Numerical calculation of the charge and potential distribution in a metal-oxide semiconductor diode

Roni Koitermaa *

December 8, 2021

Abstract

The charge density $\rho(z)$ and potential $\phi(z)$ are numerically calculated in a metal-oxide semiconductor (MOS) diode. The Poisson equation is solved using the finite difference method (FDM) to calculate $\phi(z)$. Charge carrier concentrations are obtained from Fermi-Dirac integrals and used to calculate the charge density $\rho(z)$ in p-doped silicon. A solution satisfying all equations is found by an iterative damping algorithm.

1 Introduction

Understanding of metal-oxide semiconductor (MOS) devices is crucial for the development of modern electronics, as it is the technology used in transistors (MOSFETs) [1]. One of the simplest MOS devices is the diode, which consists of a silicon bulk substrate, an oxide layer and a metal electrode or gate. This configuration allows current to flow easily in one direction but not in the other direction. For p-type silicon the majority charge carriers are holes, so when we have a positive voltage at the gate, a depletion layer forms. For a large positive voltage, minority negative charge carriers are attracted towards the gate and we get inversion. For a negative gate voltage, positive charge carriers are attracted towards the gate and current can't flow. These properties make the MOS structure function as a diode.

The diode can be modelled as a simple stack of the three materials [2]. To calculate the charge and potential distributions, we need to solve the associated equations in this domain. The dimensionality of the problem can be reduced by using symmetry and assuming that the diode is infinitely large. This allows us to only consider the z direction, where the diode has finite thickness. The metal layer can be excluded from our calculations as it is a conductor and the potential is constant. For the insulating oxide layer we have $\rho = 0$ as there are no free charge carriers, but the $\rho(z)$ in the substrate is unknown. We could calculate $\rho(z)$ using the potential $\phi(z)$, but the potential is also unknown. We have coupled differential equations for these that can be solved numerically.

2 Methods

Our problem can be divided into two parts: solution of the Poisson equation and solution of the charge density distribution. For the former task, the finite difference method can be used [3], and for the latter we will use an iterative procedure with damping [2].

The 1D Poisson equation

$$\phi''(z) = -\frac{\rho(z)}{\epsilon(z)} = f(z) \tag{1}$$

can be solved numerically using the finite difference method, where derivatives are approximated using Taylor series [3]. There are many ways to discretize this equation, and we will use the simple three point stencil that can be written as the tridiagonal matrix [4]

$$\mathrm{A} = \begin{pmatrix} -2/h^2 & 1/h^2 & 0 & \cdots & 0 \\ 1/h^2 & \ddots & & & \vdots \\ 0 & & -2/h^2 & & 0 \\ \vdots & & & \ddots & 1/h^2 \\ 0 & \cdots & 0 & 1/h^2 & -2/h^2 \end{pmatrix},$$

*roni.koitermaa@helsinki.fi

where $h = (b-a)/(N-1)$ is the step size for N points. For the right hand side we have (using f_i, where $i = 0, 1, \ldots, N-1$)

$$\boldsymbol{b} = \begin{pmatrix} f_1 - \phi(a)/h^2 \\ f_2 \\ \vdots \\ f_{N-3} \\ f_{N-2} - \phi(b)/h^2 \end{pmatrix},$$

where $\phi(a)$ and $\phi(b)$ are Dirichlet boundary conditions for the solution domain $[a, b]$. Now we can formulate the problem as a matrix equation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$. The matrix A is diagonally dominant, so the equation is easy to solve numerically.

In order to calculate the charge carrier concentrations, we need to perform the Fermi-Dirac integrals. We can transform the integrals as (Fermi level $E_F = 0$) [5]

$$n(z) = \frac{\sqrt{2} m_e^{3/2}}{\pi^2 \hbar^3} \int_{E_c(z)}^{\infty} \frac{\sqrt{E - E_c(z)}}{1 + \exp(E/k_B T)} \, \mathrm{d}E, \quad \Big| \; \varepsilon = \frac{E - E_c(z)}{k_B T}$$

$$= \frac{\sqrt{2}(m_e k_B T)^{3/2}}{\pi^2 \hbar^3} \int_{E_c(z)}^{\infty} \frac{\varepsilon^{1/2}}{1 + \exp(\varepsilon + E_c(z)/k_B T)} \, \mathrm{d}\varepsilon = \frac{\sqrt{2}(m_e k_B T)^{3/2}}{\pi^2 \hbar^3} \frac{\sqrt{\pi}}{2} F_{1/2}(-E_c(z)/k_B T), \tag{2}$$

