Technische Universität Wien

Diplomarbeit

Three-Dimensional Device Simulation with MINIMOS-NT using the WAFER-STATE-SERVER

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs
unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Klaus-Tibor Grasser
und
Univ.Ass. Dipl-Ing. Andreas Gehring
E360 - Institut für Mikroelektronik

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Robert Entner
9725877 / E754
Neustiftg. 20/6, 1070 Wien
eMail: robert@entner.net

Wien, im September 2003

To Anna and Sarah

Abstract

The fast growing market for semiconductor devices and the rapid development of new technologies in this segment has lead to the development of numerous simulation tools. They cover both the simulation of the manufacturing process of a device as well as the simulation of its electrical characteristics.

All these simulation tools cover different stages in the development of semiconductor devices. So it is obvious that choosing those tools that are best suited for the different development steps and combining them to simulate the whole production flow would yield the best results.

However, there is no standard description of simulation data and for reading and writing it from and to a file. To solve this issue the WAFER-STATE-SERVER was developed at the Institute for Microelectronics. It can handle various file formats from different vendors and can be expanded to support new formats with minimal effort.

Within this thesis the general-purpose device and circuit simulator MINIMOS-NT was enhanced with the capability of reading and writing data from and to the WAFER-STATE-SERVER. This allows MINIMOS-NT to interact with various other simulators and to incorporate it into the workflow of device development.

This work presents the design tools and data models that were necessary for the implementation of the new functionality into MINIMOS-NT and for testing it. Also the issues arising during the work and its solutions are presented. For reference a short introduction to the WAFER-STATE-SERVER is given which is used for reading, manipulating, and writing simulation data. This description should make it easy for beginners to access the WAFER-STATE-SERVER.

To demonstrate the functionality of the implementation, several three-dimensional devices are created and simulated.

Kurzfassung

Der schnelllebige Markt für Halbleiterbaulemente und die rasche Entwicklung neuer Technologien in diesem Segment hat zur Entwicklung zahlreicher Simulationswerkzeuge geführt. Sie dienen einerseits der Simulation des Herstellungsprozesses eines Bauelementes als auch seiner elektrischen Eigenschaften.

Verschiedene Stadien im Entwicklungsprozess eines Bauelementes werden durch die jeweiligen Simulationswerkzeuge abgedeckt. Da wäre es durchaus wünschenswert Werkzeuge verschiedener Hersteller für die unterschiedlichen Entwicklungsstufen zu kombinieren um die besten Ergebnisse zu erzielen.

Da es allerdings keinen einheitlichen Standard gibt um auf die Simulationsdaten zuzugreifen, ist es äußerst mühsam die Tools verschiedener Hersteller zu kombinieren. Um dieses Problem zu lösen wurde am Institut für Mikroelektronik der WAFER-STATE-SERVER entwickelt. Er kann verschiedenste Dateiformate diverser Hersteller verarbeiten und kann auch mit minimalem Aufwand erweitert werden.

In dieser Diplomarbeit wurde der Bauelement- und Schaltungssimulator MINIMOS-NT erweitert, um auf den WAFER-STATE-SERVER zugreifen zu können. Dadurch ist es MINIMOS-NT möglich mit den unterschiedlichsten Simulatoren zusammen zu arbeiten und sich in den Workflow der Bauelemententwicklung leicht einzugliedern.

Es werden jene Designwerkzeuge und Datenmodelle präsentiert, die für die Implementierung und den Test der neuen Funktionalität in MINIMOS-NT notwendig waren. Weiters wird auf die Probleme und deren Lösungen eingegangen, die während der Arbeit auftraten. Auch ist eine kurze Einleitung für die Ansteuerung des WAFER-STATE-SERVER enthalten, um einen schnellen Einstieg zu ermöglichen.

Um die Funktionalität der Implementierungen zu demonstrieren wurden einige dreidimensionale Bauelemente erstellt und simuliert.

Acknowledgment

I would like to express my deep gratitude to a number of people who supported me during my work on this Diplomarbeit and who provided me with an excellent working environment.

First of all I want to thank Prof. Siegfried Selberherr and Prof. Erasmus Langer who are responsible for the excellent working conditions at the Institute for Microelectronics. Both, the technical infrastructure and the amicable climate at the institute assisted me with my task.

Robert Klima, whose 'advertisement' on the web aroused my curiosity, was my first contact person to the Institute for Microelectronics. My thesis built up on the work he achieved in his Dissertation and he could help to increase my knowledge about the internals of the device and circuit simulator MINIMOS-NT.

Soon I got to know many helpful people at the institute. I was supported by everyone who could help me. These were namely Tibor Grasser who was the supervisor of my thesis. His knowledge about device simulation and semiconductors is tremendous. Andreas Hössinger who was not tired to answer my thousand questions about the WAFER-STATE-SERVER and its internals. Stephan Wagner who proved himself as master in debugging C++ code and in finding linker errors. Without his help I would have despaired. Enzo Ungersböck who I share my office with and answered all the little 'newbie questions'. Johann Cervenka who supplied me with a necessary C++ class. Andreas Gehring who acted as proof reader of my work. He did a great job in finding all these glitches in my thesis and in suggesting better verbalizations. Ewald Haslinger who provided me with hard-copies of the documents relevant for me. René Heinzl who was working on his thesis at the same time as I did. He could always surprise me with his skill for self motivation. And all the other members of the Institute for Microelectronics who instantly accepted me as one of theirs.

Also, I want to thank the Christian Doppler Gesellschaft for supporting this work.

But above all I want to thank my family. My loving wife Anna supported and encouraged me whenever possible. I also want to thank my parents who gave me the opportunity to attend the university and took the financial burden from me. My mother-in-law was supporting me whenever possible and proved to be one of the best cooks I am aware of.

And, of course, there is Sarah, who is still to come ...

Contents

List of Symbols

Notation

x	...	scalar
$\mathbf{x}$	...	vector
$\mathbf{x} \cdot \mathbf{y}$	...	scalar (in) product
$\mathbf{x} \times \mathbf{y}$	...	vector (ex) product
$\mathrm{div}(\cdot)$	...	divergence
$\mathrm{rot}(\cdot)$	...	curl
$\mathrm{grad}(\cdot)$	...	gradient
$\partial_t(\cdot)$	...	partial derivative with respect to t

Physical Quantities

ε	...	permittivity
μ	...	permeability
μ_b	...	mobility for carrier type b
ϱ	...	space charge density
σ	...	conductivity
$\tau_{\mathcal{E}\,b}$	...	energy relaxation time
ψ	...	electrostatic potential
n	...	electron concentration
p	...	hole concentration
C	...	net concentration of ionized dopants
T_b	...	carrier temperature
T_L	...	lattice temperature
E_C / E_V	...	band edge energy

$\mathbf{B}$	...	magnetic flux density
$\mathbf{D}$	...	dielectric flux density
$\mathbf{E}$	...	electric field
$\mathbf{H}$	...	magnetic field
$\mathbf{J}_b$	...	current density
R	...	net recombination rate

Constants

h	...	Planck's constant,	$6.6260755 \times 10^{-34}\,\mathrm{J\,s}$
k_B	...	Boltzmann's constant,	$1.3806503 \times 10^{-23}\,\mathrm{J/K}$
q	...	elementary charge,	$1.6021892 \times 10^{-19}\,\mathrm{C}$
m_0	...	electron rest mass,	$9.1093897 \times 10^{-31}\,\mathrm{kg}$

List of Figures

Terminology

Most abbreviations and technical terms in this thesis are self-explanatory. Care has been taken to use well defined and precise expressions. For an easy way to keep track of the terms and their meanings they are summarized in Table 1.

TCAD	Technology Computer Aided Design
Wafer-State	The data which describes the condition of a wafer. It contains the physical layout of the integrated devices, the structure of the simulation grid, and attributes like the impurity concentration or the temperature.
WAFER-STATE-SERVER	The class library used to store and manipulate simulated data [bind02].
MINIMOS-NT	Device and circuit simulator developed at the Institute for Microelectronics [mmnt].
SMARTV	Visualization tool developed at the Institute for Microelectronics. It is capable of handling `.wss` files from the WAFER-STATE-SERVER. A German description of the program can be found in [zohl03].
IDDL	Input Deck Description Language. It is used as a base for input files to control many process- and device simulation tools at the Institute for Microelectronics [klim02].
LAYGRID	Three-dimensional preprocessor/grid generator. It generates three-dimensional geometries and a tetrahedalized simulation grid [sap].
ADDANAIPD	Tool for adding dopant concentrations to devices. It can also be used to refine the grids.

Table 1: A glossary of terms used.

Chapter 1

Introduction

In the last decades the importance of electronic devices has grown rapidly. This success has been triggered by three scientists in the last century, namely John Bardeen, Walter Houser Brattain, and William Bradford Shockley, who invented the transistor at Bell Laboratories in December 1947. The transistor could replace the thermionic valve – also known as vacuum tube – in most applications for various reasons: Transistors are smaller in size, cheaper, easier to manufacture, have lower operation voltages, a lower power dissipation, a higher reliability, and a greater endurance.

In the sixties, the discrete transistor was replaced by integrated circuits. Without requiring much more space on the expensive wafers, one integrated circuit could replace dozens of transistors. Further progress in technology lead to a very high level of integration which enabled the development of microprocessors in the seventies.

Nowadays, progress in development is still significant. The level of integration becomes higher and higher while the size of the microchips is reduced to increase the capabilities of the chip and to decrease manufacturing costs. But as the size of integrated circuits is decreasing, new physical effects come up which have to be considered. As development time is a key factor for the manufacturers they simulate the fabrication process and the electrical behavior of transistors with Technology Computer Aided Design (TCAD) simulation programs to reduce development time.

Today, many tools from various software vendors are available to simulate the semiconductor fabrication process and the resulting devices. But unfortunately, most of them use different file formats to store their data. So it is extremely cumbersome to combine tools from different vendors to simulate the whole process and simulation flow.

The data which describes the condition of a wafer before or after a simulation process is called Wafer-State. It contains the physical layout of the integrated devices, the structure of the simulation grid, and attributes like the impurity concentration or the temperature. This information, or part of it, is needed by the simulation tools. To have a common database the WAFER-STATE-SERVER was developed at the Institute for Microelectronics. It represents an application programming interface (API) with a set of classes to store the state of a wafer. Using the WAFER-STATE-SERVER the tool developers have a common database to store and retrieve their data.

The WAFER-STATE-SERVER has its own file format but can also read and write proprietary file formats. So it can be used as a wrapper tool to feed the output from the simulation tool of one vendor to the tool of another vendor. This enables the use of the best suited tools for the various steps in the process and device simulation flow.

Within this thesis the device and circuit simulation tool MINIMOS-NT is connected to the WAFER-STATE-SERVER. With this enhancement it is now easy to integrate MINIMOS-NT in a simulation flow with the other tools developed at the Institute for Microelectronics.

Chapter 2

TCAD Design Tools and Data Models

Each vendor of Technology Computer Aided Design (TCAD) simulation tools 'invented' their own file format for storing the simulation data. Most of these file formats are optimized for the special needs of a specific tool. Therefore the file formats are not compatible and it is very difficult to translate them to another format. This is the reason why it is sometimes practically impossible to combine different tools for the simulation of one process flow. Hence, a data model was created which can store Wafer-State information in a tool-independent manner. This information must be sufficient for a wide range of tools and it should be possible to export and import the data to tool-specific file formats.

2.1 TCAD Data Model

A data model for TCAD simulation has to deal with four major aspects [bind02].

- The tool must store the results of a simulation for further processing. This could be another simulation step or visualization.
- The tool must manage different gridding tools. It must be possible to switch between gridding algorithms without much effort.
- The extraction of topological information must be supported. So, the user can manipulate the underlying geometry after a topography step which changes the structure of the device.
- All data structures and algorithms must be available in three spatial dimensions.

All above defined requirements are met in the WAFER-STATE-SERVER which is discussed in detail in Section 2.3.

2.2 Grid Theory

In TCAD applications the discretization of the simulation domain plays a major role. The continuous model must be discretized into a large number of small elements to allow the solution of the partial differential equations by numerical methods. With these partial differential equations the underlying physical behavior can be simulated. In the domain of device simulation this could be, for example, Poisson's equation for the electrostatic potential.

The discretization of a spatial domain is called mesh generation. The mesh has to fulfill different criteria depending on the problem that has to be solved.

One way of creating a mesh is the Delaunay triangulation, for which, in the first place, the Voronoï diagram has to be derived.

2.2.1 The Voronoï Diagram

Definition 2.1 (Voronoï diagram) *Let $P = \{p_1, \ldots, p_k\}$ be a finite set of points in the n-dimensional space R^n and their location vectors $\mathbf{x}_i \neq \mathbf{x}_j \; \forall \, i \neq j$. The region given by*

$$V(p_i) = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\| \; \forall \, j \neq i\}$$

is called Voronoï region (Voronoï box) associated with p_i and

$$\mathcal{V}(P) = \bigcup_{i=1}^{k} V(p_i)$$

is said to be the Voronoï diagram of P.

Figure 2.1 depicts a Voronoï diagram in two spatial dimensions. Every point within a Voronoï box is closer to the grid-point within the box than to any other grid-point.

The Voronoï theory can be expanded to three spatial dimensions. In this case the Voronoï boxes are three-dimensional and Definition 2.1 remains valid.

2.2.2 Delaunay Triangulation

The WAFER-STATE-SERVER uses tetrahedrons to describe its three-dimensional grids. To generate such grids the Delaunay triangulation can be efficiently utilized as robust tetrahedalization engine [flei99].

Starting from the Voronoï definition the Delaunay triangulation can be constructed. By joining the vertices belonging to two adjacent boxes the Delaunay triangulation depicted in Figure 2.2 is formed.

The Delaunay-based meshing approach consists of two tasks. One task addresses the mesh topography which is defined through the placement of mesh points within the simulation domain. The other task is to perform the Delaunay triangulation for this point set and thus creating the mesh topology. The sequence in which these tasks are carried out can be freely chosen and results into two different classes.

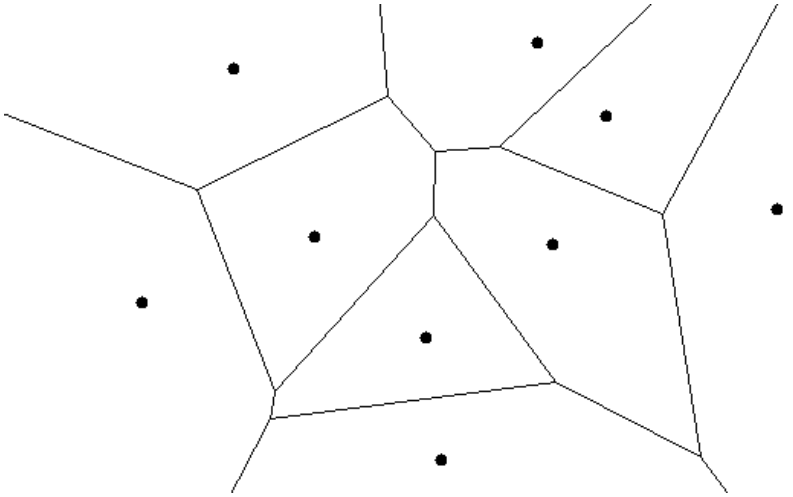

Figure 2.1: Voronoï diagram consisting of 9 points and the corresponding Voronoï boxes.

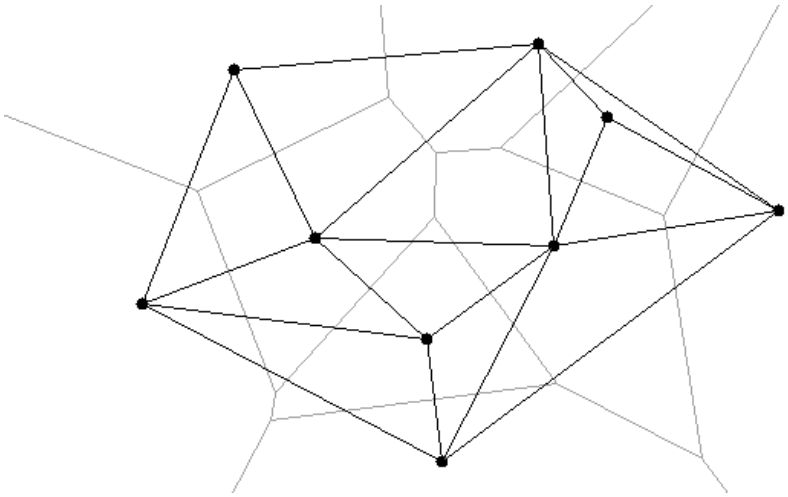

Figure 2.2: Voronoï boxes (light gray) and the corresponding Delaunay triangulation.

- The Delaunay triangulation is first computed for the boundary of the simulation domain (boundary mesh). Then the mesh points are inserted into the triangulation / tetrahedalization and the mesh topology is updated according to the Delaunay definition.
- The cluster of points filling the simulation domain is first created and the Delaunay Triangulation is performed afterwards.

Both approaches take advantage from the Delaunay triangulation and can also be combined for better results.

Creating an initial set of points at first is straightforward and easily allows to avoid overlapping or inconsistent elements (as the elements in this stage are only points). On the other hand a lot of information about the structure of the simulation domain is provided by a Delaunay boundary mesh. This information can be used to add internal mesh points in an 'intelligent' way.

