Comparison of the performance of ballistic schottky barrier Graphene Nanoribbon FET

By

Sheikh Ziauddin Ahmed (11121032)

Mashiyat Sumaiya Shawkat (11121001)

Md. Iramul Hoque Chowdhury (10221001)

A Thesis

Submitted as the partial Fulfillment for the Degree of

Bachelor of Science in Electrical and Electronic Engineering

Department of Electrical and Electronic Engineering

BRAC University

Dhaka-1212, Bangladesh

CERTIFICATE OF APPROVAL

The thesis entitled **"Comparison of the performance of ballistic schottky barrier Graphene Nanoribbon FET "** submitted by **Sheikh Ziauddin Ahmed, Mashiyat Sumaiya Shawkat** and **Md. Iramul Hoque Chowdhury** has been accepted satisfactorily in partial fulfillment of the requirement for the degree of Bachelor of Science in Electrical and Electronic Engineering on September, 2014.

Supervisor

(Dr. Sharif Mohammad Mominuzzaman)
Department of Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology (BUET)

CANDIDATE DECLARATION

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Author

Sheikh Ziauddin Ahmed

Author

Mashiyat Sumaiya Shawkat

Author

Md. Iramul Hoque Chowdhury

ACKNOWLEDGEMENT

We are grateful to Almighty ALLAH for directing us on the right path in pursuing the requirement of the full thesis work.

There are many people we would like to thank who have helped us in completing our research. First of all we would like to express our gratitude towards our respected thesis supervisor, **Dr. Sharif Mohammad Mominuzzaman,** Professor of EEE department of BUET, for his insight and direction. His faith in us and motivation has provided us with the confidence to achieve our goals.

We are indebted to **Sabbir Ahmed Khan, Mahmudul Hasan and Nirjhor Tahmidur Rouf** for referring our group to Professor Mominuzzaman and for sharing their experience, ideas and at the same time stimulating our intellectual curiosity. Also, a special thanks to **Akib Mahmud** for taking the time out of his busy schedule to help us out with our work. They have been extremely helpful and have guided us in times of difficulties.

Lastly, we thank our parents and well-wishers for supporting and encouraging us throughout our work.

ABSTRACT

Silicon has been the primary material of choice to meet current needs of the electronics industry. The advancement in technology has led to a growing demand for smaller devices with improved performance. However, silicon as a material has its own limitations; silicon based integrated circuits and the scaling of silicon MOSFET design faces complications like tunneling effect, gate oxide thickness effect etc. To overcome these problems new materials with improved characteristics are needed.

In recent times, graphene and carbon nanotube have shown huge promise as materials that can replace silicon-based materials in the future due to their outstanding electrical properties and other characteristics. Simulation studies of graphene nanoribbon field-effect transistors (GNRFETs) and carbon nanotube field-effect transistors (CNTFETs) are presented in this research paper using models that have been systematically developed and are of increasing rigor and versatility. This thesis covers the studies and modeling of graphene nanoribbon and carbon nanotube, which includes band structures and current-voltage graphical plots. Also, an analysis has been presented which shows the effect of varying temperature, relative dielectric constant, chirality, channel length and gate oxide thickness, on the device performance, in particular on the drain current.

The main purpose of this paper is to the study behaviour of schottky barrier graphene nanoribbon transistors and carbon nanotube transistors. The focus here is on the transfer and output characteristics of these transistors and observing the parameter changing effects on them. The simulation study is carried out using NanoTCAD ViDES program and the results obtained are used to make a comparative analysis of the device performance of GNRFET and CNTFET. Also, the simulation results obtained in this paper are compared with the simulation results of other research groups to verify our results.

Table of Contents

Pages

Certificate of Approval 2

Candidate Declaration 3

Acknowledgement 4

Abstract 5

Table of Contents 6

List of Tables 9

List of Figures 10

Chapter One

Introduction 12

1.1 Background and Research Motivation 12

1.2 Research Objectives 14

1.3 Scope of the Research 15

1.4 Outline of the Research Report 15

Chapter Two

2.1 Conventional SiMOSFET 17

2.1.1 Scaling of the Silicon MOSFET 19

2.1.2 Limitations of Scaling 19

2.1.2.1 Short Channel Effect 20

2.1.2.2 Tunneling Limit 21

2.1.2.3 Oxide Thickness 21

2.1.2.4 Threshold Voltage Effect 22

2.1.2.5 Theoretical Limitations 23

2.1.2.6 Power Consumption and Heat Dissipation 23

2.1.2.7 Design Limitation 24

2.2 Introduction to Carbon Nanotube 24
2.2.1 Physical structure of Carbon Nanotube 26
2.2.2 SWNT Characteristics of Electrical Transport 28
2.3 Carbon Nanotube field effect Transistor 30
2.3.1 Structure of CNTFET 30
2.3.2 Back Gate CNTFET 32
2.3.3 Top Gate CNTFET 32
2.3.4 Schottky-barrier (SB) CNTFET 33
2.3.5 MOSFET-like CNTFET 35
2.3.6 Vertical CNTFET (V-CNTFET) 36
2.4 Introduction to Graphene 37
2.4.1 Synthesis of Graphene 39
2.4.1.1 Exfoliation 39
2.4.1.1.1 Mechanical Cleavage 39
2.4.1.1.2 Solution and Chemical Exfoliation 40
2.4.1.1.3 Oxidation and Reduction 40
2.4.1.2 Chemical Vapor Deposition 41
2.4.1.3 Chemical Synthesis 42
2.4.2 Properties of Graphene 42
2.4.2.1 Structure 42
2.4.2.2 Electronic 42
2.4.2.2.1 Electronic spectrum 42
2.4.2.2.2 Dispersion Relation 43
2.4.2.2.3 Single-atom wave propagation 44
2.4.2.2.4 Electronic Transport 44
2.4.2.3 Thermal 46
2.4.3 Energy Bandstructure of Graphene 46
2.4.4 Band gap opening in Graphene devices 47
2.4.4.1 Graphene nanoribbons 47
2.4.4.2 Bilayer graphene FETs and Tunnel FETs 48
2.4.4.3 Epitaxial graphene on SiC 48

2.4.4.4 Functionalized Graphene 49
2.5 Graphene Nanoribbon 49
2.5.1 GNR Structure 49
2.5.2 Production of Graphene nanoribbon 52
2.5.3 Electronic Structure of GNR 52
2.5.4 Graphene Transistors 53
2.6 Summary 57

Chapter Three
3.1 The Model 58
3.1.1 Model Physics and the Process of Calculation 58
3.2 Result and Analysis 62
3.2.1 Effect of Temperature 63
3.2.2 Effect of Relative Dielectric Constant 68
3.2.3 Effect of Chirality 73
3.2.4 Effect of Channel Length 80
3.2.5 Effect of Gate Oxide thickness 86
3.3 Summary 90

Chapter Four
4.1 Conclusion 92
4.2 Future work 91

References 95
Appendix A 107
Appendix B 109

List of Tables

Pages

3.1 Table of Parameters 60

List of Figures

Pages

Figure 1.1: (a) Moore's law and (b) IC technology projection 13
Figure 2.1: Structure of MOSFET 18
Figure 2.2: Feature size versus time in silicon ICs 19
Figure 2.3: Short-channel-transistor leakage current mechanisms 20
Figure 2.4: Potential barrier between two transistors 24
Figure 2.5: Single-wall Carbon nanotube 25
Figure 2.6: Multi Wall Nanotube 25
Figure 2.7: Graphene sheet and rolling graphene sheet to create carbon nanotube 26
Figure 2.8: 3D model of the three types of single walled carbon nanotubes 28
Figure 2.9: Early CNTFET structure 30
Figure 2.10: (a) Back gate CNTFET , (b) Top gate CNTFET 33
Figure 2.11: Diagram of a SB-CNTFET 34
Figure 2.12: MOSFET-like CNTFET 35
Figure 2.13: Structure of Vertical CNTFET 36
Figure 2.14: Unit cell of graphene 37
Figure 2.15: Structures made of graphene 38
Figure 2.16: Exfoliated graphene 40
Figure 2.17: Synthesis of graphene by oxidation and reduction. 41
Figure 2.18: E-k diagram of graphene 47
Figure 2.19: Energy gap as a function of the chiral number 48
Figure 2.20: Affect of the origin of the GNR chiral vector 50
Figure 2.21: Examples of GNR chiral and transport vectors 51
Figure 2.22: Structure and evolution of graphene MOSFETs 56
Figure 2.23: Direct-current behaviour of graphene MOSFETs with a large-area-graphene channel 56
Figure 3.1: Flow-chart of the self-consistent 3D Poisson-Schrodinger solver. 61
Figure 3.2: (a) Structure of GNRFET (b) Structure of CNTFET 62
Figure 3.3: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different temperatures at $V_D = 0.5$ V 64 65
Figure 3.4: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different temperatures at $V_G = 0.5$ V 66
Figure 3.5: Dependence of resistance on temperature of graphene nanoribbon
Figure 3.6: (a) Schematic of a FET based on GNR arrays patterned by Block Copolymer lithography and (b) the corresponding SEM image. In (b), the contrast difference in the channel between the GNR arrays and the bare silica is evident. (c)I_{DS} - V_{DS} curves of the GNR array FET with a 9 nm ribbon width recorded at different gate voltages. (d) *I* DS - *V* G curves of the GNR array FET with a 9 nm ribbon width recorded at *V* DS = 100 mV in the temperature range of 100–300 K 67
Figure 3.7: I_D vs. V_G characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET at T= 300K and $V_D = 0.5$ V 67
Figure 3.8: I_D vs. V_D characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET at T= 300K and $V_G = 0.5$ V 68
Figure 3.9: Dielectric constant changing effect investigated by Rasmita Sahoo *et al.* which satisfied simulation result 69
Figure 3.10: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) 70

Carbon Nanotube SBFET for different relative dielectric constant at $V_D = 0.5$ V

Figure 3.11: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different relative dielectric constant at $V_G = 0.5$ V — 71

Figure 3.12: I_D vs. V_G characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for relative dielectric constant of 7.9 at $V_D = 0.5$ V — 72

Figure 3.13: I_D vs. V_D characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for relative dielectric constant of 7.9 at $V_G = 0.5$ V — 72

Figure 3.14: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different chirality at $V_D = 0.5$ V — 75

Figure 3.15: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different chirality at $V_G = 0.5$ V. — 76

Figure 3.16: (a) Transfer characteristics of GNR SBFET (b) Output Characteristics of GNR SBFET for different chirality studied by Yijiang Ouyang *et al* — 77

Figure 3.17: (a) Scaling of nanotube diameter. ID vs. VG characteristics at VD=0.4V for the nominal CNTFET with different nanotube diameter. The solid line with circles is for (13,0) CNT (with d~1nm), the sold line is for (17,0) CNT (with d~1.3nm), and the dashed line is for (25,0) CNT (with d~2nm) studied by Guo *et al.* **(b)** I_D vs. V_G characteristics of the CNTFET simulated in this thesis paper for chirality (13, 0), (17, 0) and (25, 0) at $V_D = 0.4$V. — 78

Figure 3.18: I_D vs. V_G characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for chirality (5,0) at $V_D = 0.5$ V — 79

Figure 3.19: I_D vs. V_D characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for chirality (5,0) at $V_G = 0.5$ V — 79

Figure 3.20: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different channel lengths at $V_D = 0.5$ V — 82

Figure 3.21: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different channel lengths at $V_G = 0.5$ V — 83

Figure 3.22: (a) Transfer characteristics of GNR SBFET (b) Output Characteristics of GNR SBFET with single gate geometry for different channel lengths studied by Yijiang Ouyang *et al* — 84

Figure 3.23: (a) ON state drain current as a function of channel length of CNT MOSFET (b) ON state drain current as a function of channel length of CNT SBFET — 85

Figure 3.24: I_D vs. V_G characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for $L_{ch} = 5$nm at $V_D = 0.5$ V — 85

Figure 3.25: I_D vs. V_D characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for $L_{ch} = 5$nm at $V_G = 0.5$ V — 86

Figure 3.26: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different gate oxide thickness at $V_D = 0.5$ V — 87

Figure 3.27: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different gate oxide thickness at $V_G = 0.5$ V — 88

Figure 3.28: I_D vs. V_G characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for gate oxide thickness of 2nm at $V_D = 0.5$ V. — 89

Figure 3.29: I_D vs. V_D characteristics of Graphene Nanoribbon SBFET and Carbon Nanotube SBFET for gate oxide thickness of 2nm at $V_G = 0.5$ V — 89

Chapter 1

INTRODUCTION

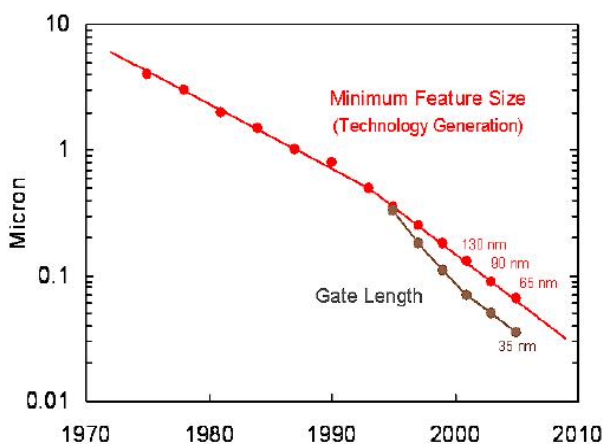
The intent of this paper is to provide a detailed discussion about ballistic graphene nanoribbon transistor and carbon nanotube transistor. This research also establishes a comparative analysis of the transfer and output characteristics of ballistic schottky barrier graphene nanoribbon transistor and ballistic schottky barrier carbon nanotube transistor. The analysis is carried out by changing different parameters on input and comparing the result with the result of other research groups. As an introduction this chapter presents the objective, background and the scope of this research work. This chapter also gives the outline of the thesis and as well as the summary of the content for each chapter.

1.1 Background and Research Motivation

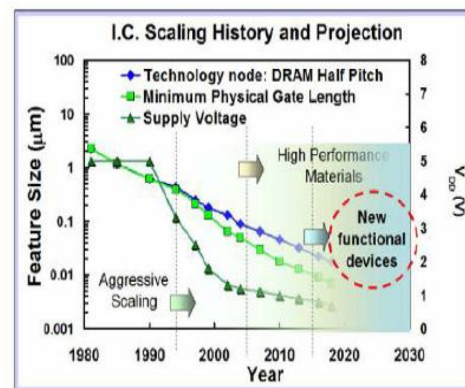
Silicon technology continues develop at a pace which outpaces the historic pace of Moore's Law [1]. However, the scaling limits of silicon are nearing the end since many problems arise as devices become smaller in size. Problems like tunneling effect, short-channel effect etc. come into the picture and these effects hinder the device performance. It is therefore vital that silicon be replaced by other materials which will take device advancement to a whole new level. Therefore, it is of intense interest to find new, molecular-scale devices that might complement a basic silicon platform by providing it with new capabilities - or that might even replace existing silicon technology and allow device scaling to continue to the atomic scale. As device sizes approach the nanoscale, new opportunities arise from harnessing the physical and chemical properties at the nanoscale. Chemical synthesis, self-assembly, and template self-assembly promise the precise fabrication of device structures or even the entire functional entity. Quantum phenomena and dimensional transport may lead to new functional devices with very different power/performance tradeoffs. New materials with novel electronic, optical, and mechanical properties emerge as a

result of the ability to manipulate matter on a nanoscale. It is now feasible to contemplate new nanoelectronic systems based on new devices with completely new system architectures, for examples: nanotubes, nanowires, molecular devices, and novel device concepts for nanoelectronics [1].

Of the various material systems and structures studied so far, graphene and carbon nanotubes have shown particular promise due to their nanoscale size and unique electronic properties. Due to their low dimensionality, nanostructures such as quantum dots, carbon nanotubes (CNTs) possess unique properties that make them promising candidates for future technology applications [2].Significant efforts have been devoted to understand how a graphene and carbon nanotube transistor operates and to improve the transistor performance [3,4,5]. Recently carbon nanotube field effect transistors (CNTFETs) and graphene nanoribbon field effect transistors (GNRFETs) have been fabricated successfully. It is reported that they have shown better performance than present silicon transistors of equivalent size.



(a)



(b)

Figure 1.1: (a) Moore’s law and (b) IC technology projection. [1]

In this paper, physical simulation of GNRFETs and CNTFETs is developed. Before carrying out the simulation, it is necessary to develop a fundamental understanding of the basic physics that governs their behavior in devices. Several researches have shown that the concepts learnt from bulk device physics do not simply carry over to nanotube devices which lead to unusual operation of a device [2]. That's why this research includes the basics characterization of carbon nanotube and graphene especially electrical transport characteristics of CNTs and GNRs.

1.2 Research Objectives

The scaling of silicon-based transistors has been the driving factor behind the large growth of the technology industry over the last few decades. However, this miniaturization imposes some limits on the silicon-based transistors. Thus, researchers have been motivated to explore and discover other alternative technology like graphene and carbon nanotubes for better functioning of the current devices. Graphene and carbon nanotubes are potential materials for future nanoelectronics, both as interconnects and as critical elements like channel materials for field-effect transistors because of their low dimensionality and outstanding electronic properties. Currently, ballistic graphene nanoribbon field-effect transistor(GNRFET) and carbon nanotube field-effect transistor (CNTFET) are treated as two of the nanoelectronic devices that have immense prospective to be treated as a switching device for future. We plan to make a detailed comparative analysis between these two types of transistors. The core objectives of the research work to summarize are:

- Understand the basic of graphene nanoribbon and carbon nanotube physics and focus on their electrical properties.
- Analyze the graphene nanoribbon and carbon nanotube device models and the limitation of Si MOSFET.
- Realize theoretical difference between graphene nanoribbon based FET and carbon nanotubes based FET.
- Understand the device characteristics, fundamental equation and mathematical model of GNRFET and CNTFET.

- Using mathematical model simulation investigate the I-V characteristics of GNRFET and CNTFET by varying different parameters and make an unalloyed comparison with different research group result.

- By examining the objective stated above, we can deduce total GNRFET and CNTFET characterization and form a complete understanding of the effect of changing different parameters on transfer and output characteristics of these two transistors.

1.3 Scopes of Work

The research paper has been limited to the following scopes of work due to lack of resources, expertise and restricted time frame.

- Using NanoTCAD ViDES [154] to simulate schottky barrier Graphene Nanoribbon field-effect transistor (GNRFET) and Carbon Nanotube field-effect transistor (CNTFET) and generate I-V curves.
- Simulate the transfer and output characteristics by changing different parameters (like gate oxide thickness, chirality (leads to diameter), temperature, dielectric constant, gate drain voltage and channel length) and collecting data.
- Comparing the obtained results from the simulation with those of other research groups' data and observe the deviation in the data.
- Comparison between GNRFET and CNTFET.

1.4 Outline of the Research Report

This thesis paper has been divided into four chapters including this one. This chapter discusses the background and research motivation of this work. Also, the research objectives and the scopes of work of this paper are given in this chapter.

In Chapter 2, an elaborate overview of Silicon MOSFETs is given along with the limitation it faces due to scaling. Then there is a detailed discussion on Carbon Nanotube. Under this discussion the

structure of CNT, chirality, single walled CNT (SWCNT), multi walled CNT (MWCNT) and properties of CNT are discussed. Next the working principles of carbon nanotube transistors are presented. Lastly, this chapter gives an elaborate discussion on graphene and graphene nanoribon, their synthesis procedures and their properties.

Chapter 3 contains the results and analysis of our main work where we generated the I-V curves of both schottky barrier GNRFET and CNTFET. This chapter mainly deals with the transfer and output characteristics of both GNRFET and CNTFET. The result and analysis section displays and discusses the effects of changing temperature, relative dielectric constant, chirality, channel length and gate oxide thickness on the transfer and output characteristics.

Finally Chapter 4 is the conclusion of the whole projects and future proposal.

Chapter 2

MOSFET AND INTRODUCTION TO CARBON NANOTUBE, GRAPHENE AND GRAPHENE NANORIBBON

2.1 Conventional Si-MOSFET

In 1930, Lilienfeld [8] patented the basic concept of the field effect transistor (FET). After thirty years in 1959, the concept was finally materialized in Si-SiO_2 by Kahng and Atalla [9], [10]. The first MOSFET was invented in 1959 and since then it has completely changed the world of digital electronics. MOSFETs have dominated all fronts of digital applications especially modern computers; because it offers many advantages to the user. MOSFETs are relatively small in size and this contributes to the fact that they can be packed in large numbers on a single integrated circuit. It is also very reliable and offers low consumption of power. The progress up to now is well described by “Moore’s law.” Gordon Moore predicted in 1965 that for each new generation of memory chip and microprocessor unit on the market, the device size would reduce by 33 percent, the chip size would increase by 50 percent, and the number of components on a chip would quadruple every three years. So far this trend has shown no signs of stopping [11].

Several properties of silicon have made these developments in microelectronics possible. Silicon can be grown in single crystals that are more than 1 m long and 30 cm across. The purity of the crystal and the number of electrically active defects can be controlled. The number of atomic crystal defects in sub-micrometresized MOSFETs is now limited to individual centers that act as traps for electrons. Such traps may be identified, individually characterized, and counted, so that single-electron transistors are possible. The reason behind Silicon being the semiconductor of choice for

MOSFETs, is its native oxide. Silicon dioxide (SiO_2) is an almost perfect insulating material with a resistivity in excess of 1016 Vcm. The insulating films of SiO_2 grown on silicon are smooth and coherent with no holes, in a thickness ranges down to single atomic layers [11].

The metal–oxide–semiconductor field-effect transistor (MOSFET) is a transistor used for amplifying or switching electronic signals. Although the MOSFET is a four-terminal device with source (S), gate (G), drain (D), and body (B) terminals; [12] the body (or substrate) of the MOSFET often is connected to the source terminal, making it a three-terminal device like other field-effect transistors. The gate terminal is a metal electrode that controls the current flow from source to drain [7]. The gate voltage needs to be higher than the threshold voltage in order for the current to flow in MOSFET. The source terminal is usually grounded and the drain voltage applied is relatively very small. As the gate voltage rises above the threshold voltage; an inversion layer or channel is created. This causes electrons to flow from source to drain terminal and as a result of which the current flows from drain to source terminal. There is no current flow to gate terminal since there is an oxide barrier which acts as an insulator. Figure 2.1 shows the structure of MOSFET.

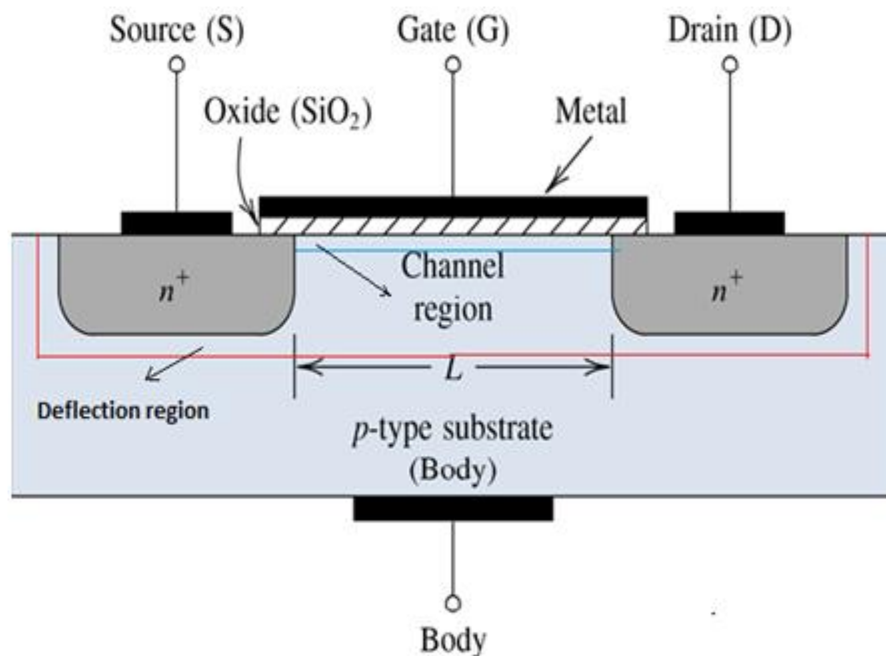


Figure 2.1: Structure of MOSFET [Source-internet image].

2.1.1 Scaling of the Silicon MOSFET

Scaling is a process which involves reducing the size of MOSFET and at the same time improving its performance. The first method was introduced in 1974 in which by reducing the MOSFET dimension, the device density, switching speed and energy was also improved. Each new generation has approximately doubled logic circuit density and increased performance by about 40% while the memory capacity has increased by four times. In ideal scaling, as the dimension and the operating voltage is reduced by a factor of 0.7, the area density doubles, switching delay decreases by a factor of 0.7 and the switching energy is halved. The switching speed can be estimated when the gate capacitance, operating voltage, and drive current are known. Switching energy is reduced as a result of the lower total combination parasitic capacitance due to smaller device size and lower operating voltage. Reduction of switching energy is very important since the overall circuit power is very crucial especially if the system is used for a long period continuously [7].

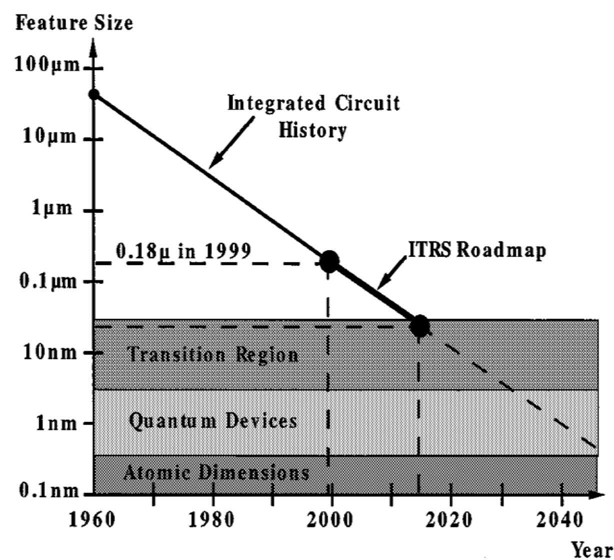


Figure 2.2: Feature size versus time in silicon ICs [13].

2.1.2 Limitations of Scaling

There have been many articles and papers on the current situation and future prospects for Si-MOSFETs; many different scaling limits for MOSFETs have also been discussed and proposed [10]. There are a number of factors which needs to be taken under consideration with continued MOSFET scaling that present challenges for the future and, ultimately, fundamental limits. There

are quite a few problems which arise as the MOSFET size reaches nanometer scale and ultimately limits the performance of the MOSFET itself. These problems are crucial and must be taken under consideration if the MOSFET is to survive in the near future.

2.1.2.1 Short Channel Effect:

The first factor to be considered is the short channel effect. The short channel effect introduces several leakage currents in MOSFET which are discussed below and shown in the figure- 2.3[14].

- Reverse bias p-n junction current occurs due to the minority carriers, diffusion near the depletion region and also due to the generation of electron-hole pairs.
- Weak reverse current occurs when gate voltage is lower than threshold voltage.
- DIBL current is present when source's potential barrier is reduced as a result of the drain's depletion region interacting with the source. The existence of DIBL lowers the threshold voltage.
- Gate-Induced Drain Lowering (GIDL) current occurs in high electric field between gate and drain, and it also occurs along the channel width between gate and drain.
- Another leakage current mechanism, punchthrough, occurs when the drain and source depletion regions touch deep in the channel.
- Narrow-width current arises when the channel length is reduced to less than 0.5µm.
- Gate-oxide tunneling current occurs when the oxide layer is made very thin and also causes gate leakage current tunneling through oxide bands.
- Hot-carrier injection occurs when hot carriers is injected into the oxide.

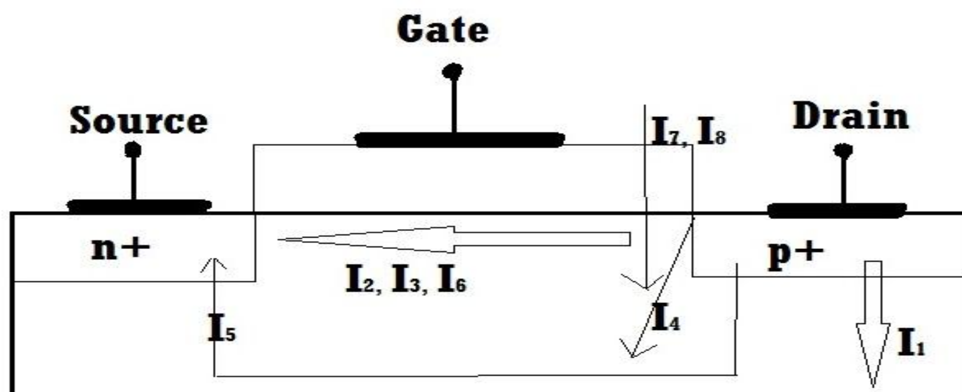


Figure 2.3: Short-channel-transistor leakage current mechanisms: reverse-bias p-n junction leakage (I_1), weak inversion (I_2), drain-induced barrier lowering (I_3), gate-induced drain leakage (I_4), punch-through (I_5), narrow-width effect (I_6), gate oxide tunneling (I_7), and hot-carrier injection (I_8) [14].

2.1.2.2 Tunneling Limit:

Normally in an operating or computational system integrated transistors are separated sufficiently enough so that operation of one transistor does not affect another transistor. The separation is made by inserting a material that acts as a barrier between two transistors. However, the barriers are also becoming small when MOSFETs are scaling down. So there is a possibility that carrier of one MOSFET crossing over another and making distortion of the performance. This effect increases exponentially as the barrier distance decrease [7].

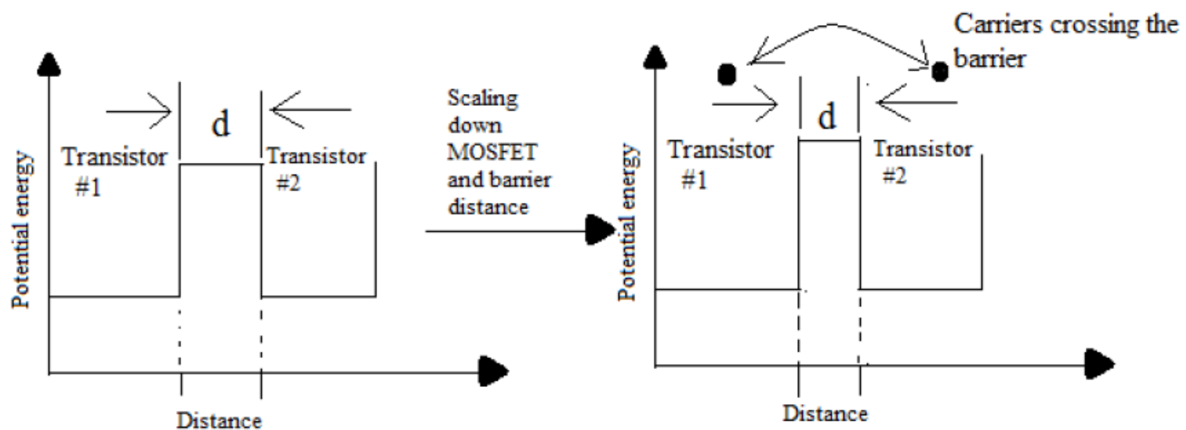


Figure 2.4: Potential barrier between two transistors [7].

2.1.2.3 Oxide Thickness:

The gate insulator in a MOSFET needs to be thin compared to the device channel length in order for the gate to exert dominant control over the channel potential. This avoids “short channel effects,” which are largely the result of the drain electric field penetrating throughout the channel and influencing the channel potential at the source of the device [13].

Gate-oxide thickness causes two kinds of limitations. Firstly, the thin layer of oxide eventually increases leakage current. This effect is also related to quantum effect tunneling that dominates in MOSFET as the oxide thickness is reduced. The tunneling current due to thick oxide layer may look negligible in comparison with “on state” current. However, it has a major effect when the chip is in

standby mode. Secondly, due to the oxide thickness there is a loss of inversion charge and also the transconductance as a result of inversion-layer quantization and polysilicon gate depletion effect [15].

The gate electrode itself also presents some significant challenges. Polysilicon has been used for more than 25 years as the gate electrode material. However, decreasing its resistivity, as shown in table-2.1, implies increasing the doping levels in the polysilicon, which minimizes the resistivity of the gate electrode. This aids in avoiding polysilicon depletion effects. However, this approach is limited by dopant solubility limits and by dopant out diffusion from the poly through to the thin gate dielectric and into the silicon. This later problem is particularly acute with p-gates because boron diffuses rapidly through SiO_2. The likely solution is again new materials. But there are no known materials solutions that are known to work in manufacturing [13].

2.1.2.4 Threshold Voltage Effect:

A notable limitation to MOSFET is that the threshold voltage is not proportionally decreasing with respect to transistor scaling. The threshold voltage held constant when the channel length is between 1µm-0.1µm and it deviates further when the channel length is below 0.1µm [6, 12]. If the transistor is scaled below 0.1µm, below the threshold voltage current does not drop to zero immediately but it decreases exponentially inversely proportional to the thermal energy [6]. There are some thermally distributed electrons at source terminal that have enough energy to overcome potential barrier controlled by gate terminal. This behavior is independent of channel length and power supply. So, higher threshold voltage causes higher leakage current. Denoting leakage current as I_{off} gives:

$$I_{off} = I_o \left(-\frac{qV_t}{mKT}\right) \quad (2.1)$$

I_o =Extrapolated current per width at threshold voltage.

m=Dimensionless ideality factor

V_t = Threshold voltage.

Lower leakage current is essential for a transistor due to reduce the power loss. However lower threshold voltage can reduce the leakage current. So, designing a transistor should be such a way so that its threshold voltage is very low. According to Sanudin, leakage current is reduced ten times for every 0.1V reduction of threshold voltage [7].

2.1.2.5 Theoretical Limitations:

Thermal limit and quanta limit are a major problem. Amount of energy needed to write a bit must be greater than the thermal function in order to avoid the bit error to occur. This is called the thermal limit. Currently CMOS needs 10-13 J to write a bit and the trend is to reduce it, in order for the power dissipation to reduce [7].

Quantum limit is associated with E/f where, E is the thermal energy and f is the frequency. Currently CMOS is operating higher than the quantum limit and if the scale reaches to 100nm then it is expected the limit is approached as E is decreased and f is increased.

2.1.2.6 Power Consumption and Heat Dissipation:

Power consumption and heat dissipation is one of the obstacles for further advancement in Si-based transistors. For the past three years power density has grown with the rate of 0.7 for every generation [16]. Large amount of power consumption boosts up the heat generation, increasing danger that transistors interfere with each other. As MOSFETs are scaling down so these small transistors are consume small amount of power but IC chip become denser because of large number of transistors on it. So it uses large amount of power to driven all transistors and therefore generates more heats. In November, 1971, Intel publicly introduced the world's first single chip microprocessor, the Intel 4004 with 2,300 Transistors at 10 µm, used tenths of watt while one of modern processor a 3.2 GHz Pentium IV extra edition consumes 135 watts [17]. Now in last few years increment in the number of transistors as 167 Million in dual core 2.8GHz Pentium D increased the power consumption to 244 watts.

Heat dissipation, Power consumption is major limitation with which traditional silicon-based MOSFET are suffering. Therefore, there is need of searching for new alternative medias, which can

overcome the limitations of conventional Si based MOSFET [18]. Here comes the idea of using carbon nanotube instead of silicon.

2.1.2.7 Design Limitation:

Scaling down MOSFET discover its limitation of current design. Present MOSFET does not work effectively when it is scaled only around 30nm. The limit is only because of the fact of Zener breakdown at source/substrate junction [7]. Leakage oat gate is also starts surface and it becomes very difficult to have a control over channel

2.2 Introduction to Carbon Nanotube

Carbon nanotubes, long, thin cylinders of carbon, were discovered in 1991 by S. Iijima. These are quasi-one-dimensional molecular structures and can be considered as a result of folding graphite (a hexagonal lattice of carbon) layers into cylinders. They may be composed of a single shell namely single wall nanotube (SWNT) or of several shells namely multi wall nanotube (MWNT). Carbon nanotubes have shown a surprising array of properties. They can conduct heat as efficiently as most diamond (only diamond grown by deposition from a vapour is better), conduct electricity as efficiently as copper, yet also be semiconducting (like the materials that make up the chips in our computers). They can produce streams of electrons very efficiently (field emission), which can be used to create light in displays for televisions or computers, or even in domestic lighting, and they can enhance the fluorescence of materials they are close to. Their electrical properties can be made to change in the presence of certain substances or as a result of mechanical stress. Perhaps the most exciting characteristics of carbon nanotubes are their unusual electronic properties. Carbon nanotubes can be metallic, semiconducting, or insulating depending on their length, diameter and rolling helicity, and do not requiring any doping. Again the energy gap of semiconducting carbon nanotubes can be varied continuously by varying the nanotube diameter. Here the band gap of semiconducting nanotubes decreases with increasing diameter. Individual carbon nanotubes are able to carry electrical current at significantly higher densities than most metals and semiconductors (maximum current density ~1013 A/m^2). Another important property is nanotubes are also inert and have no surface states, making them very compatible with other materials such as

oxides. These properties make carbon nanotubes a better choice than other molecular devices. Cutting edge research is focused on developing various devices from carbon nanotubes and on utilizing their unique properties in semiconductor technology for minimum possible feature sizes. Carbon nanotube field effect transistor is a novel outcome of this research. Next section will discuss detail on the carbon nanotube physical structure, electrical properties and with the explanation. It is strongly believed that Carbon Nanotube FET can provide better devices characteristics compared to the conventional MOSFET [1, 19-21].

As it is said before that there are two types of carbon nanotube which are single wall carbon nanotube and multiple wall carbon nanotube are shown in figure 2.5 and 2.6.

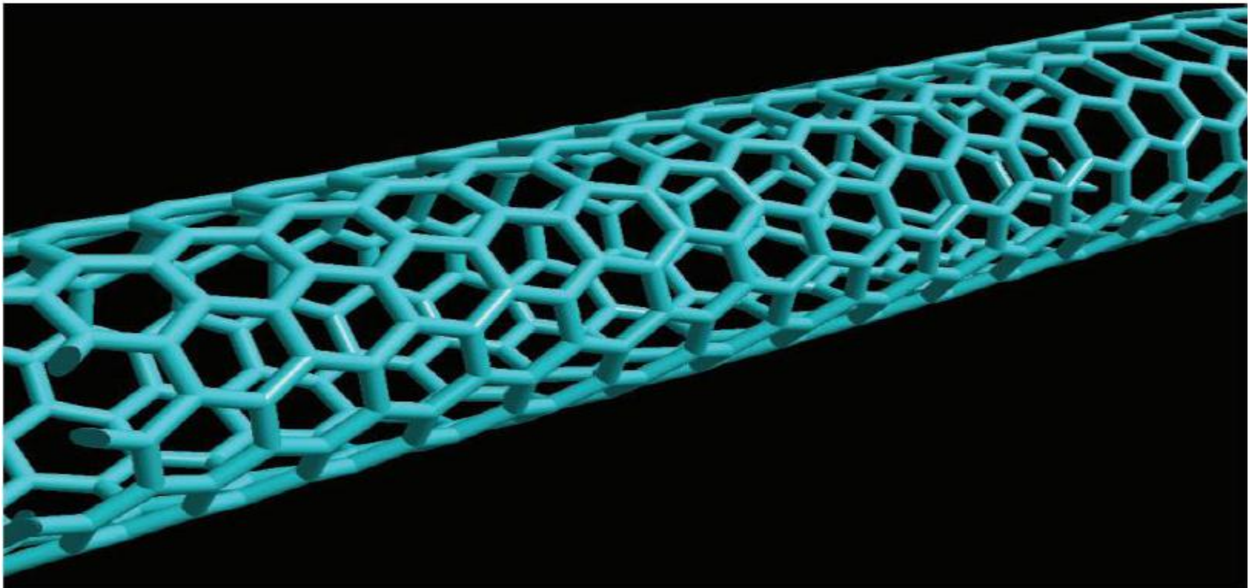


Figure 2.5: Single-wall Carbon nanotube [19].

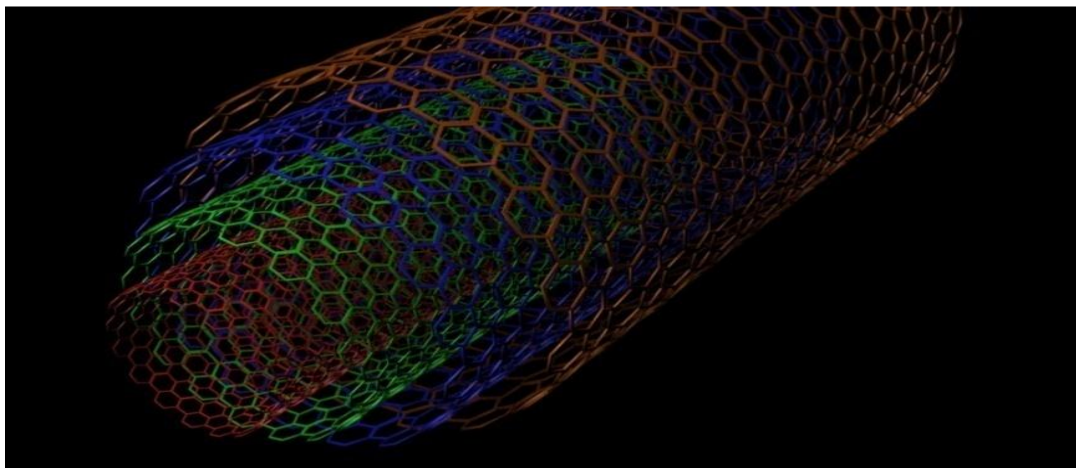


Figure 2.6: Multi Wall Nanotube (MWNT) [Internet Image].

2.2.1 Physical structure of Carbon Nanotube:

SWNTs are more pliable than their multi-walled counterparts and can be twisted, flattened and bent into small circles or around sharp bends without breaking. They can be conducting, like metal (such nanotubes are often referred to as metallic nanotubes), or semiconducting, which means that the flow of current through them can be stepped up or down by varying an electrical field. On the other hand Multi-walled carbon nanotubes are basically like Russian dolls made out of SWNTs—concentric cylindrical graphitic tubes. In these more complex structures, the different SWNTs that form the MWNT may have quite different structures (length and chirality). MWNTs are typically 100 times longer than they are wide and have outer diameters mostly in the tens of nanometers. Although it is easier to produce significant quantities of MWNTs than SWNTs, their structures are less well understood than single-wall nanotubes because of their greater complexity and variety. Multitudes of exotic shapes and arrangements, often with imaginative names such as bamboo-trunks, sea urchins, necklaces or coils, have also been observed under different processing conditions. The variety of forms may be interesting but also has a negative side—MWNTs always (so far) have more defects than SWNTs and these diminish their desirable properties [21, 22].

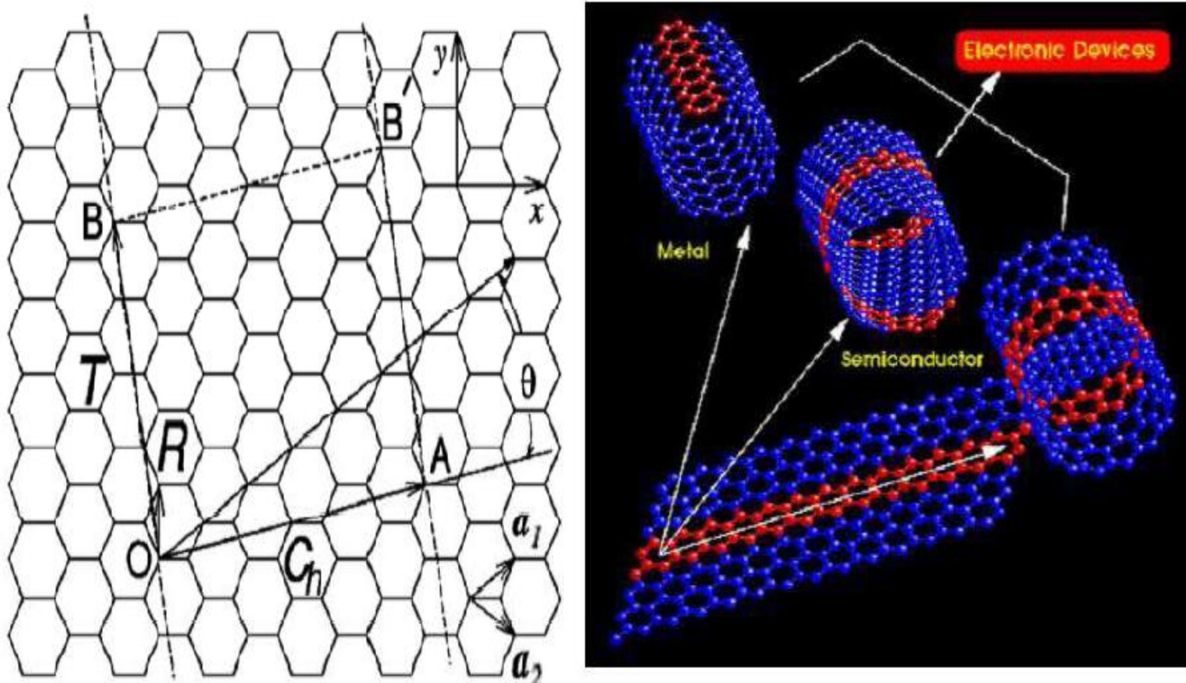


Figure 2.7: Graphene sheet [24] and rolling graphene sheet to create carbon nanotube [21].

A SWNT is described as a graphene sheet rolled up into a cylindrical shape with axial symmetry, exhibiting a spiral conformation called chirality [23]. Graphene has a hexagonal structure, and rolling up the graphene sheet in different directions and diameter would yield the nanotubes with different symmetries, which induces different electronic structures. Since electronic properties of SWNTs depend on their structures, it is very important to find a way to specify the geometric structure of a SWNT. As shown in Fig. 2.7, we can roll up the graphene sheet alone vector OA, which is perpendicular to the nanotube axis in the direction of OB. Here, we can see that O, A, B and B' are four crystallographically equivalent sites. By rolling up the paper plane and making OB overlap with AB', we get a seamless single-walled tubular structure. Then it would be straightforward to define the vectors $C_h = OA$ as chiral vector and $T = OB$ as translational vector. If we use a1 and a2 as the base vectors of graphene 2-D crystal lattice, we can have the chiral vector as [14]:

$$C_h = na_1 + ma_2 = (n, m) \qquad (2.2)$$

$0 \leq m \leq n$.

The way the graphene sheet wraps can be represented by a pair of indices (n, m) called the chiral vector. The relationship between n and m defines three categories of CNTs. Arm chair (n = m) and chiral angle equal to 30°); zigzag (n = 0 or m = 0 and chiral angle equal to 0°); and chiral (other values of *n* and *m* and chiral angles lie between 0 and 30°) [20, 23-28]. These are shown in figure 2.8

$$C_h = a/\sqrt{n^2 + m^2 + n} \qquad (2.3)$$

Where, a = 2.49 Å.

$$d_t\,(\text{diameter}) = C_h / \pi \qquad (2.4)$$

$$\theta\,(\text{chiral angle}) = \arccos\,(2n + m/2\sqrt{n^2 + m^2 + nm}) \qquad (2.5)$$

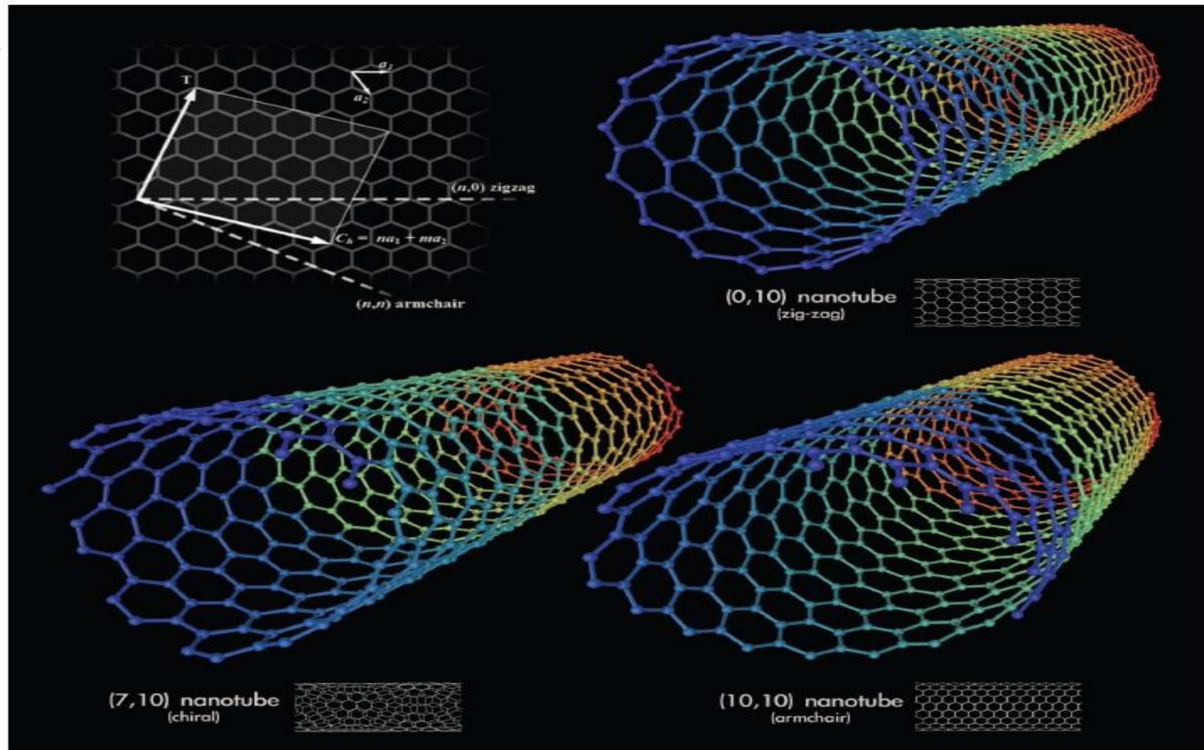


Figure 2.8: 3D model of the three types of single walled carbon nanotubes [20].

