NUMERICAL MODELING OF DOPANT DIFFUSION IN SEMICONDUCTORS

by

Makelle Faith Wininger Forson

A senior thesis submitted to the faculty of

Brigham Young University - Idaho

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics

Brigham Young University - Idaho

April 2026

BRIGHAM YOUNG UNIVERSITY - IDAHO

DEPARTMENT APPROVAL

of a senior thesis submitted by

Makelle Faith Wininger Forson

This thesis has been reviewed by the research advisor, research coordinator, and department chair and has been found to be satisfactory.

Date	Jon Paul Johnson, Advisor
Date	Matt Zachreson, Committee Member
Date	Joe Hill, Committee Member
Date	R. Todd Lines, Thesis Coordinator
Date	Evan Hansen, Department Chair

ABSTRACT

NUMERICAL MODELING OF DOPANT DIFFUSION IN SEMICONDUCTORS

Makelle Faith Wininger Forson

Department of Physics

Bachelor of Science

The fabrication of metal-oxide-semiconductor field-effect transistors (MOSFETs) requires precise control of dopant diffusion in silicon wafers. This project investigates the diffusion of impurities into a silicon wafer through computational modeling and experimental fabrication, to support the Brigham Young University–Idaho (BYU-I) Semiconductor Research Group in their efforts to develop a functioning MOSFET.

An interactive Python program was developed to calculate the initial and final characteristics of a silicon wafer after n-type or p-type dopants are driven into the substrate. The program determines key diffusion parameters—including dopant concentration, diffusion depth, temperature, and time—and generates dopant concentration profiles for the wafer. The computational results were verified through comparison with established semiconductor diffusion data.

Complementary experimental work established procedures for preparing a phosphorus spin-on dopant solution, removing silicon dioxide from silicon wafers, driving phosphorus dopants into the substrate, and measuring the resulting electrical properties. Through this work, an originally heavily doped p-type wafer was doped to become normally n-type. Together, the computational and experimental results provided foundational tools for BYU–Idaho to fine-tune their semiconductor fabrication process.

ACKNOWLEDGMENTS

Special thanks to my advisor, Dr. Jon Paul Johnson, and the BYU-Idaho physics department head, Dr. Evan Hansen, for their counsel and insights throughout this project. Also, thank you to the Idaho Technology Council in awarding the TechBridge Access Grant to me. But most importantly, I couldn't have completed this project without my loving, patient, and supportive husband, Brigham Forson, for whom I am forever grateful!

Contents

Table of Contents viii

List of Figures x

1 Introduction 1
- 1.1 Mechanics . . . 2
- 1.2 Impurity Doping . . . 3
  - 1.2.1 Diffusion . . . 5
  - 1.2.2 Ion-Implantation . . . 5
- 1.3 This Research . . . 6

2 Procedure 8
- 2.1 Computational Work . . . 8
  - 2.1.1 Required Inputs . . . 8
  - 2.1.2 Impurity Calculation . . . 9
  - 2.1.3 Diffusion Calculation . . . 13
- 2.2 Experimental Work . . . 18
  - 2.2.1 Spin-On Dopant . . . 19
  - 2.2.2 Drive-in Process . . . 20
  - 2.2.3 Wafer Characterization . . . 22

3 Results 25
- 3.1 Computational Results . . . 25
- 3.2 Experimental Results . . . 26

4 Conclusion and Future Work 29

Bibliography 30

A Main Code 32

B User Input Code 37

C Initial Concentration Code 47

D Diffusion Code **53**

E Plotting Code **63**

F Spin-On Dopant SOP **68**

List of Figures

1.1 Metal-oxide-semiconductor field-effect transistor (MOSFET) diagram. Reproduced from [1]. 2
1.2 Silicon lattice with Boron (P-type) or Phosphorus (N-type) dopant atoms 4
1.3 Impurity Doping Techniques: (a) Diffusion and (b) Ion-implantation. Reproduced from [1]. 4
1.4 Rapid Thermal Annealing Process (RTA) for Ion Implanted Silicon Wafers. Reproduced from [1]. 7

2.1 User Interface for Initial Wafer Input With Example Inputs 9
2.2 Doping Levels 12
2.3 Program Produced Resistivity vs. Doping Concentration Plot with P- and N-type Lines 12
2.4 User Interface for Diffusion Calculation With Example Inputs 13
2.5 Program Produced $C(x,t)$ Plots for P- to N-type doping and N- to P-type doping 19
2.6 Pre-deposition of Dopant Solution on a Silicon Wafer Diagram. Reproduced from [2]. 21
2.7 Treated Wafer after Application of Spin-on Dopant Solution and Before Impurity Drive-in Process 22
2.8 Treated Wafer After Impurity Drive-In Process, then Treated With Armour Etch to Remove SiO_2. 23
2.9 Four-Point Probe Diagram and Experimental Setup 24

Chapter 1

Introduction

Semiconductors are contained in nearly all our electronics and are the backbone of new technological advances[1,3]. They offer high performance with low power consumption in compact spaces in a variety of devices. Some of these devices include light emitting diodes (LED's), transistors, solar cells, and metal-oxide-semiconductor field-effect transistors (MOSFET's). The most important device used for advanced integrated circuits is the MOSFET, shown in Figure 1.1, which was developed in 1960[1]. In 2012, it constituted about 95% of the semiconductor market[1]. The MOSFET device can serve as the basis for the most advanced integrated circuit chips, containing over one trillion devices[1]. As more and more devices depend upon semiconductors in general, it is important to understand their mechanisms and how to fabricate them. Now, it is increasingly more feasible to produce semiconductors outside of major corporations.