2.3 WAFER-STATE-SERVER

To comply with the requirements of a TCAD data model the WAFER-STATE-SERVER was developed. It represents an API with a set of classes written in C++ to store and handle Wafer-State information.

The WAFER-STATE-SERVER stores all properties of the simulation domain, which mainly covers two major aspects:

- The topological structure of the simulation domain. It determines the position of various materials on the wafer that are grouped to segments. Figure 2.3 shows the topological structure of a MOS transistor.
- The distribution of physical quantities within the wafer. As an example, the dopant concentration is depicted in Figure 2.4.

Additionally, the WAFER-STATE-SERVER offers various methods to access and modify these properties. These comprise, for example, a method for interpolating an attribute from one grid to another or a method for creating a single grid from all grids of a certain segment.

2.3.1 Data Structure

The data stored by the WAFER-STATE-SERVER is organized in sections that can recursively contain subsections as shown in Figure 2.5. At the top level two sections are mandatory, namely the points and the segments sections.

In the points section the coordinates of all points of the grids are stored in a list. The grid elements themselves do not store the coordinates but only refer to the index of the points to prevent redundancy.

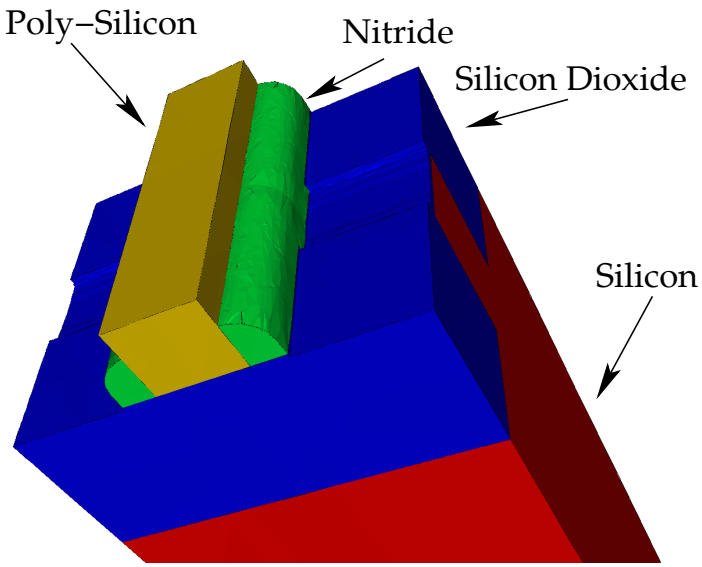

Figure 2.3: Topological structure of a MOS transistor.

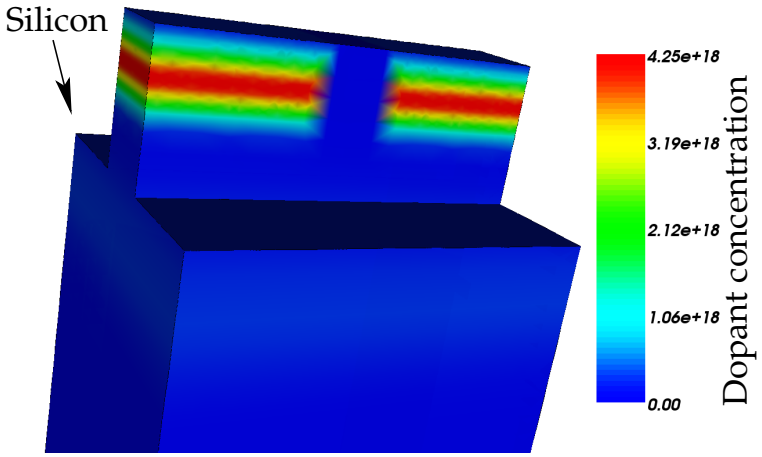

Figure 2.4: Dopant concentration of the Silicon segment shown in Figure 2.3.

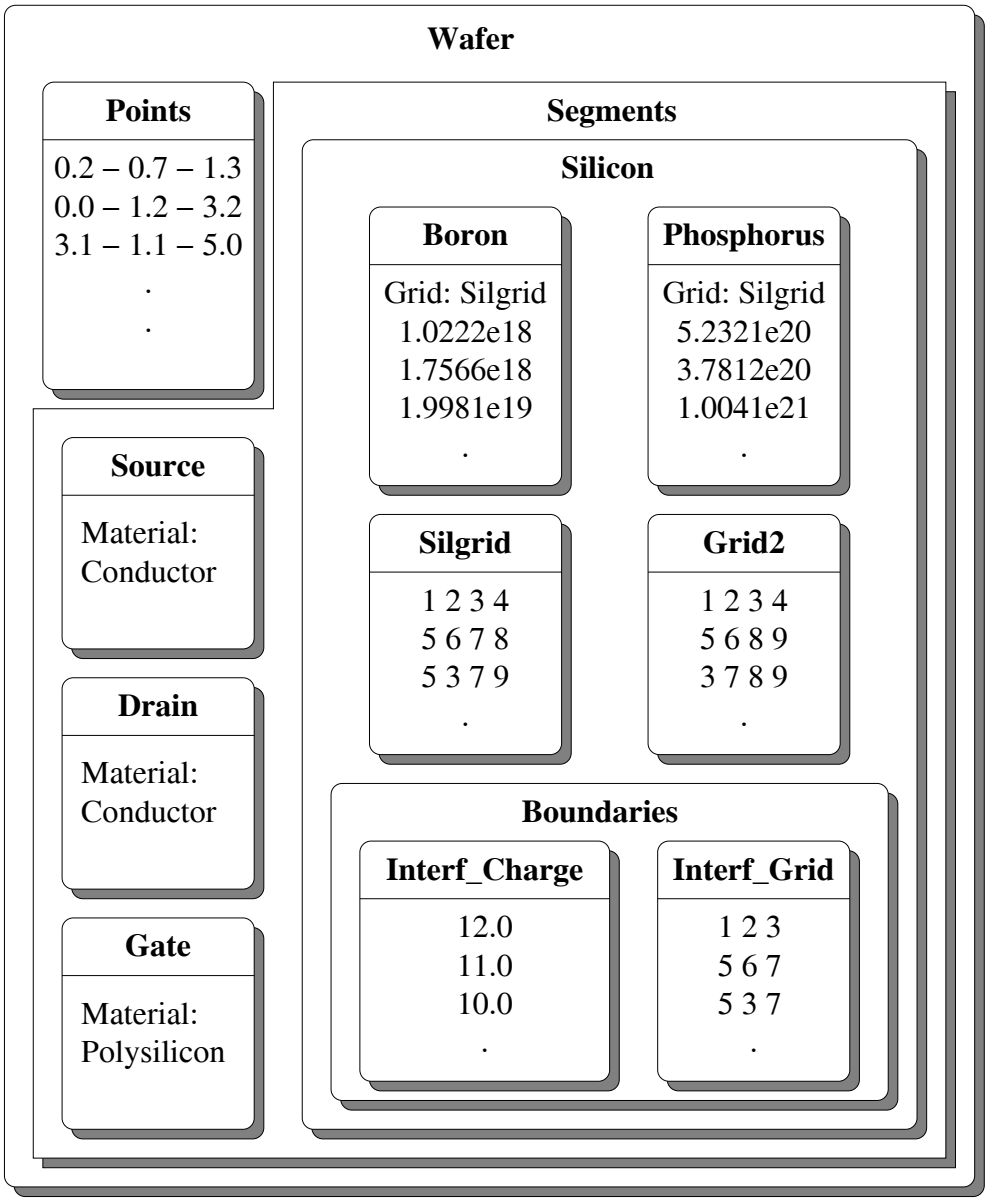

Figure 2.5: Structure of the wafer data as stored by the WAFER-STATE-SERVER.

In the segments section an arbitrary number of attributes, properties, boundaries, and stand-alone grids can be stored.

Attributes can be either quantities or properties.

Quantities are defined on a grid and store one or more values for each grid point. This could be for example the doping concentration, the electrostatic potential, or the electric field vectors.

Properties store a value for a whole segment. This could be for example the material type or the permittivity.

Boundaries define a part of the hull of a segment. They can hold an arbitrary number of quantities and properties.

Stand-alone grids represent the geometry of the segment. If at least one distributed quantity is defined on the segment, the stand-alone grid is optional. In this case the grid of the quantity can define the geometry of the segment. Otherwise it must be present.

2.3.2 Operations

The WAFER-STATE-SERVER must offer various operations on the wafer data. They are important to supply the simulation tools with the necessary information and to keep the wafer data consistent.

- Extraction of the interface between two segments.
- Extraction of the boundary of a segment.
- Extraction of the surface of a wafer.
- Interpolating the data from one grid to another. This operation must be explicitly invokable by the simulation tool.
- Merging the wafer with the result of a simulation. This is necessary for example when an etch simulation changes the topography of the wafer. It would not make sense if the removed parts of the wafer still had their grid points and quantities. The merging operation must hence compute new grids and interpolate the attributes onto the new grids.

2.3.3 Reader and Writer

The I/O module of the WAFER-STATE-SERVER comprises a set of classes and methods to write and read wafer data using different file formats.

As the classes for the reader and the writer are completely independent, it is only necessary to implement the operation needed (reading or writing) when a new file format has to be supported.

With this capability the WAFER-STATE-SERVER can be used as a file converter to transform data between different file formats. This functionality is used by the program `ioconv` developed at the Institute for Microelectronics.

Currently the I/O module supports five file formats.

WSS is the native WAFER-STATE-SERVER file format. It is human readable (ASCII), so it is possible to generate simple wafer structures with a text editor. Figure 2.6 shows an example `.wss` source file describing a simple silicon block consisting of one segment. Figure 2.7 depicts the resulting cube.

HDF (Hierarchical Data Format) is a binary format. It is maintained at the National Center for Supercomputing Applications at the University of Illinois at Urbana - Champaign [hdf]. As it is a binary file format it is smaller in size and the reading performance is better. The WAFER-STATE-SERVER supports HDF version 5 for reading and writing. HDF version 4 is not compatible to version 5 and thus neither readable nor writable by the I/O module.

FEM is the native file format of the program package SAP (Smart Analysis Programs) [sap] developed at the Institute for Microelectronics. Only the reader is implemented.

PIF is the old native file format of various tools from the Institute for Microelectronics at the Technical University of Vienna. It was initially proposed for the exchange of simulation data by Duvall [duva88] in 1988. However, there are various semantically incompatible PIF versions. The support by the I/O module has therefore been limited to some specific implementations. As all newly developed simulators from the Institute for Microelectronics are based upon the WAFER-STATE-SERVER this drawback is tolerable.

DF-ISE is the native file format of the ISE software company [ise]. To use the software suite of this vendor a reader and a writer were implemented.

2.4 The Input Deck Description Language

At the Institute for Microelectronics the powerful Input Deck Description Language (IDDL) was developed. It is used to control many of the device and process simulation tools.

The IDDL files normally have the extension `.ipd` and can be considered as control files for the applications. MINIMOS-NT for example needs a lot of information for a simulation run. There are the name of the device description file, the voltage each contact is to be set to, the name of the output file, the stepping details if for example the voltage at a contact is to be varied, just to name a few. Also the different material types like Silicon, Aluminium, or Poly-Silicon are defined in an IDDL material database. As the information is packed into the `.ipd` file one does not need an unmanageable amount of command-line parameters for the simulator.

But the `.ipd` files are not just a collection of parameters. The IDDL is a very powerful programming language capable of inheritance, various different operators, handling complex numbers,

```
VERSION "1.4"
NAME siliconblock
DIMENSION  3
POINTS
{
    10 10  0      # Point 0
     0 10  0      # Point 1
     0  0  0      # Point 2
    10 10 10      # Point 3
     5  5 10      # Point 4
    10  0  0      # Point 5
     0 10 10      # Point 6
    10  0 10      # Point 7
     0  0 10      # Point 8
}
SEGMENTS
{
    Segment1
    {
        GRID grid1
        {
            0 1 2 3
            1 2 3 4
            2 0 3 5
            2 3 4 5
            3 1 4 6
            3 4 5 7
            1 4 6 8
            1 2 4 8
            4 2 5 8
            4 5 7 8
        }
        ATTRIBUTES
        {
            MaterialType
            {
                "Si"
            }
        }
    }
}
```

Figure 2.6: A `.wss` file describing a simple silicon block consisting of one segment. The numbers in the `GRID` section refer to the coordinates defined in the `POINTS` section. Every line builds up one tetraeder consisting of four points. The resulting cube is depicted in Figure 2.7.

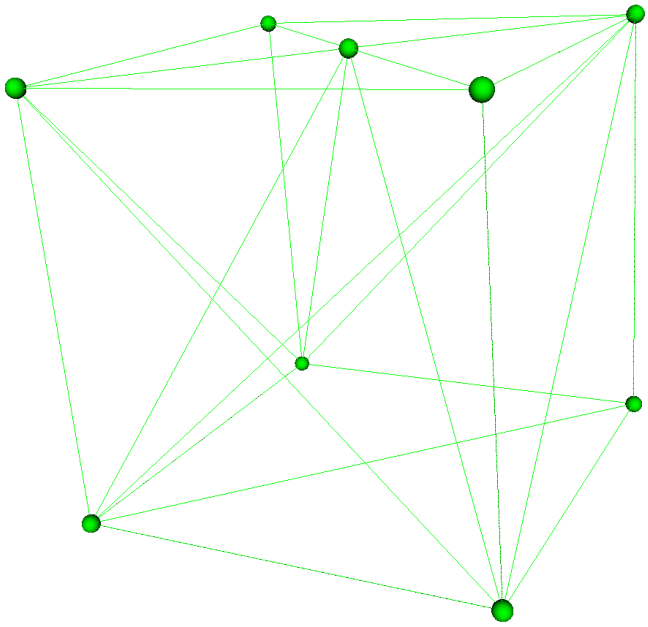

Figure 2.7: This cube was generated from the `.wss` source from Figure 2.6.

physical units, powerful functions, and much more. As it is also possible to define new functions within an `.ipd` file, the functionality can be improved. For a detailed description see [klim02].

Figure 2.8 shows an example of an `.ipd` file. It was used to simulate the diode in Section 5.3.

2.5 The Vienna Make Utility

The Vienna Make Utility, VMAKE [tupp96], is a system-independent, case-oriented make utility developed at the Institute for Microelectronics. VMAKE was developed to overcome the drawbacks of existing software engineering tools like MAKE or JAM.

MAKE lacks the ability to handle global software project information. Also dependence definitions are only handled locally.

JAM is able to use global dependences by identifying file names. The drawback JAM has to cope with is the lack of total system independence. So it is not possible to run JAM with the same set of description files on different platforms without any modifications.

Most of the process and device simulation tools and their libraries at the Institute for Microelectronics are managed by VMAKE, which can therefore be used as central software engineering tool to build the various projects.

```
  #include <defaults.ipd>  // load the default settings

  Device : DeviceDefaults // inherit the default device
  {
    Input  { file = "diode";     } // input file:  diode.wss
    Output { file = "diode_out"; } // output file: diode_out.wss

    // the voltage steps from -1V to 3V in 0.05V steps
    +LeftContact  = step(-1.0V,3.0V,0.05V);
    +RightContact = 0.0 V;

    Phys
    {
      srh = "*";
      sh  = "";  // do not simulate self heating

      +LeftContact
      {
        contactName = "LeftContact";
        Contact
        {
          Ohmic
          {
            type = "Voltage,Thermal";
            Rth  = 4.0e-4 "K cm^2/W";
          }
        }
      }
      +RightContact : LeftContact
      {
        contactName = "RightContact";
      }
    }
  }

Curve // information about a .crv output file
{
  file = "diode_out"; // output curve file: diode_out.crv
  Response            // data to include in the curve file
  {
     +I  = output("Device", "I", "LeftContact");
     +Ul = output("Device", "V", "LeftContact");
  }
}

// use the default iteration scheme
Iterate { Scheme : SchemeDefaults.DD; }
```

Figure 2.8: An example of an `.ipd` file.

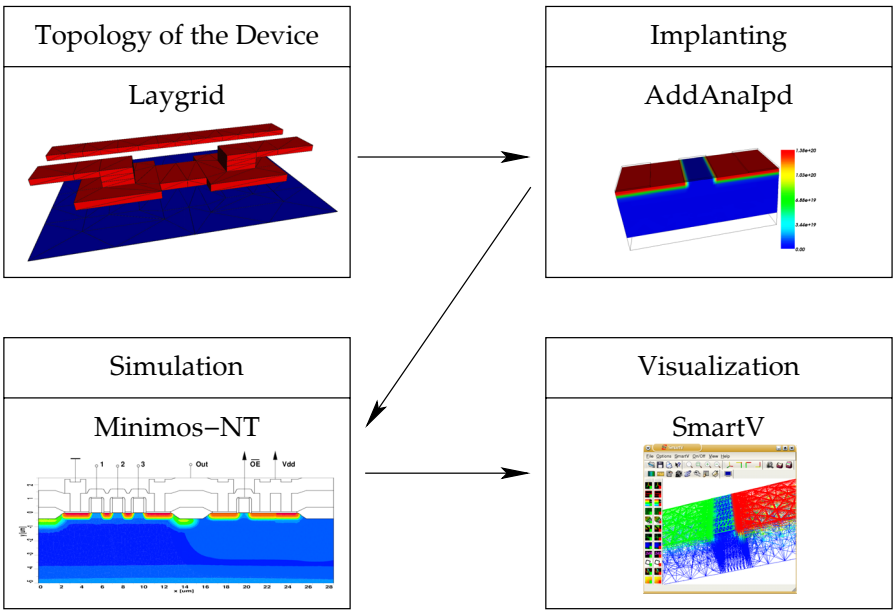

Figure 2.9: Workflow for generating, implanting, simulating, and visualization of a device.