2.2.2 SWNT Characteristics of Electrical Transport:

A determination of the band structure allows for the calculation of an energy dependent Drude conductivity for the graphene sheet that constitutes a nanotube surface, as $\left(\frac{2e^2}{h}\right)\frac{E}{\hbar v_F} l_e$. Here, the elastic scattering length (l_e) of the carriers is proportional to the electron-phonon scattering and generally increases with decreasing temperature .One characterize the electrical conductivity in two regimes:

(1) Low temperatures (kBT<EF), where in the conductivity equation above, the energy (E) replaced by EF (the Fermi energy).The conductivity in this regime is metallic. A finite zero-temperature value, the magnitude of which is determined by the static disorder, is obtained.
(2) High temperatures (kBT>EF), where in the conductivity equation, the energy (E) is replaced by kBT. The conductivity, and the carrier density, is then directly proportional to T.

At the very outset, it is not trivial to measure the intrinsic resistance of a SWNT. Any contact in addition to those at the two ends of the tube can destroy the one-dimensional nature of the SWNT and make a true interpretation difficult. Theoretically, for a strictly one-dimensional system the Landauerm formula predicts an intrinsic resistance, independent of the length is equal to $h/e^2(1/T(E_f))$. Assuming perfect transmission through ideal Ohmic contacts, i.e., $T(E_F)$ equal to one. This contact resistance arises from an intrinsic mismatch between the external contacts to the wire (which are of higher dimensionality) and the one-dimensional nanotube system and is always present. When one takes individually into account both the two-fold spin and band degeneracy of a nanotube the intrinsic resistance (R_{int}) now becomes: $(R_{int})= h/4e^2(1/T(E_f))$, which again seems length independent [29] [30].

However, in the above discussion, we have not yet considered the contribution of the external contacts. When we consider the transmission (T) through the contacts into the one dimensional channel and then to the next contact, $T=l_e/l_e+L$, where l_e is the mean free path length for scattering and L is the length of the one-dimensional conductor. The resistance is now equal to:

$$\frac{h}{4e^2}\frac{l_e+L}{l_e} \equiv \frac{h}{4e^2}\left(1+\frac{l_e}{L}\right) \quad (2.6)$$

The first term represents *Rint* while the second term denotes an Ohmic resistance *(ROhmic)* associated with scattering. In the presence of dynamically scattering impurities, such as acoustical or optical phonons, which are inevitably present at any temperature above 0 K, the Ohmic resistance should definitely be considered. It is interesting to consider the limiting cases of a large mean free path ($l_e \rightarrow$ infinity) or a small tube ($L \rightarrow 0$) i.e., in the ballistic regime, when the Ohmic resistance is seen to vanish. Finally, the material resistance of the contacts contributes an additional term: Rc. The total resistance as measured in an external circuit would now be: $R = Rint + ROhmic + Rc$. These considerations imply that a minimum resistance of $h/4e^2$ ($\sim$6.5 kohm) is present in a SWNT with a single channel of conduction. In practice however, imperfect contacts (which lead to $T<1$) and the presence of impurities lead to larger resistance values, while deviations from strict one-dimensionality or multiple channels of conduction (as in a MWNT) could lead to smaller numbers for the resistance[29].

2.3 Carbon Nanotube field effect Transistor:

2.3.1 Structure of CNTFET:

The first carbon nanotube field-effect transistors were reported in 1998. These were simple devices fabricated by depositing single-wall CNTs (synthesized by laser ablation) from solution onto oxidized Si wafers which had been pre-pattemed with gold or platinum electrodes. The electrodes served as source and drain, connected via the nanotube channel, and the doped Si substrate served as the gate. A schematic of such a device is shown in Fig. 2.9 Clear p-type transistor action was observed, with gate voltage modulation of the drain current over several orders of magnitude. The devices displayed high on-state resistance of several MΩ, low transconductance (-I_{ns}) and no current saturation, and they required high gate voltages (several volts) to turn them on [1] [7].

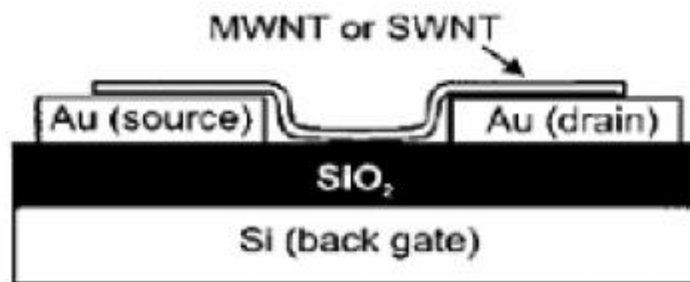


Figure 2.9: Early CNTFET structure [7].

Following these initial CNTFET results advances in CNTFET device structures and processing yielded improvements in their electrical characteristics. Rather than laying the nanotube down upon the source and drain electrodes, relying on weak van der Waals forces for contact, the electrodes were patterned on top of previously laid down CNs [2]. In addition to Au, Ti and CO were used, with a thermal annealing step to improve the metal/nanotube contact. In the case of Ti, the thermal processing leads to the formation of TiC at the metal/nanotube interface, resulting in a significant reduction in the contact resistance - from several MΩ to – 30 kΩ. On-state currents ~1 μA were measured, with transconductance - 0.3 μS. All early CNTFET were p-type, i.e., hole conductors.

Whether this was due to contact doping or doping by the adsorption of oxygen from the atmosphere was initially unclear. N-type conduction was achieved by doping from an alkali (electron donor) gas and by thermal annealing in vacuum. Doping by exposure to an alkali gas involves charge transfer within the bulk of the nanotube, analogous to doping in conventional semiconductor materials [30]. On the other hand, annealing a CNTFET in vacuum promotes electron conduction via a completely different mechanism: the presence of atmospheric oxygen near the metal/nanotube contacts affects the local bending of the conduction and valence bands in the nanotube by way of charge transfer, and the Fermi level is pinned close to the valence band, making it easier for injection of holes. When the oxygen is desorbed at high temperatures, the Fermi level may line up closer to the conduction band, allowing injection of electrons. Contrary to the case of bulk doping, there is no threshold voltage shift when going from p-type to n-type by thermal annealing. In addition, it is possible to achieve an intermediate state, in which both electron and hole injection are allowed, resulting in ambipolar conduction. The ability to make both p-type and n-type CNTFETs enabled the first carbon nanotube CMOS circuits. These were demonstrated by Derycke et al., who built simple CMOS logic gates, including an inverter in which the two CNTFETs were fabricated using a single carbon nanotube. Subsequently, more complex CN-based circuits have been built as well [1]. Carbon nanotube field effect transistor (CNTFETs) uses semiconducting carbon nanotube as the channel. Both p-channel and n-channel devices can be made from nanotubes. The physical structure of CNTFETs is very similar to that of MOSFETs and their I-V characteristics and transfer characteristics are also very promising and they suggest that CNTFETs have the potential to be a successful replacement of MOSFETs in nanoscale electronics. Of course, there are some distinct properties of CNTFETs, such as:

- The carbon nanotube is one-dimensional, which greatly reduces the scattering probability. As a result the device may operate in ballistic regime.
- The nanotube conducts essentially on its surface where all the chemical bonds are saturated and stable. In other words, there are no dangling bonds which form interface states. Therefore, there is no need for careful passivation of the interface between the nanotube channel and the gate dielectric, i.e. there is no equivalent of the silicon/silicon dioxide interface.

- The Schottky barrier at the metal-nanotube contact is the active switching element in an intrinsic nanotube device.

Because of these unique features CNTFET becomes a device of special interest. The field effect transistors made of carbon nanotubes so far can be classified into two types:

a) Back gate CNTFET
b) Top gate CNTFET

2.3.2 Back Gate CNTFET:

CNTFET was first demonstrated in 1998 by Tans et al. [31] to show a technologically exploitable switching behavior and this work marked the inception of CNTFET research progress. In this structure a single SWNT was the bridge between two noble metal electrodes on an oxidized silicon wafer. The silicon oxide substrate can be used as the gate oxide and adding a metal contact on the back makes the semiconducting CNT gateable. Here the SWCNT plays the role of channel and the metal electrodes act as source and drain. The heavily doped silicon wafer itself behaves as the back gate. These CNTFETs behaved as p-type FETs with an I (on) / I (off) ratio~10^5 [32]. This suffers from some of the limitations like high parasitic contact resistance (≥1Mohm), low drive currents (a few nanoamperes), and low transconductance gm ≈ 1nS. To reduce these limitations the next generation CNTFET developed which is known as top gate CNTFET.

2.3.3 Top Gate CNTFET:

To get better performance Wind et al. proposed the first top gate CNTFET in 2003 [32]. In the first step, single-walled carbon nanotubes are solution deposited onto a silicon oxide substrate. Then by using either atomic force microscope or scanning electron microscope the individual nanotubes are located. After which, source and drain contacts are defined and then patterned using high resolution electron beam lithography. High temperature annealing reduces the contact resistance and also increases union between the contacts and CNT. A thin top-gate dielectric is then deposited on top of the nanotube, either via evaporation or atomic layer deposition. Finally, the top gate contact is deposited on the gate dielectric. Arrays of top-gated CNTFETs can be fabricated on the same Silicon wafer, since the gate contacts are electrically isolated from each other, unlike in the

back-gated case. Also, due to the thinness of the gate dielectric, a larger electric field can be generated with respect to the nanotube using a lower gate voltage. These advantages mean top-gated devices are generally favored over back-gated CNTFETs, regardless of their more complex fabrication process [33].

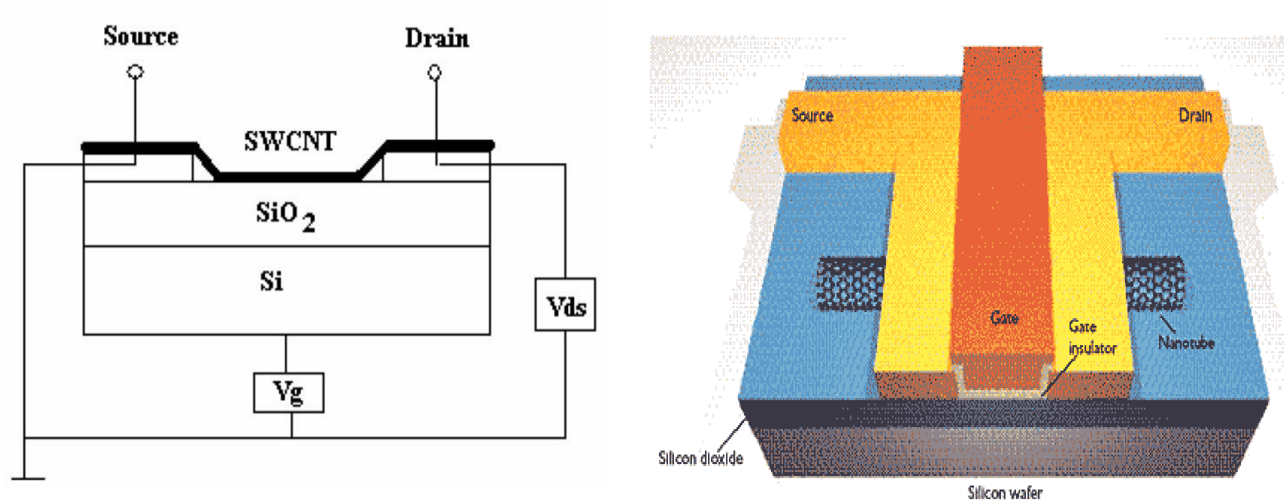


Figure 2.10: (a) Back gate CNTFET [23], **(b)** Top gate CNTFET [Source- Internet image].

2.3.4 Schottky-barrier (SB) CNTFET:

Normally, a potential barrier known as Schottky barrier (SB) exists at every contact between metal and semiconductor. The barrier height is determined by the filling of metal-induced gap states. These states become available in the energy gap of semiconductor due to interface formed with the metal. The SB is controlled by the difference of the local work functions of the metal and the carbon nanotube. SB is also extremely sensitive to changes of local environment at the contact [10]. For example, gas adsorption changes the work function of metal surfaces. Since this device employs metal as its source/drain terminals and has Schottky barrier at its terminal contact between nanotube and metal, therefore it is called Schottky-barrier CNTFET (SB-CNTFET). Diagram of SB-CNTFET is shown in Figure 2.11 below

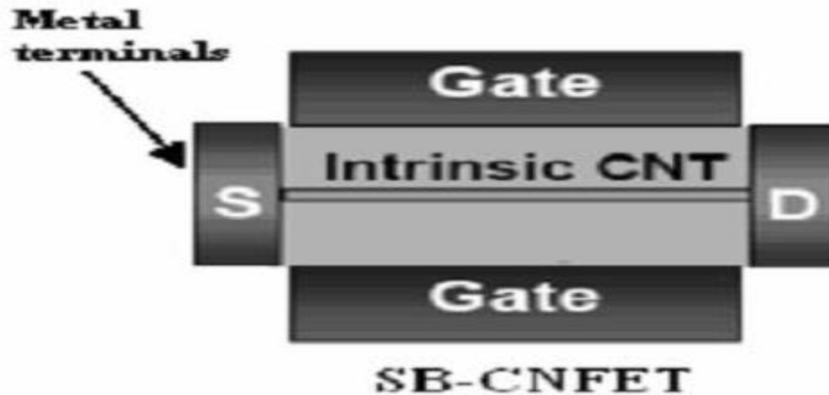


Figure 2.11 Diagram of a SB-CNTFET [7].

SB-CNTFET works on the principle of direct tunneling through the Schottky barrier at the source-channel junction. The barrier width is controlled by the gate voltage and hence the transconductance of the device depends on the gate voltage. At low gate bias, large barrier limits the current in the channel. As gate bias is increased, it reduces the barrier width, which increases quantum mechanical tunneling through the barrier, and therefore increases current flow in transistor channel. In SB-CNTFET, the transistor action occurs by modulating the transmission coefficient of the device [4, 10, 34-35].

SB-CNTFET shows very strong ambipolar conduction particularly when the gate oxide thickness is reduced even the Schottky barrier is zero [34]. This type of conduction causes leakage current to increase exponentially with supply voltage especially when the nanotube diameter is large, which results in limiting device potential. Thus, ambipolar conduction must be reduced in order to improve the performance of SB-CNTFET. One of the solutions is to increase the gate oxide thickness. If the gate oxide thickness is high, there is no ambipolar conduction exists when Schottky barrier is zero. Hence, the leakage current is reduced and as a result, the transistor performance is improved. Another alternative is to build asymmetric gate oxide, which is has been proposed recently, in order to suppress the ambipolar conduction [35, 10].

Another issue regarding on SB-CNTFET is that this type of transistor suffers from metal-induced-gap states which limit minimum channel length and thus increases source to drain tunneling. SB-CNTFET is also unable to place gate terminal close to source because it can increase parasitic capacitance.

2.3.5 MOSFET-like CNTFET:

The structure of this device is slightly dissimilar to SB-CNTFET since it uses heavily doped terminals instead of metal. This was formed in order to overcome problems in SB-CNTFET and operates like MOSFET. Unlike SB-CNTFET, source and drain terminals are heavily doped like MOSFET and hence it is called as MOSFET-like CNTFET. This device, as shown in Figure 2.12, operates on the principle of modulation the barrier height by gate voltage application. The drain current is controlled by number of charge that is induced in the channel by gate terminal.

This type of transistor has several advantages over SB-CNTFET. This device is able to suppress ambipolar conduction in SB-CNTFET. It also provides longer channel length limit because the density of metal-induced-gap-states is significantly reduced. Parasitic capacitance between gate and source terminal is greatly reduced and thus allows faster operation of the transistor. Faster operation can be achieved since length between gate and source/drain terminals can be separated by the length of source to drain, which reduces parasitic capacitance and transistor delay metric. It operates like SB-CNTFET with negative Schottky barrier height during on-state condition and thus it delivers higher on-current than SB-CNTFET. Previous work has shown that this type of device gives higher on-current compared to SB-CNTFET and therefore it can justify the upper limit of CNTFET performance. Based on the device performance, it is obvious that this device can be used to investigate the ballistic transport in CNTFET [36], [38].

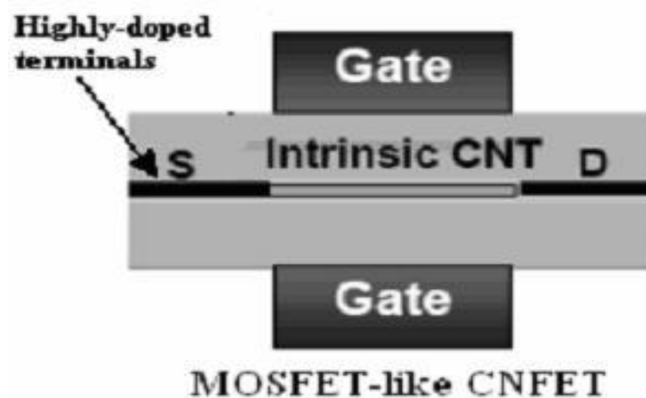


Figure 2.12: MOSFET-like CNTFET [37].

2.3.6 Vertical CNTFET (V-CNTFET):

The latest development in CNTFET progress could be the initiation of vertical CNTFET. This structure with surround-gated is suggested by Choi *et al.* in 2004 [7]. The transistor size can be as small as the diameter of carbon nanotube which corresponds to tera-level CNTFET and density of 10^{12} elements per cm^{-2}.The vertical CNTFET is prepared through the following steps: nano-pore formation by anodization followed by synthesizing the carbon nanotube, metal-electrode formation, oxide deposition and patterning and finally gate electrode formation. The silicon oxide was deposited at the top of aligned carbon nanotube by electron gun evaporation and followed by holes formation of e-beam patterning and chemical etching. The silicon oxide deposition process is then followed by deposition of top gate electrode. The structure of vertical CNTFET is illustrated in Figure 2.13. In this structure, each carbon nanotube is electrically attached to bottom electrode, source, upper electrode (drain) and gate electrode is put around the carbon nanotube. Each cross point of source and drain electrodes corresponds to a transistor element with a single vertical carbon nanotube. The number of carbon nanotube in transistor depends on the hole-diameter of gate oxide. The vertical CNTFET allows higher packing densities that can be achieved since source and drain areas can be arranged on top of each other [6]. On the other hand, real 3-D structures can be made possible because the active devices are no longer bound to the surface of mono-crystalline silicon wafer.

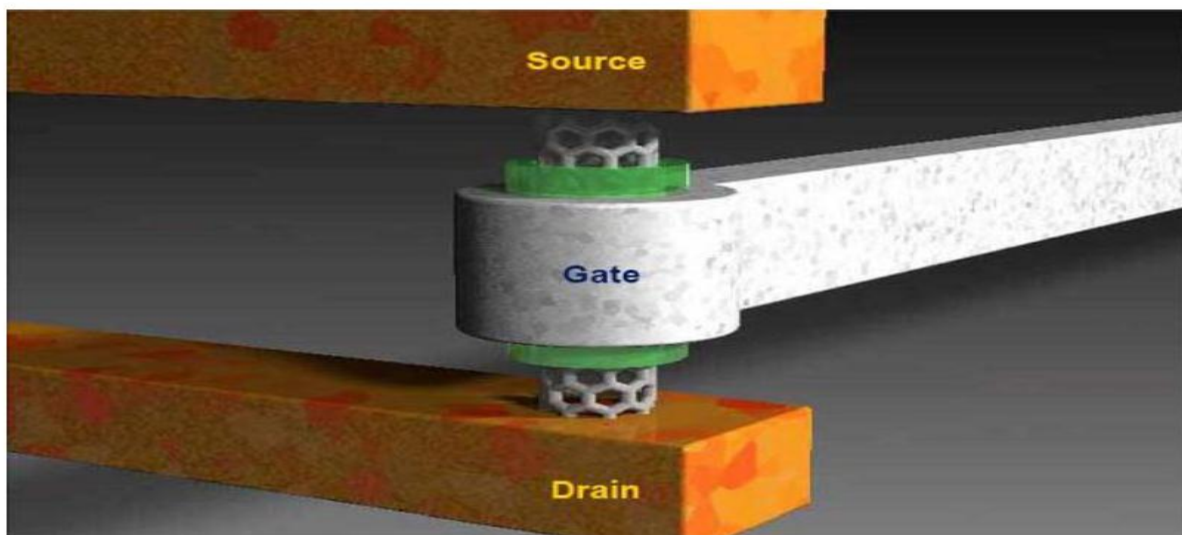


Figure 2.13: Structure of Vertical CNTFET [2].

2.4 Introduction to Graphene

Graphene is made of up pure carbon which takes the shape of extremely thin, almost transparent sheets, which are one atom thick. Despite being very light, graphene has extraordinary tensile strength, which is about 100 times higher than that of steel [39]. It also has remarkable electrical and thermal conductivity [40]. It was first isolated in a lab in 2004 [41].The aforementioned qualities of graphene are what make it a potential for making flexible conductors.

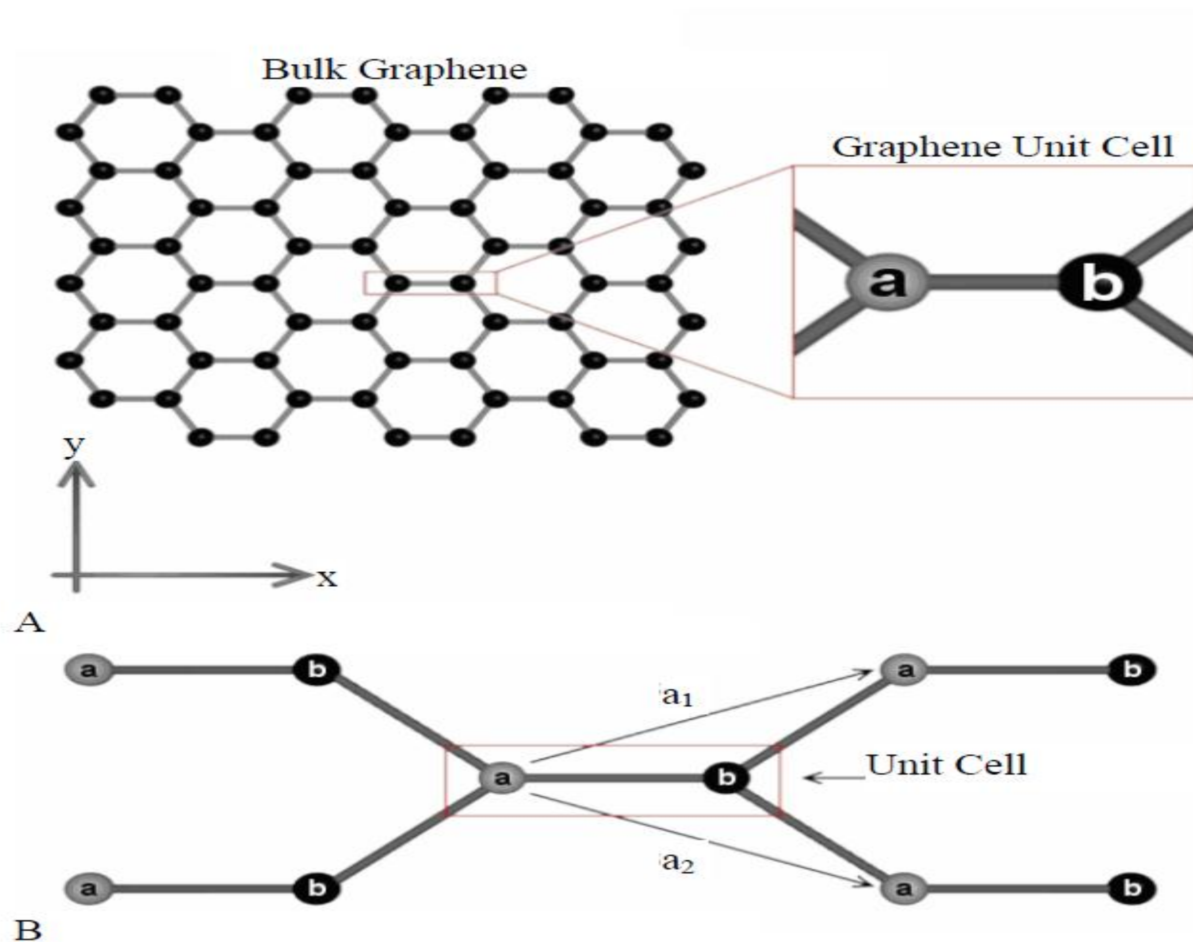


Figure 2.14:. Unit cell of graphene. A) Image of bulk graphene, with a unit cell show in the inset. B) Unit cell of graphene demonstrating its four nearest neighbors [132].

This crystalline allotrope of carbon, graphene, has 2-dimensional properties. It has a honeycomb lattice structure of carbon atoms in the sp^2 hybridization state. Each unit of cell of graphene lattice

consists of two carbon atoms that contribute one free electron each to the sea of electrons in graphene structure as shown in Fig 2.14 [42]. The carbon atoms in an atomic lattice are all tightly packed in a regular sp^2-bonded atomic-scale chicken wire (hexagonal) pattern. Graphene is simply put,a one-atom thick layer of graphite. It makes the basic structure for several other allotropes of carbon, including graphite, charcoal, carbon nanotube and fullerenes.

Although it was already known that graphite consists of hexagonal carbon sheets layered on top of each other, scientists were not still acquainted with the idea of isolating graphene sheet. Konstantin Novoselov, Andre Geim brought about a scientific revolution when they and their collaborators proposed that a single layer could be separated from graphite and that electrical characterization could be done on a few such layers. In July 2005 they published their electrical measurements on a single layer. This single layer is what introduced the scientific community to the concept of graphene [38]. Figure 2.15 shows three different structures made of honeycomb lattice.

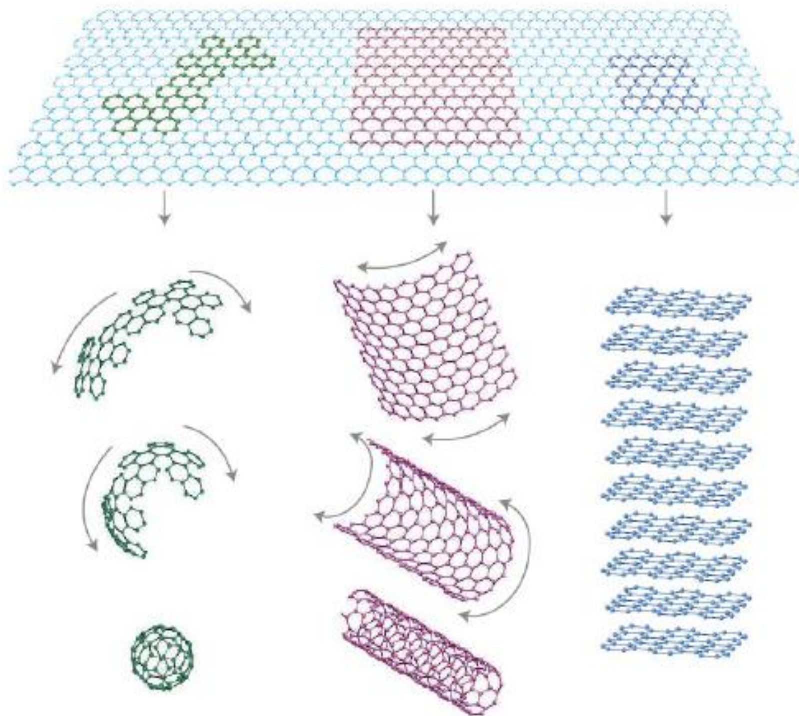


Figure 2.15: **Structures made of graphene -** fullerene molecules, carbon nanotubes, and graphite can all be thought of as being formed from graphene sheets, i.e. single layers of carbon atoms arranged in a honeycomb lattice [44].

2.4.1 Synthesis of Graphene

Research has been going on to perfect mass production techniques to generate high quality graphene for decades [45]. The structure of graphite is well known since the invention of x-ray diffraction crystallography. Solution based exfoliation of graphite (i.e, via oxidation or graphite intercalation compound (GICs)) gave a pre-mature idea about the atomic planes of carbon [46]. In the 1960s, Boehm suggested the concept of reducing exfoliated graphite oxide in order to achieve monolayers in solution [47]. There had been accounts of several successful attempts of producing monolayers of carbon in graphitic structures, on various carbides [48–50] and transition metal surfaces, [50-55] among which the most noteworthy is Van Bommel's work with SiC [56]. These studies did not equip the exploration of any electronic properties, since the strongly bound metallic surfaces disrupt the perpendicular pi-orbitals, with the exception of SiC. Synthesis techniques can be classified into micromechanical exfoliation, solution-based and chemically assisted exfoliation, chemical synthesis, epitaxial growth through sublimated SiC surfaces, and the pyrolysis of hydrocarbons on metal surfaces. Each has its own advantages, shortcomings, challenges and unique features in terms of quality, processability, scalability and cost.

2.4.1.1 Exfoliation

2.4.1.1.1 Mechanical Cleavage

Bulk graphite can be seperated into single atomic planes using this technique. It was impossible to observe isolated single layers of graphene, before the "Scotch tape method" was developed [57-59]. Micromechanical cleavage of bulk graphite has been commonly used to produce graphene samples of high quality which are also called peeled graphene. It makes use of adhesive tape to peel layers off highly oriented pyrolytic graphite, which is then pressed onto an appropriate substrate which is usually oxidized silicon. This method generates a very low output and mainly used for the study of fundamental properties such as ballistic transport, carrier mobility, thermal conductivity and so on. This method is not efficient enough to be used for practical applications even though it produces good quality graphene layer [59-65].

2.4.1.1.2 Solution and Chemical Exfoliation

This method has the potential of bulk-producing graphene. Bulk graphite is intercalated by inserting reactants between layers that weaken the cohesive van der Waals force [66]. Successful high-quality, single-layer graphene sheets, stably suspended in organic solvents were produced by Dai's group in steps of chemical intercalation, reintercalation, and sonication [67]. Expandable graphite is at first heated in sulfuric acid and nitric acid, where most of the exfoliated particles found are still multiple layers thick. This is followed by oleum treatment with tetra-butyl alcohol reintercalation for high- quality graphene. Sonification is done next in a surfactant solution based on AFM measurements. A liquid exfoliation approach was found to produce graphene by sonicating graphite powder in N-methylpyrroidone [68]. Low power sonication for weeks at a time to avoid the damage of the graphene sheets yields a high concentration (up to 1.2 mg/mL up to 4 wt%) of monolayer graphene [69]. Sonication-free, mild dissolution of graphite by synthesizing well-documented GICs achieved large graphene flakes and ribbons [70].

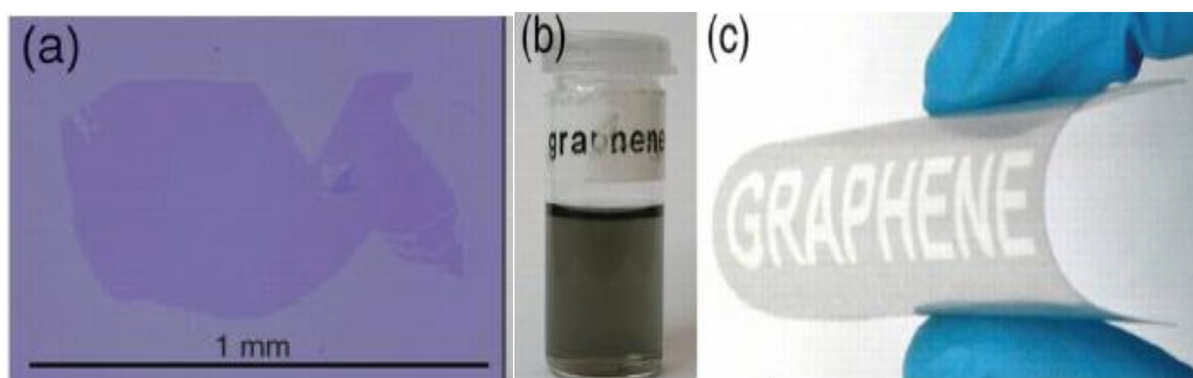


Figure 2.16: Exfoliated graphene. a) Optical microscopy image of a very large micromechanically exfoliated (tape method) monolayer of graphene. Note the considerable contrast for the single atomic layer. b) Photograph of dispersed graphene by ultrasonic exfoliation of graphite in chloroform and (c) that deposited on a bendable film [178].

2.4.1.1.3 Oxidation and Reduction

A more effective technique which gives high yields of graphene is to synthesize graphite oxide first and then exfoliate it into monolayers, followed by the removal of oxygen groups by reduction [71-72]. Each oxidized flake possesses a large number of negative charges that repel each another. The Brodie, Staudenmeier and Hummers methods are the three most common ways to oxidize graphite.

Among these, a slightly modified Hummers method has become the most prevalent in producing graphite oxide, for its relatively shorter reaction time and absence of toxic side products [72]. After oxidation, the interlayer spacing increases from 0.34 nm in graphite to above 0.6 nm, with weakened van der Waals forces between the layers. Exfoliation is typically augmented with sonication [73], yielding single layers of graphene oxide (GO) [74], which are soluble in water without the assistance of surfactants to form a stable colloidal system. The GO is then reduced using chemical [75], thermal [76], electrochemical [77] or electromagnetic flash [78], laser-scribe [79] techniques, etc.

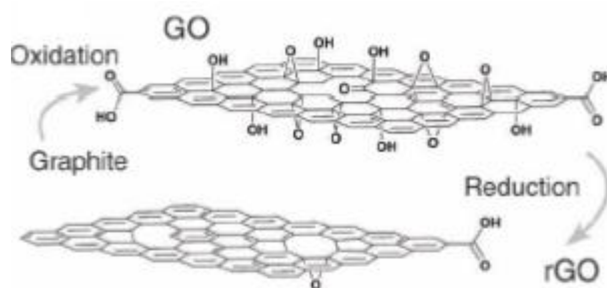


Figure 2.17: Synthesis of graphene by oxidation and reduction. Graphene oxide and reduced graphene oxide showing the remaining oxygen-rich functional groups after reduction [178].

2.4.1.2 Chemical Vapor Deposition

A very promising technique, Chemical vapor deposition (CVD) method, shows great potential for the large-scale production of single and multiple layer graphene films. Uniform, wafer size graphene films have been grown on both single crystal and polycrystalline transition metal surfaces at high temperatures by pyrolysis of hydrocarbon precursors such as methane [80-84].

The number of graphene layers is highly dependent on the carbon solubility of the substrate. For metals with relatively high carbon solubility, such as nickel [85-87], the carbon atoms can dissolve at high temperature, then, precipitate onto the metal surface and form single or multilayer graphitic films upon cooling. These non-uniform films with a wide thickness range from 1 to around 10 layers with monolayer domain sizes up to several tens of micrometers in diameter were produced on nickel substrates [88]. The cooling rate and hydrocarbon gas concentration can control the thickness and crystal ordering. On the other hand, low carbon solubility in certain transition metals, for instance copper [89] and platinum [90] enables complete monolayer coverage [91].

2.4.1.3 Chemical Synthesis

Controllable production of graphene can be achieved by another different path which is bottom-up organic synthesis. Graphene can be composed of interconnected polycyclic aromatic hydrocarbons (PAHs), which are very small two-dimensional graphene segments. This approach is attractive due to its high flexibility and compatibility with various organic synthesis techniques [92]. Müllen and coworkers are pioneers in this field, reporting synthesis of nanoribbon like PAHs with lengths over 30 nm [93-94]. Recently, the largest stable colloidal graphene quantum dots were synthesized using a benzene-based chemical route, which compose 132, 168, 170 conjugated carbon atoms. [95-96]. However, the size of the as-grown graphene dots is restricted due to decreasing solubility as sizes increase as well as an increasing number of possible side reactions, which is still a major hurdle for organic synthesis of graphene molecules with controllable shapes, sizes and edge structures.

2.4.2 Properties of Graphene

2.4.2.1 Structure

Graphene is a stable material due to a tightly packed, periodic array of carbon atoms and a sp^2 orbital hybridization - a combination of orbitals p_x and p_y that constitute the σ-bond. Graphene has three σ-bonds and one π-bond. The final p_z electron makes up the π-bond, and is key to the half-filled band that allows free-moving electrons [97]. The atomic structure of isolated, single-layer graphene was studied by transmission electron microscopy (TEM) on sheets of graphene suspended between bars of a metallic grid [98]. Electron diffraction patterns showed the expected honeycomb lattice.

2.4.2.2 Electronic

Graphene is a semi-metal or zero-gap semiconductor. It is separated from other condensed matter systems by four electronic properties.

2.4.2.2.1 Electronic spectrum

Electrons that propagate through graphene's honeycomb lattice structure effectively lose their mass and produce quasi-particles that are described by a 2D analogue of the Dirac equation rather than the Schrödinger equation for spin-1⁄2particles [99][100].

2.4.2.2.2 Dispersion Relation

By the use of a conventional tight-binding model the dispersion relation giving the energy of the electrons with wave vector *k* is[101][102]

$$E = \pm\sqrt{\gamma_0^2(1 + 4\cos^2\frac{k_y a}{2} + 4\cos\frac{k_y a}{2} \cdot \cos\frac{k_x\sqrt{3}a}{2}} \qquad (2.7)$$

With the nearest-neighbor hopping energy $\gamma_0 \approx 2.8$ eV and the lattice constant $a \approx 2.46$ Å. The conduction and valence bands, respectively, correspond to the different signs; they touch each other at six points, the "K-values" of the two-dimensional hexagonal Brillouin zone. Two of these six points are independent, while the rest are equivalent by symmetry. In the vicinity of the K-points the energy depends linearly on the wave vector, similar to a relativistic particle [101][103]. Since an elementary cell of the lattice has a basis of two atoms, the wave function has an effective 2-spinor structure.

As a consequence, at low energies, even neglecting the true spin, the electrons can be described by an equation that is formally equivalent to the massless Dirac equation. Hence, the electrons and holes are called Dirac fermions and the six corners are called the Dirac points.[98]This pseudo-relativistic description is restricted to the chiral limit, i.e., to vanishing rest mass M_0, which leads to interesting additional features [101][105]:

$$v_F\,\vec{\sigma}\cdot\nabla\psi(\mathbf{r}) = E\psi(\mathbf{r}). \qquad (2.8)$$

Here v_F ~ 106 m/s (.003 c) is the Fermi velocity in graphene, which replaces the velocity of light in the Dirac theory; $\vec{\sigma}$ is the vector of the Pauli matrices, $\psi(\boldsymbol{r})$ is the two-component wave function of the electrons, and *E* is their energy [99].

The equation describing the electrons' linear dispersion relation is

$$E = \hbar v_F\sqrt{k_x^2 + k_y^2}; \qquad (2.9)$$

where the wavevector *k* is measured from the Dirac points (the zero of energy is chosen here to coincide with the Dirac points). The equation uses a pseudospin matrix formula that describes two sublattices of the honeycomb lattice [103].

2.4.2.2.3 Single-atom wave propagation

Electron waves in graphene propagate within a single-atom layer, making them sensitive to the vicinity of other materials such as high-κ dielectrics, superconductors and ferromagnetics.

2.4.2.2.4 Electronic Transport

Experimental results obtained from transport measurements point out that at room temperature graphene has a remarkably high electron mobility, with reported values in excess of 15,000 $cm^2 \cdot V^{-1} \cdot s^{-1}$ [106].Furthermore, the symmetry of the experimentally measured conductance specifies that hole and electron mobilities is almost the same [100]. The mobility is nearly independent of temperature between 10 K and 100 K [107-109], which implies that the dominant scattering mechanism is defect scattering. Room temperature mobility is intrinsically limited to 200,000 $cm^2 \cdot V^{-1} \cdot s^{-1}$ at a carrier density of 1012 cm^{-2} by the scattering of the acoustic phonons of graphene intrinsically limits [109-110], which was later demonstrated and is greater than copper [111].

The corresponding resistivity of the graphene sheet should be in the order of 10^{-6} Ω·cm. This is less than the lowest known resistivity at room temperature which is that of silver [112]. However, for room temperature graphene on SiO_2 substrates, scattering of electrons by optical phonons of the substrate is a larger effect than scattering by graphene's own phonons. This limits mobility to 40,000 $cm^2 \cdot V^{-1} \cdot s^{-1}$[109].

In 40-nanometer-wide nanoribbons of epitaxial graphene the electrical resistance changes in discrete steps. The conductance of the ribbons exceeds predictions by a factor of 10. The ribbons can behave in a much more similar manner like the optical waveguides or quantum dots, allowing the smooth flow of electrons along the ribbon edges. In copper, the resistance increases in proportion to length as electrons bump into impurities [113- 114].

Transport is dominated by two modes. One being ballistic and temperature independent, while the other is thermally activated. Ballistic electrons resemble those in cylindrical carbon nanotubes. At

room temperature, there is an abrupt increase in resistance at a particular length—the ballistic mode at 16 micrometres and the other at 160 nanometres (1% of the former length) [113].

The ribbons were grown on the edges of three-dimensional structures etched into silicon carbide wafers. When the wafers are heated to about 1,000 °C (1,830 °F), silicon is preferentially driven off along the edges, forming nanoribbons whose structure is determined by the pattern of the three-dimensional surface. The nanoribbons had perfectly smooth edges, annealed by the fabrication process. Electron mobility measurements surpassing one million correspond to a sheet resistance of one ohm per square— two orders of magnitude lower than in two-dimensional graphene [113]. Graphene electrons can bridge micrometer distances without scattering, even at room temperature [99].

Despite zero carrier density near the Dirac points, graphene exhibits a minimum conductivity on the order of $4e^2/h$. The source of this minimum conductivity is still unclear. However, rippling of the graphene sheet or ionized impurities in the SiO_2 substrate may lead to local puddles of carriers that allow conduction [100]. Several theories suggest that the minimum conductivity should be $4e^2/(\pi h)$; however, most measurements are of order $4e^2/h$ or greater [106] and depend on impurity concentration [115].

Near zero carrier density graphene displays positive photoconductivity and negative photoconductivity at high carrier density. Theis governed by the interplay between photo induced changes of both the Drude weight and the carrier scattering rate [116].

Graphene doped with various gaseous species (both acceptors and donors) can be returned to an undoped state by gentle heating in vacuum [115][117]. Even for dopant concentrations in excess of 1012 cm^{-2} carrier mobility exhibits no observable change [114]. Graphene doped with potassium in ultra-high vacuum at low temperature can reduce mobility 20-fold [115][118]. The mobility reduction is reversible on heating the graphene to remove the potassium.

Due to graphene's two dimensions, charge fractionalization (where the apparent charge of individual pseudoparticles in low-dimensional systems is less than a single quantum[119]) is thought to occur. It may therefore be a suitable material for constructing quantum computers[120]using anyonic circuits [121].

2.4.2.3 Thermal

Graphene is a perfect thermal conductor. Its thermal conductivity was recently measured at room temperature and it is much higher than the value observed in all the other carbon structures such as carbon nanotubes, graphite and diamond (> 5000 $W \cdot m^{-1} \cdot K^{-1}$). The ballistic thermal conductance of graphene is isotropic, i.e. same in all directions. Graphite, the 3 D version of graphene, shows a thermal conductivity about 5 times smaller (1000 $W \cdot m^{-1} \cdot K^{-1}$). The phenomenon is governed by the presence of elastic waves propagating in the graphene lattice, called phonons. The study of thermal conductivity in graphene may have important implications in graphene-based electronic devices. Even on a substrate, thermal conductivity reaches 600 $W \cdot m^{-1} \cdot K^{-1}$ [111].

2.4.3 Energy Bandstructure of Graphene

The electronic structure of graphene is quite different from usual three-dimensional materials. Its Fermi surface is characterized by six double cones, as shown in Fig 2.18. In undoped graphene the Fermi level is situated at the connection points of these cones. Since the density of states of the material is zero at that point, the electrical conductivity of intrinsic graphene is actually quite low and is of the order of the conductance quantum $\sigma \sim e^2/h$; the exact prefactor is still argued. The Fermi level can however be changed by means of an electric field so that the material becomes either p-doped (with holes) or n-doped (with electrons) depending on the polarity of the applied field. Graphene can also be doped by the process of adsorption, for example, water or ammonia on its surface. The electrical conductivity for doped graphene is potentially quite high and it is possible that at room temperature it may even be higher than that of copper [153].

Close to the Fermi level the dispersion relation for electrons and holes is linear. Since the effective masses are given by the curvature of the energy bands, this corresponds to zero effective mass. The equation describing the excitations in graphene is formally identical to the Dirac equation for massless fermions which travel at a constant speed. The connection points of the cones are therefore called Dirac points [153].

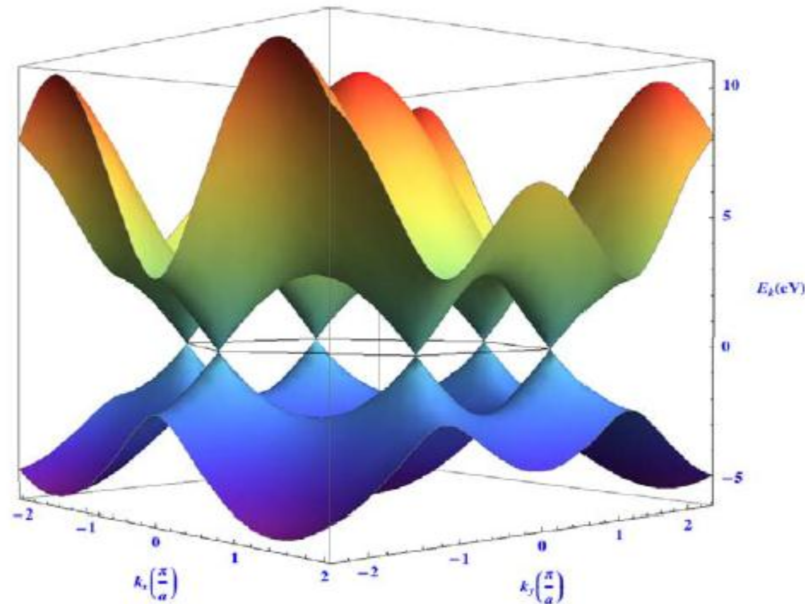


Figure 2.18: E-k diagram of graphene. The energy, E, for the excitations in graphene as a function of the wave numbers, kx and ky, in the x and y directions. The black line represents the Fermi energy for an undoped graphene crystal. Close to this Fermi level the energy spectrum is characterized by six double cones where the dispersion relation (energy versus momentum, ħk) is linear. This corresponds to massless excitations [153].

2.4.4 Band gap opening in Graphene devices

There are a number of different ways that can be considered for inducing a bandgap in graphene: *i)* lateral confinement, i.e., using graphene nanoribbons as material for FET channels, *ii)* the use of bilayer graphene, that has a gap tunable with a perpendicular electric field, *iii)* the use of epitaxial graphene on SiC, *iv)* graphene functionalization or doping. In the following of this section, we will discuss the potential of these options, evaluated through modeling. [123]

2.4.4.1 Graphene nanoribbons

Nanoribbons offer a very interesting advantage over carbon nanotubes: as can be seen in Fig. 2.19, by virtue of edge relaxation, all nanoribbons have a semiconducting gap [122].

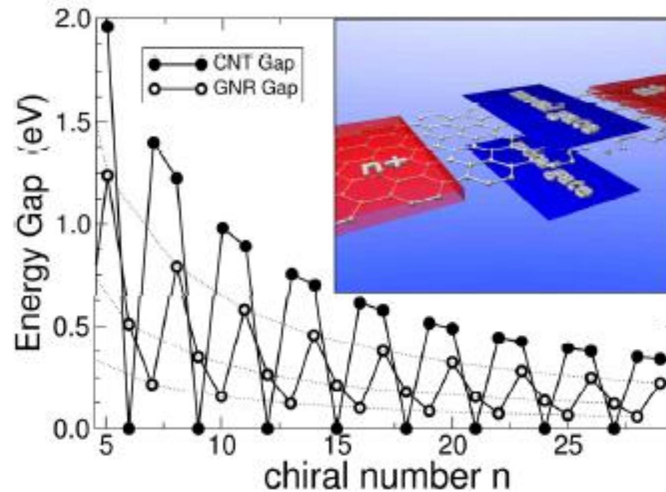


Figure 2.19: Energy gap as a function of the chiral number in zigzag carbon nanotubes (2n,0) [black] and in zigzag carbon nanoribbons (2n, 0) [white] [123].

2.4.4.2 Bilayer graphene FETs and Tunnel FETs

In bilayer graphene the bandgap can be altered by applying a vertical electric field, as has been predicted theoretically [124,125] and observed experimentally [126,127]. This fact unlocks a path to an interesting new prospect: one could devise a device in which the bandgap becomes large only when needed, i.e., when the device should be in the off state.

2.4.4.3 Epitaxial graphene on SiC

In recent experiments [128] it has been seen that a graphene layer grown by epitaxy on a SiC substrate can exhibit a gap of about 0.26 eV, measured by angle-resolved photo-emission spectroscopy. There is still need of further experimental confirmation and reproduction of results, but the point stated is very interesting, because epitaxial graphene on SiC is promising for wafer scale fabrication. The possibility of using the material as FET channel has been evaluated, exploring the design space with a semi-analytical model [129]. An I_{ON}/I_{OFF} ratio of up to 60 can be obtained for Vdd = 0.25 V, but the supply voltage (in V) cannot be larger than the channel bandgap (in eV), otherwise strong interband tunnelling results. Large current modulation is possible for smaller Vds, but again, if one consider digital applications, the applied Vds must equal the Vgs swing.

2.4.4.4 Functionalized Graphene

Chemical functionalization of graphene sheets or graphene nanoribbons is one of the many encouraging options for tuning the bandstructure and the electronic properties. Recent experiments have shown that conductance variations of up to six orders of magnitude can be obtained by reversible chemical modifications (probably hydrogenation) suggesting the possibility of realizing memory elements [130]. More recently, the experimental demonstration of graphene [131], a two-dimensional hydrocarbon with a gap of 4-5 eV obtained by hydrogenation of graphene via plasma treatment, has further proven that chemical functionalization is a viable route toward bandgap engineering of graphene-based materials. However, appropriate methods to attain good ohmic contacts and to retain high mobility (exceeding 100 cm^2/Vs) are still needed.

2.5 Graphene Nanoribbon

Graphene nanoribbon (GNR) can be thought of a strip cut off from a graphene sheet. There is a chirality vector which defines how the strip will be cut off, this vector is very different from that of CNT's. GNRs do not have periodic boundary conditions like CNTs do, because of which GNR has no closed form solution, and must be computed numerically. Following examples presented are based on the armchair edge and zigzag edge GNRs, which are similar to zigzag and armchair CNTs. [132]

2.5.1 GNR Structure

As it has been mentioned already, the most simple way of defining a graphene nanoribbon is to think of it as a strip cut off from a graphene sheet that follows a specific chiral vector. The chiral vector would indicate the direction and magnitude of the width of the GNR. The basis vectors for graphene are **a1** and **a2**, and that these vectors make the basis vectors for the CNT chirality vector. Although the origin of these vectors is not very important for CNT, it is crucial for GNRs due to the absence periodic boundary conditions. The following figure would demonstrate this fact [132].

