VISTA Status Report

T. Grasser, A. Hössinger, H. Kirchauer, M. Knaipp, R. Martins, R. Plasun,
M. Rottinger, G. Schrom, S. Selberherr

Institute for Microelectronics
Technical University Vienna
Gusshausstrasse 27-29
A-1040 Vienna, Austria

Contents

1 Lithography Simulation in VISTA **1**

1.1 Aerial Image Simulation . . . 1

1.1.1 Fundamental Results of Fourier Optics . . . 1

1.1.2 Lens Aberrations . . . 2

1.1.3 Advanced Illumination Aperture . . . 4

1.1.4 Numerical Implementation . . . 6

1.2 Handling layouts for lithography simulation . . . 9

1.3 Examples . . . 9

2 Simulation of Ion Implantation with MCIMPL **12**

2.1 Introduction . . . 12

2.2 Models . . . 12

2.3 Simulation Input . . . 14

2.4 Simulation Flow . . . 14

2.5 Examples . . . 16

3 The MINIMOS-NT Input Deck **19**

3.1 Introduction . . . 19

3.2 Features of the Input Deck Control Language . . . 19

3.2.1 Simulator Specifics . . . 21

3.3 The `default.ipd` File . . . 22

3.3.1 The `Extern` Section . . . 22

3.3.2 The `Quantities` Section . . . 22

3.3.3 The `Models` Section . . . 23

3.3.4 The `Iterate` Section . . . 24

3.4 Example . . . 25

3.4.1 Description of the Input Deck Used for Simulation of a HBT . . . 25

3.4.2 The Input Deck for the Simulation of a HBT . . . 27

3.5 Mixed-Mode Simulation with MINIMOS-NT . . . 28

3.5.1 Introduction 28

3.5.2 Using Mixed-Mode 28

4 Optimization of Analytical Doping Profiles 32

4.1 Introduction 32

4.2 Vienna Integrated System for TCAD Applications 32

4.3 Optimization 32

4.4 Application 32

1 Lithography Simulation in VISTA

The lithography tool incorporated in VISTA consists of two parts, namely an *aerial image simulator* and a *layout editor*. The aerial image tool calculates the two-dimensional light intensity incident on top of the wafer. It is capable of simulating projection printing with arbitrary shaped illumination apertures, includes models for lens aberrations, and handles layouts up to a size of 30 μm $\times$ 30 μm with a run-time less than half an hour. The layout editor is based on the PIF-editor PED. It provides functionality to convert mask files stored either in the GDSII- or CIF-file format, supports phase-shifting masks, and automatically checks printability of the mask pattern by analyzing the aerial image. In the following three sections we first describe the aerial image simulator, then we summarize the most relevant features of the layout editor, and finally demonstrate the capability of the lithography package by showing some simulation examples.

1.1 Aerial Image Simulation

The aerial image tool incorporated in VISTA provides a scalar as well as a vector model to calculate the image of optical projection printing systems. The technique of image forming with a lens employed in projection printing is similar to that used in microscopes. Hence, the involved optical phenomena including diffraction effects have been long-studied and are well-understood [1, Chapters VIII–XI]. Especially, the theory of *Fourier optics*—either in its scalar or vector form—provides a powerful physical framework to describe projection printing. We thus start with a brief sketch of the fundamental results of Fourier optics and discuss then the lens aberrations caused by the imperfection of the optical systems. Advanced apertures are also discussed and finally some aspects on the numerical implementation of the aerial image module are given.

1.1.1 Fundamental Results of Fourier Optics

The light propagation through the optical system and the light transmission through the photomask can be studied by ray-tracing of homogeneous plane waves. Thereby a linear relation between the object and the image is established. Under the assumption of the commonly used Köhler illumination [1, pp. 524–526] and the restriction to periodic mask patterns the resulting formulas can efficiently be evaluated with *Fast Fourier Transform (FFT)* algorithms.

Köhler illumination means that a point source (pq) provides a plane wave illumination of the masks. If the location of the point source is chosen in a way such that the wavevector equals $\mathbf{k}^{pq} = \left[2\pi p/a,\ 2\pi q/b,\ \sqrt{k_0^2 - (2\pi p/a)^2 - (2\pi q/b)^2}\right]$ the image amplitude follows to

$$V^{pq}(x,y) = \sum_{n,m} V_{nm}^{pq} e^{j2\pi(nx/a+my/b)}, \tag{1}$$

whereby a and b are the mask periods and the coefficients V_{nm}^{pq} are given by

$$V_{nm}^{pq} = A_{pq} T_{n-p,m-q} \mathcal{P}(n,m) e^{jk_0\Phi(n,m)}. \tag{2}$$

Here, A_{pq} is the amplitude of the plane wave, T_{nm} represents the Fourier coefficients of the photomask, $\mathcal{P}(n,m)$ stands for the pupil function, and $\Phi(n,m)$ is the aberration function of the projection lens. The general expressions (1) and (2) apply for both the scalar as well as for the vector simulation. In the

former case $V^{pq}(x,y)$ represents the scalar field amplitude, in the latter case it stands for the electric field amplitude. The pupil function is given for the scalar simulation by

$$P(n,m) = \begin{cases} M & \text{if } \sqrt{(n\frac{\lambda}{a})^2 + (m\frac{\lambda}{b})^2} \leq NA, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

and for the vector case by

$$\mathbf{P}(n,m:p,q) = \begin{cases} M\boldsymbol{\Psi}(n,m:p,q) & \text{if } \sqrt{(n\frac{\lambda}{a})^2 + (m\frac{\lambda}{b})^2} \leq NA, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

In the above two equations M is the magnification of the projection system, NA is the numerical aperture, and λ is the used wavelength. The quantity $\boldsymbol{\Psi}(n,m:p,q)$ describes the polarization state of the illumination and is computed by following the plane waves through the system [2]. The contribution $I_i^{pq}(x,y)$ of one point source to the aerial image is calculated by

$$I_i^{pq}(x,y) = \frac{1}{2}\sqrt{\frac{\mu_0}{\varepsilon_0}}|V^{pq}(x,y)|^2. \tag{5}$$

1.1.2 Lens Aberrations

The operation of an ideal lens is only diffraction limited, i.e., the lens catches rays only up to a certain degree of obliquity, but otherwise the image is composed in an ideal way. In reality fabrication imperfections of the lens degrade the image quality as the image composition is distorted by phase errors occurring during the passage of the ray through the lens. Such imperfections are commonly called *lens aberrations* and are modeled by the general phase factor $\exp(jk_0\Phi(n,m))$ in (2).

Defocus. The simplest and most basic aberration type is defocus caused by a wrong vertical position of the image plane. In case of a positioning error Δ_{Def} the plane waves do not converge at the wafer. The optical path difference depends on the orientation of the incident rays. The aberration function describing defocus follows from the plane wave decomposition (1) and writes to

$$\Phi(n,m) = \Delta_{\text{Def}} i_{z,nm},$$

whereby $i_{z,nm}$ is the vertical component of the normalized wavevector

$$\mathbf{i}_{nm} = \left[n\lambda/a,\ m\lambda/b,\ \sqrt{1-(n\lambda/a)^2-(m\lambda/b)^2}\right]^{\mathrm{T}}.$$

The paraxial approximation valid for almost vertical incident rays, i.e., $i_{x,n}, i_{y,m} \ll 1$, yields the simpler relation

$$\begin{aligned} \Phi(n,m) &= \Delta_{\text{Def}}\sqrt{1-i_{x,n}^2-i_{y,m}^2} \\ &\cong \Delta_{\text{Def}} + \frac{1}{2}(i_{x,n}^2+i_{y,m}^2)\Delta_{\text{Def}}, \end{aligned} \tag{6}$$

which will subsequently be used to study the impacts of higher-order aberration types. The obvious effect of defocus is a vertical shift of the focal point above or below the Gaussian image point depending on the sign of Δ_{Def}. However, if Δ_{Def} varies across the lens field, the surface of best imagery is not planar and the usable depth of focus is correspondingly reduced.

Power Series Representation of Primary Seidel Aberrations. The common way to describe the five primary or Seidel aberrations is a power series expansion of the aberration function $\Phi(n,m)$ [1, pp. 211–218]. It is convenient to scale the incident wavevector $\mathbf{i}_{nm}$ in the image space with the numerical aperture NA of the imaging lens first, i.e.,

$$\boldsymbol{\imath}_{nm} = \frac{1}{NA}\mathbf{i}_{nm},$$

which has the advantage that the pass-region of the pupil function (cf. (3) and (4)) transforms in the scaled domain into the unit circle:

$$i_{x,n}^2 + i_{y,m}^2 \leq NA^2 \qquad \Longrightarrow \qquad \imath_{x,n}^2 + \imath_{y,m}^2 \leq 1.$$

Hence, the optical path difference of the most obliquely incident ray does not depend on the numerical aperture and the resulting formulas provide clear information on the impact of each aberration type.

The power series expansion of the aberration function has a form of [3],

$$\Phi(n,m) = \lambda \sum_{\substack{a,b,c\geq 0\\ a+b+c=k}} \Delta_{abc}(x_o^2+y_o^2)^a(x_o\imath_{x,n}+y_o\imath_{y,m})^b(\imath_{x,n}^2+\imath_{y,m}^2)^c, \tag{7}$$

whereby Δ_{abc} denotes the maximal optical path difference in units of the wavelength between the rays entering in the center of the lens and at the boundary, while (x_o, y_o) is the position of the mask with respect to the optical axis. In the power series only polynomials of degree $2k$ in the three variables $r_o = \sqrt{x_o^2+y_o^2}$, $\rho = \sqrt{\imath_{x,n}^2+\imath_{y,m}^2}$, and $\kappa = \sqrt{x_o\imath_{x,n}+y_o\imath_{y,m}}$ occur. A term of a particular degree $2k$ is said to represent a *wave aberration of order 2k*. The aberration of lowest order ($2k = 4$) are usually called *primary* or *Seidel aberrations*. As can be seen from (7) five terms with $a+b+c=2$ exist. In Table 1 a summary of the Seidel aberrations including the resulting image shift is described.

Although the expansion of (7) is a general representation of the aberration function in that any aberration function $\Phi(n,m)$ can be represented, other polynomials are more suited to describe higher-order aberration terms.

Table 1: Seidel aberrations are described by power series expansion of the aberration function $\Phi(n,m)$. The position of best focus is shifted by $(R_f\cos\phi_o, R_f\sin\phi_o, Z_f)$.

Aberration	(a,b,c)	$\Phi(n,m)$	Focal shift	
			R_f	Z_f
Spherical	(0,0,2)	$\lambda\Delta_{\mathrm{Sph}}(\imath_{x,n}^2+\imath_{y,m}^2)^2$	0	$\frac{2\lambda}{NA^2}\Delta_{\mathrm{Sph}}$
Coma	(0,1,1)	$\lambda\Delta_{\mathrm{Com}}(x_o\imath_{x,n}+y_o\imath_{y,m})(\imath_{x,n}^2+\imath_{y,m}^2)$	$\frac{2\lambda r_o}{3NA}\Delta_{\mathrm{Com}}$	0
Astigmatism	(0,2,0)	$\lambda\Delta_{\mathrm{Ast}}(x_o\imath_{x,n}+y_o\imath_{y,m})^2$	0	$\frac{\lambda r_o^2}{NA^2}\Delta_{\mathrm{Ast}}$
Curvature*	(1,0,1)	$\lambda\Delta_{\mathrm{Cur}}(x_o^2+y_o^2)(\imath_{x,n}^2+\imath_{y,m}^2)$	0	$\frac{2\lambda r_o^2}{NA^2}\Delta_{\mathrm{Cur}}$
Distortion	(1,1,0)	$\lambda\Delta_{\mathrm{Dis}}(x_o^2+y_o^2)(x_o\imath_{x,n}+y_o\imath_{y,m})$	$\frac{\lambda r_o^3}{NA}\Delta_{\mathrm{Dis}}$	0

*For a fixed object position curvature equals "approximate" defocus (cf. (6)), whereby the amount of defocus Δ_{Def} is related to the optical path difference Δ_{Cur} by $\Delta_{\mathrm{Def}} = 2\lambda r_o^2\Delta_{\mathrm{Cur}}/NA^2$.

