET to verify the simulation results presented in this thesis.

6.2 Directions for future research

As a guidance for future research, we present the challenges and opportunities which lie ahead for the further investigation of nanowire superlatticeFETs with a geometric superlattice, both theoretically as well as experimentally. To fully understand the device behaviour of nanowire superlatticeFETs with a geometric superlattice, further theoretical research and modelling is necessary, while the experimental investigation of nanowire superlatticeFETs with a geometric superlattice has yet to be started.

6.2.1 Theoretical

In this thesis we modelled the nanowire superlatticeFET with a geometric superlattice using a single band effective mass approximation for electrons in the conduction band and for ballistic transport. The next step is to further investigate the turn-on characteristics with sub-60 mV/dec inverse subthreshold slope, in correspondence with the geometric superlattice of 5 side indentations. The continuous approach is a good starter approach to gain understanding in a first approximation, although in this approach many things are swept under the carpet from the start. Future theoretical investigations of the nanowire superlatticeFET with a geometric superlattice should incorporate the following things into the simulation model:

- A more realistic conduction band structure than the one obtained with the single band effective mass approximation. Non-parabolic band bending can be taken into account with the $\mathbf{k} \cdot \mathbf{p}$ method.

- A more realistic non-equilibrium distribution than the ballistic case which takes into account the carrier scattering.

- Atomistic tight binding simulations as a verification of the electronic structure obtained with the continuous QTB method, especially for the more confined structures simulated in chapter 4 and the Si nanostrip in chapter 5. Although the effective mass theory may describe the electronic states in straight Si

nanowires well down to 1 nm[1], this is not necessarily guaranteed for a geometric superlattice.

- The valence band structure, which was fully neglected because we only considered n-type superlatticeFETs. For p-type superlatticeFETs the valence band structure should be included.

6.2.2 Experimental

When the theoretical device behaviour of the nanowire superlatticeFET with a geometric superlattice is understood, experiments can be initiated to check and calibrate the obtained models and simulations with reality. The Si nanostrip geometric superlatticeFET proposed in chapter 5 has a smallest lateral periodic feature of 3 nm which can potentially be made with electron beam lithography in a test structure for a first experimental analysis. In a next stage, geometric superlattices may be incorporated in vertical nanowire transistors, following the semiconductor roadmap toward vertical architectures.

Appendices

Appendix A

Appendix: Mathematical derivations

A.1 Expression for the b_m^l coefficients

Due to the orthonormality of the eigenfunctions i.e. $\oint_{D_l} \chi_n^l(\xi,\eta)\left[\chi_{m\neq n}^l(\xi,\eta)\right]^* dS = \delta_{mn}$ the following statement is valid:

$$\oint_{D_l} \chi_m^l(\xi,\eta)\psi_l(\xi,\eta,\zeta=0)dS \tag{A.1a}$$

$$= \oint_{D_l} \chi_m^l(\xi,\eta)\sum_{m=0}^{\infty}\left(a_m^l + b_m^l\right)\chi_m^l(\xi,\eta)dS \tag{A.1b}$$

$$= a_m^l + b_m^l \tag{A.1c}$$

This results in the following expression for the b_m^l coefficients:

$$b_m^l = \oint_{D_l} \chi_m^l(\xi,\eta)\psi_0(\xi,\eta,\zeta=0)dS - a_m^l \tag{A.2}$$

where we have used the Dirichlet boundary condition.

A.2 Proof of the hermiticity of the Hamiltonian in the 3D anisotropic effective mass Schrödinger equation

An operator is Hermitian if the following condition is fulfilled:

$$< \widehat{A}\psi|\psi >= \int_\Omega \left(\widehat{A}\psi\right)^* \psi dV = \int_\Omega \psi^* \widehat{A}\psi dV =< \psi|\widehat{A}\psi > \tag{A.3}$$

Inserting the expression for the operator $\widehat{A}$ in the left hand side of equation A.3:

$$< \widehat{A}\psi|\psi > = \int_\Omega \left(-\frac{\hbar^2}{2}\nabla\cdot\frac{1}{M^*}\nabla\psi\right)^* \psi dV \tag{A.4a}$$

$$= -\frac{\hbar^2}{2}\int_\Omega \psi\nabla\cdot\frac{1}{M^*}\nabla\psi^* dV \tag{A.4b}$$

$$= -\frac{\hbar^2}{2}\int_\Omega \psi^*\nabla\cdot\frac{1}{M^*}\nabla\psi dV \tag{A.4c}$$

$$= \int_\Omega \psi^*\left(-\frac{\hbar^2}{2}\nabla\cdot\frac{1}{M^*}\nabla\right)\psi dV \tag{A.4d}$$

$$= \int_\Omega \psi^*\widehat{A}\psi dV \tag{A.4e}$$

$$=< \psi|\widehat{A}\psi > \tag{A.4f}$$

where we have used Green's second identity given by equation A.8 and the fact that the wavefunctions are zero on the Dirichlet boundary and that the following is valid on the Robin boundaries:

$$\frac{\partial\rho}{\partial t} + \nabla\cdot j = 0 \tag{A.5a}$$

$$\Leftrightarrow \frac{\partial}{\partial t}\int_V \psi(\mathbf{r},t)^*\psi(\mathbf{r},t)dV + \int_V \nabla\cdot j dV = 0 \tag{A.5b}$$

$$\Leftrightarrow \oint_S j\cdot\mathbf{n}_{\partial V} dS = 0 \tag{A.5c}$$

$$\Leftrightarrow \oint_S \left(\psi^*\nabla\psi\cdot\mathbf{n}_{\partial V} - \psi\nabla\psi^*\cdot\mathbf{n}_{\partial V}\right) dS = 0 \tag{A.5d}$$

$$\Leftrightarrow \oint_S \left(\psi\nabla\psi^*\cdot\mathbf{n}_{\partial V} - \psi^*\nabla\psi\cdot\mathbf{n}_{\partial V}\right) dS = 0 \tag{A.5e}$$

where we started with the continuity equation, considering that for the time independent Schrödinger equation the wavefunctions are not dependent on t. Then we used the divergence theorem and the definition of the probability current. Hence, the conservation of probability current follows from the the hermiticity of the Hamiltonian.

The hermiticity cannot be obtained for the operator $\widehat{A} = -\frac{\hbar^2}{2M^*}\nabla\cdot\nabla$ where the effective mass tensor is brought in front of the divergence:

$$< \widehat{A}\psi|\psi >= \int_\Omega \left(-\frac{\hbar^2}{2M^*}\nabla\cdot\nabla\psi\right)^* \psi dV = -\frac{\hbar^2}{2}\int_\Omega \psi\frac{1}{M^*}\nabla\cdot\nabla\psi^* dV \tag{A.6}$$

where neither Green's second identity nor Green's first identity can be used to prove the hermiticity of the operator $\widehat{A} = -\frac{\hbar^2}{2M^*}\nabla\cdot\nabla$.

Green's first identity:

$$\int_V f\nabla^2 g dV = -\int_V \nabla f\cdot\nabla g dV + \oint_{\partial V} f\left(\nabla g\cdot\boldsymbol{n}_{\partial V}\right) dS \tag{A.7}$$

Green's second identity:

$$\int_V f\nabla\cdot(\epsilon\nabla g)dV = \int_V g\nabla\cdot(\epsilon\nabla f)dV + \oint_{\partial V}\epsilon\Big(f\nabla g\cdot\boldsymbol{n}_{\partial V} - g\nabla f\cdot\boldsymbol{n}_{\partial V}\Big)dS \quad \text{(A.8)}$$

A.3 Derivation for 3D and isotropic effective mass

A.3.1 Variational boundary value problem in the device region

In the last step of the derivation of QTBM in the isotropic case, we set up a fully defined and solvable problem for $\psi_0(x,y,z)$ by rewriting the Schrödinger equation in the device region as a variational boundary value problem. This variational boundary value problem can then be solved numerically for $\psi_0(x,y,z)$ using the finite element method.

$$\left(-\frac{\hbar^2}{2m_0^*}\nabla^2 + V_0(x,y,z)\right)\psi_0(x,y,z) = E\psi_0(x,y,z) \quad \text{(A.9)}$$

Equation A.9 gives the Schrödinger equation for $\psi_0(x,y,z)$ in the device region. To rewrite the partial differential equation as a variational boundary value problem, we seek the *weak form* of A.9. To obtain the weak form of A.9, first, multiply the equation with a test function $\overline{\psi}$, second, integrate the resulting equation over the device region Ω_0 and third, apply Green's first identity given by A.7. These three steps modify equation A.9 to:

$$\frac{\hbar^2}{2m_0^*}\int_{\Omega_0}\nabla\overline{\psi}\cdot\nabla\psi_0 dV + \int_{\Omega_0}\overline{\psi}\,(V_0-E)\,\psi_0 dV = \frac{\hbar^2}{2m_0^*}\oint_{\partial\Omega_0}\overline{\psi}\,(\nabla\psi_0\cdot\mathbf{n}_{\partial\Omega_0})\,dS \quad \text{(A.10)}$$

The test function $\overline{\psi}$ is zero on the Dirichlet boundary $(\partial\Omega_0 - \sum_l D_l)$. As a consequence, the integral on the right hand side of equation A.10 is only non-zero on the Robin boundaries $\sum_{D_l}$:

$$\frac{\hbar^2}{2m_0^*}\int_{\Omega_0}\nabla\overline{\psi}\cdot\nabla\psi_0 dV + \int_{\Omega_0}\overline{\psi}\,(V_0-E)\,\psi_0 dV = \frac{\hbar^2}{2m_0^*}\sum_l\oint_{D_l}\overline{\psi}\,(\nabla\psi_0\cdot\mathbf{n}_{D_l})\,dS \quad \text{(A.11)}$$

In equation A.11 we have obtained the weak form of the Schrödinger equation in the device region. In the weak form, the QTBCs on the Robin boundaries appear as *natural* boundary conditions. Together with the *essential* boundary condition on the Dirichlet boundary, we have a fully defined and solvable variational boundary value problem for $\psi_0(x,y,z)$. The resulting variational boundary value problem in $\psi_0(x,y,z)$ is stated as follows:

Solve the weak form:

$$\frac{\hbar^2}{2m_0^*}\int_{\Omega_0}\nabla\overline{\psi}\cdot\nabla\psi_0 dV + \int_{\Omega_0}\overline{\psi}\,(V_0-E)\,\psi_0 dV = \frac{\hbar^2}{2m_0^*}\sum_l\oint_{D_l}\overline{\psi}\,(\nabla\psi_0\cdot\mathbf{n}_{D_l})\,dS \quad \text{(A.12)}$$

by applying the natural boundary conditions on $\sum_l D_l$:

$$\nabla\psi_0\cdot\mathbf{n}_{D_l}|_{D_l} = \sum_{m=0}^{\infty} ik_\zeta^{l,m}\chi_m^l(\xi,\eta)\left(-2a_m^l + \iint_{D_l}\chi_m^l(\xi,\eta)\psi_0(\xi,\eta,\zeta=0)dS\right) \quad \text{(A.13)}$$

and the essential boundary condition on $(\partial\Omega_0 - \sum_l D_l)$:

$$\psi_0|_{(\partial\Omega_0 - \sum_l D_l)} = 0 \tag{A.14}$$

A.3.2 Transmission coefficient $T(E)$

The transmission coefficient over the device region from one lead to the other leads is given by the outgoing current over the incoming current in the respective leads:

$$T(E) = \frac{I_{out}}{I_{in}} \tag{A.15}$$

The current comes in through one lead and goes out through all the other leads. We can write the currents I_{in} and I_{out} in terms of probability currents j_{in} and j_{out} in the leads:

$$T(E) = \frac{\sum_l \oint_{D_l} j^l_{out,\zeta} dS}{\oint_{D_{l'}} j^{l'}_{in,\zeta} dS} \tag{A.16}$$

where $j_{out,\zeta}$ is the component of the outgoing probability current perpendicular to D_l in lead l and $j^{l'}_{in,\zeta}$ is the component of the incoming probability current perpendicular to $D_{l'}$ with l' the injection lead. The *probability current* $\mathbf{j}$ in the leads for isotropic effective mass is defined as:

$$\mathbf{j} = \frac{\hbar}{2m^* i}\left(\psi_l^* \nabla\psi_l - \psi_l \nabla\psi_l^*\right) \tag{A.17}$$

The solutions for the wavefunction ψ_l in the leads found in equation 3.19 give the incoming and outgoing wavefunctions:

$$\begin{cases} \psi^{l'}_{in}(\xi,\eta,\zeta) = \sum_{m=0}^{\infty} a^{l'}_{m'} e^{-ik_\zeta^{l',m'}\zeta} \chi^l_m(\xi,\eta) \\ \psi^{l}_{out}(\xi,\eta,\zeta) = \sum_{m=0}^{\infty} b^{l}_{m} e^{ik_\zeta^{l,m}\zeta} \chi^l_m(\xi,\eta) \end{cases}$$

Using these wavefunctions in the expression for the probability current A.17, we find:

$$j_{in,\zeta} = \frac{\hbar}{2m_1^*}\left(\sum_{m=0}^{M_1} (a^1_m)^* (\chi^1_m(\xi,\eta))^* e^{-ik^1_{\zeta\zeta}} \sum_{m=0}^{M_1} a^1_m \chi^1_m(\xi,\eta) k^1_\zeta e^{ik^1_{\zeta\zeta}} \right. \tag{A.18a}$$

$$\left. + \sum_{m=0}^{M_1} a^1_m \chi^1_m(\xi,\eta) e^{ik^1_{\zeta\zeta}} \sum_{m=0}^{M_1} (a^1_m)^* (\chi^1_m(\xi,\eta))^* k^1_\zeta e^{-ik^1_{\zeta\zeta}} \right) \tag{A.18b}$$

$$j_{out,\zeta} = \frac{\hbar}{2m_2^*}\left(\sum_{m=0}^{M_2} (b^2_m)^* (\chi^2_m(\xi,\eta))^* e^{-ik^2_{\zeta\zeta}} \sum_{m=0}^{M_2} b^2_m \chi^2_m(\xi,\eta) k^2_\zeta e^{ik^2_{\zeta\zeta}} \right. \tag{A.19a}$$

$$\left. + \sum_{m=0}^{M_2} b^2_m \chi^2_m(\xi,\eta) e^{ik^2_{\zeta\zeta}} \sum_{m=0}^{M_2} (b^2_m)^* (\chi^2_m(\xi,\eta))^* k^2_\zeta e^{-ik^2_{\zeta\zeta}} \right) \tag{A.19b}$$

where we have considered one incoming and one outgoing lead. We can then calculate the transmission coefficient $T(E)$ given in equation A.15. Using the orthonormality of the subbands $\oint_{D_l} \chi_m^l(\xi,\eta) \left[\chi_n^l(\xi,\eta)\right]^* dS = \delta_{mn}$:

$$T(E) = \frac{m_1^*}{m_2^*} \frac{\sum_{m=0}^{M_2} |b_m^2|^2 k_\zeta^2}{\sum_{m=0}^{M_1} |a_m^1|^2 k_\zeta^1} \tag{A.20}$$

Together with equation A.2 for the b_m^l-coefficients, we obtained a formula from which we can calculate the transmission coefficient $T(E)$ in the nanowire and plot transmission spectra in the case of isotropic effective mass.