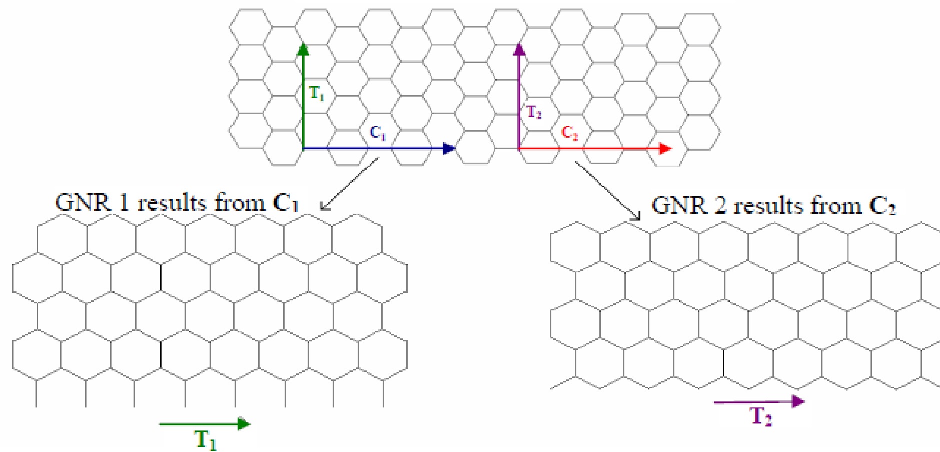


Figure 2.20: Affect of the origin of the GNR chiral vector [132].

Every GNR can be defined by a continuous infinite set of chiral vectors and their origins. Here we will discuss a specific genre of GNRs which can be defined by integer GNR chiral vectors with basis **a1**/2 and **a2**/2 and origin at either atom **a** or atom **b** of a graphene unit cell. For a graphene unit cell, **a** is the left-hand atom, and **b** is the right-hand atom. Lets call these GNRs as A-type and B-type accordingly for this example. In Fig 2.20, GNR 1 is A-type, and GNR 2 is B-type. Using this same convention, the GNR chiral vector can be defined as:

$$\vec{C} = n\,\frac{\vec{a_1}}{2} + m\frac{\vec{a_2}}{2} = \;(n,m)_{A/B} \qquad (2.10)$$

In Equation 2.10 the subscript A/B is either of value A or B, and represents the origin of the GNR chirality vector. The quantities n and m in this equation must be integers. The *transport vector* of the GNR is perpendicular and equal in magnitude to the chiral vector. Fig 2.21 shows examples of the GNR chiral and transport vectors [132].

It is imperative to note that there are some constraints on the indices that will result in stable structures. Under most conditions, n and m must be even. In the event that n and m are not even, the chiral vector will not point to a carbon atom. This rule has two exceptions. First, if m is equal to zero, then n may be odd or even. Second, if n is equal to m, then they may be odd or even [132].

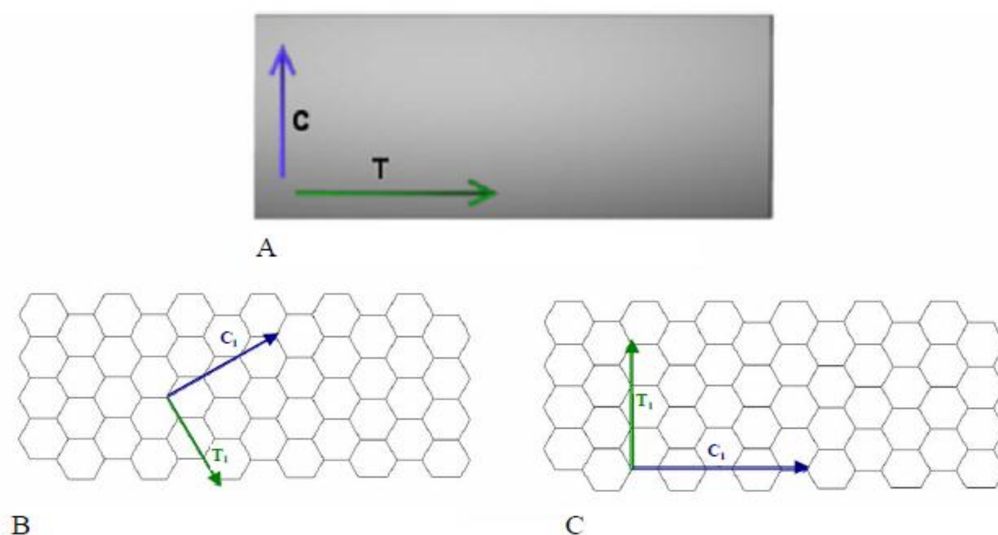


Figure 2.21: Examples of GNR chiral and transport vectors [132].

It is also important to point out that this is not the conclusive method for defining GNRs, as currently several different methods are used. One popular method has been proposed by Ezawa, which does a good job of defining physically realizable GNRs. In this method, a zigzag edge GNR with two zigzag lines that is *m* hexagons long is first defined. This chain is then displaced by an integer multiples of some translation vector to create the final nanoribbon structure. This method is not used in this paper because it has no clear relationship to the simple indexing method used to define CNTs, and thus makes direct comparison somewhat cumbersome [132].

2.5.2 Production of Graphene nanoribbon

A process called graphite nanotomy can be used to produce large quantities of width controlled GNRs [133]. Here graphite nanoblocks are produced applying a sharp diamond knife on graphite. These nano blocks can then be exfoliated to produce GNRs. Another method would be "unzipping" or cutting open nanotubes [134]. Multi-walled carbon nanotubes were unzipped in solution by action of potassium permanganate and sulfuric acid in a similar process [135]. Another method is achieved through plasma etching of nanotubes partly embedded in a polymer film [136]. Recently graphene nanoribbons have been grown onto silicon carbide (SiC) substrates using ion implantation followed by vacuum or laser annealing [137-139].

2.5.3 Electronic Structure of GNR

The electronic states of GNRs are mainly dependent on the edge structures (armchair or zigzag). Zigzag edges make available the edge localized state with non-bonding molecular orbitals near the Fermi energy. It is expected of them to have large changes in optical and electronic properties from quantization.

Calculations based on tight binding theory predict that zigzag GNRs are always metallic while armchairs can be either metallic or semiconducting, depending on the width of the armchair nanoribbons. However, it is seen from discrete Fourier transforms (DFT) calculations that armchair nanoribbons are semiconducting with an energy gap which is inversely proportional to the GNR width [140]. Experiments verified that energy gaps increase with decreasing GNR width [141]. Graphene nanoribbons with controlled edge orientation have been fabricated by scanning tunneling microscope (STM) lithography [142]. Energy gaps up to 0.5 eV in a 2.5 nm wide armchair ribbon were reported.

Zigzag nanoribbons are metallic and present spin polarized edges. Their gap opens due to an unusual antiferromagnetic coupling between the magnetic moments at opposite edge carbon atoms. This gap size is inversely proportional to the ribbon width [143] and its behavior can be traced back to the spatial distribution properties of edge-state wave functions, and the mostly local character of the exchange interaction that originates the spin polarization. Therefore, the quantum confinement, inter-edge superexchange, and intra-edge direct exchange interactions in zigzag GNR

are important for its magnetism and band gap. The edge magnetic moment and band gap of zigzag GNR are reversely proportional to the electron/hole concentration and they can be controlled by alkaline adatoms [144].

Tight-binding numerical simulation [145] obtained by means of ViDES [146] have showed that field effect transistors exploiting GNR as channel material can comply with ITRS requirements for next-generation devices.

The 2D structure, high electrical and thermal conductivity and low noise of graphene nanoribbons also makes it possible to consider GNRs a possible alternative to copper for integrated circuit interconnects. Researchers are exploring the creation of quantum dots by changing the width of GNRs at select points along the ribbon, thus leading to the creation of quantum confinement [147].

Since graphene nanoribbons possess semiconductive properties, it may be considered a technological alternative to silicon semiconductors [148] capable of sustaining microprocessor clock speeds in the vicinity of 1 THz [149]. Field-effect transistors with width less than 10 nm have been created with GNR – "GNRFETs" – with an I_{ON}/I_{OFF} ratio >10^6 at room temperature [150,151].

2.5.4 Graphene Transistors

One of the giant developments reported by the Manchester group in 2004 was a graphene MOS device. A 300-nm SiO_2 layer underneath the graphene was used as a back-gate dielectric and a doped silicon substrate operated as the back-gate (Fig. 2.22a). Such backgate devices are very handy for the purpose of proving concepts, but they are the prone to unacceptably large parasitic capacitances and cannot be integrated with other components. Therefore, practically a top gate structure is needed for graphene transistors. The first graphene MOSFET that used a top-gate was reported in 2007, marking an important milestone, and since then rapid progress has been made (Fig. 2.22b). Even though research into graphene is still in the early stages, graphene MOSFETs has shown that they can compete with devices that have benefited from decades of research and investment [152].

Top-gated graphene MOSFETs have been fabricated with exfoliated graphene i.e graphene grown on metals such as nickel and copper and epitaxial graphene; SiO_2, $Al2O_3$, and HfO_2 have been the materials of choice for the top-gate dielectric. The channels of these top-gated graphene transistors

have been made using large-area graphene, which does not have a bandgap, so it has to been possible for these transistors to switch off [152].

Large-area-graphene transistors exihibit a unique current–voltage transfer characteristic (Fig. 2.23a). The potential differences between the channel and the gates (top-gate and/or back-gate) govern th carrier density and the type of carrier (electrons or holes) in the channel. Large positive gate voltages lead to an electron accumulation in the channel (n-type channel), and large negative gate voltages promotes a p-type channel. This behavior gives rise to the two branches of the transfer characteristics separated by the Dirac point (Fig. 2.23a). The position of the Dirac point depends on several factors: the difference between the work functions of the gate and the graphene, the type and density of the charges at the interfaces at the top and bottom of the channel (Fig. 2.21), and any doping of the graphene. The on–off ratios reported for MOSFET devices with large-area-graphene channels are in the range 2–20 [152].

Recently, there have been reports of graphene MOSFETs with gigahertz capabilities. These transistors have large-area channels of exfoliated and epitaxial graphene. The fastest graphene transistor that currently exists is a MOSFET that has a 240-nm gate with a cut-off frequency of f_T = 100 GHz, which is quite higher than those of the best silicon MOSFETs that have similar gate lengths (as is the cut-off frequency of 53 GHz reported for a device with a 550-nm gate. A drawback of all radio frequency graphene MOSFETs reported so far is the unsatisfying saturation behaviour (only weak saturation or the second linear regime), which has an negative impact on the cut-off frequency, the intrinsic gain and other figures of merit for radiofrequency devices. However, outperforming silicon MOSFETs while operating with only weak current saturation is certainly quite impressive [152].

One way of introducing a bandgap into graphene for logic applications is the creation of graphene nanoribbons. Nanoribbon MOSFETs with back-gate control and having widths of even less than 5 nm, have been operated as p-channel devices and that showed on–off ratios of up to 106. Such high ratios have been obtained despite simulations showing that edge disorder leads to an undesirable decrease in the on-currents and a simultaneous increase in the off-current of nanoribbon MOSFET. This, and other evidence of a sizeable bandgap opening in narrow nanoribbons, provides sufficient

proof that nanoribbon FETs are highly suitable for logic applications. However, due to their relatively thick back-gate oxide of these devices, voltage swings of several volts were needed for switching, which is significantly more than the swings of 1 V and less needed to switch Si CMOS devices. Additionally, CMOS logic requires both n-channel and p-channel FETs with well-controlled threshold voltages, and graphene FETs that has all these properties have not yet been reported [152].

Recently, the fabrication of the first graphene nanoribbon MOSFETs with topgate control have been reported. These transistors feature a thin high-dielectric-constant (high-*k*) top-gate dielectric (1–2 nm of HfO_2), a room-temperature on–off ratio of 70 and an outstanding transconductance of 3.2 mS μm^{-1} (which is higher than the transconductances reported for state-of-the-art silicon MOSFETs and III-V HEMTs) [152].

Investigation of graphene bilayer MOSFETs have been carried out experimentally and also device simulation has been performed. Although the on–off ratios seen so far (100 at room temperature and 2,000 at low temperature83) are too small for logic applications, they note a significant improvement(of about a factor of 10) over MOSFETs in which the channel is made of large-area gapless graphene [152].

We now return to the discussion of two-dimensional nature of graphene. According to theory of scaling a thin channel region allows short-channel effects to be suppressed and thus makes it feasible to scale MOSFETs to very short gate lengths. The two dimensional nature of graphene means that the thinnest possible channel can be obtained by using graphene, so graphene MOSFETs should be more scalable than their competitors. However, it should be noticed that scaling theory is valid only for transistors with channels semiconducting in nature and does not apply to graphene MOSFETs with gapless channels. Thus, the scaling theory does describe nanoribbon MOSFETs, which not only have a bandgap but which have significantly lower mobilities than large area graphene, as discussed. Given that the high published values of mobility relate to gapless large-area graphene, the most attractive characteristic of graphene for use in MOSFETs, especially those required to switch off, is probably its ability to scale to shorter channels and higher speeds, rather than its mobility [152].

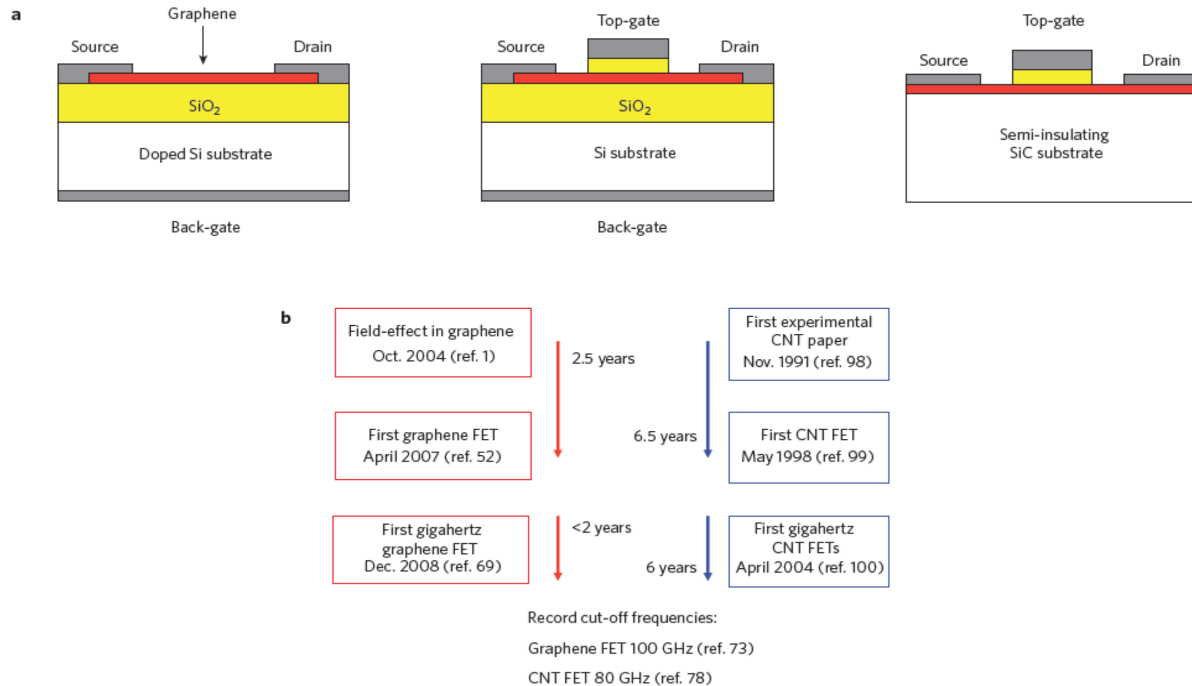


Figure 2.22: Structure and evolution of graphene MOSFETs. **(a)** Schematics of different graphene MOSFET types: back-gated MOSFET (left); top-gated MOSFET with a channel of exfoliated graphene or of graphene grown on metal and transferred to a SiO_2-covered Si wafer (middle); top-gated MOSFET with an epitaxial-graphene channel (right). The channel shown in red can consist of either large-area graphene or graphene nanoribbons. **(b)** Progress in graphene MOSFET development compared with the evolution of nanotube FETs [153].

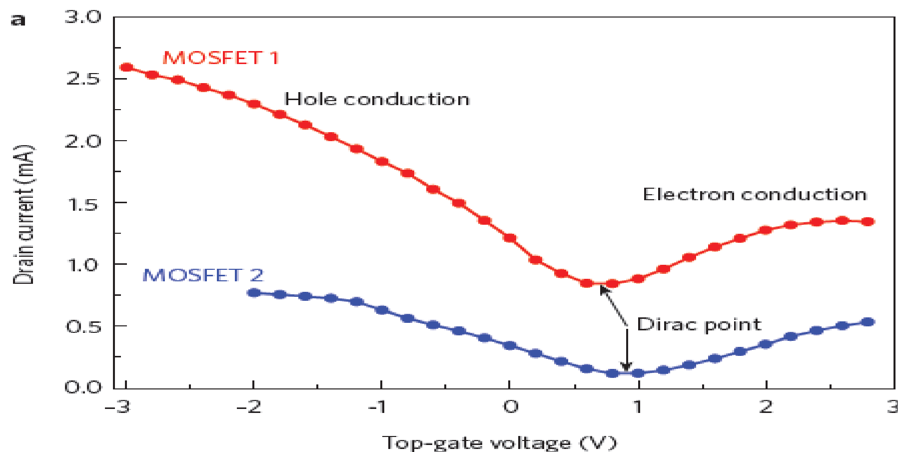


Figure 2.23: Direct-current behaviour of graphene MOSFETs with a large-area-graphene channel**.** Typical transfer characteristics for two MOSFETs with large-area-graphene channels. The on–off ratios are about 3 (MOSFET 1) and 7 (MOSFET 2), far below what is needed for applications in logic circuits. Unlike conventional Si MOSFETs, current flows for both positive and negative top-gate voltages [153].

2.6 Summary:

The limitation of conventional Si-MOSFET, properties of carbon nanotube, types of carbon nanotube, characteristics of Carbon nanotube, properties of graphene, synthesis of graphene, band structure of graphene and properties of graphene nanoribbon have already been discussed. The production techniques and electronic structure of GNR are also covered in this chapter. It is found that the performance properties of CNT and GNR have given a higher performance properties compared to conventional properties. Each type transistor is modeled in difference way based on the structure of the transistor. This discussion includes different types of CNTFET from starting CNT technology, operation of CNTFET and types of CNTFET based on this operation. Also, various types of graphene transistors are discussed in this chapter and some details about their operation and properties are also given.

Chapter 3

RESULTS AND CHARACTERIZATIONS OF CNTFET AND GNRFET USING NANOTCAD ViDES

This chapter will explain the methodology used in this project, simulation model used for simulation study, simulation result obtained, comparing those results with other reliable research group's results and finally making summary, analysis and discussion on the result.

3.1 The Model

This research implicates simulation based study to investigate the effect on I-V characteristic by changing different parameters of CNTFET and GNRFET. This python based simulation study is carried out based on the self-consistent solution of the 3-D Poisson and Schrödinger equations with open boundary conditions within the non-equilibrium Green's function formalism and a tight-binding Hamiltonian [155]. The model is built for a Schottky barrier field effective transistor in order to investigate ballistic transport in CNTFET and GNRFET. The goal is to modify the MATLAB code such a way to investigate the effect on I-V characteristics by changing major parameters of CNTFET and GNRFET and also focused the result on 2D plot.

3.1.1 Model Physics and the Process of Calculation

The potential profile in the 3-D simulation domain obeys the Poisson equation [156]

$$\nabla[\in(\vec{r})\nabla\emptyset(\vec{r})] = -q\,[\,p(\vec{r}) - n(\vec{r}) + N_D^{+}(\vec{r}) - N_A^{-}(\vec{r}) + \rho_{fix}] \qquad (3.1)$$

Where, $\emptyset(\vec{r})$ is Electric Potential, $\in(\vec{r})$ is Dielectric Constant, ρ_{fix} is Fixed Charge, N_D^{+} is concentration of ionized donor and N_A^{-} is concentration of ionized acceptor.

The electron and hole concentrations (*n* and *p*, respectively) are computed by solving the Schrödinger equation with open boundary conditions by means of the NEGF formalism [157]. A tight-binding Hamiltonian with an atomistic (*pz* orbitals) real-space basis for CNT [158] and GNR [159] has been used with a hopping parameter *t* = 2.7 eV.

Green's function can then be expressed as

$$G(E) = [EI - H - \Sigma S - \Sigma D] - 1 \qquad (3.2)$$

Where, E is Energy, I is Identity Matrix, H is Hamiltonian, Σ_S is self − energy of the source and Σ_D is self − energy of the drain. Transport here is assumed to be ballistic.

The length and chirality of CNT or GNR are now defined and the coordinates in the 3-D domain of each carbon atom are then computed [160]. After that, the 3-D domain is discretized so that a grid point is defined in correspondence with each atom, while a user-specified grid is defined in regions not including the CNT or GNR.

A point charge approximation is assumed, i.e., all the free charge around each carbon atom is spread with a uniform concentration in the elementary cell including the atom. Assuming that the chemical potential of the reservoirs is aligned at the equilibrium with the Fermi level of the CNT or GNR, and given that there are no fully confined states, the electron concentration is

$$n(\vec{r}) = 2 \int_{E_i}^{+\infty} dE \left[|\psi_S(E,\vec{r})|^2 f\left(E - E_{F_S}\right) + |\psi_D(E,\vec{r})|^2 f\left(E - E_{F_D}\right) \right] \qquad (3.3)$$

while the hole concentration is

$$p(\vec{r}) = 2 \int_{+\infty}^{E_i} dE \left[|\psi_S(E,\vec{r})|^2 [1 - f(E - E_{F_S})] + |\psi_D(E,\vec{r})|^2 [1 - f\left(E - E_{F_D}\right)] \right] \qquad (3.4)$$

Where, $\vec{r}$ is coordinate of carbon site, f is Fermi − Dirac factor,

$|\psi_S|^2$ is probability that states injected by the source reach the carbon site ($\vec{r}$),

$|\psi_D|^2$ is probability that states injected by the drain reach the carbon site ($\vec{r}$),

E_{FS} is Fermi level of the source and E_{FD} is Fermi level of the drain.

The current is computed as

$$I = \frac{2q}{h} \int_{-\infty}^{+\infty} dE\mathcal{T}(E)[f(E - E_{F_S}) - f(E - E_{F_D})] \qquad (3.5)$$

where q is the electron charge, h is Planck's constant, and $\mathcal{T}(E)$ is the transmission coefficient computed as [157]

$$\mathcal{T} = -\mathrm{Tr}\,[(\Sigma_s - \Sigma_s^{\dagger})G(\Sigma_D - \Sigma_D^{\dagger})G^{\dagger}] \qquad (3.6)$$

where Tr is the trace operator. In the present model, we only deal with the one-dimensional (1-D) transport between source and drain reservoirs, while the leakage gate current has not be taken into account. For the considered devices with channel length of a few nanometers, it can be shown that the gate current is negligible with respect to the drain current.

Detail discussion about the physics and mathematical calculation of modeling is provided in Appendix A. Figure 3.1 describes the total calculation procedure that is done in the simulation.

Table 3.1: Parameters and physical constants used in the simulation.

Input Parameters	**Default Values**
Gate Insulator Thickness, t (nm)	Boltzmann's Constant, k= 1.8×10^{-23} J/K
	Planck's constant,h= 6.63×10^{-23}
Relative dielectric constant, ε_r	Reduced Planck's constant, ħ= 1.05×10^{-34}
Temperature, T (K)	
Gate Voltage, V_G(V)	Mass of electron, m_0= 9.11×10^{-31} kg
Drain Voltage, V_D(V)	Source Fermi level, E_f= −0.32eV
	Overlap integral of tight bonding C-C model, σ= 2.7 eV
Channel Length, L_{ch}(nm)	Charge of an electron, q=1.6×10^{-19} C
Chiral axis,(n,0)	Permittivity of free space, ε_0= 8.854×10^{-12}
	C-C bond length, $a_{c\text{-}c}$= 1.42×10^{-10} m

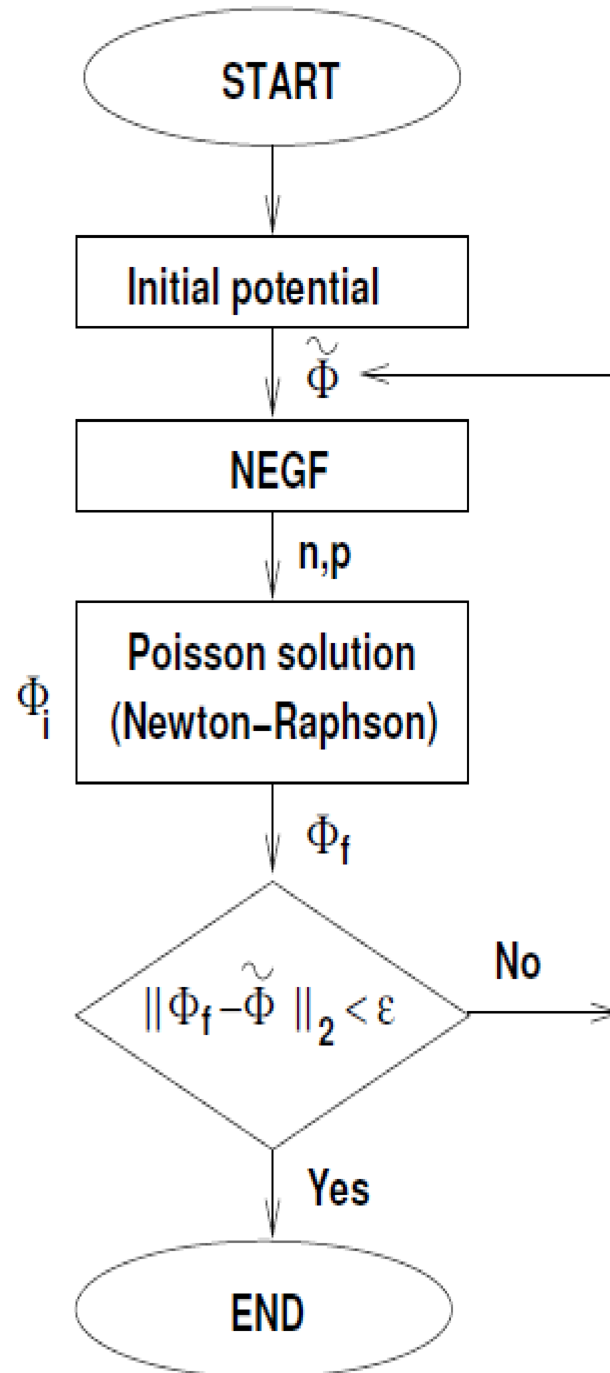


Figure 3.1: Flow-chart of the self-consistent 3D Poisson-Schrodinger solver [155].

3.2 Result and Analysis

From the simulation different parameter changing effect on I-V characteristics of CNTFET and GNRFET is shown. There are five results obtained and each result has individual effect on CNTFET and GNRFET. Result started with gate oxide thickness and gradually improved the work by identifying temperature effect, dielectric constant effect, gate and drain control parameter effect and also chirality changing effect on Carbon nanotube based field effect transistor and Graphene nanoribbon field effect transistor. The CNTFET and GNRFET structure that considered for the simulation is Schottky barrier ballistic FET is drawn in Fig 3.2.

For studying the various effects, perfectly patterned 15nm long N=12 armchair –GNR and a 15nm long zigzag-CNT having chirality (13, 0) are used as the default channel materials in the simulation due to their similar bandgaps. The default relative dielectric constant for both the FETs is taken to be 3.9. The default gate oxide thickness is 1.5nm and lateral spacing is 0.5nm. The default temperature is taken to be 300K. This default values are used in the simulation if the user does not specify the values of the parameters mentioned above.

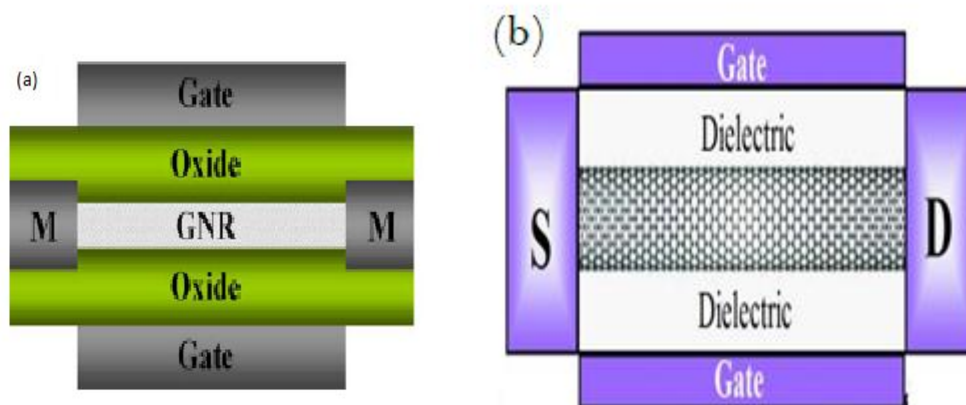


Figure 3.2: (a) Structure of GNRFET [179] (b) Structure of CNTFET [Source-internet image].

3.2.1 Effect of Temperature

Now the concern is the effect of changing temperature on the transfer and output characteristics of graphene nanoribbon and carbon nanotube SBFET. Figure 3.3 and 3.4 deals with the temperature changing effect. The simulation is carried out at temperatures 273K, 300K, 323K, 343K and 373K.
In Fig 3.3, both GNR SBFET and CNT SBFET display ambipolar characteristics. Both the FETS have off-state leakage currents which are of the same order of magnitude. The on-state drain currents are also similar for both GNRFET and CNTFET. It is observed that for different temperatures the drain current remains same for different gate voltages. The reason for this behavior is the ballistic consideration of the model.

The effect of changing temperatures on the output characteristics for both the FETs is shown in Fig 3.4. As the temperature varied between 273K and 373K, there is a small increase in the on-state drain current for each FET. This is increase is due to the decrease in channel resistance of the channel materials. Fig 3.5 shows the how the resistance of graphene nanoribbon decreases as temperature is increased from 198K to 373K studied by Huaqing Xie *et al* [161]. At any particular temperature, the on-state drain current of GNRFET is higher compared to the on-state drain current of CNTFET. The GNRFET has drain current in the order of 10^{-6} and the drain current of CNTFET has an order of 10^{-7}. In Figure 3.4, At V_D=0.25V, GNRFET has a current of 1.53×10^{-6} A at T=300K while the CNTFET has a current of 5.32×10^{-7} A. Thus, we can conclude that the channel resistance of graphene nanoribbon is lower compared to the channel resistance of carbon nanotube at any particular temperature resulting in a higher on-state drain current for GNR. The off-state leakage current for both GNRFET and CNTFET can be found by extrapolation of the output characteristics graphs and it can be concluded that the GNRFET will have a higher off-state leakage current than CNTFET.

Fig 3.6 shows the schematic of a FET based on GNR arrays patterned by BCP lithography [162].Fig 3.6 (c) and (d) shows its transfer and output characteristics recorded by Son *et al* in the temperature range 100K-300K. It is observed in the figure that as the temperature of the patterned FET increases, its ambipolar behavior decreases. It is seen in Fig 3.3 (a) that for the GNR SBFET

simulated in this thesis paper, the ambipolar behavior decreases with increasing temperature. Thus it can be concluded that ambipolar behavior decreases with increasing temperature

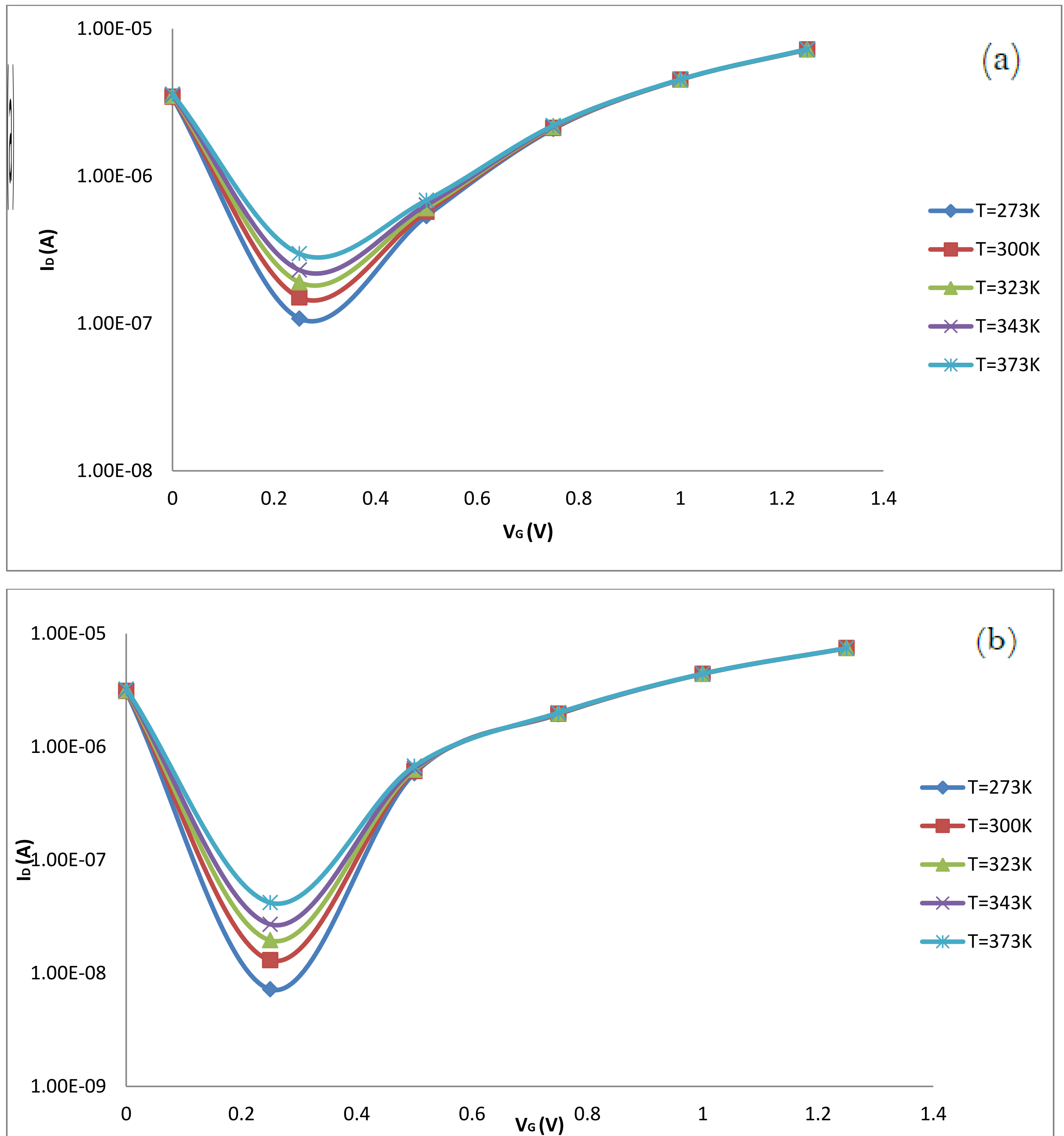
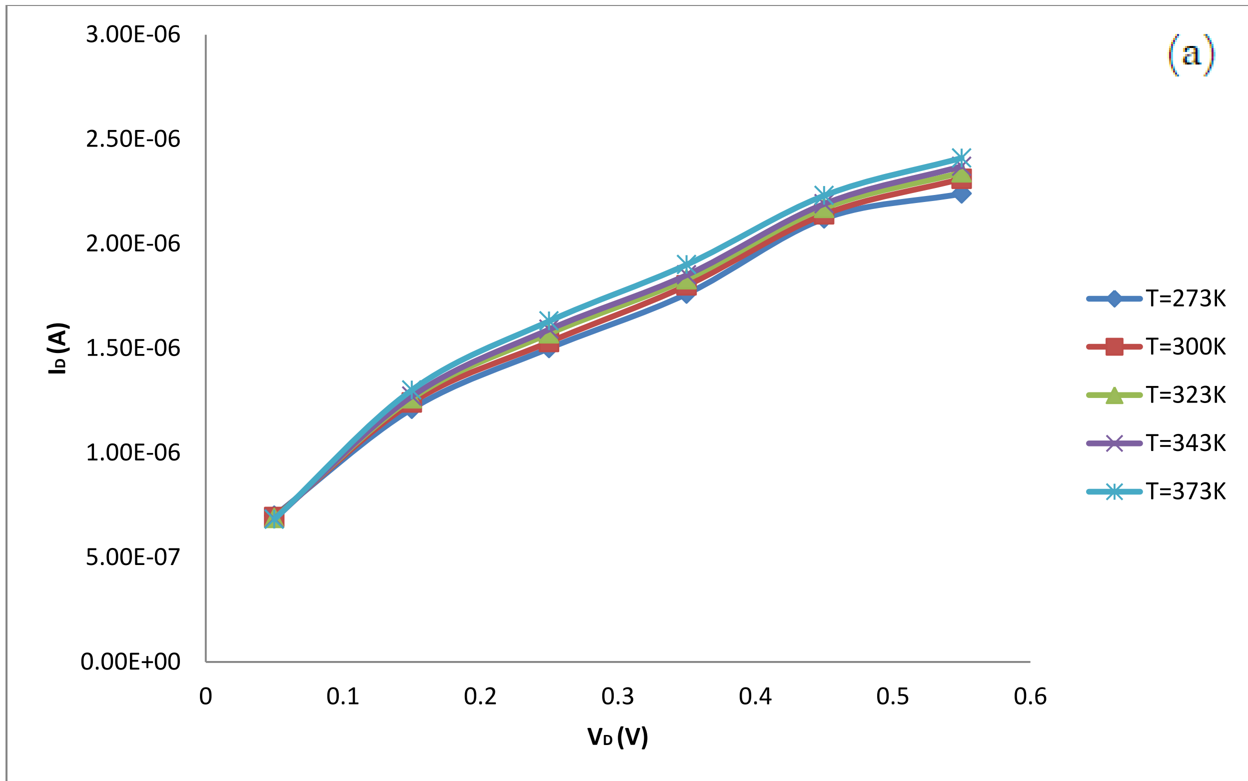


Figure 3.3: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different temperatures at V_D = 0.5 V.



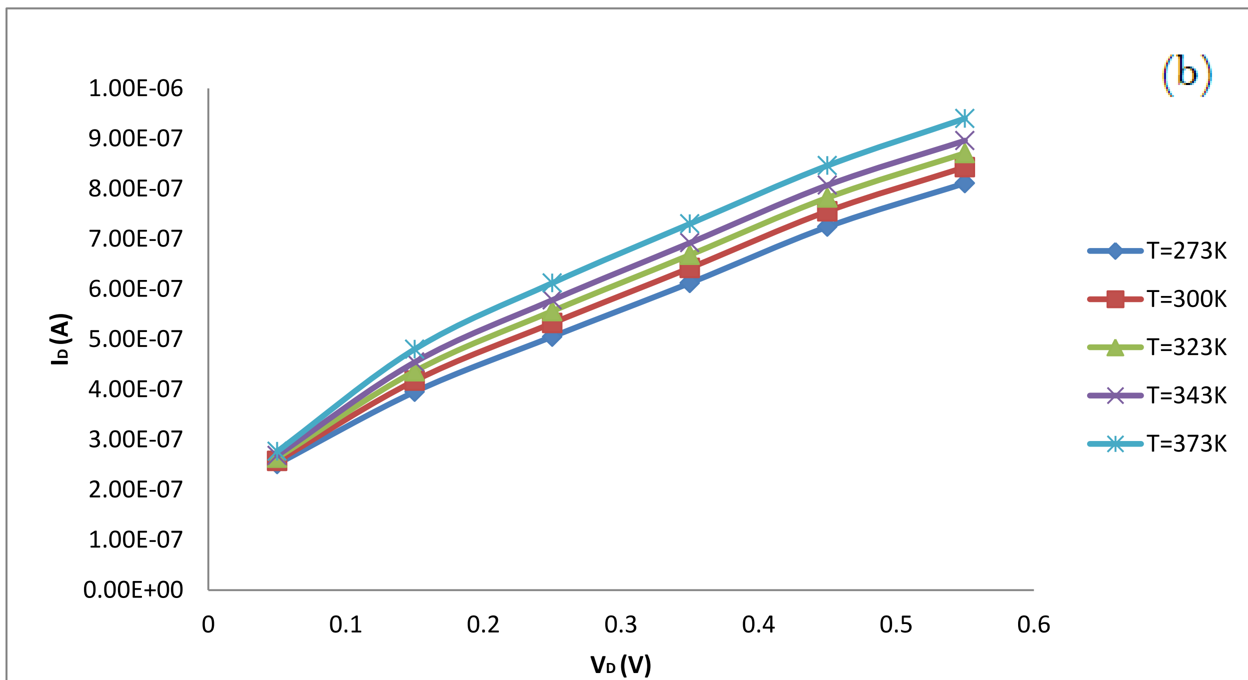


Figure 3.4: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different temperatures at $V_G = 0.5$ V.

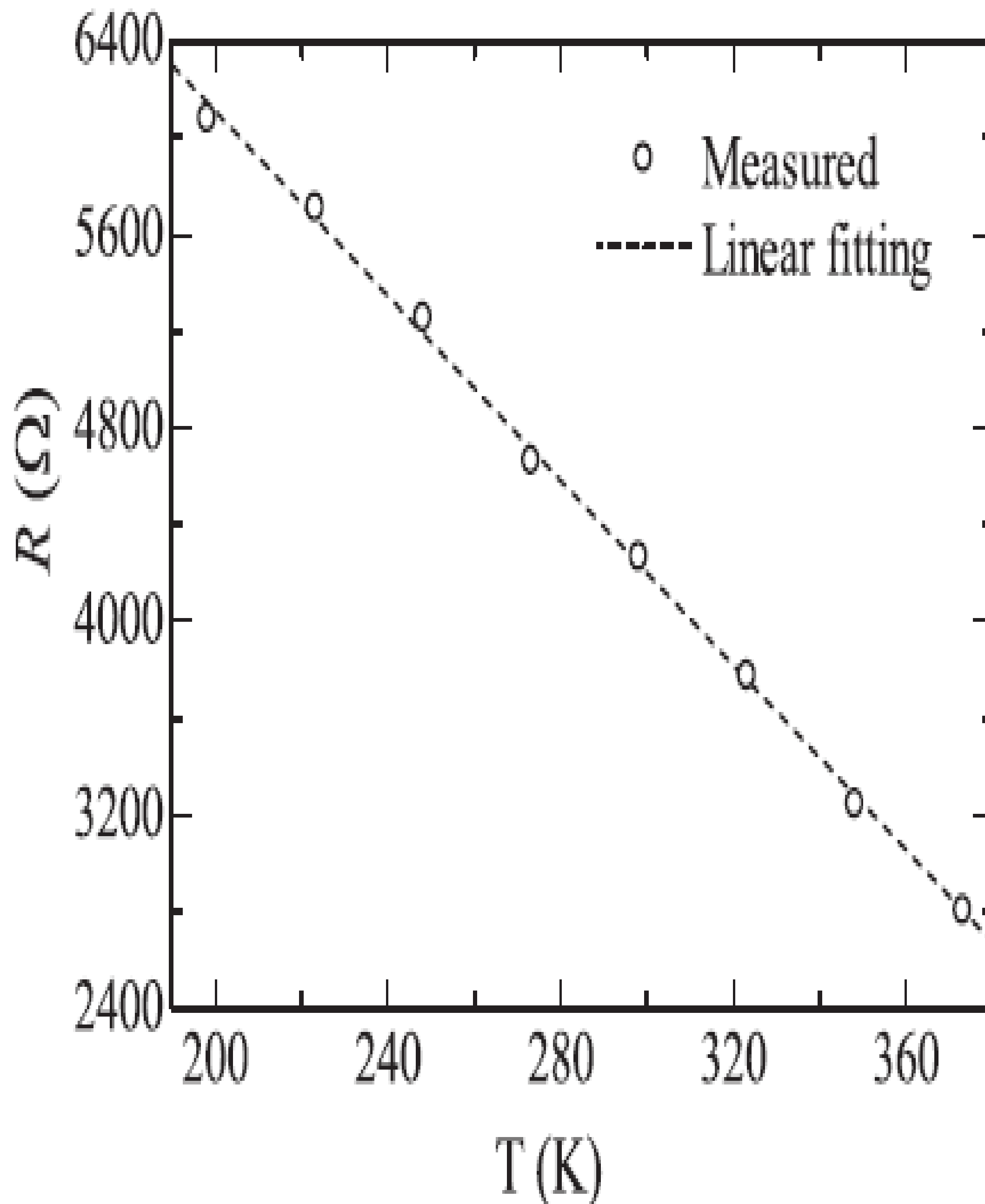


Figure 3.5: Dependence of resistance on temperature of graphene nanoribbon [161].

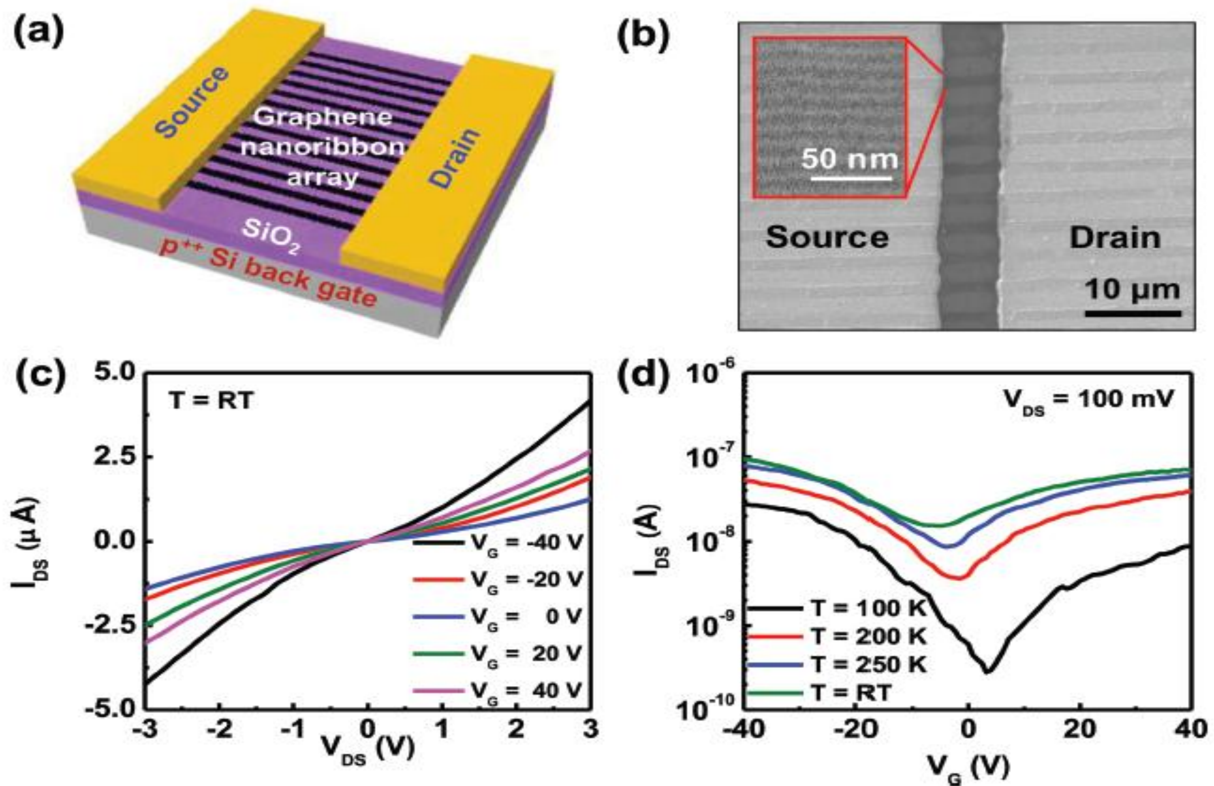


Figure 3.6: (a) Schematic of a FET based on GNR arrays patterned by Block Copolymer lithography and (b) the corresponding SEM image. In (b), the contrast difference in the channel between the GNR arrays and the bare silica is evident. (c) *I* DS - *V* DS curves of the GNR array FET with a 9 nm ribbon width recorded at different gate voltages. (d) *I* DS - *V* G curves of the GNR array FET with a 9 nm ribbon width recorded at *V* DS = 100 mV in the temperature range of 100–300 K [162].

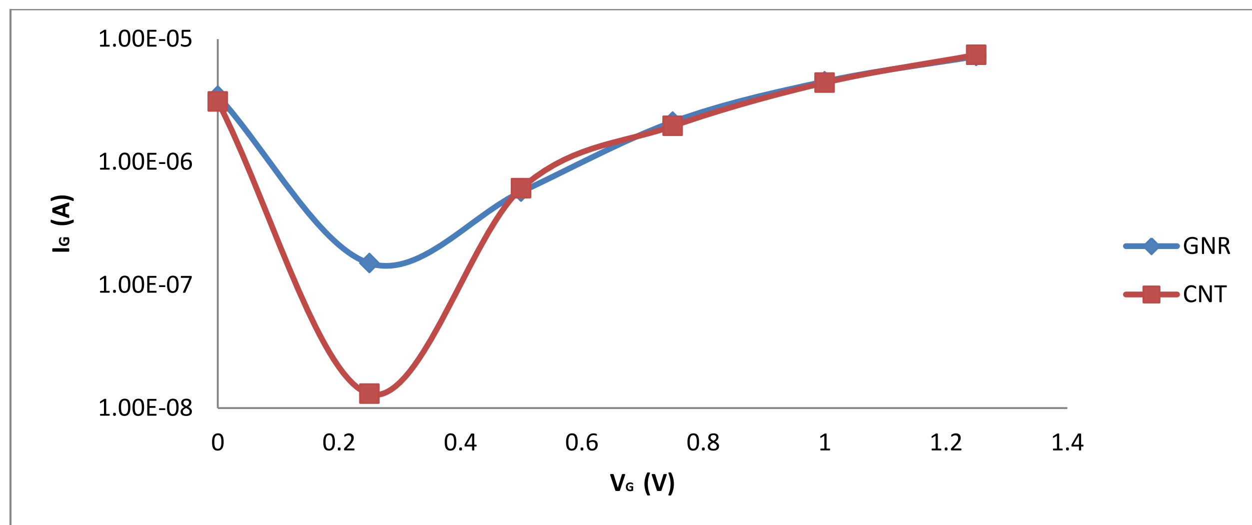


Figure 3.7: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET at T= 300K and V_D = 0.5 V.