$$p(z) = \frac{\sqrt{2} m_e^{3/2}}{\pi^2 \hbar^3} \int_{-\infty}^{E_v(z)} \frac{\sqrt{E_v(z) - E}}{1 + \exp(E/k_B T)} \, \mathrm{d}E, \quad \Big| \; \varepsilon = \frac{E_v(z) - E}{k_B T}$$

$$= -\frac{\sqrt{2}(m_e k_B T)^{3/2}}{\pi^2 \hbar^3} \int_{-\infty}^{0} \frac{\varepsilon^{1/2}}{1 + \exp(E_v(z)/k_B T - \varepsilon)} \, \mathrm{d}\varepsilon = \frac{\sqrt{2}(m_e k_B T)^{3/2}}{\pi^2 \hbar^3} \frac{\sqrt{\pi}}{2} F_{1/2}(E_v(z)/k_B T). \tag{3}$$

The function $F_j(z)$ is the Fermi-Dirac integral with $j = 1/2$ that has been implemented in GSL. The valence and conduction band energies are defined as [2]

$$E_v(z) = -k_B T \ln \left[\frac{2}{N_A} \left(\frac{m_h k_B T}{2\pi\hbar^2} \right)^{3/2} \right] - e\phi(z),$$

$$E_c(z) = E_v(z) + E_g,$$

where the effective mass m_h and band gap E_g depend on the material. The doping concentration N_A is assumed to be constant.

The charge density distribution can now be calculated using [2]

$$\rho(z) = e\left(-N_A - n(z) + p(z)\right). \tag{4}$$

Using this directly as the RHS of the Poisson equation results in instability, so this is damped by a factor ω by combining with the previous iteration value $\rho^{(i-1)}(z)$:

$$\rho^{(i)}(z) = \omega\rho(z) + (1-\omega)\rho^{(i-1)}(z).$$

We can monitor the convergence of our algorithm by calculating a residual $|\rho^{(i)}(z) - \rho^{(i-1)}|$ and comparing it to a convergence criterion C. The whole algorithm can be summarized as [2]:

1. Start with an initial guess for $\rho^{(i)}(z)$,
2. Solve Poisson equation using $\rho^{(i)}(z)$,
3. Compute Fermi-Dirac integrals and calculate new $\rho^{(i)}$ using damping,
4. If not converged i.e. residual $|\rho^{(i)}(z) - \rho^{(i-1)}(z)| > C|\rho^{(i)}(z)|$, jump to 2.

3 Implementation

The procedure described in section 2 was implemented in C++ using LAPACK and GSL. The software consists of two main parts: the Poisson solver in file `src/solver.cpp` and the main solver that computes the charge density in `src/sim.cpp`. The program reads in an input file e.g. `diode.in` that contains the properties of the problem

Constant	k_B (eV K^{-1})	m_e (eV fs^2 nm^{-2})	$\hbar$ (eV fs)	ϵ_0 (e nm^{-1} V^{-1})	e
Value	8.6173×10^{-5}	5.6856	6.5821×10^{-1}	5.5263×10^{-2}	1

Table 1: Physical constants.

such as the number of elements and material properties. This is used to configure the main solver and then the Poisson solver. The solution domain $[a, b]$ is divided into $N-1$ equally spaced elements using N points. The Poisson equation is solved iteratively until the desired convergence criterion is reached, at which point the solution is printed to an output file e.g. `diode.out`. This file contains the charge density, potential and charge carrier concentration at each point, and it can be be plotted by using the Python script `plot.py`.

The Poisson solver is configured by setting the boundary conditions and RHS $\boldsymbol{b}$. Then the stencil matrix A is constructed and the matrix equation is ready to be solved. The size of this system is $N-2$. The Poisson solver uses LAPACK to solve the matrix equation by calling `dgesv_()`. The solution $\phi(z)$ is placed in the RHS array. To evaluate the potential, the method `eval()` is used. This method uses linear interpolation to compute values of $\phi(z)$ between the grid points if required.