A.4 Proof: no need to split $v = v_{real} + iv_{imag}$

We start from a general *weak formulation* in x:

$$\int_0^1 u'(x)v'(x)dx = \int_0^1 f(x)v(x)dx$$

where $v(x)$ is a general *test function* from the *test space* V and $f(x)$ is a complex function $f_1(x) + if_2(x)$. Inserting $f(x)$ and $u(x) = u_{real} + iu_{imag}$ gives:

$$\int_0^1 (u'_{real}(x) + iu'_{imag}(x))v'(x)dx = \int_0^1 (f_1(x) + if_2(x))v(x)dx$$

This equation can be split in a real and complex part:

$$\int_0^1 u'_{real}(x)v'_1(x)dx = \int_0^1 f_1(x)v_1(x)dx \tag{A.21a}$$

$$\int_0^1 u'_{imag}(x)v'_2(x)dx = \int_0^1 f_2(x)v_2(x)dx \tag{A.21b}$$

Equation A.21 denotes two weak formulations with both an unknown function $u_{real}(x)$ or $u_{imag}(x)$. In finite element modelling, the unknown functions are assumed to be built of *basis functions* $\phi_n(x)$ from a certain subset of the test space V:

$$u_{real}(x) = \xi_1\phi_1(x) + ... + \xi_N\phi_N(x) \tag{A.22a}$$

$$u_{imag}(x) = \xi_{N+1}\phi_{N+1}(x) + ... + \xi_{2N}\phi_{2N}(x) \tag{A.22b}$$

Inserting $u_{real}(x)$ and $u_{imag}(x)$ into equation A.21, results in:

$$\begin{cases} \int_0^1 (\xi_1\phi'_1(x) + ... + \xi_N\phi'_N(x))v'_1(x)dx & = \int_0^1 f_1(x)v_1(x)dx \\ \int_0^1 (\xi_{N+1}\phi'_{N+1}(x) + ... + \xi_{2N}\phi'_{2N}(x))(x)v'_2(x)dx & = \int_0^1 f_2(x)v_2(x)dx \end{cases}$$

By evaluating the previous two expressions for every basis function ϕ, a full system of equations can be obtained from which the coëfficiënts ξ_n can be computed:

$$\begin{cases} \int_0^1 (\xi_1\phi_1'(x) + ... + \xi_N\phi_N'(x))\phi_1'(x)dx & = \int_0^1 f_1(x)\phi_1(x)dx \\ \qquad\qquad ... & \\ \int_0^1 (\xi_1\phi_1'(x) + ... + \xi_N\phi_N'(x))\phi_N'(x)dx & = \int_0^1 f_1(x)\phi_N(x)dx \\ \int_0^1 (\xi_{N+1}\phi_{N+1}'(x) + ... + \xi_{2N}\phi_{2N}'(x))(x)\phi_{N+1}'(x)dx & = \int_0^1 f_2(x)\phi_{N+1}(x)dx \\ \qquad\qquad ... & \\ \int_0^1 (\xi_{N+1}\phi_{N+1}'(x) + ... + \xi_{2N}\phi_{2N}'(x))(x)\phi_{2N}'(x)dx & = \int_0^1 f_2(x)\phi_{2N}(x)dx \end{cases}$$

Inserting the found coëfficiënts ξ_n in equation A.22 results in the functions $u_{real}(x)$ and $u_{imag}(x)$.

Suppose we would have used a complex test function $v(x) = v_{real}(x) + iv_{imag}(x)$ in equation A.21. Then we would have for the real part (equation A.21a):

$$\int_0^1 u_{real}'(x)(v_{real}'(x) + iv_{imag}'(x))dx = \int_0^1 f_1(x)(v_{real}(x) + iv_{imag}(x))dx \quad \text{(A.23a)}$$

which would again result in a real and a complex part:

$$\int_0^1 u_{real}'(x)v_{real}'(x)dx = \int_0^1 f_1(x)v_{real}(x)dx \quad \text{(A.24a)}$$

$$\int_0^1 u_{real}'(x)v_{imag}'(x)dx = \int_0^1 f_1(x)v_{imag}(x)dx \quad \text{(A.24b)}$$

However, equation A.24 denotes two weak formulations for the same unknown function $u_{real}(x)$, only solved with a different subset of testfunctions than the v_1 from before. This will result in linearly dependent rows in the system of equations. Thus no new information can be obtained by taking the test function as a complex function.