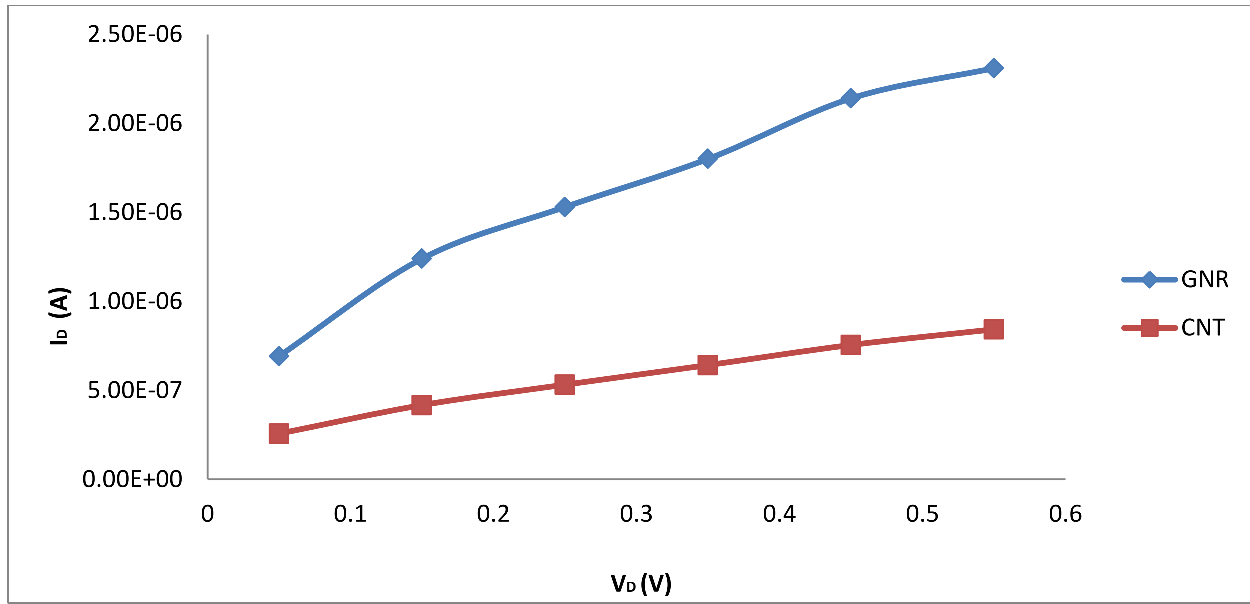


Figure 3.8: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET at T= 300K and V_G = 0.5 V.

3.2.2 Effect of Relative Dielectric Constant

At this stage of report now the concern is to investigate the effect on GNRFET and CNTFET transfer and output characteristics by changing the dielectric constant. Naturally SiO_2 is used as a gate oxide material which has a dielectric constant of 3.9. But other material can be used as an oxide material for better performance. At this case dielectric constant will change definitely. Figure 3.9 shows the results of changing dielectric constant studied by Rasmit Sahoo *et al* [164].

Because of scaling, bulk Si MOSFET suffers from many limitations like short channel effect, tunneling etc. as it is said before in chapter 2. To overcome these limitations many solutions were proposed by different researchers. Use of high dielectric material as gate insulator was one of the proposed solutions [163-167]. Keeping this in eye it is tried to see the effect of using different dielectric materials as gate insulator in GNRFET and CNTFET. In this case the dielectric constant is changed within a range of 3.9 to 15.9 and interval is 4 keeping other parameter constant as usual. At this inspect the temperature is kept at 300K which was a subject of change in our previous experiment. In Fig 3.10, the transfer characteristics of GNRFET and CNTFET are shown for different values of relative dielectric constant. Both the FETs show ambipolar characteristics. From the figure,

it is observed that transfer characteristics for both GNRFET and CNTFET are similar. It is seen that as the relative dielectric constant is increased, the on-state drain current at any particular voltage increases. For GNRFET, at V_G=0.5V the value of drain current at k=3.9 is 5.7×10^{-7} A and at k=15.9 is 5.74×10^{-6} A. For CNTFET, at V_G=0.5V the value of drain current at k=3.9 is 6.05×10^{-7} A and at k=15.9 is 1.56×10^{-6} A.

Fig 3.11 shows the output characteristics of the FETs for different values of relative dielectric constant. When the value of relative dielectric constant is 3.9, the off-state leakage current of CNTFET is observed to be lower than the off-state current of GNRFET. The on-state drain current of GNRFET at k=3.9 is of the order of 10^{-6}A and that of CNTFET is of the order of 10^{-7}A. As the value of relative dielectric constant increases, both the off-state leakage current and the on-state drain current increases for both GNRFET and CNTFET. The GNRFET has a higher saturation current compared to the CNTFET. For k=15.9, the current in the GNRFET is 5µA and the current in the CNTFRT is 2.33 µA at V_D=0.55V. It is clear from the plot is that the saturation current increases for increasing dielectric constant but degree of this positive effect reduces as going for higher dielectric material [166] . This means that going for higher and higher dielectric material the increment in *ID* with respect to k reduces. These results also match with the result of Rasmita Sahoo *et al* Fig 3.9 [164].

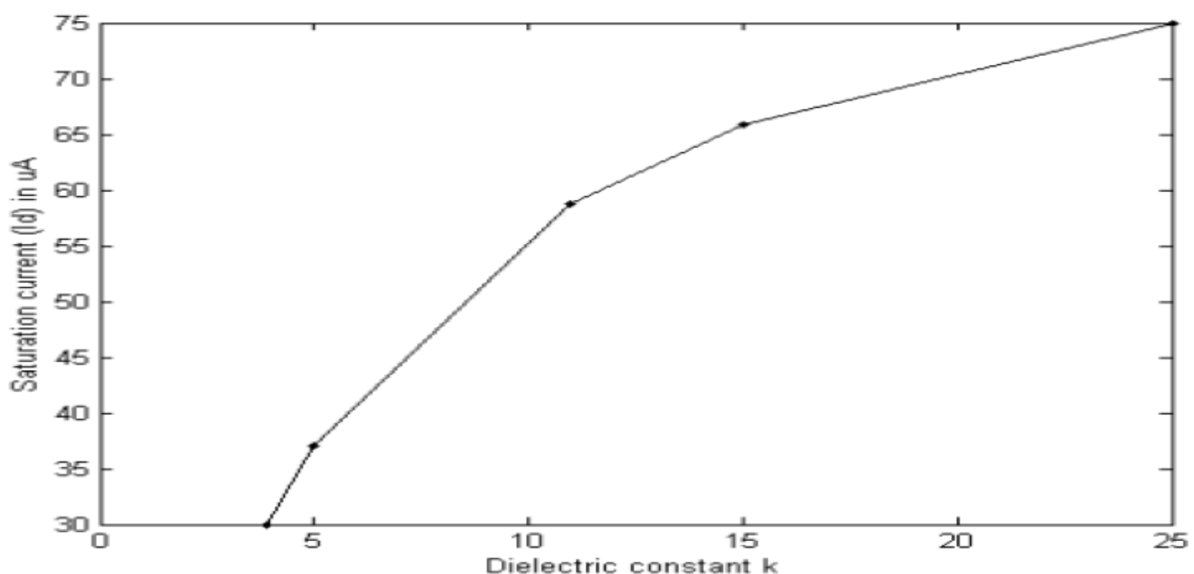
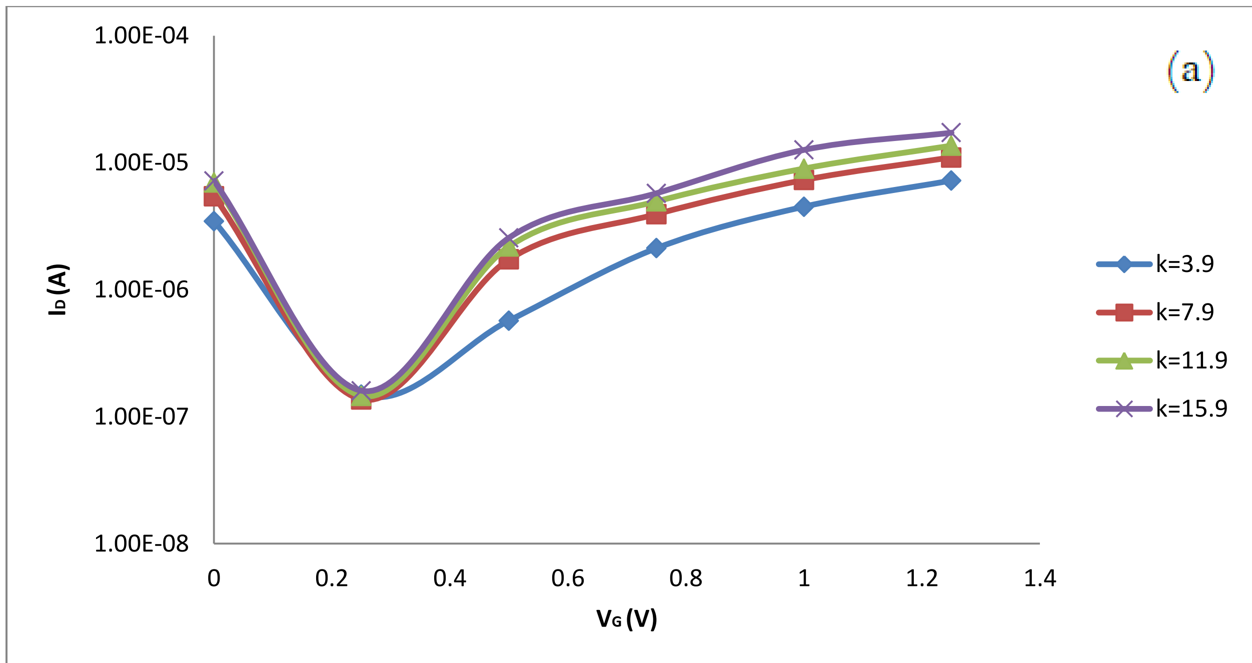


Figure 3.9: Dielectric constant changing effect investigated by Rasmita Sahoo *et al.* which satisfied simulation result [164].



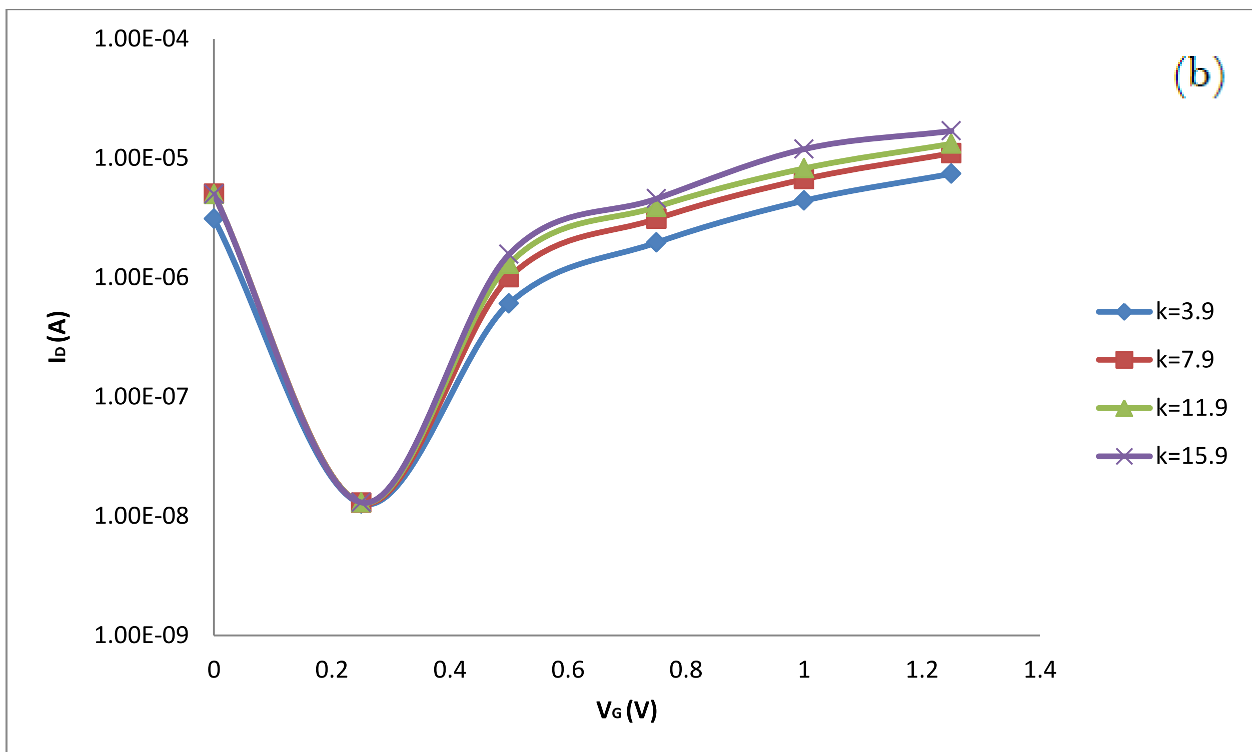
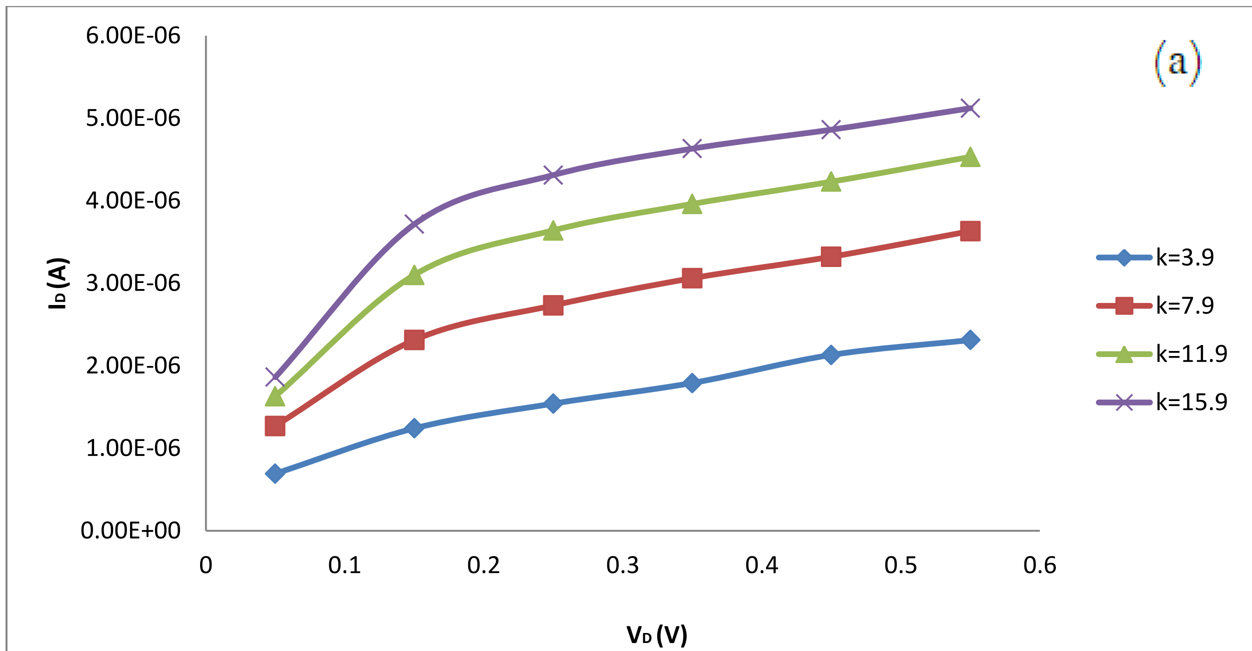


Figure 3.10: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different relative dielectric constant at $V_D = 0.5$ V.



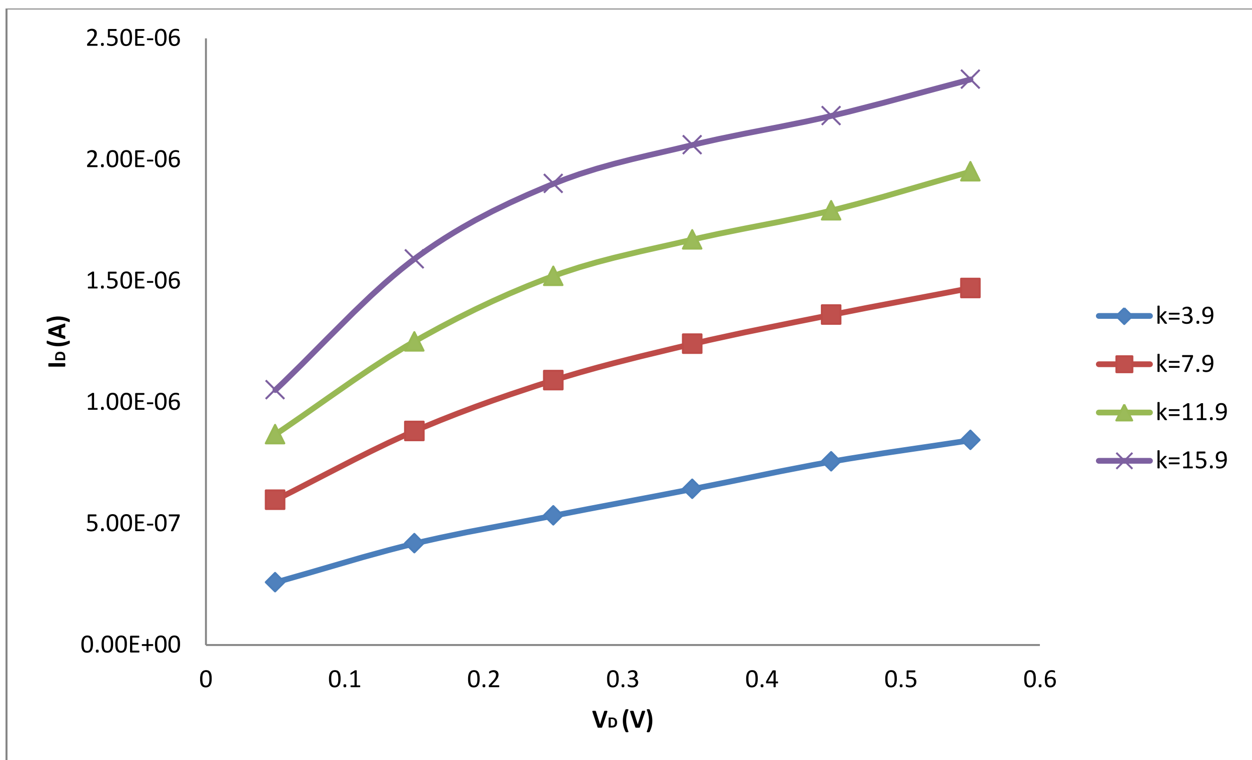


Figure 3.11: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different relative dielectric constant at $V_G = 0.5$ V.

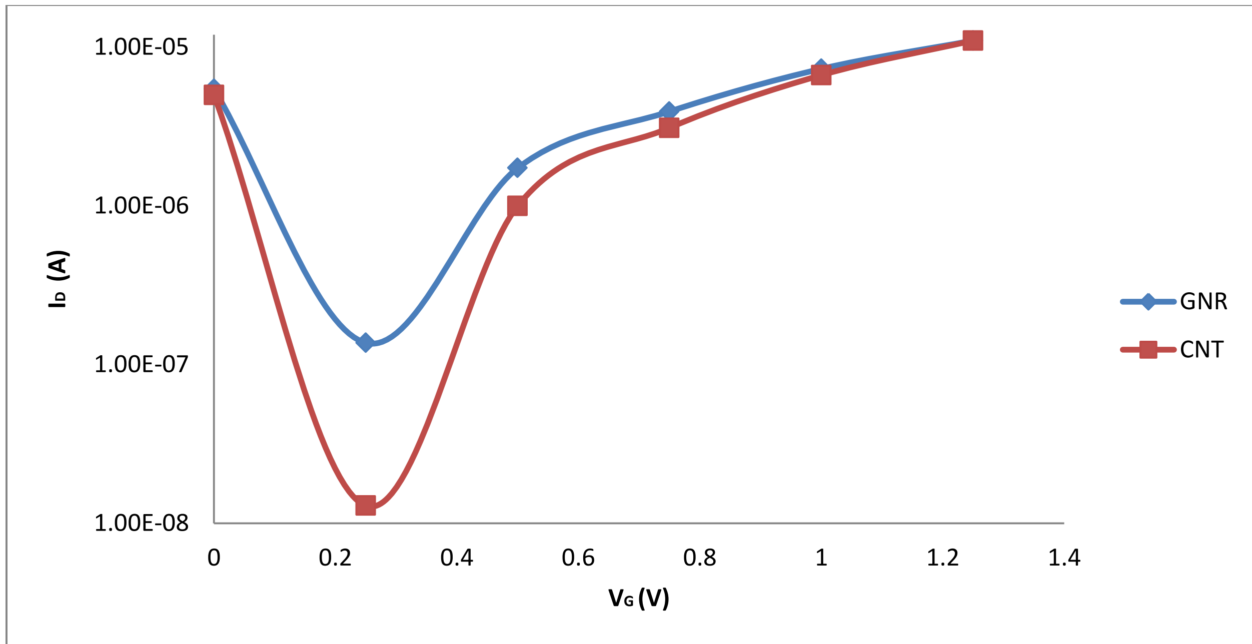


Figure 3.12: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for relative dielectric constant of 7.9 at $V_D = 0.5$ V.

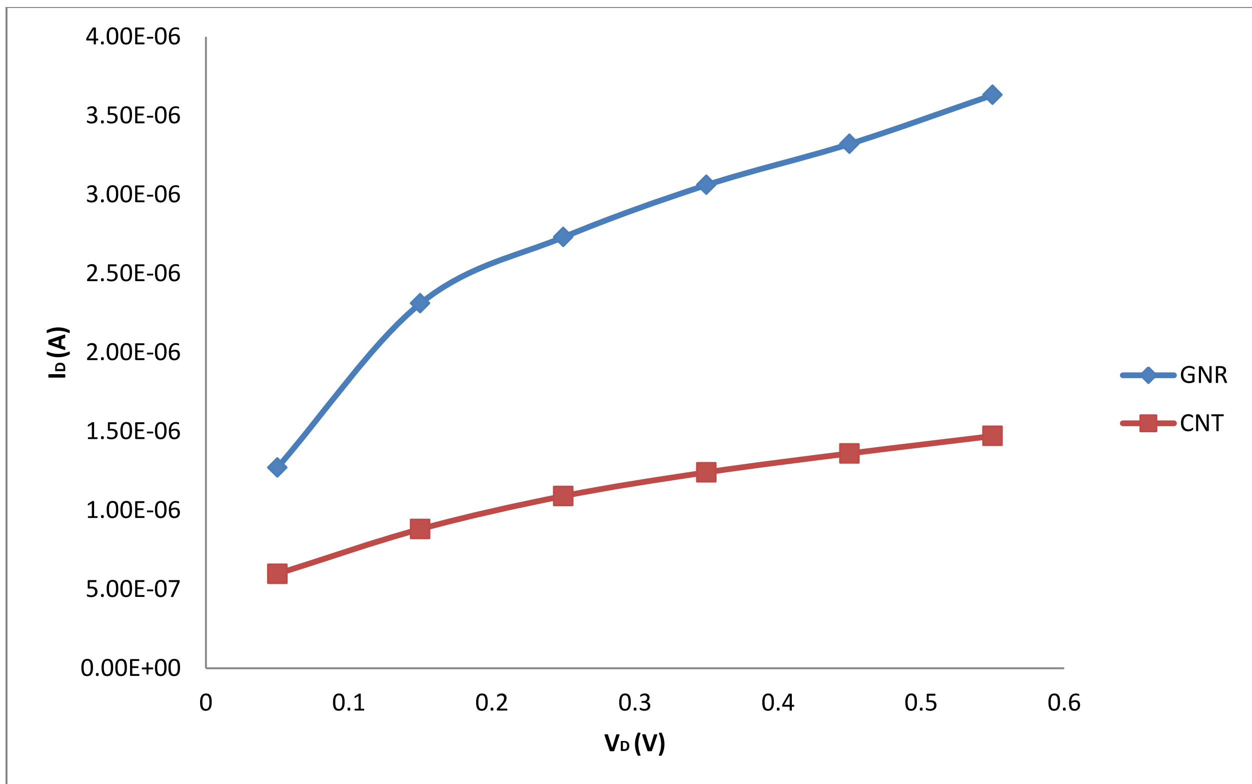


Figure 3.13: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for relative dielectric constant of 7.9 at $V_G = 0.5$ V.

3.2.3 Effect of Chirality:

Now the discussion will relate the changing effect of chirality on GNRFET and CNTFET. Actually chirality relates with the diameter and diameter changing effect is very important for FET. That's why chirality changing effect is very important for any Graphene Nanoribbon or Carbon Nanotube based design. The equation that relates chirality and diameter is:

$$d_t = a\sqrt{n^2 + m^2 + nm}/\pi \tag{3.7}$$

Where, m and n is the chiral axis (n, m). Here n should be always greater than m. The width of Graphene Nanoribbon is equal to the perimeter of the Carbon Nanotube [168].

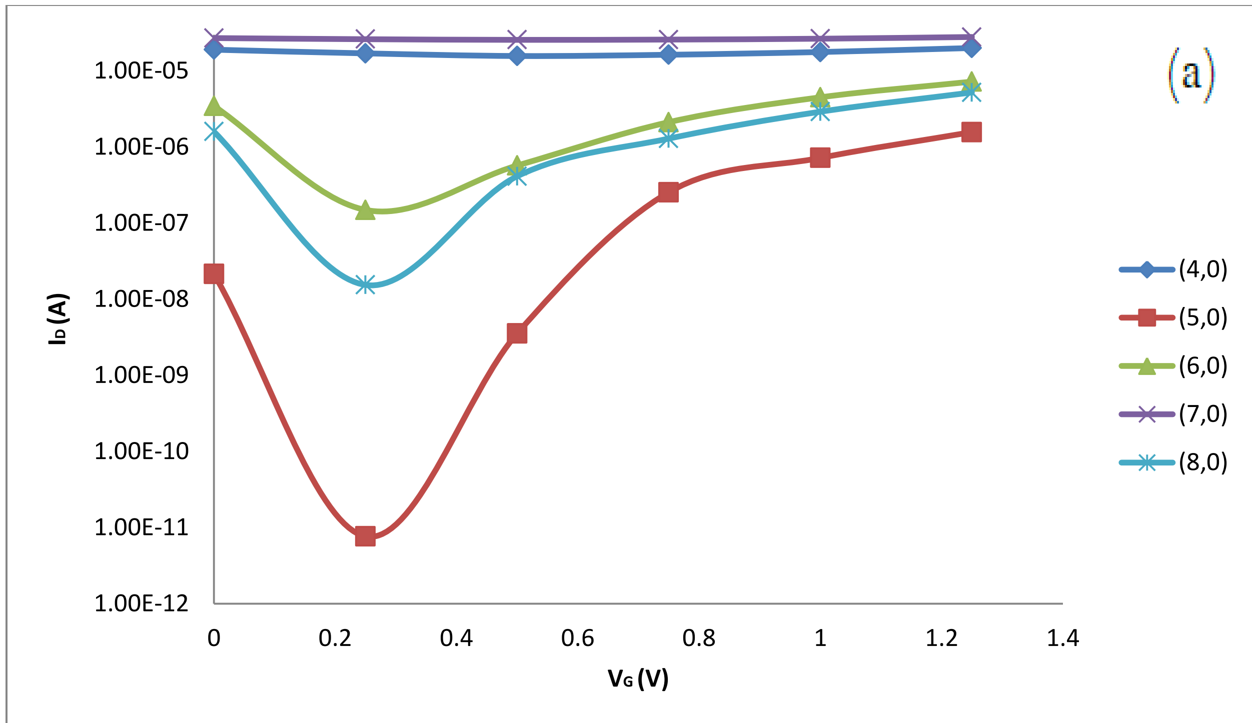
The energy bandgap of the CNT is inversely proportional to the nanotube diameter *(Egap α 1/Diameter)* and also inversely proportional to the width of the GNR(*Egap α 1/Width).* Since the drain current of CNFET is dependent on the total charge that filled up the first subband, therefore it is possible that the drain current too depends on the diameter of CNT [169]. So ultimately the drain current depends on the chirality. In this experiment it will observe one chiral axis changing effect that is n. When it is changing n we will keep m value constant and the range of n value should be such a value that is always greater than m value to observe the effect on output. Other parameter will maintain their default value as like previous experiment.

Fig 3.14 shows the transfer characteristics of both GNR SBFET and CNT SBFET for different chirality. In Fig 3.14 (a), variation in I_D vs. V_G graph is seen for different chiralities of Graphene Nanoribbon. Graphene Nanoribbon can be metallic or semiconducting [170]. It is metallic when *N*= 3M-1, where *M* is an integer and N=2n. Thus, N=8 and N=14 GNR are metallic. Hence, the gate voltage has little control over the drain currents as shown in in Fig 3.14(a). N=10 GNR has a higher bandgap and thus it shows higher ambipolar characteristics. N=12 GNR has a lower bandgap than N=16 GNR. This is because the two GNRs belong to different semiconducting families [171]. Thus, it is observed that N=12 GNR displays less ambipolar characteristics and also has a higher off-state leakage current and on-state drain current than N=16 GNR. Yijian Ouyang *et al* also studied the effect of chirality on GNR

[167]. The result is shown in Fig 3.16. It is observed that for semi-conducting GNRs, the bandgap decreases with increasing chirality as width increases. This observation agrees with the result obtained by this solution. In Fig 3.14(b) the ambipolar characteristics decreases and drain current increases as CNT diameter increases with increasing chirality. CNT with chirality (6, 0) is the only exception to the above conclusion since it is metallic while the rest is semiconducting. Comparing Fig 3.14(a) and Fig 3.14(b) it is seen that the order of the drain current of GNR SBFET is greater than the order of the drain current of CNT SBFET of the same chirality. For (5, 0) GNRFET, at V_G=0.5V the value of drain current is 3.55×10^{-7} A and for (5, 0) CNTFET, the value of drain current is 6.3×10^{-17} A and at k=15.9 is 1.56×10^{-6} A.

The output characteristics of GNRFET and CNTFET for different chirality are shown in Fig 3.15. In Fig 3.15 (a) the metallic Graphene Nanoribbons have significantly higher on-state drain currents compared to the semiconducting ones. Fig 3.15 (b) shows the output characteristics of a CNTFET. It is observed that the metallic CNTFET has a much higher drain current than the semi-conducting ones. (6, 0) CNTFET has a drain current in the order of 10^{-5} A whereas the drain current of the semiconducting CNTFETs is significantly lower. Also, the order of magnitude of the drain current of GNRFET is much higher than the order of magnitude of the drain current of CNTFET. The output characteristics of GNR SBFET for different chirality studied by Yijian Ouyang *et al* are shown in Fig 3.16 (b). Their result matches with our simulation result for N=12 GNR.

Fig 3.17(a) shows the transfer characteristics of a schottky barrier CNTFET for chirality (13, 0), (17,0) and (25, 0) studied by Guo *et al* [172]. Fig 3.17 (b) shows the results of the simulation of the CNTFET presented in this thesis at the same values of chirality at V_D= 0.4V. It is observed that both the figures display similar results. In Fig 3.17(a) the leakage current for chirality (13, 0) is of the order 10^{-7} A and as the diameter increases the leakage current also increases. The leakage current for (25,0) CNTFET is of the order of 10^{-5}A. In Fig 3.13(b) the leakage current for (13,0) CNTFET is 5.04×10^{-7}A and for (25,0) CNTFET is 1.09×10^{-5}A. It is also seen that the on-state current increases with increasing diameter.



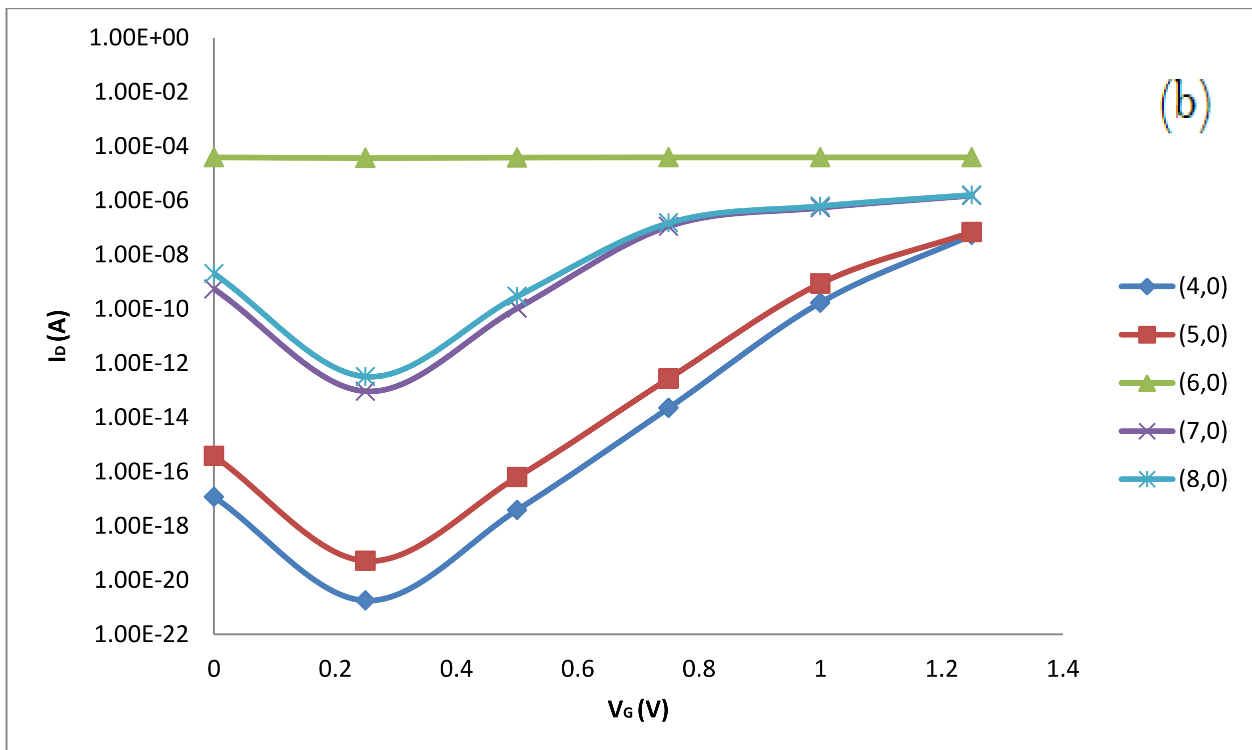
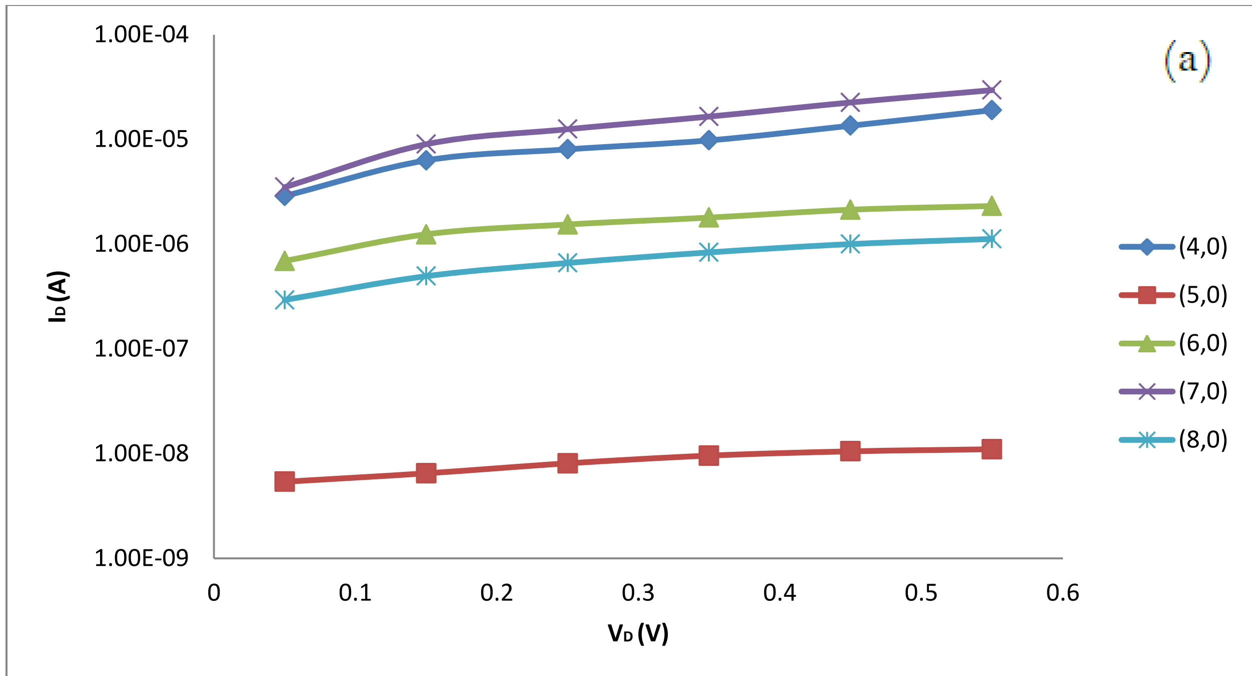


Figure 3.14: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different chirality at $V_D = 0.5$ V.



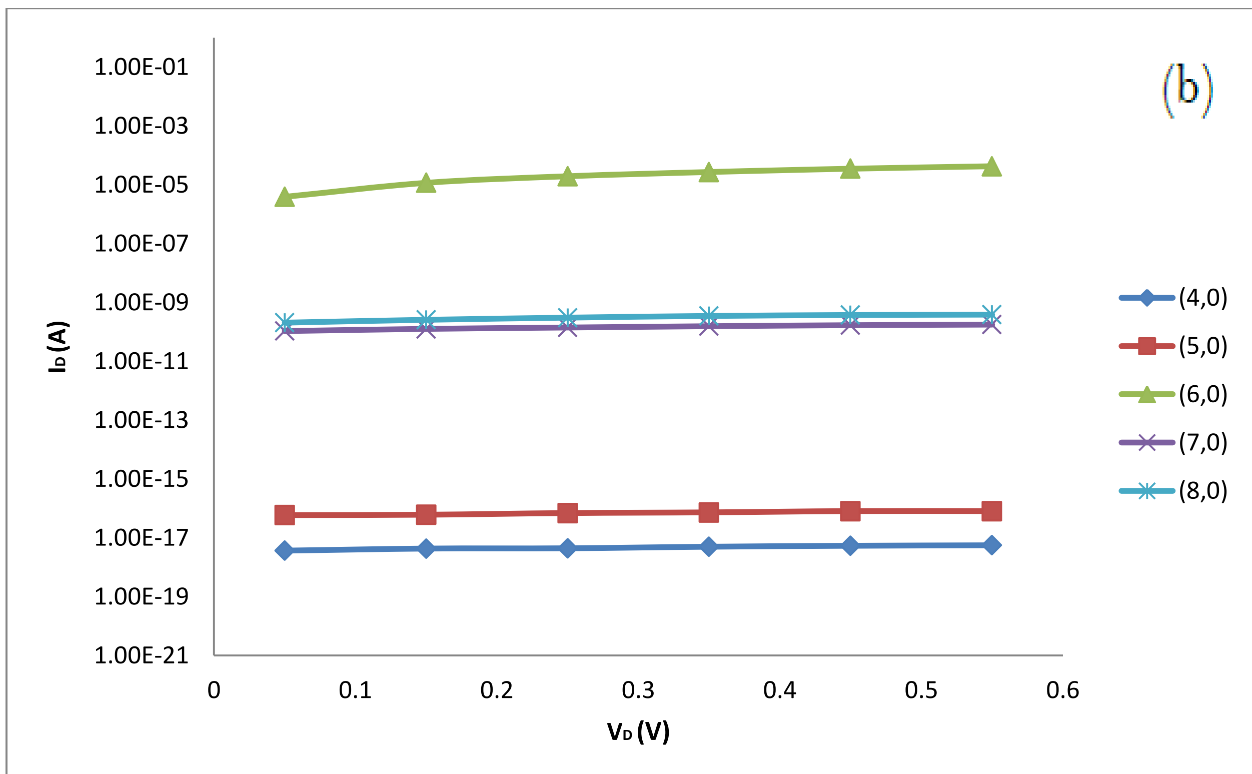


Figure 3.15: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different chirality at $V_G = 0.5$ V.

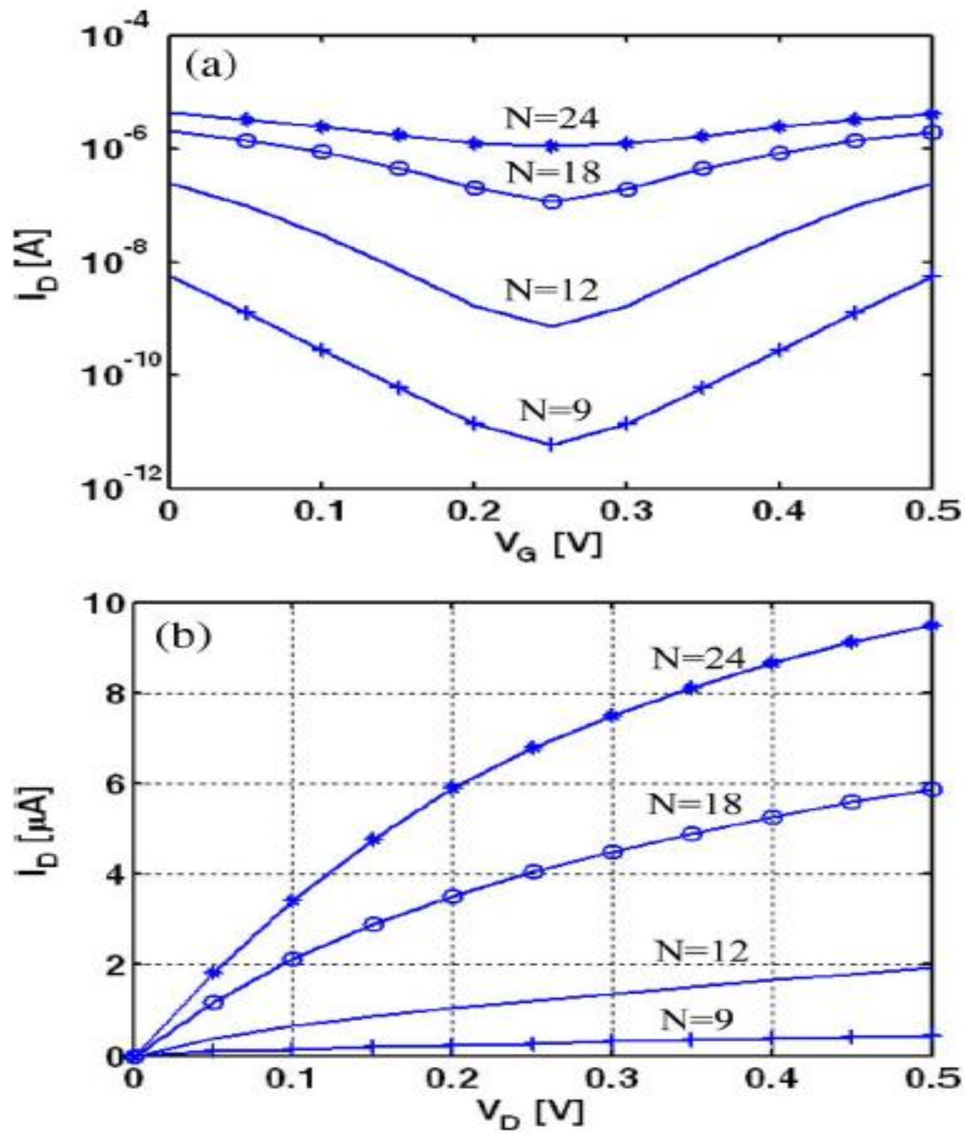
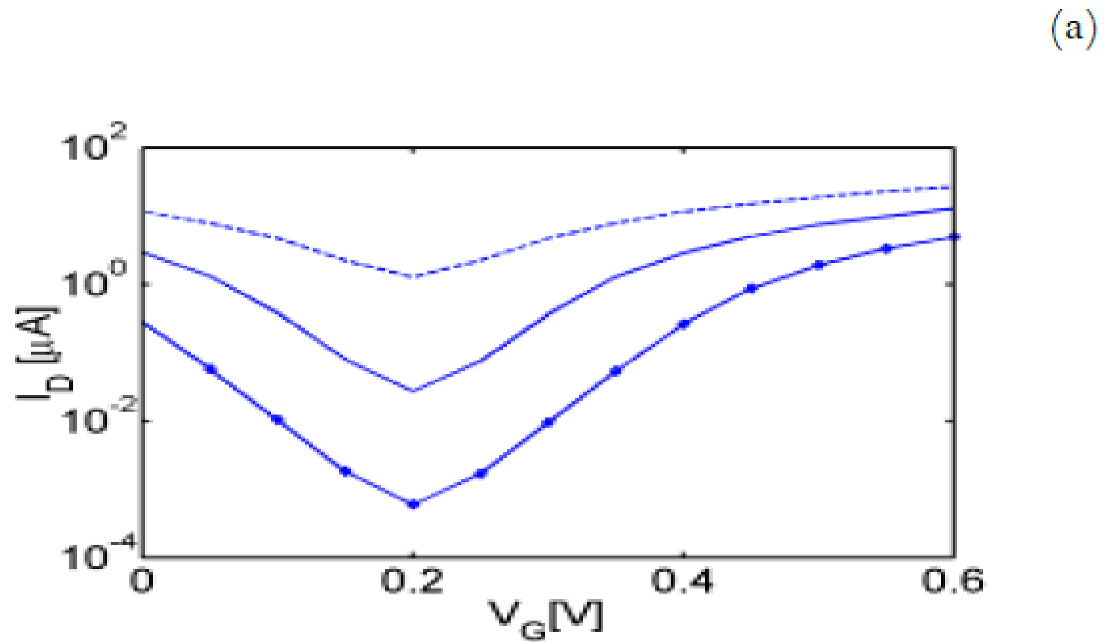


Figure 3.16: (a) Transfer characteristics of GNR SBFET (b) Output Characteristics of GNR SBFET for different chirality studied by Yijiang Ouyang *et al* [167].



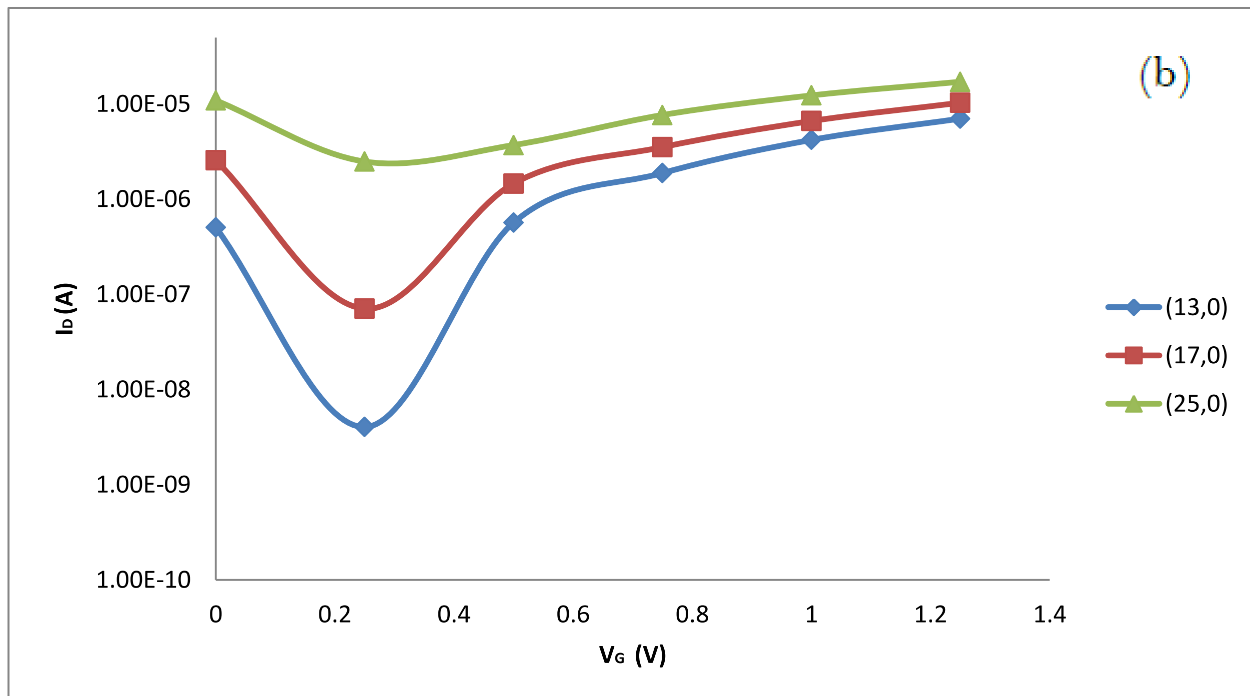


Figure 3.17: (a) Scaling of nanotube diameter. ID vs. VG characteristics at VD=0.4V for the nominal CNTFET with different nanotube diameter. The solid line with circles is for (13,0) CNT (with d~1nm), the sold line is for (17,0) CNT (with d~1.3nm), and the dashed line is for (25,0) CNT (with d~2nm) studied by Guo *et al* [172]. **(b)** I_D vs. V_G characteristics of the CNTFET simulated in this thesis paper for chirality (13, 0), (17, 0) and (25, 0) at V_D = 0.4V.