Zernike Polynomials for General Aberration Terms General aberration terms are described most conveniently by the *circle polynomials* $\mathcal{Z}_b^c(\rho, \phi)$ *of Zernike* [1, pp. 464–468]. The aberration function $\Phi(n, m)$ is expanded as follows:

$$\Phi(n,m) = \lambda \sum_{\substack{a,b \geq 0,\, c \\ a-|c| \text{ even}}} \Delta_{abc} A_{ac} \mathcal{Z}_b^c(\rho_{nm}, \phi_{nm}).$$

As before, Δ_{abc} refers to the maximal optical path difference of the lens measured in units of the wavelength, while the factor A_{ac} is determined by the object location $(r_o \cos\phi_o, r_o \sin\phi_o)$,

$$A_{ac} = \begin{cases} r_o^{2a+|c|} \cos(c\phi_o) & \text{if } c \geq 0, \\ r_o^{2a+|c|} \sin(c\phi_o) & \text{otherwise.} \end{cases}$$

The orientation of the incident wavevector is described by the polar coordinates (ρ_{nm}, ϕ_{nm}),

$$\rho_{nm} = \sqrt{i_{x,n}^2 + i_{y,m}^2} \qquad \text{and} \qquad \phi_{nm} = \arctan\left(\frac{i_{x,n}}{i_{y,m}}\right).$$

The Zernike polynomials in real-valued form are defined as

$$\mathcal{Z}_b^c(\rho, \phi) \stackrel{\text{def}}{=} \begin{cases} \mathcal{R}_b^c(\rho) \cos(c\phi) & \text{if } b \geq 0, \\ \mathcal{R}_b^c(\rho) \sin(c\phi) & \text{otherwise,} \end{cases}$$

whereby $\mathcal{R}_b^c(\rho)$ are the so-called orthogonal *radial polynomials*. A thorough discussion can be found in, e.g., [1, Appendix VII]. In Table 2 the explicit forms of these polynomials for the first few values of the indices b and c are listed. The normalization has been chosen so that $\mathcal{R}_b^c(1) = 1$ for all permissible values of b and c.

Table 2: Radial polynomials $\mathcal{R}_b^c(\rho)$ for $b \leq 7$, $c \leq 7$ (after [1, p. 465]).

$c \backslash b$	0	1	2	3	4	5	6	7
0	1		$2\rho^2 - 1$		$6\rho^4 - 6\rho^2 + 1$		$20\rho^6 - 30\rho^4 + 12\rho^2 - 1$	
1		ρ		$3\rho^3 - 2\rho$		$10\rho^5 - 12\rho^3 + 3\rho$		$35\rho^7 - 60\rho^5 + 30\rho^3 - 4\rho$
2			ρ^2		$4\rho^4 - 3\rho^2$		$15\rho^6 - 20\rho^4 + 6\rho^2$	
3				ρ^3		$5\rho^5 - 4\rho^3$		$21\rho^7 - 30\rho^5 + 10\rho^3$
4					ρ^4		$6\rho^6 - 5\rho^4$	
5						ρ^5		$7\rho^7 - 6\rho^5$
6							ρ^6	
7								ρ^7

1.1.3 Advanced Illumination Aperture

The illuminator of a projection printing system is not a simple point source. Spatially extended apertures are often applied to enhance the imaging performance. A convenient method to model imaging with such advanced illuminators is due to *Abbe* [1, pp. 418–424]. This method is based on a spatial discretization of the source into discrete point sources, whereby the point sources are mutually incoherent due to the thermal

nature of the light source. The statistical independence is of crucial importance for the performance of projection systems as only a local spatial coherence is introduced on the mask. The image is then calculated by taking one incident ray at a time, finding its diffraction, and summing the collected field.

For the image calculation of one coherent source point either the scalar or the vector formulation can be used. However, the aerial image is obtained by an incoherent superposition of all the contributions (5) and formally writes to

$$I_i(x,y) = \sum_{p,q} I_i^{pq}(x,y). \tag{8}$$

The point sources (pq) are located within the illumination aperture (cf. Figure 1). Beside this no further restrictions on the shape of the aperture are made.

The spacing between the individual point sources has to satisfy a specific criterion. As mentioned before, it is chosen in such a way that the lateral wavevector components $k_{x,p}$ and $k_{y,q}$ of the waves which incident on the photomask equal an integer multiple of the sampling frequencies $2\pi/a$ and $2\pi/b$. This choice guarantees that the image on the wafer surface is periodic with periods a and b, which enables an extremely fast implementation of the resulting formulas. This requirement is illustrated in the wavevector diagram of Figure 1 for the case of conventional illumination.

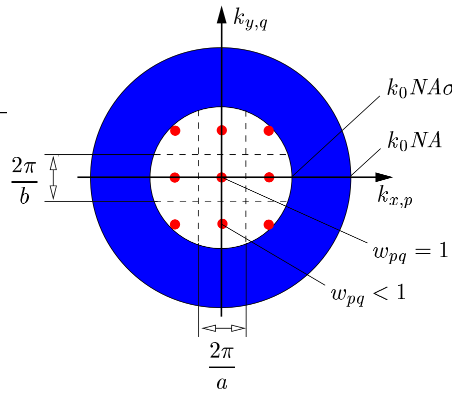


Figure 1: Source discretization of a conventional partially coherent illumination system. The spacing between the individual point sources is chosen to yield a periodic field incident on the photomask.

What is missing now is the determination of the amplitudes A_{pq} of the point sources. Clearly, they depend on the total imaging intensity I_0 as well as on the discretization scheme illustrated in Figure 1. The intensity of one point source is related to the discretization area w_{pq} and to the square of the light amplitude A_{pq}^2. Hence, we write

$$A_{pq}^2 = A_0^2 w_{pq}, \tag{9}$$

where A_0 is a simple normalizing constant. By considering a missing or "perfectly transparent", we obtain the relationship between A_{pq} and I_0 looked for [4],

$$\text{scalar theory:} \qquad A_0 = \frac{1}{M}\sqrt{\frac{2\eta_0 I_0}{\sum_{p,q} w_{pq}}}$$

$$\text{vector theory:} \qquad A_0 = \frac{1}{M}\sqrt{\frac{2\eta_0 I_0}{\sum_{p,q} w_{pq}\sqrt{1-(p\lambda/a)^2-(q\lambda/b)^2}}}.$$

1.1.4 Numerical Implementation

From the above discussion it can be seen that basically two Fourier transforms are required:

- **Forward Transform:** The Fourier coefficients T_{nm} of the mask transmission function $t(x,y)$ have to be calculated. As in our model $t(x,y)$ consists of piecewise constant regions with ideal sharp transitions, a simple sampling is not adequate because $t(x,y)$ is definitely no low-pass function. Hence, we developed a semi-analytical method to avoid errors arising from aliasing.
- **Backward Transform:** The Fourier coefficients T_{nm} are weighted with the pupil function and have then to be transformed back to the spatial domain to obtain the image field. As the projection lens acts as a low-pass filter due to the finite extent of the pupil function the image has only spectral components up to a certain order. Therefore sampling can be employed for the backward transform without introducing any aliasing.

Consequently, the calculation of the aerial image is in a certain sense "exact", i.e., no additional discretization errors are introduced by the numerical implementation. All approximations made throughout can be physically explained and therefore constitute a sound physical and mathematical basis of the aerial image simulator.

"Alias Free" Forward Transform: The basic idea is to decompose the mask transmission function $t(x,y)$ into elementary geometrical patterns. Triangles are most suited for this purpose because any function $t(x,y)$ consisting of piecewise constant transmission areas can be triangulated and an analytical expression for the Fourier coefficients of a triangular shaped area can easily be derived [4]. As a typical photomask often consists of rectangular areas we consider them as the second type of elementary patterns. Formulas for the Fourier coefficients of rectangles can also be found in [4].

Let us now find a mathematical description for the proposed algorithm. The area of a general elementary pattern is described by its indicator function $\mathrm{PAT}(x,y)$ defined as

$$\mathrm{PAT}(x,y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } (x,y) \text{ inside the pattern,} \\ 0 & \text{otherwise.} \end{cases}$$

In our situation the elementary patterns are either triangles or rectangles. In principle any other pattern type can be used as long as analytical expressions for its Fourier coefficients can be derived.[1] The decomposition of the mask transfer function writes as

$$t(x,y) = \sum_k t_k \mathrm{PAT}_k(x,y), \tag{10}$$

whereby t_k is the transmission of the respective elementary pattern. In the case of binary masks all transmissions t_k equal unity, for phase-shifting masks t_k is a complex number with module less or equal one, i.e.,

$$\begin{aligned} \text{binary masks:} \quad & t(x,y) \equiv 1 \\ \text{phase-shifting masks:} \quad & |t(x,y)| \leq 1 \ \wedge \ t(x,y) \in \mathsf{C}. \end{aligned}$$

[1] The Fourier coefficients of polygonal areas can always be computed analytically; circular or elliptic areas can also be treated analytically.

Due to the linearity of the Fourier transform the coefficients T_{nm} of the mask transmission function $t(x,y)$ can be computed as follows:

$$\begin{aligned} T_{nm} &= \frac{1}{ab} \int_0^a \int_0^b t(x,y) e^{-j2\pi(nx/a+my/b)}\, dxdy \\ &= \frac{1}{ab} \int_0^a \int_0^b \left[\sum_k t_k \mathrm{PAT}_k(x,y) \right] e^{-j2\pi(nx/a+my/b)}\, dxdy \\ &= \sum_k t_k \left[\frac{1}{ab} \int_0^a \int_0^b \mathrm{PAT}_k(x,y) e^{-j2\pi(nx/a+my/b)}\, dxdy \right]. \end{aligned} \tag{11}$$

The term in square brackets in the last row, i.e.,

$$\mathcal{P}_{k,nm} = \frac{1}{ab} \iint_{(x,y)\in \mathrm{PAT}_k} e^{-j2\pi(nx/a+my/b)}\, dxdy, \tag{12}$$

can be calculated analytically for the elementary patterns. With (10) and (11) the Fourier coefficients T_{nm} follow to

$$T_{nm} = \sum_k t_k \mathcal{P}_{k,nm}.$$

The algorithm has the big advantage that the Fourier coefficients T_{nm} are calculated exactly, i.e., no sampling of the mask transfer coefficients is involved. In addition to accuracy speed is the second important benefit. The calculation is faster when compared to a numerical FFT transform, because the formulas for $\mathcal{P}_{k,nm}$ can easily be evaluated. This second point is especially important for a dense layout as the mask can be decomposed into a relatively small number of geometrical patterns.

Numerical Backward Transform: The spectrum V_{nm}^{pq} due to one point source is determined by the following three quantities: source amplitude A_{pq}, Fourier coefficients T_{nm} of the mask transfer function, and discrete pupil function $P(n,m)$ or $\mathbf{P}(n,m:p,q)$. For each of them formulas are provided above. Hence, the spectrum can be calculated and the field is obtained by evaluating the transform in (2). This last step is accomplished as follows.

The finite extent of the pupil function in (3) or (4) shows that the image spectrum is always a low-pass function, i.e.,

$$V_{nm}^{pq} = 0 \qquad \text{for} \qquad \sqrt{\left(\frac{n}{a}\right)^2 + \left(\frac{m}{b}\right)^2} > \frac{NA}{\lambda}. \tag{13}$$

Hence, the fundamental results of the *sampling theory* [5] can be applied, which states that any bandlimited, finite-energy function $f(x)$ can be reconstructed from discrete equally-spaced samples $f(n\Delta x)$, if the spacing between the samples Δx meets the *Nyquist criterion* $\Delta x \leq 1/2W$, whereby W is the highest frequency occurring in $f(x)$. Self-evidently the sampling theorem is also valid for functions defined in a higher dimensional space. As can be seen from (13) the aerial image spectrum V_{nm}^{pq} is circularly bandlimited with cut-off frequency NA/λ. Consequently, it suffices to evaluate the field $V^{pq}(x,y)$ at discrete points only without any loss of information. We choose the spacings Δx and Δy to meet the Nyquist criterion and require that the numbers of samples within the mask periods N_x and N_y equal a power of

two,[2] i.e.,

$$\Delta x \leq \frac{\lambda}{2NA} \qquad \wedge \qquad \frac{a}{\Delta x} = 2^{r_x}$$
$$\Delta y \leq \frac{\lambda}{2NA} \qquad \wedge \qquad \frac{b}{\Delta y} = 2^{r_y}.$$

Thus the powers $r_x = \mathrm{ld}\, N_x$ and $r_y = \mathrm{ld}\, N_y$ are chosen as the smallest integers meeting

$$r_x \geq \mathrm{ld}\left(\frac{aNA}{2\lambda}\right) \qquad \text{and} \qquad r_y \geq \mathrm{ld}\left(\frac{bNA}{2\lambda}\right).$$

The field disturbance on the equally spaced ortho-product grid is given now by

$$V^{pq}(k\Delta x, l\Delta y) = \sum_{n=-N_x/2}^{N_x/2-1} \sum_{m=-N_y/2}^{N_y/2-1} V_{nm}^{pq} e^{-j2\pi(nk/N_x + ml/N_y)}.$$

This formula can efficiently be evaluated with a radix-2 FFT algorithm [6, Chapter 12], [7, Chapter 11]. We have chosen a public-domain software package that employs heavily inlined FFT routines and precalculated weight tables to achieve maximal speed.[3] The length of the weight tables is 4096 and the unroll value is 128. Above the unroll value conventional recursive routines are employed. To check the correctness of the algorithm we compared the results with the well-established packet FFTPACK [8] from the NETLIB.[4] Information on the computational performance of the aerial image tool, e.g., run-time and memory consumption, is given in connection with simulation examples.