Appendix B

Appendix B: Python code

B.1 Calculation of the states in 3D and with effective mass tensor

```
import dolfin as df
import logging
import numpy as np
import tables as tb
from auxiliary.materials import Oxide
from auxiliary.settings import unitsim
from dolfin import dx, inner, nabla_grad
mat = lambda m: df.as_backend_type(m).mat()
import matplotlib.pyplot as plt
import scipy

prms = unitsim.parameters
logger = logging.getLogger('unit_simulation_logger')
df.parameters['allow_extrapolation'] = True
df.parameters['form_compiler']['optimize'] = True
df.parameters['form_compiler']['cpp_optimize'] = True


class States(object):
    def __init__(self, device):
        self.device = device
        self.schroedinger = SchroedingerPDE(self.device)
        self.iteration_nr = 0

        # define the structure of the h5-file where these states will be saved
        solution_size = self.schroedinger.V.dim()
        solution_type = df.Function(self.schroedinger.V).vector().array().dtype.name
        nr_of_contacts = len(self.device.descr.lead_boundrs)

        class State(tb.IsDescription):
            number = tb.UInt16Col()
```

```
32              coeffs_real = tb.Col.from_sctype(solution_type,
    solution_size)
33              coeffs_imag = tb.Col.from_sctype(solution_type,
    solution_size)
34              transmission = tb.Float64Col((nr_of_contacts, 2))
35              degeneracy = tb.UInt8Col()
36              inj_lead = tb.UInt8Col()
37              inj_subband = tb.UInt8Col()
38              inj_subband_energy = tb.Float64Col()
39              rel_subband = tb.UInt8Col()
40              rel_subband_energy = tb.Float64Col()
41              energy = tb.Float64Col()
42              energy_ubnd = tb.Float64Col()
43              energy_lbnd = tb.Float64Col()
44              next_state = tb.UInt16Col()
45              prev_state = tb.UInt16Col()
46              refine_iter = tb.UInt8Col()
47              inj_lead_effective_mass_along_transport = tb.Float16Col()
48              valley = tb.StringCol(16)
49          self.state_class = State
50
51      def update_states(self, potential):
52          """Find an initial amount of states.
53
54          Since nothing can be known about the resonance energies here,
    and all states contribute evenly to the current,
55          a uniform energy grid is defined for which states are found.
56
57          """
58
59          self.iteration_nr += 1
60          self.schroedinger.update_subbands_and_pot_energy(potential)
61          self.subbands = self.schroedinger.leads.subbands
62          h5fileloc = "results/qtbm_states_iter" + str(self.iteration_nr)
     + ".h5"
63          h5file = tb.open_file(h5fileloc, mode='w', title="Information
    of states.")
64          table = h5file.create_table('/', 'states', self.state_class, "
    Table containing a state with all attributes.")
65
66          # define a range of energies for which states will be sought
67          leadpotentials = self.device.environment.potentials
68          all_sb_enrgs = np.array([enrgy for l in self.subbands.keys()
    for v in self.subbands[l].keys() if v != 'number of' for enrgy in
    self.subbands[l][v]['energies']])
69          min_enrgy = np.min(all_sb_enrgs)
70          elstruct_settings = prms['structure']['states']
71          cutoff_energy = elstruct_settings['cutoff energy'] if 'cutoff
    energy' in elstruct_settings else 0.01102494   # 0.3 eV
72          max_enrgy = -np.min([leadpotentials.values()]) + cutoff_energy
73          ediff = elstruct_settings['ediff'] if 'ediff' in
    elstruct_settings else 0.000367498  # 0.01 eV
74          esmall = 8.0e-5  # to avoid division by zero later
75          egrid = np.linspace(min_enrgy + esmall, max_enrgy, (max_enrgy-
    min_enrgy)//ediff)
```

```
76          initial_energies = np.unique(np.concatenate((egrid,
    all_sb_enrgs + esmall), axis=0))
77          self.schroedinger.
    redefine_lead_effective_masses_along_transport()
78
79          state = table.row
80          nr = 0
81          for e in initial_energies:
82              self.schroedinger.update_parameters(energy=e)
83              for l in self.device.descr.lead_boundrs:
84                  max_sb_enrgy = max_enrgy
85                  for v in self.schroedinger.valleys:
86                      self.schroedinger.
    redefine_effective_mass_tensor_terms(v)
87                      for sb in range(self.subbands[l][v]['number of']):
88                          self.schroedinger.update_parameters(inj_lead=l,
     rel_subband=sb)
89                          if self.subbands[l][v]['energies'][sb] < e <
    max_sb_enrgy:
90                              self.schroedinger.redefine_rhs(v)
91                              psi = self.schroedinger.solve(v)
92                              psi_real, psi_imag = psi.split(deepcopy=
    True)
93                              transm = self.schroedinger.
    find_transmissions(psi, l, sb, v)
94                              if nr % 100 == 0: # or v == "001_valley" or
     sb == 2:
95                                  print "State nr.: ", nr, "Lead: ", l, "
       Subband: ", sb, "   Subband energy: ", self.subbands[l][v]['
    energies'][sb], "   Energy: ", e, "     Valley: ", v
96                                  print "Transmission of this state is: "
     + str(transm)
97                              state['number'] = nr
98                              state['coeffs_real'] = psi_real.vector().
    array()
99                              state['coeffs_imag'] = psi_imag.vector().
    array()
100                             state['inj_lead'] = l
101                             state['rel_subband'] = sb
102                             state['inj_subband_energy'] = self.subbands
    [l][v]['energies'][sb]
103                             state['inj_subband'] = self.subbands[l][v][
    'indices'][sb]  # indices are the numbers of the sorted energies in
    the lead
104                             state['
    inj_lead_effective_mass_along_transport'] = self.schroedinger.
    lead_effective_masses_along_transport[l][v]
105                             state['energy'] = e
106                             state['transmission'] = transm
107                             state['degeneracy'] = 2
108                             state['refine_iter'] = 0
109                             state['valley'] = v
110                             state.append()
111                             nr += 1
112         table.flush()
```

```
113         logger.info("Calculated an initial amount of %i states.", nr)
114         plt.title("Subbands")
115         plt.xlabel("Wavevector $k_z$ [$nm^{-1}$]", fontsize=14)
116         plt.ylabel("Energy [eV]", fontsize=14)
117         plt.ylim(-0.1, prms['structure']['states']['cutoff energy'
    ]*27.211)
118         plt.grid()
119         plt.savefig("subbands.png", dpi=72)
120         logger.info("Saved a subbands figure for the first lead.")
121 
122         h5file.close()
123         iternr = 0
124         maxiter = prms['structure']['states']['max refinements']
125         while iternr < maxiter:
126             iternr += 1
127             self.refine_states(h5fileloc, iternr, leadpotentials)
128         self.set_state_energy_intervals(h5fileloc)
129 
130     def refine_states(self, h5fileloc, iternr, leadpotentials):
131         """Add more states to the .h5 file, particularly those near
    transmission peaks."""
132         h5file = tb.open_file(h5fileloc, mode='r+')
133         table = h5file.root.states
134         state = table.row
135         new_state_nr = table.nrows
136         before = table.nrows
137         for l in self.device.descr.lead_boundrs:
138             for sb in range(self.subbands[l]['number of']):
139                 for l_out in self.device.descr.lead_boundrs:
140                     if l_out != l:
141                         sb_state_nrs = np.array([s['number'] for s in
    table.iterrows() if s['inj_lead'] == l and s['inj_subband'] == sb])
142                         if sb_state_nrs.size > 2:
143                             valley = table.cols.valley[sb_state_nrs[0]]
144                             sb_enrgy = table.cols.inj_subband_energy[
    sb_state_nrs[0]]
145                             rel_sb = table.cols.rel_subband[
    sb_state_nrs[0]]
146                             self.schroedinger.
    redefine_effective_mass_tensor_terms(valley)
147                             self.schroedinger.update_parameters(
    inj_lead=l, inj_subband=sb, rel_subband=rel_sb)
148                             sb_degeneracy = table.cols.degeneracy[
    sb_state_nrs[0]]
149                             dummy_transm = table.cols.transmission[
    sb_state_nrs[0]]
150                             entry = np.argwhere(dummy_transm[:, 0] ==
    l_out)[0][0]
151                             transmissioncfs = table.cols.transmission
    [:][sb_state_nrs, entry, 1]
152                             energies = table.cols.energy[:][
    sb_state_nrs]
153                             s = np.argsort(energies)
154                             sb_state_nrs = sb_state_nrs[s]
155                             transmissioncfs = transmissioncfs[s]
```

```
156                             energies = energies[s]
157                             h1 = np.diff(energies)[:-1].astype(np.
    float128)
158                             h2 = np.diff(energies)[1:].astype(np.
    float128)
159                             ti = transmissioncfs[1:-1].astype(np.
    float128)
160                             tim1 = transmissioncfs[:-2].astype(np.
    float128)
161                             tip1 = transmissioncfs[2:].astype(np.
    float128)
162                             ddt = ((h1 * tip1 + h2 * tim1) / (h1 * h2))
     * (2.0 / (h1 + h2)) - (2 * ti) / (h1 * h2)
163                             refstates = prms['structure']['states']['
    refinement states']
164                             s = np.argsort(np.abs(ddt))[-refstates:] +
    1
165                             new_energies = []
166                             for i in s:
167                                 e = energies[i]
168                                 em1 = energies[i - 1]
169                                 ep1 = energies[i + 1]
170                                 new1 = 2.0 / 3.0 * e + 1.0 / 3.0 * ep1
171                                 new2 = 1.0 / 3.0 * em1 + 2.0 / 3.0 * e
172                                 new_energies.extend([new1, new2])
173                             for e in new_energies:
174                                 self.schroedinger.update_parameters(
    energy=e)
175                                 self.schroedinger.redefine_rhs(valley)
176                                 psi = self.schroedinger.solve(valley)
177                                 psi_real, psi_imag = psi.split(deepcopy
    =True)
178                                 transm = self.schroedinger.
    find_transmissions(psi, l, rel_sb, valley)
179                                 state['number'] = new_state_nr
180                                 state['coeffs_real'] = psi_real.vector
    ().array()
181                                 state['coeffs_imag'] = psi_imag.vector
    ().array()
182                                 state['inj_lead'] = l
183                                 state['inj_subband'] = sb
184                                 state['energy'] = e
185                                 state['inj_subband_energy'] = sb_enrgy
186                                 state['transmission'] = transm
187                                 state['degeneracy'] = sb_degeneracy
188                                 state['refine_iter'] = iternr
189                                 state['valley'] = valley
190                                 state['rel_subband'] = rel_sb
191                                 state.append()
192                                 new_state_nr += 1
193                 table.flush()
194         logger.info("Refined with %i states.", new_state_nr - before)
195         h5file.close()
196
197     def set_state_energy_intervals(self, h5fileloc):
```

```
198         """Set previous/next state within the same subband and energy
    range for all states."""
199         h5file = tb.open_file(h5fileloc, mode='r+')
200         table = h5file.root.states
201         emax = table.cols.energy[:].max() + 0.1
202         for l in self.device.descr.lead_boundrs:
203             for sb in range(self.subbands[l]['number of']):
204                 sb_data = [[state['number'], state['energy']] for state
     in table.iterrows() if state['inj_lead'] == l and state['
    inj_subband'] == sb]
205                 if len(sb_data) == 1:
206                     state_nr = sb_data[0][0]
207                     table.cols.prev_state[state_nr] = state_nr
208                     table.cols.next_state[state_nr] = state_nr
209                     table.cols.energy_lbnd[state_nr] = self.
    schroedinger.leads.sorted_sb_enrgs[l][sb]
210                     table.cols.energy_ubnd[state_nr] = emax
211                 elif len(sb_data) > 1:
212                     sb_state_nrs = np.array(sb_data, dtype=int)[:, 0]
213                     sb_state_enrgs = np.array(sb_data)[:, 1]
214                     s = np.argsort(sb_state_enrgs)
215                     sb_state_nrs = sb_state_nrs[s]
216                     sb_state_enrgs = sb_state_enrgs[s]
217                     for i, state_nr in enumerate(sb_state_nrs):
218                         if i == 0:
219                             table.cols.prev_state[state_nr] = state_nr
220                             table.cols.next_state[state_nr] =
    sb_state_nrs[i + 1]
221                             table.cols.energy_lbnd[state_nr] = self.
    schroedinger.leads.sorted_sb_enrgs[l][sb]
222                             prev_ubnd = (sb_state_enrgs[i] +
    sb_state_enrgs[i + 1])/2.0
223                             table.cols.energy_ubnd[state_nr] =
    prev_ubnd
224                         elif i == s.size - 1:
225                             table.cols.prev_state[state_nr] =
    sb_state_nrs[i - 1]
226                             table.cols.next_state[state_nr] = state_nr
227                             table.cols.energy_lbnd[state_nr] =
    prev_ubnd
228                             table.cols.energy_ubnd[state_nr] = emax
229                         else:
230                             table.cols.prev_state[state_nr] =
    sb_state_nrs[i - 1]
231                             table.cols.next_state[state_nr] =
    sb_state_nrs[i + 1]
232                             table.cols.energy_lbnd[state_nr] =
    prev_ubnd
233                             prev_ubnd = (sb_state_enrgs[i] +
    sb_state_enrgs[i + 1])/2.0
234                             table.cols.energy_ubnd[state_nr] =
    prev_ubnd
235         table.flush()
236         h5file.close()
237
```

```
238
239 class SchroedingerPDE(object):
240     def __init__(self, device):
241         self.device = device
242         self.leads = Leads(self.device)
243         self.mesh = device.fe.mesh
244         degree = prms['structure']['schroedinger']['shapefunctions
    degree']
245         self.V = df.FunctionSpace(device.fe.mesh, 'CG', degree)
246         self.Vcomplex = self.V * self.V
247         self.psi = df.Function(self.Vcomplex)
248         self.lead_boundrs = device.descr.lead_boundrs
249
250         self.valleys = self.leads.valleys
251
252         def check_if_lead(regionr):
253             for l in device.leads:
254                 if regionr == device.leads[l].regionr:
255                     return True
256             return False
257
258         def find_device_regionr():
259             for regnr in device.descr.regions:
260                 if not check_if_lead(regnr):
261                     return regnr
262             logger.info("Something went wrong with finding the device
    region number")
263
264         self.device_regionr = find_device_regionr()
265         self.material = device.descr.regions[self.device_regionr]["
    material"]
266
267         self.bcs = []
268         for bnd in device.descr.gate_boundrs + device.descr.air_boundrs
    :
269             dir_bc_real = df.DirichletBC(self.Vcomplex.sub(0), df.
    Constant(0.0), device.fe.boundaries, bnd)
270             dir_bc_imag = df.DirichletBC(self.Vcomplex.sub(1), df.
    Constant(0.0), device.fe.boundaries, bnd)
271             self.bcs.append(dir_bc_real)
272             self.bcs.append(dir_bc_imag)
273         if not prms['structure']['schroedinger']['wavefunction in oxide
    ']:
274             for regnr, reg in device.descr.regions.items():
275                 if isinstance(reg['material'], Oxide):
276                     bcr = self.device.fe.get_dirichletbc_for_region(
    self.Vcomplex.sub(0), regnr, 0.0)
277                     bci = self.device.fe.get_dirichletbc_for_region(
    self.Vcomplex.sub(1), regnr, 0.0)
278                     self.bcs.append(bcr)
279                     self.bcs.append(bci)
280
281         # weak forms that stay the same regardless
282         self.dx = self.device.fe.dx
283         self.u_real, self.u_imag = df.TrialFunctions(self.Vcomplex)
```

```
284         self.v_real, self.v_imag = df.TestFunctions(self.Vcomplex)
285 
286         ae = df.Constant(-2.0) * (self.u_real * self.v_real + self.
    u_imag * self.v_imag) * dx  # multiply with e
287         self.Ae = mat(df.assemble(ae))
288 
289     def update_subbands_and_pot_energy(self, potential):
290         """Update whenever the potential (i.e. also the potential
    energy) or the valley changes; this also affects the subbands."""
291         self.leads.update_subbands(potential)
292         dx = self.device.fe.dx
293         ds = self.device.fe.ds
294         u_real, u_imag = df.TrialFunctions(self.Vcomplex)
295         v_real, v_imag = df.TestFunctions(self.Vcomplex)
296         mat = lambda m: df.as_backend_type(m).mat()
297         vec = lambda v: df.as_backend_type(v).vec()
298 
299         # update subband terms
300         self.subbands = self.leads.subbands
301         if prms['structure']['states']['solver'] != "Krylov":
302             self.solver = df.LUSolver()
303             R = df.FunctionSpace(self.device.fe.mesh, 'R', 0)
304             r = df.TrialFunction(R)
305             s = df.TestFunction(R)
306             self.qtbc_terms = {}
307             self.righthandsides = {}
308             for l in self.lead_boundrs:
309                 self.qtbc_terms[l] = {}
310                 self.righthandsides[l] = {}
311                 for v in self.valleys:
312                     if v != 'number of':
313                         interplt_sb_arrays = self.subbands[l][v]['
    interpolated arrays']
314                         sb = df.Function(self.V)
315                         subbandterms = []
316                         rhsterms = []
317                         for interplt_sb_array in interplt_sb_arrays:
318                             sb.vector()[:] = interplt_sb_array
319                             ur = df.assemble(s * sb * u_real * ds(l))
320                             ui = df.assemble(s * sb * u_imag * ds(l))
321                             vr = df.assemble(r * sb * v_real * ds(l))
322                             vi = df.assemble(r * sb * v_imag * ds(l))
323                             kr = mat(vr).matMult(mat(ui)) - mat(vi).
    matMult(mat(ur))
324                             ki = mat(vr).matMult(mat(ur)) + mat(vi).
    matMult(mat(ui))
325                             subbandterms.append((kr, ki))
326                             rhs = df.assemble(sb * v_imag * ds(l))
327                             rhsterms.append(vec(rhs))
328                         self.qtbc_terms[l][v] = subbandterms
329                         self.righthandsides[l][v] = rhsterms
330 
331         # initialise a template for A (the LHS) and its sparsity
    pattern
```

```
332             nonzero_pattern = 0  # `0` stands for `
    DIFFERENT_NONZERO_PATTERN`
333             # self.A = df.PETScMatrix().mat()
334             self.A = self.Ae.copy()
335             # self.A.axpy(1.0, self.Ae, nonzero_pattern)
336             for l in self.lead_boundrs:
337                 for v in self.valleys:
338                     self.A.axpy(1.0, self.qtbc_terms[l][v][0][0],
    nonzero_pattern)
339                     self.A.axpy(1.0, self.qtbc_terms[l][v][0][1],
    nonzero_pattern)
340 
341         # update potential energy term
342         U = self.device.get_property('conduction_band') - potential
343         au = df.Constant(2.0) * U * (u_real * v_real + u_imag * v_imag)
     * dx
344         self.Au = mat(df.assemble(au))
345 
346     def update_parameters(self, **kwargs):
347         energy_updated = rel_subband_updated = inj_lead_updated = False
348         for key, value in kwargs.items():
349             setattr(self, key, value)
350             if key == 'energy':
351                 energy_updated = True
352             if key == 'inj_lead':
353                 inj_lead_updated = True
354             if key == 'inj_subband':
355                 inj_subband_updated = True
356             if key == 'rel_subband':
357                 rel_subband_updated = True
358             if key == 'valley':
359                 valley_updated = True
360         if rel_subband_updated or inj_lead_updated:
361             pass
362         if energy_updated:
363             self.ks = {}
364             for i, l in self.device.leads.iteritems():
365                 self.ks[i] = {}
366                 for v in self.valleys:
367                     eigvals = self.subbands[i][v]['energies']
368                     ks = [complex((2.0 * self.
    lead_effective_masses_along_transport[i][v] * (self.energy - eigval)
    ), 0)**0.5 for eigval in eigvals]
369                     self.ks[i][v] = np.array([(k.real, k.imag) for k in
     ks])
370 
371     def redefine_rhs(self, valley):
372         kr, ki = self.ks[self.inj_lead][valley][self.rel_subband]
373         if ki == 0.0:
374             rhs_vec = self.righthandsides[self.inj_lead][valley][self.
    rel_subband].copy()
375             rhs_vec.scale(2.0 * kr * (1.0 / self.
    lead_effective_masses_along_transport[self.inj_lead][valley]))
376             self.rhs = rhs_vec
377         else:
```

```
378             raise NameError("Something is wrong here.")
379
380     def redefine_effective_mass_tensor_terms(self, valley):
381         self.valley = self.material.valleys[valley]
382         self.inverse_effective_mass_tensor = self.valley.
    get_inverse_effective_mass_tensor_in_global_coordinate_system(
383             self.device_regionr, self.mesh, self.device)
384
385         ac = inner(self.inverse_effective_mass_tensor*nabla_grad(self.
    u_real), nabla_grad(self.v_real)) * dx \
386              + inner(self.inverse_effective_mass_tensor*nabla_grad(self
    .u_imag), nabla_grad(self.v_imag)) * self.dx
387         self.Ac = mat(df.assemble(ac))
388
389     def redefine_lead_effective_masses_along_transport(self):
390         self.lead_effective_masses_along_transport = {}
391         for l in self.device.leads.itervalues():
392             self.lead_effective_masses_along_transport[l.leadnr] = {v:
    l.material.valleys[v].
    get_lead_effective_mass_along_transport_in_global_coordinate_system(
    l.mesh) for v in self.valleys}
393
394     def solve(self, valley):
395         A = self.A.copy()
396         A.zeroEntries()
397         nonzero_pattern = 1  # `1` stands for `SUBSET_NONZERO_PATTERN`
398         A.axpy(1.0, self.Ac, nonzero_pattern)
399         A.axpy(self.energy, self.Ae, nonzero_pattern)
400         A.axpy(1.0, self.Au, nonzero_pattern)
401
402         for l, terms in self.qtbc_terms.iteritems():
403             for sbnr, term in enumerate(terms[valley]):
404                 kr = (1.0 / self.lead_effective_masses_along_transport[
    l][valley])*self.ks[l][valley][sbnr][0]
405                 ki = (1.0 / self.lead_effective_masses_along_transport[
    l][valley])*self.ks[l][valley][sbnr][1]
406                 kr_mat = term[0]
407                 ki_mat = term[1]
408                 A.axpy(-kr, kr_mat, nonzero_pattern)
409                 A.axpy(-ki, ki_mat, nonzero_pattern)
410
411                 if l == 1 and not self.ks[l][valley][sbnr][0] < 0.001:
412                     if valley == '100_valley':
413                         plt.plot(self.ks[l][valley][sbnr
    ][0]*(1/18.89716), self.energy*27.211, 'go-')
414                         plt.plot(-self.ks[l][valley][sbnr
    ][0]*(1/18.89716), self.energy*27.211, 'go-')
415                     elif valley == '010_valley':
416                         plt.plot(self.ks[l][valley][sbnr
    ][0]*(1/18.89716), self.energy*27.211, 'ro-')
417                         plt.plot(-self.ks[l][valley][sbnr
    ][0]*(1/18.89716), self.energy*27.211, 'ro-')
418                     elif valley == '001_valley':
419                         plt.plot(self.ks[l][valley][sbnr
    ][0]*(1/18.89716), self.energy*27.211, 'bo-')
```

```
420                         plt.plot(-self.ks[l][valley][sbnr
    ][0]*(1/18.89716), self.energy*27.211, 'bo-')
421                     plt.hold(True)
422
423                 if l == 1:
424                     if valley == '100_valley':
425                         plt.plot(0, self.subbands[l]["100_valley"][
    'energies'][sbnr]*27.211, 'go-')
426                     elif valley == '010_valley':
427                         plt.plot(0, self.subbands[l]["010_valley"][
    'energies'][sbnr]*27.211, 'ro-')
428                     elif valley == '001_valley':
429                         plt.plot(0, self.subbands[l]["001_valley"][
    'energies'][sbnr]*27.211, 'bo-')
430                 plt.hold(True)
431
432         A = df.PETScMatrix(A)
433         B = df.PETScVector(self.rhs)
434         for bc in self.bcs:
435             bc.apply(A, B)
436         self.solver.set_operator(A)
437         self.solver.solve(self.psi.vector(), B)
438         return self.psi
439
440     def find_transmissions(self, psi, inj_lead, rel_subband, v):
441         """Given a function, return the transmission out of each
    contact."""
442         ds = self.device.fe.ds
443         subband = df.Function(self.V)
444         psi_real, psi_imag = psi.split()
445         transmissions = {}
446         for i, l in enumerate(self.lead_boundrs):
447             if l != inj_lead:
448                 prefactor = self.lead_effective_masses_along_transport[
    inj_lead][v] \
449                     / (self.ks[inj_lead][v][rel_subband][0] * self.
    lead_effective_masses_along_transport[l][v])
450                 tot = 0.0
451                 for j, sb in enumerate(self.subbands[l][v]['
    interpolated arrays']):
452                     subband.vector()[:] = sb
453                     tr = df.assemble(subband * psi_real * ds(l))
454                     ti = df.assemble(subband * psi_imag * ds(l))
455                     t = complex(tr, ti)
456                     kr, ki = self.ks[l][v][j]
457                     if ki == 0.0:
458                         tot += kr * abs(t)**2
459                 transmissions[l] = prefactor * tot  # TODO make sure
    this is real
460         transmissions[inj_lead] = 1 - np.sum(transmissions.values())  #
     reflection
461         transmissions_array = []
462         for l, transm in transmissions.iteritems():
463             transmissions_array.append([l, transm])
```

```
464         return np.array(transmissions_array, dtype=float)  # TODO
    define dtype to get (int, float) instead
465
466 class Leads(object):
467     """Eigenfunctions of all the leads (i.e. boundaries labeled as '
    contact'), used in the QTB Method."""
468     def __init__(self, device):
469         self.device = device
470         self.valleys = prms['structure']['states']['valleys']
471
472     def sort_and_number_subband_energies(self):
473         self.sorted_sb_enrgs = {}
474         self.sorted_sb_eigfcts = {}
475         self.index_array = {}
476         self.index_argsorted = {}
477         for l in self.device.descr.lead_boundrs:
478             sb_enrgs_in_lead = np.array([enrgy for v in self.valleys
    for enrgy in self.subbands[l][v]['energies']])
479             sb_eigfct_in_lead = np.array([ef for v in self.valleys for
    ef in self.subbands[l][v]['arrays']])
480             self.index_argsorted[l] = np.argsort(sb_enrgs_in_lead)
481             self.index_array[l] = np.array([sorted(enumerate(self.
    index_argsorted[l]), key=lambda x:x[1]).pop(i)[0] for i in range(
    len(sb_enrgs_in_lead))])
482             for i, valley in enumerate(self.valleys):
483                 self.subbands[l][valley]['indices'] = np.split(self
    .index_array[l], len(self.valleys))[i]
484             self.sorted_sb_enrgs[l] = sb_enrgs_in_lead[self.
    index_argsorted[l]]
485             self.sorted_sb_eigfcts[l] = sb_eigfct_in_lead[self.
    index_argsorted[l]]
486
487     def update_subbands(self, potential):
488         self.potential = potential
489         self.subbands = {}
490         for l in self.device.descr.lead_boundrs:
491             self.subbands[l] = {}
492             for v in self.valleys:
493                 self.subbands[l][v] = {}
494                 schrodingerEVP = SchrodingerEVP(self.device, self.
    potential, l, v)
495                 solutions = schrodingerEVP.solve()
496                 self.subbands[l][v]['energies'] = solutions[0]
497                 self.subbands[l][v]['arrays'] = solutions[1]
498                 self.subbands[l][v]['interpolated arrays'] = solutions
    [2]
499                 self.subbands[l][v]['number of'] = solutions[3]
500             self.subbands[l]['number of'] = sum([self.subbands[l][i]['
    number of'] for i in self.valleys])
501         self.sort_and_number_subband_energies()
502
503 class SchrodingerEVP(object):
504     def __init__(self, device, potential, leadnr, valley):
505         self.device = device
506         self.leadnr = leadnr
```

```
507         self.mesh = device.fe.leads[self.leadnr]['mesh']
508         self.meshtot = device.fe.mesh
509         self.valley = device.leads[leadnr].material.valleys[valley]
510         self.regionr = device.leads[leadnr].regionr
511         degree = prms['structure']['schroedinger']['shapefunctions
    degree']
512         self.V = df.FunctionSpace(self.mesh, 'CG', degree)
513         self.Vtot = df.FunctionSpace(device.fe.mesh, 'CG', degree)
514         self.pot_enrgy = -df.interpolate(potential, self.V)
515         cb_offset = df.interpolate(device.get_property('conduction_band
    '), self.V)
516         self.U = cb_offset + self.pot_enrgy
517         self.u = df.TrialFunction(self.V)
518         self.v = df.TestFunction(self.V)
519
520         self.bcs = [df.DirichletBC(self.V, df.Constant(0.0), df.
    DomainBoundary())]
521         oxide_bcs = []
522         if not prms['structure']['schroedinger']['wavefunction in oxide
    ']:
523             for regnr, reg in device.descr.regions.items():
524                 if isinstance(reg['material'], Oxide):
525                     bc = self.device.fe.
    get_dirichletbc_for_region_in_lead(self.V, leadnr, regnr, 0.0)
526                     if bc is not None: oxide_bcs.append(bc)
527         self.bcs += oxide_bcs
528
529         self.A = df.PETScMatrix()
530         self.B = df.PETScMatrix()
531         self.eigensolver = df.SLEPcEigenSolver(self.A, self.B)
532         self.eigensolver.parameters['spectrum'] = 'smallest magnitude'
533         # self.eigensolver.parameters['solver'] = 'krylov-schur'
534         # self.eigensolver.parameters['solver'] = 'arnoldi'
535
536         self.inverse_effective_mass_tensor = self.valley.
    get_inverse_effective_mass_tensor_in_global_coordinate_system(
537             self.regionr, self.mesh, device)     # Make the effective
    mass tensor first on the total mesh
538
539         self.dx = device.fe.leads[leadnr]['dx']
540
541     def solve_eigenequation(self, nr_of_eigenfunctions):
542
543         a = df.Constant(1.0 / 2.0) * inner(self.
    inverse_effective_mass_tensor*nabla_grad(self.u), nabla_grad(self.v)
    ) \
544             * self.dx + self.U * self.u * self.v * self.dx
545         b = self.u * self.v * self.dx
546         bcs = self.bcs
547         df.assemble(a, tensor=self.A)
548         df.assemble(b, tensor=self.B)
549         for bc in bcs:
550             bc.apply(self.A)  # bc.apply(B) is implied
551         self.eigensolver.solve(nr_of_eigenfunctions)
552
```

```
553     def solve(self):
554         nr_of_eigenfunctions = prms['structure']['states']['subbands']
555         eigenvals = []
556         eigenvects = []
557         minimum_eigenvalue = df.project(self.pot_enrgy, self.V).vector
    ().array().min()
558         self.solve_eigenequation(nr_of_eigenfunctions)
559         for i in range(nr_of_eigenfunctions):
560             val, ival, vect, ivect = self.eigensolver.get_eigenpair(i)
561             if ival != 0.0:
562                 raise NameError("Imaginary eigenvalues detected. Abort.
    ")
563             if val < minimum_eigenvalue:  # to get rid of unphysical
    solutions
564                 continue
565             eigenvals.append(val)
566             eigenvects.append(vect.array())
567         eigenvals = np.array(eigenvals)
568         eigenvects = np.array(eigenvects)
569 
570         # normalize the eigenfunctions and interpolate
571         psi = df.Function(self.V)
572         interpolated_eigenvects = np.empty(nr_of_eigenfunctions, dtype=
    np.ndarray)
573         for i, vect in enumerate(eigenvects):
574             psi.vector()[:] = vect
575             scalefactor = df.assemble(psi**2 * self.dx)
576             normalized_vect = vect / scalefactor**0.5
577             eigenvects[i] = normalized_vect
578             psi.vector()[:] = normalized_vect
579             interpolated_eigenvects[i] = self.device.fe.
    map_function_on_boundary_to_function_on_entire_device(psi).vector().
    array()
580             f = df.File("eigenfunction" + str(i) + "_" + str(self.
    valley.type) + ".pvd")
581             f << psi
582         return eigenvals, eigenvects, interpolated_eigenvects,
    nr_of_eigenfunctions
```