2.6 TCAD Data Tools

Several tools for the generation, modification, and visualization of three-dimensional devices are developed at the Institute for Microelectronics. This section introduces the most important tools that have been used to simulate a complete process workflow.

Figure 2.9 depicts the workflow for the whole generation, implantation, simulation, and visualization process used within this work. Only analytic ways for generating a device and its properties were used. So no process simulation steps like etching, diffusion, or lithography were simulated. Instead the tools described in the following sections were utilized.

2.6.1 LAYGRID

The tool LAYGRID is part of the Smart Analysis Programs (SAP) program package. SAP was designed for the numerical simulation (finite elements method) and for the investigation of the thermal and the electrical characteristics of interconnect lines within integrated circuits. It consists of two finite element simulators (SCAP and STAP), three input preprocessors (CUTGRID, LAYGRID, and TRANSGRID) and three visualization tools (FEMPOST, SAPVIEW, and SMARTV). Table 2.1 depicts the tools of the package.

LAYGRID can be used to generate three-dimensional grid structures. It is basically a console tool, that reads the control information from a `.lay` file. The control files are human readable text-files.

Program	Description
CUTGRID	Two-dimensional preprocessor and grid generator. It generates two-dimensional surfaces and a triangulated grid.
LAYGRID	Three-dimensional preprocessor and grid generator. It generates three-dimensional geometries and a tetrahedalized simulation grid.
TRANSGRID	Three-dimensional preprocessor and grid generator. It generates three-dimensional geometries and a simulation grid with either tetrahedrons (which consist of four points and four faces) or hexaedrons (eight points and six faces).
SCAP	Extracts capacities from a structure of conductors. The insulators and conductors are assumed ideal.
STAP	Transient and static thermical and electrical simulation.
FEMPOST	Simple visualization tool.
SAPVIEW	Advanced visualization tool.
SMARTV	Sophisticated three-dimensional visualization tool. It is capable of reading the new `.wss` file format.

Table 2.1: The tools of the SAP package.

The three-dimensional geometry is modeled in terms of horizontal layers. These horizontal layers are defined by two-dimensional geometrical objects. The objects can be rectangles, polygons, or circles. Figure 2.10 depicts a part of an interconnect structure which has been generated with LAYGRID.

Figure 2.11 shows the `.lay` file that was used to generate the MOS transistor described in Section 5.3. In the first part of the `.lay` file the global scaling factor is defined. All geometric length specifications are multiplied with the factor `unitlength`.

Then the different layers are defined with the keyword `mask`. In this example the structures within the layers consist only of rectangles. The first number pair in the rectangle definition specifies the lower left corner of the object. The second number pair defines the width and the height.

In the third part of the example file the thickness of the layers and the order in which they are stacked is defined with the keyword `layerstructure`. Within this statement, the origin of the layers is defined first. Then the different layers are referenced by their label (`M0-M3`) and their thickness. The layers are separated by lines containing one ore more '-' signs.

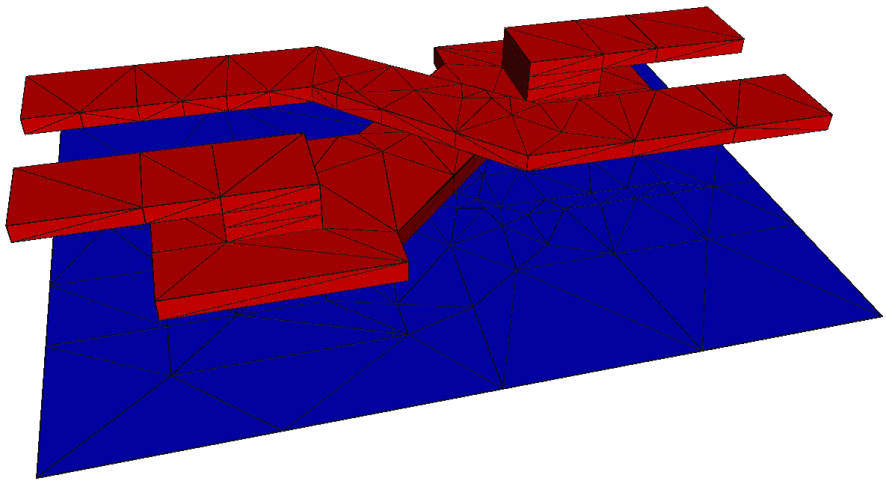

Figure 2.10: A typical interconnect structure generated with LAYGRID.

The last part defines the material types of the different geometrical objects. As one can see the gate contact is modeled with poly silicon.

When a `.wss` file has to be generated by LAYGRID, then the following command line can be used:

```
laygrid --wss --max-area 0.1 layfile.lay
```

The `--max-area` argument limits the maximum area of the generated triangles. This comes in very handy when the whole grid should be refined.

A drawback of this method is that the whole grid is refined. If an electrically interesting area of the simulation device is refined using this method, then also uninteresting areas of the device like for example in the lower parts of the substrate of a MOS transistor are refined and thus leads to an unnecessarily large amount of points. The following section shows some 'tricks' to refine the grid more intelligently.

```
unitlength { 1.0 um }  /* defines the unit of all numbers */

/* definition of the 2d layers */
mask { M0 rectangle { R1 Bulk    { -2 -1 } { 5 2 }}}
mask { M1 rectangle { R1 Si      { -2 -1 } { 5 2 }}
          rectangle { R2 Si      {  0 -1 } { 1 2 }}
          rectangle { R3 Si      {  0.05 -1 } { 0.9 2 }}
          rectangle { R4 Si      {  0.10 -1 } { 0.8 2 }}
          rectangle { R5 Si      {  0.15 -1 } { 0.7 2 }}
          rectangle { R6 Si      {  0.25 -1 } { 0.5 2 }}
          rectangle { R7 Si      {  0.35 -1 } { 0.3 2 }}
          rectangle { R8 Si      {  0.45 -1 } { 0.1 2 }}
}
mask { M7 rectangle { R18 SiO2   { -2 -1 } { 5 2 }}
          rectangle { R11 Source { -2 -1 } { 1 2 }}
          rectangle { R12 Drain  {  2 -1 } { 1 2 }}
}
mask { M8 rectangle { R10 SiO2   { -2 -1 } { 5 2 }}
          rectangle { R11 Source { -2 -1 } { 1 2 }}
          rectangle { R13 Gate   {  0 -1 } { 1 2 }}
          rectangle { R12 Drain  {  2 -1 } { 1 2 }}
}

/* the 2d layers are stacked and have the specified thickness */
layerstructure {
      origin   { 0 0 0 }
      plane    { ------------ }
      layer    { M0 0.5 }
      plane    { ------------ }
      layer    { M1 1.5995 }
      plane    { ------------ }
      layer    { M1 0.0005 }
      plane    { ------------ }
      layer    { M7 0.0025 }
      plane    { ------------ }
      layer    { M8 0.05 }
      plane    { ------------ }
}

/* definition of the material types of the segments */
material { Source Al   }
material { Gate   Poly }
material { Drain  Al   }
material { Bulk   Al   }
material { SiO2   SiO2 }
material { Si     Si   }
```

Figure 2.11: An example of a `.lay` file.

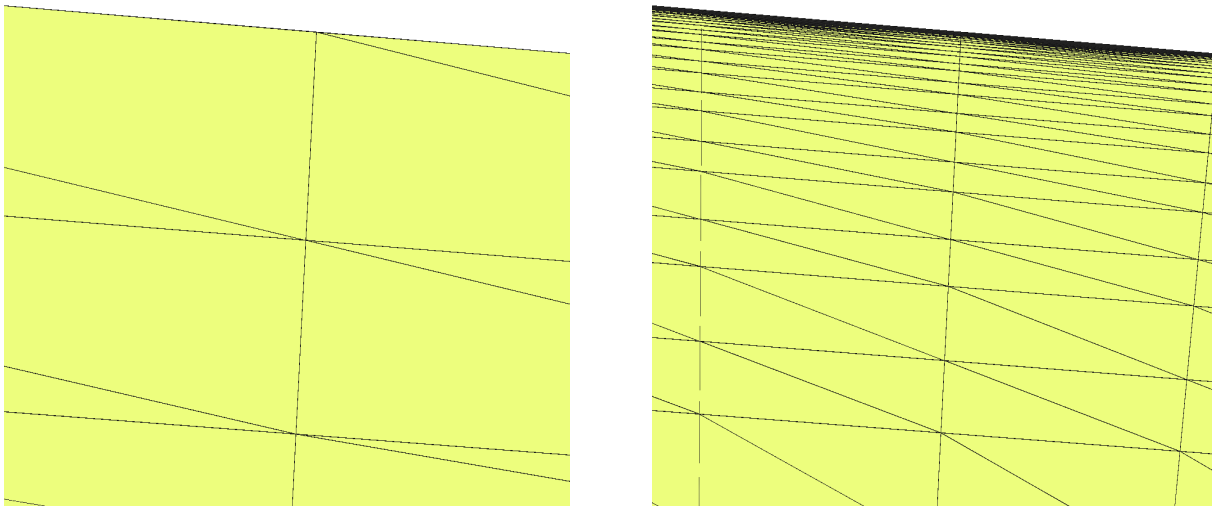

Figure 2.12: Part of a MOSFET grid without (left) and with grid refinement via a dummy layer.

Grid Refinement

LAYGRID offers some 'tricks' to refine the grid of the generated device in specific areas.

When a MOSFET is simulated, for example, the interesting area of the device is the uppermost part of the substrate where the channel builds up. So the grid in the first few nanometers under the gate oxide must be very fine and with growing depth the grid has to become coarser to avoid unnecessary grid points. Figure 2.12 depicts a part of a grid without and with this grid refinement.

LAYGRID offers a method to generate such grids by introducing so called 'dummy layers' into the grid structure. If we take the substrate layer of a MOSFET as example it can be split up into two layers where the first one has nearly the thickness of the original layer and the second builds up the rest. So, LAYGRID is forced to add a grid plane between the two layers. Since the grid line spacing cannot change by more than a predefined factor between consecutive grid lines, LAYGRID adds additional grid planes to the substrate to build up a strong refinement close to the dummy layer which gets coarser when moving away from this layer. This is exactly the type of grid refinement needed for the simulation of a MOSFET and its channel on the upper side of the substrate.

Figure 2.13 shows the different layerstructures for a grid with- and without refinement. The resulting device structure is exactly the same, that means that no 'extra segment' is generated by the dummy layer. Only the grid is different.

Another way to refine the grid structure is to use so called 'dummy faces'. They do not lead to a better refined grid in z direction, but instead refine the grid in the x and y directions.

```
/* without grid-refinement */
layerstructure {
  origin { 0 0 0 }
  plane  { --- }
  layer  { Bulk 0.05 }
  plane  { --- }
  layer  { Substrate 1.6 }
  plane  { --- }
  layer  { GateOxide 0.0025 }
  plane  { --- }
  layer  { Contacts 0.05 }
  plane  { --- }
}
```

```
/* grid-refinement via a dummy layer */
layerstructure {
  origin { 0 0 0 }
  plane  { --- }
  layer  { Bulk 0.05 }
  plane  { --- }
  layer  { Substrate 1.599 }
  plane  { --- }
  layer  { Substrate 0.001 } /* dummy layer */
  plane  { --- }
  layer  { GateOxide 0.0025 }
  plane  { --- }
  layer  { Contacts 0.05 }
  plane  { --- }
}
```

Figure 2.13: An example of grid refinement by using a dummy layer.

A dummy face is an additional geometrical object in a mask definition. Instead of using a simple rectangle to describe a gate contact of a transistor, for example, two rectangles are used. The first one stays the same as without dummy face. The second rectangle is positioned within the first one, and forces the grid generation algorithm to add new grid points. Figure 2.14 depicts two grids generated with and without a dummy face.

2.6.2 ADDANAIPD

The next step in the generation of a device like the MOS transistor of the examples section is the doping of the material and the refinement of the grid in the areas of interest. Those areas are, for example, in the substrate directly under the gate contact where the channel is formed. This partial refinement is important because a global refinement would lead to a grid with a tremendously higher amount of grid points.

The input files of ADDANAIPD are written in the Input Deck Description Language (IDDL) described in Section 2.4. Figure 2.15 shows parts of the input file used for the MOS transistor of Section 5.3.

In the `GridSpacing` section the global grid can be refined. The coordinates denote the maximum distance between two grid points in the x, y, and z direction. The refinement of the whole grid with this command should be used with care. The size of the simulation data can grow rapidly when the whole grid is refined instead of just the interesting areas.

The `GridRefinement` section is used to refine the grid within a given area. `BoxLL` and `BoxUR` specify the lower left and the upper right corners of the box. The `BoxSpacing` command is used in the same way as `GridSpacing`, but in this case only the grid within the box is refined. For all the areas within the simulation domain where a quantity changes its value very quickly a refinement box should be used to increase the grid resolution.

In the `Wafer` section the in- and output files are given.

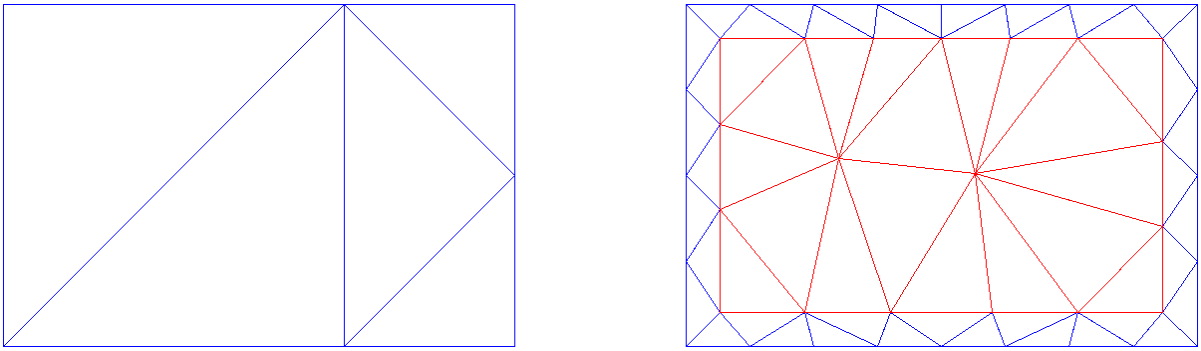

Figure 2.14: A Grid generated without (left) and with a dummy face. The refinement is achieved by the inner rectangle which has the same material type as the original but forces the grid generator to insert additional grid points.

Finally, in the `Attributes` section a quantity can be written on the segments of the device. In the example depicted in Figure 2.15 this is the p-type doping of the substrate. The doping profile has been specified via analytic functions as seen in the example. The n-type doping of the device was not included in this example to increase its readability, but the definition looks very much the same as in the p-type case.

This example file makes use of many sophisticated capabilities of the IDDL. New variables are defined and functions are used to calculate the dopant profile. The `VAL` variable returns the resulting value and depends on the X, Y, and the Z coordinate of the point. To get a deeper insight into the IDDL see [klim02].

ADDANAIPD produces a `.wss` file which includes new grids and new quantities, namely impurity concentrations. This file can then be used as direct input for the device simulator MINIMOS-NT.

```
AddAnalytical
{
  GridSpacing // global refinement of the grid
  {
    XYZ = [0.3,0.5,0.3];
  }
  GridRefinement // refinement of the n-channel area
  {
    BoxChannel
    {
      BoxLL      = [0.0, -1.0, 1.645];
      BoxUR      = [1.0,  1.0, 1.655];
      BoxSpacing = [0.1,  0.4, 0.001];
    }
  }
  Wafer // in- and output files
  {
    Input
    {
      filetype = "wss";
      filename = "simple.wss";
    }
    Output
    {
      filetype = "wss";
      filename = "simple_doped.wss";
    }
  }
  Attributes
  {
    Si
    {
      Boron_Active_Body
      {
        ext X;
        ext Y;
        ext Z;
        species = "Boron_Active";
        C       = 1.6e17;
        C2      = 2.0e16;
        height  = 1.65;
        value   = if(Z>height-0.09, 1.0, 0.0);
        delta   = if(Z>height-0.025, 1.0, 1/(0.09-0.025) * abs(height-0.09-Z));
        VAL     = (C2 + (C * value * delta)) * 1.0 "1/cm^3";
      }
      // n-type doping omitted
    } // Si
  } // Attributes
} // AddAnalytical
```

Figure 2.15: An example of an ADDANAIPD input file.