This project was developed and executed to provide the Brigham Young University-Idaho Semiconductor Research Group with computational and experimental data as they fine-tune their MOSFET fabrication process. The self-produced computational program calculates the initial and final dopant concentration after the drive-in process

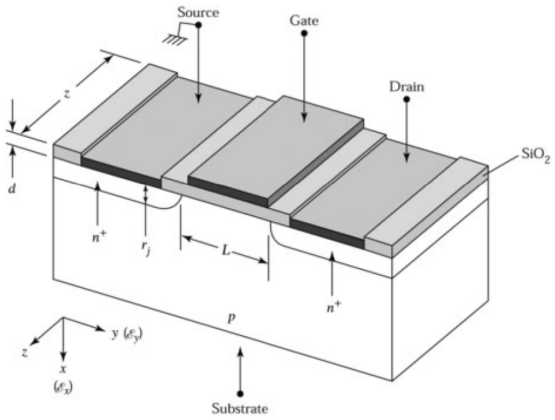

Figure 1.1 Metal-oxide-semiconductor field-effect transistor (MOSFET) diagram. Reproduced from [1].

and displays the results to the user. The program also generates two plots: dopant concentration as a function of resistivity and dopant concentration as a function of depth and time. The experimental data focused on establishing procedures for a phosphorus spin-on dopant solution, etching silicon dioxide from the wafer, driving the phosphorus dopants into the wafer, and characterizing the final wafer.

The following sections provide background information regarding the mechanics of semiconductors and their overall fabrication process.

1.1 Mechanics

Elements such as silicon, germanium, or gallium arsenide form the foundation of a semiconductor. On the atomic level, these intrinsic or pure solids have a "valence band" filled with electrons. The next band, the "conduction band", is empty. These bands are separated by the "band gap", where no electrons are present. For a material to conduct, electrons need enough energy to jump across the band gap from the valence band to the conduction band. Insulators have the same innate properties as semiconductors, although the band gap is larger[4]. Semiconductors with smaller

band gaps require less energy, increasing the probability for electrons near the top of the valence band to jump the band gap into the conduction band. This gives the semiconductor material the potential to be more conductive.

By implanting impurities within an intrinsic semiconductor (also known as doping), the conductive properties of the semiconductor can be tuned, making the semiconductor extrinsic (intrinsic semiconductor doped with a small amount of impurity atoms). The conductivity varies depending on the density of impurities in the semiconductor. Each impurity will either donate an electron or a hole (missing an electron or a space devoid of electrons), carrying a negative or positive charge, respectively[1]. This is shown in Figure 1.2. Impurities that donate an electron are called n-type because they are negative charge carriers. Those that accept electrons or donate a hole are called p-type. The variability between the semiconductor atoms and the impurities allows electrons to "flow" in the solid's crystal structure. Creating regions of different types of charge carriers (holes or electrons) in the semiconductor leads to the formation of junctions between n-type and p-type semiconductors that provide useful electrical properties[1]. These junctions are the basis of diodes and transistors, which act as simple electronic switches without mechanical parts, making them ideal for a wide range of devices, such as microchips in a supercomputer.

1.2 Impurity Doping

To produce a silicon wafer that can be used in an integrated circuit, the semiconductor's electrical properties must be altered. Impurities must be embedded in the semiconductor's crystal structure. This is what enables the semiconductor to have its versatile electrical capabilities. For silicon, the most prominent impurities used are boron, phosphorus, and arsenic, due to their relative size and solubility in silicon

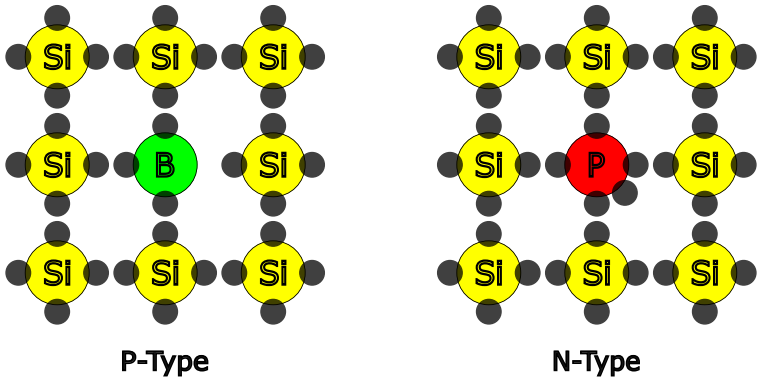

Figure 1.2 Silicon lattice with Boron (P-type) or Phosphorus (N-type) dopant atoms

at high temperatures. Boron introduces p-type impurities (positive charge carriers) into the silicon wafer. On the other hand, phosphorus and arsenic introduce n-type (negative charge carriers).

The impurities can be implanted into the silicon wafer using two different methods: diffusion or ion-implantation. Each method produces a different distribution of impurities in the silicon wafer, as shown in Figure 1.3.

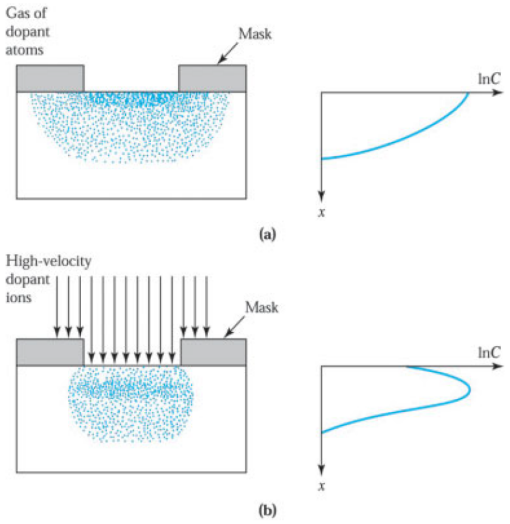

Figure 1.3 Impurity Doping Techniques: (a) Diffusion and (b) Ion-implantation. Reproduced from [1].