B.2 Device description and finite elements

```
1  import dolfin as df
2  import auxiliary.materials as mat
3  import geoparser
4  import logging
5  import numpy as np
6  import os
7  import yaml
8  from auxiliary.settings import unitsim
9  from fdint import fdk
10 from scipy.optimize import brentq
11 from sh import gmsh, python, which
12 from structure.generic import ElectronicStructure
13 from structure.poisson import PoissonPDE
14 
```

```
15 prms = unitsim.parameters
16 df.parameters['allow_extrapolation'] = True
17 df.set_log_level(df.INFO)
18 logger = logging.getLogger('unit_simulation_logger')
19
20
21 class Device(object):
22     def __new__(cls, *args, **kwargs):
23         if prms['device']['type'] == 'axial symmetric':
24             return object.__new__(AxialSymmetricDevice, *args, **kwargs
    )
25         if prms['device']['type'] == 'realistic':
26             return object.__new__(RealisticDevice, *args, **kwargs)
27         if prms['device']['type'] == 'linear':
28             return object.__new__(LinearDevice, *args, **kwargs)
29
30     def __init__(self, geofile, descriptionfile):
31         if not os.path.isfile("mesh.xml"):
32             parsed_geometry = geoparser.process(geofile)
33             with open(geofile, 'w') as f:
34                 f.write(parsed_geometry)
35             if prms['device']['type'] in ['axial symmetric']:
36                 gmsh('-2', geofile)
37             elif prms['device']['type'] in ['realistic']:
38                 gmsh('-3', geofile)
39             elif prms['device']['type'] in ['linear']:
40                 gmsh('-1', geofile)
41             dolfin_convert = which("dolfin-convert")
42             python(dolfin_convert, "mesh.msh", "mesh.xml")
43             os.rename("mesh_facet_region.xml", "mesh_boundaries.xml")
44             os.rename("mesh_physical_region.xml", "mesh_regions.xml")
45         self.type = prms['device']['type']
46         self.descr = DeviceDescription(descriptionfile)
47         self.fe = FiniteElements("mesh.xml", "mesh_regions.xml", "
    mesh_boundaries.xml", self)
48         self.leads = {}
49         for l in self.descr.lead_boundrs:
50             self.leads[l] = Lead(l, self)
51         self.descr.define_gates(self)
52         if prms['structure']['states']['tensor']:
53             self.tensor_flag = True
54         else:
55             self.tensor_flag = False
56
57     def set_environment(self, environmentfile):
58         self.environment = DeviceEnvironment(environmentfile)
59         for l in self.leads.itervalues():
60             l.set_environment(self.environment)
61
62     def update_environment(self, env, include_quantum=True):
63         self.environment.update(env.__dict__)
64         for l in self.leads.itervalues():
65             l.set_environment(self.environment, include_quantum)
66
67     def find_electronic_structure(self):
```

```
68          self.elstruct = ElectronicStructure(self, self.environment)
69
70      def get_property(self, prop, domain='normal'):
71          """Return the requested property as a function.
72
73          For band-related properties, the conduction band (at a given
    point) in the reference contact is defined to be
74          zero. Offsets are either defined from literature or using a
    difference in electron affinity.
75
76          """
77
78          devprops = {'doping': 'doping'}
79          matprops = {'effective_mass': 'me_eff_gamma', 'permittivity': '
    permittivity',
80                      'electron_mobility': 'elec_mobility', '
    electron_diffusion': 'elec_diffusion',
81                      'effective_electron_density': 'Nc', 'bandgap': '
    bandgap',
82                      'hole_mobility': 'hole_mobility', 'hole_diffusion':
     'hole_diffusion',
83                      'effective_hole_density': 'Nv'}
84          bandprops = ['conduction_band', 'valence_band']
85          if domain == 'normal':
86              mesh = self.fe.mesh
87              regions_meshfct = self.fe.regions
88              propattr = prop
89          elif domain == 'oxideless':
90              mesh = self.fe.oxideless_mesh
91              regions_meshfct = self.fe.oxideless_mesh_regions
92              propattr = 'oxideless_' + prop
93          if not hasattr(self, propattr):
94              propfct = df.Function(df.FunctionSpace(mesh, 'DG', 0))
95              temp = np.array(regions_meshfct.array(), dtype=int)
96              regions = self.descr.regions
97              if prop in matprops.keys():
98                  vals = [getattr(regions[i]['material'], matprops[prop])
     for i in range(len(regions))]
99              elif prop in devprops.keys():
100                 vals = [regions[i][devprops[prop]] for i in range(len(
    regions))]
101             elif prop in bandprops:
102                 ref_material = self.leads[self.descr.reference_contact
    ].material
103                 vals = [ref_material.find_offset_with(regions[i]['
    material']) for i in range(len(regions))]
104                 if prop == 'valence_band':
105                     bgvals = [getattr(regions[i]['material'], 'bandgap'
    ) for i in range(len(regions))]
106                     vals = [i - j for i, j in zip(vals, bgvals)]
107             propfct.vector()[:] = np.choose(temp, vals)
108             setattr(self, propattr, propfct)
109         return getattr(self, propattr)
110
111     def get_effective_mass_values(self, prop):
```

```
112         """Return the effective mass value me_long or me_trans for a
    region to make the effective mass tensor
113         for that region"""
114         effective_masses = {'me_long': 'me_long', 'me_trans': 'me_trans
    '}
115 
116         if not hasattr(self, prop):
117             regions = self.descr.regions
118             if prop in effective_masses.keys():
119                 vals = [getattr(regions[i]['material'],
    effective_masses[prop]) for i in range(len(regions))]
120                 setattr(self, prop, vals)
121         return getattr(self, prop)
122 
123     def plot_mesh(self):
124         df.plot(self.fe.mesh, interactive=True)
125 
126     def plot_regions(self):
127         df.plot(self.fe.regions, interactive=True)
128 
129     def plot_boundaries(self):
130         df.plot(self.fe.boundaries, interactive=True)
131 
132 
133 class DeviceDescription(object):
134     """All information to describe the regions and boundaries as
    defined in the `FiniteElements` class."""
135     def __init__(self, descriptionfile):
136         with open(descriptionfile, 'r') as f:
137             description = yaml.load(f)
138         self.regions = description["regions"]
139         for r in self.regions.itervalues():
140             r['material'] = eval('mat.' + r['material'])
141         self.boundaries = description["boundaries"]
142         if "workfunctions" in description.keys():
143             self.workfunctions = description["workfunctions"]
144         self.lead_boundrs = [i for i, kind in self.boundaries.iteritems
    () if kind == "lead"]
145         self.gate_boundrs = [i for i, kind in self.boundaries.iteritems
    () if kind == "gate"]
146         self.contact_boundrs = self.lead_boundrs + self.gate_boundrs
147         self.air_boundrs = [i for i, kind in self.boundaries.iteritems
    () if kind == "vacuum"]
148         self.reference_contact = min(self.lead_boundrs)
149         if "region_groups" in description.keys():
150             self.region_groups = description["region_groups"]
151 
152     def define_gates(self, device):
153         self.gates = {}
154         def find_regionr():
155             """This method assumes that a gate is connected to one
    region only (e.g. an oxide region)."""
156             for f in df.facets(device.fe.mesh):
157                 if device.fe.boundaries[f.index()] == g:
158                     for cell in df.cells(f):
```

```
159                         regionnr = device.fe.regions[cell.index()]
160                         material = device.descr.regions[regionnr]['
    material']
161                         return regionnr
162         for g in self.gate_boundrs:
163             gate = {}
164             regionr = find_regionr()
165             gate['regionr'] = regionr
166             gate['material'] = self.regions[regionr]['material']
167             try:
168                 gate['workfunction'] = self.workfunctions[g]
169             except:
170                 gate['workfunction'] = None
171             self.gates[g] = gate
172 
173 
174 class FiniteElements(object):
175     """A class that contains the basis data used in the finite element
    method."""
176     def __init__(self, meshfile, meshregionfile, meshboundaryfile,
    device):
177         self.mesh = df.Mesh(meshfile)
178         self.regions = df.MeshFunction('size_t', self.mesh,
    meshregionfile)
179         self.boundaries = df.MeshFunction('size_t', self.mesh,
    meshboundaryfile)
180         self.dx = df.Measure('dx')[self.regions]
181         self.ds = df.Measure('ds')[self.boundaries]
182         self.boundarymesh = df.BoundaryMesh(self.mesh, 'exterior')
183         self.device = device
184         bdim = self.boundarymesh.topology().dim()
185         boundary_boundaries = df.MeshFunction('size_t', self.
    boundarymesh, bdim)
186         boundary_boundaries.set_all(0)
187         for i, facet in enumerate(df.entities(self.boundarymesh, bdim))
    :
188             parent_meshentity = self.boundarymesh.entity_map(bdim)[i]
189             parent_boundarynumber = self.boundaries.array()[
    parent_meshentity]
190             boundary_boundaries.array()[i] = parent_boundarynumber
191         self.leads = {}
192         for l in device.descr.lead_boundrs:
193             self.leads[l] = {}
194             subm = df.SubMesh(self.boundarymesh, boundary_boundaries, l
    )
195             self.leads[l]['mesh'] = subm
196             cf = df.CellFunction('size_t', subm)
197             cf.set_all(0)
198             self.leads[l]['dx'] = df.Measure('dx')[cf]
199 
200     def create_oxideless_mesh(self):
201         logger.info("Creating a mesh for the semiconductor regions (no
    oxide).")
202         oxideless_region = df.CellFunction('size_t', self.mesh)
203         oxideless_region.set_all(0)
```

```
204         for i, r in self.device.descr.regions.items():
205             if not isinstance(r['material'], mat.Oxide):
206                 oxideless_region.array()[self.regions.array() == i] = 1
207         self.oxideless_mesh = df.SubMesh(self.mesh, oxideless_region,
    1)
208         self.oxideless_mesh_regions = df.CellFunction('size_t', self.
    oxideless_mesh)
209         self.oxideless_mesh_regions.set_all(0)
210         self.oxideless_mesh_boundaries = df.FacetFunction('size_t',
    self.oxideless_mesh)
211         self.oxideless_mesh_boundaries.set_all(0)
212         dim = self.mesh.topology().dim()
213         vmap = self.oxideless_mesh.data().array('parent_vertex_indices'
    , 0)
214         cmap = self.oxideless_mesh.data().array('parent_cell_indices',
    dim)
215         self.oxideless_mesh.init(dim - 1, dim)
216         for submesh_cell in df.cells(self.oxideless_mesh):
217             parent_cell = cmap[submesh_cell.index()]
218             self.oxideless_mesh_regions.array()[submesh_cell.index()] =
     self.regions.array()[parent_cell]
219         self.interface_bnd = max(self.device.descr.boundaries.keys()) +
     1
220         for submesh_facet in df.facets(self.oxideless_mesh):
221             if submesh_facet.entities(dim).size == dim - 1:  # we are
    dealing with a boundary facet
222                 submesh_facet_vertices = vmap[submesh_facet.entities(0)
    ]
223                 for facet in df.facets(self.mesh):
224                     if self.boundaries.array()[facet.index()] == 0:
225                         if facet.entities(dim).size != dim - 1:  # we
    are dealing with a new boundary
226                             self.oxideless_mesh_boundaries.array()[
    submesh_facet.index()] = self.interface_bnd
227                         continue
228                     elif set(facet.entities(0)) == set(
    submesh_facet_vertices):
229                         self.oxideless_mesh_boundaries.array()[
    submesh_facet.index()] = self.boundaries.array()[facet.index()]
230                         break
231         self.oxideless_facetfunction = df.FacetFunction('size_t', self.
    mesh)
232         self.oxideless_facetfunction.set_all(0)
233         self.mesh.init(1, 2)
234         for facet in df.facets(self.mesh):
235             if 1 in oxideless_region.array()[facet.entities(2)]:
236                 self.oxideless_facetfunction[facet.index()] = 1
237         self.oxideless_mesh_exists = True
238
239     def map_function_on_boundary_to_function_on_entire_device(self,
    boundary_function):
240         """Used when surface integrals are calculated, but a function
    on V is needed."""
241         family = boundary_function.element().family()
242         degree = boundary_function.element().degree()
```

```
243         V = df.FunctionSpace(self.mesh, family, degree)
244         Vbnd = boundary_function.function_space()
245         bnd_mesh = boundary_function.function_space().mesh()
246         gdim = self.mesh.geometry().dim()
247         Vbnd_dofcoords = Vbnd.dofmap().tabulate_all_coordinates(
    bnd_mesh).reshape(-1, gdim)
248         V_dofcoords = V.dofmap().tabulate_all_coordinates(self.mesh).
    reshape(-1, gdim)
249         Vbnd_to_V_map = {}
250         if self.device.type == 'realistic':
251             V_dofcoords_order = np.lexsort(np.fliplr(V_dofcoords).T)
252             sorted_V_dofcoords = V_dofcoords[V_dofcoords_order]
253             for Vbnd_dof_nr, Vbnd_dof_coord in enumerate(Vbnd_dofcoords
    ):
254                 found = False
255                 to_search_through = sorted_V_dofcoords.copy()
256                 idx = 0
257                 dim = 3 if self.device.type == 'realistic' else 1
258                 for xi in range(3):
259                     lidx = np.searchsorted(to_search_through[:, xi],
    Vbnd_dof_coord[xi], side='left')
260                     uidx = np.searchsorted(to_search_through[:, xi],
    Vbnd_dof_coord[xi], side='right')
261                     to_search_through = to_search_through[lidx:uidx, :]
262                     idx += lidx
263                 if np.array_equal(V_dofcoords[V_dofcoords_order[idx]],
    Vbnd_dof_coord):
264                     Vbnd_to_V_map[Vbnd_dof_nr] = V_dofcoords_order[idx]
265                     found = True
266                 if not found:
267                     logger.error("Degrees of freedom don't correspond."
    )
268         elif self.device.type == 'axial symmetric':
269             for Vbnd_dof_nr, Vbnd_dof_coord in enumerate(Vbnd_dofcoords
    ):
270                 corresponding_dofs = [i for i, coords in enumerate(
    V_dofcoords) if np.array_equal(coords, Vbnd_dof_coord)]
271                 if len(corresponding_dofs) == 1:
272                     Vbnd_to_V_map[Vbnd_dof_nr] = corresponding_dofs[0]
273                 else:
274                     logger.error("Degrees of freedom don't correspond."
    )
275         fct = df.Function(V)
276         for Vbnd_dof, V_dof in Vbnd_to_V_map.iteritems():
277             fct.vector()[V_dof] = boundary_function.vector().array()[
    Vbnd_dof]
278         return fct
279 
280     def get_dirichletbc_for_region(self, functionspace, region, value):
281         """Effectively defines a Dirichlet boundary condition on a
    region instead of on a boundary."""
282         dim = self.mesh.topology().dim()
283         b = df.MeshFunction('size_t', self.mesh, dim - 1)
284         b.set_all(0)
285         self.mesh.init(dim - 1, dim)
```

```
286         for f in df.facets(self.mesh):
287             if region in self.regions.array()[f.entities(dim)]:
288                 b[f.index()] = 1
289         if isinstance(value, df.Function) or isinstance(value, df.
    Expression):
290             return df.DirichletBC(functionspace, value, b, 1)
291         else:
292             return df.DirichletBC(functionspace, df.Constant(value), b,
     1)
293 
294     def get_dirichletbc_for_region_in_lead(self, functionspace, leadnr,
     region, value):
295         """If `region` is part of the leadboundary of lead `leadnr`,
    return a BC on this region."""
296         lead_mesh = self.leads[leadnr]['mesh']
297         dim = lead_mesh.topology().dim()
298         self.mesh.init(dim, dim + 1)
299 
300         # create a CellFunction that is 1 if the boundary cell
    corresponds to the requested region
301         cfct = df.MeshFunction('size_t', lead_mesh, dim)
302         cfct.set_all(0)
303         for c in df.cells(lead_mesh):
304             lead_mesh_idx = c.index()
305             bnd_mesh_idx = lead_mesh.data().array('parent_cell_indices'
    , dim)[lead_mesh_idx]
306             mesh_idx = self.boundarymesh.entity_map(dim).array()[
    bnd_mesh_idx]
307             mesh_cell = df.Facet(self.mesh, mesh_idx).entities(dim + 1)
    [0]
308             bnd_region = self.regions.array()[mesh_cell]
309             if region == bnd_region:
310                 cfct[lead_mesh_idx] = 1
311 
312         # rewrite this CellFunction to a FacetFunction and use it to
    define the boundary condition
313         bfct = df.MeshFunction('size_t', lead_mesh, dim - 1)
314         bfct.set_all(0)
315         lead_mesh.init(dim - 1, dim)
316         for f in df.facets(lead_mesh):
317             if 1 in cfct.array()[f.entities(dim)]:
318                 bfct[f.index()] = 1
319         if bfct.array().nonzero()[0].size > 0:
320             return df.DirichletBC(functionspace, df.Constant(value),
    bfct, 1)
321         else:
322             return None
323 
324 
325 class Lead(object):
326     """A class containing data of the leads, particularly useful to
    find boundary conditions for Poisson."""
327     def __init__(self, leadnr, device):
328         self.leadnr = leadnr
329         self.device = device
```

```
330         self.mesh = device.fe.leads[leadnr]['mesh']
331 
332         def find_regionr():
333             """This method for the moment assumes that a lead consists
    of exactly 1 semiconductor
334             with 1 level op doping."""
335             for f in df.facets(device.fe.mesh):
336                 if device.fe.boundaries[f.index()] == leadnr:
337                     for cell in df.cells(f):
338                         regionr = device.fe.regions[cell.index()]
339                         material = device.descr.regions[regionr]['
    material']
340                         if isinstance(material, mat.Semiconductor) and
    not isinstance(material, mat.Oxide):
341                             return regionr
342         self.regionr = find_regionr()
343         self.material = device.descr.regions[self.regionr]['material']
344         self.doping = device.descr.regions[self.regionr]['doping']
345         self.eff_mass = self.material.me_eff_gamma
346         self.sc_fermilevel_offset = None
347         self.qm_fermilevel_offset = None
348         self.qm_fermilevel_offsets = None
349 
350     def set_environment(self, environment, include_quantum=True):
351         self.temperature = environment.temperature
352         self.potential = environment.potentials[self.leadnr]
353         self.find_semiclassical_fermilevel_offset()
354         if include_quantum:
355             self.find_quantummechanical_fermilevel_offset()
356         else:
357             self.qm_fermilevel_offset = None
358 
359     def find_semiclassical_fermilevel_offset(self):
360         """This method returns the semiclassical difference between
    conductionband and Fermilevel.
361 
362         E.g. the returned value is positive if the Fermilevel lies
    within the bandgap.
363 
364         """
365 
366         if self.sc_fermilevel_offset is None:
367             dopingtype = prms['device']['doping']
368             self.sc_fermilevel_offset = self.material.find_fermi_level(
    self.doping, self.temperature, dopingtype=dopingtype)
369         return self.sc_fermilevel_offset
370 
371     def find_quantummechanical_fermilevel_offset(self):
372         """This method returns the quantummechanical difference between
     conductionband and Fermilevel.
373 
374         Just like in the semiclassical case, the returned value is
    positive if the Fermilevel lies within the bandgap.
375 
```

```
376         If the device is a realistic device and tensor == True, return
    a list with offsets
377         (for each valley in the lead there is a different offset)
378 
379         """
380 
381         logger.info("Calculating quantummechanical offset for lead " +
    str(self.leadnr))
382         if self.qm_fermilevel_offset is None:
383             zero_potential = df.Function(df.FunctionSpace(self.device.
    fe.mesh, 'CG', 1))
384             dummy_one = df.Function(df.FunctionSpace(self.device.fe.
    mesh, 'DG', 0))
385             dummy_one.vector()[:] = 1.0
386             ds = self.device.fe.ds
387             if isinstance(self.device, RealisticDevice):
388                 if prms['structure']['states']['tensor']:
389                     from structure.qtbmimpl.realistic_1band_tensor
    import SchrodingerEVP
390                     schrodingerEVP = SchrodingerEVP(self.device,
    zero_potential, self.leadnr, '100_valley')
391                     self.effectivemassa = 0.19
392                 elif not prms['structure']['states']['tensor']:
393                     from structure.qtbmimpl.realistic_1band import
    SchrodingerEVP
394                     schrodingerEVP = SchrodingerEVP(self.device,
    zero_potential, self.leadnr)
395                     self.effectivemassa = 0.063
396                 solutions = schrodingerEVP.solve()
397                 subband_enrgs = solutions[0]
398                 self.cross_surface = df.assemble(dummy_one * ds(self.
    leadnr))
399                 def neg_Q(mu):
400                     fact = np.sqrt(2.0 * self.effectivemassa * self.
    temperature / np.pi)
401                     neg_charge = 0.0
402                     for enrgy in subband_enrgs:
403                         deg = 1
404                         gammafactor = deg / np.pi**0.5
405                         fditg = fdk(-0.5, ((mu - enrgy) / self.
    temperature))
406                         neg_charge += gammafactor * fditg
407                     return fact * neg_charge
408                 if prms['device']['doping'] == 'real':
409                     pos_Q = lambda mu: self.cross_surface * self.doping
     * (1 / (1 + 2.0 * np.exp(mu/self.temperature)))
410                 elif prms['device']['doping'] == 'ionized':
411                     pos_Q = lambda mu: self.cross_surface * self.doping
412                 self.qm_fermilevel_offset = -brentq(lambda mu: -neg_Q(
    mu) + pos_Q(mu), -0.1, 0.1)
413                 from auxiliary.atomicunits import eV
414                 print "Quantummechanical offset is %f eV." % float(self
    .qm_fermilevel_offset / eV) + "for lead " + str(self.leadnr)
415             if isinstance(self.device, AxialSymmetricDevice):
```

```
416                 from structure.qtbmimpl.axialsymmetric_unipolar import
    SchrodingerEVP
417                 schrodingerEVP = SchrodingerEVP(self.device,
    zero_potential, self.leadnr)
418                 solutions = schrodingerEVP.solve()
419                 subband_ns = solutions[0]
420                 subband_enrgs = solutions[1]
421                 self.cross_surface = df.assemble(dummy_one * ds(self.
    leadnr))**2 * np.pi
422                 def neg_Q(mu):
423                     if prms['structure']['states']['tensor'] == True:
424                         fact = np.sqrt(2.0 * self.
    lead_effective_mass_along_transport * self.temperature / np.pi)
425                     else:
426                         fact = np.sqrt(2.0 * self.eff_mass * self.
    temperature / np.pi)
427                     neg_charge = 0.0
428                     for m, enrgy in zip(subband_ns, subband_enrgs):
429                         deg = 1 if m == 0 else 2
430                         gammafactor = deg / np.pi**0.5
431                         fditg = fdk(-0.5, ((mu - enrgy) / self.
    temperature))
432                         neg_charge += gammafactor * fditg
433                     return fact * neg_charge
434                 pos_Q = lambda mu: self.cross_surface * self.doping *
    (1 / (1 + 2.0 * np.exp(mu/self.temperature)))
435                 self.qm_fermilevel_offset = -brentq(lambda mu: -neg_Q(
    mu) + pos_Q(mu), -0.1, 0.1)
436                 from auxiliary.atomicunits import eV
437                 print "Quantummechanical offset is %f eV." % float(self
    .qm_fermilevel_offset / eV)
438             if isinstance(self.device, LinearDevice):
439                 self.qm_fermilevel_offset = self.sc_fermilevel_offset
440         return self.qm_fermilevel_offset
441
442
443 class DeviceEnvironment(object):
444     """This class describes the temperature, applied potentials, etc.
    """
445     def __init__(self, environmentfile):
446         with open(environmentfile, 'r') as f:
447             env = yaml.load(f)
448         self.temperature = env['temperature']
449         self.potentials = env['potentials']
450
451     def update(self, env):
452         self.temperature = env['temperature']
453         self.potentials = env['potentials']
454
455
456 class RealisticDevice(Device):
457     """This class describes all devices that are 3 dimensional and thus
     give rise to realistic devices."""
458     def __init__(self, *args, **kwargs):
459         Device.__init__(self, *args, **kwargs)
```