[2] The second requirement is not obligatory, but the resulting formulas can then be evaluated with the most efficient radix-2 FFT algorithm.

[3] This package was written by Richard Krukar and is available at http://hpux.u-aizu.ac.jp/hppd/hpux/~~/Maths/Misc/ffts_in_C-1.0/readme.html.

[4] The package FFTPACK was written by Paul N. Swarztrauber and is widely available, e.g., from the original NETLIB location at http://www.netlib.org/fftpack/index.html.

1.2 Handling layouts for lithography simulation

The PED tool can be used to edit or import from exterior frameworks (if in CIF or GDSII) layout information. This can be handled in an interactive way in order to prepare it to the simulators inside VISTA. Cut-lines or areas of interest can be defined if respectively two or three-dimensional simulator are used.

As with lithography simulation is concerned, we extended the basic features of a layout editor, with the capabilities for phase-shift masks edition, as this feature is not available in conventional ECAD layout editors. For the background as well to every polygon of any mask, we can specify values to the modulus and phase attributes of their transmittance (see Figure 2).

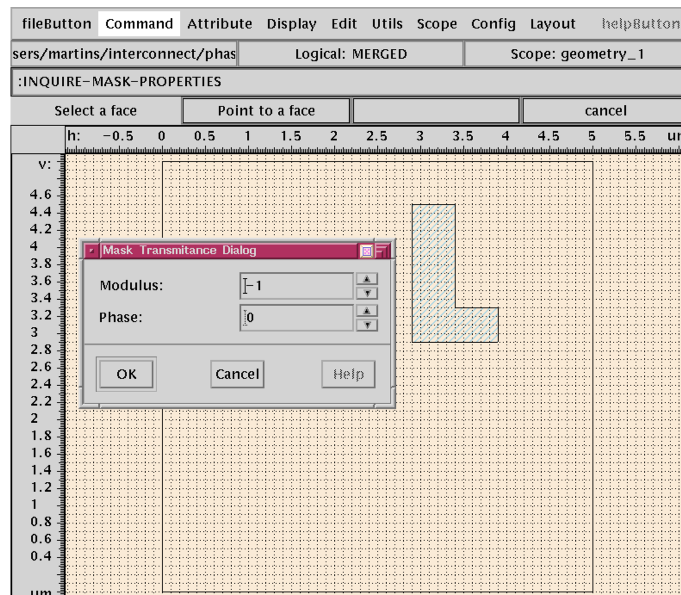


Figure 2: The capabilities of PED are extended to edit phase-shift masks.

We introduce also a button in the PED panel to make easy the task of checking the printability of one layout. It evokes the simulator described previously, and from the aerial image result, automatically identifies possible problems with the printability of the features in the layout. This is done by applying a threshold to the resulting aerial image to create a binary image. Using digital imaging processing techniques we detect the contours of the binary aerial image, and then we write this information in a layout file, that can be used by topography simulators. From comparisons between this layout and the original, we investigate its printability.

1.3 Examples

Figure 3 shows simulation results of one layout with nested elbows. The minimum feature size is 0.4 μm. We compare conventional binary masks (left) with alternating phase-shifting masks (right). The simulation parameters were a numerical aperture of $NA = 0.55$, a wavelength of365 nm (I-line), and a focus error of $1 \mu m$. Figure 3-(iv) demonstrate the superior performance of phase-shifting masks.

The lithography package can handle large pieces of layout. We show in Figure 4 an example where the simulation domain was 29 μm $\times$ 23 μm large. With a spatial grid of 256 $\times$ 256 and 1056 point sources the simulation time is about 5 minutes on a DEC-600 workstation. A quadrupole source was used with $\sigma = 0.22$ (see Figure 1). The distance between the four sources must be optimized according the minimum

feature size, and yield for this layout: X=Y=0.556. We used again a numerical aperture of $NA = 0.55$ and a wavelength of 365 nm.

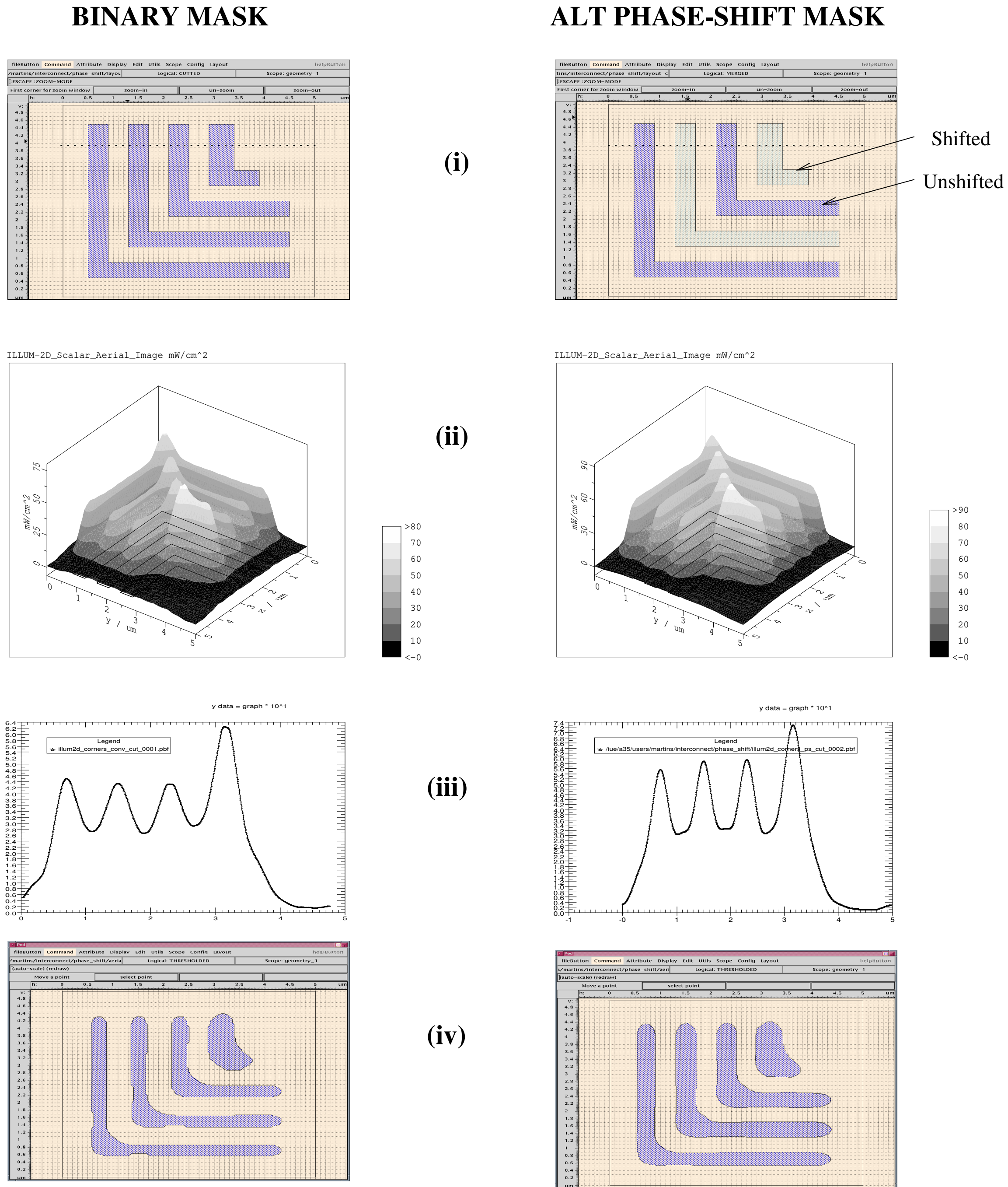
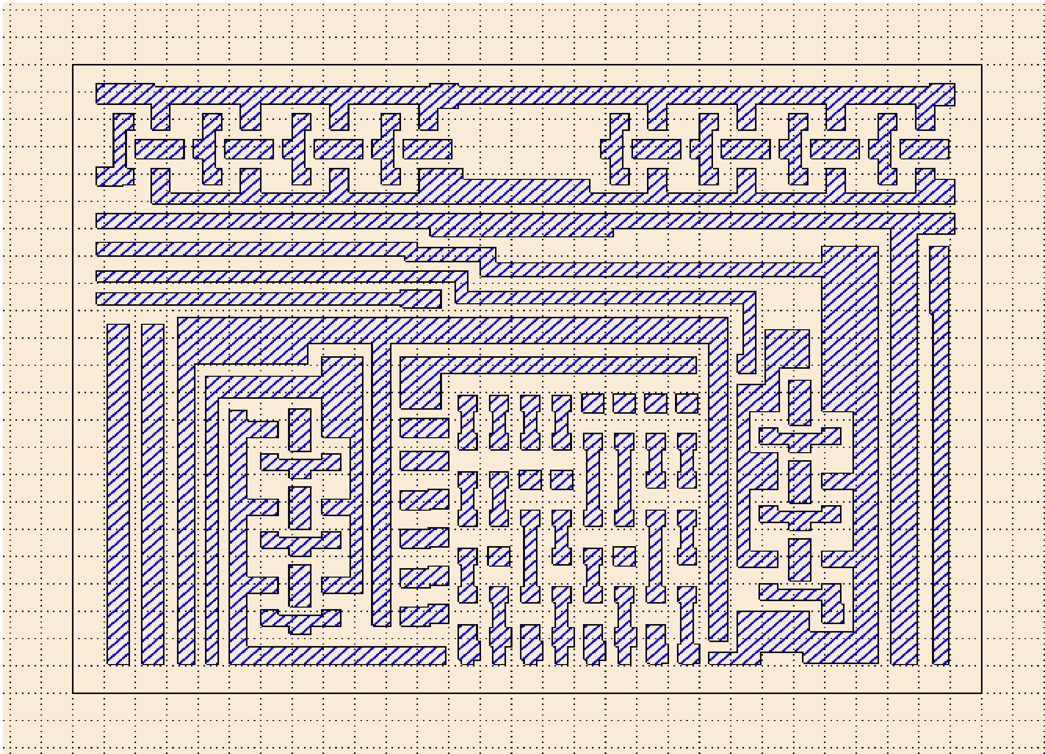
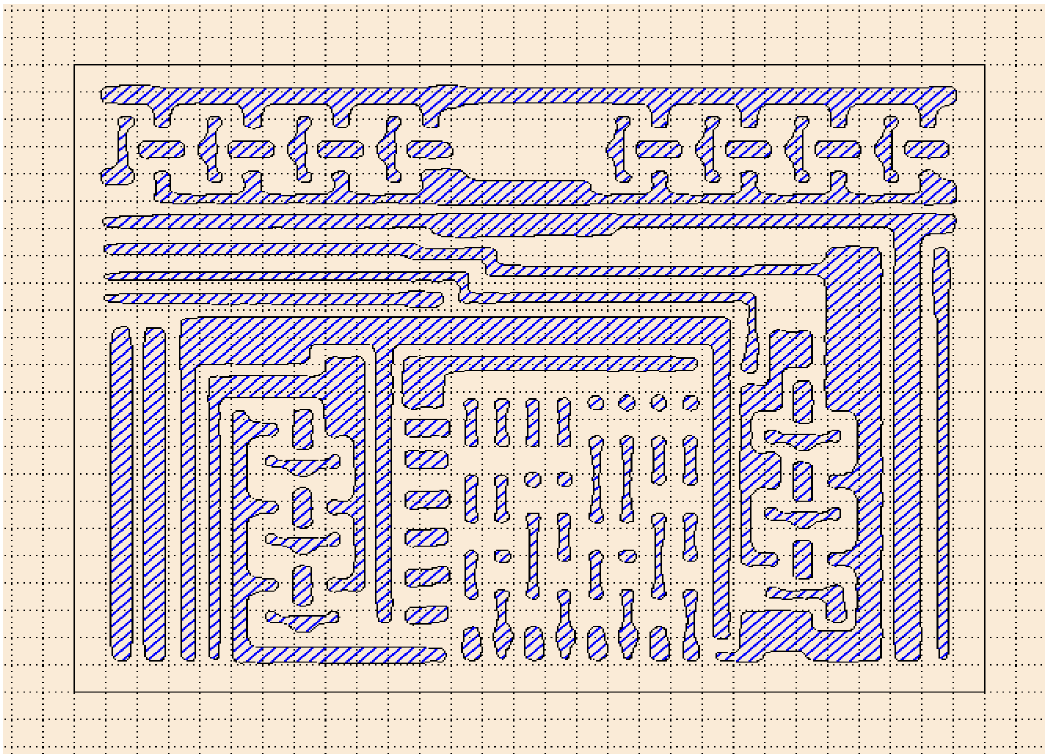


Figure 3: Simulation of nested elbows: (i) Layout; (ii) Aerial image; (iii) Aerial image cutted at horizontal line $y = 3.95\ \mu m$; (iv) Binary mask after applying threshold (see text).