1.2.1 Diffusion

For diffusion, the surface of the silicon wafer is exposed to the impurity atoms. This can come from solid, liquid, or gaseous sources, with liquid being the most common. Sometimes, the semiconductor is exposed to doped-oxide sources as well.

To accelerate the diffusion process, the exposed wafer is baked an elevated temperature, ranging between 800°C and 1200°C for silicon, for a set amount of time[1]. The impurities diffuse into the wafer, fanning out with increased depth. The highest concentration is at the surface and decreases with depth. This creates a gradient area of deeply diffused impurities (a junction)[1]. Deep junctions are ideal for high voltage tolerance and power handling semiconductor devices.

1.2.2 Ion-Implantation

The ion-implantation of impurities requires an ion beam, which projects the impurities towards the surface of the wafer. Unlike the diffusion method, the highest concentration of impurities lies beneath the surface at a certain depth. Later in the fabrication process, the top section of the targeted area is removed, leaving a shallow junction. Shallow junctions enhance gate control over the channel and minimize current leakage in semiconductor devices (see Figure 1.1).

This method allows the user to control the overall penetration of the impurities, although there are many particles that travel farther or less than the desired target depth. The silicon wafer also experiences implant damage (lattice disorder) due to nuclear collisions during the process. The lattice disorder severely degrades semiconductor's mobility parameter (how strongly an electron or hole's motion is affected by an applied electric field), as well as its overall lifetime[1].

Annealing

To restore the lattice, annealing is applied at an appropriate combination of time and temperature[1]. Heating the semiconductor allows individual atoms in its lattice to gain energy to move and reorder. Then upon cooling the atoms can be restored its original lattice shape. The conventional annealing process requires heating the semiconductor to high temperatures in an isothermal environment for an extended period. This causes diffusion, resulting in the loss of precise placement of implanted impurities in a specific region or depth. Figure 1.4 shows the Rapid thermal annealing (RTA) process, which can be used instead of the conventional annealing process. RTA greatly reduces the amount of diffusion during annealing. The semiconductor during the RTA process is not in thermal equilibrium with its environment. In the system's chamber, the overhanging tungsten-halogen lamps (1500-2500°C) are much hotter than the semiconductor (600-1100°C), but its walls are much cooler (20-500°C)[1]. These extreme temperature differences through radiation help to rapidly heat and cool the semiconductor, greatly reducing the probability of diffusion during annealing process.

To summarize, producing a functional semiconductor requires several meticulous steps during the impurity doping process.

1.3 This Research

This project focuses on improving the etching and impurity doping processes for the Brigham Young University-Idaho (BYU-I) Semiconductor Research Group, who are seeking to produce a MOSFET (see Figure 1.1).

To do so, first, a numerical model in python was developed to characterize the initial silicon wafer. The program also takes in parameters from the user in order to

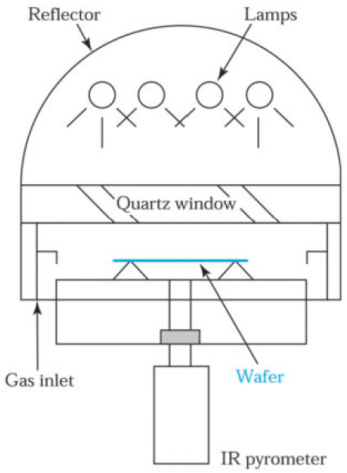

Figure 1.4 Rapid Thermal Annealing Process (RTA) for Ion Implanted Silicon Wafers. Reproduced from [1].

estimate the new characteristics of the wafer after it has been doped.

To validate the created numerical model, experimental work was performed with the group. Specifically, a spin-on dopant solution was fabricated. Then etching process (removal of silicon dioxide (SiO_2) on the surface of the wafer) was performed. And finally the treated wafer was characterized. With the computational and experimental work, we were able to fine-tune the process of creating a functioning semiconductor.

The computational model calculated the initial and final doping concentrations of different silicon wafer types, accounting for the type of dopant agent used (boron or phosphorus). We found that our dopant solution, drive-in process, and characterization tests were successful. Our tests showed that we were able to dope a heavily doped p-type silicon wafer to a normally doped n-type.

Chapter 2

Procedure

2.1 Computational Work

The purpose of the computational work was to create a program that can estimate the dopant profile of a silicon wafer before and after it is doped with either n- or p-type dopant solution, given **temperature**, **time**, **depth**, and **dopant concentration**. The dopant profile describes the concentration of impurities as a function of position within the silicon wafer. The user can specify which variable to solve for, requiring an input for the other three variables.

2.1.1 Required Inputs

In order for the program to accurately predict the initial dopant profile of a silicon wafer, the program requires several inputs from the user, as shown in Figure 2.1. The inputs include:

- wafer size, measured in inches for whole wafer or cm for partial wafer,
- thickness and its tolerance in microns,

- intrinsic or extrinsic. If it is extrinsic, the resistivity of the wafer is also required in Ω·cm,
- boron (p-type) or phosphorus (n-type) for the dopant agent,
- and molarity in moles per liter of dopant agent