B.3 Poisson equation

```
import dolfin as df
import logging
import numpy as np
from auxiliary.atomicunits import eV
from auxiliary.physics import fd
from fdint import fdk, ifd1h
from auxiliary.settings import unitsim
from dolfin import div, inner, grad, nabla_grad, exp, log

prms = unitsim.parameters
df.parameters['allow_extrapolation'] = True
logger = logging.getLogger('unit_simulation_logger')


class PoissonPDE(object):
    def __new__(cls, *args, **kwargs):
        """Depending on the method defined in the settings file, return the correct class."""
        if prms['device']['type'] == 'axial symmetric':
            return object.__new__(AxialSymmetricPoisson, *args, **kwargs)
        elif prms['device']['type'] == 'realistic':
            return object.__new__(RealisticPoisson, *args, **kwargs)
        elif prms['device']['type'] == 'linear':
            return object.__new__(LinearPoisson, *args, **kwargs)

    def __init__(self, device, shapefct_degree=None):
        self.device = device
        self.mesh = device.fe.mesh
        degree = shapefct_degree or prms['structure']['poisson']['shapefunctions degree']
        self.V = df.FunctionSpace(self.mesh, 'CG', degree)

    def update_boundary_conditions(self, type, equilibrium=False, lead_bc_type='Neumann'):
        logger.info("Defining boundary conditions for Poisson.")
        reference_lead = self.device.leads[self.device.descr.reference_contact]
        reference_material = reference_lead.material
        reference_affinity = reference_lead.material.affinity
        self.lead_bcs = []
        self.gate_bcs = []
        bc_vals = {}
        if type == 'semiclassical':
            offset = 'sc_fermilevel_offset'
        elif type == 'quantum':
            offset = 'qm_fermilevel_offset'
        for i, lead in self.device.leads.iteritems():
            conduction_band_difference = reference_material.find_offset_with(lead.material)
            fermilevel_offset = getattr(lead, offset)
            if not equilibrium:
                potential = lead.potential
```

```
48              else:
49                  potential = self.device.leads[self.device.descr.
    reference_contact].potential
50              dir_value = potential + conduction_band_difference -
    fermilevel_offset
51              dir_bc = df.DirichletBC(self.V, df.Constant(dir_value),
    self.device.fe.boundaries, i)
52              bc_vals[i] = dir_value
53              self.lead_bcs.append(dir_bc)
54          for g in self.device.descr.gate_boundrs:
55              # if no workfuntion is provided, it is chosen such that vgs
    =0 corresponds to flatband
56              gate = self.device.descr.gates[g]
57              if prms['structure']['formalism'] == 'qtbm':
58                  workfunction = gate['workfunction'] or getattr(
    reference_lead, 'qm_fermilevel_offset') + reference_affinity
59              else:
60                  workfunction = gate['workfunction'] or getattr(
    reference_lead, 'sc_fermilevel_offset') + reference_affinity
61              # the workfunction in the semiclassical case is adjusted
    such that, going from
62              # a semiclassical potential to that for the quantum case,
    the potential can just be
63              # shifted with a given value
64              if type == 'semiclassical' and prms['structure']['formalism
    '] == 'qtbm':
65                  shift = reference_lead.sc_fermilevel_offset -
    reference_lead.qm_fermilevel_offset
66                  workfunction += shift
67              dir_value = reference_lead.potential + self.device.
    environment.potentials[g] - \
68                          (workfunction - gate['material'].affinity) +
    reference_material.find_offset_with(gate['material'])
69              dir_bc = df.DirichletBC(self.V, df.Constant(dir_value),
    self.device.fe.boundaries, g)
70              bc_vals[g] = dir_value
71              self.gate_bcs.append(dir_bc)
72          if type == 'semiclassical':
73              self.bcs = self.gate_bcs
74          elif type == 'quantum' and lead_bc_type == 'Neumann':
75              self.bcs = self.gate_bcs
76          elif type == 'quantum' and lead_bc_type == 'Dirichlet':
77              pass
78          else:
79              logger.error("No proper boundary type for the leads defined
    .")
80          return bc_vals
81
82      def return_dummy_piecewise_constant_potential(self, method='
    semiclassical'):
83          """Return a simple piecewise contant potential.
84
85          Regions determines whether a domain takes the source/gate/drain
     potential, eg.
```

```
86          `region_groups` = {2: [1, 2], 4: [3], 3: [4]} where keys refer
    to boundaries, and values to lists of regions.
87          If not all regions belong to a contact, those that do are set
    to the corresponding value, and the others are
88          interpolated by solving Poisson.
89
90          """
91          fixed_regions = set([r for regs in [reg_list for reg_list in
    self.device.descr.region_groups.values()] for r in regs])
92          if fixed_regions == set(self.device.descr.regions.keys()):
93              bc_vals = self.update_boundary_conditions(method)
94              temp = np.array(self.device.fe.regions.array(), dtype=int)
95              vals = []
96              for r in range(len(self.device.descr.regions)):
97                  for c, regs in self.device.descr.region_groups.
    iteritems():
98                      if r in regs:
99                          corresponding_contact = c
100                         val = bc_vals[c]
101                         break
102             vals.append(val)
103         V = df.FunctionSpace(self.device.fe.mesh, 'DG', 0)
104         piecewise_cte_potential = df.Function(V)
105         piecewise_cte_potential.vector()[:] = np.choose(temp, vals)
106         piecewise_cte_potential = df.project(
    piecewise_cte_potential, self.V, solver_type='lu')
107     else:
108         bc_vals = self.update_boundary_conditions(method)
109         bcs = []
110         for fr in fixed_regions:
111             for c, regs in self.device.descr.region_groups.
    iteritems():
112                 if fr in regs:
113                     corresponding_contact = c
114                     val = bc_vals[c]
115                     break
116             bc = self.device.fe.get_dirichletbc_for_region(self.V,
    fr, val)
117             bcs.append(bc)
118         zerofct = df.Function(self.V)
119         zerofct.vector()[:] = 0.0
120         piecewise_cte_potential = self.solve(zerofct, bcs=bcs)
121     return piecewise_cte_potential
122
123
124 class RealisticPoisson(PoissonPDE):
125     def __init__(self, *args, **kwargs):
126         PoissonPDE.__init__(self, *args, **kwargs)
127
128     def solve(self, charge, bcs=None):
129         u = df.TrialFunction(self.V)
130         v = df.TestFunction(self.V)
131         phi = df.Function(self.V)
132         dx = self.device.fe.dx
133         eps = self.device.get_property('permittivity')
```

```
134         bcs = bcs or self.bcs
135         a = inner(grad(u), grad(v)) * dx
136         L = charge * v / eps * dx
137         df.solve(a == L, phi, bcs)
138         return phi
139     pass
```