The current density is calculated from the charge carrier concentrations. Domain-specific constants in our calculations such as $\epsilon(z)$ are evaluated using methods such as `getEps(z)`, which pick the appropriate constant based on the position z. The Fermi-Dirac integrals 2 and 3 can be evaluated using the function `gsl_sf_fermi_dirac_half()`. This function is part of the special function library of GSL [1]. According to the GSL source code, these integrals are evaluated using fitted Chebyshev polynomials, which is likely faster (and potentially more accurate) than direct numerical integration.

The code uses the following units: nm (nanometer), fs (femtosecond), eV (electronvolt), V (volt), K (kelvin) and e (elementary charge). Other units and constants are expressed primarily using these units. For example, mass is $[m] = \text{eV fs}^2\ \text{nm}^{-2}$. These units were chosen as they are close in magnitude and relevant to the scale of the problem. The definitions of the physical constants used can be found in table 1.

Calculation of the inversion voltage V_I is implemented by increasing V_G by some V_{step}. When the inversion voltage calculation is enabled, at convergence the voltage is increased incrementally until the charge carrier concentrations reach $n(z) > p(z)$ at some point on the bulk $z \leq t_B$. At this point the voltage has reached V_I and the program terminates.

4 Using the code

Compilation

The code is compiled using a makefile, and it can be compiled by running either `make` or `make release`. The code is compiled with GCC (`g++`) using the C++11 standard. Compilation requires the libraries `lapack` (Linear Algebra Package), `gsl` (GNU Scientific Library), `gslcblas` (BLAS required by GSL) and `m` (C math libraries). The development versions of these libraries are available in most distributions. The makefile produces the executable `difidi`. The makefile target `clean` removes the executable and object files.

Running

Input and output for the program is done using files provided as command line arguments. The executable can be run using e.g. `./difidi diode.in diode.out` or `./run.sh`. The script `all.sh` compiles and runs the code, and plots the results afterwards.

The configurations in `diode.in` have been documented using comments. Variable names in this file match those in the `Sim` class. The inversion voltage calculation is enabled if `Vstep` is larger than zero (or set). The output file `diode.out` has the format `z rho(z) phi(z) n(z) p(z)`.

Plotting is done from the file `diode.out` using the Python script `plot.py`, which can be run using `python3 plot.py`. The plotting code uses NumPy and Matplotlib/Pyplot. The plots are saved as PDFs in the base directory.

[1] `https://www.gnu.org/software/gsl/doc/html/specfunc.html#fermi-dirac-function`

5 Results

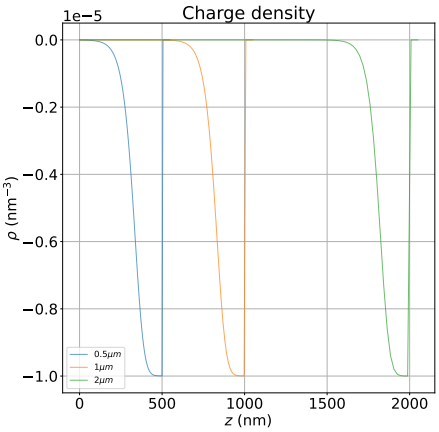

Figure 1: Charge density for different t_B.

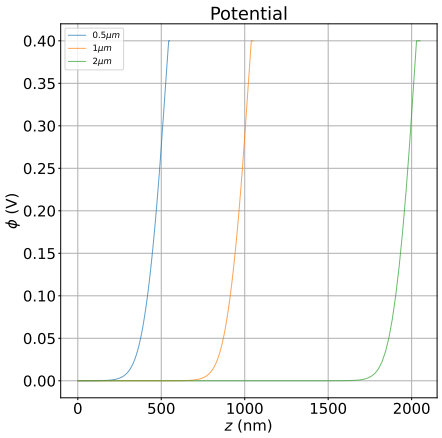

Figure 2: Potential for different t_B.

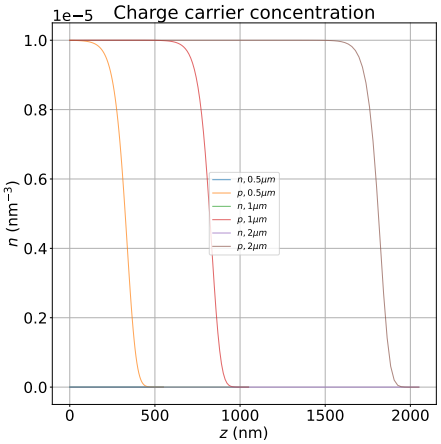

Figure 3: Charge carrier concentration for different t_B.