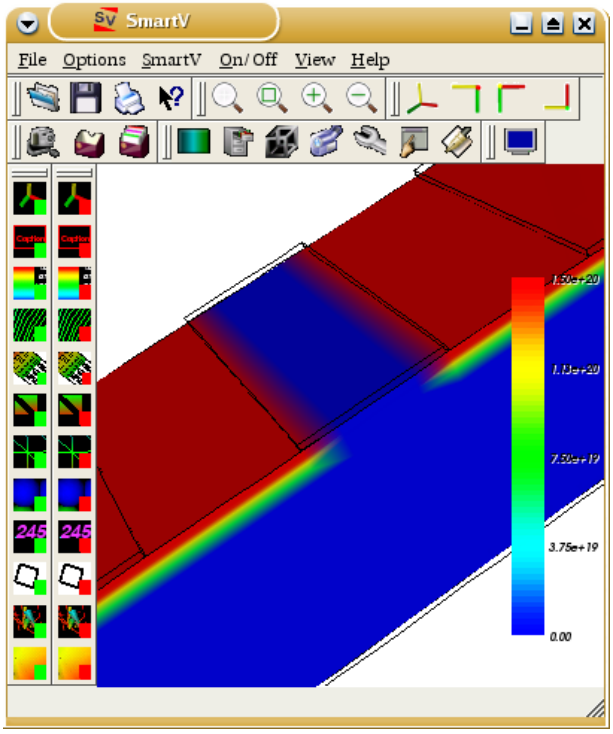

Figure 2.16: The SMARTV main window. Parts of the doping profile of a MOS transistor are visualized.

2.6.3 SMARTV

SMARTV can be used to visualize the topology of a simulation domain and the quantities and their distribution across the simulated device. The pictures in Section 5.3 have been created with SMARTV.

SMARTV has recently been developed at the Institute for Microelectronics [zohl03]. It is written in C++ using the QT widget set from Trolltech [trol] and the Visualization Toolkit (VTK) [vtk]. Figure 2.16 depicts the main window and its function buttons.

SMARTV is equipped with a wide range of functions. The buttons on the left hand side can be used to switch different kinds of visualization on or off. These are, for example, the grid structure, the grid points, the surface, the outline, and many more.

The buttons on top are mainly for invoking different kinds of options menus. For example the color-bar option where the range or the scale of the color-bar can be changed, or the screenshot function which was used to generate many images within this thesis.

A good documentation of the tool, which is only available in German right now, and a step by step manual on installing SMARTV, can be found in [zohl03].

```
#n              Vl              Id
#u               V               A
   -7.60000000e+00 -1.26206244e-17
   -7.50000000e+00 -1.26206245e-17
   -7.40000000e+00 -1.26206245e-17
   -7.30000000e+00 -1.26206244e-17
        ...             ...
```

Figure 2.17: An example of a `.crv` file.

2.6.4 XCRV

MINIMOS-NT writes simulation output which has been specified in the `Curve.Response` section to a `.crv` file.

A `.crv` file can hold all characteristic values when an input parameter is stepped. So it is for example possible to calculate and store the characteristic curve of a diode by stepping the potential of one of the contacts and store the potentials together with the calculated currents in a `.crv` file.

Figure 2.17 shows the first few lines of an example `.crv` file. The first line after the # symbol determine the names of the columns. The optional second line determines the units. Then the values follow in a list. When more input parameters are stepped in a simulation run, for example to get the characteristic curves at different simulation temperatures, then the different data blocks are separated by an empty line.

For visualization of such `.crv` files the Python script XCRV has been developed at the Institute for Microelectronics. XCRV is a preprocessor for the plotting tool XMGRACE[1].

XMGRACE is a powerful WYSIWYG two-dimensional plotting tool for the X Window System. Figure 2.18 depicts the main window with the plot of a dopant profile. It has a convenient point-and-click graphical user interface and is capable of displaying two-dimensional curves in various different styles with different line types, different marker symbols, different colors and so on. The settings can all be changed by using the Motif/Lesstif user interface of XMGRACE. In order to have a more convenient way of modifying the style settings for more than one plot the program is also capable of setting the parameters via console options. This is where the work of XCRV begins. In order to have a uniform design for all curves within one report it offers style files. Within a style file all options for the visualization of the curves are set. This style file can then be used to generate all plots within one report to guarantee a uniform design.

XCRV not only offers style files but has also some special functions that can be applied to columns of the input file. So it is for example possible to multiply all currents of a simulated device by 2 before the data is displayed with XMGRACE. This functionality comes in very handy when for example only one half of a symmetric device has been simulated and the resulting currents have to be multiplied by 2.

[1] XMGRACE was developed by the Grace Development Team and can be downloaded at: `http://plasma-gate.weizmann.ac.il/Grace/`.

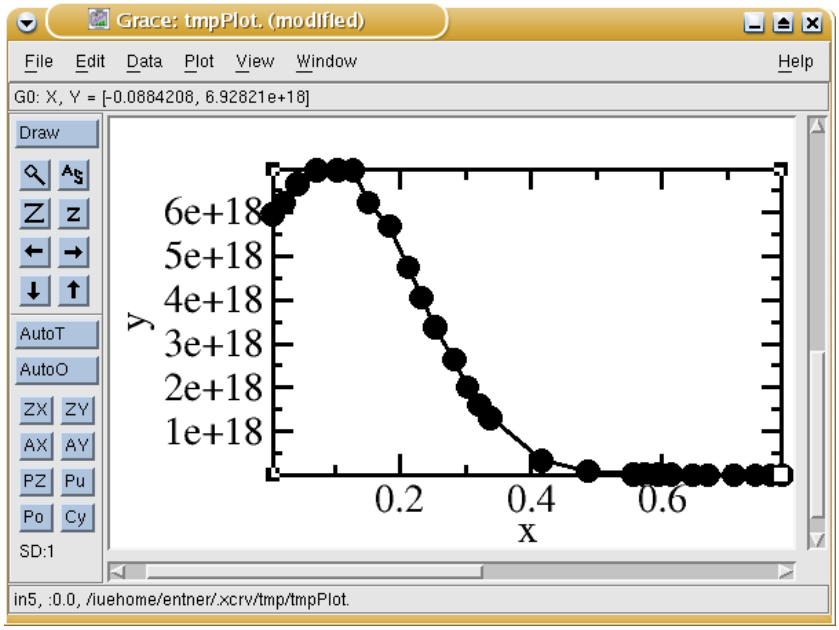

Figure 2.18: The main window of XMGRACE showing the plot of a dopant profile within a MOSFET.

Example

An example command-line to generate a plot of an output characteristics of a MOSFET could look like this[2]:

```
xcrv mosfet_out.crv -x Drain Id/1e-6 -xl 'Drain Voltage [V]' -yl \
     'Drain Current [\xm\f{}A]' -lt 'V\sGS\N = 0.8 V' -r 0,0,2,6
```

The file `mosfet_out.crv` is loaded and the columns `Drain` and `Id` are plotted where the values of `Id` are divided by 10^6. This causes the unit of the currents to be converted from [A] to [µA] in this example.

The parameters `-xl` and `-yl` set the axis description texts, `-lt` sets the legend text for the curve, and `-r` limits the data range from x/y: 0/0 to x/y: 2/6.

More Information about XCRV and its options can be found in [mmnt].

[2]The trailing \ indicates that the whole text is put into one line.

Chapter 3

Device Simulation with MINIMOS-NT

MINIMOS-NT is a general purpose device and circuit simulator. In its actual release 2.0 it is capable of two-dimensional simulations. It was extended to a multi-dimensional device simulator capable of two- and three-dimensional device simulations by Klima [klim02].

MINIMOS-NT can calculate carrier transport using either the drift-diffusion (DD) or the hydrodynamic (HD) transport model. The following section provides a glimpse on the physical models implemented in MINIMOS-NT.

3.1 Device Simulation Equations

Simulating microelectronic devices means to calculate the charge transport within the simulation domain. Charge transport can be described by the motion of electrons and holes which leads to the current flow through the device.

The basic equations solved by a device simulator are the Poisson equation, which can be derived from Maxwell's equations, and the continuity equations for electrons and holes [selb84, mmnt].

$$\operatorname{div}(\varepsilon \cdot \operatorname{grad} \psi) = \mathrm{q} \cdot (n - p - C) \tag{3.1}$$

$$\operatorname{div} \mathbf{J}_n = \mathrm{q} \cdot \left(R + \frac{\partial n}{\partial t} \right) \tag{3.2}$$

$$\operatorname{div} \mathbf{J}_p = -\mathrm{q} \cdot \left(R + \frac{\partial p}{\partial t} \right) \tag{3.3}$$

C denotes the net concentration of the ionized dopants, ε is the dielectric permittivity of the semiconductor. The expression $R = R_n - G_n = R_p - G_p$ determines the net recombination rate where R_n/G_n and R_p/G_p are the recombination/generation rates for electrons and holes. The term $\mathrm{q} \cdot (n - p - C)$ forms the negative space charge density $-\varrho$.

The electrostatic potential ψ and the electron and hole concentrations n and p are the unknown quantities of this equation system and are to be calculated.

3.1.1 Drift-Diffusion Transport Model

The component of the current which is caused by the electric field is called drift current. The current density and the electric field are related by Ohm's law. The drift current can be split up into two equations for the electrons (n) and the holes (p), respectively.

$$\mathbf{J}_n^{\text{drift}} = \sigma_n \mathbf{E} \tag{3.4}$$
$$\mathbf{J}_p^{\text{drift}} = \sigma_p \mathbf{E} \tag{3.5}$$

The conductivities are:

$$\sigma_n = \mathrm{q} n \mu_n \tag{3.6}$$
$$\sigma_p = \mathrm{q} p \mu_p \tag{3.7}$$

Where μ_n and μ_p denote the mobility of the electrons and the holes.

The component which is caused by the thermal motion of the carriers is called diffusion current. It is driven by a gradient in the carrier concentration. The expressions for the diffusion currents are:

$$\mathbf{J}_n^{\text{diffusion}} = \mathrm{q} D_n \text{grad}\, n \tag{3.8}$$
$$\mathbf{J}_p^{\text{diffusion}} = -\mathrm{q} D_p \text{grad}\, p \tag{3.9}$$

Where D_n and D_p are the diffusion coefficients for electrons and holes. For conditions close to thermal equilibrium and for non-degenerated carrier systems (Boltzmann statistics), the diffusion coefficients are related to the mobilities by the Einstein relation:

$$D_n = \mu_n \frac{\mathrm{k_B} T_n}{\mathrm{q}} \tag{3.10}$$
$$D_p = \mu_p \frac{\mathrm{k_B} T_p}{\mathrm{q}} \tag{3.11}$$

The drift-diffusion transport model takes, as the name suggests, both described carrier transport effects into account and thus forms the drift-diffusion current relations:

$$\mathbf{J}_n = \mathrm{q} n \mu_n \mathbf{E} + \mathrm{q} D_n \text{grad}\, n \tag{3.12}$$
$$\mathbf{J}_p = \mathrm{q} p \mu_p \mathbf{E} - \mathrm{q} D_p \text{grad}\, p \tag{3.13}$$

3.1.2 The Hydrodynamic Transport Model

The hydrodynamic transport model expands the drift-diffusion transport model. With this model the carrier temperatures are allowed to differ from the lattice temperature [mmnt].

The current relations take the form:

$$\mathbf{J}_n = \mathrm{q}\mu_n n \left(\mathrm{grad}\left(\frac{E_C}{\mathrm{q}} - \psi \right) + \frac{\mathrm{k_B}}{\mathrm{q}} \cdot \frac{N_{C,0}}{n} \cdot \mathrm{grad}\left(\frac{nT_n}{N_{C,0}} \right) \right) \tag{3.14}$$

$$\mathbf{J}_p = \mathrm{q}\mu_p p \left(\mathrm{grad}\left(\frac{E_V}{\mathrm{q}} - \psi \right) - \frac{\mathrm{k_B}}{\mathrm{q}} \cdot \frac{N_{V,0}}{p} \cdot \mathrm{grad}\left(\frac{pT_p}{N_{V,0}} \right) \right) \tag{3.15}$$

T_n and T_p denote the carrier temperatures, E_C and E_V the position-dependent band edge energies, and $N_{C,0}$ and $N_{V,0}$ the effective density of states. The density of states are evaluated at some reference temperature T_0.

3.1.3 The Lattice Heat Flow Equation

MINIMOS-NT is capable of taking self heating effects in semiconductor devices into account. Therefore the lattice heat flow equation has to be solved:

$$\mathrm{div}(\kappa_L \cdot \mathrm{grad}\, T_L) = \rho_L \cdot c_L \cdot \frac{\partial T_L}{\partial t} - H \tag{3.16}$$

The coefficients ρ_L, c_L, and κ_L denote the mass density, the specific heat, and the thermal conductivity of the material. The model used to calculate the heat generation H depends on the transport model used.

When the drift-diffusion transport model is used, H equals the Joule heat:

$$H = \mathrm{grad}\left(\frac{E_C}{\mathrm{q}} - \psi \right) \cdot \mathbf{J}_n + \mathrm{grad}\left(\frac{E_V}{\mathrm{q}} - \psi \right) \cdot \mathbf{J}_p + R \cdot (E_C - E_V) \tag{3.17}$$

When the hydrodynamic transport model is chosen, then the relaxation terms are used:

$$H = \frac{3 \cdot \mathrm{k_B}}{2} \cdot \left(n \cdot \frac{T_n - T_L}{\tau_{\epsilon,n}} + p \cdot \frac{T_p - T_L}{\tau_{\epsilon,p}} \right) \tag{3.18}$$

3.1.4 The Quasi-Fermi Potential

In unipolar devices, like MOS capacitors or MOSFETs, it is possible to assume a constant quasi-Fermi potential for one carrier type. With this assumption the current density for this carrier type can be neglected.

Within MINIMOS-NT, for each device in a circuit a quasi-Fermi potential can be specified. The quasi-Fermi potential is then used to calculate the local concentration of the considered carrier type. This reduces the simulation time dramatically as no continuity equation needs to be solved for this carrier system. For the drift-diffusion transport model the equation system size is reduced by about 1/3 and for the hydrodynamic transport model the equation system is reduced by about 2/5 because the corresponding continuity equations and the energy flux equation are not solved.

3.1.5 Magnetic Effects

MINIMOS-NT can simulate the deflection of carriers due to an applied magnetic field (Lorentz force) in semiconductors. To achieve this the Poisson and continuity equations are solved together with the following equation system [mmnt]:

$$\mathbf{J}_n = -\sigma_n \cdot \operatorname{grad} \phi_n - \sigma_n \cdot \frac{1}{1 + (\mu_n^* \cdot \mathbf{B})^2}((\mu_n^* \cdot \mathbf{B} \times \operatorname{grad} \phi_n) - \mu_n^* \cdot \mathbf{B}(\mu_n^* \cdot \mathbf{B} \times (\mathbf{B} \times \operatorname{grad} \phi_n))) \quad (3.19)$$

$$\mathbf{J}_p = -\sigma_p \cdot \operatorname{grad} \phi_p - \sigma_p \cdot \frac{1}{1 + (\mu_p^* \cdot \mathbf{B})^2}((\mu_p^* \cdot \mathbf{B} \times \operatorname{grad} \phi_p) - \mu_p^* \cdot \mathbf{B}(\mu_p^* \cdot \mathbf{B} \times (\mathbf{B} \times \operatorname{grad} \phi_p))) \quad (3.20)$$

The symbols σ_n and σ_p denote the electric conductivities of the electrons and holes, ϕ_n and ϕ_p the electron and hole quasi-Fermi potentials, B is the magnetic field, μ_n^* and μ_p^* the hall mobility related to the normal mobility where $\mu_n^* = r_n \cdot \mu_n$ and r_n is the hall scattering factor. In MINIMOS-NT r_n and r_p are set to 1.15 and 0.7 by default and can be changed via the input deck (see Section 2.4).

3.2 Generation and Recombination Models

This section shows different models for calculating the generation and recombination of electrons and holes. The overall recombination rate R is computed as the sum of the recombination rates calculated by these models ($R = R_{\text{model1}} + R_{\text{model2}} + R_{\text{model3}} \cdots$). Whether MINIMOS-NT should use a certain model or not can be specified in the input deck.

Shockley-Read-Hall (SRH) and Surface Recombination describes carrier generation in space charge regions and recombination in high injection regions.

Auger Recombination is a process where three particles are involved and happens in equilibrium.

Direct (radiative) Recombination is of importance for direct bandgap semiconductors. It is proportional to the carrier concentrations.

Trap Assisted Band-to-Band Tunneling describes the generation of carriers in high field regions. It is modeled similar to the SRH process.

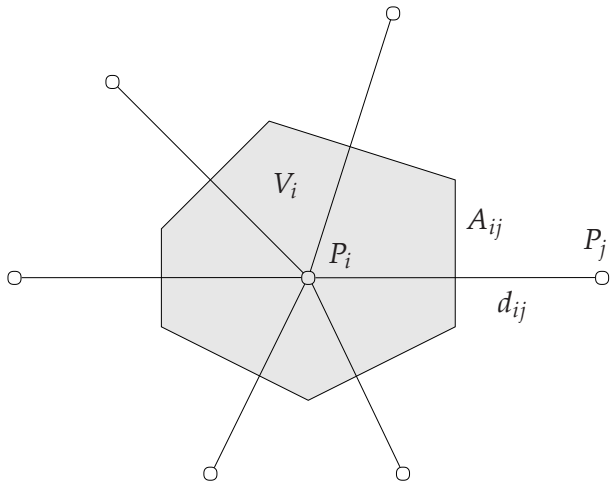

Figure 3.1: Control volume for the box integration method and its characteristics. V_i is the volume of the Voronoï box, P_i and P_j are two grid points, A_{ij} the area of the box segment between the boxes of point P_i and P_j, and d_{ij} is the distance between those two points.