B.4 Schrödinger-Poisson

```
1  import dolfin as df
2  import logging
3  import numpy as np
4  import os
5  import tables as tb
6  import yaml
7  from auxiliary.atomicunits import eV
8  from auxiliary.settings import unitsim
9  from dolfin import exp
10 from scipy.optimize import anderson
11 from structure.poisson import PoissonPDE
12 from structure.semiclassical import SemiclassicalStructure
13
14 prms = unitsim.parameters
15 logger = logging.getLogger('unit_simulation_logger')
16
17
18 class QtbmStructure(object):
19     """Contains the microscopic structure of the device, following the
       Quantum Transmitting Boundary Method.
20
21     The method above is used to find all energy eigenstates; these need
        be found selfconsistently with the potential.
22     Some transport formalism is to be used to determine the occupation
       of these states... which through feedback
23     again changes the potential, and hence the states themselves.
24
25     """
26
27     def __init__(self, device, environment):
28         self.device = device
29         self.environment = environment
30         self.states = QtbmStates(self.device)
31         self.poisson = PoissonPDE(self.device)
32         self.density_operator = DensityOperator(self.device, self.
       states, self.environment)
33         self.potential = df.Function(self.poisson.V)
34
35         # define the structure of the h5-file where iterative (and
       final) results will be saved
36         solution_size = self.poisson.V.dim()
37         solution_size_el = self.states.schroedinger.V.dim()
38         solution_type = df.Function(self.poisson.V).vector().array().
       dtype.name
39         nr_of_leads = len(self.device.descr.lead_boundrs)
40
```

```
41          class Structure(tb.IsDescription):
42              iter_nr = tb.UInt8Col()
43              phi_coeffs = tb.Col.from_sctype(solution_type,
    solution_size)
44              el_dens_coeffs = tb.Col.from_sctype(solution_type,
    solution_size)
45              dop_dens_coeffs = tb.Col.from_sctype(solution_type,
    solution_size)
46              charge_coeffs = tb.Col.from_sctype(solution_type,
    solution_size)
47              efn_coeffs = tb.Col.from_sctype(solution_type,
    solution_size)
48              current = tb.Float64Col((nr_of_leads, 2))
49
50          self.h5file = tb.open_file("results/qtbm_structure.h5", mode='w
    ', title="Potential and states.")
51          self.h5file.create_table('/', 'structure', Structure, "Table
    containing a state with all attributes.")
52          self.iteration_nr = 1
53          self.solve_poisson_schroedinger_iteratively()
54
55      def solve_poisson_schroedinger_iteratively(self):
56          # find the initial guess potential
57          logger.info("Defining the initial guess potential.")
58          if prms['structure']['poisson schroedinger']['initial guess']
    == 'semiclassical':
59              self.device.semicl_struct = SemiclassicalStructure(self.
    device, self.device.environment)
60              initial_potential_guess = df.project(self.device.
    semicl_struct.V, self.poisson.V, solver_type='lu')
61              # correct the semiclassical potential to account for the
    change in lead-Fermilevels
62              bcs = []
63              for i, l in self.device.leads.iteritems():
64                  corresponding_regions = self.device.descr.region_groups
    [i]
65                  qm_correction = l.sc_fermilevel_offset - l.
    qm_fermilevel_offset
66                  for r in corresponding_regions:
67                      bc = self.device.fe.get_dirichletbc_for_region(self
    .poisson.V, r, qm_correction)
68                      bcs.append(bc)
69              zerofct = df.Function(self.poisson.V)
70              zerofct.vector()[:] = 0.0
71              to_shift = self.poisson.solve(zerofct, bcs=bcs)
72              initial_potential_guess = df.project(
    initial_potential_guess + to_shift, self.poisson.V, solver_type='lu'
    )
73              Efn = df.project(self.device.semicl_struct.Efn, self.
    poisson.V, solver_type='lu')
74          elif prms['structure']['poisson schroedinger']['initial guess']
     == 'piecewise constant':
75              initial_potential_guess = self.poisson.
    return_dummy_piecewise_constant_potential(method='quantum')
76          bcs = []
```

```
77          for i, l in self.device.leads.iteritems():
78              corresponding_regions = self.device.descr.region_groups[i]
79              for r in corresponding_regions:
80                  bc = self.device.fe.get_dirichletbc_for_region(self.
    poisson.V, r, -l.potential)
81                  bcs.append(bc)
82          zerofct = df.Function(self.poisson.V)
83          zerofct.vector()[:] = 0.0
84          Efn = self.poisson.solve(zerofct, bcs=bcs)
85
86          # derive the ionized dopants
87          doping = self.device.get_property('doping')
88          T = df.Constant(self.device.environment.temperature)
89          if prms['device']['doping'] == 'real':
90              Nd = doping * (1.0 - 1.0 / (1 + 2.0 * exp((-
    initial_potential_guess - Efn) / T)))
91          else:
92              Nd = doping
93          dopants = df.project(Nd, self.poisson.V, solver_type='lu')
94
95          self.poisson.update_boundary_conditions('quantum', lead_bc_type
    ='Neumann')
96          table = self.h5file.root.structure
97          elec_struct = table.row
98
99          # run an initial iteration and save
100         logger.info("Finding states for the first time and calculating
    initial electron density.")
101         self.potential.vector()[:] = initial_potential_guess.vector().
    array()
102         self.states.update_states(self.potential)
103         logger.info("Finding electron density.")
104         self.electron_density = self.density_operator.
    calculate_electron_density(self.iteration_nr)
105         elec_struct['iter_nr'] = self.iteration_nr; elec_struct['
    phi_coeffs'] = self.potential.vector().array()
106         elec_struct['el_dens_coeffs'] = self.electron_density.vector().
    array()
107         charge = dopants - self.electron_density
108         elec_struct['charge_coeffs'] = df.project(charge, self.poisson.
    V, solver_type='lu').vector().array()
109         logger.info("Finding current.")
110         current = self.density_operator.find_current(self.iteration_nr)
111         elec_struct['current'] = current
112         elec_struct.append()
113         table.flush()
114         self.iteration_nr += 1
115
116         def F(ne_in_cfs):
117             # run another iteration
118             logger.info("Finding updated potential.")
119             self.electron_density.vector()[:] = ne_in_cfs
120             self.potential = self.poisson.solve_quantum_nonlinear(self.
    electron_density, self.potential)
121             logger.info("Finding corresponding new states.")
```

```
122             self.states.update_states(self.potential)
123             logger.info("Calculating new electron density.")
124             electron_density = self.density_operator.
    calculate_electron_density(self.iteration_nr)
125             ne_out_cfs = electron_density.vector().array()
126 
127             # save this iteration step
128             elec_struct['iter_nr'] = self.iteration_nr; elec_struct['
    phi_coeffs'] = self.potential.vector().array()
129             elec_struct['el_dens_coeffs'] = self.electron_density.
    vector().array()
130             charge = dopants - self.electron_density
131             elec_struct['charge_coeffs'] = df.project(charge, self.
    poisson.V, solver_type='lu').vector().array()
132             current = self.density_operator.find_current(self.
    iteration_nr)
133             elec_struct['current'] = current
134             elec_struct.append()
135             table.flush()
136             self.iteration_nr += 1
137 
138             return ne_out_cfs
139 
140         if prms['structure']['poisson schroedinger']['method'] == '
    underrelaxation':
141             ne_cfs = self.electron_density.vector().array()
142             eps = 1.0
143             iter = 1
144             tol = prms['structure']['poisson schroedinger']['
    convergence threshold']
145             maxiter = prms['structure']['poisson schroedinger']['max
    iterations']
146             maxdiff = prms['structure']['poisson schroedinger']['max
    difference']
147             while eps > tol and iter < maxiter:
148                 iter += 1
149                 logger.info("Starting iteration %i (max. iterations = %
    i), eps = %f (threshold = %f)", iter, maxiter, eps, tol)
150                 old_pot_cfs = self.potential.vector().array()
151                 ne_out_cfs = F(ne_cfs)
152                 new_pot_cfs = self.potential.vector().array()
153 
154                 d = np.linalg.norm(new_pot_cfs - old_pot_cfs, ord=np.
    Inf)
155                 alpha = min(1.0, maxdiff / d)
156                 logger.info("Using underrelaxation where alpha = %f",
    alpha)
157                 ne_next_cfs = alpha * ne_out_cfs + (1.0 - alpha) *
    ne_cfs
158                 diff = new_pot_cfs - old_pot_cfs
159                 eps = np.linalg.norm(diff, ord=np.Inf)
160                 ne_cfs = ne_next_cfs
161             if eps < tol:
162                 logger.info("Convergence! (eps = " + str(eps) + ")")
163             else:
```

```
164                 logger.error("No convergence reached in the Poisson-
    Schroedinger loop.")
165         self.h5file.close()
166
167
168 class QtbmStates(object):
169     """A class to find and contain all states for a given
    deviceconfiguration."""
170     def __new__(cls, *args, **kwargs):
171         if prms['structure']['states']['bands'] == 'conductionband only
    ' and \
172                 prms['device']['type'] == 'axial symmetric':
173             from structure.qtbmimpl.axialsymmetric_unipolar import
    States
174             return States(*args, **kwargs)
175         elif prms['structure']['states']['bands'] == 'conductionband
    only' and \
176                 prms['device']['type'] == 'realistic':
177             if prms['structure']['states']['tensor']:
178                 from structure.qtbmimpl.realistic_1band_tensor import
    States
179             elif not prms['structure']['states']['tensor']:
180                 from structure.qtbmimpl.realistic_1band import States
181             return States(*args, **kwargs)
182         elif prms['structure']['states']['bands'] == 'conductionband
    only' and \
183                 prms['device']['type'] == 'linear':
184             from structure.qtbmimpl.linear_1band import States
185             return States(*args, **kwargs)
186
187
188 class DensityOperator(object):
189     def __new__(cls, *args, **kwargs):
190         if prms['transport']['formalism'] == 'ballistic':
191             if prms['device']['type'] == 'axial symmetric':
192                 from structure.qtbmimpl.ballistic_do import
    AxialSymmetricBallisticDensityOperator
193                 return object.__new__(
    AxialSymmetricBallisticDensityOperator, *args, **kwargs)
194             elif prms['device']['type'] == 'linear':
195                 from structure.qtbmimpl.ballistic_do import
    LinearBallisticDensityOperator
196                 return object.__new__(LinearBallisticDensityOperator, *
    args, **kwargs)
197             elif prms['device']['type'] == 'realistic':
198                 from structure.qtbmimpl.ballistic_do import
    RealisticBallisticDensityOperator
199                 return object.__new__(RealisticBallisticDensityOperator
    , *args, **kwargs)
200
201     def __init__(self, device, states, environment):
202         self.device = device
203         self.states = states
204         self.environment = environment
```