(i)



(ii)

Figure 4: Simulation of a large piece of layout. (i) Original Layout (ii) The result. Note: Grid spacing= 1 μm.

2 Simulation of Ion Implantation with MCIMPL

2.1 Introduction

For the simulation of ion-implantation mainly two approaches are used. Firstly, an analytical method where the doping profile is represented by probability functions, with probability moments depending on various implantation parameters. Secondly, a Monte Carlo method where the trajectories of all ions are followed on their way through the target until they come to rest or leave the simulation domain.

The ion-implantation simulator MCIMPL uses the Monte Carlo method. MCIMPL can handle fully three dimensional applications, but also lower dimensional problems. The simulation domain can consist of several amorphous and crystalline segments made up of various materials normally used in semiconductor fabrication. The only restriction is, that all crystalline segments have to consist of silicon. MCIMPL can not only handle single atomic ions, but also molecular ions, either via a simple model where the energy of the doping material is weighted according to the mass ratio in the molecule, or via a more accurate approach where the trajectories of all atoms of the molecule are calculated.

In the following sections the simulator MCIMPL is described more in detail. Some input parameters related to the text are stated in parentheses.

2.2 Models

For the simulation of ion-implantation using a Monte-Carlo method mainly two types of physical models are required. When the ion enters the target material, the nearest target atom is estimated, either randomly for amorphous materials or by checking the positions of the atoms in the crystal. Then the scattering of the ion by the target atom is calculated. Therefore a binary collision model for two charged particles is used. The interaction potential of the two atoms is described by the function [9] [10] [11] [12] [13]

$$V_c(r) = \frac{Z_1 Z_2 e^2}{r} \cdot \Phi(r) \tag{14}$$

where Z_1 and Z_2 are the core charges and r is the distance of the centers of mass of the two atoms. The function $\Phi(r)$ describes the screening of the core charges by the surrounding electrons. MCIMPL uses the universal screening potential

$$\Phi(r) = \sum_{i=1}^{n} a_i \cdot \exp(-b_i \frac{r}{a_I}) \tag{15}$$

with the coefficients

a_i	b_i
0.18175	3.19980
0.50986	0.94230
0.28022	0.40290
0.02817	0.20160

and the screening length

$$a_I = \frac{0.468}{Z_1^{0.23} + Z_2^{0.23}} \tag{16}$$

For computation time reduction a pre-calculated table of scattering angles is used from which the amount of energy which is transfered to the target atom is derived.

After one scattering event, the ion is considered as flying along a straight line until the next collision. The length of the free flight-path is estimated by looking for the next target atom if the ion moves through a crystalline material, or an average free flight-path, which is a function of the density of the material, is taken if the ion moves through amorphous material.

Along the free flight-path the ion looses energy because of the interaction with electrons of the target material. To handle this electronic stopping process MCIMPL uses a partly empirical model which is based on the stopping of charged particles moving through a plasma of electrons [14]. For crystalline materials this model is modified by some empirical expressions that take into consideration, that the density of the electrons varies within a solid, mainly depending on the distance from lattice atoms [15] [16] [17] [18] [19]. The loss of energy ΔE of an ion in the electron gas is

$$x_{nl} = \max(y_{nl}E^q), a = f\frac{a_I}{0.3} \tag{17}$$

$$x_{loc} = 1 - x_{nl} \tag{18}$$

$$\Delta E_{loc} = x_{loc}\frac{k_{corr}k\sqrt{E}\Delta R}{2a^2\pi} \cdot \sum_{i=1}^{n}\left(\exp\left(-\frac{p_i}{a}\right)\right) \tag{19}$$

$$\Delta E_{nl} = Nk_{corr}k\sqrt{E}\Delta R \cdot \left[x_{nl} + x_{loc} \cdot \left(1 + \frac{p_{max}}{a}\right) \cdot \exp\left(-\frac{p_{max}}{a}\right)\right] \tag{20}$$

$$\Delta E = \Delta E_{nl} + \Delta E_{loc} \tag{21}$$

where n is the number of atoms around the current ion position that contribute to the local part of the electronic stopping power, p is their distance from the ion and p_{max} is the maximum distance that is searched for surrounding atoms. ΔR is the length of the flight-path, E is the current ion energy and N is the density of the atoms. The coefficient a_I is the screening length as described above. k_{corr} , f , q , y_{nl} are empirical coefficients which depend on the ion species.

For amorphous materials the energy loss caused by electronic stopping is calculated by

$$\Delta E = Nk_{corr}k\sqrt{E}\Delta R \tag{22}$$

The parameters are the same as mentioned for the crystalline case.

When the ion hits an atom of the target material, energy is passed to the target. If this energy is higher than the displacement energy of the target atom, it is pushed away from its lattice cite. This recoil itself can afterwards remove other atoms from their original positions in the lattice and a cascade of collisions is initiated. Because of that a lot of interstitial atoms are generated. But it is possible that some of these interstitials move to regular lattice sites. In MCIMPL an empirical model is used to describe the damage generation, which also takes recombination effects into account. The number of interstitials that are generated by one cascade are calculated using the modified Kinchin-Pease-Model [20] [21].

$$N_I = \begin{cases} 0 & 0 \le E_\nu \le E_d \\ 1 & E_d \le E_\nu \le 2.5E_\nu \\ \frac{0.8E_\nu}{2E_d} & 2.5E_d \le E_\nu \end{cases} \tag{23}$$

where E_d is the displacement energy of one lattice atom and E_v is that part of the transfered energy, that is available for damage generation. To consider the recombination, two additional parameters are introduced to the model [13] and the number of remaining defects is

$$N_{IR} = N_I f_{rec} \cdot \left(1 - \frac{N_\nu}{N_{sat}}\right). \tag{24}$$

N_ν is the density of interstitials that already exists at the current ion position. The parameter N_{sat} describes a saturation density of point defects, whereas f_{rec} is an inverse recombination rate. Both parameters of this point defect model depend on the ion species.

Although there are good default values available for the modeling parameters k_{corr} (lindhardCorrection), f (screeningPrefactor), q (nonlocalPower), y_{nl} (nonlocalPrefactor), N_{sat} (saturation), f_{rec} (recombinationFactor), p_{max} (searchDistance), E_d (displacementEnergy) they can be modified by the user, as well as several other parameters like the final energy (finalEnergy) of an ion and the amplitude of the lattice atom vibration (atomVibration).

2.3 Simulation Input

MCIMPL needs a description of the simulation domain in a PIF file which has to provide the structure of the geometry and the material types of segments. In the case of a two dimensional simulation it is also possible to provide a triangular input grid (useGrid). MCIMPL uses the points of the input-grid to represent the simulation result and adds some grid-points if the resolution of the initial grid is not sufficient to fulfill the grid criteria that can be specified by the user (minGridAngle, maxDoseError, maxGradient).

In order to accurately describe the simulation conditions various parameters are available. The

name of the implanted ions (ionName),

energy of the implanted ions (energy),

implantation dose (dose),

tilt (tilt) and twist angle (twist) of the ion beam,

divergence (beamDivergence) of the ion beam,

orientation of the silicon crystal (crystalOrientation)

Furthermore it is possible to rotate the ion beam during the simulation, either continuously (revolving) or step-by-step (nbRotation).

2.4 Simulation Flow

First there is an initialization step where the input-geometry is prepared for the simulation and a rectangular implantation window is defined automatically if the user does not force the use of a special implantation window (implantationWindow). The staring positions of all calculated ions are equipartitioned within this implantation window, which is parallel to the x/y-plane and has the minimum possible area with respect to all ion beam directions during the simulation.

Before the simulation is started, the shape of the geometry is modified in order to treat the boundary conditions correctly. During the simulation the boundaries of the geometry are treated as real physical boundaries. The effect is that the doping concentration decreases in the vicinity of the left, right, front and back side of the geometry. In order to get the correct doping profiles over the whole simulated device, it has to be considered that just the top of the geometry is a real physical boundary, whereas all other boundaries are just artificial boundaries.

To avoid the problem of the decreasing doping profiles, the geometry is expanded on the left, right, front, and back side, assuming that the shape of the real device outside the simulated geometry can be derived

from the shape of the boundary. This means that the surrounding geometry can be generated by adding new prismatic solids to each face of the artificial boundaries. A special treatment is necessary for faces which contain points that belong to edges of the simulated geometry. In this case some lateral faces of the added prismatic solid have to be tilted in order to create a fully closed expanded geometry. The size of the expanded geometry depends on the penetration depth of the ions.

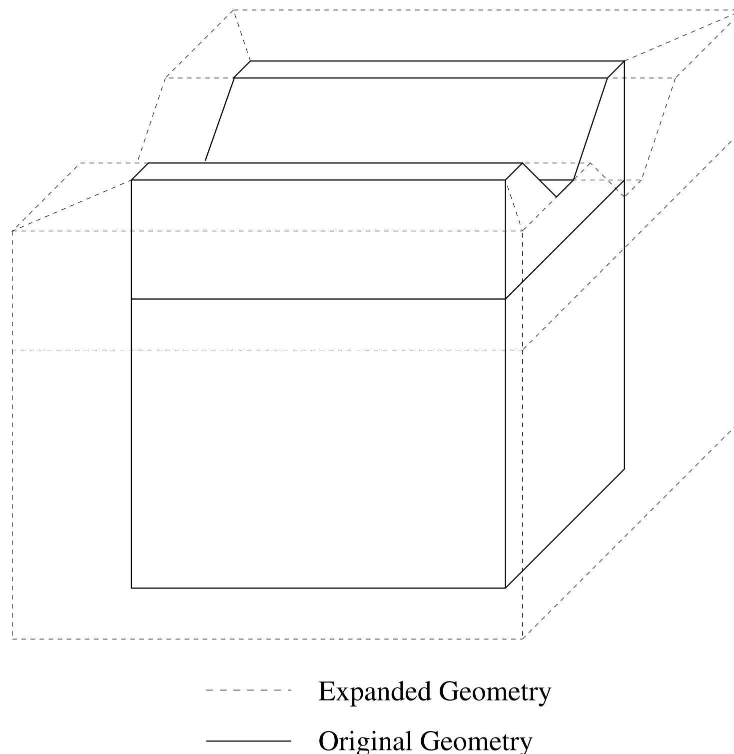


Figure 5: Shape of the simulation domain and the expanded geometry

While writing the output doping concentrations, only those points of the grid are considered which are a part of original simulation domain.

After expanding the geometry an ortho-product-grid is generated. In the two dimensional case the ortho-product-grid is automatically restricted to regions which can be reached by ion trajectories whereas in the three dimensional case the grid size in the x- and y-direction is determined automatically and the depth of the grid has to be set by the user (simulationDepth).

During the simulation all results are stored in the boxes that are bound by the grid lines. In general just the concentration of the doping profiles is calculated, but if damage is considered (siliconRecoil) also the distribution of the silicon interstitials and the damage energy density are evaluated. The damage energy density can be used to locate amorphous regions that have been generated during the implantation.

The results of the simulation are finally interpolated to an output grid. In the three dimensional case the previously generated ortho-product-grid is used as an output grid. In the two dimensional case the data are interpolated to a geometry conforming triangular grid which is refined according to the doping concentration as mentioned above. The results are written to a PIF file that contains the grid and the profiles. Additionally it is possible to generate a one dimensional output for each calculated species (output1d), which is the average of the profile at a certain depth.