Band-to-Band Tunneling describes the carrier generation in the high field region due to the field emission of valence electrons leaving back holes. Within MINIMOS-NT the trap assisted band-to-band tunneling process is replaced by the band-to-band tunneling process when the magnitude of the electric field increases.

Impact Ionization model can be used in the drift-diffusion (DD) and in the hydrodynamic (HD) transport models of MINIMOS-NT. In the DD model it is electric field and in the HD model carrier-temperature dependent.

3.3 Box Integration Method

The simulation domain has to be discretized in order to solve the system of partial differential equations. Within MINIMOS-NT the box integration method is used. The simulation domain is partitioned into sub domains, so called Voronoï boxes (see Section 2.2).

Figure 3.1 shows a control volume and the characteristic parameters used for the box integration method.

For the numerical solution of Poisson's equation (3.1) it needs to be discretized on a grid. The discretization can be done as follows:

$$\sum_j D_{ij} A_{ij} = \varrho_i V_i. \tag{3.21}$$

Here D_{ij} is an approximation for the projection of the dielectric flux density. It is calculated by

the finite difference approximation:

$$D_{ij} = -\varepsilon \frac{\psi_j - \psi_i}{d_{ij}}. \tag{3.22}$$

The value of ε is approximated by the average value of the permittivity of the two regions:

$$\varepsilon = \frac{\varepsilon_i + \varepsilon_j}{2}. \tag{3.23}$$

Since the vertices connecting two grid points must be chosen in such a way that a negative coupling between the grid points is avoided (A_{ij} can be negative when the grid is not properly formed), the box integration method demands a very sophisticated grid generator. Expecially in three spatial dimensions, the problem of generating a grid for arbitrary geometries is quite demanding.

3.4 PIF File Format

The original file format of MINIMOS-NT is the PIF (Profile Interchange Format) file format. It was initially proposed for the exchange of simulation data by Duvall [duva88] in 1988. It is used by MINIMOS-NT to load and save simulation data such as geometry information, material types, doping data, the simulation grid, and quantities like the current density or the electrostatic potential.

Within MINIMOS-NT the so called PIF-libraries are used to access PIF files. But as the PIF-libraries are optimized for two spatial dimensions, expanding the PIF-libraries to three-dimensional capabilities would require a complete redesign. Also, MINIMOS-NT would still be restricted to the grids, file-formats, and functionality supported by these libraries. Therefore, a new interface was developed which can use any kind of library for accessing simulation data. This could be for example the WAFER-STATE-SERVER which is described in Section 2.3.

Figure 3.2 depicts this new interface within MINIMOS-NT which is split up into two parts. One part of the interface is responsible for reading and writing the grid information which comprises the mesh points, the segment grids, the Voronoï information, and the boundary information. On the other hand there is the interface part responsible for reading and writing the quantity information like the impurity concentration, the electrostatic potential, or the hole concentration.

The PIF-libraries comprise the following modules:

- The GAS (Geometry Attribute Support) library implements functions for accessing quantity information within PIF files.
- The GRS (Grid Support) library provides functions for reading, writing, and manipulating grids. It is also capable of interpolating quantities to newly introduced grid points.

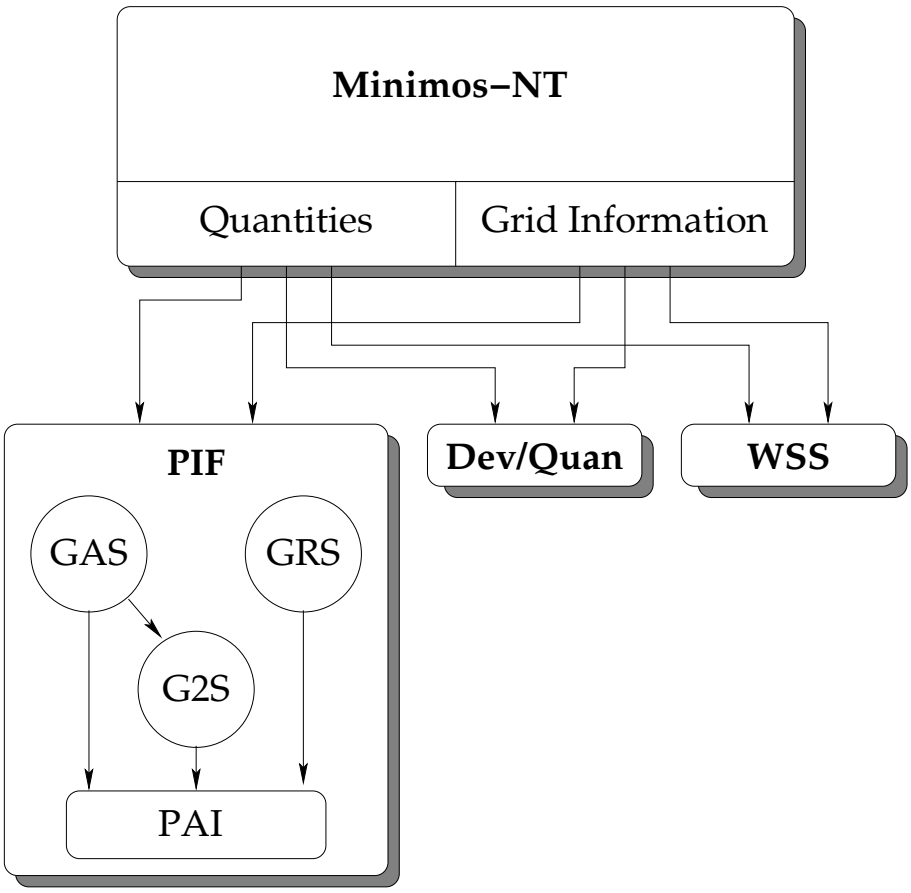

Figure 3.2: Libraries used within MINIMOS-NT to access the different file formats.

- The G2S (Geometry Two-dimensional Support) library provides a central geometry-handling facility. It provides functions for geometry checking and allows applications to query information from geometries, like which lines belong to what face boundary, or which faces or segments touch each other.
- The PAI (Pif Application Interface) offers a procedural interface for accessing PIF files.

3.5 DEV/QUAN File Format

To bypass the problems of the PIF file format with regard to three dimensional in- and output files the DEV/QUAN file format was created. The description of the wafer consists of two files. The `.dev` part stores the device description consisting of the grids, the segments, and the Voronoï information. The `.quan` part stores the attributes of the wafer. This is for example the impurity concentration of semiconductor segments.

The DEV/QUAN file format was only intended as temporary solution. It is not included in the release version of MINIMOS-NT. Instead, the file format has been replaced by an interface to the WAFER-STATE-SERVER within this thesis.

3.6 The Internal Data Structure

MINIMOS-NT needs a lot of information about the device it has to simulate. These are:

- The physical grid points. They are stored in a list. The segments themselves do not store the physical points but references to the list. Thus redundancy is avoided.
- An arbitrary number of segments. They contain:
  - References to the physical points that build up the topography of the segment.
  - The volumes of the Voronoï boxes belonging to the referenced points (see Section 2.2.1).
  - The connections between the segment points. These connections consist of the indices of the two points, the flux between the points – which is defined as the area between the two Voronoï boxes assigned to the points (see Section 2.2.1) divided by the distance –, and their distance.
- An arbitrary number of boundaries. They contain:
  - A reference to the two segments the boundary lies in between.
  - A list of the boundary points of the two segments.
  - The area between the two Voronoï boxes assigned to the points (see Section 2.2.1).
- The physical quantities that serve as input for certain models are generated by other models. These are, for example, the electrostatic potential or the dielectric permittivity.

This is the required information that has to be acquired from the WAFER-STATE-SERVER. The data is then stored within the internal data structure of MINIMOS-NT.

Chapter 4

Implementation

The primary target of this thesis was to enhance the device and circuit simulator MINIMOS-NT so that it can handle the file format `.wss`. This is the native file format of the WAFER-STATE-SERVER.

This enhancement is important because the Institute for Microelectronics is establishing a standard file format for its simulation programs. The process simulation tools and also some analytical device generation tools are capable of using the WAFER-STATE-SERVER to access in- and output files. Hence, it is a logical step to enhance the device simulator for seamless integration in the tool-chain of the Institute.

MINIMOS-NT needs a lot of information about the simulation domain. These are, for example, the coordinates of the grid points of the mesh, the quantities on these grid points – as the dopant concentration –, and the Voronoï information (see Section 2.2.1).

A part of the necessary information about the simulation domain is provided directly by the WAFER-STATE-SERVER. Other data, namely the Voronoï information, has to be generated from the grid. The internal data structure of MINIMOS-NT is then filled with this information and the simulation run is started.

Both, the WAFER-STATE-SERVER and MINIMOS-NT are – mostly – implemented using the object oriented programming language C++. This fact was of great help at the development process and avoided the need for a language wrapper.

4.1 Accessing the WAFER-STATE-SERVER

The following sections give a short introduction to using the WAFER-STATE-SERVER API and its methods. For a deeper insight into the WAFER-STATE-SERVER see [bind02].

```
/* instantiate a default configuration */
Config_h waferConfig(new Config());

/* instantiate a .wss reader */
Reader_h rd = instantiateReader(waferConfig, "wss", "mosfet.wss");

/* instantiate a Wafer object with the data stored in mosfet.wss */
Wafer_h  wafer = newWafer(rd, waferConfig);
```

Figure 4.1: Reading a `.wss` file and instantiating a new `Wafer` object.

4.1.1 Reading a `.wss` File

Care has been taken when implementing the interface of the WAFER-STATE-SERVER. The programmers focused their attention on straightforward instantiation and use of the WAFER-STATE-SERVER. This reduces the code for reading a `.wss` file to three lines.

Figure 4.1 depicts the code snipped to aquire an instance of a `Wafer` object with the data from a `.wss` file.

MINIMOS-NT does not use the described `newWafer` method of the WAFER-STATE-SERVER. Instead the `Wafer` is instantiated via the method `newWaferPointer`, which has the same functionality but returns a `Wafer*` instead of a `Wafer_h`. The reason for this is that MINIMOS-NT does not use handles but relies on pointers instead (see Section 4.2.3).

4.1.2 Reading from the `Wafer` Object

After the instantiation of the `Wafer` object we can use it to retrieve, modify, or write the wafer data.

Figure 4.2 shows how to use the `Wafer` object to retrieve the segments, the attributes, and the grid points of a device.

As seen in the code example retrieving information from the `Wafer` object leads to iterating through the data structure of the wafer. The `nextSegment` method is used for iterating the segments, `nextWafPoint` is used for iterating the `WaferPoints`, and so on.

The WAFER-STATE-SERVER has two different types of points. These are the `Point` and the `WaferPoint`. The first one denotes a physical point and stores the coordinates. But as one point can be part of two or more different segments – this is the case for example at the interface between two segments – the `WaferPoint` was introduced. It stores the attributes of the segment point and a handle to a `Point`. This prevents the WAFER-STATE-SERVER from storing one physical point multiple times and therefore avoids redundancy.

It is of course possible to retrieve the physical point from a wafer point with the `toPoint_h()` method. On the other hand it is not possible to cast a `Point` to a `WaferPoint`.

```
/* The segment object stores the grids and attributes of a segment. */
Segment_h seg;
unsigned int segIt = 0;

/* iterating over all segments */
while ((seg = wafer->nextSegment(segIt)))
{
  Attr_h attr;
  unsigned int attrIt = 0;

  cout << "Segment: " << seg->getName() << endl;

  /* iterate over all attributes of the segment */
  while ((attr = seg->nextAttribute(attrIt)))
  {
    cout << "Attribute: " << attr->getName() << endl;

    /* only quantities are distributed on a grid */
    if (attr->isQuantity())
    {
      Quantity_h quan = castToQuantity(attr);
      Locater_h  loc  = quan->getLocater();

      WafPoint_h wafPoint;
      unsigned int pointIt = 0;

      /* iterate over all points of the quantity */
      while ((wafPoint = loc->nextWafPoint(pointIt)))
      {
        Point_h point = wafPoint->toPoint_h();

        cout << "Point[x,y,z]: [" << point->x << ","
             << point->y << "," << point->z << "]" << endl;

        unsigned int valIt = 0;
        Val_h        val;

        /* iterate over all values stored on the point */
        while ((val = wafPoint->getVal(valIt)))
        {
          cout << "Value: " << val->str_rep() << endl;
        }
      }
    }
  }
}
```

Figure 4.2: Reading from the `Wafer` object and dumping information about the segments, attributes, and points to standard output.

```
unsigned int waferDimension = 3;

/* instantiate a .wss writer object */
Writer_h writer(instantiateWriter(waferDimension, "wss",
                                  "Testwafer", "testwafer.wss"));

/* dump the wafer data to the file testwafer.wss */
dumpWafer(wafer, writer);
```

Figure 4.3: Writing the `Wafer` data to a `.wss` file.

4.1.3 Writing to a `.wss` File

To save the data stored in a `Wafer` object to a file an appropriate `Writer` has to be instantiated. Then the function `dumpWafer` is called which dumps all data of the `Wafer` object to the file identified by the `Writer` object. Figure 4.3 shows an example of the writing process.

4.2 Implementation Issues

The WAFER-STATE-SERVER and MINIMOS-NT are projects of high complexity. This implies that both projects depend on a lot of libraries and other projects. As they were developed totally separated from each other in the past, some compatibility issues arose when they had to be combined.

4.2.1 Identical File Names

One of the first issues encountered was VMAKE's lack of namespaces for the source file structure of the projects it manages. Hence, two source files may not have the same file name. If, for example, two header files had the same name then VMAKE would not know which file to include at an `#include` statement.

Because of the complexity of the projects that had to be linked it was no surprise that some source files of the two projects had the same name. Table 4.1 shows the concerned files. On the MINIMOS-NT side the files were not directly in the MINIMOS-NT project but in the VLIBCXX project which it depends on. On the WAFER-STATE-SERVER side one of the concerned files was in the DYNLIB project.

Filename	MINIMOS-NT	WAFER-STATE-SERVER
`vector.hh`	`vlibcxx/src/vector.hh`	`wafer/geo-lib/vector.hh`
`typeid.hh`	`vlibcxx/src/typeid.hh`	`dynlib/typeid.hh`

Table 4.1: The same filenames in two different projects.

This issue was solved by simply renaming the source files. The author decided to rename the files that were part of the MINIMOS-NT project. The reason was simply that these files were not included in that many other projects as the files of the WAFER-STATE-SERVER.

To avoid this problem a naming convention could be established. Within MINIMOS-NT, for example, every file name starts with three letters according to the functionality of the file. As an example the prefix `mmq` stands for **M**ini**M**os-**Q**uantity and files with this prefix handle the quantity information within the project.

The problem is that such a naming convention must be introduced at the beginning of a software project. Changing the filenames afterwards is error-prone and the programmers have to learn the new scheme. Also it is rather unusual to use prefixes in today's software development.

After renaming the files and changing several `#include` statements both projects could reside in one VMAKE project. This does not imply that they can be linked together, which leads to the second issue.

4.2.2 Identical Class Names

The second issue was the lack of namespaces. As the variables of both projects were not wrapped by namespaces it occurred that two classes had the same name. So the compiler stopped with an error when the class was to be defined the second time.

This issue was solved by renaming two classes of the MINIMOS-NT project. Doing so lead to editing numerous files that were using these classes.

But as identical names for variables only cause problems when both definitions are included by one file, this issue is not totally solved yet. It can recur whenever another header file from one project is included in another project and they use identical names for global variables.

To avoid this issue all global variables and class definitions should be wrapped by namespaces. By doing so the global namespace remains clean. Of course the problem re-arises when two namespaces have the same name, so care has to be taken when choosing the namespace names.

4.2.3 Pointers and Handles

The WAFER-STATE-SERVER is a rather recent project. This is the reason why the developers could choose a very 'modern' approach for class design. So they abstained totally from the use of pointers and used a self defined-handle class instead. This approach is recommended in [stro00] and provides a simple but efficient way of garbage collection. Figure 4.4 shows the usage of both, a handle and a pointer.

A handle consists of two pointers. One points to the data to manage (the representation pointer) and the other to an integer variable with a reference counter. When the first handle to a data object is created the representation pointer is directed to the data object, memory for the reference counter is allocated, and the counter is set to 1. For every handle that is assigned to another handle, the left hand handle's reference counter is decremented by 1 and the object is

```
void handleTest()
{
  /* A handle to a class named someClass is defined and an object of
     someClass instantiated. */
  Handle<someClass> someClass_h(new someClass());

  /* A pointer to a newly allocated someClass object is defined. */
  someClass*        someClass_p = new someClass();

  /* The methods of someClass_h can be used the same way as the
     methods of someClass_p. */
  someClass_h->method1();
  someClass_h->method2();
  someClass_p->method1();
}
/* The someClass_h object is automatically destroyed when the handle
   gets out of scope. The someClass_p object is not destroyed and
   wastes memory that cannot be used outside the function
   handleTest(). */
```

Figure 4.4: Comparison between handles and pointer.

deleted if the reference counter is 0. Then the left hand handle copies the pointers of the right hand handle and the reference counter is incremented by 1. Figure 4.5 depicts the steps in an assignment process.