B.5 Calculation of the charges and currents

```
import dolfin as df
import numpy as np
import tables as tb
from auxiliary.physics import fd
from auxiliary.settings import unitsim
from structure.qtbm import DensityOperator
from scipy.integrate import quad

prms = unitsim.parameters

class RealisticBallisticDensityOperator(DensityOperator):
    def __init__(self, device, states, environment):
        DensityOperator.__init__(self, device, states, environment)
        self.states = states

    def calculate_electron_density(self, iter_nr):
        temperature = self.environment.temperature
        degree = prms['structure']['schroedinger']['shapefunctions degree']
        V = df.FunctionSpace(self.device.fe.mesh, 'CG', degree)
        electron_density = df.Function(V)
        electron_density.vector()[:] = 0.0
        with tb.open_file("results/qtbm_states_iter" + str(iter_nr) + ".h5", mode='r') as h5file:
            psi_real = df.Function(V)
            psi_imag = df.Function(V)
            probability_density = df.Function(V)
            for state in h5file.root.states.iterrows():
                psi_real.vector()[:] = state['coeffs_real']
                psi_imag.vector()[:] = state['coeffs_imag']
                density_cfs = psi_real.vector().array()**2 + psi_imag.vector().array()**2
                probability_density.vector()[:] = density_cfs
                degeneracy = state['degeneracy']
                m_eff = self.device.leads[state['inj_lead']].material.me_long
                sb_enrgy = state['inj_subband_energy']
                mu = -self.environment.potentials[state['inj_lead']]
                f = lambda e: fd(e, mu, temperature)
                fact = (m_eff / 2.0)**0.5 / (2.0 * np.pi)
                fk = lambda k: fd(sb_enrgy + k**2, mu, temperature)
                if state['energy_lbnd'] - sb_enrgy < 0:
                    pass
                occupation = fact * 2.0 * quad(fk, (state['energy_lbnd'] - sb_enrgy)**0.5, (state['energy_ubnd'] - sb_enrgy)**0.5)[0]
                state_density = probability_density.vector().array() * occupation * degeneracy
                electron_density.vector()[:] = electron_density.vector().array() + state_density
        degree = prms['structure']['poisson']['shapefunctions degree']
        electron_density = df.project(electron_density, df.FunctionSpace(self.device.fe.mesh, 'CG', degree), solver_type='lu')
```

```
45          electron_density.vector()[:] = electron_density.vector().array
    ().clip(min=0.0)
46          return electron_density
47
48      def find_current_density(self):
49          """Calculate the current density as a vectorfunctions from all
    the states."""
50          pass
51
52      def find_current(self, iter_nr):
53          """Based on the transmission coefficients, calculate the
    current.
54
55          By convention, we define here current out of the device and
    hence into the leads as positive.
56
57          """
58
59          temperature = self.environment.temperature
60          currents = {}
61          for l in self.device.descr.lead_boundrs:
62              currents[l] = 0.0
63          with tb.open_file("results/qtbm_states_iter" + str(iter_nr) + "
    .h5", mode='r') as h5file:
64              table = h5file.root.states
65              for l in self.device.descr.lead_boundrs:
66                  mu = self.environment.potentials[l]
67                  for sb in range(self.states.subbands[l]['number of']):
68                      sb_state_nrs = np.array([state['number'] for state
    in table.iterrows() if state['inj_lead'] == l and state['inj_subband
    '] == sb])
69                      if len(sb_state_nrs) > 0:
70                          sb_degeneracy = table.cols.degeneracy[
    sb_state_nrs[0]]
71                          sb_state_enrgs = table.cols.energy[:][
    sb_state_nrs]
72                          s = np.argsort(sb_state_enrgs)
73                          sb_state_nrs = sb_state_nrs[s]
74                          sb_state_enrgs = sb_state_enrgs[s]
75                          sb_enrgy_lbnd = table.cols.energy_lbnd[
    sb_state_nrs[0]]
76                          sb_enrgy_ubnd = table.cols.energy_ubnd[
    sb_state_nrs[-1]]
77                          for l_out in self.device.descr.lead_boundrs:
78                              dummy_transm = table.cols.transmission[
    sb_state_nrs[0]]
79                              entry = np.argwhere(dummy_transm[:, 0] ==
    l_out)[0][0]
80                              if l_out != l:
81                                  transmissioncfs = table.cols.
    transmission[:][sb_state_nrs, entry, 1]
82                                  occ = lambda x: np.interp(x,
    sb_state_enrgs, transmissioncfs) * fd(x, mu, temperature)
83                                  currents[l_out] -= sb_degeneracy * quad
    (occ, sb_enrgy_lbnd, sb_enrgy_ubnd)[0]
```

```
                            else:
                                transmissioncfs = 1 - table.cols.
    transmission[:][sb_state_nrs, entry, 1]
                                occ = lambda x: np.interp(x,
    sb_state_enrgs, transmissioncfs) * fd(x, mu, temperature)
                                currents[l] += sb_degeneracy * quad(occ
    , sb_enrgy_lbnd, sb_enrgy_ubnd)[0]
        for l in currents.iterkeys():
            currents[l] /= 2.0 * np.pi
        currents_array = []
        for l, curr in currents.iteritems():
            currents_array.append([l, curr])
        return np.array(currents_array, dtype=float)
```

B.6 Properties of Si material

```
from auxiliary.atomicunits import *
from fdint import fdk
from numpy import exp, pi
from auxiliary.settings import unitsim
from scipy.optimize import brentq
from valley import Valley
import numpy as np

prms = unitsim.parameters

class Material(object):
    def __init__(self, **kwargs):
        for key in kwargs:
            setattr(self, key, kwargs[key])


class Semiconductor(Material):
    def find_fermi_level(self, doping, temperature=None, dopingtype='
    ionized'):
        """Find the conduction band minimum relative to the Fermi level
    .

        Atomic units are assumed. Not all dopant atoms are necessarily
    active/charged. This model makes use of
        effective density of states for electrons and holes and hence
    is very simple.

        """
        degenerate = True
        temp = temperature if temperature is not None else
    0.00095004462 # 300 K
        eg = self.bandgap
        Nc, Nv, = self.Nc, self.Nv
        if not degenerate:
            n = lambda Ef: Nc * exp(Ef / temp)
            p = lambda Ef: Nv * exp(-(eg + Ef) / temp)
        else:
            n = lambda Ef: Nc * 2.0 / pi**0.5 * fdk(0.5, Ef / temp)
            p = lambda Ef: Nv * fdk(k=0.5, phi=-(eg + Ef) / temp)
```

```
35          if dopingtype == 'real':
36              d = lambda Ef: doping * (1 - 1 / (1 + 2.0 * exp(-Ef/temp)))
37          elif dopingtype == 'ionized':
38              d = lambda Ef: doping
39          qnet = lambda Ef: d(Ef) - n(Ef) #+ p(Ef)
40          fermilevel = brentq(qnet, -eg - 0.1, 0.1, xtol=1e-14)   # 0.1
    Ha = 2.72 eV
41          return -fermilevel
42
43      def find_effective_carrier_density(self, temperature=None):
44          temp = temperature if temperature is not None else
    0.00095004462 # 300 K
45          if prms['structure']['states']['tensor'] == True:
46              self.Nc = (2.0 * self.me_long * temp / pi)**1.5 / 4.0
47              self.Nv = (2 * self.mhh * temp / pi)**1.5 / 4.0 + (2 * self
    .mlh * temp / pi)**1.5 / 4.0
48          else:
49              self.Nc = (2.0 * self.me_eff_gamma * temp / pi)**1.5 / 4.0
50              self.Nv = (2 * self.mhh * temp / pi)**1.5 / 4.0 + (2 * self
    .mlh * temp / pi)**1.5 / 4.0
51
52      def find_offset_with(self, material):
53          if self.band_offsets.has_key(material.name):
54              return self.band_offsets[material.name]
55          else:
56              return self.affinity - material.affinity
57
58
59  class Dopant(Material):
60      pass
61
62
63  class Oxide(Semiconductor):
64      pass
65
66  Si = Semiconductor()
67  Si.name = 'Si'
68  Si.permittivity = 11.68 * eps_0
69  Si.bandgap = 1.11 * eV
70  if prms['structure']['states']['tensor'] == True:
71      Si.me_trans = 0.19
72      Si.me_long = 0.98
73      Si.valleys = {}
74      if len(prms['structure']['states']['valleys']) == 3:
75          for v, valley in enumerate(prms['structure']['states']['valleys
    ']):
76              Si.valleys[prms['structure']['states']['valleys'][v]] =
    Valley(prms['structure']['states']['valleys'][v])
77      elif len(prms['structure']['states']['valleys']) == 1:
78          for v in prms['structure']['states']['valleys']:
79              Si.valleys[v] = Valley(v)
80
81  Si.mhh = 0.49
82  Si.mlh = 0.16
83  Si.affinity = 4.05 * eV
```

```
Si.me_eff_gamma = 0.19
Si.find_effective_carrier_density()
```

B.7 Creation of the effective mass tensors

```
import dolfin as df
import logging
import numpy as np
logger = logging.getLogger('unit_simulation_logger')
import os.path



class Valley(object):
    """A class that corresponds to a particular valley with all the
    characteristics defined for that valley.

    The valley is a conduction band valley of a specific semiconductor
    material.

    Each valley object holds its (inverse) effective mass tensor
    expressed in the valley coordinate system and the
    necessary rotation matrices to rotate the inverse effective mass
    tensor from the valley coordinate system to the
    global coordinate system.

            M*(gcs)^-1 = [ R(vcs <-- gcs) ]^T * M*(vcs)^-1 * [ R(vcs
    <-- gcs) ]

            with [ R(vcs <-- gcs) ] = [ R(vcs <-- ccs) ] * [ R(ccs <--
    lcs) ] * [ R(lcs <-- gcs) ]

            where vcs: "valley coordinate system", ccs: "crystal
    coordinate system", lcs: "lead coordinate system",
            gcs: "global coordinate system"
    """

    def __init__(self, valley_type):
        self.type = valley_type
        self.inverse_effective_mass_tensors_in_global_coordinate_system
    = {}

    def make_rotation_matrix_from_global_to_valley(self, valley_type):
        if valley_type == "100_valley":
            self.rotation_matrix_from_crystal_to_valley = df.as_matrix
    ([[0.0, 0.0, -1.0], [0.0, 1.0, 0.0], [1.0, 0.0, 0.0]])  # R(Y, 90)

        elif valley_type == "010_valley":
            self.rotation_matrix_from_crystal_to_valley = df.as_matrix
    ([[1.0, 0.0, 0.0], [0.0, 0.0, -1.0], [0.0, 1.0, 0.0]])  # R(X, 90)

        elif valley_type == "001_valley":
            self.rotation_matrix_from_crystal_to_valley = df.as_matrix
    ([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]])   # I

```

```
40          else:
41              logger.error("Something went wrong with the rotation
    matrices or the definition of the valleys")
42
43
44          self.rotation_matrix_from_lead_to_crystal = df.as_matrix([[1.0,
     0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]])    # I
45          self.rotation_matrix_from_global_to_lead = df.as_matrix([[1.0,
    0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]])    # I
46
47          self.rotation_matrix_from_global_to_valley = self.
    rotation_matrix_from_crystal_to_valley * \
48                                                       self.
    rotation_matrix_from_lead_to_crystal * \
49                                                       self.
    rotation_matrix_from_global_to_lead
50
51      def make_effective_mass_tensor_in_valley_coordinate_system(self,
    regionr, mesh, device):
52          """ Return the effective mass tensor in the valley coordinate
    system.
53
54          The region specifies automatically the material and the chosen
    valley for this region.
55
56          Return a 3D tensor in (k_transverse1, k_transverse2,
    k_longitudinal). This tensor is always diagonal.
57
58          Use me_long and me_trans from the material to make the
    effective mass tensor.
59          """
60
61          topological_dimension = mesh.ufl_cell().topological_dimension()
62
63          # Define MeshFunctions over each cell of the mesh
64          mxx = df.MeshFunction("double", mesh, topological_dimension)
65          mxy = df.MeshFunction("double", mesh, topological_dimension)
66          mxz = df.MeshFunction("double", mesh, topological_dimension)
67
68          myx = df.MeshFunction("double", mesh, topological_dimension)
69          myy = df.MeshFunction("double", mesh, topological_dimension)
70          myz = df.MeshFunction("double", mesh, topological_dimension)
71
72          mzx = df.MeshFunction("double", mesh, topological_dimension)
73          mzy = df.MeshFunction("double", mesh, topological_dimension)
74          mzz = df.MeshFunction("double", mesh, topological_dimension)
75
76          # This is the valley in the transport direction. Assumption:
    transport is in the z-direction.
77
78          # For each cell store the value of the component of the tensor
79          for cell in df.cells(mesh):
80              mxx[cell] = device.get_effective_mass_values('me_trans')[
    regionr]
81              mxy[cell] = 0.0
```

```
82              mxz[cell] = 0.0
83
84              myx[cell] = 0.0
85              myy[cell] = device.get_effective_mass_values('me_trans')[
    regionr]
86              myz[cell] = 0.0
87
88              mzx[cell] = 0.0
89              mzy[cell] = 0.0
90              mzz[cell] = device.get_effective_mass_values('me_long')[
    regionr]
91
92          # Store the meshfunctions (components of the tensor) into a
    file
93          mesh_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
     self.type + "/mesh.xml")
94
95          mxx_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
96                             "/mxx.xml")
97          mxy_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
98                             "/mxy.xml")
99          mxz_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
100                            "/mxz.xml")
101
102         myx_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
103                            "/myx.xml")
104         myy_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
105                            "/myy.xml")
106         myz_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
107                            "/myz.xml")
108
109         mzx_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
110                            "/mzx.xml")
111         mzy_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
112                            "/mzy.xml")
113         mzz_file = df.File("./3Dtensors/Region" + str(regionr) + "/" +
    self.type +
114                            "/mzz.xml")
115         mesh_file << mesh
116         mxx_file << mxx
117         mxy_file << mxy
118         mxz_file << mxz
119
120         myx_file << myx
121         myy_file << myy
122         myz_file << myz
123
```

```
124         mzx_file << mzx
125         mzy_file << mzy
126         mzz_file << mzz
127 
128         # Code for C++ evaluation of the tensor
129         self.tensorcode = """
130 
131         class Tensor : public Expression
132         {
133         public:
134             // Create expression with 9 components
135             Tensor() : Expression(9) {}
136 
137             // Function for evaluating expression on each cell
138             void eval(Array<double>& values, const Array<double>& x,
    const ufc::cell& cell) const
139           {
140             const uint D = cell.topological_dimension;
141             const uint cell_index = cell.index;
142             values[0] = (*mxx)[cell_index];
143             values[1] = (*mxy)[cell_index];
144             values[2] = (*mxz)[cell_index];
145 
146             values[3] = (*myx)[cell_index];
147             values[4] = (*myy)[cell_index];
148             values[5] = (*myz)[cell_index];
149 
150             values[6] = (*mzx)[cell_index];
151             values[7] = (*mzy)[cell_index];
152             values[8] = (*mzz)[cell_index];
153 
154           }
155 
156           // The data stored in mesh functions
157           std::shared_ptr<MeshFunction<double> > mxx;
158           std::shared_ptr<MeshFunction<double> > mxy;
159           std::shared_ptr<MeshFunction<double> > mxz;
160 
161           std::shared_ptr<MeshFunction<double> > myx;
162           std::shared_ptr<MeshFunction<double> > myy;
163           std::shared_ptr<MeshFunction<double> > myz;
164 
165           std::shared_ptr<MeshFunction<double> > mzx;
166           std::shared_ptr<MeshFunction<double> > mzy;
167           std::shared_ptr<MeshFunction<double> > mzz;
168 
169         };
170 
171         """
172 
173         # Define tensor expression and matrix
174         mxx = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
175                               "/" + self.type + "/mxx.xml")
```

```
176         mxy = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
177                               "/" + self.type + "/mxy.xml")
178         mxz = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
179                               "/" + self.type + "/mxz.xml")
180
181         myx = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
182                               "/" + self.type + "/myx.xml")
183         myy = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
184                               "/" + self.type + "/myy.xml")
185         myz = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
186                               "/" + self.type + "/myz.xml")
187
188         mzx = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
189                               "/" + self.type + "/mzx.xml")
190         mzy = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
191                               "/" + self.type + "/mzy.xml")
192         mzz = df.MeshFunction("double", mesh, "./3Dtensors/Region" +
    str(regionr) +
193                               "/" + self.type + "/mzz.xml")
194
195         t = df.Expression(cppcode=self.tensorcode)   # t stands for the
     tensor expression
196         t.mxx = mxx
197         t.mxy = mxy
198         t.mxz = mxz
199
200         t.myx = myx
201         t.myy = myy
202         t.myz = myz
203
204         t.mzx = mzx
205         t.mzy = mzy
206         t.mzz = mzz
207
208         self.effective_mass_tensor_in_valley_coordinate_system = df.
    as_tensor(((t[0], t[1], t[2]), (t[3], t[4], t[5]),
209
            (t[6], t[7], t[8])))
210
211     def rotate_from_valley_to_global_coordinate_system(self):
212         self.inverse_effective_mass_tensor_in_global_coordinate_system
    = df.transpose(
213             self.rotation_matrix_from_global_to_valley) * self.
    inverse_effective_mass_tensor_in_valley_coordinate_system\
214             * self.rotation_matrix_from_global_to_valley
215
216     def get_inverse_effective_mass_tensor_in_global_coordinate_system(
    self, regionr, mesh, device):
```

```
217         if not str(regionr) in self.
    inverse_effective_mass_tensors_in_global_coordinate_system.keys():
218 
219             logger.info("Creating a new effective mass tensor for
    region " + str(regionr) + " and valley type " + self.type)
220             self.make_rotation_matrix_from_global_to_valley(self.type)
221             self.make_effective_mass_tensor_in_valley_coordinate_system
    (regionr, mesh, device)
222             self.
    inverse_effective_mass_tensor_in_valley_coordinate_system = df.inv(
    self.effective_mass_tensor_in_valley_coordinate_system)
223             self.rotate_from_valley_to_global_coordinate_system()
224 
225             # add the assembled effective mass tensor to a dictionary
    for later use
226 
227             self.
    inverse_effective_mass_tensors_in_global_coordinate_system[str(
    regionr)] = self.
    inverse_effective_mass_tensor_in_global_coordinate_system
228 
229             return self.
    inverse_effective_mass_tensor_in_global_coordinate_system
230 
231         else:
232             logger.info("Looking up an existing effective mass tensor
    for region " + str(regionr) + " and valley type " + self.type)
233 
234             return self.
    inverse_effective_mass_tensors_in_global_coordinate_system[str(
    regionr)]
235 
236     def
    get_lead_effective_mass_along_transport_in_global_coordinate_system(
    self, mesh):
237         """ Return the effective mass in the lead along the transport
    direction.
238 
239         For now it is assumed that this is the zz-component of the
    tensor expressed in the global axis system.
240 
241         """
242         self.effective_mass_tensor_in_global_coordinate_system = df.inv
    (
243             self.
    inverse_effective_mass_tensor_in_global_coordinate_system)
244 
245         T = df.TensorFunctionSpace(mesh, 'CG', 1)
246         self.effective_mass_tensor_in_global_coordinate_system = df.
    project(self.effective_mass_tensor_in_global_coordinate_system, T)
247 
248         f2 = df.Function(df.FunctionSpace(mesh, 'CG', 1))
249 
250         df.assign(f2, self.
    effective_mass_tensor_in_global_coordinate_system.sub(8))
```

```
251         f2 = f2.vector().array()[0]
252         return f2
```