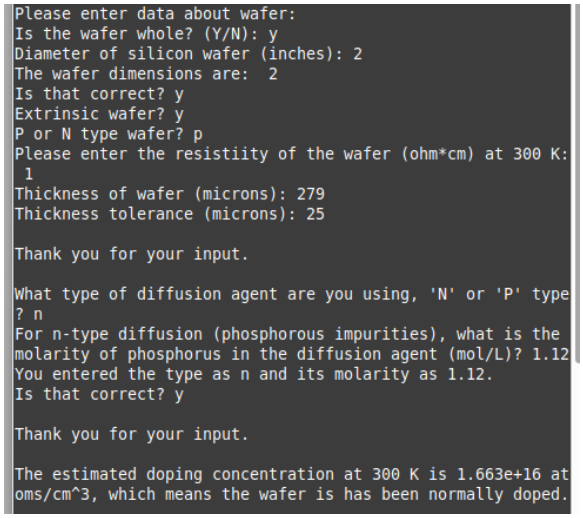

Figure 2.1 User Interface for Initial Wafer Input With Example Inputs

2.1.2 Impurity Calculation

After the values are inputted into the program by the user, the program uses them to solve for the initial doping concentration.

Intrinsic

If the initial wafer is intrinsic, the dopant concentration should be close to zero. This means the resistivity is very large and no indepth calculation is needed. Thus program uses the intrinsic values from Sze[1] for the characterization of the initial wafer.

$$\rho = \frac{1}{\sigma} = \frac{1}{q(\mu_n + \mu_p)n_i} \tag{2.1}$$

where q is the elementary charge[1], μ_n and μ_p are the charge carrier mobilities for electrons and holes[2], respectively, and n_i is the intrinsic carrier concentration (atoms/cm^3). The intrinsic carrier concentration is the number of carriers (either electrons or holes) per unit volume in pure silicon. In this case,

$$n = p = n_i = 9.65 \times 10^9 \tag{2.2}$$

for intrinsic wafers. The intrinsic carrier concentration is also the initial dopant concentration. The program displays the dopant concentration to the user and saves the value for further calculations during the drive-in process.

Extrinsic

For extrinsic wafers, there are no set values for the mobility, μ. Plus, the initial dopant concentration may be unknown. Thus, the program uses the user entered resistivity value of the wafer, ρ, to solve for the charge carrier mobility, μ, and the doping concentration, C, of the wafer at room temperature (300 K)[1, 5],

$$\rho = \frac{1}{q\mu C} \tag{2.3}$$

where q is the elementary charge in coulombs. The resistivity value provided by the user is assigned as the *resistivity target*. This target represents the value that the program attempts to match while solving for the two unknowns in the equation above.

With this equation, two unknowns are present; therefore, Brent's Method and the Caughey-Thomas Empirical Formula are used to determine the charge mobility, μ, and the doping concentration, C[6, 7]. The Caughey-Thomas Empirical Formula specifically solves for the charge mobility, μ.

[1] 1.602×10^{-19} C

[2] $\mu_n = 1450$ cm^2/V*s and $\mu_p = 505$ cm^2/V*s

Brent's Method is a root-finding algorithm that combines the bisection method, the secant method, and inverse quadratic interpolation. Brent states that convergence is guaranteed for functions computable within [a,b][6].

Assuming this condition holds, the values a and b are assigned as initial guesses for C, which are 1×10^{11} atoms/cm^3 and 1×10^{22} atoms/cm^3 respectively. These values represent the lowest and highest dopant concentrations possible (see Figure 2.2). Using these two possible doping concentrations, each value is substituted into the resistivity equation and the Caughey-Thomas Empirical Formula,

$$\mu = \mu_{min} + \frac{\mu_{max} - \mu_{min}}{1 + (C/C_{ref})^{\alpha}} \tag{2.4}$$

to determine a resistivity value that matches the resistivity target. Using this formula and the guesses a and b, the program verifies whether the calculated resistivity matches the resistivity target provided by the user.

Here, μ_{min} and μ_{max} represent the minimum and maximum charge carrier mobility (cm^2/V*s), respectively, which are derived from previous empirical studies [7]. C_{ref} is the reference doping concentration (atoms/cm^3), and α is an empirical fitting parameter. Both depend on the type of diffusion agent used.

For the guess a, the calculation proceeds as follows[3]:

$$\mu_a = \frac{\mu_{min} + (\mu_{max} - \mu_{min})}{1 + (a/C_{ref})^{\alpha}} \tag{2.5}$$

$$\rho_a = \frac{1}{q\mu_a a} \tag{2.6}$$

Once the a and b guesses for the resistivity are formulated, the resistivity target is subtracted from each calculated value. If the two guesses have the same sign, then the program outputs a message indicating that the function does not change signs over the given bracket, $[a, b]$, and suggests adjusting the brackets or inputs. If the guesses

[3]The value for guess b is calculated using the same method.

have different signs, the program solves for a and b using the Brent's Method solver in "root_scalar". The program then checks whether the calculated result converges. If successful, it saves and displays the calculated root as the doping concentration, C. It also determines the level of doping present according to the scale in Figure 2.2.

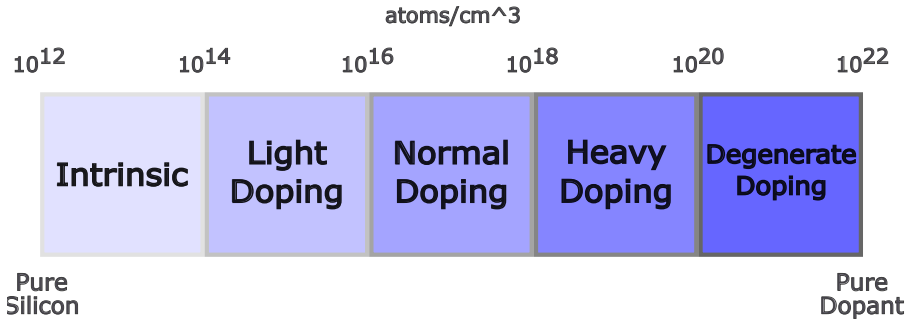

Figure 2.2 Doping Levels

The last part of the initial characterization section displays a doping concentration vs. resistivity plot, highlighting the calculated doping concentration of the initial wafer with a "+", as shown in Figure 2.3.