The literature gives the following values for the properties of Si and SiO_2: $E_{g,\mathrm{Si}} = 1.12$ eV [1], $E_{g,\mathrm{SiO_2}} = 9.$ eV [1], $\epsilon_{\mathrm{r,\ Si}} = 11.9$ [1], $\epsilon_{\mathrm{r,\ SiO_2}} = 3.9$ [1], $m_e = 1.08$ [6], $m_h = 0.81$ [6]. Other constant values across all runs are the doping concentration $N_A = 10^{-5}$ nm^{-3}, oxide thickness $t_{\mathrm{ox}} = 50$ nm, (room) temperature $T = 300$ K and initial guess $\rho_0 = 0$.

We use a damping factor of $\omega = 0.001$ with a convergence criterion $C = 10^{-2}$. Number of elements $N = 100$ appears to be sufficient for all runs and reasonably fast using the general solver from LAPACK. This is not optimal, since we have a highly diagonally dominant matrix. We could disregard all the zeros which form the majority of the matrix. Doing this would reduce solution time and memory requirements, but the scale of the problem is small enough that optimizations are not crucial. Using these settings the solver converges in roughly 10000 iterations.

The charge density, potential and charge carrier distributions have been plotted in figures 1, 2 and 3 for a gate voltage of $V_G = 0.4$ V and various substrate thicknesses. The input files for these runs can be found in `run/diode0.5mic.in`, `run/diode1mic.in` and `run/diode2mic.in`. The output files are named similarly.

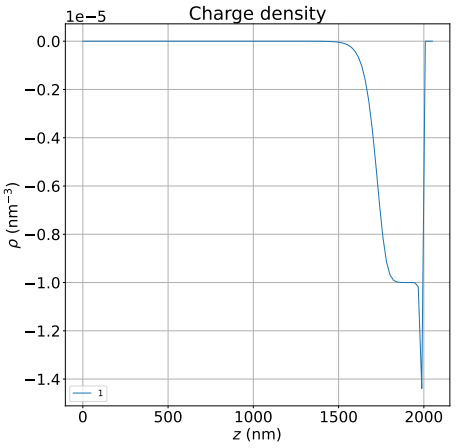

Figure 4: Charge density at $V_G = 0.8$ V.

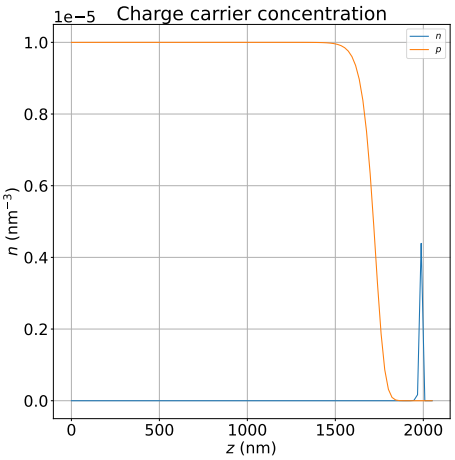

Figure 5: Charge carrier concentration at $V_G = 0.8$ V.

There are multiple sources of numerical error in our calculations. The convergence criterion C determines the accuracy of the iterative solution of $\rho(z)$. If we don't have a small enough damping factor, the solution oscillates and we also get error. Discretization size N influences the step size h and therefore the error of the Poisson solver, which is $\mathcal{O}(h^2)$ for the three point stencil [3]. There is also the error of the matrix solver, which depends on how this is implemented in LAPACK. For iterative solvers e.g. Gauss-Seidel there would be some kind of convergence criterion.

References

[1] Sze, S. M. & Ng, K. K. Physics of Semiconductor Devices. Wiley, 3rd ed., 2007.

[2] MATR322 Numerical methods in scientific computing 2021, Final Project. University of Helsinki, 2021.

[3] Chen, L. Finite difference methods for Poisson equation. University of California Irvine, 2020. Available at: `https://www.math.uci.edu/~chenlong/226/FDM.pdf`.

[4] Reisch, B. & Lehtola, S. Numerical Methods in Scientific Computing, 12. Differential equations. University of Helsinki, 2021.

[5] Kim, R., Wang, X. & Lundstrom, M. Notes on Fermi-Dirac integrals. arXiv:0811.0116.

[6] Van Zeghbroeck, B. J. Effective mass in semiconductors. University of Colorado Boulder, 1997. Available at: `https://ecee.colorado.edu/~bart/book/effmass.htm`.