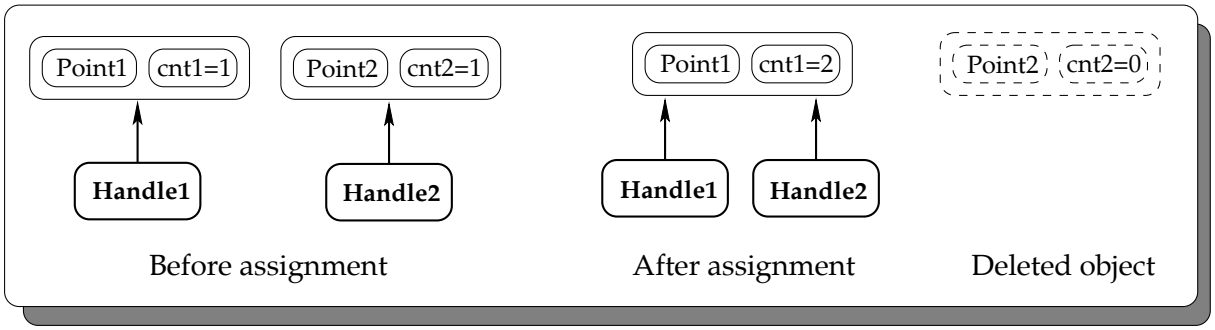

Figure 4.5: Assignment of handles. Before the assignment two independent handles with independent objects exist (the data object and the reference counter object). After the assignment the second handle points to the same objects as the first handle and its objects are destroyed because no handle is referencing them.

When a handle is deleted the reference counter is decremented by 1 and the object is deleted if the counter is 0.

With this method it is always guaranteed that an object is automatically deleted when no more handles are referencing to it.

To ensure full syntax compatibility to a real pointer the handle class overloads the operators *, ->, <, and >.

MINIMOS-NT was first reported by C. Fischer [fisc94] and T. Simlinger [siml96] in 1994. As the Standard Template Library (STL) did not exist at this time and the handle concept is based on the STL, the design of MINIMOS-NT was strongly based on pointers.

This divergence would not matter at all, but as the WAFER-STATE-SERVER did not offer the facility to return pointers to its objects MINIMOS-NT would be forced to use STL handles.

Therefore, a new method has been implemented in the WAFER-STATE-SERVER that generates a new Wafer object and returns a pointer to it instead of a handle. This approach of course partly violates the concept of handles and would make it possible to bypass the automatic garbage collection.

4.2.4 Project Dependencies

One issue was the size of the MINIMOS-NT project. The developers of MINIMOS-NT should not be forced to get and compile the whole WAFER-STATE-SERVER project and its dependencies when they do not need the `.wss` capabilities of MINIMOS-NT. Therefore all the changes made to the MINIMOS-NT project for WAFER-STATE-SERVER accessibility are wrapped by preprocessor commands and turned off per default.

Enabling the WAFER-STATE-SERVER capabilities in MINIMOS-NT

To enable the WAFER-STATE-SERVER capabilities in MINIMOS-NT the following steps are necessary:

1. In the C++ header file `vproject/mmnt/mms/mmsmain.hh` the preprocessor directive: `#define USE_MMNT_WAFER 0` must be set to `1`.
2. In the C++ header file `vproject/mmnt/mmg/mmgdelaunay.hh` the following include directives have to be restored:
   - `#include "wafer.hh"`
   - `#include "wafertools.hh"`
   - `#include "attr.hh"`
   - `#include "wafelem.hh"`
3. In the C++ source file `vproject/mmnt/mmq/mmqquan.cc` the following include directives have to be restored:
   - `#include "wafer.hh"`
   - `#include "reader.hh"`
4. In the C++ source file `vproject/mmnt/mms/mmsdevice.cc` the following include directives have to be restored:

- `#include "wafer.hh"`
- `#include "reader.hh"`

5. In the VMAKE configuration file `vproject/mmnt/vmfile.mk` the following project libraries have to be restored:
   - `IueCxxLibrary`
   - `Wafer-Attributes-Library`
   - `Wafer-AllGridder-Library`
   - `DynWrLibrary`
   - `DynRdLibrary`
   - `Sap-Library`
6. In the VMAKE configuration file `vproject/mmnt/vmake.prj` the following project dependencies have to be restored:
   - `IueCxxLibraryProject`
   - `WaferStateServer`
7. The whole WAFER-STATE-SERVER project and its dependencies have to be loaded. These are namely:
   - `vproject/wafer`
   - `vproject/iuecxx`
   - `vproject/ipdcxx`
   - `vproject/antlr`
   - `vproject/dynlib`
   - `vproject/serlib`
   - `vproject/pthreadcxx`
   - `vproject/deLink`
8. The JAVA software developers package has to be installed.

The reason why all the `#include` commands had to be commented out is that VMAKE is not able to ignore code that is disabled by preprocessor commands.

After building the `wafer` project the `mmnt` project can be compiled and linked.

Chapter 5

Practical Aspects and Examples

5.1 Using `.wss` Files as Input for MINIMOS-NT

For reading a `.wss` file as device and quantity description instead of the `.pif` format a new command line argument (`-wss`) was introduced in MINIMOS-NT. The input deck can stay untouched but instead of an input file with the extension `.pif` an input file with the extension `.wss` has to exist. The output is written to the file specified in the input deck with the extension `.wss`.

For visualization of a device and the calculated quantities the tool SMARTV can be used. It was developed at the Institute for Microelectronics [zohl03] and can read `.wss` files natively. Most of the pictures of devices and their quantities within this work have been made with SMARTV. A short introduction to this tool is given in Section 2.6.3.

5.2 The Gridding Issue

One of the most important aspects in the area of device simulation is the discretization of the simulation domain. As described in Section 2.2 this is not a trivial task at all. Especially when it comes to three-dimensional mesh generation and when the Delaunay criterion has to be fulfilled – as it has to be for simulations with MINIMOS-NT – this task is not fully solved yet.

The gridder DELINK is used at the Institute for Microelectronics within the WAFER-STATE-SERVER and all tools that are based on it. Unfortunately it turned out to not being perfectly suited for all areas of application.

LAYGRID has been used for generating the three-dimensional device structures. Figure 5.1 depicts a grid structure which is part of a MOSFET that was generated with LAYGRID. The tetrahedrons – or their representing triangles on the surface – are well ordered with no big gaps or anomalies. On the top of the structure the grid is refined because this is the electrically interesting part where the channel is built up.

LAYGRID has the big advantage that it does not need a sophisticated three-dimensional grid

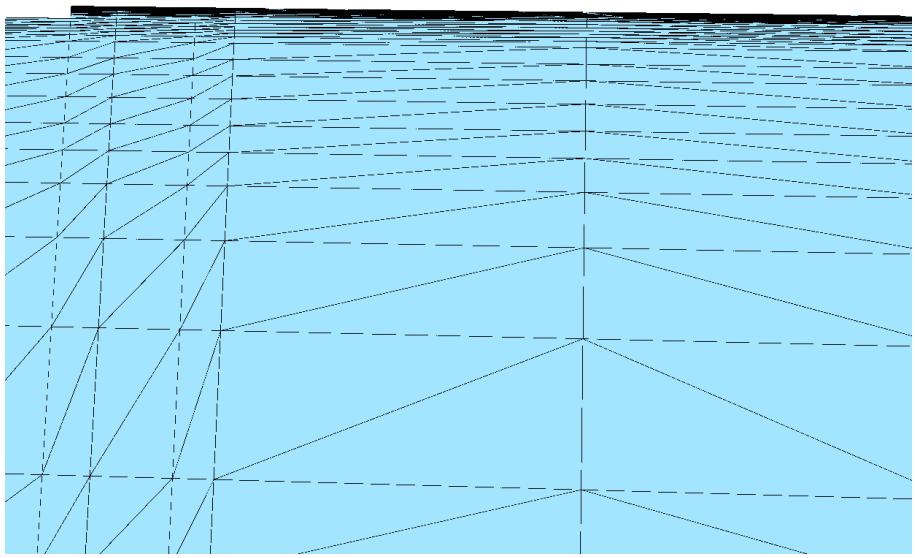

Figure 5.1: Example of a 'nice' mesh.

generator. Instead two-dimensional outlines of the structure are generated which are then triangulated with a two-dimensional triangulation library as for example TRIANGLE [tria]. This structure is then extended to the third dimension. This method forms so called 'Manhattan' structures.

After the topology of the device is formed the substrate of the transistor has to be doped. This task is performed with the tool ADDANAIPD. It can be used to put any distributed quantity on a device structure. The quantity concentrations are described via analytical functions with the coordinates as parameters and the quantity concentration as return value.

Figure 5.2 depicts the grid structure that was generated with LAYGRID. The device was doped with Boron via ADDANAIPD which used DELINK as gridder. The impurity concentration is visualized. Although the doping meant to be a clean, horizontal bar the malformed grid made it chaotic. Also on the uppermost part of the structure where the grid should be refined to very close horizontal lines the gridder broke this concept by using very large tetrahedrons – which appear as triangles in the figure.

These malformed grids not only cause extreme inaccuracies but also impose big problems on MINIMOS-NT whose numeric solver has difficulties to converge with these meshes.

To oppose this problem a preprocessor for DELINK is being developed at the Institute for Microelectronics. It solves the Poisson equation on the device and feeds the information about the iso-potential contours to DELINK. By this means a for many applications good mesh can be generated.

Most examples simulated within this work are not meshed with DELINK to circumvent the

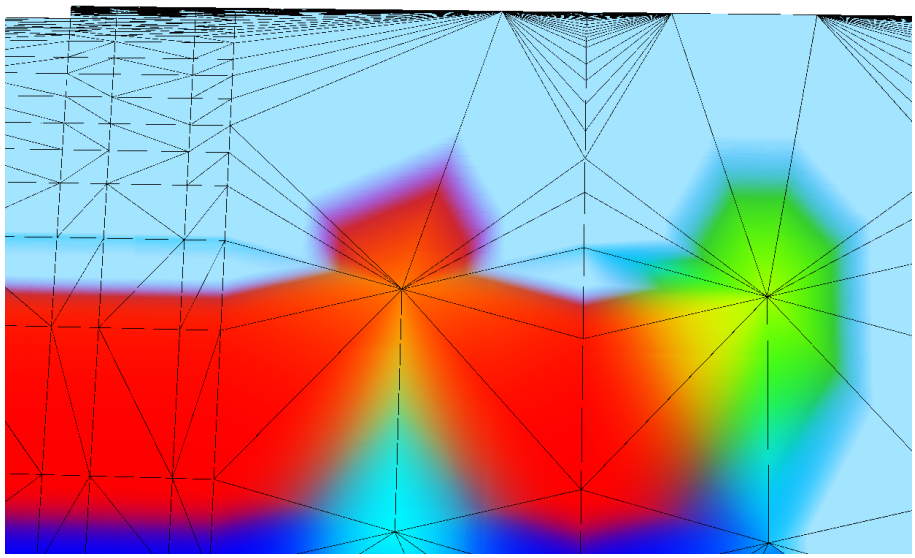

Figure 5.2: Example of a 'bad' mesh.

problems arising from inadequate grids. Therefore the grids are generated by LAYGRID and then not altered anymore. To achieve this new functionality had to be implemented in the tool ADDANAIPD.

5.3 Examples

In this section some devices and their simulation results are presented.

5.3.1 Si-Block

The first device under test is a silicon block as depicted in Figure 5.3. It was chosen because it has a very simple structure and can proof the basic interoperability of MINIMOS-NT and the WAFER-STATE-SERVER.

The block is made of pure silicon and has two aluminium contacts. As the silicon is not doped it acts as a pretty good insulator.

The two aluminium contacts were set to 0 V and 10 V. The computed electrostatic potential in the block must be evenly distributed across the whole length of the silicon segment.

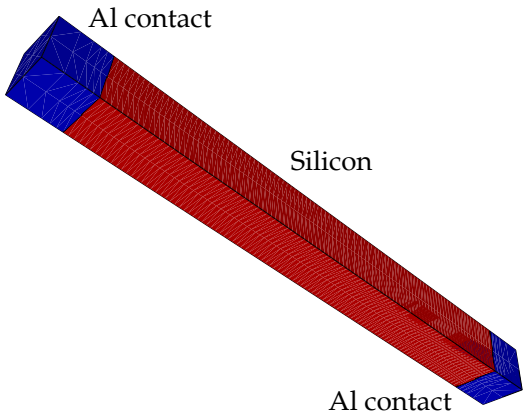

Figure 5.3: Structure of the silicon block.

Results

The arrangement was simulated with MINIMOS-NT in three spatial dimensions. Figure 5.4 shows the distribution of the electrostatic potential within the silicon block. As expected the potential is evenly distributed from 0 to 10 Volt across the material.

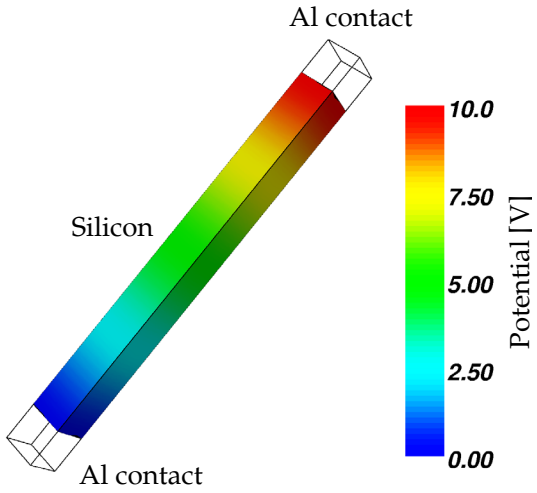

Figure 5.4: Distribution of the electrostatic potential within a silicon block. The aluminium contacts are set to 10 respectively 0 Volt.

This result shows that the wafer data was passed from and to the WAFER-STATE-SERVER correctly and that the quantity information (in this case the electrostatic potential and the electrical field) was written correctly.

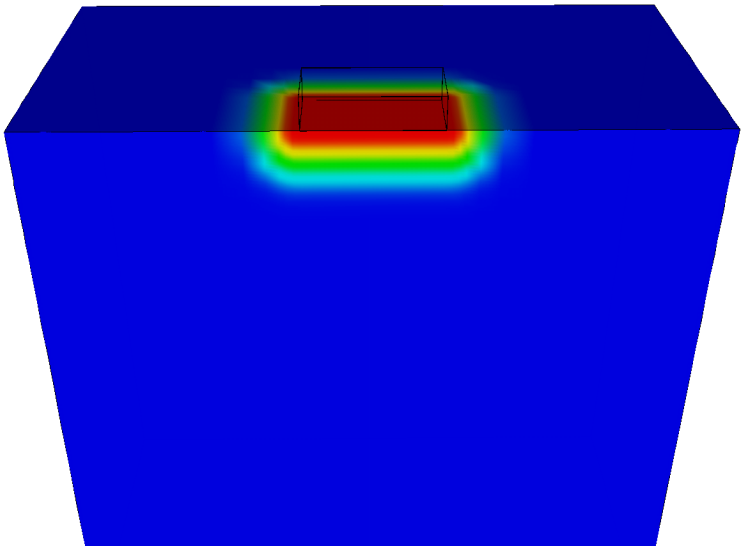

Figure 5.5: Dopant profile of a Silicon pn-diode.

5.3.2 Silicon pn-Diode

The silicon pn-diode is the next step in increasing the complexity of simulation data.

To simulate the behavior of a pn-diode a Silicon block was created with a constant Boron impurity concentration of $2 \times 10^{17}\,\mathrm{cm}^{-3}$. This block was then doped with a Phosphorus concentration of $5 \times 10^{18}\,\mathrm{cm}^{-3}$ on the top side and contacted with Aluminium. The device Structure can be seen in Figure 5.5.

In the simulation run with MINIMOS-NT 0 V were applied to the cathode (the contact on the lower side of the block) and the voltage on the anode was stepped from 0 V to 2 V. This way the diods output characteristics could be generated. It is depicted in Figure 5.6.

The figures 5.7 and 5.8 depict the electron and hole currents, respectively. For these results an anode voltage of 1 V was applied and the cathode was set to 0 V.