2.5 Examples

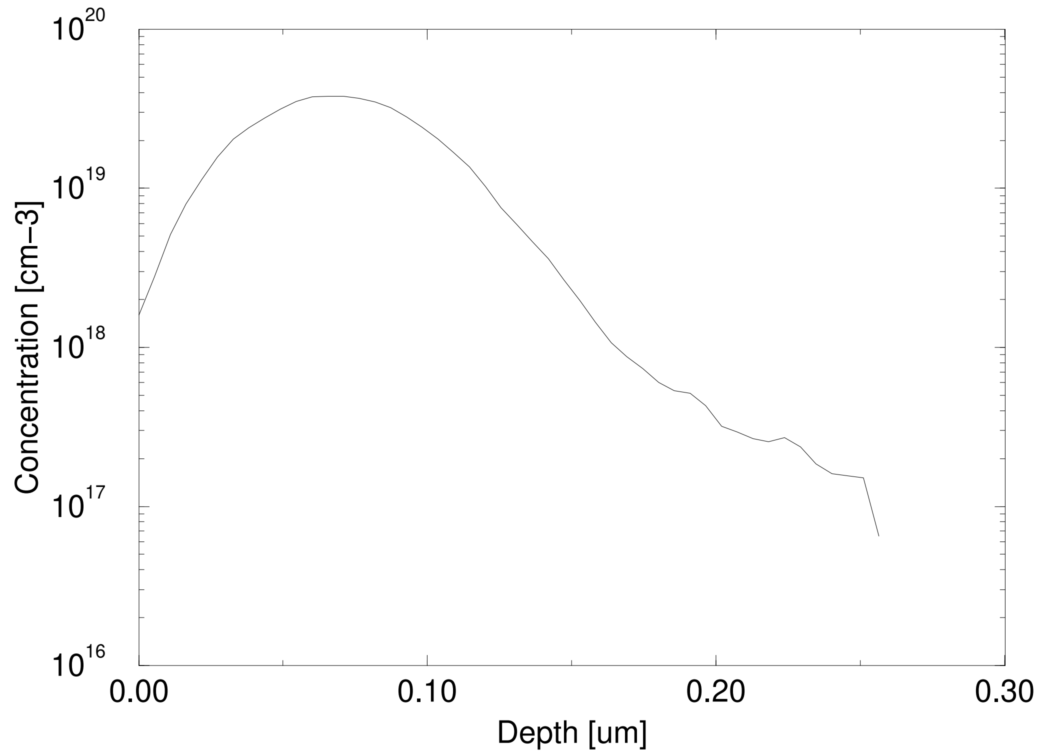


Figure 6: One-dimensional Simulation: Phosphorus implanted into bare silicon with 50keV, $3{\cdot}10^{14}\mathrm{cm}^{-2}$, 7° tilt, 180° twist

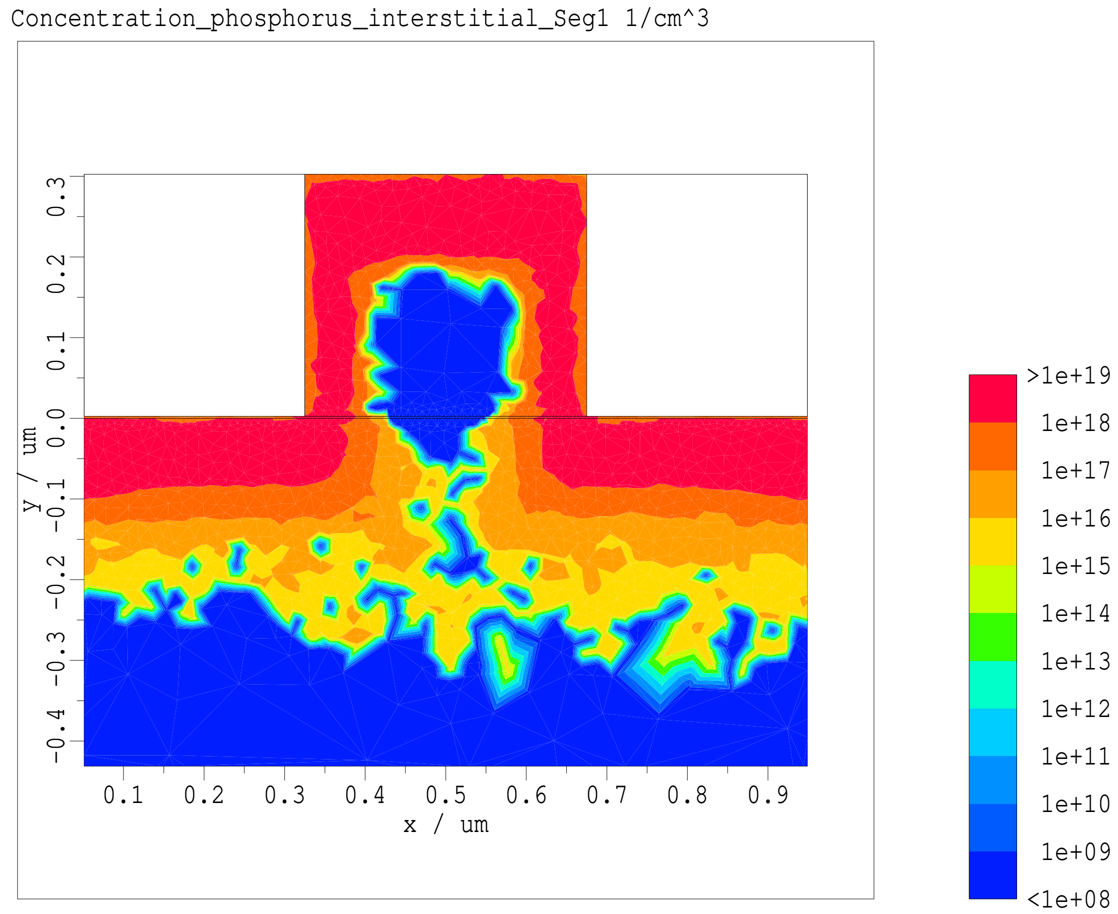


Figure 7: Two-dimensional Simulation: LATID Phosphorus implant with three rotation steps and 40keV, $4{\cdot}10^{13}\mathrm{cm}^{-2}$, 50° tilt, 50° twist

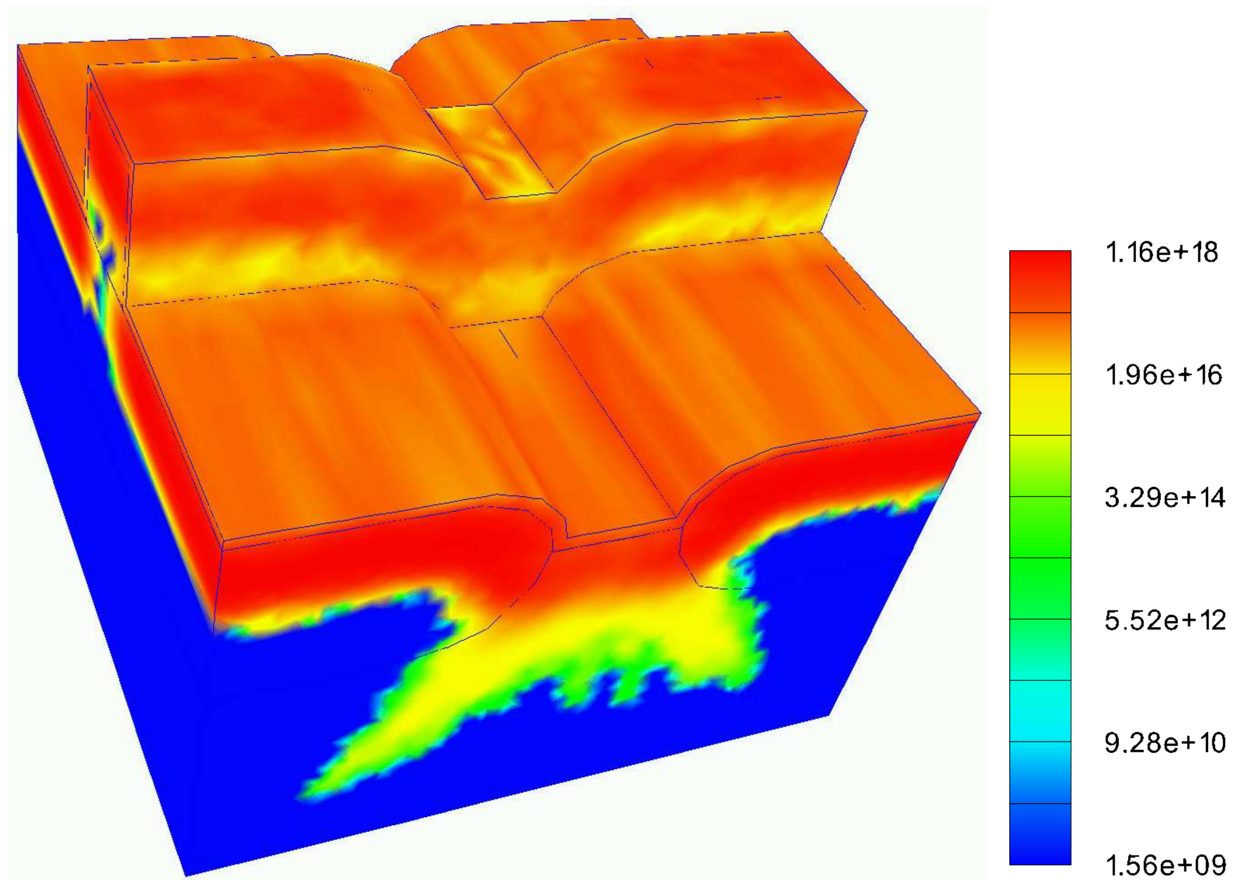


Figure 8: Three-dimensional Simulation: LAT Boron implant with one rotation step and 25keV, $7{\cdot}10^{12}\mathrm{cm}^{-2}$, 45° tilt, 0° twist

3 The MINIMOS-NT Input Deck

3.1 Introduction

A complex, general purpose device and circuit simulator like MINIMOS-NT puts heavy demands on its controlling language. In fact, a simple input deck as collection of static keywords is not sufficient. Even if this keywords are allowed to be functions, tables, or other complex objects, the burden of evaluation lies on the simulator. Normally, to control complex simulations like hydrodynamic transient mixed mode simulations, dozens of keywords are introduced to control all possible events. For instance, if the simulation failed to converge, several decisions for continuation are imaginable.

In the case of MINIMOS-NT a different approach has been taken. The main idea is to limit the number of keywords and group them in so called sections. These keywords are allowed to be time dependent functions, tables, expressions and the like. To accomplish this, the input deck is handled as a database which is only queried when the desired keyword is required. The simulator stores internal variables which should be externally accessible in a special section. These external variables (e.g., the iteration counter, update norms) can then be used to form keyword expressions in the input deck. When the simulator inquires a special keyword from the input deck, its current value is returned.

The entire input deck functionality has been implemented as a library to handle the complex situations which can arise. This library can be customized to the special needs since external functions written in the programming language C can be added to the internal function list of the input deck library. These external functions can be used like any other function for every keyword. This feature is used for the inclusion of so-called point clouds, which are general data tables stored in an ASCII file. Due to this powerful feature, the functionality of the simulator can be extended without changing one line of simulator code. For instance, the contact voltage of every voltage source can be given as a general function of time.

Another case where these features are utilized is the control of block iteration. Especially, hydrodynamic simulation requires such flexibility, because convergence is usually hard to achieve. For the definition of each of these iteration blocks, a section in the input deck exists. For each block several keywords such as the list of quantities to be solved, the damping scheme to use, parameters for this damping scheme, and a stop expression are supplied to control the iteration. These sections can be arbitrarily nested, forming complex sequences of iteration blocks.

3.2 Features of the Input Deck Control Language

The input deck language is a programming language of its own, which has been designed with simulator control in mind. Several typical features of programming languages are available, e.g., `if` statements, function definitions, etc. All necessary data-types for variables are available:

Type	Example
boolean	`a = true;`
integer	`a = 3;`
float	`a = 3.1415;`
quantity	`a = 3.1415 m;`
	`a = 3.1415 "m/s";`
string	`a = "This is a test";`
array	`a = [ 1, 2, 3 ];`
list	`a = [ 1, "pi", 3 A];`

For these data types, the basic mathematical operations are defined, strings may be concatenated by using the + operator, whereby non-string values convert to their string representation. Variables which are inquired by the simulator are called keywords.

All expressions are evaluated at runtime except constant expressions.

```
a = 3 + 4;
b = 4;
c = a + b;
```

In this example, the expression for `c` is stored and evaluated each time its value is inquired by the simulator, since the values for `a` and `b` may be overwritten by the simulator and may therefore change. On the other hand, the expression for `a` is evaluated only once, since there is no way for the simulator to overwrite `3` and `4` in the expression for `a`.

Variables may be grouped into sections, e.g.,

```
MySection
{
  a = "test";
  b = a + 3.0;
}
```

Sections can be arbitrarily nested. It is important to note, that the user can define as many variables and sections as needed to structure his personal data.

Functions can easily be defined by

```
min(a,b) = if (a<b, a, b);
```

Each variable or function has a unique name given by its path in the section hierarchy.

```
a = 1;

Test
{
  a = 2;

  Test
  {
    a = 3;
  }

  b1 = ^a;                // relative addressing
  b2 = ~a;                // absolute addressing

  b3 = a;                 // relative addressing
  b4 = ~Test.a;           // absolute addressing

  b5 = Test.a;            // relative addressing
  b6 = ~Test.Test.a;      // absolute addressing
}
```

In this case `b1 = b2 = 1`, `b3 = b4 = 2`, and `b5 = b6 = 3`. The tilde character (~) is used to denote an absolute path, the accent character (^) is used to go up one level in the hierarchy (several ^ can be used to go up more than one directory).