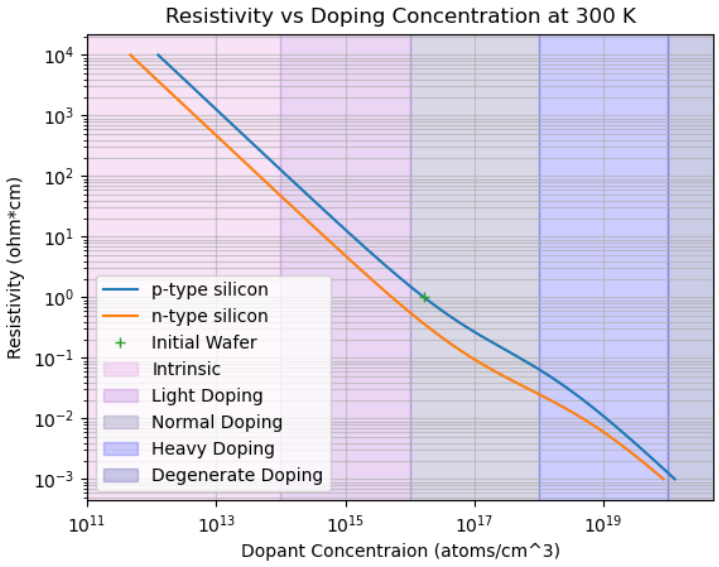

Figure 2.3 Program Produced Resistivity vs. Doping Concentration Plot with P- and N-type Lines

2.1.3 Diffusion Calculation

Diffusion is governed by four main parameters: temperature, time, depth, and dopant concentration. Three must be supplied by the user, then the program calculates the other. The user is asked to specify which will be solved for via the interface prompt shown in Figure 2.4:

```
To characterize the diffusion process, there are four crucial
variables to consider: baking temperature (T,degree C), time (
t,s), diffusion depth (x,microns), and dopant concentration (C
, atoms/cm^3). One will be the unknown variable.
To specify the unknown variable to solve, please indicate with
 the following number:
temperature = 1
time = 2
diffusion depth = 3
dopant concentration = 4
Which variable is the unknown? 4

Thank you. Please provide info for the known variables.

Temperature wafer is baked at (C): 1000
Time wafer will be baked (s): 1800
Desired depth of dopant (microns): .15
Dope agent is phosphorus

Initial Concentration = 1.663e+16
New Concentration = 1.775e+20
Final dopant concentration = 1.775e+20 atoms/cm^3, and type: n
```

Figure 2.4 User Interface for Diffusion Calculation With Example Inputs

The program then uses equations 2.7, 2.8, 2.9, and one of the diffusion coefficient equations, 2.10 or 2.13, to determine the requested output[1, 2, 5]:

Dopant Concentration (atoms/cm^3):

$$C = C_s e^{(\frac{-x^2}{4Dt})} \tag{2.7}$$

where t is the time in seconds and D is the diffusion coefficient in cm^2/s.

Surface Concentration (atoms/cm^3):

$$C_s = \frac{S}{\sqrt{\pi D t}} \tag{2.8}$$

Total amount of dopant per unit area (atoms/cm^2):

$$S = \frac{M N_A t_f}{1000} \tag{2.9}$$

where M is the molarity of the spin-on dopant solution (mol/L), N_A is Avogadro's number[4], and t_f is the thickness of the spin-on dopant solution (cm).

The program assumes the thickness of the spin-on dopant solution to be 1×10^{-5} cm. This value is estimated based on observations of the viscosity of the photoresist used by the BYU–Idaho research group during the photolithography process. The photoresist solution has a film thickness of approximately 3×10^{-6} cm [8]. Although a full analysis of the dopant solution thickness is not currently available for the research group, this assumption for the spin-on dopant thickness is reasonable for this project. In the future, the research group may develop a method to measure the thickness of the dopant solution. At that time, the program can be modified to allow the user to input the measured dopant solution film thickness instead of using the estimated value currently programmed into the code. Improved precision in the spin-on dopant thickness would reduce the uncertainty in the program's dopant concentration calculation. Until such measurements are available, the estimated thickness value of 1×10^{-5} cm is used.

The diffusion coefficient, or diffusivity value, is dependent on the dopant type, which is shown in equations 2.10 and 2.13[2].

Boron Diffusion Coefficient (cm^2/s):

$$D_x = D^0 + D^+[\frac{p}{n_i}] \tag{2.10}$$

$$D^0 = 0.037e^{-3.46/kT} \tag{2.11}$$

$$D^+ = 0.76e^{-3.46/kT} \tag{2.12}$$

Phosphorus Diffusion Coefficient (cm^2/s):

$$D_x = D^0 + D^=[\frac{n}{n_i}]^2 \tag{2.13}$$

[4] 6.02214×10^{23} atoms/mol

$$D^0 = 3.85e^{-3.66/kT} \tag{2.14}$$

$$D^+ = 44.2e^{-4.37/kT} \tag{2.15}$$

where p and n are the amounts of holes and electrons, respectively. For boron, since $p >> n_i$, $p \approx N_A = C(x,t)$, the concentration of dopant atoms at depth, x. The temperature dependent intrinsic carrier concentration of silicon in atoms/cm^3, n_i, is shown in equation 2.16:

$$n_i = 5.29 \times 10^{19} * (\frac{T}{300})^{2.54} * e^{-6726/T} \tag{2.16}$$

where T is the temperature in kelvin.