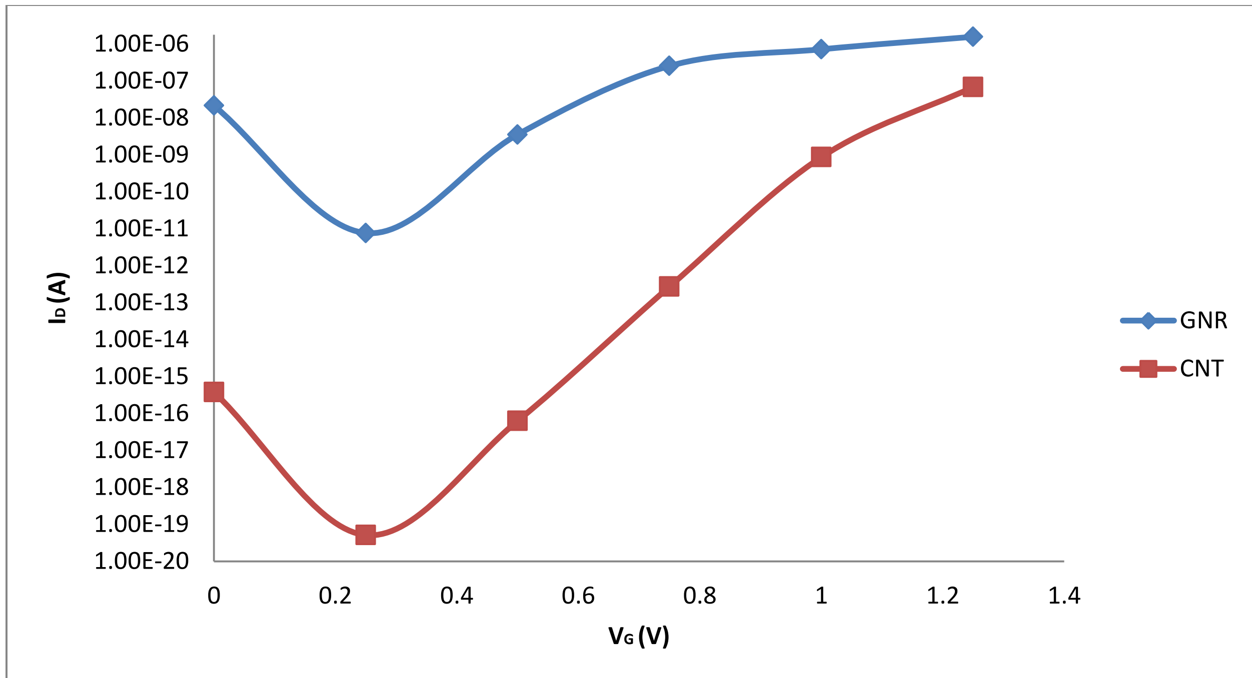


Figure 3.18: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for chirality (5,0) at V_D = 0.5 V.

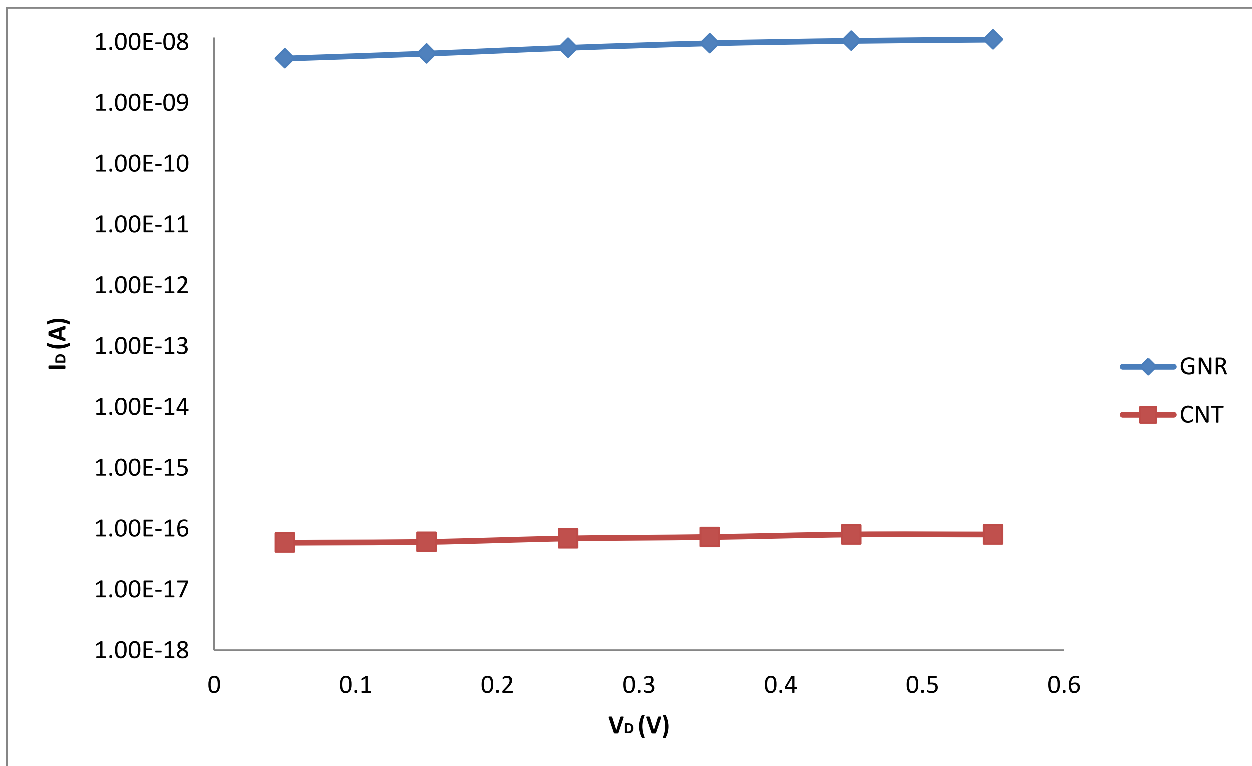


Figure 3.19: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for chirality (5,0) at V_G = 0.5 V.

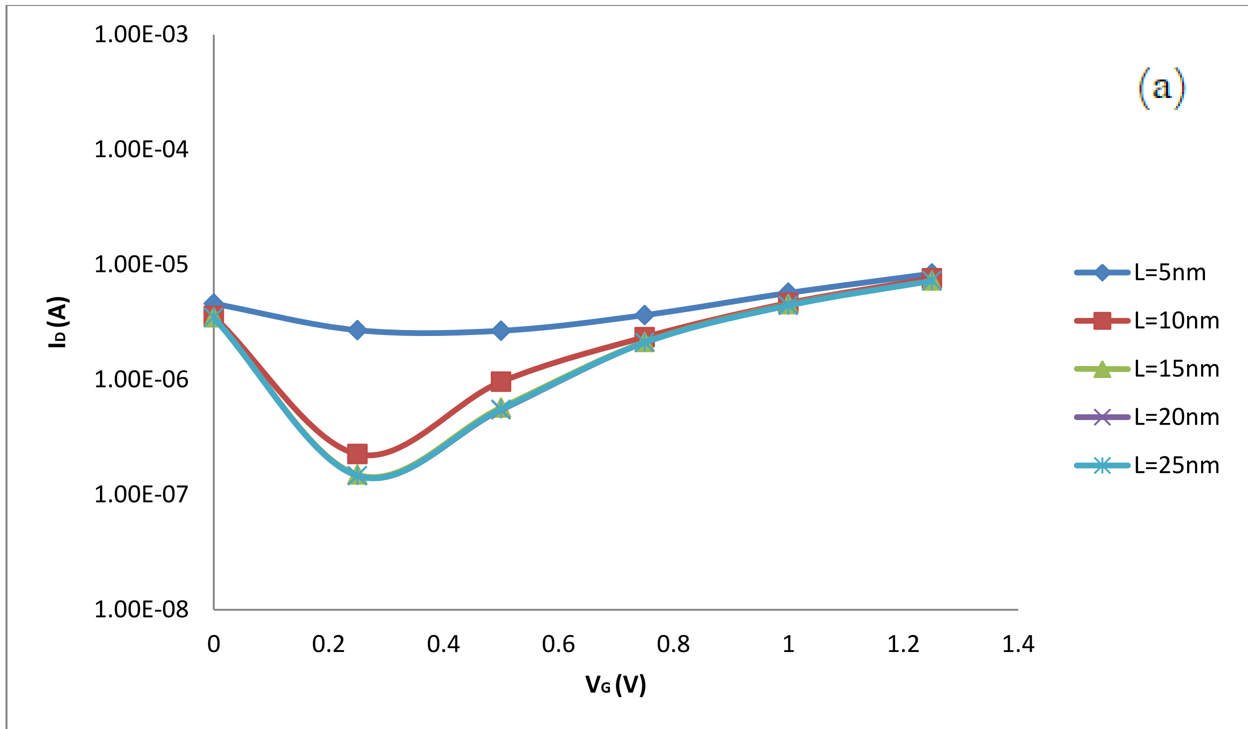
3.2.4 Effect of Channel Length:

In this section the effect of the changing channel length L_{ch} on GNRFET and CNTFET characteristics will be discussed. Initially the channel length is taken to be 5nm. Then it is increased in increments of 5nm till 25nm. The transfer characteristics of GNR SBFET and CNT SBFET for different channel lengths are shown in Fig 3.20. In Fig 3.20, it is observed that for both GNR SBFET and CNT SBFET when channel length is 5nm, the gate voltage has very little control over the on-state drain current. As a result, at this channel length the FETs show very little ambipolar characteristics. When channel length is 10nm, the FETs have a higher on-state drain current at V_G = 0.25V compared to the FETs with channel length greater than 10nm. For higher channel lengths the transfer characteristics of the FETs is similar as can be seen in the figure 3.20. The transconductance remains approximately same for L_{ch}> 15nm [167]. This is because the current does not depend on the channel length in the ballistic regime. However, for L_{ch}< 15nm the transconductance decreases [167]. As a result, variation in transfer characteristics for L_{ch}= 5nm and L_{ch}=10nm is observed in Fig 3.20.

Fig 3.21 shows the output characteristics of GNR SBFET and CNT SBFET. In Fig 3.21 (a), it is observed that for L_{ch}=5nm, the on-state drain current is significantly larger than the current for L_{ch}>10nm. For 5-nm-long channel direct tunneling from the source to drain and electrostatic short channel effects are severe; therefore, no decent saturation of I_D–V_D characteristics is observed [167]. For L_{ch}>10nm, the output characteristics remains similar as the transconductance remains same. In Fig 3.21 (b), for L_{ch}= 5nm direct tunneling effect is only observed at V_D>0.35V. However, for higher channel lengths, L_{ch}>10nm, as V_D is increased the drain current for the different channel lengths have the same value as seen in Fig 3.21(b).

Fig 3.22 shows the results of changing channel lengths on a single geometry GNR SBFET studied by Yijiang Ouyang *et al* [167]. Comparing these results to our simulation results it is observed that for L_{ch}>10nm, the performance of a single geometry and a double geometry GNR SBFET is similar. However, at L_{ch}=5nm, the double geometry GNR SBFET used in this simulation allows a higher drain current to flow as shown ins Fig 3.20 and 3.21 compared to the single geometry used by Yijiang Ouyang *et al* in Fig 3.22.

A graph of on-state drain current per unit width as a function of channel length for CNT MOSFET simulated by Fiori *et al* is shown in Fig 3.23 (a) [156]. There it is seen that, a (11,0) CNT MOSFET has a drain current per unit width of 15000 μA/μm for a channel length of 5nm at V_G= 0.8V. After that, as the channel length increases the on-state drain current decreases. Fig 3.23 (b) shows the drain current per unit width as a function of channel length of the schottky barrier CNTFET simulated in this thesis at V_G = V_D = 0.8V. The CNTFET simulated for this figure has chirality of (11, 0). The CNT SBFET has an on-state drain current per unit width of 826 μA/μm at a channel length of 5nm. As the channel length increases the drain current per unit width decreases. So it can be concluded that for CNT the drain current per unit width decreases with increasing channel length. However, the order of magnitude of the on- state drain current per unit width of CNT MOSFET is higher than the on-state drain current per unit width of CNT SBFET.



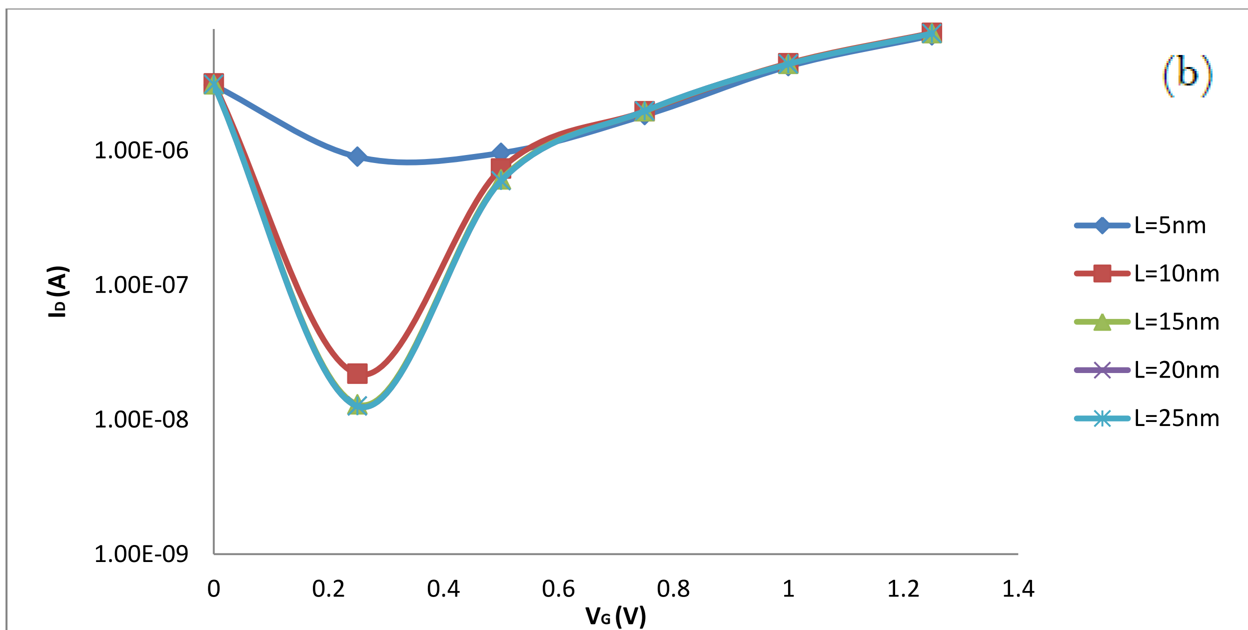
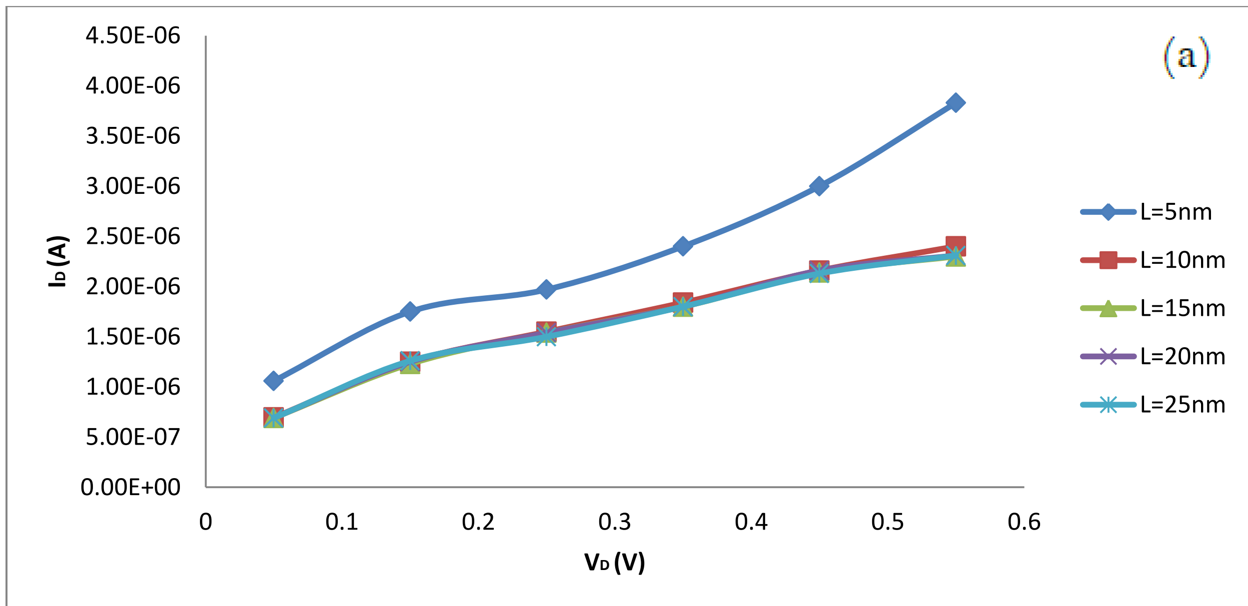


Figure 3.20: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different channel lengths at $V_D = 0.5$ V.



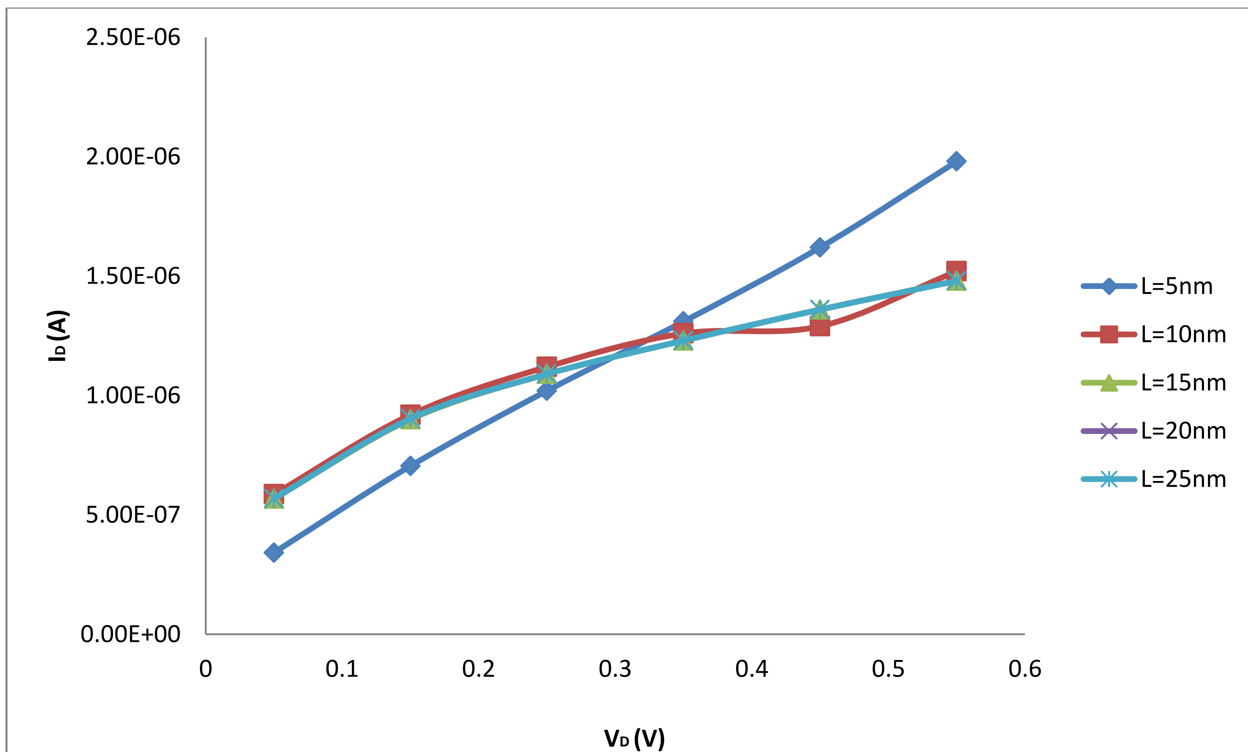


Figure 3.21: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different channel lengths at V_G = 0.5 V.

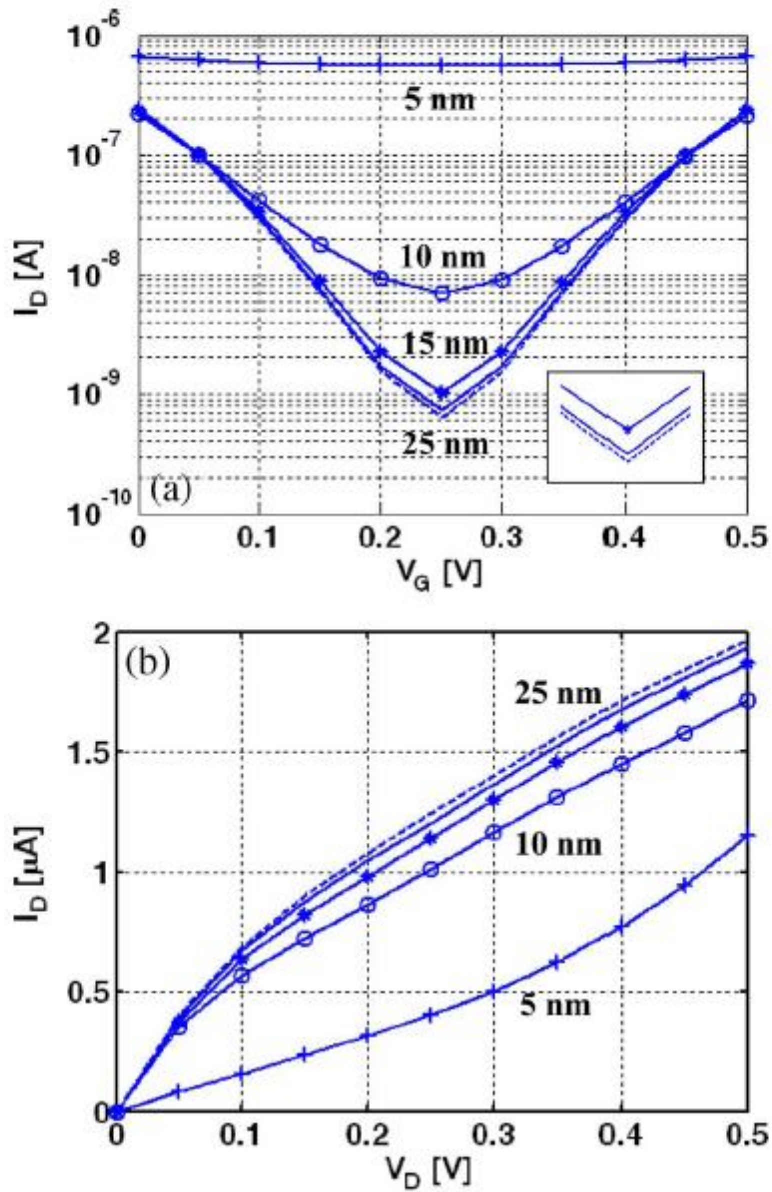


Figure 3.22: (a) Transfer characteristics of GNR SBFET (b) Output Characteristics of GNR SBFET with single gate geometry for different channel lengths studied by Yijiang Ouyang *et al* [167].

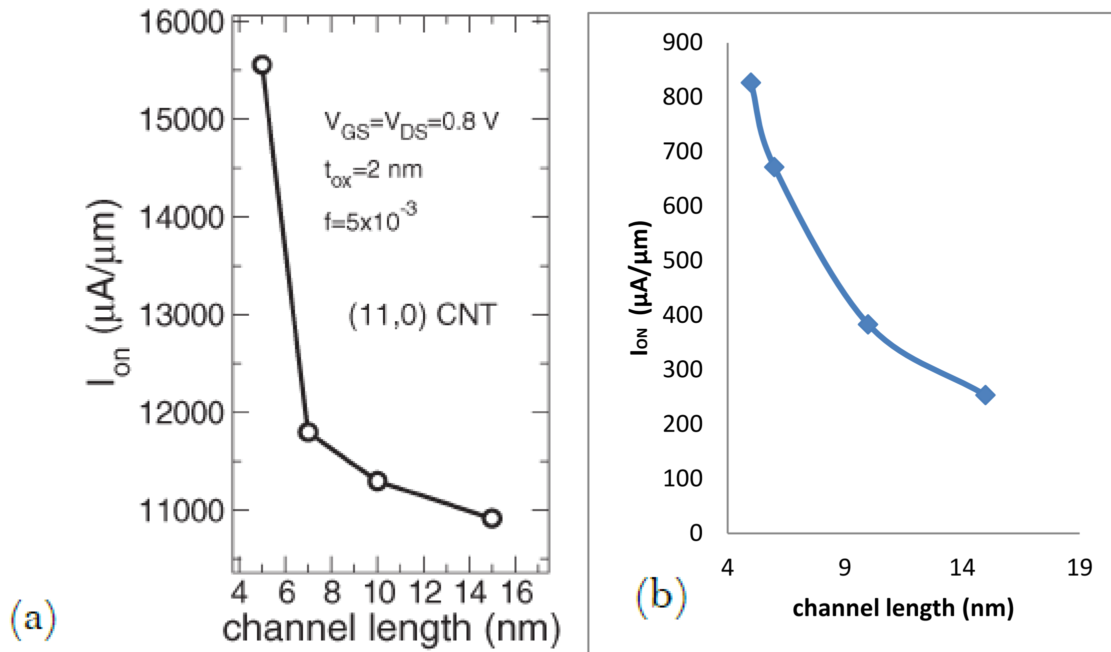


Figure 3.23: (a) ON state drain current as a function of channel length of CNT MOSFET [18]

(b) ON state drain current as a function of channel length of CNT SBFET.

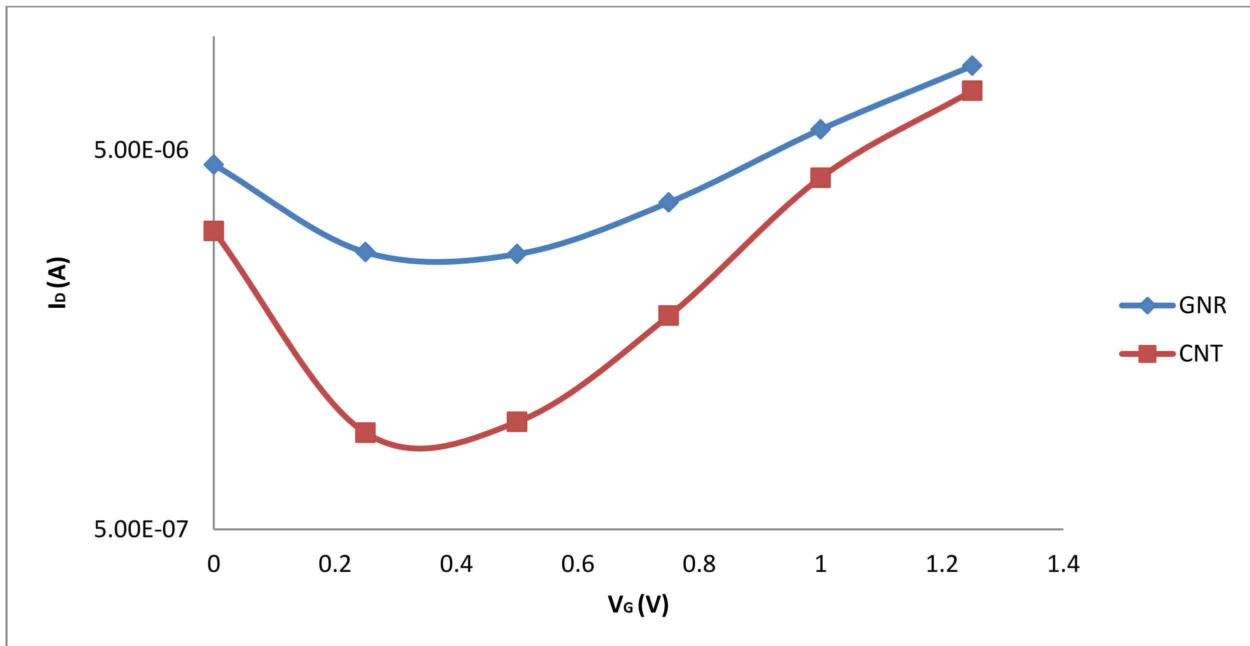


Figure 3.24: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for L_{ch} = 5nm at V_D = 0.5 V.

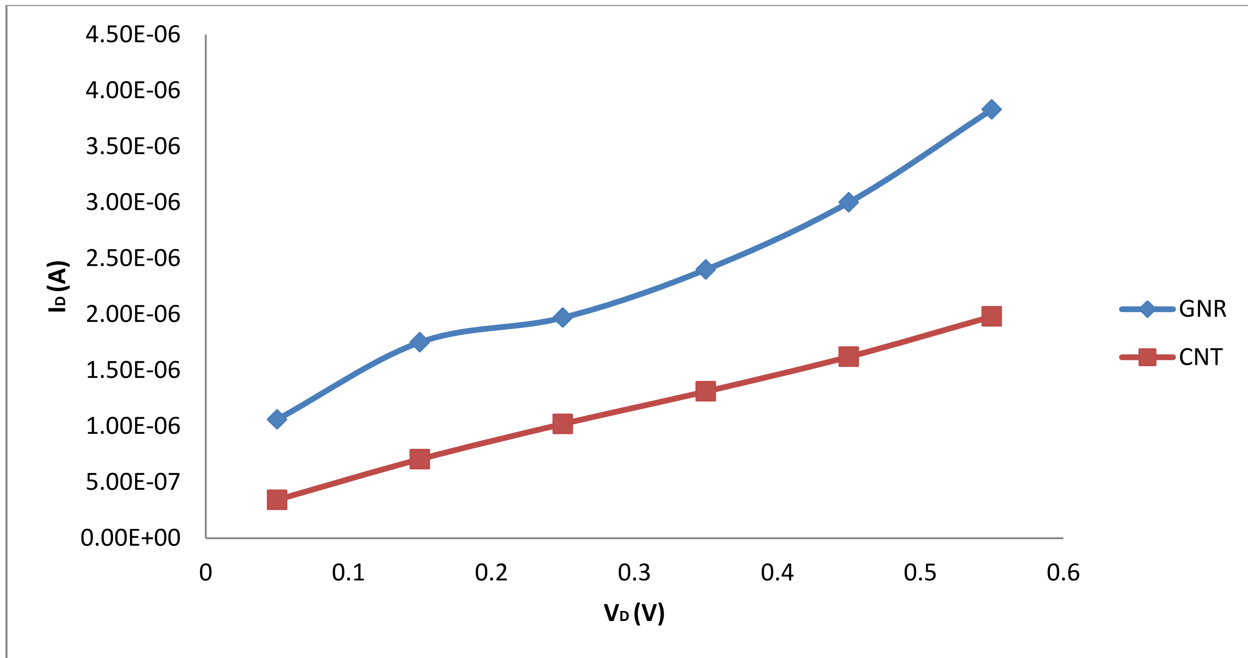


Figure 3.25: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for L_{ch} = 5nm at V_G = 0.5 V.

3.2.5 Effect of Gate Oxide thickness:

Gate oxide thickness has strong effect on GNR and CNT transistor performance. The gate oxide thickness is varied from 1nm to 2.5nm in increments of 0.5nm. The transfer and output characteristics of GNR SBFET and CNT SBFET obtained are shown in Fig 3.26 and Fig 3.27 respectively. In Fig 3.26, as gate voltage is increased till 0.25V, the transfer characteristics for both GNRFET and CNTFET with different gate oxide thickness are similar. As gate voltage is increased further, it is observed that the on-state drain current at any gate voltage is higher for lower gate oxide thickness. In Fig 3.27 (a), it is observed that the on-state drain current of GNRFET at any drain voltage is higher at lower values of gate oxide thickness. Here at V_D = 0.55V, GNRFET with t_{ox}=1nm has the highest current of 2.7×10^{-6} A. As gate oxide thickness is incremented further the drain current falls. In Fig 3.27(b), it is seen that the CNTFET with the lowest gate oxide thickness has the highest drain current and the current increases with decreasing gate oxide thickness. Thus, it can be concluded that the gate electrostatic control gets better and also, the drain current increases with decreasing gate oxide thickness.

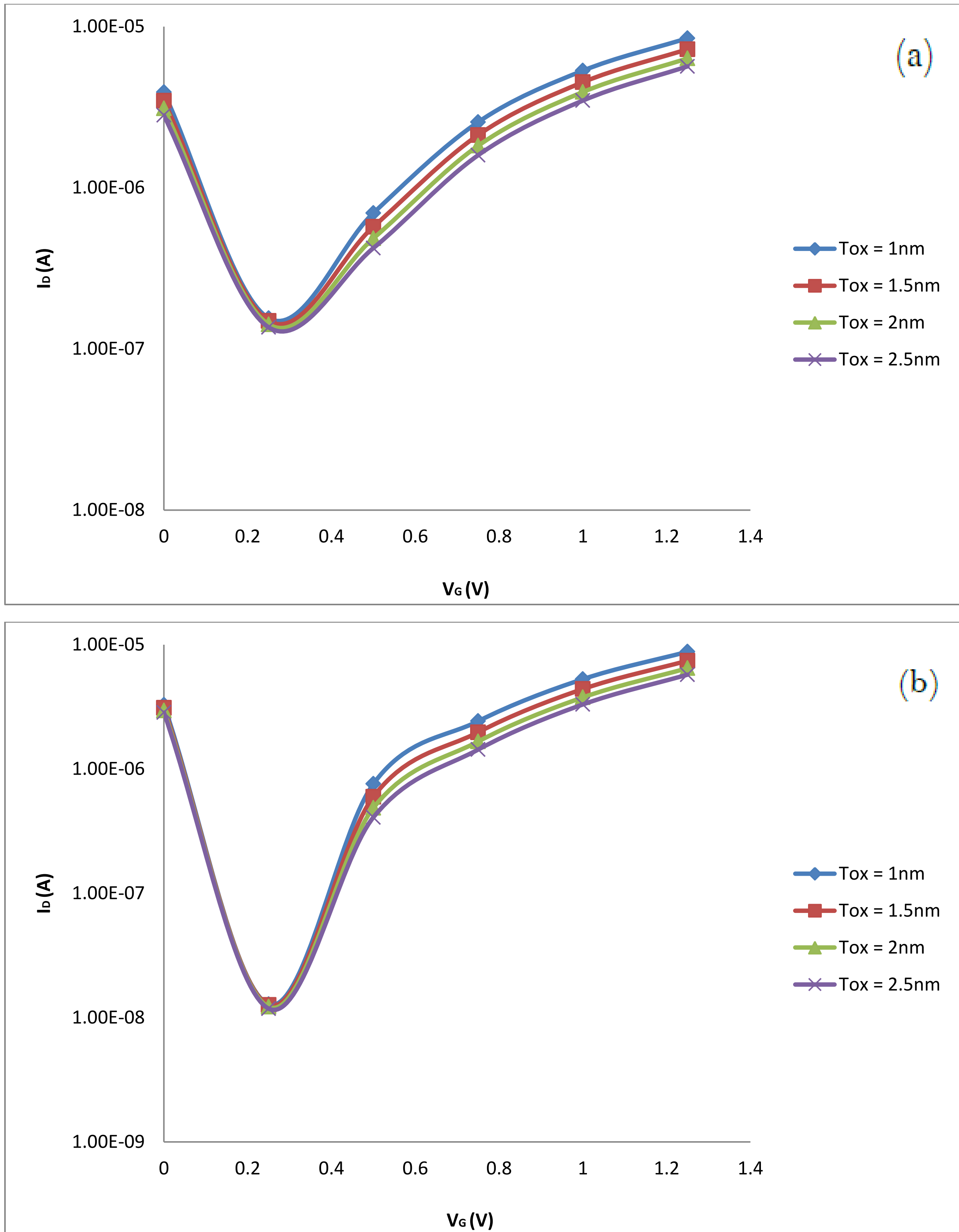
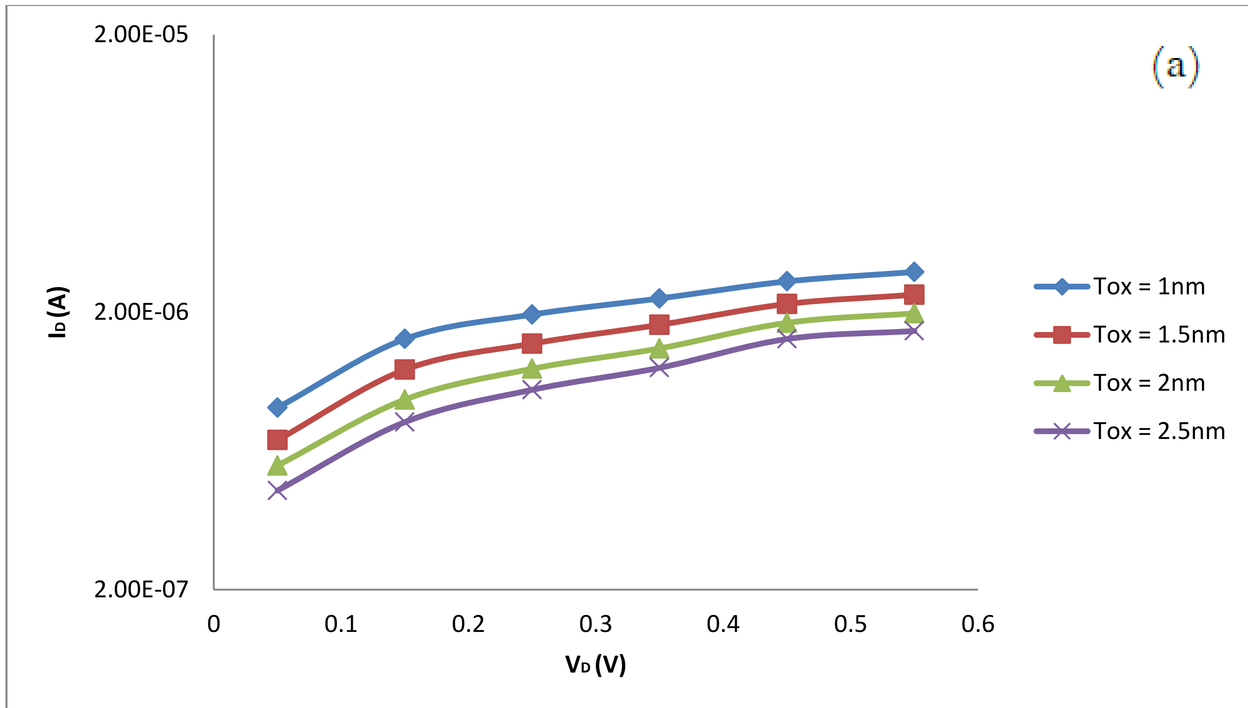


Figure 3.26: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different gate oxide thickness at $V_D = 0.5$ V.



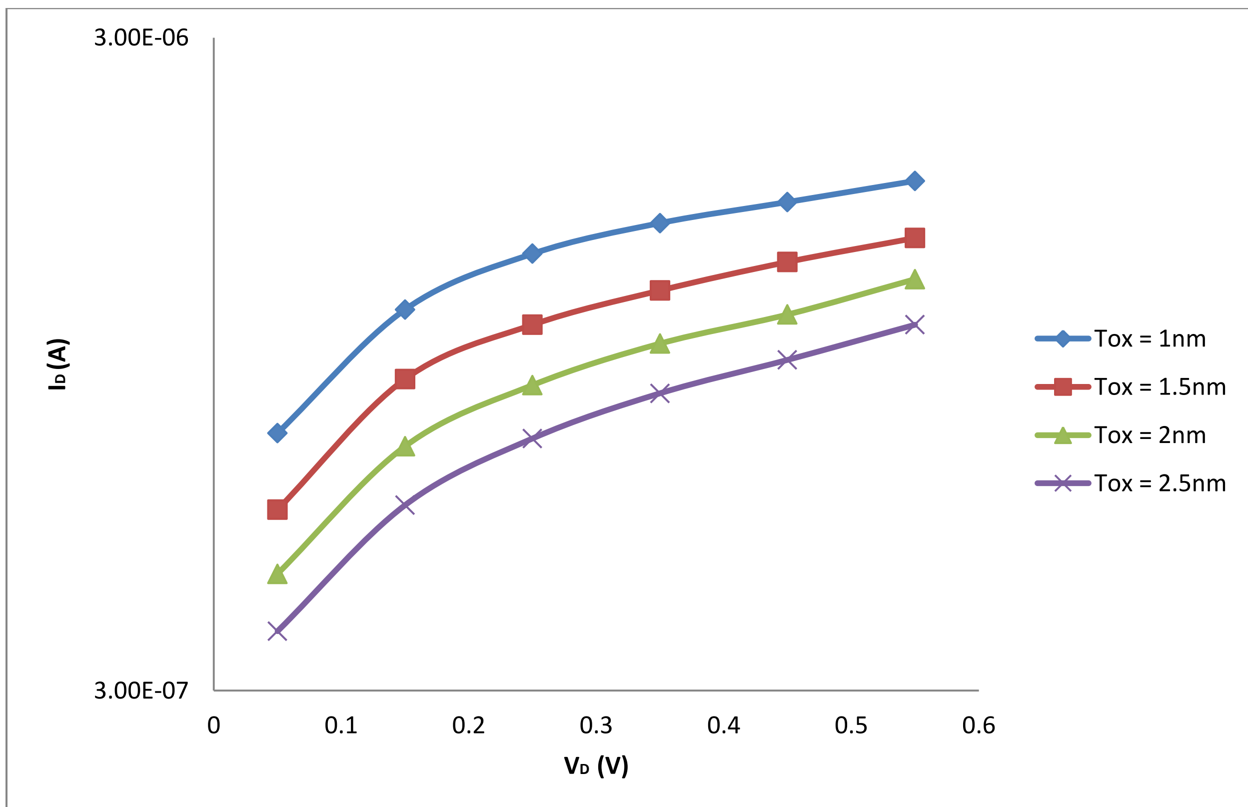


Figure 3.27: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for different gate oxide thickness at $V_G = 0.5$ V.

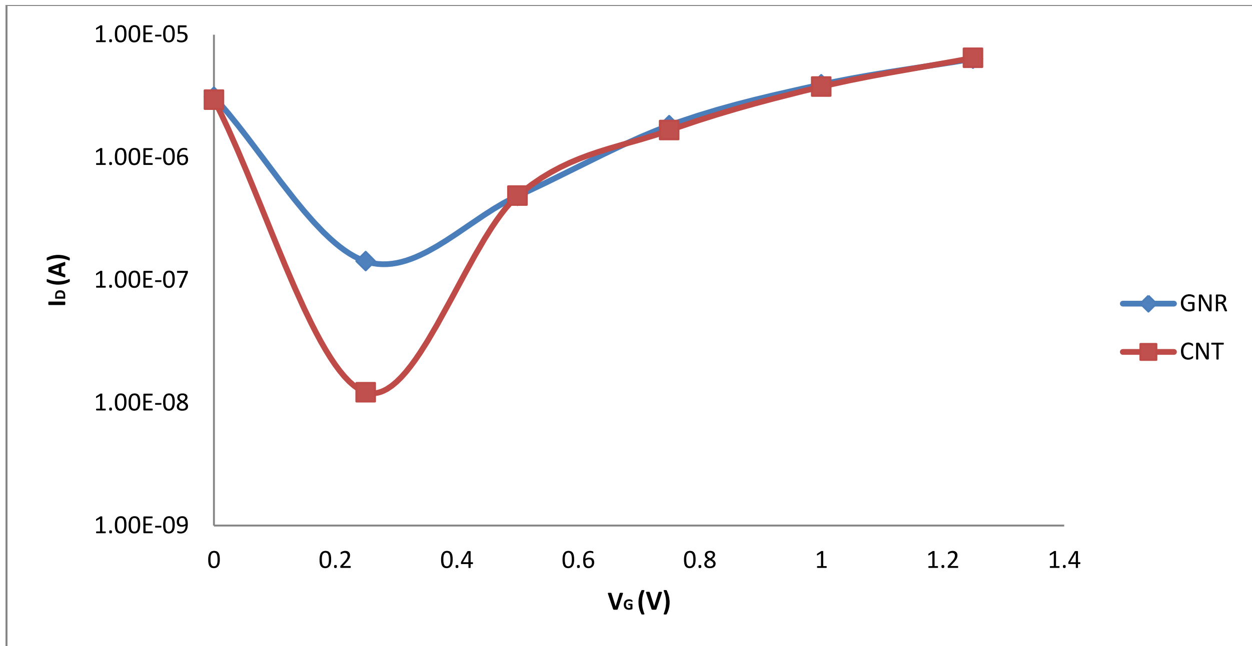


Figure 3.28: I_D vs. V_G characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for gate oxide thickness of 2nm at $V_D = 0.5$ V.

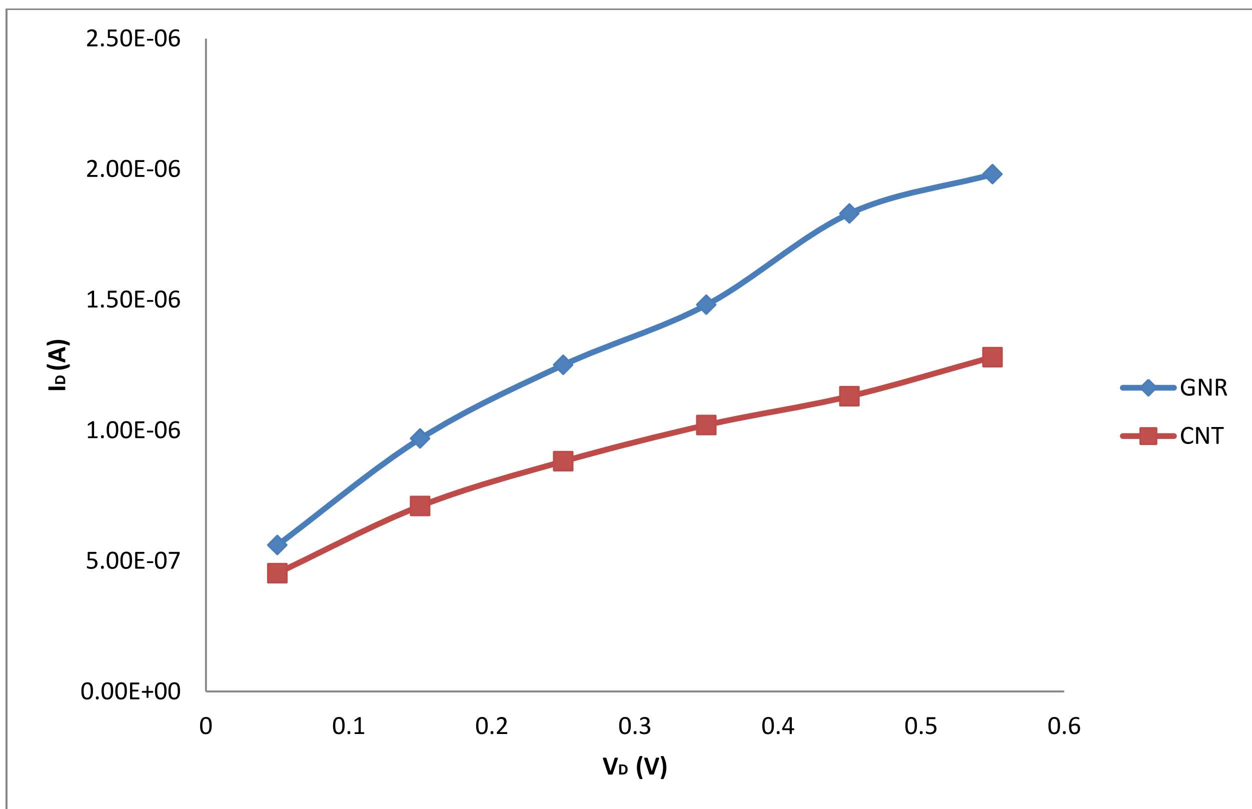


Figure 3.28: I_D vs. V_D characteristics of (a) Graphene Nanoribbon SBFET (b) Carbon Nanotube SBFET for gate oxide thickness of 2nm at $V_G = 0.5$ V.

3.3 Summary

This research primarily investigates the dependence of the transfer and output characteristics of schottky barrier GNRFET and CNTFET with respect to temperature, relative dielectric constant, chirality, channel length and gate oxide thickness. A comparative analysis of the transfer and output characteristics between the GNRFET and CNTFET is done in this chapter. Also, the results of temperature dependence of sub-10 nm Graphene Nanoribbon Array Field-Effect Transistors Fabricated by Block Copolymer Lithography are given in this paper and are compared to the temperature dependence of the GNRFET simulated in this research paper. The results of changing chirality of both GNRFET and CNTFET are compared with the results of two other groups and it is observed that both the results are similar. The transfer and output characteristics of the simulated GNRFET are compared to the results of another group and very little deviation is observed. For CNTFET, the on-state drain current per unit width as a function of channel length is compared to the results of a CNT MOSFET simulated by another group. It is seen that the the on-state current for a CNT MOSFET is higher than that of a CNT SBFET.

So, to summarize the results:

- The drain current increases only slightly with respect to temperature for both GNR SBFET and CNT SBFET.
- For both the transistors, the saturation current increases with respect to relative dielectric constant.
- For metallic GNRFET and CNTFET, gate voltage has very little control over the drain current
- Ambipolar behavior decreases with respect to diameter of CNT and width of GNR for semiconducting transistors.
- The drain current is directly proportional to the diameter (CNT) or width (GNR) for both semiconducting GNRFET and CNTFET,
- The drain current is inversely proportional to channel length for channel lengths of less than 10nm and is constant for higher values of channel length for both transistors.

- The drain current per unit width as a function of channel length of a CNT MOSFET is higher than the drain current per unit width as a function of channel length of CNT SBFET.
- For schottky barrier GNRFET, the on-state drain current is inversely proportional to the gate oxide thickness.
- For schottky barrier CNTFET, the on-state drain current is inversely proportional to the gate oxide thickness.
- GNRFET has comparatively higher drain current than CNTFET.

Chapter 4
CONCLUSION AND FUTURE WORK

4.1 Conclusion

Presently, graphene and carbon nanotube transistors are the most promising of all the nanoscale devices. In recent years plenty of research has been carried out on these materials and their devices. This has improved the understanding of these devices and thus has led to swift improvements in the performances of these devices. The limitations faced by current silicon based MOSFETs due to scaling has created the demand for a new range of devices and their application in logic circuit design which can provide better performance and also significantly. Ideal graphene and carbon transistors are capable of providing significant performance and energy benefits over traditional CMOS technology. However, such nanoscale devices faces problems like high defect rates and material variation due to the fundamental confinements of the fabrication techniques. Graphene and carbon nanotube display exceptional electronic properties like high mobility, and due to their nanoscale size it is possible to design devices with very high standards of performance.

In this research paper a detailed comparative analysis is carried out between schottky barrier graphene nanoribbon field-effect transistor and schottky barrier carbon nanotube field-effect transistor. A set of five different parameters are varied for each transistor in this analysis. The parameters changed are temperature, relative dieletric constant, chirality, channel length and gate oxide thickness. I_D vs. V_G curves and I_D vs. V_D curves are obtained for each parameter varied for each of the transistors and comparison is then made between the two transistors.