Sections can be copied using the **:** operator

```
Test1             // Create section Test1
{
  a = 1;
  b = 2;
}

Test2 : Test1;    // Create copy of section Test1
Test3 : Test1
{
  b = 3;          // Modify variable b of section Test3 only
  c = 4;          // Append variable c to section Test3
}

Test1             // Re-open section Test1
{
  a = 5;          // Modify variable b of section Test1 only
  d = 6;          // Append variable d to section Test1 only
}
```

Dumping the example above yields

```
Test1
{
  a = 5;
  b = 2;
  d = 6;
}

Test2
{
  a = 1;
  b = 2;
}

Test3
{
  a = 1;
  b = 3;
  c = 4;
}
```

3.2.1 Simulator Specifics

There is only a small number of predefined sections in the top level which are queried by the simulator. However, there are a lot of other sections defined for instance in the `defaults.ipd` file to allow grouping of related data, e.g., the `Devices` section. The top level sections queried by the simulator are the following:

- `SteppingControl`: Contains the stepping control algorithm for stepping simulations.

- `TimestepControl`: Contains the stepping control algorithm for transient simulations.
- `Circuit`: Contains the circuit net list.
- `Num`: Contains damping, solver, and scaling information.
- `Solve`: Contains information about the actual simulation.
- `Output`: Determines the kind of information which is written to distinct output files.

3.3 The `default.ipd` File

The file `defaults.ipd` is part of the MINIMOS-NT package. This file is the fundamental MINIMOS-NT input deck and has to be included into all user definded input decks. Despite of user-interaction with MINIMOS-NT, the `defaults.ipd` file is a very useful tool to test and develop new MINIMOS-NT code.

Usually the information in the `defaults.ipd` file is not time critical and therefore can be interpreted without any recompilation.

To get an idea what the file `defaults.ipd` does, some important parts of this file are described in the next sections. A complete description of `defaults.ipd` can be found in the MINIMOS-NT Users Manual.

3.3.1 The `Extern` Section

This section contains the keywords which are updated by the simulator. These keywords can be used afterwards to control the simulation. To update these keywords, the MINIMOS-NT executable has to call ipd-library functions.

```
Extern
{
  updateNorm        = 0.0;
  updateNormPot     = 0.0;
  updateNormLast    = 0.0;  // Update norm of the last parent block
  updateNormPotLast = 0.0;  // Update norm pot of the last parent block
  iterationCount    = 0;    // A counter of the current iteration block
}
```

3.3.2 The `Quantities` Section

In this section variables are defined, which group together quantities that are solved by MINIMOS-NT. The section typically looks like:

```
Quantities
{
  contact          = "Contact*";
  electrons        = "ElectronConcentration,BoundaryElectronCurrent";
  holes            = "HoleConcentration,BoundaryHoleCurrent";
  driftDiffusion   = contact + "," + electrons + "," + holes +
```

```
                    ",Potential,Boundary*,DisplacementField,Occ*";
  hydro           = "ElectronTemperature,HoleTemperature";
  mixed           = "NodeVoltage,FixNodeVoltage,BranchCurrent";
}
```

The variable `contact` groups together all quantities preceded by the letters ”`Contact`”. Examples for these quantities are `ContactPotential, ContactCurrent, ContactCharge`. The group `driftDiffusion` depends on the subgroup `contact` and several other quantities. The groups are connected with the + sign and separated with colons.

A list of all available quantities can be found in the MINIMOS-NT Users Manual.

Arrangement of quantities into groups is essential for the `iterate` section, where these groups are used for the iteration scheme.

3.3.3 The `Models` Section

This section defines segment specific information whether certain additive components for the equation system should be used. These components are built up from predefined component sets. Examples for components are SRH recombination, impact ionization, band to band tunneling etc. For example, the used impact ionization model calculates a generation rate which is entered as additive inhomogeneity to the right hand side of the equation system. The type of impact ionization model (the formula how to calculate the generation rate) is not specified. This is done in the `SegmentDefaults` section were one of the available impact models is chosen. The available model sets are can be found in the MINIMOS-NT Users Manual. The `Models` section reads:

```
Models
{
  testSRH           = ~getDevices("Phys.srh");     // local test variable (array)
  testHydro         = ~getDevices("Phys.hydro");   // local test variable (array)

  useDCSRH          = "staticSRH["      + testSRH[yes]   + "]";
  useHydro          = "electronsHydro[" + testHydro[yes] + "],electrons["
                       + testHydro[no] + "]," + "holesHydro["
                       + testHydro[yes] + "],holes["     + testHydro[no] + "]";

  driftDiffusionDC = "potential,electrons,holes," + useDCSRH;
  hydroDC           = "potential," + useHydro;
  hydroTrans        = useTransient + "potential," + useHydro + "," + useTransSRH;
}
```

The function `~getDevices("Phys.srh")` looks in all devices for the `Phys` section. When the section is found it searches for the `srh` keyword. The variable `testSRH` consists of a one-dimensional string array with two entries. The Entry `testHydro[yes]` holds all elements (segments) where the search was successful. All other elements (segments) are stored in the entry `testHydro[no]`.

An example for the `Phys` section reads:

```
Phys
{
  srh = "L*";
}
```

In this example all segments with names starting with a capital `L` are inserted in the `testSRH[yes]` array. Therefore the variable `useDCSRH` holds all segment names where SRH-Recombination should be used and whose name starts with a capital `L`. This information is used in the variable `driftDiffusionDC` which finally holds information about all additive components for the equation system on individual segments. In a similar way the variable `hydroDC` is built up where `useHydro` holds information about the equation components for the hydrodynamic model in each segment. The variable `useHydro` contains all additive terms of the energy flux equation (Energ flux term, Joule heat term, Relaxation term,...) for the segments where hydrodynamic simulation is performed.

Note: The difference between `Quantities` and `Models` is that quantities specify variables which can be solved by the simulator. This is why a single quantity represents a complete equation in the equation system. `Models` are the internal representation of additive terms in the implemented equations.

3.3.4 The `Iterate` Section

The `Iterate` section describes how to process a simulation. The section defines the equation systems and how they should be solved. The `Iterate` section can handle subsections which are evaluated recursively. Each subsection needs at least a `models` and a `quantities` keyword. In case of a transient simulation additional terms are required to describe the transient term of the differential equations.

In the example below the boolean keyword `transient` which is defined in the `Solve` section decides whether to simulate stationary or transient.

The keyword `quantities` defines the quantities of the equation system which should be solved. This is necessary to specify the sub-equation systems. In the example a mixed mode simulation is performed where `~Quantities.mixed` holds branch currents, node potentials etc.

The iteration block `HydroDD` solves the hydrodynamic equations with the drift-diffusion quantities. This means that we solve the Poisson equations and the continuity equations where the carrier temperature is fixed. The number of iterations in the `HydroDD` block depends on the `while` condition which depend on the evaluated norms. The GUMMEL/ASHER transformation can be turned on an off with the keyword `transform`.

```
Iterate
{
  MixedHydro
  {
    HydroDD
    {
      models     = ~if(~Solve.transient, ~Models.hydroTrans, ~Models.hydroDC);
      quantities = ~Quantities.mixed + "," + ~Quantities.driftDiffusion;
      while      = ~Extern.updateNorm     > ~Num.finalNorm &&
                   ~Extern.iterationCount < ~Num.iterationLimit/100;
      transform  = yes;
    }

    // Now we have a good initial guess for the actual hydro simulation.
    Hydro
    {
      models     = ~if(~Solve.transient, ~Models.hydroTrans, ~Models.hydroDC);
      quantities = ~Quantities.mixed + "," + ~Quantities.driftDiffusion +
                    "," + ~Quantities.hydro;
      dampScheme = "potential";
```

```
        ignoreDer  = ~if (~Extern.updateNorm > 1.0, "EleDEtm,HolDHtm", "");
        print      = ~if (~Extern.updateNorm > 1, "", "UseCompleteDer");
        CarrierTempBlock
        {
          quantities = "ElectronTemperature,HoleTemperature,
                        ElectronConcentration,HoleConcentration";
          dampScheme = "global";
          models = ~Models.hydro;
          limit = ~if(~Extern.updateNormPotLast < 1e-6, 100, 1);
          while = ~Extern.updateNorm > ~Num.finalNorm &&
                  ~Extern.iterationCount < limit;
        }
        while = ~Extern.updateNorm     > ~Num.finalNorm &&
                ~Extern.iterationCount < ~Num.iterationLimit;
      }
    }
  }
```

In the example the `HydroDD` block evaluates an initial guess for the `Hydro` block. The `Hydro` block contains the `CarrierTempBlock` as sub-block to solve the carrier temperature system until the `while` keyword evaluates to false. The `quantities` in the `Hydro` block now include all variables which should be solved in a regular hydrodynamic simulation. A damping scheme is used which depends on the potential update-norm.

The MINIMOS-NT input deck is able to directly control the derivatives of the JACOBIAN. If the update norm is higher than 1.0, the derivatives of the continuity equations to the carrier temperatures are neglected.

Each iteration block can contain a `print` keyword to write any desired information to the log file.

The `CarrierTempBlock` solves the carrier temperature equations, decoupled from the concentration and potential equations. This is done until the `while` keyword evaluates to false. The condition depends on the defined variable `limit` which is calculated using the last potential update-norm of the parent block. After solving the `CarrierTempBlock`, the `while` keyword of the `Hydro` block checks whether the simulation has finished.

3.4 Example

This section demonstrates some basic features of the MINIMOS-NT input deck which are used for the simulation of the output characteristic of a heterojunction bipolar transistor (HBT). The simulated device consists of three semiconductor segments and three contact segments. The material of the collector and the emitter segments is `Si` and `SiGe` for the base segment. Fig. 9 shows the geometry and the material composition of the simulated HBT.

The next subsection describes in detail some of the input deck entries used in the example. The complete input deck can be found in subsection 3.4.2.

3.4.1 Description of the Input Deck Used for Simulation of a HBT

Using the `#include` statement in the first line of the input deck the contents of the file `defaults.ipd` are inserted into the input deck, similar to including C-Header files into C-Source files.

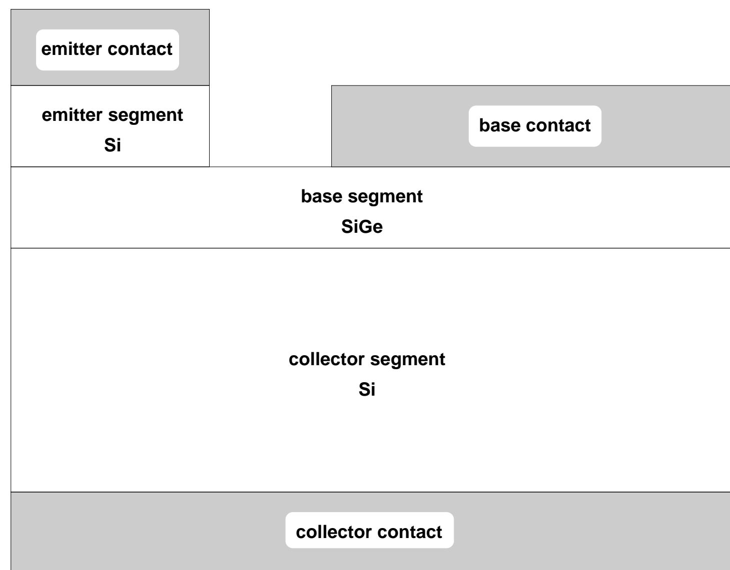


Figure 9: Geometry and material composition of the simulated HBT.

The most important input deck section for simulation of single devices is the `Solve` section. In this simple example the `Solve` section consists of only two user defined subsections. These are the `Device` and the `Iterate` subsections.

The `Device` section is copied by the statement `Device : ~Devices.Device`. Afterwards some variables of the `Device` section are assigned new values. For example the contact voltage for the emitter contact is specified by `EmitterContact = 0.0 V`. The next line `BaseContact = [0.7V, 1.0V, 0.1V]` specifies that the base contact voltage will be stepped from 0.7V to 1.0V in steps of 0.1V. The collector voltage is stepped from 0V to 2.0V in 0.1V steps. This results in two nested loops for calculating the bias points of the output characteristic. Fig. 10 shows the simulated output characteristic.

In the `Files` section the names of the `inputPBF` and the `outputPBF` are specified for reading the geometry and the doping information and writing resulting distributed quantities respectively.

In the `Phys` section physical models and parameters for the device are specified. With the line `EmitterContact { contact = "Voltage"; }` the segment named `EmitterContact` is defined to have a voltage boundary condition.

For the `SiGe` segment named `Base` special physical models (permittivity, band gap energy, etc.) for this material are specified.

The grid used for the simulation is specified in the `Grid` section by `inputGrid = "SimulationGrid";` and is read from the `inputPBF`. With the line `gridRefinement = 2.0;` the ratio of the grid spacing is limited to a value of 2. In case this ratio is larger than 2 additional grid lines are inserted.