Temperature:

To solve for the temperature during the drive-in process, the user provides the desired diffusion time, diffusion depth, and the final dopant concentration of the wafer. With these values entered, the program first calculates the total amount of dopant, S, using equation 2.9 and the molarity input from the user.

The program then calculates the effective diffusivity from the Gaussian profile found in equation 2.7 with equation 2.8 substituted in.

To calculate T in the diffusion coefficient, the program first solves for D in equation 2.7 using the Lambert W function.

The Lambert W function is a multi-valued function with infinitely many branches, each branch providing a separate solution to $u = we^w$ [9, 10]. Two of these branches produce real-valued solutions: the principal branch, $k = 0$, and the $k = -1$ branch. The principal branch is valid for values $u > -1/e$, while the $k = -1$ branch applies for $-1/e < u < 0$. All valid solutions in the program require that $k = -1$.

First, the program substitutes equation 2.8 into Equation 2.7, squares it, and then

rearranges the expression to isolate D outside the exponential term.

$$D = \frac{S^2}{C^2 \pi t} e^{(\frac{-2x^2}{4Dt})} \tag{2.17}$$

The value in the exponent is set equal to u and the program rearranges the expression to isolate D. Then the program substitutes the expression into D in equation 2.17 and rearranges to isolate u.

$$u = (\frac{-C^2 x^2 \pi}{2S^2}) e^{-u} \tag{2.18}$$

To account for the u in the exponent, the program applies the Lambert W function, which is the inverse function of ue^u. In other words, $W(u)$ satisfies the relationship $u = W(u)e^{W(u)}$ for any complex number u. The Lambert W function is imported from scipy.special.

$$u = W\left(\frac{-C^2 x^2 \pi}{2S^2}\right) \tag{2.19}$$

The program substitutes the u value back into the equation and solves for D.

$$\frac{-x^2}{2Dt} = W\left(\frac{-C^2 x^2 \pi}{2S^2}\right) \tag{2.20}$$

$$D = \frac{-x^2}{2tW\left(\frac{-C^2 x^2 \pi}{2S^2}\right)} \tag{2.21}$$

Next, the program solves for the baking temperature, T, using either equation 2.10 or 2.13 and their corresponding equations, which depend on the dopant agent. With multiple equations, the program uses fsolve from scipy.optimize and an initial guess of $T_{guess} = 1273.15$ K to find the actual temperature. The final result then displays to the user in kelvin.

Time:

The calculation of the diffusion time in the program follows a similar procedure to the temperature calculation. In this case, the program calculates S the same way and uses the Lambert W function to find D. The program begins with equation 2.7, where $u = \frac{-x^2}{2Dt}$ and $t = \frac{-x^2}{2Du}$ are substituted, and the equation is rearranged to solve for u:

$$u = (\frac{-C^2x^2\pi}{2S^2})e^{-u} \tag{2.22}$$

Here, u is similar to the value used in the temperature calculation, but t is now the unknown variable instead of D.

The program then applies the Lambert W function to solve for the t in the exponent:

$$u = W\left(\frac{-C^2x^2\pi}{2S^2}\right) \tag{2.23}$$

The code solves for t by substituting the value of u back into the equation and rearranging it for t:

$$t = \frac{-x^2}{2DW\left(\frac{-C^2x^2\pi}{2S^2}\right)} \tag{2.24}$$

The baking time corresponding to the specified diffusion parameters then displays to the user in seconds.

Diffusion Depth:

The diffusion depth calculation, x, during the drive-in process is much simpler. It requires only simple algebra and does not require the Lambert W function, unlike the temperature and time calculations.

The program solves for x using equation 2.7. The remaining values are either provided by the user or calculated by the program using equations 2.8, 2.9, and equation 2.10 or 2.13, depending on the dopant agent.

$$x = 2\sqrt{-Dt \ln\left(C/C_s\right)} \tag{2.25}$$

The diffusion depth displays to the user in micrometers for convenience, although the program initially calculates the depth in centimeters.

Dopant Concentration:

Lastly, the dopant concentration calculation, C, is straightforward. Using the temperature, time, and depth parameters provided by the user, the program calculates the dopant concentration using equations 2.7, 2.8, 2.9, and one of the diffusion coefficient equations, 2.10 or 2.13. The value then displays to the user in atoms/cm^3.

After the designated unknown variable is calculated and displayed, the program generates a dopant concentration versus depth plot over time (Figure 2.5). The plot indicates the final characteristics of the doped wafer and the various doping level regions.

2.2 Experimental Work

The purpose of the experimental work was to fine tune the doping process and characterization of a silicon wafer. This included developing a spin-on dopant solution, testing an etchant for silicon dioxide, diffusing impurities into a silicon wafer, and testing the wafer's conductivity and diffusivity.