In the results part, the first analysis shows the effect of changing temperature on both types of transistors. Since we have assumed ballistic transport in our model, the transfer characteristics for different temperature is quite similar. Also, in the output characteristics for GNRFET and CNTFET a small increase in drain current is observed with increase in temperature. A higher drain current is seen for GNRFET compared to CNTFET at any temperature. Then, our model of GNRFET is compared to a FET based on GNR arrays patterned by Block Copolymer lithography.

In our second analysis, the effects of changing relative dielectric constant on GNRFET and CNTFET are shown. Both the transfer and output characteristics here show that the on-state drain current increases with respect to relative dielectric constant for both the FETs. The transfer characteristics for both the transistors are similar while in the output characteristics the graphene nanoribbon transistor has a higher drain current. From this analysis we can conclude that materials with higher relative dielectric constant will improve performance of both graphene nanoribbon and carbon nanotube transistors.

The third part of our result and analysis section compares the changing effects of chirality of the two transistors. It is seen that GNR and CNT can be metallic or semiconducting depending on their chirality. Also, the diameter increases with respect to chirality. For metallic GNR and CNT it is observed that gate voltage has only slight control over the drain current. For semiconducting FETs, it is observed that bandgap decreases with increasing diameter or width. As a result, drain current increases with respect to diameter or width. Lastly, the results obtained in this simulation are compared to the results of other research groups.

In the fourth comparison, the effect of changing channel length is studied. For channel lengths greater than 10nm, the transfer characteristics for both GNRFET and CNTFET are seen to be similar. For channel lengths less than 5nm, the drain current increases as the trasconductance decreases. The results of the GNRFET simulated in this research paper are compared to those of another research group who simulated a GNRFET with single gate geometry.

The effect of changing gate oxide thickness is shown in our last comparison. It is observed that the on-state drain current increases with decreasing gate oxide thickness for GNRFET. Also,m for CNTFET the drain current increases with respect to decreasing gate oxide thickness.

Lastly, by examining the results, it can be safely said that graphene nanoribbon and carbon nanotube field effect transistors show huge promise in the electronics industry and this devices can be used to usher in a new era in the world of electronics.

4.2 Future work:

There remain a large number of areas of graphene nanoribbon and carbon nanotube transistor modeling that can be explored further and which offer further scopes for improvement and development. The following would be the future prospects of this research:

- In future, the study can expand our study to the effects of varying transconductance and conductance of GNRFET and CNTFET with schottky contacts
- GNRFET and CNTFET with doped reservoirs can be modeled and simulate these transistors to make comparative analysis between these two transistors.
- In the future the performances of the GNRFET and CNTFET with schottky contacts and, GNRFET and CNTFET with doped reservoirs can be compared to determine which type of structure is better.
- Expanding comparison of GNRFET and CNTFET to traditional MOSFET to get a clearer picture of the performance of these transistors compared to MOSFET.

- Lastly, GNRFET and CNTFET can be moelle using different model physics and make simulation for each structure and make comparison between each model performances.

References

[1] Heng Chin Chuan, "Modeling and Analysis of Ballistic Carbon Nanotube Field Effect Transistor (CNTFET) with Quantum Transport Concept" MAY 2007, Unpublished.

[2] M P Anantram, and FL ´Léonard, "Physics of carbon nanotube electronic devices" Institute of Physics Publishing, Rep. Prog. Phys. 69,pp: 507–561, 2006.

[3] P. Avouris, J. Appenzeller, R. Martel, and S. J. Wind, "Carbon nanotube electronics", Proceedings of the IEEE, Vol. 91, pp. 1772-1784, 2003.

[4] P. L. McEuen, M. S. Fuhrer, and H. K. Park, "Single-walled carbon nanotube electronics", IEEE Transactions on Nanotechnology, Vol. 1, pp. 78-85, 2002.

[5] Zhu, Y., Murali, S., Cai, W., Li, X., Suk, J. W., Potts, J. R. and Ruoff, R. S. Graphene and Graphene Oxide: Synthesis, Properties, and Applications. Adv. Mater., 22: 3906–3924. doi: 10.1002/adma.201001068, 2010.

[6] D. Jiménez, X. Cartoixà, E. Miranda, J. Suñé, F. A. Chaves, and S. Roche, "A simple drain current model for Schottky-barrier carbon nanotube field effect transistors" Bercelona,Spain pp: 1-18, 2007.

[7] Rahmat Bin Sanudin, "Characterization of Ballistic Carbon Nanotube Field-Effect Transistor," Thesis report, University Technology Malaysia, November 2005.

[8] Compano, R. ed. "Technology Roadmap for Nanoelectronics," Microelectronics Advanced Research Initiative, 2000.

[9] David J. Frank et al, "Device Scaling Limits of Si MOSFETs and Their ApplicationDependencies," proceedings of the IEEE, vol. 89, no. 3, March 2001.

[10] Chen, Z. et. al. "IDDQ Testing for Deep-Submicron ICs: Challenges and Solutions," IEEE Design and Test of Computers.pp: 24-33, 2002.

[11] Max Schulz, "The end of the road for Silicon?", Nature, Vol 399, 24 June, 1999.

[12] Yuhua Cheng, Chenming Hu. " MOSFET classification and operation," MOSFET modeling & BSIM3 user's guide. Springer. p. 13. ISBN 0-7923-8575-6, 1999.

[13] James D. Plummer, Peter B. Griffin."Material and Process Limits in SiliconVLSI Technology," PROCEEDINGS OF THE IEEE, VOL. 89, NO. 3, MARCH 2001.

[14] Taur, Y., "CMOS design near the limit of scaling," IBM J. Ref. & Dev.,Volume: 46(2):pp:213-220, 2002.

[15] T Surukai, "Perspectives on power aware electronics", Keynote presentation, F, ISSCC Conference, pp.26-29, 2003.

[16] John L. Hennessy and David A. Patterson "Computer Architecture, A Qualitative Approach", Ed. 3, pp. 14, 2003.

[17] Shazia Hassan et al, "Limitation of Silicon Based Computation and Future Prospects" Second International Conference on Communication Software and Networks, pp. 559-561, 2010.

[18] R. Bruce, S. Subramoney., "Carbon Nanotubes," summer, The Electrochemical Society Interface, 2006.

[19] T. W. Ebbesen., "Carbon Nanotubes," American Institute of Physics, June, 1996.

[20] J.Seetharamappa et al., "Carbon Nanotubes Next Generation of Electronic Materials," summer, The Electrochemical Society Interface, 2006.

[21] T.Grace, "An Introduction to Carbon Nanotubes," summer, Stanford University, 2003.

[22] Xiaolei Liu, "Synthesis, Devices and Applications of Carbon Nanotubes," Thesis report, University of Southern California, January 2006.

[23] Paul Holister, Tim Harper and Christina Roman Vas, "Nanotubes White Paper," CMP Cientifica, January 2003.

[24] Xiaolei Liu,"Synthesis, Devices and Applications of Carbon Nanotubes," Thesis report, University of Southern California, January 2006.

[25] J.Seetharamappa et al., "Carbon Nanotubes Next Generation of Electronic Materials," summer, The Electrochemical Society Interface, 2006.

[26] M. S. Dresselhaus, G. Dresselhaus, Ph. Avouris,"Carbon nanotubes," Topics in Applied Physics, Springer-Verlag, Berlin, 2001.

[27] T. W. Odom, J. L. Huang et al.,"Atomic structure and electronic properties of single-walled Carbon Nanotubes," Nature, Volume 391, January, 1998.

[28] P.R. Bandaru., "Electrical Properties and Applications of Carbon Nanotube Structures," Journal of Nanoscience and Nanotechnology Vol.7, 1–29, 2007.

[29] S.Datta, "Electronic Transport in Mesoscopic Systems," Cambridge University Press, New York 1995.

[30] Tans, S. J. et. al. "Individual single-wall carbon nanotubes as quantum wires," Nature. Volume 386(3), pp: 474-477, 1997.

[31] Forro, L. and Schoenberger, C. "Physical Properties of Multi-wall Nanotubes", Berlin, Germany: Springer-Verlag Berlin Heidelberg, pp: 329-390; 2001.

[32] Rasmita Sahoo and R. R. Mishra, "Simulations of Carbon Nanotube Field EffectTransistors," International Journal of Electronic Engineering Research ISSN 0975- 6450 Volume 1 Number 2, pp. 117–125, 2009.

[33] Avouris et al., "Vertical scaling of carbon nanotube field-effect transistors using top gate electrodes," Applied Physics Letters, 2002.

[34] Choi, W. B. et. al. "Aligned carbon nanotubes for nanoelectronics," Institute of Physics Publishing, Volume: 15, pp: 512-516, 2002.

[35] Derycke, V. et. al. "Carbon Nanotube Inter- and Intramolecular Logic Gates," Nano Letters, Volume:1(9), pp: 453-456, 2001.

[36] W. Shockley and G.L. Pearson, Phys. Rev. 74, 232, 1948.

[37] D. Kahng and M.M. Atalla, Silicon-Silicon Dioxide Field Induced Surface Devices, IRE Solid-State Device Research Conference, Carnegie Institute of Technology, Pittsburgh, PA, 1960.

[38] S.J. Tans, A.R.M. Verschueren, and C. Dekker, Nature 393, 49, 1998.

[39] Andronico, Michael (14 April 2014). "5 Ways Graphene Will Change Gadgets Forever". Laptop.

[40] "Graphene properties". www.graphene-battery.net. 2014-05-29. Retrieved 2014-05-29.

[41] "This Month in Physics History: October 22, 2004: Discovery of Graphene".APS News. Series II 18 (9): 2. 2009.

[42] Wu, Yudong. Simulation of graphene electronic devices. Ph.D. thesis, University of Birmingham, 2011.

[43] "The Nobel Prize in Physics 2010 - Advanced Information". Nobelprize.org.Nobel Media AB 2014. Web. 22 Aug 2014. http://www.nobelprize.org/nobel_prizes/physics/laureates/2010/advanced.html

[44] A. K. Geim and K. S. Novoselov, Nature Materials 6, 183, 2007.

[45] D. R. Dreyer , R. S. Ruoff , C. W. Bielawski , Angew. Chem. Int. Ed.2010 , 49 , 9336.

[46] M. S. Dresselhaus , G. Dresselhaus , Adv. Phys. 1981 , 30 , 139.

[47] V. H. P. Boehm , A. Clauss , G. O. Fischer , U. Hofmann , Z. Naturforschg.1962 , 17 , 150 .

[48] P. M. Stefan , M. L. Shek , I. Lindau , W. E. Spicer , L. I. Johansson , F. Herman , R. V. Kasowski , G.Brogen , Phys. Rev. B 1984 , 29 , 5423 .

[49] T. Aizawa , R. Souda , S. Otani , Y. Ishizawa , C. Oshima , Phys. Rev. Lett. 1990 , 64 , 768 .

[50] A. Nagashima , K. Nuka , H. Itoh , T. Ichinokawa , C. Oshima , S. Otani , Surf. Sci. 1993 , 291 , 93 .

[51] Chuhei Oshima et al 1977 Jpn. J. Appl. Phys. 16 965. doi:10.1143/JJAP.16.965

[52] R. Rosei , M. De Crescenzi , F. Sette , C. Quaresima , A. Savoia ,P. Perfetti , Phys. Rev. B 1983 , 28 , 1161 .

[53] N. A. Kholin , E. V. Rut'kov , A. Y. Tontegode , Surf. Sci. 1984 , 139 , 155.

[54] N. R. Gall , S. N. Mikhailov , E. V. Rut'kov , A. Y. Tontegode , Sov. Phys.Solid State 1985 , 27 , 1410.

[55] H. Zi-Pu , D. F. Ogletree , M. A. Van Hove , G. A. Somorjai , Surf. Sci.1987 , 180 , 433 .

[56] A. J. Van Bommel , J. E. Crombeen , A. Van Tooren , Surf. Sci. 1975 , 48 , 463 .

[57] Y. Zhang , J. Small , W. Pontius , P. Kim , Appl. Phys. Lett. 2005 , 86 ,073104 .

[58] D. D. L. Chung , J. Mater. Sci. 1987 , 22 , 4190 .

[59] X. Lu , M. Yu , H. Huang , R. Ruoff , Nanotechnology 1999 , 10 , 269 .

[60] X. Du , I. Skachko , A. Barker , E. Y. Andrei , Nat. Nanotechnol. 2008 , 3 , 491 .

[61] S. V. Morozov , K. S. Novoselov , M. I. Katsnelson , F. Schedin , D. C. Elias , J. A. Jaszczak , A. K. Geim , Phys. Rev. Lett. 2008 , 100 , 016602.

[62] E. V. Castro , H. Ochoa , M. I. Katsnelson , R. V. Gorbachev , D. C. Elias , K. S. Novoselov , A. K. Geim , F. Guinea , Phys. Rev. Lett. 2010 , 105 , 266601.

[63] A. A. Balandin , S. Ghosh , W. Bao , I. Calizo , D. Teweldebrhan , F. Miao , C. N. Lau , Nano Lett. 2008 , 8 , 902.

[64] C. Lee , X. Wei , J. W. Kysar , J. Hone , Science 2008 , 321 , 385.

[65] S. Berciaud , S. Ryu , L. E. Brus , T. F. Heinz , Nano Lett. 2009 , 9 , 346.

[66] C. Soldano , A. Mahmood , E. Dujardin , Carbon 2010 , 48 , 2127.

[67] X. L. Li , G. Y. Zhang , X. D. Bai , X. M. Sun , X. R. Wang , E. G. Wang , H. J. Dai , Nat. Nanotechnol. 2008 , 3 , 538 .

[68] Y. Hernandez , V. Nicolosi , M. Lotya , F. M. Blighe , Z. Sun , S. De , I. T. McGovern , B. Holland , M. Byrne , Y. K. Gun'ko , J. J. Boland , P. Niraj , G. Duesberg , S. Krishnamurthy , R. Goodhue , J. Hutchison , V. Scardaci , A. C. Ferrari , J. N. Coleman , Nat. Nanotechnol. 2008 , 3 , 563 .

[69] U. Khan , A. O'Neill , M. Lotya , S. De , J. N. Coleman , Small 2010 , 6 , 864 .

[70] C. Valles , C. Drummond , H. Saadaoui , C. A. Furtado , M. He , O. Roubeau , L. Ortolani , M. Monthioux , A. Penicaud , J. Am. Chem. Soc. 2008 , 130 , 15802 .

[71] J. S. Moon , D. Curtis , M. Hu , D. Wong , C. McGuire , P. M. Campbell , G. Jernigan , J. L. Tedesco , B. VanMil , R. Myers-Ward , C. J. Eddy , D. K. Gaskill , IEEE Electron. Device Lett. 2009 , 30 , 650 .

[72] O. C. Compton , S. T. Nguyen , Small 2010 , 6 , 711 .

[73] Y. Si , E. T. Samulski , Nano Lett. 2008 , 8 , 1679 .

[74] D. R. Dreyer , S. Park , C. W. Bielawski , R. S. Ruoff , Chem. Soc. Rev. 2010 , 39 , 228 .

[75] S. Park , J. An , I. Jung , R. D. Piner , S. J. An , X. Li , A. Velamakanni , R. S. Ruoff , Nano Lett. 2009 , 9 , 1593 .

[76] M. A. Rafi ee , J. Rafi ee , Z. Wang , H. Song , Z.-Z. Yu , N. Koratkar , ACS Nano 2009 , 3 , 3884 .

[77] Y. Shao , J. Wang , M. Engelhard , C. Wang , Y. Lin , J. Mater. Chem. 2010 , 20 , 743 .

[78] L. J. Cote , R. Cruz-Silva , J. Huang , J. Am. Chem. Soc. 2009 , 131 , 1 1027 .

[79] V. Strong , S. Dubin , M. F. El-Kady , A. Lech , Y. Wang , B. H. Weiller , R. B. Kaner , ACS Nano 2012 , 6 , 1395 .

[80] Y. Dedkov , A. Shikin , V. Adamchuk , S. Molodtsov , C. Laubschat , A. Bauer , G. Kaindl , Phys. Rev. B 2001 , 64 , 035405

[81] P. W. Sutter , J.-I. Flege , E. A. Sutter , Nat. Mater. 2008 , 7 , 406 .

[82] T. A. Land , T. Michely , R. J. Behm , J. C. Hemminger , G. Comsa , Surf. Sci. 1992 , 264 , 261 .

[83] Y. S. Dedkov , M. Fonin , C. Laubschat , Appl. Phys. Lett. 2008 , 92 , 052506 .

[84] A. T. N'Diaye , J. Coraux , T. N. Plasa , C. Busse , T. Michely , New J. Phys. 2008 , 10 , 043033 .

[85] Y. Wang , C. Miao , B.-c. Huang , J. Zhu , W. Liu , Y. Park , Y.-h. Xie , J. C. S. Woo , IEEE Trans. Electron Devices 2010 , 57 , 3472 .

[86] W. Cai , R. D. Piner , Y. Zhu , X. Li , Z. Tan , H. C. Floresca , C. Yang , L. Lu , M. J. Kim , R. S. Ruoff , Nano Res. 2009 , 2 , 851 .

[87] W. Liu , C.-H. Chung , C.-Q. Miao , Y.-J. Wang , B.-Y. Li , L.-Y. Ruan , K. Patel , Y.-J. Park , J. Woo , Y.-H. Xie , Thin Solid Films 2010 , 518 , S128 .

[88] A. Reina , X. Jia , J. Ho , D. Nezich , H. Son , V. Bulovic , M. S. Dresselhaus , J. Kong , Nano Lett. 2009 , 9 , 30 .

[89] G. Lopez , E. Mittemeijer , Scr. Mater. 2004 , 51 , 1 .

[90] G. L. Selman , P. J. Ellison , A. s. Darling , Platinum Metals Rev. 1970 , 14 , 14 .

[91] S. Bhaviripudi , X. Jia , M. S. Dresselhaus , J. Kong , Nano Lett. 2010 , 10 , 4128 .

[92] J. Wu , W. Pisula , K. Muellen , Chem. Rev. 2007 , 107 , 718 .

[93] X. Yang , X. Dou , A. Rouhanipour , L. Zhi , H. J. Rader , K. Müllen , J. Am. Chem. Soc. 2008 , 130 , 4216 .

[94] J. Cai , P. Ruffi eux , R. Jaafar , M. Bieri , T. Braun , S. Blankenburg , M. Muoth , A. P. Seitsonen , M. Saleh , X. Feng , K. Muellen , R. Fasel , Nature 2010 , 466 , 470 .

[95] X. Yan , X. Cui , L.-s. Li , J. Am. Chem. Soc. 2010 , 132 , 5944 .

[96] X. Yan , X. Cui , B. Li , L.-s. Li , Nano Lett. 2010 , 10 , 1869 .

[97] Kusmartsev, F. V.; Wu, W. M.; Pierpoint, M. P.; Yung, K. C. . "Application of Graphene within Optoelectronic Devices and Transistors", 2014. arXiv:1406.0809 [cond-mat.mtrl-sci].

[98] Meyer, J.; Geim, A. K.; Katsnelson, M. I.; Novoselov, K. S.; Booth, T. J.; Roth, S. "The structure of suspended graphene sheets". Nature 446 (7131): 60–63, 2007. arXiv:cond-mat/0701379.Bibcode:2007Natur.446...60M. doi:10.1038/nature05545. PMID 17330039.

[99] Neto, A Castro; Peres, N. M. R.; Novoselov, K. S.; Geim, A. K.; Geim, A. K.. "The electronic properties of graphene" (PDF). Rev Mod Phys 81: 109, 2009.

[100] Charlier, J.-C.; Eklund, P.C.; Zhu, J.; Ferrari, A.C. "Electron and Phonon Properties of Graphene: Their Relationship with Carbon Nanotubes". In Jorio, A.; Dresselhaus and, G.; Dresselhaus, M.S. Carbon Nanotubes: Advanced Topics in the Synthesis, Structure, Properties and Applications (Berlin/Heidelberg: Springer-Verlag), 2008.

[101] Semenoff, G. W. "Condensed-Matter Simulation of a Three-Dimensional Anomaly".Physical Review Letters 53 (26): 2449, 1984.

[102] Avouris, P.; Chen, Z.; Perebeinos, V. "Carbon-based electronics". Nature Nanotechnology2 (10):605-15, 2007.

[103] Semenoff, G. W. "Condensed-Matter Simulation of a Three-Dimensional Anomaly".Physical Review Letters 53 (26): 2449, 1984.

[104] Lamas, C.A.; Cabra, D.C.; Grandi, N. "Generalized Pomeranchuk instabilities in graphene".Physical Review B 80 (7): 75108, 2009.

[105] Lamas, C.A.; Cabra, D.C.; Grandi, N. "Generalized Pomeranchuk instabilities in graphene".Physical Review B 80 (7): 75108, 2009.

[106] Geim, A. K.; Novoselov, K. S. "The rise of graphene". Nature Materials 6 (3): 183–91, 2007.

[107] Novoselov, K. S.; Geim, A. K.; Morozov, S. V.; Jiang, D.; Katsnelson, M. I.; Grigorieva, I. V.; Dubonos, S. V.; Firsov, A. A. "Two-dimensional gas of massless Dirac fermions in graphene".Nature 438 (7065): 197–200, 2005.

[108] Morozov, S.V.; Novoselov, K.; Katsnelson, M.; Schedin, F.; Elias, D.; Jaszczak, J.; Geim, A. "Giant Intrinsic Carrier Mobilities in Graphene and Its Bilayer". Physical Review Letters 100 (1): 016602, 2008.

[109] Chen, J. H.; Jang, Chaun; Xiao, Shudong; Ishigami, Masa; Fuhrer, Michael S. "Intrinsic and Extrinsic Performance Limits of Graphene Devices on SiO2". Nature Nanotechnology 3 (4): 206–9, 2008.

[110] Akturk, A.; Goldsman, N. "Electron transport and full-band electron–phonon interactions in graphene". Journal of Applied Physics 103 (5): 053702, 2008.

[111] Kusmartsev, F. V.; Wu, W. M.; Pierpoint, M. P.; Yung, K. C. "Application of Graphene within Optoelectronic Devices and Transistors", 2014.

[112] Physicists Show Electrons Can Travel More Than 100 Times Faster in Graphene :: University Communications Newsdesk, University of Maryland. Newsdesk.umd.edu (2008-0324).

[113] Jens Baringhaus et al., Exceptional ballistic transport in epitaxial graphene nanoribbons, Nature, 2014, DOI: 10.1038/nature12952

[114] Baringhaus, J.; Ruan, M.; Edler, F.; Tejeda, A.; Sicot, M.; Taleb-Ibrahimi, A.; Li, A. P.; Jiang, Z.; Conrad, E. H.; Berger, C.; Tegenkamp, C.; De Heer, W. A. "Exceptional ballistic transport in epitaxial graphene nanoribbons". Nature 506 (7488): 349, 2014.

[115] Chen, J. H.; Jang, C.; Adam, S.; Fuhrer, M. S.; Williams, E. D.; Ishigami, M. "Charged Impurity Scattering in Graphene". Nature Physics 4 (5): 377–381, 2008.

[116] A. J. Frenzel, C. H. Lui, Y. C. Shin, J. Kong, N. Gedik, Semiconducting-to-Metallic Photoconductivity Crossover and Temperature-Dependent Drude Weight in Graphene, Physical Review Letters, 2014, DOI: 10.1103/PhysRevLett.113.056602

[117] Schedin, F.; Geim, A. K.; Morozov, S. V.; Hill, E. W.; Blake, P.; Katsnelson, M. I.; Novoselov, K. S. "Detection of individual gas molecules adsorbed on graphene". Nature Materials 6 (9): 652–655, 2007.

[118] Adam, S.; Hwang, E. H.; Galitski, V. M.; Das Sarma, S. "A self-consistent theory for graphene transport". Proc. Nat. Acad. Sci. USA 104 (47): 18392–7, 2007.

[119] Steinberg, Hadar; Barak, Gilad; Yacoby, Amir; et al. "Charge fractionalization in quantum wires (Letter)". Nature Physics 4 (2): 116–119, 2008.

[120] Trisetyarso, Agung. "Dirac four-potential tunings-based quantum transistor utilizing the Lorentz force". Quantum Information & Computation 12 (11–12): 989, 2012.

[121] Chen, Shanshan; Wu, Qingzhi; Mishra, Columbia; Kang, Junyong; Zhang, Hengji; Cho, Kyeongjae; Cai, Weiwei; Balandin, Alexander A.; Ruoff, Rodney S. "Thermal conductivity of isotopically modified graphene". Nature Materials (2012-01-10) 11 (3): 203, 2012.

[122] Y.-W. Son, M. L. Cohen, S. G. Louie, "Energy gaps in graphene nanoribbons", Phys. Rev. Lett. Vol. 97, n. 21, 216803-1-4, 2006.

[123] G. Iannaccone, G. Fiori, M. Macucci, P. Michetti, M. Cheli, A. Betti, P. Marconcini, "Perspectives of graphene nanoelectronics: probing technological options with modeling", IEDM 2009, 7-9 December, Baltimore, USA.

[124] E. McCann, V. I. Fal'ko, "Landau-level degeneracy and quantum Hall effect in a graphite layer", Phys. Rev. Lett. Vol. 96, n. 8, 086805-1-4, 2006.

[125] E.V. Castro, K. S. Novoselov, V. Morozov, N.M.R. Peres, J.M.B. Lopes dos Santos, J. Nilsson, F. Guinea, A. K. Geim, and A. H. Castro Neto, “Biased bilayer graphene: Semiconductor with a gap tunable by the electric field effect”, Phys. Rev. Lett., Vol. 99, p. 216902, 2007.

[126] T. Ohta et al., “Controlling the electronic structure of bilayer graphene”, Science, Vol. 313, n. 5789, pp. 951-954, 2008.

[127] Y. Zhang, T.T. Tang, C. Girit, Z. Hao, M. C. Martin, A. Zettl, M. F. Crommie, Y. R. Shen, F. Wang, "Direct observation of a widely tunable bandgap in bilayer graphene", Nature,Vol. 459, p. 820-823, 2009.

[128] S. Y. Zhou et al., “Substrate-induced bandgap opening in epitaxial graphene”, Nat. Mat., Vol. 6, n. 10, p. 770, 2007.

[129] M. Cheli, P. Michetti, G. Iannaccone, “Physical Insights on Nanoscale FETs based on epitaxial graphene on SiC”, Proc. ESSDERC 2009, 193-196, 2009.

[130] T. J. Echtermeyer, M. C. Lemme, M. Baus, B. N. Szafranek, A. K. Geim, H. Kurz, “Nonvolatile Switching in Graphene Field-Effect Devices”, IEEE EDL, Vo. 29, pp. 952-954, 2009.

[131] T. J. Echtermeyer, M. C. Lemme, M. Baus, B. N. Szafranek, A. K. Geim, H. Kurz, “Nonvolatile Switching in Graphene Field-Effect Devices”, IEEE EDL, Vo. 29, pp. 952-954, 2009.

[132] Fodor, J. K. Simulations involving carbon nanotubes and nanoribbons. Gainesville, Fla.: University of Florida, 2007.

[133] Nihar Mohanty, David Moore, Zhiping Xu, T. S. Sreeprasad, Ashvin Nagaraja, Alfredo A. Rodriguez and Vikas Berry. "Nanotomy Based Production of Transferrable and Dispersible Graphene-Nanostructures of Controlled Shape and Size". Nature Communications 3 (5): 844, 2012.

[134] Brumfiel, G. "Nanotubes cut to ribbons New techniques open up carbon tubes to create ribbons", 2009.

[135] Kosynkin, Dmitry V.; Higginbotham, Amanda L.; Sinitskii, Alexander; Lomeda, Jay R.; Dimiev, Ayrat; Price, B. Katherine; Tour, James M. "Longitudinal unzipping of carbon nanotubes to form graphene nanoribbons". Nature 458 (7240): 872-6, 2009.

[136] Liying Jiao, Li Zhang, Xinran Wang, Georgi Diankov & Hongjie Dai. "Narrow graphene nanoribbons from carbon nanotubes". Nature 458 (7240): 877–80, 2009.

[137] "Writing Graphene Circuitry With Ion 'Pens'". ScienceDaily. Mar 27, 2012.

[138] "AIP's Physics News Highlights March 27, 2012". American Institute of Physics (AIP). 2012-03-28.

[139] S. Tongay, M. Lemaitre, J. Fridmann, A. F. Hebard, B. P. Gila, and B. R. Appleton. "Drawing graphene nanoribbons on SiC by ion implantation".Appl. Phys. Lett. 100 (073501), 2012.

[140] Barone, V., Hod, O., and Scuseria, G. E. "Electronic Structure and Stability of Semiconducting Graphene Nanoribbons". Nano Letters 6 (12): 2748–54, 2006.

[141] Han., M.Y., Özyilmaz, B., Zhang, Y., and Kim, P. "Energy Band-Gap Engineering of Graphene Nanoribbons". Physical Review Letters 98 (20), 2007.

[142] Tapasztó, Levente; Dobrik, Gergely; Lambin, Philippe; Biró, László P. "Tailoring the atomic structure of graphene nanoribbons by scanning tunnelling microscope lithography". Nature Nanotechnology 3 (7): 397–401, 2008.

[143] Son Y.-W., Cohen M. L., and Louie S. G. "Energy Gaps in Graphene Nanoribbons". Physical Review Letters 97 (21), 2006.

[144] L. F. Huang, G. R. Zhang, X. H. Zheng, P. L. Gong, T. F. Cao, and Z. Zeng. "Understanding and tuning the quantum-confinement effect and edge magnetism in zigzag graphene nanoribbon". J. Phys.: Condens. Matter 25 (5): 055304, 2013.

[145] Fiori G., Iannaccone G. "Simulation of Graphene Nanoribbon Field-Effect Transistors". IEEE Electron Device Letters 28 (8): 760, 2007.

[146] Gianluca Fiori; Giuseppe Iannaccone, "NanoTCAD ViDES," https://nanohub.org/resources/vides, 2009. (DOI: 10.4231/D3RJ48T8X).

[147] Wang, Z. F., Shi, Q. W., Li, Q., Wang, X., Hou, J. G., Zheng, H., Yao, Y., Chen, J. "Z-shaped graphene nanoribbon quantum dot device". Applied Physics Letters 91 (5): 053109, 2007.

[148] Bullis, Kevin. "Graphene Transistors". Technology Review (Cambridge: MIT Technology Review, Inc), 2008-01-28.

[149] Bullis,Kevin. "TR10: Graphene Transistors". Technology Review(Cambridge: MIT Technology Review, Inc), 2008-02-25.

[150] Wang, Xinran; Ouyang, Yijian; Li, Xiaolin; Wang, Hailiang; Guo, Jing; Dai, Hongjie. "Room-Temperature All-Semiconducting Sub-10-nm Graphene Nanoribbon Field-Effect Transistors". *Physical Review Letters* 100 (20), 2008. arXiv:0803.3464

[151] Jing; Dai, Hongjie. "Room-Temperature All-Semiconducting Sub-10-nm Graphene Nanoribbon Field-Effect Transistors". Physical Review Letters 100 (20), 2008.

[152] Ballon, M. S. Carbon nanoribbons hold out possibility of smaller, speedier computer chips. Stanford Report, 2008-05-28.

[153] Schwierz F. Graphene transistors. Nat. Nanotechnol. 5, 487–496, 2010.

[154] "The Nobel Prize in Physics 2010 - Advanced Information". Nobelprize.org.Nobel Media AB 2014. Web. 22 Aug 2014. http://www.nobelprize.org/nobel_prizes/physics/laureates/2010/advanced.html

[155] Gianluca Fiori; Giuseppe Iannaccone, "NanoTCAD ViDES," https://nanohub.org/resources/vides, 2009. (DOI: 10.4231/D3RJ48T8X).

[156] G. Fiori, G. Iannaccone, "Simulation of Graphene Nanoribbon Field-Effect Transistors", IEEE, Electron Device Letters, Vol. 28, Issue 8, pp. 760 - 762, 2007.

[157] G. Fiori, G. Iannaccone, G. Klimeck, "A Three-Dimensional Simulation Study of the Performance of Carbon Nanotube Field-Effect Transistors With Doped Reservoirs and Realistic Geometry", IEEE Transaction on Electron Devices, Vol. 53, Issue 8, pp. 1782-1788, 2006. doi: 10.1109/TED.2006.878018

[158] S. Datta, "Nanoscale device modeling: Green's function method," Superlattices Microstruct., vol. 28, no. 4, pp. 253–277, Jul. 2000.

[159] J. Guo et al., "Performance analysis and design optimization of near ballistic carbon nanotube field-effect transistors," in IEDM Tech. Dig., 2004, pp. 703–706

[160] Y.W. Son, M.L. Cohen, and S.G. Louie, "Energy gaps in graphene nanoribbons", Phys. Rev. Lett., Vol. 97, pp.216803, 2006.

[161] R. Saito, G. Dresselhaus, and M. S. Dresselhaus, Physical Properties of Carbon Nanotubes. London, U.K.: Imperial College Press, 2003, pp. 35–58.

[162] Xie, Huaqing; Chen, Lifei; Yu, Wei; Wang, Bingqian, "Temperature dependent thermal conductivity of a free-standing graphene nanoribbon," Applied Physics Letters , vol.102, no.11, pp.111911,111911-4, Mar 2013.

[163] J. G. Son, M. Son, K. J. Moon, B. H. Lee, J. M. Myoung, M. S. Strano, et al.,"Sub-10 nm Graphene Nanoribbon Array Field-Effect Transistors Fabricated by Block Copolymer Lithography," Advanced Materials, vol. 25, pp. 4723-4728, 2013.

[164] S. Mudanai, Y Y Fang, Q Quyang, "Modeling of direct tunneling current through gate dielectric stacks", IEEE Trans. Electron Dev, vol. 47, Number. 10, pp. 1851–1857, 2000.

[165] Rasmita Sahoo et al., "Carbon Nanotube Field Effect Transistor: Basic Characterization and Effect of High Dielectric Material" International Journal of Recent Trends in Engineering, Vol 2, No. 7, November 2009.

[166] A. Kawamoto., J. Jameson, K. Cho, and R. Dutton, "Challenges for Atomic Scale Modeling in Alternative Gate Stack Engineering," IEEE Trans. Electron Dev, vol. 47, pp. 1787-1794, October 2000.

[167] Cheng B, Cao M, Rao R, et al, "The impact of high-k gate dielectrics and metal gate electrodes on sub-100 nm MOSFETs," IEEE Trans. Electron Dev, vol. 46, 1537-1544, July 1999.

[168] Youngki Yoon; Yijian Ouyang; Guo, Jing, "Scaling Behaviors of Graphene Nanoribbon FETs," Device Research Conference, 2007 65th Annual , vol., no., pp.271,272, 18-20 June 2007 doi: 10.1109/DRC.2007.4373750.

[169] Obradovic, B.; Kotlyar, R.; Heinz, F.; Matagne, P.; Rakshit, T.; Giles, M.D.; Stettler, M.A; Nikonov, D.E., "Analysis of graphene nanoribbons as a channel material for field-effect transistors," Applied Physics Letters , vol.88, no.14, pp.142102,142102-3, Apr 2006.

[170] Chen, Z. et. al. "IDDQ Testing for Deep-Submicron ICs: Challenges and Solutions",IEEE Design and Test of Computers.pp: 24-33, 2002.

[171] Sai-Kong Chin; Dawei Seah; Kai-Tak Lam; Samudra, G.S.; Liang, Gengchiau, "Device Physics and Characteristics of Graphene Nanoribbon Tunneling FETs," Electron Devices, IEEE Transactions on , vol.57, no.11, pp.3144,3152, Nov. 2010.

[172] Guo, Jing; Datta, Supriyo; Lundstrom, Mark, "A numerical study of scaling issues for Schottky-barrier carbon nanotube transistors," Electron Devices, IEEE Transactions on , vol.51, no.2, pp.172,177, Feb. 2004.

[173] Nakada K., Fujita M., Dresselhaus G. and Dresselhaus M.S. (1996). "Edge state in graphene ribbons: Nanometer size effect and edge shape dependence". Physical Review B 54 (24): 17954.

[174] R. Lake, G. Klimeck, R. C. Bowen, and D. Jovanovic, "Single and Multiband modeling of quantum electron transport through layered semiconductors devices", J. Appl. Phys., Vol. 81, pp.7845-7869, Feb. 1997.

[175] A. Svizhenko, M. P. Anantram, T. R. Govindam, and B. Biegel, "Two dimensional quantum mechanical modeling of nanotransistors", J. Appl.Phys., Vol. 91, pp. 2343-2354, Nov. 2001.

[176] J. Guo, S. Datta, M. Lundstrom, and M. P. Anantram, Multi-scale modelingof carbon nanotube transistors, Int. J. Multiscale Comput. Eng., Vol. 2, pp.257260, 2004.

[177] A. Trellakis, A. T. Galick, A. Pacelli, and U. Ravaioli, "Iteration scheme for the solution of the two-dimensional Schrdinger-Poisson equations in quantum structures", J. Appl. Phys, Vol. 81, p. 7880-7884, 1997.

[178] Weiss, N. O., Zhou, H., Liao, L., Liu, Y., Jiang, S., Huang, Y. and Duan, X. 2012. *Adv. Mat.*, 24: 5782–5825.

[179] Youngki Yoon; Fiori, G.; Seokmin Hong; Iannaccone, G.; Guo, Jing, "Performance Comparison of Graphene Nanoribbon FETs With Schottky Contacts and Doped Reservoirs," *Electron Devices, IEEE Transactions on* , vol.55, no.9, pp.2314,2323, Sept. 2008

Appendix A

Numerical Implementation

The Green's function is computed by means of the Recursive Green's Function (RGF) technique [174, 175]. Particular attention must be put in the definition of each self-energy matrix, which can be interpreted as a boundary condition of the Schrödinger equation. In particular, in our simulation we have considered a self-energy for semi-infinite leads as boundary conditions, which enables to consider the CNT/GNR as connected to infinitely long CNTs/GNRs at its ends.

In addition, Schottky contacts are considered for both CNT and GNR following a phenomenological approach described in [176].

From a numerical point of view, the code is based on the Newton-Raphson (NR) method with a predictor/corrector scheme [177]. In Fig. 3.1 we sketched a flow-chart of the whole code. In particular, the Schrödinger /NEGF equations are solved at the beginning of each NR cycle, starting from an initial potential $\widetilde{\Phi}$ and the charge density in the CNT/GNR and SNWT is kept constant until the NR cycle converges (i.e. the correction on the potential is smaller than a predetermined value). The algorithm is then repeated cyclically until the norm of the difference between the potential computed at the end of two subsequent NR cycles is smaller than a predetermined value.

Some convergence problems however may be encountered using this iterative scheme. Indeed, since the electron density is independent of the potential within a NR cycle, the Jacobian is null for points of the domain including carbon atoms/SNWT region, losing control over the correction of the potential. We have used a suitable expression for the charge predictor, in order to give an approximate expression for the Jacobian at each step of the NR cycle. To this purpose, we have used an exponential function for the predictor In particular, if n is the electron density the electron density n_i at the $i\text{-}th$ step of the NR cycle can be expressed as

$$n_i = n\, exp\,(\frac{\Phi_i - \widetilde{\Phi}}{V_T}) \quad (5.1)$$

where $\widetilde{\Phi}$ and $\Phi_i$are the electrostatic potentials computed at the first and ith step of the NR cycle, respectively, and V_T is the thermal voltage. Same considerations follow for the hole concentration. Since the electron density n is extremely sensitive to small changes of the electrostatic potential between two NR cycles, the exponential function acts in the overall procedure as a dumping factor for charge variations. In this way, convergence has been improved in the subthreshold regime and in the strong inversion regime. Convergence is still difficult in regions of the device where the charge is not compensated by fixed charge, where the right-hand term of the Poisson equation is considerably large.

An under-relaxation of the potential and of the charge can also be performed in order to help convergence. In particular, three different under-relaxations can be performed inside ViDES :

• relaxation on the potential at each NR cycle Φ_i

$$\Phi_i^{new} = \Phi_i^{new} + \varepsilon(\Phi_i^{old} - \Phi_i^{new}) \tag{5.2}$$

• relaxation on the potential at the end of each NR cycle Φ_f

$$\Phi_f = \Phi_f + \varepsilon(\widetilde{\Phi} - \Phi_f) \tag{5.3}$$

• relaxation of the charge density ρNEGF computed by the NEGF modules

$$\rho_{NEGF}^{new} = \rho_{NEGF}^{new} + \varepsilon(\rho_{NEGF}^{old} - \rho_{NEGF}^{new}) \tag{5.4}$$

Appendix B

B.1 Pyhthon script for simulating transfer characteristics of CNT SBFET

```
from NanoTCAD_ViDES import *

# I define the nanotube
CNT=nanotube(13,15);

# I create the grid
x=nonuniformgrid(array([-2,0.3,0,0.2,2,0.3]))
y=nonuniformgrid(array([-2,0.3,0,0.2,2,0.3]));

grid=grid3D(x,y,CNT.z,CNT.x,CNT.y,CNT.z);

#I define the contacts
CNT.contact='Schottky'

# Now I define the gate regions
top_gate=gate("hex",grid.xmax,grid.xmax,grid.ymin,grid.ymax,grid.zmi
n,grid.zmax)
bottom_gate=gate("hex",grid.xmin,grid.xmin,grid.ymin,grid.ymax,grid.
zmin,grid.zmax)

# I take care of the solid
SiO2=region("hex",-2,2,-2,2,grid.gridz[0],grid.gridz[grid.nz-1]);
SiO2.eps=3.9;

# I create the interface
p=interface3D(grid,top_gate,bottom_gate,SiO2);

# I work in the mode space, using 2 modes
p.modespace="yes"
CNT.Nmodes=2;

# Vds = 0.5 V
CNT.mu2=-0.5;

# I start the Vgs sweep. In particular 0<=Vgs<=1.25 V, with
# with 0.25V as voltage step
Vgmin=0.0;
Vgmax=1.25;
Vgstep=0.25;

#I create the vectors in which I store the data
vg=zeros(20);
current=zeros(20);
```

```
counter=0;
Vgs=Vgmin;
while (Vgs<=Vgmax):
    # I set the Fermi level of the top and bottom gate
    top_gate.Ef=-Vgs;
    set_gate(p,top_gate);
    bottom_gate.Ef=-Vgs;
    set_gate(p,bottom_gate);

    #If the first voltage, then I compute the initial solution
    if (Vgs==Vgmin):
        # I compute the initial solution
        p.normpoisson=1e-3;

        solve_init(grid,p,CNT);

    p.normpoisson=1e-1;
    p.normd=5e-2;
    solve_self_consistent(grid,p,CNT);
    vg[counter]=Vgs;
    current[counter]=CNT.current();
    counter=counter+1;
    Vgs=Vgs+Vgstep;

tempo=[vg,current]
savetxt("transfer1.out",transpose(tempo));
```

B.2 Pyhthon script for simulating transfer characteristics of GNR SBFET

```
from NanoTCAD_ViDES import *

# The width of the nanoribbon is 1.37 nm, and it is 15 nm long
GNR=nanoribbon(6,15);

# I create the grid
xg=nonuniformgrid(array([-2,0.3,0,0.2,2,0.3]))
yg=nonuniformgrid(array([-1,0.3,0,0.2,1.37,0.2,2.37,0.3]));
grid=grid3D(xg,yg,GNR.z,GNR.x,GNR.y,GNR.z);

# I define Schottky contacts
GNR.contact='Schottky'

# Now I define the gate regions
top_gate=gate("hex",grid.xmax,grid.xmax,grid.ymin,grid.ymax,grid.zmi
n,grid.zmax)
bottom_gate=gate("hex",grid.xmin,grid.xmin,grid.ymin,grid.ymax,grid.
zmin,grid.zmax)
```

```
# I take care of the solid
SiO2=region("hex",-2,2,-2,2,grid.zmin,grid.zmax);
SiO2.eps=3.9;

#I create the interface
p=interface3D(grid,top_gate,bottom_gate,SiO2);



# Vds = 0.5 V
GNR.mu2=-0.5;

# I start the Vgs sweep. In particular 0<=Vgs<=1.25 V, with
# with 0.25V as voltage step
Vgmin=0.0;
Vgmax=1.25;
Vgstep=0.25;

#I create the vectors in which I store the data
vg=zeros(20);
current=zeros(20);

counter=0;
Vgs=Vgmin;
while (Vgs<=Vgmax):
    # I set the Fermi level of the top and bottom gate
    top_gate.Ef=-Vgs;
    set_gate(p,top_gate);
    bottom_gate.Ef=-Vgs;
    set_gate(p,bottom_gate);

    #If the first voltage, then I compute the initial solution
    if (Vgs==Vgmin):
        # I compute the initial solution
        p.normpoisson=1e-3;

        solve_init(grid,p,GNR);

    p.normpoisson=1e-1;
    p.normd=5e-2;
    solve_self_consistent(grid,p,GNR);
    vg[counter]=Vgs;
    current[counter]=GNR.current();
    counter=counter+1;
    Vgs=Vgs+Vgstep;

tempo=[vg,current]
savetxt("transfer2.out",transpose(tempo));
```

B.3 Pyhthon script for simulating output characteristics of CNT SBFET

```
from NanoTCAD_ViDES import *

# I define the nanotube
CNT=nanotube(13,15);

# I create the grid
x=nonuniformgrid(array([-2,0.3,0,0.2,2,0.3]));
y=nonuniformgrid(array([-2,0.3,0,0.2,2,0.3]));
grid=grid3D(x,y,CNT.z,CNT.x,CNT.y,CNT.z);

#I define the contacts
CNT.contact='Schottky'

# Now I define the gate regions
top_gate=gate("hex",2,2,-2,2,grid.gridz[0],grid.gridz[grid.nz-1])
bottom_gate=gate("hex",-2,-2.5,-
2,2,grid.gridz[0],grid.gridz[grid.nz-1])

# I take care of the solid
SiO2=region("hex",-2,2,-2,2,grid.gridz[0],grid.gridz[grid.nz-1]);
SiO2.eps=3.9;

# I create the interface
p=interface3D(grid,top_gate,bottom_gate,SiO2);


# I work in the mode space, using 2 modes
p.modespace="yes"
CNT.Nmodes=2;


# I set set Vgs= 0.5V
top_gate.Ef=-0.5;
set_gate(p,top_gate);
bottom_gate.Ef=-0.5;
set_gate(p,bottom_gate);

p.normpoisson=1e-3;
solve_init(grid,p,CNT);

# I start the Vds sweep. In particular 0.05<=Vds<=0.55 V, with
# with 0.1V as voltage step
Vdsmin=0.05;
```

```
Vdsmax=0.55;
Vdstep=0.1;

Np=int(abs(Vdsmin-Vdsmax)/Vdstep)+1;
vg=zeros(Np);
current=zeros(Np);
p.underel=0.1;

counter=0;
Vds=Vdsmin;
while (Vds<=Vdsmax):

    CNT.mu2=-Vds;
    p.normpoisson=1e-1;
    p.normd=5e-3;
    solve_self_consistent(grid,p,CNT);
    vg[counter]=Vds;
    current[counter]=CNT.current();
    # I save the output files
    if (rank==0):
        string="./datiout/Phi%s.out" %Vds;
        savetxt(string,p.Phi);
        string="./datiout/ncar%s.out" %Vds;
        savetxt(string,p.free_charge);
        a=[CNT.E,CNT.T];
        string="./datiout/T%s.out" %Vds;
        savetxt(string,transpose(a));
        string="./datiout/jayn%s.out" %Vds;
        fp=open(string,"w");
        string2="%s" %current[counter];
        fp.write(string2);
        fp.close();
    counter=counter+1;
    Vds=Vds+Vdstep;


tempo=[vg,current]
savetxt("idvd1.out",transpose(tempo));
```

B.4 Pyhthon script for simulating output characteristics of GNR SBFET

```
from NanoTCAD_ViDES import *

# The width of the nanoribbon is 1.37 nm, and it is 15 nm long
GNR=nanoribbon(6,15);

# I create the grid
xg=nonuniformgrid(array([-2,0.3,0,0.2,2,0.3]))
yg=nonuniformgrid(array([-1,0.3,0,0.2,1.37,0.2,2.37,0.3]));
grid=grid3D(xg,yg,GNR.z,GNR.x,GNR.y,GNR.z);

# I define Schottky contacts
GNR.contact='Schottky'

# Now I define the gate regions
top_gate=gate("hex",grid.xmax,grid.xmax,grid.ymin,grid.ymax,grid.zmin,
grid.zmax)
bottom_gate=gate("hex",grid.xmin,grid.xmin,grid.ymin,grid.ymax,grid.zm
in,grid.zmax)

# I take care of the solid
SiO2=region("hex",-2,2,-2,2,grid.zmin,grid.zmax);
SiO2.eps=3.9;

# I create the interface
p=interface3D(grid,top_gate,bottom_gate,SiO2);


# I set Vgs= 0.5V
top_gate.Ef=-0.5;
set_gate(p,top_gate);
bottom_gate.Ef=-0.5;
set_gate(p,bottom_gate);

p.normpoisson=1e-3;
solve_init(grid,p,GNR);


# I start the Vds sweep. In particular 0.05<=Vds<=0.55 V, with
# with 0.1V as voltage step

Vdsmin=0.05;
Vdsmax=0.55;
Vdstep=0.1;

Np=int(abs(Vdsmin-Vdsmax)/Vdstep)+1;
```

```
vg=zeros(Np);
current=zeros(Np);
p.underel=0.1;

counter=0;
Vds=Vdsmin;
while (Vds<=Vdsmax):

    GNR.mu2=-Vds;
    p.normpoisson=1e-1;
    p.normd=5e-3;
    solve_self_consistent(grid,p,GNR);
    vg[counter]=Vds;
    current[counter]=GNR.current();
    # I save the output files
    if (rank==0):
        string="./datiout/Phi%s.out" %Vds;
        savetxt(string,p.Phi);
        string="./datiout/ncar%s.out" %Vds;
        savetxt(string,p.free_charge);
        a=[GNR.E,GNR.T];
        string="./datiout/T%s.out" %Vds;
        savetxt(string,transpose(a));
        string="./datiout/jayn%s.out" %Vds;
        fp=open(string,"w");
        string2="%s" %current[counter];
        fp.write(string2);
        fp.close();
    counter=counter+1;
    Vds=Vds+Vdstep;


tempo=[vg,current]
savetxt("idvds2.out",transpose(tempo));
```