For the `Iterate` subsection, which is the second subsection of the `Solve` section in this example, the values for the drift-diffusion iteration scheme are just copied from the `defaults.ipd` file.

In the `Num` section parameters for the linear and the nonlinear equation solver can be set. By `linearAccuracy = 1e-10;` the accuracy of the solution for the linear equation system is specified.

The form of output is controlled by the `Output` section. By specifying `printDeviceInfo = "*";` the contact voltages and currents are written to the log file for each bias point. For plotting the output

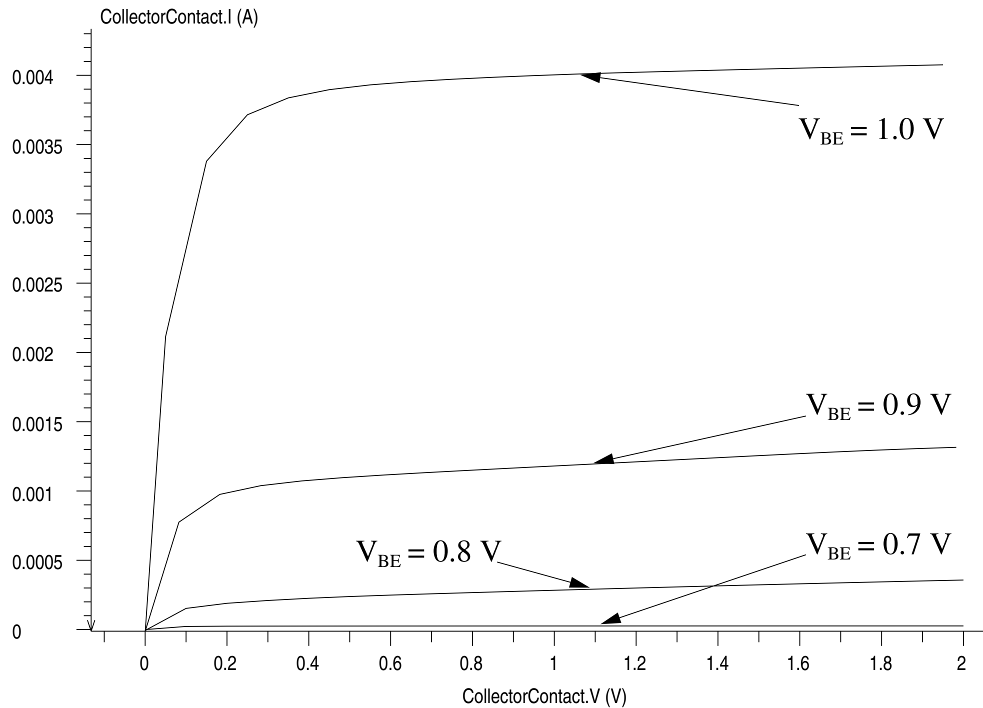


Figure 10: Simulated output characteristic.

characteristic the contact voltages and currents for all bias points are written to the file `hbt.crv` in the `curve` format.

3.4.2 The Input Deck for the Simulation of a HBT

```
#include <defaults.ipd>

Solve
{
  Device : ~Devices.Device
  {
     EmitterContact   = 0.0 V;
     BaseContact      = [0.7V, 1.0V, 0.1V];
     CollectorContact = [0V, 2V, 0.1V];

     Files
     {
        inputPBF  = "SiGeHBT";
        outputPBF = "SiGeHBT_out";
     }

     Phys
     {
        EmitterContact   { contact = "Voltage"; }
        BaseContact      { contact = "Voltage"; }
        CollectorContact { contact = "Voltage"; }

        Base
        {
           permittivity    = "Compound";
           bandGapEnergy   = "Ltm_SiGe";
```

```
            densityOfStates = "Ltm_SiGe";
            relativeMass    = "K300_SiGe";
         }
      }

      Grid
      {
         inputGrid = "SimulationGrid";
         gridRefinement = 2.0;
      }
   }
   Iterate : ~Iterate.DD;
}

Num
{
   linearAccuracy = 1e-10;
}

Output
{
   printDeviceInfo = "*";
   outputCrv = "hbt.crv";
}
```

3.5 Mixed-Mode Simulation with MINIMOS-NT

3.5.1 Introduction

Traditionally, TCAD based device design is restricted to the analysis of single devices, which are characterized in terms of IV curves and device parameters. In many cases, however, it would be desirable to simulate one or more devices in the circuit environment in which they are actually used. Typical candidates are power devices like DMOS transistors, RF devices such as HEMTs and HBTs, ESD protection devices, or MOSFETs operating under unusual conditions like low supply voltages. In many of these cases standard compact models for circuit simulation are not readily available or at least not straight forward to apply. For this purpose MINIMOS-NT has been equipped with an extensive mixed-mode capability, which allows the user to simulate circuits that contain one or more 'distributed' devices, i.e., devices which are simulated by solving the semiconductor equations rather than evaluating a compact model.

The distributed devices are handled in a two-step manner. First, for each type of device an entry is made in the devices' section (in this case `MyDevices)` just as in the case of a normal device simulation. Second, in the circuits section one or more instances of a device can be created by copying an entry in the devices' section. Internally one logical working PIF file is created for each device instance to store the distributed quantities.

3.5.2 Using Mixed-Mode

The mixed-mode functionality is accessed through the MINIMOS-NT input deck by storing the circuit in the `Circuit` section and by selecting the appropriate iteration scheme in the `Solve` section. Currently, there are two predefined schemes for mixed mode: `MixedDD` and `MixedHydro`. Of course, it is still possible to override the details for individual devices or segments, which is useful especially in the case of mixed-mode hydrodynamic simulations with more than one heterojunction device. Subcircuits are

supported in a flexible way so that certain standard circuits like, e.g., measuring or benchmark circuits can be put into libraries. The following is the complete mixed-mode input deck of a 5-stage ring oscillator circuit:

```
#include <defaults.ipd>

Mydevices
{
  NMOS : ~Devices.Device
  {
     vScal =  1.0;
     iScal =  1.0;

     Files
     {
        inputPBF  = "nmos";
        outputPBF = "ro.pbf";
        output = "";
     }
     Phys
     {
        Source { contact = "Voltage"; }
        Drain  { contact = "Voltage"; }
        Bulk   { contact = "Voltage"; }
        Gate   { contact = "Voltage"; Ew = -0.55 eV; }
        Semiconductor { epsr = 11.7; }
        GateInsulator { epsr = 3.9; }
     }
     Grid
     {
        inputGrid = "Gr_1";
     }
  }

  PMOS : ~Mydevices.NMOS
  {
     vScal = -1.0;
     iScal = -1.0;
  }
}

Subcircuits
{
  Inv
  {
     Load = 5.0fF;
     in  = "";
     out = "";
     Ma : ~Mydevices.NMOS {Source = "gnd";Gate = ^in;Bulk = "gnd";Drain = ^out;}
     Mb : ~Mydevices.PMOS {Source = "Vdd";Gate = ^in;Bulk = "Vdd";Drain = ^out;}
     CL : ~Devices.C  {N1 = ^out;N2 = "gnd";C = ^Load;}
  }
}

Vdd = 1.5V; // supply voltage

Circuit
{
  Vdd  : ~Devices.V { P = "Vdd";   M = "gnd"; V0 = ~Vdd;}

  Inv1 : ~Subcircuits.Inv        { in = "N1"; out = "N2"; }
```

```
  Inv2 : ~Subcircuits.Inv        { in = "N2"; out = "N3"; }
  Inv3 : ~Subcircuits.Inv        { in = "N3"; out = "N4"; }
  Inv4 : ~Subcircuits.Inv        { in = "N4"; out = "N5"; }
  Inv5 : ~Subcircuits.Inv        { in = "N5"; out = "N1"; }
}

Solve
{
  transient = yes;
  startTime = 0.0 s;
  endTime   = 0.2e-9 s;
  minStepSize =  2e-12 s;
  maxStepSize =  minStepSize;

  FixNode // start-up condition
  {
     N1 = ~Vdd;
     N2 = 0;
     N3 = ~Vdd;
     N4 = 0;
     N5 = ~Vdd;
  }

  Iterate : ~Iterate.MixedDD
}

Num
{
  finalNorm = 1e-0;
  directSolver = no;
  linearAccuracy = ~if(~Extern.t == 0s, 1e-9, 1e-5); //cute eh?
  maxFillin = 50+30*(~Extern.t == 0s);
}
```

The parameters `iScal` and `vScal` are used for current and voltage scaling to account for different device widths or to easily emulate complementary devices like a PMOS transistor. This is done by electrically mirroring the NMOS transistor by setting `iScal = -1` and `vScal = -1` (note the use of the copy operator '`:`' in the devices' section). This technique allows, e.g., to optimize a single device based on simulated circuit performance data.

Segments of a distributed device like, e.g., contacts, are referenced in the `Phys` section by their segment names, which are stored in the `SegmentName` attributes in the PIF file containing the device structure. This mechanism is used to override default values for certain segments or to specify contact types.

The circuit is made up of five inverters `Inv` which are defined in the `Subcircuits` section. Note, that more than one section with arbitrary names can contain subcircuits. Circuits and subcircuits may also be parameterized (like, e.g., the `Load` parameter of the `Inv` subcircuit).

The `Solve` section, which controls the simulation, also contains the settings for the start-up condition, which is necessary in this case. In the `Num` section, which controls the simulation numerics, the accuracy of the linear solver is set to a small value for the initial DC solution and is made larger for the transient simulation. Fig. 11 shows the result of the simulation. This simulation and simulations of an inverter were used to verify algorithms of an efficient VLSI performance metric [22].

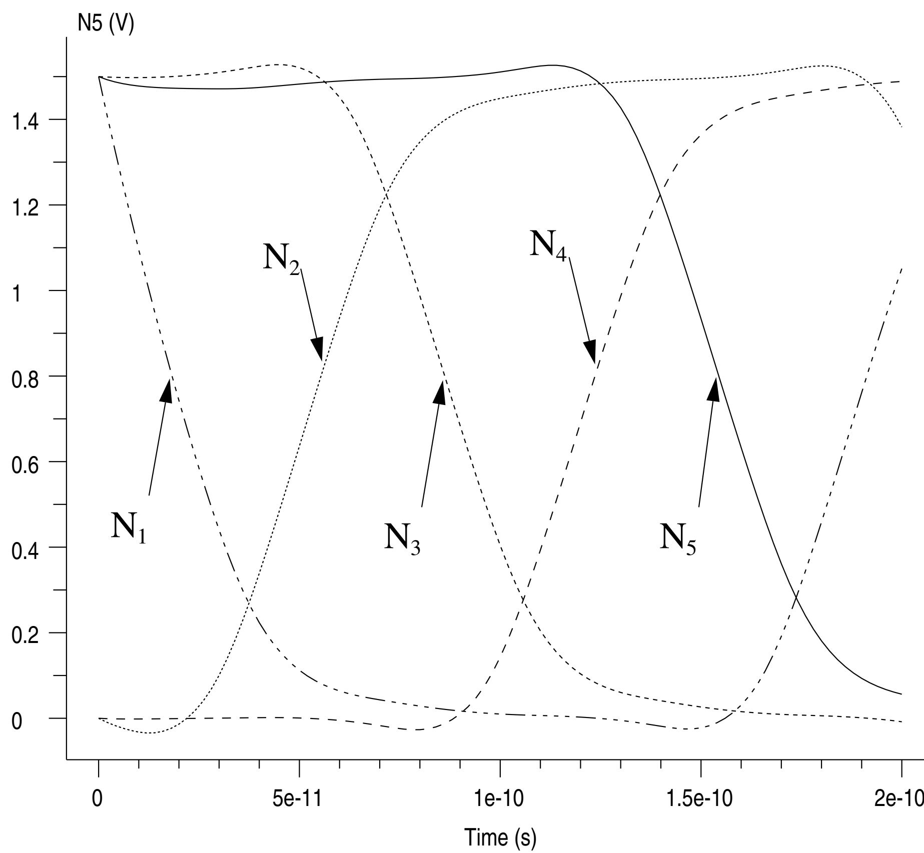


Figure 11: Mixed-mode simulation of a 5-stage ring oscillator circuit with 0.18μm NMOS transistors and electrically mirrored PMOS transistors

4 Optimization of Analytical Doping Profiles

4.1 Introduction

Advanced analysis features are implemented in the Vienna Integrated System for TCAD Applications (VISTA) simulation environment. These functionalities support automatic experiment generation (Design of Experiments), model fitting (Response Surface Methodology), optimization and calibration. They interact with the core functionality of the framework dealing with the simulation of the manufacturing process and electrical characterization of semiconductor devices. The presented example deals with the optimization of analytical doping profiles of MOS transistors.