B.8 Definition of the input parameters

```
1  from auxiliary.atomicunits import *
2
3  default_unitsimulation_parameters = {
4      'device': {
5          'type': 'axial symmetric',
6          'doping': 'ionized'
7          },
8      'structure': {
9          'formalism': 'qtbm',
10         'semiclassical':{
11             'shapefunctions degree': 2
12             },
13         'poisson schroedinger': {
14             'initial guess': 'semiclassical',
15             'method': 'underrelaxation',
16             'nonlinear poisson': True,
17             'max iterations': 5,
18             'convergence threshold': 1e-4,
19             'max difference': 0.025 * eV
20             },
21         'states': {
22             'bands': 'conductionband only',
23             'subbands': 1,
24             'tensor': True,
25             'valleys': ["100_valley", "010_valley", "001_valley"],
26             'ediff': 0.01 * eV,
27             'cutoff energy': 0.25 * eV,
28             'refinement states': 20,
29             'max refinements': 2,
30             'solver': 'direct'
31             },
32         'schroedinger': {
33             'shapefunctions degree': 2,
34             'wavefunction in oxide': False
35             },
36         'poisson': {
37             'shapefunctions degree': 2,
38             'accurate shapefunctions degree': 4
39             }
40         },
41     'transport': {
42         'formalism': 'ballistic'
43         }
44     }
45
46 def dict_merge(dct, merge_dct):
47     """ Recursive dict merge. Inspired by :meth:``dict.update()``,
       instead of
48     updating only top-level keys, dict_merge recurses down into dicts
       nested
```

```
    to an arbitrary depth, updating keys. The ``merge_dct`` is merged
    into
    ``dct``.
    :param dct: dict onto which the merge is executed
    :param merge_dct: dct merged into dct
    :return: None
    """
    for k, v in merge_dct.iteritems():
        if (k in dct and isinstance(dct[k], dict)
                and isinstance(merge_dct[k], dict)):
            dict_merge(dct[k], merge_dct[k])
        else:
            dct[k] = merge_dct[k]


class UnitsimulationSettings(object):

    def __init__(self):
        self.default_parameters = default_unitsimulation_parameters

    def set_parameters(self, d):
        self.parameters = self.default_parameters.copy()
        dict_merge(self.parameters, d)

unitsim = UnitsimulationSettings()

"""
device:
  type: axial symmetric | realistic | linear
  doping: real | ionized
structure:
  formalism: qtbm | semiclassical
  semiclassical:
    shapefunctions degree: 4
  poisson schroedinger:
    initial guess: semiclassical | piecewise constant
    method: underrelaxation | anderson
    nonlinear poisson: True
    max iterations: 10
    convergence threshold: 0.00000001
    underrelaxation parameter: 0.5
  states:
    bands: conductionband only
    subbands: 10
    tensor: False
    valleys: ["100_valley", "010_valley", "001_valley"]
    ediff: 0.002 # eV
    cutoff energy: 0.25 # eV
    refinement states: 20
    max refinements: 2
    solver: direct
  schroedinger:
    shapefunctions degree: 2
    wavefunction in oxide: True
  poisson:
```

```
103     shapefunctions degree: 1
104     accurate shapefunctions degree: 2
105 transport:
106     formalism: ballistic
107 """
```

Bibliography

[1] Validity of effective mass theory for energy levels in si quantum wires. *Physica B: Condensed Matter*, 227(1-4):336–338, 1996. Proceedings of the Third International Symposium on New Phenomena in Mesoscopic Structures.

[2] Arm acquires sensinode to accelerate the internet of things and support 30 billion connected devices by 2020. *Economics-Business Week*, page 195, Sep 14 2013.

[3] Intel discloses newest microarchitecture and 14 nanometer manufacturing process technical details, Aug 11 2014. Journal Business Wire 2014; Last updated - 2014-08-11.

[4] N. Abele, R. Fritschi, K. Boucart, F. Casset, P. Ancey, and A. M. Ionescu. Suspended-gate mosfet: bringing new mems functionality into solid-state mos transistor. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pages 479–481, Dec 2005.

[5] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. 2005.

[6] H. R. Aliabad, H. Akbari, and M. Saeed. Evaluation of magneto-optic properties of new superconductors by {DFT}. *Computational Materials Science*, 106:5 – 14, 2015.

[7] M. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. Rognes, and G. Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[8] F. Alted, I. Vilata, et al. PyTables: Hierarchical datasets in Python, 2002. Open-source software.

[9] M. Bjoerk, S. Karg, J. Knoch, H. Riel, W. Riess, and H. Schmid. Metal-oxide-semiconductor device including a multiple-layer energy filter, 2012. US Patent 8,129,763.

[10] Christophe Geuzaine and Jean-Francois Remacle. Gmsh, a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. http://gmsh.info/, 2016. Open-source software.

[11] M. V. de Put. Band-to-band tunneling in iii-v semiconductor heterostructures. In *EUROCON, 2013 IEEE*, pages 2134–2139, July 2013.

[12] E. Gnani, A. Gnudi, S. Reggiani, and G. Baccarani. Steep-slope nanowire field-effect transistor. In *2010 International Conference on Simulation of Semiconductor Processes and Devices*, pages 69–72, Sept 2010.

[13] E. Gnani, A. Gnudi, S. Reggiani, and G. Baccarani. Theory of the junctionless nanowire fet. *IEEE Transactions on Electron Devices*, 58(9):2903–2910, Sept 2011.

[14] E. Gnani, P. Maiorano, S. Reggiani, A. Gnudi, and G. Baccarani. An investigation on steep-slope and low-power nanowire fets. In *2011 Proceedings of the European Solid-State Device Research Conference (ESSDERC)*, pages 299–302, Sept 2011.

[15] E. Gnani, P. Maiorano, S. Reggiani, A. Gnudi, and G. Baccarani. Performance limits of superlattice-based steep-slope nanowire fets. In *Electron Devices Meeting (IEDM), 2011 IEEE International*, pages 5.1.1–5.1.4, Dec 2011.

[16] E. Gnani, S. Reggiani, A. Gnudi, and G. Baccarani. Steep-slope nanowire fet with a superlattice in the source extension. In *2010 Proceedings of the European Solid State Device Research Conference*, pages 380–383, Sept 2010.

[17] E. Gnani, S. Reggiani, A. Gnudi, and G. Baccarani. Superlattice-based steep-slope switch. In *Solid-State and Integrated Circuit Technology (ICSICT), 2010 10th IEEE International Conference on*, pages 1227–1230, Nov 2010.

[18] K. Gopalakrishnan, P. B. Griffin, and J. D. Plummer. I-mos: a novel semiconductor device with a subthreshold slope lower than kt/q. In *Electron Devices Meeting, 2002. IEDM '02. International*, pages 289–292, Dec 2002.

[19] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864–B871, Nov 1964.

[20] IMEC magazine, April 2016 issue, Bart Preneel. Security and privacy: a must for the Internet of Things. http://magazine.imec.be/data/90/reader/reader.html?t=1463009424589#!preferred/1/package/90/pub/96/page/9, 2016. Online; accessed 20 April 2016.

[21] International Technology Roadmap for Semiconductors. ITRS Report. http://www.itrs2.net/, 2016. Online; accessed 6 May 2016.

[22] A. M. Ionescu and H. Riel. Tunnel field-effect transistors as energy-efficient electronic switches. *Nature*, 479(7373):329–37, Nov 17 2011.

[23] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. S. Kwak. The internet of things for health care: A comprehensive survey. *IEEE Access*, 3:678–708, 2015.

[24] F. Jammes. Internet of things in energy efficiency: The internet of things (ubiquity symposium). *Ubiquity*, 2016(February):2:1–2:8, Feb. 2016.

[25] H. Kam, D. T. Lee, R. T. Howe, and T.-J. King. A new nano-electro-mechanical field effect transistor (nemfet) design for low-power electronics. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pages 463–466, Dec 2005.

[26] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, 2003.

[27] C. S. Lent and D. J. Kirkner. The quantum transmitting boundary method. *Journal of Applied Physics*, 67(10):6353–6359, 1990.

[28] A. Logg, K.-A. Mardal, and G. Wells. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Springer Publishing Company, Incorporated, 2012.

[29] C. lozzio. Power to the internet of things. *Scientific American*, 311(6):30, 2014.

[30] F. M. Asadzadeh. A finite element crash course, 2004. Online accessed: 6 Sept. 2015.

[31] V. R. Manfrinato, L. Zhang, D. Su, H. Duan, R. G. Hobbs, E. A. Stach, and K. K. Berggren. Resolution limits of electron-beam lithography toward the atomic scale. *Nano Letters*, 13(4):1555–1558, 2013. PMID: 23488936.

[32] J. S. Naoya Morioka, Hironori Yoshioka and T. Kimoto. Quantum-confinement effect on holes in silicon nanowires: Relationship between wave function and band structure. *Journal of Applied Physics*, 2011(109), 2011.

[33] T. K. P. Vogl. The non-equilibrium green's function method: an introduction. http://www.wsi.tum.de/Portals/0/Media/Publications/047559a5-94fa-4aa7-9bb9-f9b09fd17a86/NEGF_CompElectron.pdf. Accessed 9 May 2016.

[34] A.-T. Pham, B. Sorée, W. Magnus, C. Jungemann, B. Meinerzhagen, and G. Pourtois. Quantum simulations of electrostatics in si cylindrical junctionless nanowire nfets and pfets with a homogeneous channel including strain and arbitrary crystallographic orientations. *Solid-State Electronics*, 71:30 – 36, 2012. Selected Papers from the {ULIS} 2011 Conference.

[35] E. Polizzi and N. B. Abdallah. Subband decomposition approach for the simulation of quantum electron transport in nanostructures. *Journal of Computational Physics*, 202(1):150 – 180, 2005.

[36] J. Quinn, G. Kawamoto, and B. McCombe. Subband spectroscopy by surface channel tunneling. *Surface Science*, 73:190 – 196, 1978.

[37] G. Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, 1995.

[38] A. K. S. E. Laux and M. V. Fischetti. Analysis of quantum ballistic electron transport in ultrasmall silicon devices including space-charge and geometric effects. *Journal of Applied Physics*, 2004(95), 2004.

[39] S. Salahuddin, , and S. Datta. Use of negative capacitance to provide voltage amplification for low power nanoscale devices. *Nano Letters*, 8(2):405–410, 2008. PMID: 18052402.

[40] S. Sarmah, A. K. Guha, and A. K. Phukan. Donor-acceptor complexes of normal and abnormal n-heterocyclic carbenes with group 13 (b, al, ga) elements: A combined dft and atoms-in-molecules study. *European Journal of Inorganic Chemistry*, 2013(18):3233–3239, 2013.

[41] The HDF Group. Hierarchical Data Format, version 5, 1995. Open-source software.

[42] The Statistics Portal, Ed. Statista Ltd London. Global semiconductor sales from January 2012 to February 2016. http://www.statista.com/statistics/277404/global-semiconductor-sales-by-month/, 2016. Online; accessed 6 May 2016.

[43] C. Thelander, P. Agarwal, S. Brongersma, J. Eymery, L. Feiner, A. Forchel, M. Scheffler, W. Riess, B. Ohlsson, U. Gosele, and L. Samuelson. Nanowire-based one-dimensional electronics. *Materials Today*, 9(10):28 – 35, 2006.

[44] C. Verdouw, J. Wolfert, A. Beulens, and A. Rialland. Virtualization of food supply chains with the internet of things. *Journal of Food Engineering*, 176:128 – 136, 2016.

[45] A. S. Verhulst, B. Sorée, D. Leonelli, W. G. Vandenberghe, and G. Groeseneken. Modeling the single-gate, double-gate, and gate-all-around tunnel field-effect transistor. *Journal of Applied Physics*, 107(2), 2010.

[46] D. Verreck, A. S. Verhulst, M. Van de Put, B. Sorée, W. Magnus, A. Mocuta, N. Collaert, A. Thean, and G. Groeseneken. Full-zone spectral envelope function formalism for the optimization of line and point tunnel field-effect transistors. *Journal of Applied Physics*, 118(13), 2015.

[47] A. M. Walke, A. S. Verhulst, A. Vandooren, D. Verreck, E. Simoen, V. R. Rao, G. Groeseneken, N. Collaert, and A. V. Y. Thean. Part i: Impact of field-induced quantum confinement on the subthreshold swing behavior of line tfets. *IEEE Transactions on Electron Devices*, 60(12):4057–4064, Dec 2013.

[48] V. William. *Quantum transport in tunnel field-effect transistors for future nano-CMOS applications*. PhD thesis, KU Leuven, 2012.

[49] L. Xiao and Z. Wang. Internet of things: a new application for intelligent traffic monitoring system. *Journal of Networks*, 6(6), 2011.

[50] X. Zhao et al. *Superlattice-source nanowire FET with steep Subthreshold characteristics*. PhD thesis, Massachusetts Institute of Technology, 2012.

[51] D. Zwillinger. *Handbook of Differential Equations*. Academic Press, 1989.