NanoTCAD ViDES main python script

```
#
================================================================================
===============
#  Copyright (c) 2010-2012, G. Fiori, G. Iannaccone, University of
Pisa
#
#  This file is released under the BSD license.
#  See the file "license.txt" for information on usage and
#  redistribution of this file, and for a DISCLAIMER OF ALL
WARRANTIES.
#
================================================================================
===============

from numpy import *
from NanoTCAD_ViDESmod import *
from section import *
import sys
import types

writeout("\n")
writeout("--------------------------------------------------------------------
-------\n")
writeout("                         NanoTCAD ViDES ")
writeout("                      Version 1.4 (rel-1-4)")
writeout("                    Last Modified 29 Aug 2013")
writeout("                     Copyright (C) 2004-2013      \n")
writeout("--------------------------------------------------------------------
-------\n")
writeout("\n")

NEmax=5e3;
DIGIT_PRECISION=20;
max_number_of_cores_on_a_server=8;

#I check if mpi4py is installed on the machine or not
try:
    from mpi4py import MPI
    mpi4py_loaded = True
    sizeMPI = MPI.COMM_WORLD.Get_size()
except ImportError:
    mpi4py_loaded = False



#I check if pylab is installed on the machine or not
try:
    if (mpi4py_loaded):
        if (sizeMPI<=max_number_of_cores_on_a_server):
            from pylab import *
```

```
            pylab_loaded = True
    else:
        from pylab import *
        pylab_loaded = True
#except ImportError:
except Exception:
    pylab_loaded = False
    writeout("pylab not installed on this machine or not set up
correctly DISPLAY variable")

#definition of constants
kboltz=1.3807e-23
hbar=1.05459e-34
m0=9.1095e-31
q=1.60219e-19
eps0=8.85e-12
#Slater-Costner parameter for sp3d5s* tight-binding Hamiltonian in Si
thop_Si=array([-1.95933,-4.24135,-1.52230,3.02562,3.15565,-2.28485,-
0.80993,4.10364,-1.51801,-1.35554,2.38479,-1.68136,2.58880,-1.81400]);
onsite_Si=array([-
2.15168,4.22925,4.22925,4.22925,13.78950,13.78950,13.78950,13.78950,13
.78950,19.11650]);

def MPIze(channel):
    if (mpi4py_loaded):
        del channel.E;
        channel.E=zeros(NEmax);
        Eupper_save=channel.Eupper;
        Elower_save=channel.Elower;
        vt=kboltz*channel.Temp/q;
        sizeMPI = MPI.COMM_WORLD.Get_size()
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
            channel.rank=rank;
        # I compute the maximum and the minimum
        # of the energy interval
        if ((channel.Eupper>900)&(channel.Elower<-900)):
            Eupper=max(max(channel.mu1,max(-
channel.Phi)),channel.mu2)+0.5*channel.gap()+10*vt;
            Elower=min(min(channel.mu1,min(-
channel.Phi)),channel.mu2)-0.5*channel.gap()-10*vt;
        else:
            Eupper=channel.Eupper;
            Elower=channel.Elower;
#         string="Eupper and Elower %s %s " %(Eupper,Elower)
#         if (rank==0): writeout(string)
        E=arange(Elower,Eupper,channel.dE);
        arraydim=size(E)/sizeMPI;

        excess=size(E)-sizeMPI*arraydim
        if (rank<excess):
            channel.Elower=E[rank*(arraydim+1)];
```

```
            channel.Eupper=E[(rank+1)*(arraydim+1)-1];
        else:
            channel.Elower=E[(rank-
excess)*arraydim+excess*(arraydim+1)];
            if (rank==(sizeMPI-1)):
                channel.Eupper=E[size(E)-1];
            else:
                channel.Eupper=E[(rank-excess+1)*arraydim-
1+excess*(arraydim+1)];

#         string="Inizio rank %s %s %s"
%(rank,channel.Elower,channel.Eupper)
#         writeout(string)
        channel.charge_T();
        #writeout("Finito rank "),rank,channel.Elower,channel.Eupper;

        # I send the charge and the transmission coefficient
        if (rank!=0):
            temp=array(channel.charge);
            MPI.COMM_WORLD.Send([temp, MPI.DOUBLE],dest=0,tag=11);
            del temp;

            NPE=zeros(1,int);
            NPE[0]=int(ceil((channel.Eupper-
channel.Elower)/channel.dE))+1;
            #size(arange(channel.Elower,channel.Eupper,channel.dE));
            #int((channel.Eupper-channel.Elower)/channel.dE);
            #size(nonzero(channel.E));
            temp=array(channel.T[:NPE[0]]);
            temp2=array(channel.E[:NPE[0]]);
#             NPE[0]=size(temp);
            MPI.COMM_WORLD.Send([NPE, MPI.INT],dest=0,tag=10);
            MPI.COMM_WORLD.Send([temp, MPI.DOUBLE],dest=0,tag=12);
            MPI.COMM_WORLD.Send([temp2, MPI.DOUBLE],dest=0,tag=14);
            #writeout("Spedito rank "),rank
            del temp;
            del temp2;
        else:
            channel.charge=array(channel.charge);
            NNEE=int(ceil((channel.Eupper-
channel.Elower)/channel.dE))+1;
#size(arange(channel.Elower,channel.Eupper,channel.dE));
#             NNEE=((channel.Eupper-channel.Elower)/channel.dE);
#             size(nonzero(channel.E));
            channel.T=array(channel.T[:NNEE]);
            channel.E=array(channel.E[:NNEE]);
            for i in range(1,sizeMPI):
                temp=empty(size(channel.charge),dtype=double);
                MPI.COMM_WORLD.Recv([temp,
MPI.DOUBLE],source=i,tag=11);
                channel.charge=channel.charge+temp;
                del temp;
```

```
                NPE=empty(1,int);
                MPI.COMM_WORLD.Recv([NPE, MPI.INT],source=i,tag=10);
                temp=empty(NPE[0],dtype=double);
                MPI.COMM_WORLD.Recv([temp,
MPI.DOUBLE],source=i,tag=12);
                temp2=empty(NPE[0],dtype=double);
                MPI.COMM_WORLD.Recv([temp2,
MPI.DOUBLE],source=i,tag=14);
                channel.T=concatenate((channel.T,temp));
                channel.E=concatenate((channel.E,temp2));
                del temp;
                del temp2;
                #writeout("Preso rank "),i

        channel.charge = MPI.COMM_WORLD.bcast(channel.charge, root=0)
        channel.T = MPI.COMM_WORLD.bcast(channel.T, root=0)
        channel.E = MPI.COMM_WORLD.bcast(channel.E, root=0)
        channel.Eupper=Eupper_save;
        channel.Elower=Elower_save;
#        MPI.Finalize();
    else:
        writeout("*********************************")
        writeout("MPI not installed on this machine")
        writeout("*********************************")
    return;

def MPIze_kt(channel):
    if (mpi4py_loaded):

        kmin_save=channel.kmin;
        kmax_save=channel.kmax;
        vt=kboltz*channel.Temp/q;
        sizeMPI = MPI.COMM_WORLD.Get_size()
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
            channel.rank=rank;
        # I compute the maximum and the minimum
        # of the wave-vector kt
        kt_max=channel.kmax;
        kt_min=channel.kmin;
        if (rank==0): writeout("kt_max, kt_min"),kt_max,kt_min
        k=arange(kt_min,kt_max,channel.dk);
        arraydim=size(k)/sizeMPI;
        channel.kmin=k[rank*arraydim];
        if (rank==(sizeMPI-1)):
            channel.kmax=k[size(k)-1];
        else:
            channel.kmax=k[(rank+1)*arraydim-1];

        channel.charge_T();

        NE=size(channel.E);
```

```
        # I send the charge and the transmission coefficient
        if (rank!=0):
            temp=array(channel.charge);
            MPI.COMM_WORLD.Send([temp, MPI.DOUBLE],dest=0,tag=11);
            del temp;

            temp=array(channel.T);
            MPI.COMM_WORLD.Send([temp, MPI.DOUBLE],dest=0,tag=12);
            del temp;
        else:
            channel.charge=array(channel.charge);
            channel.T=array(channel.T);
            for i in range(1,sizeMPI):
                temp=empty(size(channel.charge),dtype=double);
                MPI.COMM_WORLD.Recv([temp,
MPI.DOUBLE],source=i,tag=11);
                channel.charge=channel.charge+temp;
                del temp;
                temp=empty(NE,dtype=double);
                MPI.COMM_WORLD.Recv([temp,
MPI.DOUBLE],source=i,tag=12);
                channel.T=channel.T+temp;
                del temp;

        channel.charge = MPI.COMM_WORLD.bcast(channel.charge, root=0)
        channel.T = MPI.COMM_WORLD.bcast(channel.T, root=0)
        channel.kmin=kmin_save;
        channel.kmax=kmax_save;
#        MPI.Finalize();

    else:
        writeout("*********************************")
        writeout("MPI not installed on this machine")
        writeout("*********************************")
    return;

def set_gate(interface,gate):
    interface.boundary_conditions[gate.index]=gate.Ef;

def solve_init(grid,interface,channel):

    # I get the rank
    if (mpi4py_loaded):
        channel.rank = MPI.COMM_WORLD.Get_rank()
    # I set the rank
    if (mpi4py_loaded):
        rank = MPI.COMM_WORLD.Get_rank()
        interface.rank=rank;
    else:
        interface.rank=0;
```

```
    # I first give an estimation of the density of states
    # when computing the flat band potential in the regions
    # where the fixed_charge is not equal to zero, assuming
    # full ionization

    # I save the temperature, mu1, mu2, the potential, n, Nc, Eupper,
Elower
#    temp_save=channel.Temp;
    mu1_save=channel.mu1;
    mu2_save=channel.mu2;
    Nc_save=channel.Nc;
    Eupper_save=channel.Eupper;
    Elower_save=channel.Elower;
    boundary_conditions_save=copy(interface.boundary_conditions);
    normpoisson_save=interface.normpoisson;

    interface.normpoisson=1e-3;
    # I impose a low-temperature, so to compute the LDOS, instead of
the
    # LDOS multiplied by the Fermi-Dirac
    name=grid.__class__.__name__;
    name_channel=channel.__class__.__name__;
    if (name=="grid3D"):
        if (name_channel=="multilayer_graphene"):
            channel.Nc=8;
            x_save=channel.x
            y_save=channel.y
            z_save=channel.z
            channel.atoms_coordinates();
        else:
            channel.Nc=6;
        channel.Phi=zeros(channel.n*channel.Nc);
        channel.mu1=0;
        channel.mu2=0;
        vt=kboltz*channel.Temp/q;
        channel.Eupper=channel.gap()+10*vt;
        channel.Elower=0;
        # I compute the NEGF
        #    if (interface.modespace=="yes"):
        #        channel.mode_charge_T();
        #    else:
        #        if (interface.MPI=="yes"):
        #            MPIze(channel);
        #        else:
        channel.charge_T();

    #
N1D=abs(sum(channel.charge))/(6*channel.Nc)/(3*channel.acc)*1e9;
        Ec=channel.gap()*0.5;

N1D=sum(abs(channel.charge))/(6*channel.n)/(4*channel.acc)*1e9*exp(Ec/
vt);
```

```
#    return N1D

        # I compute the mean z: if atoms have a z-coordinate > zmean
=> I impose the electrochemical potential mu2
        # if atoms have a z-coordinate < zmean => I impose the
electrochemical potential mu1
        zmean=(grid.zmin+grid.zmax)*0.5;
        indexS=nonzero((abs(interface.fixed_charge)>1e-
20)&(grid.z3D<zmean));
        indexD=nonzero((abs(interface.fixed_charge)>1e-
20)&(grid.z3D>=zmean));
        potential=zeros(grid.Np);

argoS=(abs(interface.fixed_charge[indexS])*grid.surf[indexS,5]/N1D);

argoD=(abs(interface.fixed_charge[indexD])*grid.surf[indexD,5]/N1D);

        potential[indexS]=(vt*(log(exp(argoS)-
1))+Ec)*sign(interface.fixed_charge[indexS])+mu1_save;
        potential[indexD]=(vt*(log(exp(argoD)-
1))+Ec)*sign(interface.fixed_charge[indexD])+mu2_save;

        interface.boundary_conditions[indexS]=potential[indexS];
        interface.boundary_conditions[indexD]=potential[indexD];


        solve_Poisson(grid,interface);
    elif (name=="grid2D"):
        channel.Nc=8;
        channel.Phi=zeros(channel.n*channel.Nc);
        channel.mu1=0;
        channel.mu2=0;
        vt=kboltz*channel.Temp/q;
        channel.Eupper=channel.gap()+10*vt;
        channel.Elower=0;
        # I compute the NEGF
        #    if (interface.modespace=="yes"):
        #        channel.mode_charge_T();
        #    else:
        #if (interface.MPI_kt=="yes"):
        #    MPIze_kt(channel);
        #else:
        channel.charge_T();

        Ec=channel.gap()*0.5;

N1D=sum(abs(channel.charge))/(8*channel.n)/(8*channel.acc)*1e9*exp(Ec/
vt);

        # I compute the mean z: if atoms have a z-coordinate > zmean
=> I impose the electrochemical potential mu2
```

```
        # if atoms have a z-coordinate < zmean => I impose the 
electrochemical potential mu1
        ymean=(grid.ymin+grid.ymax)*0.5;
        indexS=nonzero((abs(interface.fixed_charge)>1e-
20)&(grid.y2D<ymean));
        indexD=nonzero((abs(interface.fixed_charge)>1e-
20)&(grid.y2D>=ymean));
        potential=zeros(grid.Np);
        argoS=(abs(interface.fixed_charge[indexS])/N1D);
        argoD=(abs(interface.fixed_charge[indexD])/N1D);

        potential[indexS]=(vt*(log(exp(argoS)-
1))+Ec)*sign(interface.fixed_charge[indexS])+mu1_save;
        potential[indexD]=(vt*(log(exp(argoD)-
1))+Ec)*sign(interface.fixed_charge[indexD])+mu2_save;

#        potential[indexS]=Ec;
#        potential[indexD]=Ec;

        interface.boundary_conditions[indexS]=potential[indexS];
        interface.boundary_conditions[indexD]=potential[indexD];

        solve_Poisson(grid,interface);

    #going back to the old values
    channel.Nc=Nc_save
    channel.mu2=mu2_save;
    channel.mu1=mu1_save;
    channel.Eupper=Eupper_save;
    channel.Elower=Elower_save;
    interface.boundary_conditions=boundary_conditions_save;
    interface.normpoisson=normpoisson_save;
    if (name_channel=="multilayer_graphene"):
        channel.x=x_save
        channel.y=y_save
        channel.z=z_save
        del x_save,y_save,z_save
    #deleting the save variables
    del 
mu1_save,mu2_save,Nc_save,Eupper_save,Elower_save,boundary_conditions_
save;

    return;




def solve_self_consistent(grid,interface,channel):
    normad=1e30;
#    Phiold=1.0*interface.Phi;
    interface.Phiold=interface.Phi.copy();
    counter=1;
```

```
    if (mpi4py_loaded):
        rank = MPI.COMM_WORLD.Get_rank()
    else:
        rank=0;

    while (normad>interface.normd):
        # I pass the potential in correspondence of the
        # atoms of the material for which I compute the NEGF
        channel.Phi=interface.Phi[grid.swap]
        # I compute the NEGF

#        channel.Phi=zeros(size(grid.swap));
#        savetxt("Phi.before",interface.Phi[grid.swap]);

        if (interface.modespace=="yes"):
            channel.mode_charge_T();
        else:
            if (interface.MPI=="yes"):
                MPIze(channel);
            elif (interface.MPI_kt=="yes"):
                MPIze_kt(channel);
            else:
                channel.charge_T();

#        savetxt("Phi.temp2",interface.Phi);

#        a=[channel.E,channel.T];
#        savetxt("T.temp",transpose(a));

        if (rank==0):
            writeout("--------------------------------------------")
            string="            CURRENT = %s A/m"
%(channel.current());
            writeout(string);
            writeout("--------------------------------------------")

        # I pass back the free_charge term to
        # the 3D domain
        interface.free_charge[grid.swap]=channel.charge

        if (rank==0):
            savetxt("ncar.ini",interface.free_charge);
            savetxt("Phi.ini",interface.Phi);

        # I solve Poisson
        solve_Poisson(grid,interface);
#        normad=sqrt(sum((interface.Phiold-interface.Phi)**2));
#        Phiold=zeros(grid.Np);
        normad=max(abs(interface.Phiold-interface.Phi))

interface.Phi=interface.Phi+(interface.underel)*(interface.Phiold-
interface.Phi)
```

```
        del interface.Phiold;
#        del Phiold;
#        Phiold=1.0*interface.Phi;
        interface.Phiold=interface.Phi.copy();

        if (rank==0): print()
        string="Iteration # %s; ||Phi-Phiold||2 = %s"
%(counter,normad)
        if (rank==0): writeout(string)
        if (rank==0): print()
        counter=counter+1;
        if (counter>600):
            return;



def solve_Poisson(grid,interface):
    name=grid.__class__.__name__;
    if (name=="grid3D"):
        solvePoisson(grid,interface);
    elif (name=="grid2D"):
        solvePoisson2D(grid,interface);
    elif (name=="grid1D"):
        solvePoisson1D(grid,interface);
    interface.Phi=array(interface.Phi)
    return;

def nonuniformgrid(argu):
    #This is a wrapper for the nonuniformgridmod function
    #so to convert both the argument and the output to numpy arrays
    #I convert the argument in an array
    argarr=array(argu);
    out=nonuniformgridmod(argarr);
    # I return a pyarray
    outarr=array(out);
    return outarr;

#Fermi-Dirac Function
def Fermi(x):
    return 1/(1+exp(x));

def delete_class(class_obj):
    del_class(class_obj);
    del class_obj;
    return;

# This is the class for the nanotube
class nanotube:
    acc=0.144;
    def __init__(self,n,L):
        self.Nc=int(4*(floor((floor(L/nanotube.acc)-1)/3))+2);
        self.n=n;
```

```
        self.Phi=zeros(n*self.Nc);
        self.Eupper=1000.0;
        self.Elower=-1000.0;
        self.dE=1e-3;
        self.thop=-2.7;
        self.eta=1e-5;
        self.mu1=0;
        self.mu2=0;
        self.Temp=300;
        self.contact="doped";
        self.E=zeros(NEmax);
        self.T=zeros(NEmax);
        self.charge=zeros(self.n*self.Nc);
        self.Nmodes=n;
        self.x=zeros(n*self.Nc);
        self.y=zeros(n*self.Nc);
        self.z=zeros(n*self.Nc);
        self.L=int(self.Nc/2+((self.Nc-1)-
self.Nc*0.5)*0.5)*nanotube.acc;
        self.atoms_coordinates();
        self.rank=0;
    def gap(self):
        return abs(2*self.acc*self.thop*pi/(self.n*sqrt(3)*self.acc));
    def atoms_coordinates(self):
        CNT_atoms_coordinates(self);
        self.x=array(self.x);
        self.y=array(self.y);
        self.z=array(self.z);
        return;
    def charge_T(self):
        CNT_charge_T(self);
        self.E=array(self.E);
        self.T=array(self.T);
        self.charge=array(self.charge);
        return;
    def mode_charge_T(self):
        CNTmode_charge_T(self);
        self.E=array(self.E);
        self.T=array(self.T);
        self.charge=array(self.charge);
        return
    def current(self):
        vt=kboltz*self.Temp/q;
        E=self.E;
        T=self.T;
        arg=2*q*q/(2*pi*hbar)*T*(Fermi((E-self.mu1)/vt)-Fermi((E-
self.mu2)/vt))*self.dE;
        return sum(arg);


# This is the class for the nanoribbon
class GNRphonon:
```

```
    def __init__(self,dimer):
        self.N=1000; # number of points qx (longitudinal direction)
        while (((((self.N)-1)%(dimer/2))!=0) | (((self.N)%2)==0)):
            (self.N)+=1;
        self.dimer=dimer;   # numero dimer lines
        self.rank=0;
        self.phi=0.0; # channel potential (midgap)
        self.numberAC=2; # number of AC modes of different simmetry
considered (=2: LA+TA, =1: only LA)
        self.Ecutoff=1.0; # cutoff energy
        self.delta=2; # integer: it specifies the sampling along the
kx direction
        self.deltak=0;
        self.kyE=zeros(dimer);  # transverse electron wavevector
        self.qy=zeros(dimer); # transverse phonon wavevector
        self.kx=zeros(self.N); # longitudinal electron wavevector
        self.qx=zeros(self.N); # longitudinal phonon wavevector
        self.qx0=0.0; # fixed value for qx (computation of graphene
branches)
        self.qy0=0.0; # fixed value for qy (computation of graphene
branches)
        self.kxup=0; # maximum value for kx (computation of rates)
        self.kxdown=0; # minimum value for kx (computation of rates)
        self.dim1=self.N;
        self.dim2=dimer;
        self.dim3=6;
        self.mmin=0;
        self.mmax=dimer-1;
        self.kxmin=0;
        self.kxmax=0;
        self.Phi_r1=39.87*10.0;  # first neighbors
        self.Phi_ti1=17.28*10.0;
        self.Phi_to1=9.89*10.0;
        self.Phi_r2=7.29*10.0;   # second neighbors
        self.Phi_ti2=-4.61*10.0;
        self.Phi_to2=-0.82*10.0;
        self.Phi_r3=-2.64*10.0;  # third neighbors
        self.Phi_ti3=3.31*10.0;
        self.Phi_to3=0.58*10.0;
        self.Phi_r4=0.10*10.0;  # fourth neighbors
        self.Phi_ti4=0.79*10.0;
        self.Phi_to4=-0.52*10.0;
        self.energyE=zeros((self.dim1,(2*self.dim2))) # GNR electron
curves
        self.energyP2D=zeros((self.dim1,(self.dim2*self.dim3))) # GNR
phonon subbranches
        self.minAC=zeros((self.dim2,3));# minimum of the acoustic
subbranches
        self.Egraphene=zeros(self.dim3); # graphene
        self.rateAA=zeros((self.dim1,self.dim2));
        self.rateAE=zeros((self.dim1,self.dim2));
        self.rateOA=zeros((self.dim1,self.dim2));
```

```
        self.rateOE=zeros((self.dim1,self.dim2));
        self.Dac=4.5*(1.60219e-19); # deformation potential value (eV)
        self.temp=300; # temperature (K)
        self.thop=2.7; # hopping parameter (eV)
        self.aCC=0.144e-9; # lattice constant (m)
    def electron_GNR(self):
        electron_GNR(self);
        self.kx=array(self.kx);
        self.kyE=array(self.kyE);
        self.energyE=array(self.energyE);
        return;
    def phonon_GNR(self):
        phonon_GNR(self);
        self.qx=array(self.qx);
        self.qy=array(self.qy);
        self.energyP2D=array(self.energyP2D);
        return;
    def phonon_graphene(self):
        phonon_graphene(self);
        self.Egraphene=array(self.Egraphene);
        return;
    def rateACABS(self):
        rateACABS(self);
        self.rateAA=array(self.rateAA);
        return;
    def rateACEM(self):
        rateACEM(self);
        self.rateAE=array(self.rateAE);
        return;
    def rateOPTABS(self):
        rateOPTABS(self);
        self.rateOA=array(self.rateOA);
        return;
    def rateOPTEM(self):
        rateOPTEM(self);
        self.rateOE=array(self.rateOE);
        return;

# This is the class for the nanoribbon
class nanoribbon:
    acc=0.144;
    def __init__(self,n,L):
        self.Nc=int(4*(int((int(L/nanoribbon.acc)-1)/3))+2);
        self.n=n;
        self.Phi=zeros(n*self.Nc);
        self.Eupper=1000.0;
        self.Elower=-1000.0;
        self.dE=1e-3;
        self.thop=-2.7;
        self.eta=1e-5;
        self.mu1=0;
        self.mu2=0;
```

```
        self.Temp=300;
        self.contact="doped";
        self.E=zeros(NEmax);
        self.T=zeros(NEmax);
        self.charge=zeros(self.n*self.Nc);
        self.defects="no";
        self.roughness="no";
        self.rank=0;
        self.atoms_coordinates();
    def atoms_coordinates(self):
        GNR_atoms_coordinates(self);
        self.x=array(self.x);
        self.y=array(self.y);
        self.z=array(self.z);
        return;
    def gap(self):
        return GNRgap(self);
    def charge_T(self):
        GNR_charge_T(self);
        self.E=array(self.E);
        self.T=array(self.T);
        self.charge=array(self.charge);
        return;
    def current(self):
        vt=kboltz*self.Temp/q;
        E=array(self.E);
        T=array(self.T);
        arg=2*q*q/(2*pi*hbar)*T*(Fermi((E-self.mu1)/vt)-Fermi((E-
self.mu2)/vt))*self.dE
        return sum(arg);

# This is the class for the graphene
class graphene:
    acc=0.144;
    n=1;
    def __init__(self,L):
        self.Nc=int(4*(floor((floor(L/graphene.acc)-1)/3)));
        self.Phi=zeros(self.Nc);
        self.Ei=zeros(self.Nc);
        self.Eupper=1000.0;
        self.Elower=-1000.0;
        self.delta=sqrt(3)*graphene.acc;
        self.kmax=pi/self.delta;
        self.kmin=0;
        self.dk=0.1;
        self.dE=1e-3;
        self.thop=-2.7;
        self.eta=1e-8;
        self.mu1=0.0;
        self.mu2=0.0;
        self.Temp=300;
        self.E=zeros(NEmax);
```

```
        self.T=zeros(NEmax);
        self.charge=zeros(self.Nc);
        self.rank=0;
        self.atoms_coordinates();
        self.gap();
        self.T2D="no"
    def atoms_coordinates(self):
        GNR_atoms_coordinates(self);
        self.y=array(self.z);
        self.x=zeros(size(self.y));
        del self.z;
        return;
    def gap(self):
        return 0;
    def charge_T(self):
        # Number of slices and atoms
        slices=self.Nc;
        atoms=1;
        # I define the vector of the k-wave vector
        kvect=arange(self.kmin,self.kmax,self.dk)
        # I start defining the Hamiltonian for the graphene flake
        h=zeros((2*slices,3),dtype=complex);
        h[0][0]=1;
        for i in range(1,slices+1):
            h[i][0]=i
            h[i][1]=i

        kk=1;
        for ii in range(slices+1,2*slices):
            if ((ii%2)==1):
                h[ii][0]=kk;
                h[ii][1]=kk+1;
                h[ii][2]=self.thop;
            kk=kk+1;

        # I then compute the charge and the T for each energy and k
and perform the integral
        i=0;
        k=self.kmin;
        H = Hamiltonian(atoms, slices)
        if (self.T2D=="yes"):
            EE=arange(self.Elower,self.Eupper,self.dE);
            kvect=arange(self.kmin,self.kmax+self.dk,self.dk);
            X,Y=meshgrid(EE,kvect);
            Z=zeros((size(EE),size(kvect)))
        while (k<=(self.kmax+self.dk*0.5)):
            if (self.rank==0): writeout("------------------------------
-----")
            string="    kx range: [%s,%s] " %(self.kmin,self.kmax);
            if (self.rank==0): writeout(string)
            string="    iteration %s " %i;
            if (self.rank==0): writeout(string);
```

```
            if (self.rank==0): writeout("-----------------------------
-----")
            flaggo=0;
            kk=1;
            # I fill the Hamiltonian for the actual wavevector k in
the cycle
            for ii in range(slices+1,2*slices):
                if ((ii%2)==0):
                    h[ii][0]=kk;
                    h[ii][1]=kk+1;
                    if ((flaggo%2)==0):

h[ii][2]=self.thop+self.thop*exp(k*self.delta*1j);
                    else:
                        h[ii][2]=self.thop+self.thop*exp(-
k*self.delta*1j);
                    flaggo=flaggo+1;
                kk=kk+1;

            H.Eupper = self.Eupper;
            H.Elower = self.Elower;
            H.rank=self.rank;
            H.H = h
            H.dE=self.dE;
            H.Phi=self.Phi;
            H.Ei=-self.Phi;
            H.eta=self.eta;
            H.mu1=self.mu1;
            H.mu2=self.mu2;
            H.Egap=self.gap();


            # I then compute T and the charge for the actual kx
            H.charge_T()

            # I sum up all the contribution
            if (i==0):
                self.E=H.E;
                # the factor 2 is because I integrate over kx>0
                self.T=H.T*(2*self.dk/(2*pi));
                self.charge=H.charge*(2*self.dk/(2*pi));
            else:
                # the factor 2 is because I integrate over kx>0
                self.T=self.T+H.T*(2*self.dk/(2*pi));
                self.charge=self.charge+H.charge*(2*self.dk/(2*pi));

            if (self.T2D=="yes"):
                Z[:,i]=H.T[:size(EE)];
            k=k+self.dk
            i=i+1;

        if (self.T2D=="yes"):
```

```
            plt.imshow(Z, interpolation='bilinear', cmap=cm.gray,
                       origin='lower',
extent=[self.kmin,self.kmax,self.Elower,self.Eupper])
            show()

        del H;
        self.E=array(self.E);
        self.T=array(self.T)*1e9;
        self.charge=array(self.charge)*1e9;
        del kvect,h;
        return;

    def current(self):
        vt=kboltz*self.Temp/q;
        E=array(self.E);
        T=array(self.T);
        arg=2*q*q/(2*pi*hbar)*T*(Fermi((E-self.mu1)/vt)-Fermi((E-
self.mu2)/vt))*self.dE
        return sum(arg);

# This is the class for the graphene bilayer
class bilayer_graphene:
    acc=0.144;
    acc_p=0.35;
    n=2;
    def __init__(self,L):
        self.Nc=int(4*(floor((floor(L/bilayer_graphene.acc)-1)/3)));
        self.n=2;
        self.Phi=zeros(bilayer_graphene.n*self.Nc);
        self.Ei=zeros(bilayer_graphene.n*self.Nc);
        self.Eupper=1000.0;
        self.Elower=-1000.0;
        self.delta=sqrt(3)*bilayer_graphene.acc;
        self.kmax=pi/self.delta;
        self.kmin=0;
        self.dk=0.1;
        self.dE=1e-3;
        self.thop=-2.7;
        self.tp=-0.35;
        self.eta=1e-8;
        self.mu1=0.0;
        self.mu2=0.0;
        self.Temp=300;
        self.E=zeros(NEmax);
        self.T=zeros(NEmax);
        self.charge=zeros(bilayer_graphene.n*self.Nc);
        self.rank=0;
        self.atoms_coordinates();
        self.gap();
        self.T2D="no"
    def atoms_coordinates(self):
        n_save=self.n;
```

```
        self.n=1;
        GNR_atoms_coordinates(self);
        ydown=array(self.z);
        yup=ydown-self.acc*0.5;
        NN=size(ydown);
        kkk=0;
        self.y=zeros(2*NN);
        for i in range(0,NN):
            self.y[kkk]=ydown[i];
            self.y[kkk+1]=yup[i];
            kkk=kkk+2;
        self.x=zeros(size(self.y));
        i=linspace(0,size(self.y)-1,size(self.y))
        i_even=nonzero((i%2)==0);
        i_odd=nonzero((i%2)==1);
        self.x[i_even]=0;
        self.x[i_odd]=bilayer_graphene.acc_p;
        del self.z,i,i_even,i_odd;
        self.n=n_save;
        return;
    def gap(self):
        # This is an rough exstimation of
        # the Energy gap: for sure this is
        # the largest attainable value, within
        # the pz tight-binding model
        return abs(self.tp);
    def charge_T(self):
        # Number of slices and atoms
        slices=self.Nc;
        atoms=self.n;
        # I define the vector of the k-wave vector
        kvect=arange(self.kmin,self.kmax,self.dk)
        # I start defining the Hamiltonian for the bilayer graphene
        h=zeros((4*slices+2*(slices/4)-2,3),dtype=complex);
        h[0][0]=1;
        for i in range(1,2*slices+1):
            h[i][0]=i
            h[i][1]=i
            h[i][2]=0.0;

        # I then compute the charge and the T for each energy
        # and k and perform the integral
        i=0;
        k=self.kmin;
        H = Hamiltonian(atoms, slices)
        if (self.T2D=="yes"):
            EE=arange(self.Elower,self.Eupper,self.dE);
            kvect=arange(self.kmin,self.kmax+self.dk,self.dk);
            X,Y=meshgrid(EE,kvect);
            Z=zeros((size(EE),size(kvect)))
        while (k<=(self.kmax+self.dk*0.5)):
```

```
            if (self.rank==0): writeout("----------------------------
-----")
            string="    kx range: [%s,%s] " %(self.kmin,self.kmax);
            if (self.rank==0): writeout(string);
            string="    k: %s " %k;
            if (self.rank==0): writeout(string);
            if (self.rank==0): writeout("----------------------------
-----")

            # -------------------------------------------------
            # BEGINNING OF THE HAMILTONIAN DEFINITION
            # FOR THE GRAPHENE BILAYER
            # -------------------------------------------------

            # I work on the bottom graphene layer
            kk=1;
            flaggo=0;
            for ii in range(2*slices+1,3*slices):
                if ((ii%2)==1):
                    h[ii][0]=kk;
                    h[ii][1]=kk+2;
                    h[ii][2]=self.thop;
                    kk=kk+2;
                else:
                    h[ii][0]=kk;
                    h[ii][1]=kk+2;
                    if ((flaggo%2)==0):

h[ii][2]=self.thop+self.thop*exp(k*self.delta*1j);
                    else:
                        h[ii][2]=self.thop+self.thop*exp(-
k*self.delta*1j);
                    kk=kk+2;
                    flaggo=flaggo+1;

            # I work on the top graphene layer
            kk=2;
            flaggo=1;
            for ii in range(3*slices,4*slices-1):
                if ((ii%2)==0):
                    h[ii][0]=kk;
                    h[ii][1]=kk+2;
                    h[ii][2]=self.thop;
                    kk=kk+2;
                else:
                    h[ii][0]=kk;
                    h[ii][1]=kk+2;
                    if ((flaggo%2)==0):

h[ii][2]=self.thop+self.thop*exp(k*self.delta*1j);
                    else:
```

```
                        h[ii][2]=self.thop+self.thop*exp(-
k*self.delta*1j);
                    kk=kk+2;
                    flaggo=flaggo+1;


            # I now work on the perpendicular hopping parameter
            kk=3;
            for ii in range(4*slices-1,4*slices+int(slices/2)-2):
                h[ii][0]=kk;
                h[ii][1]=kk+3;
                h[ii][2]=self.tp;
                kk=kk+4;

            # --------------------------------------------------
            #          END OF THE HAMILTONIAN
            # --------------------------------------------------

            H.Eupper = self.Eupper;
            H.Elower = self.Elower;
            H.H = h
            H.rank=self.rank;
            H.dE=self.dE;
            H.Phi=self.Phi;
            ind_even=arange(0,size(H.Phi),2);
            ind_odd=ind_even+1;
            H.Ei[ind_even]=-
(self.Phi[ind_even]+self.Phi[ind_odd])*0.5;
            H.Ei[ind_odd]=-(self.Phi[ind_even]+self.Phi[ind_odd])*0.5;
            H.Ei_flag="no"
            H.eta=self.eta;
            H.mu1=self.mu1;
            H.mu2=self.mu2;
            H.Egap=self.gap();

#            return H.H

            # I then compute T and the charge for the actual kx
            H.charge_T()

            # I sum up all the contribution
            if (i==0):
                self.E=H.E;
                # the factor 2 is because I integrate over kx>0
                self.T=H.T*(2*self.dk/(2*pi));
                self.charge=H.charge*(2*self.dk/(2*pi));
#                self.charge=H.charge;
            else:
                # The spin is taken into account in the integral for
the current
                # the factor 2 is because I integrate over kx>0
                self.T=self.T+H.T*(2*self.dk/(2*pi));
```

```
                # 2 because I take into account
                # that I integrate over kx>0
                self.charge=self.charge+H.charge*(2*self.dk/(2*pi));

            if (self.T2D=="yes"):
                Z[:,i]=H.T[:size(EE)];
            k=k+self.dk
            i=i+1;

        if (self.T2D=="yes"):
            plt.imshow(Z, interpolation='bilinear', cmap=cm.gray,
                       origin='lower',
extent=[self.kmin,self.kmax,self.Elower,self.Eupper])
            show()

        del H;
        self.E=array(self.E);
        self.T=array(self.T)*1e9;
        self.charge=array(self.charge)*1e9;
#        self.charge=array(self.charge);
        del kvect,h;
        return;

    def current(self):
        vt=kboltz*self.Temp/q;
        E=array(self.E);
        T=array(self.T);
        arg=2*q*q/(2*pi*hbar)*T*(Fermi((E-self.mu1)/vt)-Fermi((E-
self.mu2)/vt))*self.dE
        return sum(arg);



# This is the class for the general Hamiltonian
class Hamiltonian:
    def __init__(self, n, Nc):
        self.Nc=Nc;
        self.n=n;
        self.x=zeros(n*self.Nc);
        self.y=zeros(n*self.Nc);
        self.z=zeros(n*self.Nc);
        self.Phi=zeros(n*self.Nc);
        self.Ei=zeros(n*self.Nc);
        self.Eupper=1000.0;
        self.Elower=-1000.0;
        self.dE=0.001;
        self.eta=1e-8;
        self.mu1=0;
        self.mu2=0;
        self.Temp=300;
        self.E=zeros(NEmax);
        self.T=zeros(NEmax);
```

```
        self.charge=zeros(n*self.Nc);
        self.Egap=0;
        self.rank=0;
        # if this flag is set to "yes" then Ei=-Phi
        self.Ei_flag="yes"
# The +1 will be then replaced by the number of orbitals per atoms in
the nearest neighbourgh approximation
#     self.H=zeros((((Nc*n)*(Nc*n+1)/2),2+100+10));
    def current(self):
        vt=kboltz*self.Temp/q;
        E=array(self.E);
        T=array(self.T);
        arg=2*q*q/(2*pi*hbar)*T*(Fermi((E-self.mu1)/vt)-Fermi((E-
self.mu2)/vt))*self.dE
        return sum(arg);
    def charge_T(self):
        if (self.Ei_flag=="yes"):
            self.Ei=-self.Phi;
        H_charge_T(self);
        self.E=array(self.E);
        self.T=array(self.T);
        self.charge=array(self.charge);
    def gap(self):
        return 0.5;
# This is the class for the zincblend structures
# This is the class for the zincblend structures
class Zincblend:
    def __init__(self, material, sqci, tilt, edge, zmax):
        self.material = material
        if self.material == 'Si':
            self.aux = [-2.15168,
                         4.22925,
                         19.11650,
                         13.78950,
                         -1.95933,
                         -4.24135,
                         -1.52230,
                         3.02562,
                         3.15565,
                         -2.28485,
                         -0.80993,
                         4.10364,
                         -1.51801,
                         -1.35554,
                         2.38479,
                         -1.68136,
                         2.58880,
                         -1.81400,
                         ]
            self.skparameters = array(self.aux, dtype=float)
            self.a0 = 5.431
            self.flag = 0
```

```
        if self.material == 'Ge':
            self.aux = [-1.95617,
                         5.30970,
                         19.29600,
                         13.58060,
                         -1.39456,
                         -3.56680,
                         -2.01830,
                         2.73135,
                         2.68638,
                         -2.64779,
                         -1.12312,
                         4.28921,
                         -1.73707,
                         -2.00115,
                         2.10953,
                         -1.32941,
                         2.56261,
                         -1.95120
                         ]
            self.skparameters = array(self.aux, dtype=float)
            self.a0 = 5.6575
            self.flag = 0

        if self.material == 'InAs':
            self.aux = [ -5.500420,
                          4.151070,
                          -0.581930,
                          6.971630,
                          19.710590,
                          19.941380,
                          13.031690,
                          13.307090,
                          -1.694350,
                          -4.210450,
                          -2.426740,
                          -1.159870,
                          2.598230,
                          2.809360,
                          2.067660,
                          0.937340,
                          -2.268370,
                          -2.293090,
                          -0.899370,
                          -0.488990,
                          4.310640,
                          -1.288950,
                          -1.731410,
                          -1.978420,
                          2.188860,
                          2.456020,
                          -1.584610,
```

```
                    2.717930,
                    -0.505090
                    ]
            self.skparameters = array(self.aux, dtype=float)
            self.a0 = 6.0583
            self.flag = 1

        self.sqci=sqci;
        self.tilt=tilt;
        self.edge=edge;
        self.zmax=zmax;

        layers = int(4*self.zmax/(self.a0) + 1)

        if (rank==0):
            writeout("prima="), layers

        if layers%4==1:
            layers-=1
        elif layers%4==2:
            layers-=2
        elif layers%4==3:
            layers+=1

        if layers%4!=0:
            writeout("INTERRUPT AT WIRE"), material, parameters[0][i]
            writeout("NUMBER OF SLICES NOT MULTIPLE OF 4")
            quit()

        layers += 8
        self.L = (self.a0/4)*(layers-1)
        self.n_aux = int((4*self.edge/self.a0)*(4*self.edge/self.a0))
+ 10;
        #forse se ci si leva il +10 non cambia nulla (provare)
        self.Nc_aux = int((4*self.zmax/self.a0)) + 10;
        self.zmax=self.L

        self.atoms=zeros(1);
        self.slices=zeros(1);
        self.max=zeros(1);
        self.rank=0;
        self.deltae=20.0;
        self.ics = zeros(self.n_aux*self.Nc_aux);
        self.ipsilon = zeros(self.n_aux*self.Nc_aux);
        self.zeta = zeros(self.n_aux*self.Nc_aux);
        self.H_aux=zeros(
(self.Nc_aux*self.n_aux)*((self.Nc_aux*self.n_aux+1)/2)*(2+100));

self.H=zeros((((self.Nc_aux*self.n_aux)*(self.Nc_aux*self.n_aux+1)/2),
2+100));

        self.Zinc();
```

```
        self.n = int(self.atoms[0]);
        self.Nc= int(self.slices[0]);
        self.x = self.ics;
        self.y = self.ipsilon;
        self.z = self.zeta;

        self.Phi=zeros(self.n*self.Nc);
        self.Ei=zeros(self.n*self.Nc);
        self.Eupper=1000.0;
        self.Elower=-1000.0;
        self.dE=0.001;
        self.eta=1e-8;
        self.mu1=0;
        self.mu2=0;
        self.Temp=300;
        self.E=zeros(NEmax);
        self.T=zeros(NEmax);
        self.charge=zeros(self.n*self.Nc);
        self.Egap=0;
    def gap(self):
        return 0;
    def current(self):
        vt=kboltz*self.Temp/q;
        E=array(self.E);
        T=array(self.T);
        arg=2*q*q/(2*pi*hbar)*T*(Fermi((E-self.mu1)/vt)-Fermi((E-
self.mu2)/vt))*self.dE
        return sum(arg);
    def charge_T(self):
        H_charge_T(self);
        self.E=array(self.E);
        self.T=array(self.T);
        self.charge=array(self.charge);
        return;
    def Zinc(self):

        writeout(self.skparameters)
#        quit()

        Zinc(self);
#        self.zeta = array(self.zeta);
#        ics1 = []
#        ipsilon1 = []
#        zeta1 = []
#        i = 0
#        j = 0
#        k = 0
#        temp = self.zeta[0]- self.a0
#        zeta1.append(temp)
#        aux = []
#        for ln in self.zeta:
#            if (self.zeta[i]- self.a0) == temp:
```

```
#                 #temp = self.zeta[i]- self.a0
#                 i = i + 1
#                 j = j + 1
#             else:
#                 zeta1.append(self.zeta[i]- self.a0)
#                 temp = self.zeta[i]- self.a0
#                 i = i + 1
#                 aux.append(j)
#                 j=1;
#
#         print aux
#         print self.zeta


#         for i in range (100):
        #print zeta1

#         print 'slices =', int(self.slices[0])
#         print 'atoms =', int(self.atoms[0])

#         zeta2 = []
#         for i in range (int(self.slices[0])):
#             for j in range(int(self.atoms[0])):
#                 zeta2.append(zeta1[i])
#
#         print 'ECCOLO'
        #print zeta2

#         self.zeta = zeta2

        #print self.zeta

        H_back = []

        i = 0
        j = 0
        bound = int(self.max[0]/102)
        writeout(bound)

        for i in range ( bound ):
            row = []
            for j in range(102):
                row.append(self.H_aux[j + 102*i])
            H_back.append(row)
            #print row
            del row



        #print H_back[40]

        new = array(H_back, dtype=complex)
```

```
        self.H = new

#        print self.H[17]

#        quit()

        return;

def ciccione(vettore,n,Nc,z,a0):
        ics1 = []
        ipsilon1 = []
        zeta1 = []
        i = 0
        j = 0
        k = 0
        temp = z[0]- a0
        z1=[];
        z1.append(temp)
        aux = []
        for ln in arange(0,n*Nc):
            if (z[i]- a0) == temp:
                #temp = self.zeta[i]- self.a0
                i = i + 1
                j = j + 1
            else:
                z1.append(z[i]- a0)
                temp = z[i]- a0
                i = i + 1
                aux.append(j)
                j=1;

#       TODO: the following sum is equal to the total number of
#       atoms, really present in the simulated nanowire
#
#       Ntot_atoms=sum(aux[:Nc])
#
#
        array2 = []
        for i in range(Nc):
            k=0;
            if (aux[i]==n):
                for j in arange(sum(aux[:i]),sum(aux[:i])+n):
                    array2.append(vettore[j])
            else:
                for j in arange(sum(aux[:i]),sum(aux[:i])+aux[i]):
                    array2.append(vettore[j]);
                for j in arange(sum(aux[:i])+aux[i],sum(aux[:i])+n):
                    array2.append(0)

        return array(array2, dtype=float);
```

```
class grid3D:
    def __init__(self,*args):
        # I initialize the rank
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
        else:
            rank=0;
        # args is a tuple and len(args) return
        # the number of arguments
        # the number of arguments can be either 3 or 6
        # if 3, the first three inputs are the grid along the
        # x,y,z axis
        # if 6, the first three inputs are the grid along the
        # x,y,z axis, while the last three inputs are the x-y-z
        # coordinates of the atoms
        if (len(args)>3):
            xg=around(args[0],5);
            yg=around(args[1],5);
            zg=around(args[2],5);
            xC=around(args[3],5);
            yC=around(args[4],5);
            zC=around(args[5],5);
            npC=size(xC);
        else:
            xg=around(args[0],5);
            yg=around(args[1],5);
            zg=around(args[2],5);
            npC=0;

        #I create the grid
        if (npC!=0):
            #find the unique values for xC,yC and zC
            uxC=unique(xC);
            uyC=unique(yC);
            uzC=unique(zC);

            # I find the only the additional values which are in xg
and not in uxC
            # the same for the other axis
            exg=intersect1d(setxor1d(xg,uxC),xg);
            eyg=intersect1d(setxor1d(yg,uyC),yg);
            ezg=intersect1d(setxor1d(zg,uzC),zg);

        if (npC!=0):
            x=unique(concatenate((uxC,xg),1));
            y=unique(concatenate((uyC,yg),1));
            z=unique(concatenate((uzC,zg),1));
        else:
            x=xg;
            y=yg;
            z=zg;
```

```
        # I start to compute the volume associated to each grid point
        X,Y=meshgrid(x,y);

        #Number of points
        nx=size(x);
        ny=size(y);
        nxy=nx*ny;
        nz=size(z);
        Np=nxy*nz;
        string="Number of grid points %s " %Np
        if (rank == 0): writeout(string)


######################################################################
##############
        #I create the Volume elements using the sorted grid
        xd=avervect(x);
        yd=avervect(y);
        zd=avervect(z);
        X,Y=meshgrid(x,y);
        X,Z=meshgrid(x,z);

        XD,ZD=meshgrid(xd,zd);
        surfxz=XD*ZD;
        YD,ZD=meshgrid(yd,zd);
        surfyz=YD*ZD;
        XD,YD=meshgrid(xd,yd);
        surfxy=XD*YD;

        #The volumes for the sorted grid are finally computed
        a,b=meshgrid((XD*YD).flatten(),zd);
        dVes=abs((a*b).flatten());

        if (rank == 0): writeout("Volumes computed")


######################################################################
##############
        # I create the dist vectors
        dists=zeros((Np,6));

        # I take care of dists[:,1]
        i=arange(0,nx);
        ip1=i+1;
        ip1[nx-1]=nx-1;
        xdistp=x[ip1]-x[i];

dists[:,1]=meshgrid(meshgrid(xdistp,y)[0].flatten(),z)[0].flatten();
        del ip1,xdistp;

        # I take care of dists[:,0]
```

```
        im1=i-1;
        im1[0]=0;
        xdistm=x[i]-x[im1];

dists[:,0]=meshgrid(meshgrid(xdistm,y)[0].flatten(),z)[0].flatten();
        del i,im1,xdistm;

        # I take care of dists[:,3]
        j=arange(0,ny);
        jp1=j+1;
        jp1[ny-1]=ny-1;
        ydistp=y[jp1]-y[j];

dists[:,3]=meshgrid(meshgrid(x,ydistp)[1].flatten(),z)[0].flatten();
        del jp1,ydistp;

        # I take care of dists[:,2]
        jm1=j-1;
        jm1[0]=0;
        ydistm=y[j]-y[jm1];

dists[:,2]=meshgrid(meshgrid(x,ydistm)[1].flatten(),z)[0].flatten();
        del j,jm1,ydistm;

        # I take care of dists[:,5]
        k=arange(0,nz);
        kp1=k+1;
        kp1[nz-1]=nz-1;
        zdistp=z[kp1]-z[k];

dists[:,5]=meshgrid(meshgrid(x,y)[1].flatten(),zdistp)[1].flatten();
        del kp1,zdistp;

        # I take care of dists[:,4]
        km1=k-1;
        km1[0]=0;
        zdistm=z[k]-z[km1];

dists[:,4]=meshgrid(meshgrid(x,y)[1].flatten(),zdistm)[1].flatten();
        del k,km1,zdistm;



######################################################################
##############
        #Now I work on the surfaces

        surfs=zeros((Np,6));

        #surf 0
        XD,YD=meshgrid(xd,yd)
        ##YD[:,0]=0;
```

```
        a,b=meshgrid(YD.flatten(),zd)
        surfs[:,0]=abs((a*b).flatten());
        #surf 1
        XD,YD=meshgrid(xd,yd)
        ##YD[:,nx-1]=0;
        a,b=meshgrid(YD.flatten(),zd)
        surfs[:,1]=abs((a*b).flatten());
        #surf 2
        XD,YD=meshgrid(xd,yd)
        ##XD[0,:]=0;
        a,b=meshgrid(XD.flatten(),zd)
        surfs[:,2]=abs((a*b).flatten());
        #surf 3
        XD,YD=meshgrid(xd,yd)
        ##XD[ny-1,:]=0;
        a,b=meshgrid(XD.flatten(),zd)
        surfs[:,3]=abs((a*b).flatten());
        #surf 4
        XD,YD=meshgrid(xd,yd)
        a,b=meshgrid((XD*YD).flatten(),z)
        surfs[:,4]=abs(a.flatten());
        ##surfs[0:nx*ny-1,4]=0;
        #surf 5
        XD,YD=meshgrid(xd,yd)
        a,b=meshgrid((XD*YD).flatten(),z)
        surfs[:,5]=abs(a.flatten());
        ##surfs[(nz-1)*(nx*ny):nz*nx*ny,5]=0;

        if (rank == 0): writeout("Surfaces created")



######################################################################