4.2 Vienna Integrated System for TCAD Applications

The VISTA framework and its modules form a programmable simulation environment for TCAD applications. The framework is based on the VLISP interpreter [23]. This object oriented LISP language is similar to the popular JAVA [24] programming language with class inheritance and polymorphism. The interpreter provides also interfaces to the MOTIF widget-set [25] and some additional widgets.

The advantage of VLISP compared to Tcl/Tk [26] is the object system and the large number of sophisticated basis classes for data manipulation, user interface generation, and for handling asynchronous events.

As an open framework a large number of process and device simulators from different vendors like TSUPREM [27] and MEDICI [28] from TMA and ATHENA [29] from SILVACO are accessible. Integrated wrapper programs are used to convert the input files to the corresponding data format of the simulator.

The distribution of the workload is done by job farming. For an efficient use of the resources the framework polls periodically the load of the available hosts and compares it with the allowed maximum number of jobs for automatic load balancing.

The functions of the framework are designed to operate also without the user interface. By defining a small sequence using the VLISP extension language, the framework can be used for off-line execution.

4.3 Optimization

For optimizing device performance parameters over a given input variable space, a constrained optimizer has been integrated. The use of a constrained optimization algorithm is important to prevent possible unphysical optima. The integrated nonlinear constrained optimizer uses an augmented Lagrangian method [30]. It minimizes the target function which can be assembled out of input and output values. The gradient is calculated by evaluating adaptive finite differences.

For calibration and parameter fitting tasks an optimizer based on the Levenberg Marquardt Algorithm [31] is also integrated in the framework.

4.4 Application

In this section another successful TCAD application based on the presented framework is described. The doping profile for a specified device structure is optimized to achieve certain performance improvements.

Analytical doping profiles are generated by a template based device generator. Parameters are global technology values like the geometry and doping level of a layer. These parameters are used as control parameters for the optimization process.

In this example, an NMOS device with $0.25\mu\mathrm{m}$ geometry gate length and $5\mathrm{nm}$ gate oxide thickness is optimized for $1.5\mathrm{V}$ supply voltage. The analytical doping profile consists of a retrograde well and a channel implant, both generated with Gauß-functions. Control parameters are the depth, the deviation, and the doping level of the channel implant and the retrograde profile.

The optimization target is defined for achieving maximum on-current. Without a constraint, this optimization would result in a decrease of the threshold voltage, only. The off-current would be drastically increased because of its exponential dependence on threshold voltage in the sub-threshold region [32]. A large off-current leads to high system standby power which does not meet the requirements for future MOSFET technology [33]. Therefore, the off-current is entered as a constraint for the optimization process and kept at a constant value. Two device simulation steps have to be performed to extract the two current-values. To increase speed of the optimization, these two steps are done in parallel.

A uniformly doped device is used as the initial device, the on-current is improved drastically by the presented optimization process. Together with the flexible framework features, tasks like reverse engineering of doping profiles can be realized.

In this example the parameter space is very large and the simulation flow consists only of the generation step and the electrical characterization. Therefore, a direct optimization of the device is performed.

Figure 12 shows the analytical acceptor and donor doping profile of the optimized transistor with the source and drain regions and the channel implant.

The on-current of the optimized transistor ($I_{on} = 2.98\ 10^{-4}\mathrm{A}$) was increased by 20% compared with a uniformly doped device ($I_{on} = 2.49\ 10^{-4}\mathrm{A}$) with the same off-current I_{off}.

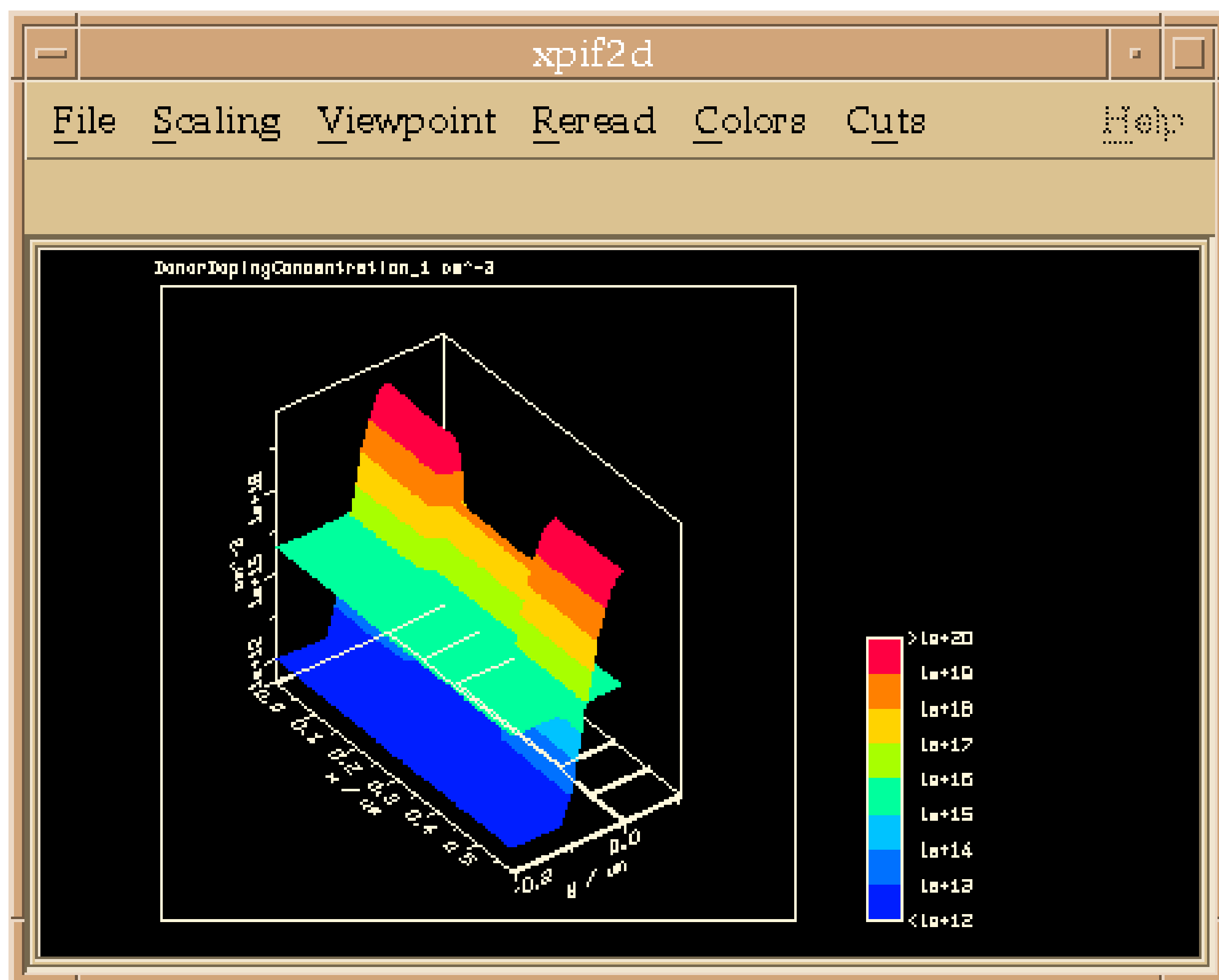


Figure 12: Analytical acceptor and donor doping profile of the optimized MOS transistor.

References

[1] M. Born and E. Wolf. *Principles of Optics*. Pergamon Press, 6th edition, 1993.

[2] M.S.C. Yeung. Modeling High Numerical Aperture Optical Lithography. In *Proc.SPIE Optical/Laser Microlithography*, volume 922, pp 149–167, 1988.

[3] K.K.H. Toh and A.R. Neureuther. Identifying and Monitoring Effects of Lens Aberrations in Projection Printing. In *Proc.SPIE Optical Microlithography VI*, volume 722, pp 202–209, 1987.

[4] H. Kirchauer. *Photolithography Simulation*. PhD thesis, TU Vienna, Institute for Microelectronics, *in preparation* (January 1998).

[5] C.E. Shannon. Communication in the Presence of Noise. *Proceedings of the IRE*, 37:10–21, 1949.

[6] W.H. Press, S.A. Teukolsk, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 2nd edition, 1995.

[7] Ch. Überhuber. *Computer-Numerik*, volume 2. Springer, Berlin, 1995. (in German).

[8] P.N. Swarztrauber. Vectorizing the FFTs. In G. Rodrigue, editor, *Parallel Computations*, pp 51–83. Academic Press, 1982.

[9] N. Bohr. On the Theory of the Decrease of Velocity of Moving Electrified Particles on Passing Through Matter. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 6:10–31, 1913.

[10] W. Lenz. Über die Anwendbarkeit der statistischen Methode auf Ionengitter. *Z.Phys.*, pp 713–721, 1932.

[11] A. Sommerfeld. Asymptotische Integration der Differentialgleichung des Thomas-Fermischen Atoms. *Z.Phys.*, 78:283–309, 1932.

[12] G. Molière. Theorie der Streuung schneller geladener Teilchen I: Einzelstreuung am abgeschirmten Coulomb-Feld. *Z.Naturforschung*, 2a:133–145, 1947.

[13] J.F. Ziegler, J.P. Biersack, and U. Littmark. *The Stopping and Range of Ions in Solids*, volume 1 of *The Stopping and Ranges of Ions in Matter*. Pergamon Press, New York, 1985.

[14] J. Lindhard. On the Properties of a Gas of Charged Particles. *Mat.Fys.Medd.Dan.Vid.Selsk.*, 28(8):57, 1954.

[15] O.S. Oen and M.T. Robinson. Computer Studies of the Reflection of Light Ions from Solids. *Nucl.Instr.Meth.*, 132:647–653, 1976.

[16] G. Hobler, H. Pötzl, L. Palmetshofer, R. Schork, J. Lorenz, C. Tian, S. Gara, and G. Stingeder. An Empirical Model for the Electronic Stopping of Boron in Silicon. *COMPEL*, 10(4):323–330, 1991.

[17] G. Hobler and H.W. Pötzl. Electronic Stopping of Channeled Ions in Silicon. In *Mat.Res.Soc.Symp.Proc.*, volume 279, pp 165–170, 1993.

[18] G. Hobler, A. Simionescu, L. Palmetshofer, F. Jahnel, R. von Criegern, C. Tian, and G. Stingeder. Verification of Models for the Simulation of Boron Implantation into Crystalline Silicon. In Ehrstein et al. [34], pp 14.1–14.9.

[19] G. Hobler, A. Simionescu, L. Palmetshofer, C. Tian, and G. Stingeder. Boron Channeling Implantations in Silicon: Modeling of Electronic Stopping and Damage Accumulation. *J.Appl.Phys.*, 77(8):3697–3703, 1995.

[20] M.J. Norgett, M.T. Robinson, and I.M. Torrens. A Proposed Method of Calculating Displacement Dose Rates. *Nucl.Eng.Des.*, 33:50–54, 1975.

[21] G.H. Kinchin and R.S. Pease. The Displacement of Atoms in Solids by Radiation. *Reports on Progress in Physics*, 18:1–51, 1955.

[22] G. Schrom, V. De, and S. Selberherr. VLSI Performance Metric Based on Minimum TCAD Simulations. In *International Conference on Simulation of Semiconductor Processes and Devices*, pp 25–28, Cambridge, Massachusetts, 1997.

[23] Institut für Mikroelektronik, Technische Universität Wien, Austria. *VISTA Documentation 1.3-1, VLISP Manual*, 1996.

[24] K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 1996.

[25] D. Heller and P.M. Ferguson. *Motif Programming Manual*, volume Six A. O'Reilly & Associates, 1994.

[26] B. Welch. *Practical Programming in Tcl and Tk*. Prentice-Hall, 1995.

[27] Technology Modeling Associates, Inc., Palo Alto, CA. *TMA TSUPREM-4, Two-Dimensional Process Simulation Program, Version 6.2*, 1995.

[28] Technology Modeling Associates, Inc., Palo Alto, CA. *TMA MEDICI, Two-Dimensional Device Simulation Program, Version 2.0*, 1994.

[29] Silvaco. *ATHENA: 2D Process Simulation Framework*, 1993. User's Manual.

[30] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, 1995.

[31] N. Khalil, J. Faricelli, C.L. Huang, and S. Selberherr. Two-Dimensional Dopant Profiling of Submicron MOSFET's Using Nonlinear Least Squares Inverse Modeling. In Ehrstein et al. [34], pp 6.1–6.9.

[32] Y.P. Tsividis. *Operation and Modeling of the MOS Transistor*. McGraw-Hill, 1987.

[33] P.V. Voorde. MOSFET Scaling into the Future. *Hewlett-Packard Journal*, 1997.

[34] J. Ehrstein, R. Mathur, and G. McGuire, editors. *Proc. 3rd Int. Workshop on Measurement and Characterization of Ultra-Shallow Doping Profiles in Semiconductors*, 1995.