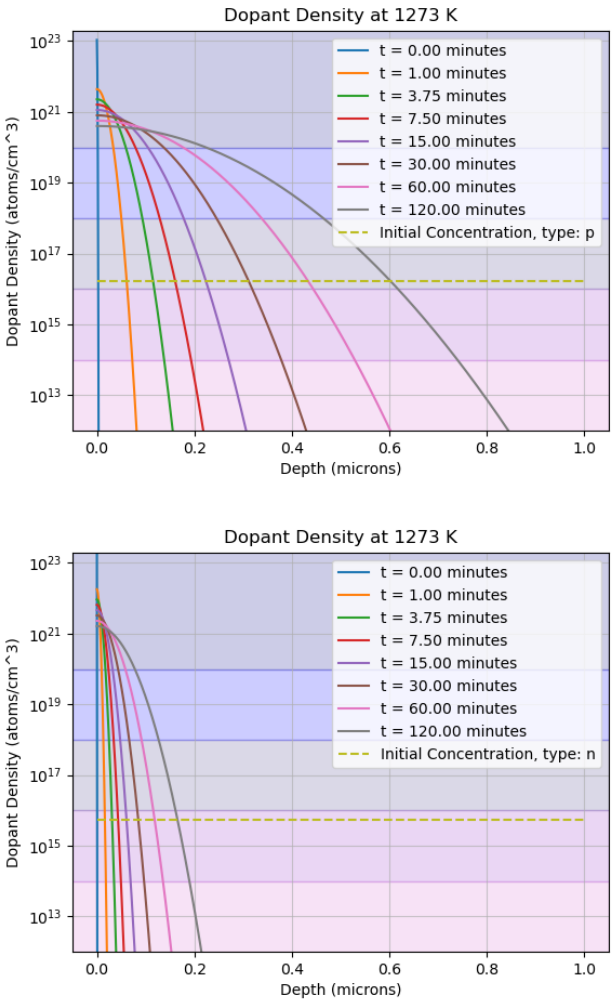

Figure 2.5 Program Produced $C(x,t)$ Plots for P- to N-type doping and N- to P-type doping

2.2.1 Spin-On Dopant

To begin, a spin-on dopant needed to be made. Previously, the BYU-Idaho Semiconductor Research team developed a spin-on dopant solution using a reflux condenser, but the solution was too viscous. This was an issue because the solution would not adhere to the silicon wafer when it was spun on. This time, when the solution was prepared, the duration in the reflux condenser was altered. The solution required[11]:

- 1 mL tetraethyl orthosilicate (TEOS)

- 1 mL deionized water
- 4 mL pure food grade ethanol (90% or better purity)
- 0.5 mL 85% phosphoric acid (H_3PO_4)

The solution had a shelf life of three months. For a full detailed explanation of the reflux condenser procedure, refer to Appendix F.

Next, the molarity of the solution was calculated. It helps determine the amount of impurities available for diffusion into a silicon wafer during the drive-in process, which is required for the diffusion numerical model. The molarity is found by calculating the moles of the dopant agent. In this experiment, it is phosphoric acid.

$$mol_{H_3PO_4} = \frac{V_{H_3PO_4} * \rho_{H_3PO_4} * c_{H_3PO_4}}{M_{H_3PO_4}} = \frac{(0.5)(1.685)(0.85)}{97.995} = 7.31 \times 10^{-3} \quad (2.26)$$

where the volume of H_3PO_4, $V_{H_3PO_4}$, is in mL, the density, $\rho_{H_3PO_4}$, is in g/mL at 25°C, and the molecular weight, $M_{H_3PO_4}$, is g/mol. Finally, the mass of the dopant agent is divided by the total volume of the solutions:

$$M = \frac{mol_{H_3PO_4}}{V_{total}} = \frac{7.31 \times 10^{-3}}{6.5 \times 10^{-3}} = 1.12 \quad (2.27)$$

to calculate the molarity of the dopant agent in mol/L.

2.2.2 Drive-in Process

With the spin-on dopant prepared, the solution was ready to be spun onto a silicon wafer (see Figure 2.6). The wafer used was a heavily doped, p-type silicon wafer, manufactured by Waferpro. The original p-type wafer provides the positive charge carriers needed for the MOSFET substrate. The dopant solution was applied to the wafer by securing it in place on the center of a spin coater that was held in place with a vacuum chuck. Before applying the solution, the spin coater was initially set at its

lowest setting of 300 rpm. Using a pipette, 5 mL of the solution was applied to the center of the wafer, slowly increasing the rotation rate to allow the solution to spread to the edges.

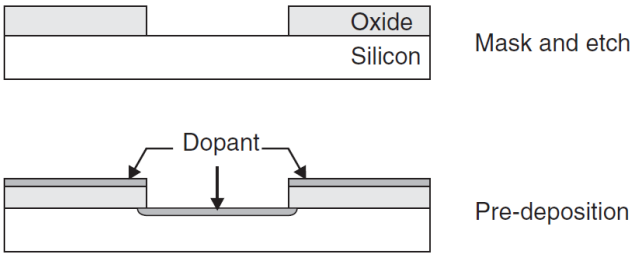

Figure 2.6 Pre-deposition of Dopant Solution on a Silicon Wafer Diagram. Reproduced from [2].

After the dopant solution was applied, the wafer was carefully removed from the spin coater and placed on a preheated hotplate set at 500°C for 20 minutes. This was to help the diffusion process get a jump start before performing the baking process, which took place two weeks later due to scheduling difficulties. Once we were ready to drive the impurities deeper into the silicon wafer, we noticed that the outer ring of the silicon wafer did not have the iridescence feature that the center did. This can be seen in Figure 2.7. From this, we assumed that the dopant solution did not reach the edge of the silicon wafer. We accounted for this in our characterization tests by only measuring from the center of the wafer. In the same outer ring section, we noticed that small crystallization had formed. This further supported our assumption that a sufficient amount of dopant solution did not reach the edges of the wafer.