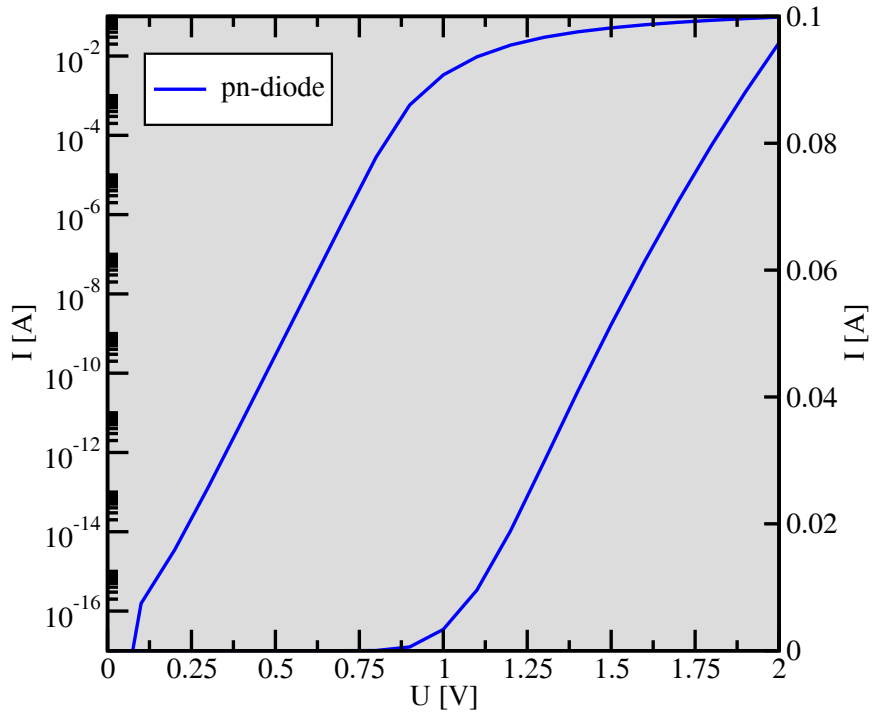

Figure 5.6: Output characteristics of a silicon pn-diode in logarithmic (left) and linear scale.

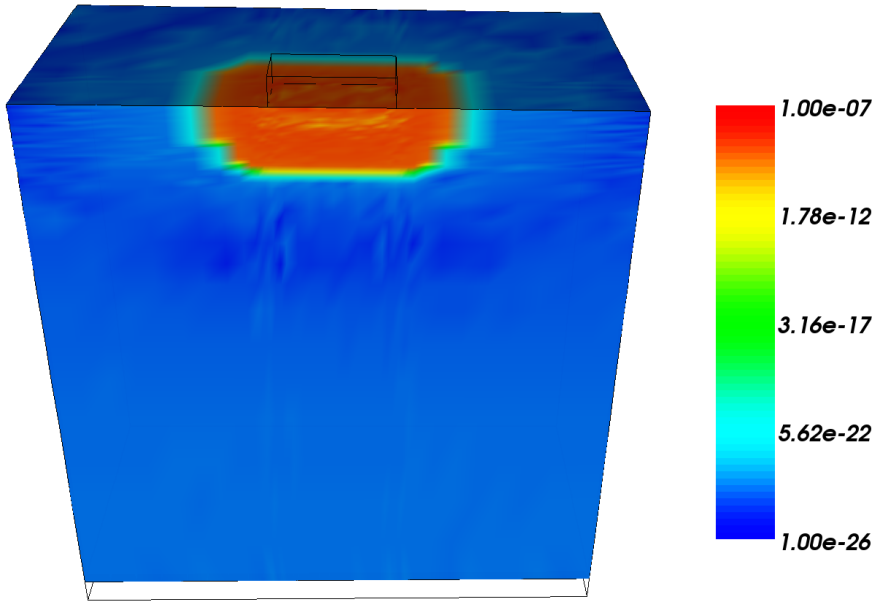

Figure 5.7: Magnitude of the electron current density within the diode in logarithmic scale.

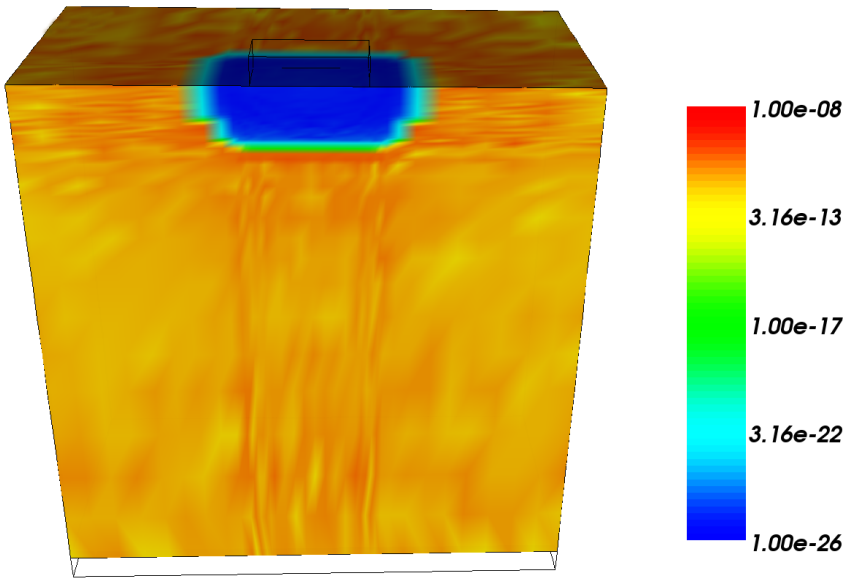

Figure 5.8: Magnitude of the hole current density within the diode in logarithmic scale.

5.3.3 MOSFET Transistor

The Metal Oxide Semiconductor Field Effect Transistor (MOSFET) consists of a source and a drain, two highly conducting n-type semiconductor regions in the p-type substrate. A polycrystalline gate covers the region between source and drain, but it is separated from the semiconductor by the gate oxide. On the lower side of the substrate is the bulk contact. The charge on the gate of the device controls the movement of charge between the source and drain through the channel under the gate.

Figure 5.9 shows this structure as it was simulated. To reduce unnecessary simulation data the device was cut alongside the channel. This is possible as the device is perfectly symmetric. All calculated currents have then to be multiplied by two.

In this example an n-channel MOSFET was chosen. Therefore under the source and the drain contacts wells with Arsenic (n-type) are formed as seen in Figure 5.9. The concentration goes up to $1.6 \times 10^{20}\,\mathrm{cm}^{-3}$. Underneath the gate contact a small film with Boron (p-type) was implanted with a concentration of $1.4 \times 10^{17}\,\mathrm{cm}^{-3}$. The whole substrate has a minimum Boron concentration of $2 \times 10^{16}\,\mathrm{V}$.

The dimensions of the device are:

- width: 5 µm
- height: 1.6 µm
- depth: 5 µm

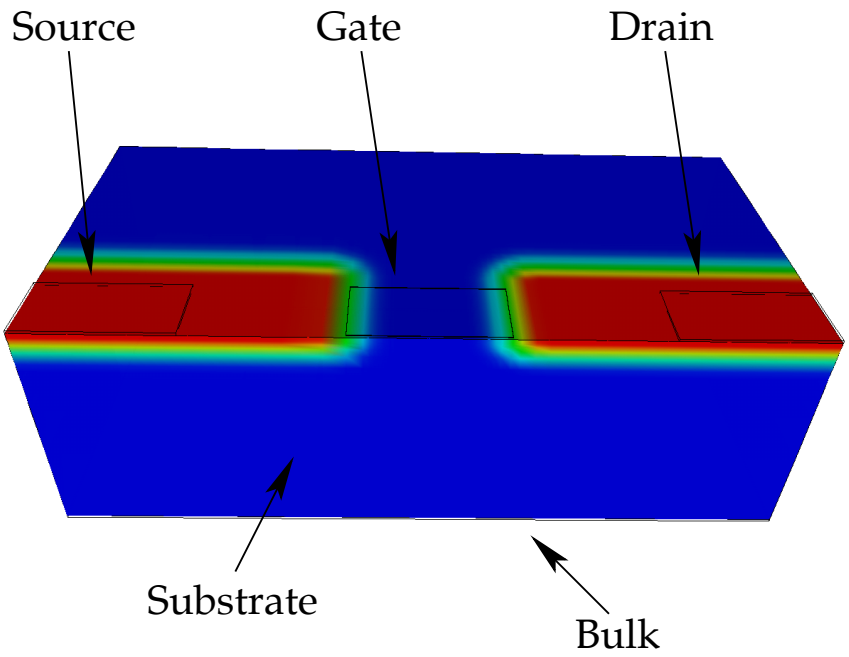

Figure 5.9: Topology of a simple MOSFET and the concentration of the Arsenic doping.

- gate length: 1 µm
- gate width: 1 µm
- gate-oxide thickness: 15 nm

Gridding Issues

The MOSFET turned out to be very demanding concerning the grid generation. In contrast to the silicon block and the pn-diode the mesh of the transistor has to be very carefully refined in important areas.

Using LAYGRID with the refinement method described in Section 2.6.1 a perfectly suitable grid could be generated. But as LAYGRID is only capable of generating the topology of the simulation domain and cannot be used to add dopant concentrations to the device ADDANAIPD has to be used for doping. Unfortunately, ADDANAIPD, at this stage, used DELINK as gridding tool. This combination generated grids like seen in Figure 5.2. With grids of this quality no representative simulation run could be performed.

Alternative Gridding

Another way of generating the mesh for the MOSFET was achieved by using a newly implemented functionality of the ADDANAIPD tool which allows the use of the grid generated by LAYGRID for the doping process. LAYGRID turned out to be powerful enough to generate a suitable grid with the 'dummy layer' and 'dummy face' mechanism described in Section 2.6.1.

Also the tool CREATEORTHO was very useful for the generation of three-dimensional devices. CREATEORTHO is written in PYTHON and generates orthogonal grid structures.

Simulation

The first simulation run was made for calculating the output characteristics of the MOSFET. Therefore the following voltages were applied to the electric contacts of the device:

- Source: 0 V
- Drain: Stepped from 0 V to 1.2 V in 0.1 V steps
- Gate: Stepped from 0.7 V to 1.2 V in 0.1 V steps
- Bulk: 0 V

Figure 5.10 depicts the output characteristics of the mosfet. The following figures 5.11, 5.12, 5.13, and 5.14 depict the electrostatic potential, the electron concentration, the electric field, and the electron current with the following input voltages:

- Source: 0 V
- Drain: 1.2 V
- Gate: 1.2 V
- Bulk: 0 V

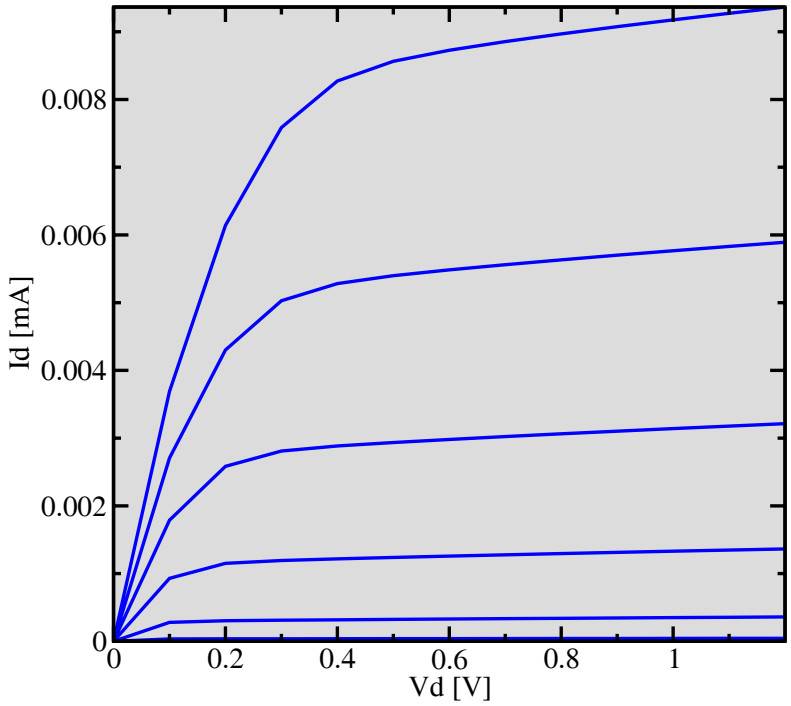

Figure 5.10: Output characteristics of the MOSFET. The gate voltages range from 0.7 V to 1.2 V.

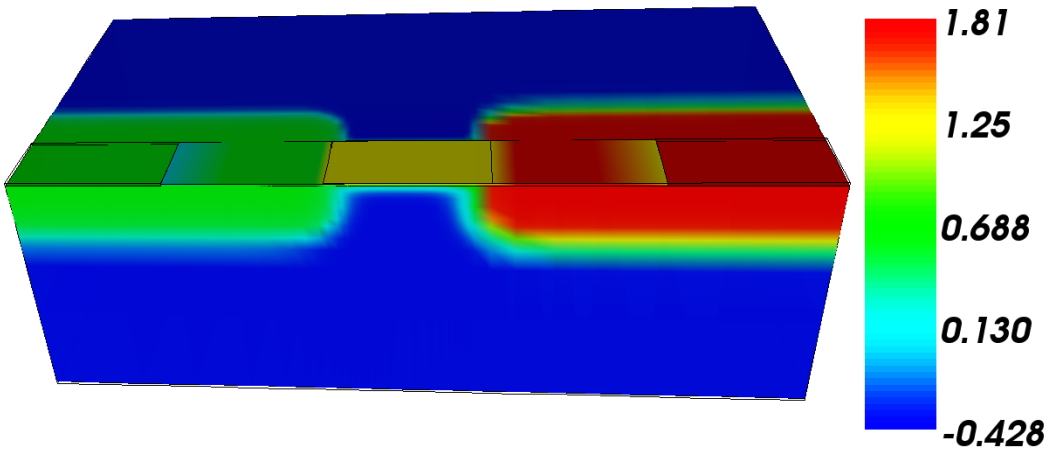

Figure 5.11: Distribution of the electrostatic potential across the MOSFET.

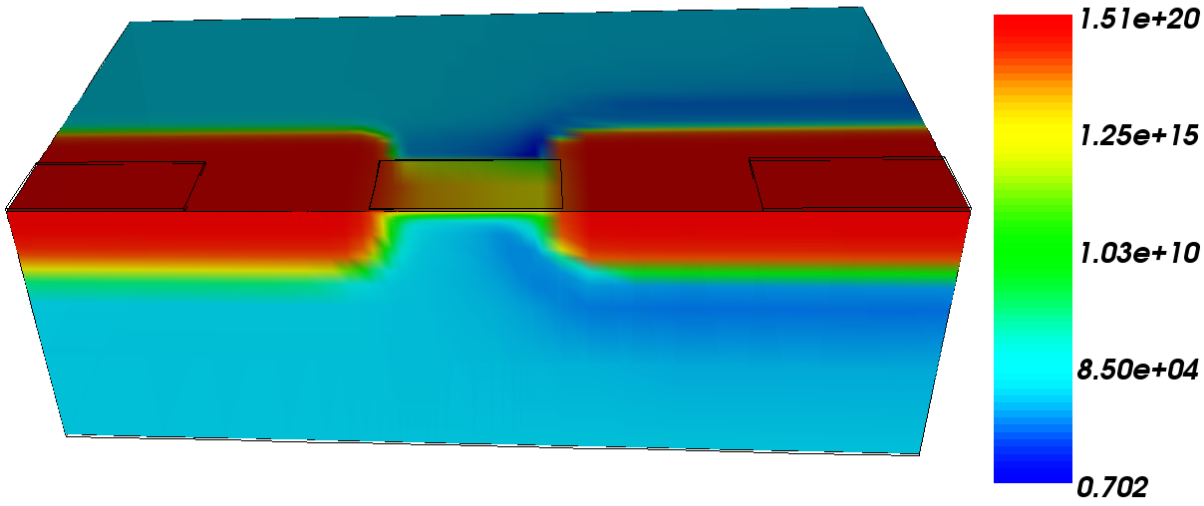

Figure 5.12: Electron concentration in the MOSFET in logarithmic scale.

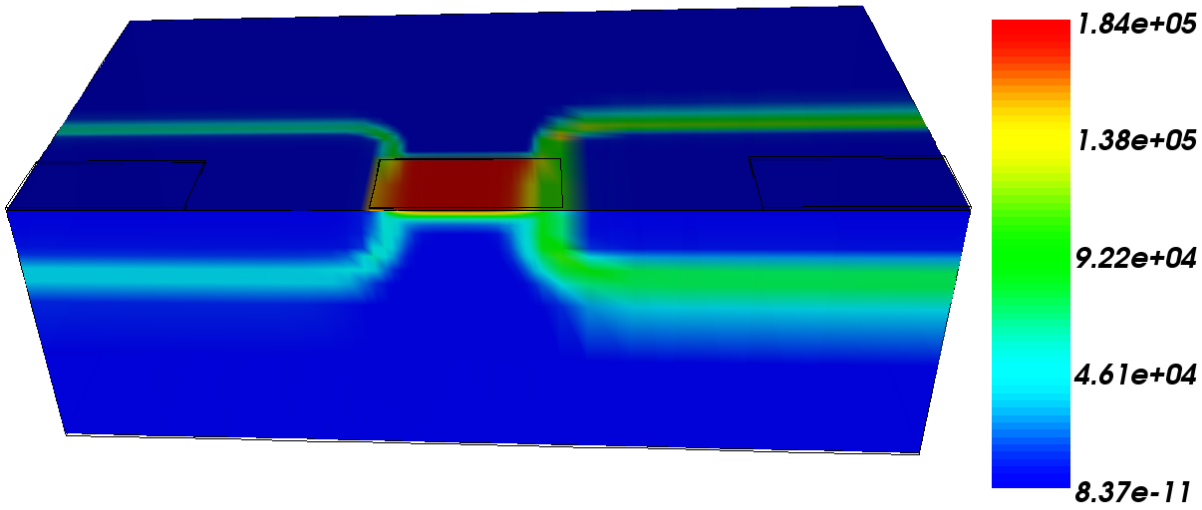

Figure 5.13: Magnitude of the electric field in the MOSFET.