##############
        #Now I have to go back to the unsorted grid.
        #I create the sorted and unsorted coordinates
        #vectors as a function of the index

        #sorted positions
        x3Ds=meshgrid(meshgrid(x,y)[0].flatten(),z)[0].flatten();
        y3Ds=meshgrid(meshgrid(x,y)[1].flatten(),z)[0].flatten();
        z3Ds=meshgrid(meshgrid(x,y)[1].flatten(),z)[1].flatten();

        #unsorted positions

        if (npC!=0):
            xtemp=unique(concatenate((uxC,xg),1));
            ytemp=unique(concatenate((uyC,yg),1));
            ztemp=unique(concatenate((uzC,zg),1));

            if (rank == 0): writeout("I work on the swap array");
            NpC=size(xC);
```

```
            swap=array(arange(0,NpC),int);
            for i in range(0,NpC):
                ixC=nonzero(xtemp==xC[i])[0][0];
                iyC=nonzero(ytemp==yC[i])[0][0];
                izC=nonzero(ztemp==zC[i])[0][0];
                ii=ixC+iyC*nx+izC*nx*ny;
                swap[i]=ii;



#######################################################################
##############
        # I now fill the attributes of the istance of the grid class
        self.x3D=x3Ds;
        self.y3D=y3Ds
        self.z3D=z3Ds
        self.dVe=dVes;
        self.surf=surfs;
        self.dist=dists;
        self.nx=nx;
        self.ny=ny;
        self.nz=nz;
        self.Np=Np;
        self.gridx=x;
        self.gridy=y;
        self.gridz=z;
        if (npC!=0):
            self.swap=swap;
        self.xmin=min(x);
        self.xmax=max(x);
        self.ymin=min(y);
        self.ymax=max(y);
        self.zmin=min(z);
        self.zmax=max(z);
        return;

class grid2D:
    def __init__(self,*args):

        # I initialize the rank
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
        else:
            rank=0;

        # args is a tuple and len(args) return
        # the number of arguments
        # the number of arguments can be either 2 or 4
        # if 2, the first two inputs are the grid along the
        # x,y axis
        # if 4, the first two inputs are the grid along the
        # x,y axis, while the last two inputs are the x-y
```

```
        # coordinates of the atoms
        if (len(args)>2):
            xg=around(args[0],5);
            yg=around(args[1],5);
            xC=around(args[2],5);
            yC=around(args[3],5);
            npC=size(xC);
        else:
            xg=around(args[0],5);
            yg=around(args[1],5);
            npC=0;

        #I create the grid
        if (npC!=0):
            #find the unique values for xC,yC and zC
            uxC=unique(xC);
            uyC=unique(yC);

            # I find the only the additional values which are in xg
and not in uxC
            # the same for the other axis
            exg=intersect1d(setxor1d(xg,uxC),xg);
            eyg=intersect1d(setxor1d(yg,uyC),yg);

        if (npC!=0):
            x=unique(concatenate((uxC,xg),1));
            y=unique(concatenate((uyC,yg),1));
        else:
            x=xg;
            y=yg;


        #Number of points
        nx=size(x);
        ny=size(y);
        nxy=nx*ny;
        Np=nxy;
        string="Number of grid points %s " %Np
        if (rank == 0): writeout(string)


######################################################################
##############
        #I create the Volume elements using the sorted grid
        xd=avervect(x);
        yd=avervect(y);
        X,Y=meshgrid(x,y);

        XD,YD=meshgrid(xd,yd);
        surfxy=XD*YD;

        if (rank == 0): writeout("Volumes computed")
```

```
##########################################################################
##############
        # I create the dist vectors
        dists=zeros((Np,4));

        # I take care of dists[:,1]
        i=arange(0,nx);
        ip1=i+1;
        ip1[nx-1]=nx-1;
        xdistp=x[ip1]-x[i];
        dists[:,1]=meshgrid(xdistp,y)[0].flatten();
        del ip1,xdistp;

        # I take care of dists[:,0]
        im1=i-1;
        im1[0]=0;
        xdistm=x[i]-x[im1];
        dists[:,0]=meshgrid(xdistm,y)[0].flatten()
        del i,im1,xdistm;

        # I take care of dists[:,3]
        j=arange(0,ny);
        jp1=j+1;
        jp1[ny-1]=ny-1;
        ydistp=y[jp1]-y[j];
        dists[:,3]=meshgrid(x,ydistp)[1].flatten()
        del jp1,ydistp;

        # I take care of dists[:,2]
        jm1=j-1;
        jm1[0]=0;
        ydistm=y[j]-y[jm1];
        dists[:,2]=meshgrid(x,ydistm)[1].flatten();
        del j,jm1,ydistm;


##########################################################################
##############
        #Now I work on the surface

        XD,YD=meshgrid(xd,yd)
        surfs=(XD*YD).flatten();

        if (rank == 0): writeout("Surface created")



##########################################################################
##############
        #Now I have to go back to the unsorted grid.
```

```
        #I create the sorted and unsorted coordinates
        #vectors as a function of the index

        #sorted positions
        x2Ds=meshgrid(x,y)[0].flatten();
        y2Ds=meshgrid(x,y)[1].flatten();

        #unsorted positions

        if (npC!=0):
            xtemp=unique(concatenate((uxC,xg),1));
            ytemp=unique(concatenate((uyC,yg),1));

            if (rank == 0): writeout("I work on the swap array");
            NpC=size(xC);
            swap=array(arange(0,NpC),int);
            for i in range(0,NpC):
                ixC=nonzero(xtemp==xC[i])[0][0];
                iyC=nonzero(ytemp==yC[i])[0][0];
                ii=ixC+iyC*nx;
                swap[i]=ii;



######################################################################
##############
        # I now fill the attributes of the istance of the grid class
        self.x2D=x2Ds;
        self.y2D=y2Ds
        self.surf=surfs;
        self.dist=dists;
        self.nx=nx;
        self.ny=ny;
        self.Np=Np;
        self.gridx=x;
        self.gridy=y;
        if (npC!=0):
            self.swap=swap;
        self.xmin=min(x);
        self.xmax=max(x);
        self.ymin=min(y);
        self.ymax=max(y);
        return;

class grid1D:
    def __init__(self,*args):

        # I initialize the rank
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
        else:
            rank=0;
```

```
        # args is a tuple and len(args) return
        # the number of arguments
        # the number of arguments can be either 1 or 2
        # if 1, the first input is the grid along the
        # x axis
        # if 2, the first input is the grid along the
        # x axis, while the second input is the x
        # coordinates of the atoms
        if (len(args)>1):
            xg=around(args[0],5);
            xC=around(args[1],5); # attenzione: modificato il
28/5/2011
            npC=size(xC);
        else:
            xg=around(args[0],5);
            npC=0;

        #I create the grid
        if (npC!=0):
            #find the unique values for xC
            uxC=unique(xC);

            # I find the only the additional values which are in xg
and not in uxC
            exg=intersect1d(setxor1d(xg,uxC),xg);

        if (npC!=0):
            x=unique(concatenate((uxC,xg),1));
        else:
            x=xg;


        #Number of points
        nx=size(x);
        Np=nx;
        if (rank == 0): print(("Number of grid points ",Np))


######################################################################
##############
        # I create the dist vectors
        dists=zeros((Np,4));

        # I take care of dists[:,1]
        i=arange(0,nx);
        ip1=i+1;
        ip1[nx-1]=nx-1;
        xdistp=x[ip1]-x[i];
        dists[:,1]=xdistp;
        del ip1,xdistp;
```

```
        # I take care of dists[:,0]
        im1=i-1;
        im1[0]=0;
        xdistm=x[i]-x[im1];
        dists[:,0]=xdistm;
        del i,im1,xdistm;


######################################################################
##############
        #Now I have to go back to the unsorted grid.
        #I create the sorted and unsorted coordinates
        #vectors as a function of the index

        if (npC!=0):
            xtemp=unique(concatenate((uxC,xg),1));

            if (rank == 0): print("I work on the swap array");
            NpC=size(xC);
            swap=array(arange(0,NpC),int);
            for i in range(0,NpC):
                ixC=nonzero(xtemp==xC[i])[0][0];
                ii=ixC;
                swap[i]=ii;



######################################################################
##############
        # I now fill the attributes of the istance of the grid class
        self.x=x;
        self.dist=dists;
        self.nx=nx;
        self.Np=Np;
        self.gridx=x;
        if (npC!=0):
            self.swap=swap;
        self.xmin=min(x);
        self.xmax=max(x);
        return;

class region:
    def __init__(self,*args):
        self.name="none";
        self.geometry="hex";
        self.eps=3.9;
        self.rho=0;
        if (args[0]=="hex"):
            if (len(args)>5):
                self.xmin=args[1];
                self.xmax=args[2];
                self.ymin=args[3];
```

```
                self.ymax=args[4];
                self.zmin=args[5];
                self.zmax=args[6];
            elif ((len(args)>3)&(len(args)<=5)):
                self.xmin=args[1];
                self.xmax=args[2];
                self.ymin=args[3];
                self.ymax=args[4];
            elif (len(args)<=3):
                self.xmin=args[1];
                self.xmax=args[2];
    def set_material(self,material):
        if (material.lower()=="sio2"):
            self.eps=3.9;
            self.mel=0.5;
            self.met=0.5;
            self.Egap=8.05;
            self.chi=0.95;
            self.mhole=0.42
        if (material.lower()=="si"):
            self.eps=11.8;
            self.mel=0.916;
            self.met=0.19;
            self.Egap=1.124519;
            self.chi=4.05;
            self.mhole=0.549;

class gate:
    def __init__(self,*args):
        self.geometry="hex";
        self.Ef=0;
        self.wf=4.1;
        if (args[0]=="hex"):
            if (len(args)>5):
                self.xmin=args[1];
                self.xmax=args[2];
                self.ymin=args[3];
                self.ymax=args[4];
                self.zmin=args[5];
                self.zmax=args[6];
            elif ((len(args)>3)&(len(args)<=5)):
                self.xmin=args[1];
                self.xmax=args[2];
                self.ymin=args[3];
                self.ymax=args[4];
            elif (len(args)<=3):
                self.xmin=args[1];
                self.xmax=args[2];
        if (args[0]=="cyl"):
            self.xc=args[1];
            self.yc=args[2];
            self.radius=args[3];
```

```
            self.geometry="cyl"
        if (args[0]=="trapz"):
            self.xmin=args[1];
            self.xmax=args[2];
            self.y1=args[3];
            self.z1=args[4];
            self.y2=args[5];
            self.z2=args[6];
            self.y3=args[7];
            self.z3=args[8];
            self.y4=args[9];
            self.z4=args[10];
            self.geometry="trapz"

class interface3D:
    def __init__(self,*args):

        # I set the rank
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
            self.rank=rank;
        else:
            self.rank=0;

        # I compute the number of arguments (classes)
        Narg=size(args);
        # I first find the index of the class grid
        igrid=-10;
        for i in range(0,Narg):
            name=args[i].__class__.__name__
            if (name=="grid3D"):
                igrid=i;
        # If no grid class is specified I exit
        if (igrid==-10):
            writeout("ERROR: grid not passed to structure")
            return;

        # I create the arrays to be used
        self.eps=zeros(args[igrid].Np);

        # I create the vector, where the boundary conditions
        # are specified:
        # if 2000   : inner point
        # if 1001   : Neumann 1
        # if 1002   : Neumann 2
        # if 1003   : Neumann 3
        # if 1004   : Neumann 4
        # if 1005   : Neumann 5
        # if 1006   : Neumann 6
        # if <= 1000: Fermi level of the gate

        # I start defining all the points as inner points
```

```
        self.boundary_conditions=2000*ones(args[igrid].Np);


######################################################################
#########################
        # Now I impose the Neumann Boundary conditions on
        # the surfaces delimiting the 3D domain

######################################################################
#########################

        # I take care of Neumann1
        indexNeu1=nonzero(args[igrid].x3D==min(args[igrid].gridx));
        self.boundary_conditions[indexNeu1]=1001;

        # I take care of Neumann2
        indexNeu2=nonzero(args[igrid].x3D==max(args[igrid].gridx));
        self.boundary_conditions[indexNeu2]=1002;

        # I take care of Neumann3
        indexNeu3=nonzero(args[igrid].y3D==min(args[igrid].gridy));
        self.boundary_conditions[indexNeu3]=1003;

        # I take care of Neumann4
        indexNeu4=nonzero(args[igrid].y3D==max(args[igrid].gridy));
        self.boundary_conditions[indexNeu4]=1004


        # I take care of Neumann5 and Neumann6
        indexNeu5=nonzero(args[igrid].z3D==min(args[igrid].gridz));
        self.boundary_conditions[indexNeu5]=1005;
        indexNeu6=nonzero(args[igrid].z3D==max(args[igrid].gridz));
        self.boundary_conditions[indexNeu6]=1006;


######################################################################
#########################
        # I check to which class the args belongs to
        # and I proceed accordingly

######################################################################
#########################

        for i in range(0,Narg):
            name=args[i].__class__.__name__
            # I check if the class is a gate
            if (name=="gate"):
                #I check if the geometry is an hexahedron
                if (args[i].geometry=="hex"):
                    # I find the indexes of the 3D grid which belongs
to the gate
                    # with hex geometry
```

```
index=nonzero((args[i].xmin<=args[igrid].x3D)&(args[i].xmax>=args[igri
d].x3D)&

(args[i].ymin<=args[igrid].y3D)&(args[i].ymax>=args[igrid].y3D)&

(args[i].zmin<=args[igrid].z3D)&(args[i].zmax>=args[igrid].z3D));
                    self.boundary_conditions[index]=args[i].Ef;
                    args[i].index=index;
                if (args[i].geometry=="trapz"):
                    # I find the indexes of the 2D grid which belongs
to the gate
                    # with trapezoidal geometry
                    if (args[i].y2==args[i].y1):
                        m1=(args[i].z2-args[i].z1)/(args[i].y2-
args[i].y1+1e-3)
                    else:
                        m1=(args[i].z2-args[i].z1)/(args[i].y2-
args[i].y1)
                    if (args[i].y3==args[i].y2):
                        m2=(args[i].z3-args[i].z2)/(args[i].y3-
args[i].y2+1e-3)
                    else:
                        m2=(args[i].z3-args[i].z2)/(args[i].y3-
args[i].y2)
                    if (args[i].y4==args[i].y3):
                        m3=(args[i].z4-args[i].z3)/(args[i].y4-
args[i].y3+1e-3)
                    else:
                        m3=(args[i].z4-args[i].z3)/(args[i].y4-
args[i].y3)
                    if (args[i].y4==args[i].y1):
                        m4=(args[i].z4-args[i].z1)/(args[i].y4-
args[i].y1+1e-3)
                    else:
                        m4=(args[i].z4-args[i].z1)/(args[i].y4-
args[i].y1)


index=nonzero((args[igrid].z3D>=(m1*(args[igrid].y3D-
args[i].y1)+args[i].z1))&

(args[igrid].z3D>=(m2*(args[igrid].y3D-args[i].y2)+args[i].z2))&

(args[igrid].z3D<=(m3*(args[igrid].y3D-args[i].y3)+args[i].z3))&

(args[igrid].z3D<=(m2*(args[igrid].y3D-args[i].y1)+args[i].z1))&

(args[i].xmin<=args[igrid].x3D)&(args[i].xmax>=args[igrid].x3D));
                    self.boundary_conditions[index]=args[i].Ef;
                    args[i].index=index;
```

```
            elif (name=="region"):
                if (args[i].geometry=="hex"):
                    # I find the indexes of the 3D grid which belongs
to the gate
                    # with hex geometry

index=nonzero((args[i].xmin<=args[igrid].x3D)&(args[i].xmax>=args[igri
d].x3D)&

(args[i].ymin<=args[igrid].y3D)&(args[i].ymax>=args[igrid].y3D)&

(args[i].zmin<=args[igrid].z3D)&(args[i].zmax>=args[igrid].z3D));
                    self.eps[index]=args[i].eps;
            elif (name=="grid3D"):
                #dummy line
                name;
            else:
                writeout("ERROR: Unrecognized input")
                return;


######################################################################
#########################
        # I fill the field of the interface class

######################################################################
#########################

        #self.boundary already filled
        #self.eps already filled
        self.Phiold=zeros(args[igrid].Np)
        self.Phi=zeros(args[igrid].Np);
        self.normpoisson=1e-3;
        self.tolldomn=1e-1;
        self.underel=0;
        self.free_charge=zeros(args[igrid].Np);
        self.fixed_charge=zeros(args[igrid].Np);
        self.normd=5e-2;
        self.modespace="no"
        self.MPI="no"
        self.MPI_kt="no"
        return;

class interface2D:
    def __init__(self,*args):

        # I set the rank
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
            self.rank=rank;
        else:
            self.rank=0;
```

```
        # I compute the number of arguments (classes)
        Narg=size(args);
        # I first find the index of the class grid
        igrid=-10;
        for i in range(0,Narg):
            name=args[i].__class__.__name__
            if (name=="grid2D"):
                igrid=i;
        # If no grid class is specified I exit
        if (igrid==-10):
            writeout("ERROR: grid not passed to structure")
            return;

        # I create the arrays to be used
        self.eps=zeros(args[igrid].Np);

        # I create the vector, where the boundary conditions
        # are specified:
        # if 2000   : inner point
        # if 1001   : Neumann 1
        # if 1002   : Neumann 2
        # if 1003   : Neumann 3
        # if 1004   : Neumann 4
        # if <= 1000: Fermi level of the gate

        # I start defining all the points as inner points
        self.boundary_conditions=2000*ones(args[igrid].Np);


######################################################################
#########################
        # Now I impose the Neumann Boundary conditions on
        # the surfaces delimiting the 3D domain

######################################################################
#########################

        # I take care of Neumann1
        indexNeu1=nonzero(args[igrid].x2D==min(args[igrid].gridx));
        self.boundary_conditions[indexNeu1]=1001;

        # I take care of Neumann2
        indexNeu2=nonzero(args[igrid].x2D==max(args[igrid].gridx));
        self.boundary_conditions[indexNeu2]=1002;

        # I take care of Neumann3
        indexNeu3=nonzero(args[igrid].y2D==min(args[igrid].gridy));
        self.boundary_conditions[indexNeu3]=1003;

        # I take care of Neumann4
        indexNeu4=nonzero(args[igrid].y2D==max(args[igrid].gridy));
```

```
        self.boundary_conditions[indexNeu4]=1004



######################################################################
#########################
        # I check to which class the args belongs to
        # and I proceed accordingly

######################################################################
#########################

        for i in range(0,Narg):
            name=args[i].__class__.__name__
            # I check if the class is a gate
            if (name=="gate"):
                #I check if the geometry is an hexahedron
                if (args[i].geometry=="hex"):
                    # I find the indexes of the 2D grid which belongs
to the gate
                    # with hex geometry

index=nonzero((args[i].xmin<=args[igrid].x2D)&(args[i].xmax>=args[igri
d].x2D)&

(args[i].ymin<=args[igrid].y2D)&(args[i].ymax>=args[igrid].y2D));
                    self.boundary_conditions[index]=args[i].Ef;
                    args[i].index=index;
                #I check if the geometry is an cylindrical
                if (args[i].geometry=="cyl"):
                    # I find the indexes of the 2D grid which belongs
to the gate
                    # with cyl geometry
                    index=nonzero(((args[i].xc-
args[igrid].x2D)**2+(args[i].yc-
args[igrid].y2D)**2)<(args[i].radius)**2);
                    self.boundary_conditions[index]=args[i].Ef;
                    args[i].index=index;
            elif (name=="region"):
                if (args[i].geometry=="hex"):
                    # I find the indexes of the 2D grid which belongs
to the gate
                    # with hex geometry

index=nonzero((args[i].xmin<=args[igrid].x2D)&(args[i].xmax>=args[igri
d].x2D)&

(args[i].ymin<=args[igrid].y2D)&(args[i].ymax>=args[igrid].y2D));
                    self.eps[index]=args[i].eps;
            elif (name=="grid2D"):
                #dummy line
                name;
```

```
            else:
                writeout("ERROR: Unrecognized input")
                return;


######################################################################
#########################
        # I fill the field of the interface class

######################################################################
#########################

        #self.boundary already filled
        #self.eps already filled
        self.Phiold=zeros(args[igrid].Np)
        self.Phi=zeros(args[igrid].Np);
        self.normpoisson=1e-3;
        self.tolldomn=1e-1;
        self.underel=0;
        self.free_charge=zeros(args[igrid].Np);
        self.fixed_charge=zeros(args[igrid].Np);
        self.normd=5e-2;
        self.modespace="no"
        self.MPI="no"
        self.MPI_kt="no"
        return;

class interface1D:
    def __init__(self,*args):

        # I set the rank
        if (mpi4py_loaded):
            rank = MPI.COMM_WORLD.Get_rank()
            self.rank=rank;
        else:
            self.rank=0;

        # I compute the number of arguments (classes)
        Narg=size(args);
        # I first find the index of the class grid
        igrid=-10;
        for i in range(0,Narg):
            name=args[i].__class__.__name__
            if (name=="grid1D"):
                igrid=i;
        # If no grid class is specified I exit
        if (igrid==-10):
            print("ERROR: grid not passed to structure")
            return;

        # I create the arrays to be used
        self.eps=zeros(args[igrid].Np);
```

```
        self.mel=zeros(args[igrid].Np);
        self.met=zeros(args[igrid].Np);
        self.chi=zeros(args[igrid].Np);
        self.Egap=zeros(args[igrid].Np);
        self.fixed_charge=zeros(args[igrid].Np);
        self.mhole=zeros(args[igrid].Np);

        # I create the vector, where the boundary conditions
        # are specified:
        # if 2000   : inner point
        # if 1001   : Neumann 1
        # if 1002   : Neumann 2
        # if <= 1000: Fermi level of the gate

        # I start defining all the points as inner points
        self.boundary_conditions=2000*ones(args[igrid].Np);


######################################################################
#########################
        # Now I impose the Neumann Boundary conditions on
        # the surfaces delimiting the 3D domain

######################################################################
#########################

        # I take care of Neumann1
        indexNeu1=nonzero(args[igrid].x==min(args[igrid].gridx));
        self.boundary_conditions[indexNeu1]=1001;

        # I take care of Neumann2
        indexNeu2=nonzero(args[igrid].x==max(args[igrid].gridx));
        self.boundary_conditions[indexNeu2]=1002;



######################################################################
#########################
        # I check to which class the args belongs to
        # and I proceed accordingly

######################################################################
#########################

        for i in range(0,Narg):
            name=args[i].__class__.__name__
            # I check if the class is a gate
            if (name=="gate"):
                #I check if the geometry is an hexahedron
                if (args[i].geometry=="hex"):
                    # I find the indexes of the 2D grid which belongs
to the gate
```

```
                    # with hex geometry

index=nonzero((args[i].xmin<=args[igrid].x)&(args[i].xmax>=args[igrid]
.x));
                    self.boundary_conditions[index]=args[i].Ef;
                    args[i].index=index;
            elif (name=="region"):
                if (args[i].geometry=="hex"):
                    dist=avervect(args[igrid].x)*1e-9;
                    # I find the indexes of the 2D grid which belongs
to the gate
                    # with hex geometry

index=nonzero((args[i].xmin<=args[igrid].x)&(args[i].xmax>=args[igrid]
.x));
                    self.eps[index]=args[i].eps;
                    self.mel[index]=args[i].mel;
                    self.met[index]=args[i].met;
                    self.chi[index]=args[i].chi;
                    self.Egap[index]=args[i].Egap;
                    self.fixed_charge[index]=args[i].rho*dist[index];
                    self.mhole[index]=args[i].mhole;

            elif (name=="grid1D"):
                #dummy line
                name;
            else:
                print("ERROR: Unrecognized input")
                return;


#######################################################################
#########################
        # I fill the field of the interface class

#######################################################################
#########################

        #self.boundary already filled
        #self.eps already filled
        self.Phiold=zeros(args[igrid].Np)
        self.Phi=zeros(args[igrid].Np);
        self.normpoisson=1e-3;
        self.tolldomn=1e-1;
        self.underel=0;
        self.free_charge=zeros(args[igrid].Np);
        self.normd=5e-2;
        self.modespace="no"
        self.MPI="no"
        self.MPI_kt="no"
        return;
```

```
def dope_reservoir(grid,interface,channel,molar_fraction,bbox):
    name=grid.__class__.__name__;
    if (name=="grid3D"):
        xmin=bbox[0];
        xmax=bbox[1];
        ymin=bbox[2];
        ymax=bbox[3];
        zmin=bbox[4];
        zmax=bbox[5];

index=nonzero((xmin<=grid.x3D[grid.swap])&(xmax>=grid.x3D[grid.swap])&

(ymin<=grid.y3D[grid.swap])&(ymax>=grid.y3D[grid.swap])&

(zmin<=grid.z3D[grid.swap])&(zmax>=grid.z3D[grid.swap]))
        interface.fixed_charge[grid.swap[index]]=molar_fraction;
    elif (name=="grid2D"):
        xmin=bbox[0];
        xmax=bbox[1];
        ymin=bbox[2];
        ymax=bbox[3];

index=nonzero((xmin<=grid.x2D[grid.swap])&(xmax>=grid.x2D[grid.swap])&

(ymin<=grid.y2D[grid.swap])&(ymax>=grid.y2D[grid.swap]))

interface.fixed_charge[grid.swap[index]]=molar_fraction/channel.delta*
1e9;
    elif (name=="grid1D"):
        xmin=bbox[0];
        xmax=bbox[1];

index=nonzero((xmin<=grid.x[grid.swap])&(xmax>=grid.x[grid.swap]));

interface.fixed_charge[grid.swap[index]]=molar_fraction/(channel.delta
z*channel.deltay)*1e18;
# MODIFICATO IL 6/6/2011: aggiunto il deltay e deltaz

    return index;

class Device:
    def __init__(self):
        self.Nregions=1;
        self.regions=[];
        self.E=zeros(NEmax);
    def test(self):
        return self.E;
```

```
def test_var_args(farg, *args):
    writeout("formal arg:"), size(args)
    for arg in args:
        writeout("another arg:"), arg

def avervect(x):
    # This function compute the length of
    # the Voronoi segment of a one-dimensional array x
    nx=size(x);
    xd=zeros(nx);
    xini=x[0];
    xd[0]=abs(x[0]-x[1])*0.5;
    for i in range(1,nx-1):
        xd[i]=abs((x[i+1]-x[i-1])*0.5);
    xd[nx-1]=abs(x[nx-1]-x[nx-2])*0.5
    return xd;

def save_format_xyz(outputfile,x,y,z,atom):

    if sys.version > '3':
        import subprocess;
    else:
        import subprocess

    out=[x*10,y*10,z*10]
    fp=open(outputfile,"w");
    fp.write(str(size(x)));
    fp.write("\n");
    fp.write("\n");
    for i in range(0,size(x)):
        string="%s %s %s %s" %(atom,out[0][i],out[1][i],out[2][i]);
        fp.write(string);
        fp.write(" ");
        fp.write("\n");
    fp.close()
    return;

"""def convert_pdb(filename,thop):
    fp=open(filename,"r");
    hh=[];
    atoms=0;
    i=0;
    x=[];
    y=[];
    z=[];
    h=[];
    h.append([1,0,0]);
    for line in fp:
        hh.append(line);
        atoms=atoms+(hh[i].split()).count('HETATM');
```

```
        if
(((hh[i].split()).count('HETATM')==1)|((hh[i].split()).count('ATOM')==
1)):
            x.append((hh[i].split())[5]);
            y.append((hh[i].split())[6]);
            z.append((hh[i].split())[7]);

h.append([int((hh[i].split())[1]),int((hh[i].split())[1]),0]);
        if ((hh[i].split()).count('CONECT')==1):
            a=(hh[i].split());
            NPV=size(a)-1
            for j in range(0,NPV):
                a1=int(a[1]);
                if (a1<int(a[j+1])):
                    h.append([a1,int(a[j+1]),thop])
        if ((hh[i].split()).count('CRYST1')==1):
            a=(hh[i].split());
            if (double(a[1])>=100):
                deltax=0.0;
            else:
                deltax=double(a[1])/10.0;
            if (double(a[2])>=100):
                deltay=0.0;
            else:
                deltay=double(a[2])/10.0;
            if (double(a[3])>=100):
                deltaz=0.0;
            else:
                deltaz=double(a[3])/10.0;


        i=i+1;
    fp.close()
    H=array(h,dtype(complex));
    x=array(x,dtype(float))/10.0;
    y=array(y,dtype(float))/10.0;
    z=array(z,dtype(float))/10.0;
    return H,x,y,z,deltax,deltay,deltaz;"""

def create_H_from_xyz(x,y,z,orbitals,onsite,thop,d_bond,Nbond):
    # WE ASSUME THAT:
    #
    # 1) TRANSPORT IS IN THE Z DIRECTION
    # 2) THE STRUCTURE IS COMPOSED BY THE SAME TYPE OF ATOMS
    # 3) ALONG THE Z-DIRECTION THE STRUCTURE IS PERIODIC WITH PERIOD
EQUAL TO 4 SLICES
    #

    # I find the minimum and maximum coordinates at the border
    # so to take care of the passivation of the atoms at the borders
    xmin=min(x);
    xmax=max(x);
```

```
    ymin=min(y);
    ymax=max(y);
    zmin=min(z);
    zmax=max(z);

    # I compute the number of slices (ASSUMPTION 2)
    Nc=int(size(unique(z)));
    #  I have already computed n at the beginning
    #    n=int(size(nonzero(z==zmin)));
    # I compute the number of atoms in the first 4 slices
    temp=unique(z);
    Natom_slices=size(nonzero(z<=temp[3]));
    del temp;

    # I check the maximum number of atoms on each slice;
    u=unique(z);
    Nuz=size(u);
    n=-1;
    for i in range(0,Nuz):
        nnew=size(nonzero(z==u[i]));
        if (nnew>=n):
            n=nnew;
    del i;

    # Now I start doing though stuff
    # I fill x,y and z with dummy atoms
    # If it is a dummy atom, the coordinate is equal to dummy_coord

    dummy_coord=10000;
    xa=[];
    ya=[];
    za=[];
    k=0;
    for i in range(0,Nuz):
#        print ya
        nnew=size(nonzero(z==u[i]));
        for j in range(0,nnew):
            xa.append(x[k]);
            ya.append(y[k]);
            za.append(z[k]);
            k=k+1;
        if (nnew<n):
            for j in range(nnew,n):
                xa.append(dummy_coord);
                ya.append(dummy_coord);
                za.append(dummy_coord);
#                k=k+1;

    del x,y,z,u,i
    x=array(xa,dtype(float));
    y=array(ya,dtype(float));
    z=array(za,dtype(float));
```

```
    del xa,ya,za

    Np=size(x);
    Ncol_max=10;
    NN=zeros((Np,Ncol_max),dtype(int));
    border=[]
    # I first find the Nearest Neighbour
    for i in range(0,Np):
        ind=nonzero((sqrt((x-x[i])**2+(y-y[i])**2+(z-
z[i])**2)<=d_bond)&(sqrt((x-x[i])**2+(y-y[i])**2+(z-z[i])**2)>1e-
10))[0];
        if (size(ind)>Ncol_max):
            print()
            writeout("ERROR IN create_H_from_xyz subroutine in
NanoTCAD_ViDES.py file")
            writeout("Use a larger value for Ncol_max")
            print()
            exit(0);
#        print i
        NN[i,0]=i+1;
        NN[i,1:size(ind)+1]=ind+1;
        NPV=size(nonzero(NN[i,:]))-1;
        if (NPV<Nbond):
            border.append(i);


    # Now I work on the Hamiltonian
    atoms=0;
    i=0;
    h=[];

    # I fill the h list with the number of orbitals
    ll=[orbitals,0];
    fill=zeros(orbitals**2);
    h.append(ll+list(fill))
    del ll,i




    # I take care of the diagonal elements
    for i in range(0,Np):
        if ((x[i]<dummy_coord)):
            if (orbitals>1):
                # (ASSUMPTION 1)
                if i in border:
                    xfn=zeros(4);
                    yfn=zeros(4);
                    zfn=zeros(4);
                    if (z[i]==zmin):
                        NPV=size(nonzero(NN[i+4*n,:]))-1;
                        xfn=x[NN[i+n*4,1:NPV+1]-1];
```

```
                          yfn=y[NN[i+n*4,1:NPV+1]-1];
                          zfn=z[NN[i+n*4,1:NPV+1]-1];
                          xp=x[i+n*4];
                          yp=y[i+n*4];
                          zp=z[i+n*4];
                      elif (z[i]==zmax):
                          NPV=size(nonzero(NN[i-4*n,:]))-1;
                          xfn=x[NN[i-n*4,1:NPV+1]-1];
                          yfn=y[NN[i-n*4,1:NPV+1]-1];
                          zfn=z[NN[i-n*4,1:NPV+1]-1];
                          xp=x[i-n*4];
                          yp=y[i-n*4];
                          zp=z[i-n*4];
                      else:
                          NPV=size(nonzero(NN[i,:]))-1;
                          xfn=x[NN[i,1:NPV+1]-1];
                          yfn=y[NN[i,1:NPV+1]-1];
                          zfn=z[NN[i,1:NPV+1]-1];
                          xp=x[i];
                          yp=y[i];
                          zp=z[i];

                      deltae=20.0;

tempM=Sipassivation(xp,yp,zp,NPV,xfn,yfn,zfn,deltae);
    #                  print tempM
    #                  print x[i],y[i],z[i]
    #                  print xfn
    #                  print yfn
    #                  print zfn
    #                  exit(0);
                      B=zeros((10,10));
                      B[:4,:4]=tempM.reshape(4,4);

h.append([i+1,i+1]+list((diag(onsite)+B).flatten()));
    #
h.append([i+1,i+1]+list((diag(onsite)).flatten()));
                      del B,tempM,xfn,yfn,zfn;
                  else:

h.append([i+1,i+1]+list((diag(onsite)).flatten()));
              else:
                  h.append([i+1,i+1]+list(fill));
          else:
              # If the atom is dummy then I mark it with the 77777 value
              # Right now it works only for one orbital
              h.append([i+1,i+1]+list(77777*ones(orbitals**2)));


    # I take care of the off-diagonal elements
    for i in range(0,Np):
        NPV=size(nonzero(NN[i,:]))-1;
```

```
        for j in range(0,NPV):
            a1=int(NN[i,0]);
            if (a1<int(NN[i,j+1])):
                if (orbitals>1):
                    # I compute the cosine
                    module=sqrt(((double(x[a1-1])-
double(x[int(NN[i,j+1])-1]))**2)+(double(y[a1-1])-
double(y[int(NN[i,j+1])-1]))**2+(double(z[a1-1])-
double(z[int(NN[i,j+1])-1]))**2);
                    cosx=(-double(x[a1-1])+double(x[int(NN[i,j+1])-
1]))/module;
                    cosy=(-double(y[a1-1])+double(y[int(NN[i,j+1])-
1]))/module;
                    cosz=(-double(z[a1-1])+double(z[int(NN[i,j+1])-
1]))/module;
#                    print a1,int(NN[i,j+1]),cosx,cosy,cosz,module
#                    input=hstack((array([cosx,cosy,cosy]),thop));
#                    print input
#                    matrix_thop=Simatrix(input);
                    matrix_thop=Simatrix(cosx,cosy,cosz,thop);
#                    print matrix_thop
#                    print "----------------"
                    h.append([a1,int(NN[i,j+1])]+list(matrix_thop));
                else:
                    h.append([a1,int(NN[i,j+1]),thop])
    H=array(h,dtype=complex);
    return H,n,Nc;

def get_xyz_from_file(filename):
    fp=open(filename,"r");
    xa=[]
    ya=[]
    za=[]
    for line in fp:
        if (size(line.split())>3):
            xa.append((line.split())[1]);
            ya.append((line.split())[2]);
            za.append((line.split())[3]);
    x=array(xa,dtype(float));
    y=array(ya,dtype(float));
    z=array(za,dtype(float));
    del xa,ya,za
    return x,y,z;


def convert_pdb(filename,orbitals,thop):
    # ASSUMPTION: ALL THE ATOMS ARE OF THE SAME MATERIAL

    # I first read the atoms coordinates
    hh=[];
    deltax=0;
    deltay=0;
```

```
    deltaz=0;
    x=[];
    y=[];
    z=[];
    i=0;
    fp=open(filename,"r");
    for line in fp:
        hh.append(line);
        if
(((hh[i].split()).count('HETATM')==1)|((hh[i].split()).count('ATOM')==
1)):
#            ATOM_TYPE=(hh[i].split())[2];
            x.append((hh[i].split())[5]);
            y.append((hh[i].split())[6]);
            z.append((hh[i].split())[7]);
        i=i+1;
    fp.close()
    del hh;

    # Now I work on the Hamiltonian
    hh=[];
    atoms=0;
    i=0;
    h=[];

    # I fill the h list with the number of orbitals
    ll=[orbitals,0];
    fill=zeros(orbitals**2);
    h.append(ll+list(fill))
    del ll

    # I fill the rest of the h list
    fp=open(filename,"r");
    for line in fp:
        hh.append(line);
        atoms=atoms+(hh[i].split()).count('HETATM');
        if
(((hh[i].split()).count('HETATM')==1)|((hh[i].split()).count('ATOM')==
1)):
             if (orbitals>1):

h.append([int((hh[i].split())[1]),int((hh[i].split())[1])]+list((diag(
onsite)).flatten()));
             else:

h.append([int((hh[i].split())[1]),int((hh[i].split())[1])]+list(fill))
;
        if ((hh[i].split()).count('CONECT')==1):
            a=(hh[i].split());
            NPV=size(a)-1
            for j in range(0,NPV):
                a1=int(a[1]);
```

```
                if (a1<int(a[j+1])):
                    if (orbitals>1):
                        # I compute the cosine
                        module=sqrt(((double(x[a1-1])-
double(x[int(a[j+1])-1]))**2)+(double(y[a1-1])-double(y[int(a[j+1])-
1]))**2+(double(z[a1-1])-double(z[int(a[j+1])-1]))**2);
                        cosx=(double(x[a1-1])-double(x[int(a[j+1])-
1]))/module;
                        cosy=(double(y[a1-1])-double(y[int(a[j+1])-
1]))/module;
                        cosz=(double(z[a1-1])-double(z[int(a[j+1])-
1]))/module;
                        cosx=1;cosy=1;cosz=1;
                        input=hstack((array([cosx,cosy,cosy]),thop));
                        matrix_thop=Simatrix(input);
                        h.append([a1,int(a[j+1])]+list(matrix_thop));
                    else:
                        h.append([a1,int(a[j+1]),thop])

        if ((hh[i].split()).count('CRYST1')==1):
            a=(hh[i].split());
            if (double(a[1])>=100):
                deltax=0.0;
            else:
                deltax=double(a[1])/10.0;
            if (double(a[2])>=100):
                deltay=0.0;
            else:
                deltay=double(a[2])/10.0;
            if (double(a[3])>=100):
                deltaz=0.0;
            else:
                deltaz=double(a[3])/10.0;


        i=i+1;
    fp.close()

    H=array(h,dtype(complex));
    x=array(x,dtype(float))/10.0;
    y=array(y,dtype(float))/10.0;
    z=array(z,dtype(float))/10.0;
    return H,x,y,z,deltax,deltay,deltaz;

def Hamiltonian_per(H,x,y,z,deltax,deltay,deltaz,aCC,thop,k):
    Np=size(x);
    Hnew=H.copy();
    conn_per=[]
    for ii in range(0,Np):
        xc=x[ii];
        yc=y[ii];
        zc=z[ii];
```

```
        # Here I compare with 1.05*aCC in order to take into account 
numerical tollerances
        indp=nonzero(sqrt((x-xc+deltax)**2+(y-yc+deltay)**2+(z-
zc+deltaz)**2)<aCC*1.05)[0]+1;
        indm=nonzero(sqrt((x-xc-deltax)**2+(y-yc-deltay)**2+(z-zc-
deltaz)**2)<aCC*1.05)[0]+1;
        if (size(indp)>0):
            for j in range(0,size(indp)):
                conn_per.append([ii+1,indp[j]]);
        if (size(indm)>0):
            for j in range(0,size(indm)):
                conn_per.append([ii+1,indm[j]]);


    del ii
    Nconn=len(conn_per);
    for ii in range(Nconn):

ind=nonzero((H[:,0]==conn_per[ii][0])&(H[:,1]==conn_per[ii][1]))[0]
        if (size(ind)>0):
            if (deltax>0):
                segno=sign(x[int(abs(H[ind,0]))-1]-
x[int(abs(H[ind,1]))-1]);
                Hnew[ind,2]=H[ind,2]+thop*exp(-segno*k*deltax*1j);
            elif (deltay>0):
                segno=sign(y[int(abs(H[ind,0]))-1]-
y[int(abs(H[ind,1]))-1]);
                Hnew[ind,2]=H[ind,2]+thop*exp(-segno*k*deltay*1j);
            else:
                segno=sign(z[int(abs(H[ind,0]))-1]-
z[int(abs(H[ind,1]))-1]);
                Hnew[ind,2]=H[ind,2]+thop*exp(-segno*k*deltaz*1j);
        else:
            if (conn_per[ii][0]<conn_per[ii][1]):
                if (deltax>0):
                    segno=sign(x[conn_per[ii][0]-1]-x[conn_per[ii][1]-
1]);

temp=array([conn_per[ii][0],conn_per[ii][1],thop*exp(-
segno*k*deltax*1j)]);
                elif (deltay>0):
                    segno=sign(y[conn_per[ii][0]-1]-y[conn_per[ii][1]-
1]);

temp=array([conn_per[ii][0],conn_per[ii][1],thop*exp(-
segno*k*deltay*1j)]);
                else:
                    segno=sign(z[conn_per[ii][0]-1]-z[conn_per[ii][1]-
1]);

temp=array([conn_per[ii][0],conn_per[ii][1],thop*exp(-
segno*k*deltaz*1j)]);
```

```
                Hnew=vstack([Hnew,temp]);
    del ii
    return Hnew


class nanoribbon_fast_ohmic:
    acc=0.144;
    def __init__(self,n,L):
        self.Nc=int(4*(floor((floor(L/nanoribbon_fast_ohmic.acc)-
1)/3)));
        self.n=n;
        self.Phi=zeros(n*self.Nc);
        self.Eupper=1000.0;
        self.Elower=-1000.0;
        self.dE=1e-3;
        self.thop=-2.7;
        self.eta=1e-8;
        self.mu1=0;
        self.mu2=0;
        self.Temp=300;
        self.E=zeros(NEmax);
        self.T=zeros(NEmax);
        self.charge=zeros(self.n*self.Nc);
        self.rank=0;
        self.atoms_coordinates();
        self.defects_list=[]
        self.onsite_E=-1.5;
    def atoms_coordinates(self):
        GNR_atoms_coordinates(self);
        self.x=array(self.x);
        self.y=array(self.y);
        self.z=array(self.z);
        return;
    def gap(self):
        return GNRgap(self);
    def charge_T(self):

        M=self.Nc;
        N=self.n;
        t=self.thop;
        Energy = 0.0
        Ene = 0.0
        p = 0.0
        d = 0.0
        orbitals = [1, 0]
        hamiltonian = []
        zeroes = [0, 0, 0, 0]
        ene = [Energy, 0, 0, Ene]
        coupling1 = [t, 0, 0, p]
        coupling2 = [t*1.12, 0, 0, p]
        orbitals = orbitals + zeroes
        hamiltonian.append(orbitals)
```

```
for j in range(M):
    for i in range(N):
        n = i + 1 + j*N
        p = [n,n]
        p =  p + ene
        hamiltonian.append(p)



for j in range(1, M-1, +4):
    for i in range(1, N):
        n = i + 1 + j*N
        m = i + (j+1)*N
        p = [n,m]
        p =  p + coupling1
        hamiltonian.append(p)
  #      hamiltonian.append([m, n, t, p, d])

for j in range(3, M-1, +4):
    for i in range(0, N-1):
        n = i + 1 + j*N
        m = i + 2 + (j+1)*N
        p = [n,m]
        p =  p + coupling1
        hamiltonian.append(p)
   #     hamiltonian.append([m, n, t, p, d])


# nell'if ripristinare il fattore t*1.12
for j in range(0, M-1, +4):
    for i in range(N):
        n = i + 1 + j*N
        m = i + 1 + (j+1)*N
        if i == 0:
              p = [n,m]
              p =  p + coupling2
              hamiltonian.append(p)
    #          hamiltonian.append([m, n, t*1.12, p, d])
        else :
              p = [n,m]
              p =  p + coupling1
              hamiltonian.append(p)
     #        hamiltonian.append([m, n, t, p, d])

for j in range(1, M-1, +4):
    for i in range(N):
        n = i + 1 + j*N
        m = i + 1 + (j+1)*N
        p = [n,m]
        p =  p + coupling1
        hamiltonian.append(p)
```

```
#             hamiltonian.append([m, n, t, p, d])

        # nell'if ripristinare il fattore t*1.12
        for j in range(2, M-1, +4):
            for i in range(N):
                n = i + 1 + j*N
                m = i + 1 + (j+1)*N
                if i == (N-1):
                      p = [n,m]
                      p =  p + coupling2
                      hamiltonian.append(p)
                #                 hamiltonian.append([m, n, t*1.12, p,
d])
                else :
                      p = [n,m]
                      p =  p + coupling1
                      hamiltonian.append(p)
#                 hamiltonian.append([m, n, t, p, d])

        for j in range(3, M-1, +4):
            for i in range(N):
                n = i + 1 + j*N
                m = i + 1 + (j+1)*N
                p = [n,m]
                p =  p + coupling1
                hamiltonian.append(p)
#          hamiltonian.append([m, n, t, p, d])

        H = Hamiltonian(N,M)
        # I work on the defects
        ind=array(self.defects_list,dtype=int);
        H.H=array(hamiltonian,dtype=complex)
        H.H[ind,2]=self.onsite_E;
        H.Eupper = self.Eupper;
        H.Elower = self.Elower;
        H.rank=self.rank;
        H.dE=self.dE;
        H.Phi=self.Phi;
        H.Ei=-self.Phi;
        H.eta=self.eta;
        H.mu1=self.mu1;
        H.mu2=self.mu2;
        H.Egap=self.gap();

        H.charge_T()

        self.E=array(H.E);
        self.T=array(H.T);
        self.charge=array(H.charge);
        del hamiltonian,H
        return;
```

```
    def current(self):
        vt=kboltz*self.Temp/q;
        E=array(self.E);
        T=array(self.T);
        arg=2*q*q/(2*pi*hbar)*T*(Fermi((E-self.mu1)/vt)-Fermi((E-
self.mu2)/vt))*self.dE
        return sum(arg);


# This is the class for the solution of the 1D drift-diffusion
class multisubband1D:
    def __init__(self, nx, ny, Neig):
        self.ny=ny;
        self.nx=nx;
        self.x=zeros(nx);
        self.y=zeros(ny);
        self.Phi=zeros(nx*self.ny);
        self.Ei=zeros(nx*self.ny);
        self.Egap=zeros(nx*self.ny);
        self.Temp=300;
        self.charge=zeros(nx*self.ny);
        self.rank=0;
        self.Neig=Neig;
        self.Psi=zeros((nx*ny,Neig));
        self.eig=zeros((ny,Neig));
        self.mass=zeros((nx,ny));
        self.mu=100e-4*ones(self.ny);
        self.genric=zeros(self.ny);
        self.n1d=zeros(self.ny);
        self.ecs=zeros(self.ny);
        self.charge_left_contact=0;
        self.charge_right_contact=0;
        self.tolljay=1e-3;

# This is the class for the solution of the QM 1D
class QM1D:
    def __init__(self, nx, Neig,gridx,p=None,charge_T=None):
        if charge_T is not None:
            self.charge_T=types.MethodType(charge_T,self);
        self.nx=nx;
        self.x=zeros(nx);
        self.ny=1;
        ny=1;
        self.Phi=zeros(nx*self.ny);
        self.Ei=zeros(nx*self.ny);
        self.Temp=300;
        self.charge=zeros(nx*self.ny);
        self.rank=0;
        self.Neig=Neig;
        self.Psi=zeros((nx*ny,Neig));
        self.eig=zeros((ny,Neig));
        if p is not None:
```

```
            self.Egap=p.Egap;
            self.massl=p.mel
            self.masst=p.met;
            self.massh=p.mhole
            self.chi=p.chi
            self.mass=p.mel;
        else:
            self.Egap=zeros(nx*self.ny)
            self.massl=zeros(nx*self.ny)
            self.masst=zeros(nx*self.ny)
            self.massh=zeros(nx*self.ny)
            self.chi=zeros(nx*self.ny)
            self.mass=zeros(nx*self.ny)
        self.Ef=0;
        self.x=gridx;
        self.ecs=zeros(self.ny);
    def charge_T(self):
        del self.charge
        self.charge=zeros(self.nx*self.ny);
        self.Ei=-self.Phi;
        # I compute the confined electrons
        dist=avervect(self.x)
#        self.Ei=4.05-self.Phi-self.chi-self.Egap*0.5
        self.mass=self.massl;
        solve_schroedinger_1D(self);
        vt=self.Temp*kboltz/q;
        for i in range(0,self.Neig):
            self.charge=self.charge-2*dist*1e-
9*(self.Psi[:,i])**2*self.masst*m0*kboltz*self.Temp/pi/hbar**2*log(1+e
xp(-(self.eig[0,i]-self.Ef)/vt));
        self.mass=self.masst;
        solve_schroedinger_1D(self);
        vt=self.Temp*kboltz/q;
        for i in range(0,self.Neig):
            self.charge=self.charge-4*dist*1e-
9*(self.Psi[:,i])**2*self.massl*m0*kboltz*self.Temp/pi/hbar**2*log(1+e
xp(-(self.eig[0,i]-self.Ef)/vt));
        # I now add the holes
        for i in range(0,size(self.charge)):
            self.charge[i]=self.charge[i]+dist[i]*1e-
9*(2/sqrt(pi))*2*(vt/(2*pi)*(self.massh[i]*m0/hbar)*(q/hbar))**1.5*fph
alf((self.Ei[i]-self.Egap[i]*0.5-self.Ef)/vt)

        return;
    def current(self):
        return 0;
```