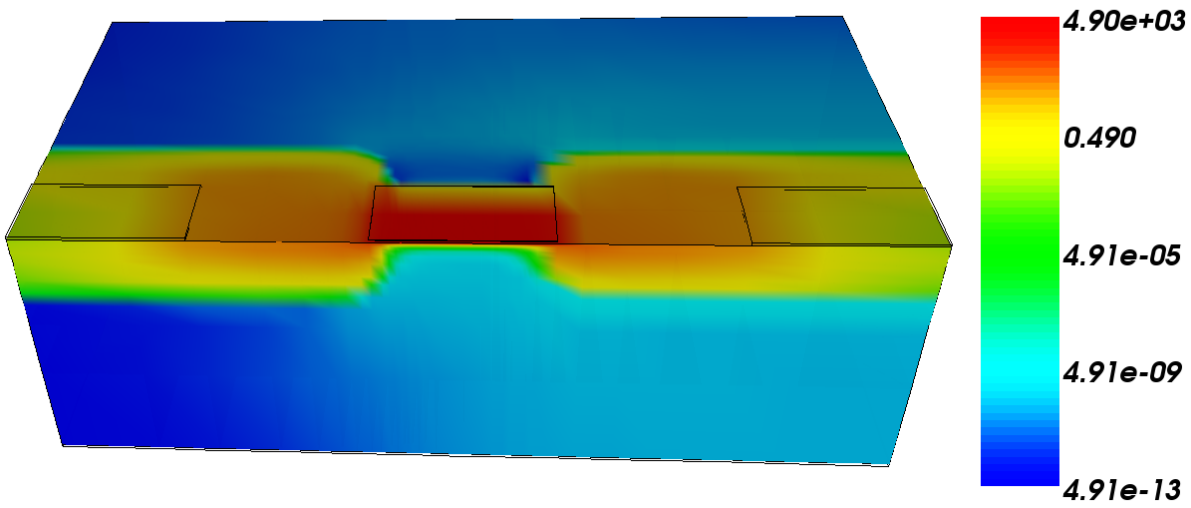

Figure 5.14: Magnitude of the electron current density in the MOSFET. The scale is logarithmic.

5.3.4 Carbon Nanotube Field Effect Transistor

Shrinking of silicon CMOS technology has its limits. Problems arise at feature sizes smaller than 50 nm due to physical aspects. Manufacturing problems related to lithography or interconnects are also increasing. Therefore much attention is paid to new technologies like nanotechnology.

Carbon nanotubes are very promising candidates for future nanoelectronic applications. A nanotube can be seen as a single sheet of graphite that has been rolled up into a tube. Depending on the direction in which the sheet was rolled up, the tubes are either metallic with high electrical conductivity, or semiconductors with relatively large band gaps. They gain their promising electrical properties as a consequence of the electric band structure which depends on the exact position of the carbon atoms forming the tube.

In contrast to single-wall nanotubes, which only consist of one tube of carbon, multi-wall nanotubes are built from several concentric tubes, nested inside each other like Russian dolls.

Semiconductor nanotubes can be used as active elements in field-effect transistor (FET) designs. Figure 5.15 depicts the device structure of a lateral and an axial Carbon Nanotube Field Effect Transistor (CNT-FET) [unge03].

The lateral CNT-FET resembles conventional MOSFET structures where the silicon channel is replaced by a single-wall or multi-wall carbon nanotube with semiconducting properties. It is connecting the source and drain contacts.

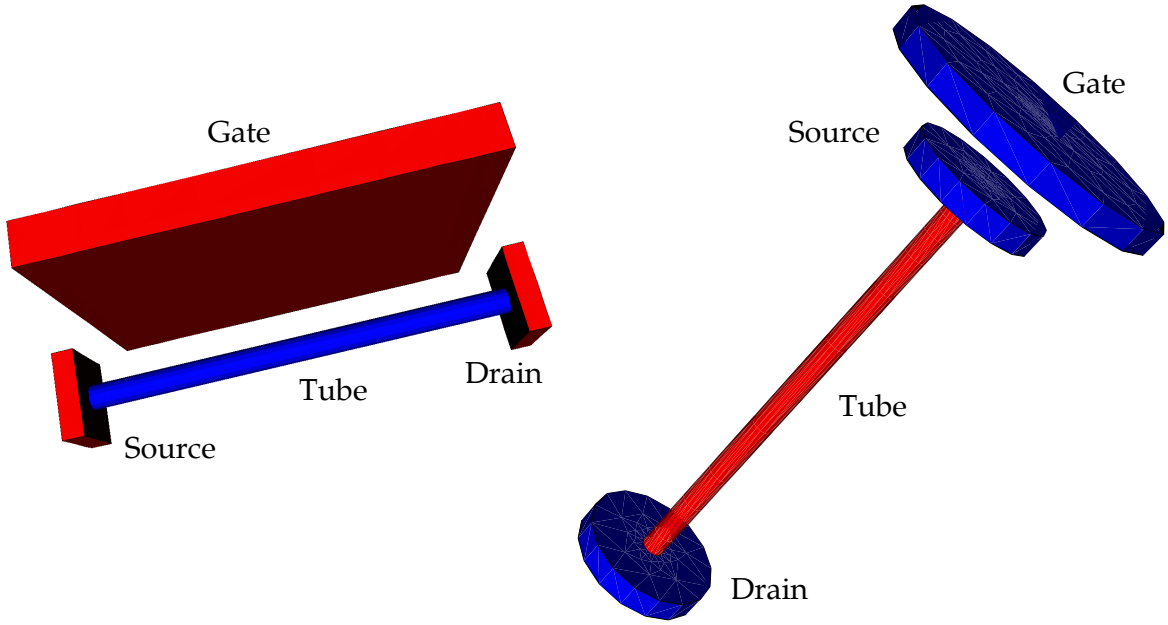

Figure 5.15: The structure of a lateral (left) and axial carbon nanotube field effect transistor.

The gate contact of the axial CNT-FET lies above the source contact. They are separated by a thin gate oxide. At this positioning the capacitive coupling between the gate and the tube is much weaker than at the lateral positioning of the gate. The advantage is the much smaller footprint of the structure.

Lateral CNT-FETs show good performance with high I_{ON}/I_{OFF} ratios. But as the manufacturability challenges are still significant – especially for large scale integration – transistors with axially aligned carbon nanotubes are considered. They show worse device characteristics but are more suitable for large scale integration.

Simulation

As the carbon nanotube in a CNT-FET has semiconducting attributes, the tube could be modeled with silicon for the simulation. In the simulation run with MINIMOS-NT the gates of both CNT-FETs were set to $-10\,\mathrm{V}$. The source and drain contacts were kept at $0\,\mathrm{V}$.

Figures 5.17 and 5.16 show the distribution of the electrostatic potential at the carbon nanotube in the axial CNT-FET and the lateral CNT-FET. What can be seen from the figures is the much stronger influence of the gate field on the tube in the lateral CNT-FET. The maximum of the absolute value of the electrostatic potential is spread over the whole tube. This leads to better device characteristics than can be achieved in lateral devices.

In the axial CNT-FET the peak of the electrostatic potential is smaller and concentrated in a limited area of the tube. The rest of the tube is not as strongly coupled to the gate field.

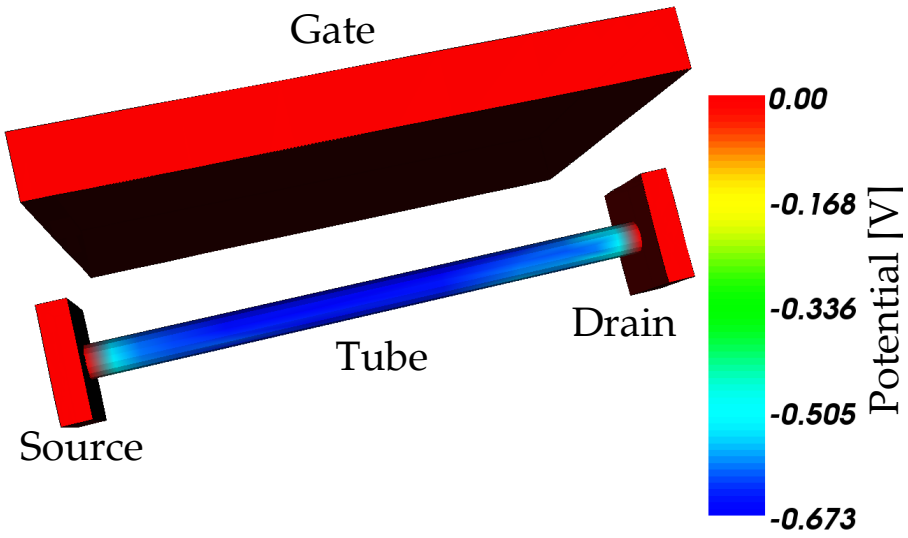

Figure 5.16: A lateral carbon nanotube field effect transistor simulated with MINIMOS-NT. Source and drain are set to 0 V and the gate is set to −10 V. The electrostatic potential in the carbon tube is visualized.

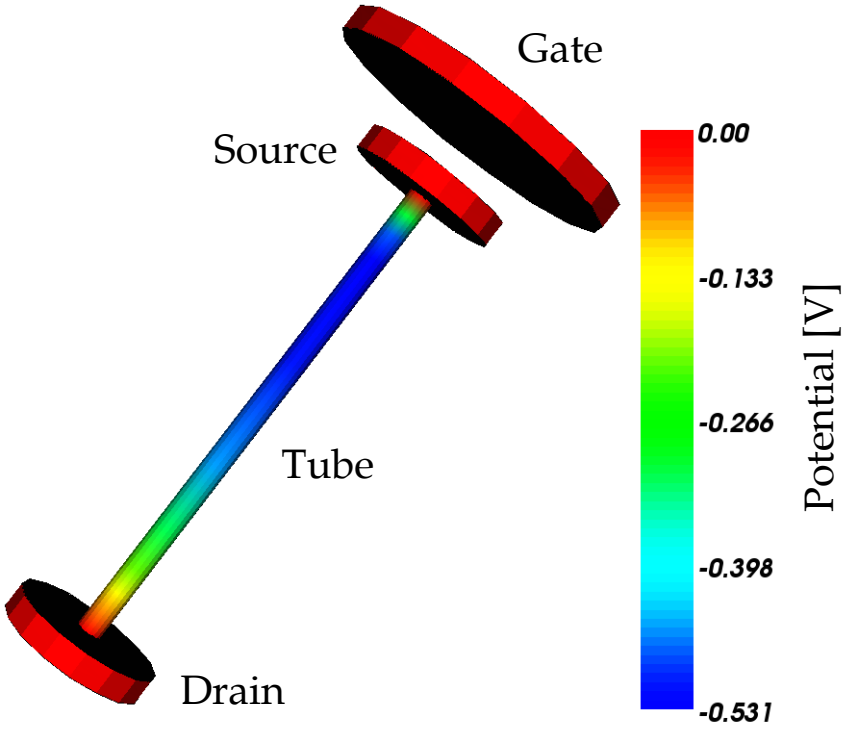

Figure 5.17: An axial carbon nanotube field effect transistor simulated with MINIMOS-NT. Source and drain are set to 0 V and the gate is set to −10 V. The electrostatic potential in the carbon tube is visualized.

Chapter 6

Summary and Outlook

The device and circuit simulator MINIMOS-NT has been coupled to the WAFER-STATE-SERVER. Therefore, MINIMOS-NT is now capable of reading and writing the `.wss` file format. It is possible to integrate MINIMOS-NT seamlessly into the process and device simulation flow of the simulation tools developed at the Institute for Microelectronics. The fabrication process of a semiconductor device can be simulated by the process group at the Institute for Microelectronics and the resulting device can then be fed into MINIMOS-NT to simulate the electrical behavior.

This work presented the design tools and data models that were necessary for the implementation of the new functionality into MINIMOS-NT. Also the issues arising from combining two major software projects and their solutions were shown. Special care has been taken to show how devices can be constructed in three-dimensions with the software tools developed at the Institute for Microelectronics and how optimized simulation grids can be constructed.

The applicability of the new features has been demonstrated with several examples where typical device structures have been created and simulated. A simple pn-diode, a MOSFET, but also advanced structures like axial and lateral carbon nanotubes have been simulated in three spatial dimensions. As visualization tools exist for the `.wss` file format the results could be visualized easily.

The consistent miniaturization and the increasing demands on semiconductor devices makes the development of increasingly complex device structures necessary. As these improvements let three-dimensional effects become stronger and stronger it is important to simulate such devices in three spatial dimensions. But as the complexity the simulator has to cope with explodes with this expansion, the computational effort compared to two-dimensional simulations is enormous. Therefore, methods to increase the simulation speed have to be developed. This involves very sophisticated gridding tools that are capable of refining the grid in sensitive areas of the simulation domain and keep the grid coarse in other regions.

Especially the grid generation in three spatial dimensions turned out to be a very demanding task. The WAFER-STATE-SERVER and most tools which generate three-dimensional structures or add distributed quantities to devices use DELINK as gridder. Although DELINK is a very sophisticated gridding tool, it needs a good initial set of points as basis. At the Institute for

Microelectronics a new tool is developed that acts as such a preprocessor for DELINK. The quality of grids for device simulations can be greatly enhanced with this new tool.

The current release 2.0 of MINIMOS-NT is restricted to two spatial dimensions. Based on [klim02] and this thesis the release of a new version of MINIMOS-NT can be prepared. The new release will be capable of simulating three dimensional device structures and read and write an increasing number of different file formats from different vendors.

Bibliography

[bind02] T. Binder, "Rigorous Integration of Semiconductor Process and Device Simulators"
Dissertation, Technische Universität Wien, 2002
http://www.iue.tuwien.ac.at/phd/binder

[ise] ISE Integrated Systems Engineering
http://www.ise.ch/

[duva88] S.G. Duvall, "An Interchange Format for Process and Device Simulation"
IEEE Trans.Computer-Aided Design, CAD-7(7):741-754, 1988.

[fisc94] C. Fischer, "Bauelementsimulation in einer computergestützten Entwurfsumgebung"
Dissertation, Technische Universität Wien, 1994
http://www.iue.tuwien.ac.at/phd/fischer

[flei99] P. Fleischmann, "Mesh Generation for Technology CAD in Three Dimensions"
Dissertation, Technische Universität Wien, 1999
http://www.iue.tuwien.ac.at/phd/fleischmann

[hdf] Hierarchical Data Format (HDF)
http://hdf.ncsa.uiuc.edu/

[klim02] R. Klima, "Three-Dimensional Device Simulation with MINIMOS-NT"
Dissertation, Technische Universität Wien, 2002
http://www.iue.tuwien.ac.at/phd/klima

[mmnt] MINIMOS-NT Documentation
Technische Universität Wien, 2003
http://www.iue.tuwien.ac.at/software/minimos-NT/

[sap] Smart Analysis Programs (SAP)
http://www.iue.tuwien.ac.at/software/sap/

[selb84] S. Selberherr, "Analysis and Simulation of Semiconductor Devices"
Springer, 1984, ISBN 3-211-81800-6

[siml96] T. Simlinger, "Simulation von Heterostruktur-Feldeffekttransistoren"
Dissertation, Technische Universität Wien, 1996
`http://www.iue.tuwien.ac.at/phd/simlinger`

[stro00] B. Stroustrup, "The C++ Programming Language Special Edition"
Addison-Wesley, 2000, ISBN 0-201-70073-5

[tria] Triangle Library
`http://www-2.cs.cmu.edu/ quake/triangle.html`

[trol] Trolltech Inc.
`http://www.trolltech.de/`

[tupp96] W. Tuppa, "VMAKE A CASE-Oriented Configuration Management Utility"
Dissertation, Technische Universität Wien, 1996
`http://www.iue.tuwien.ac.at/phd/tuppa`

[unge03] E. Ungersböck, "Simulation of Carrier Transport in Carbon Nanotube Field Effect Transistors"
European Solid-State Device Research Conference, Estoril, Portugal, September 16–18, 2003

[vtk] The Visualization Toolkit (VTK)
`http://www.vtk.org/`

[zohl03] M. Zohlhuber, "Visualisierung von Simulationsdaten"
Diplomarbeit, Technische Universität Wien, 2003

Curriculum Vitae

December 2002
Started the Diplomarbeit at the Institute for Microelectronics.

1994 - 2001
Summer jobs at: Landhotel Bierpeter, AVL-List Graz, EDS Graz, Austriamicrosystems Unterpremstätten.

October 1997
Enrolled in Electrical Engineering at Vienna University of Technology.

1996 - 1997
Military service.

June 1996
Graduation at Bundesrealgymnasium Keplerstraße Graz (with honors).

April 1995
Wifi Diploma: Certified Netware Administrator.

July 1994
Wifi Diploma: Software-Entwickler.

July 1993
Wifi Diploma: System-Betreuer.

January 1993
Wifi Diploma: PC-Administrator.

July 1988
Graduation at Private Volksschule Projektschule Graz.

October 22nd, 1977
Born in Graz, Austria.