The drive-in process is essential because the extreme heat applied to the silicon wafer enables the impurities to diffuse deeper into its crystal lattice. Using a SentroTech horizontal furnace, the dopant-applied silicon wafer was heated to 1000°C for roughly 40-50 minutes in a quartz boat: 10 minute of heat up time, 10 minute steady-temperature, and the remaining time for cool down in the oven until the oven

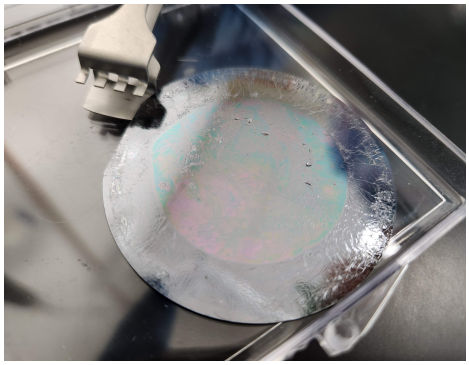

Figure 2.7 Treated Wafer after Application of Spin-on Dopant Solution and Before Impurity Drive-in Process

temperature reached 100°C. To decrease the cool down time, we used two external fans that blew towards the furnace tube from a reasonable distance, trying not to cause any thermal shock to the quartz tube for slowing it down too quickly. Once the furnace had cooled to 100°C, the wafer was carefully removed and prepared for characteristic tests.

2.2.3 Wafer Characterization

To verify that the phosphorus impurities diffused into the silicon wafer, several characterization tests were performed on the wafer: the four-point probe, hot-point probe, and resistance tests. Before the tests were performed, the wafer was etched with Armour Etch one more time to remove the phosphosilicate glass that had formed on the surface of the wafer just after the drive-in process. The difference can be seen in Figure 2.8.

Four-Point Probe Test

The four-point probe test was used to determine the bulk resistivity of diffused layers in the silicon wafer. The resulting resistivity can then be used to determine the impurity content [5, 12].

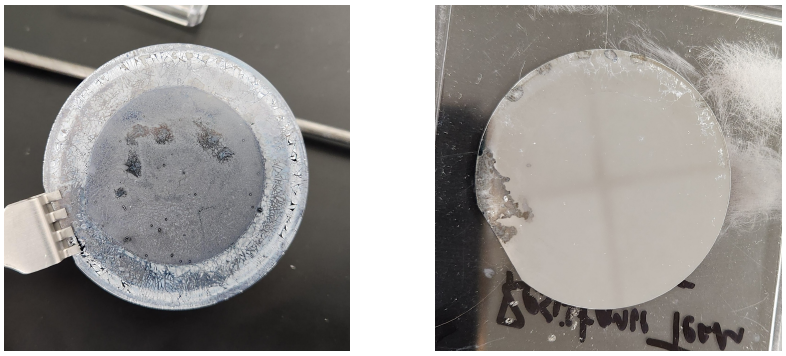

Figure 2.8 Treated Wafer After Impurity Drive-In Process, then Treated With Armour Etch to Remove SiO_2.

The four-point probe contains four small probes that are applied to the surface of the silicon wafer. The two outside probes are connected to an ammeter, allowing current to flow through them and the wafer. The two interior probes are connected to a voltmeter to measure the voltage drop across the inner part of the wafer (see Figure 2.9). To accurately connect the bulk resistivity of the diffused layer to the voltage drop and the current, the spreading current from the outside probe needs to be accounted for, which was completed by Grove[5]:

$$\rho = \frac{\pi}{\ln 2}\frac{V}{I}x_j = 4.532\frac{V}{I}x_j \tag{2.28}$$

The V is the voltage drop, I is the current, and x_j is the depth of the junction. This calculation is valid for materials whose depth is small compared to the spacing between probes, s, and whose lateral dimensions are large. Due to lack of equipment, we were not able to determine the depth of the junction, so we calculated the bulk resistivity, changing x_j to d, the thickness of the silicon wafer.

With the average resistivity calculated, the average dopant concentration in atoms/cm^3 was determined by rearranging equation 2.3, if the mobility is known:

$$C = \frac{1}{q\mu\rho_B} \tag{2.29}$$

The dopant concentration can also be determined by looking at Figure 2.3.

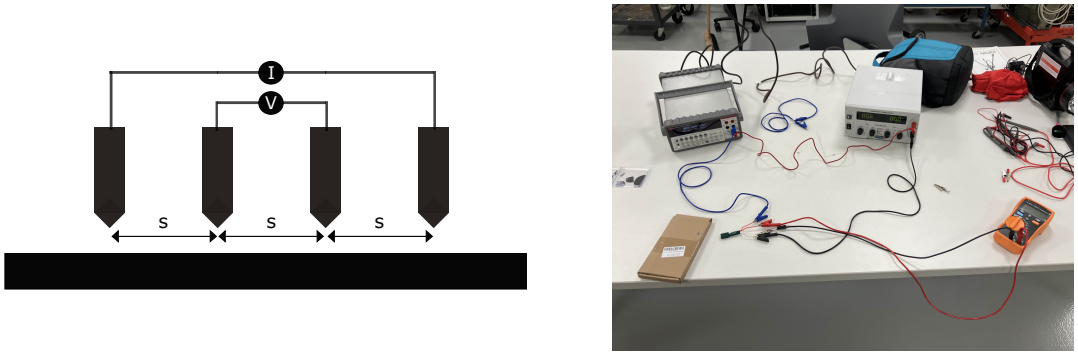

Figure 2.9 Four-Point Probe Diagram and Experimental Setup

Hot-Point Probe Test

The hot-point probe test is a quick method to determine whether the majority of carriers in semiconductors are holes (p-type) or electrons (n-type). It can also verify if all the oxide has been removed from the surface during the etching process.

To perform the test, a heated soldering iron was placed on or near the positive probe of a handheld multimeter. The negative probe is kept separate from the heat source. Both probes are placed on the surface of the wafer a few centimeters apart, and the current is read. If the current reading is positive, then the material has negative charge carriers, or electrons (n-type). If it's negative, then there are positive charge carriers, or holes (p-type). If there is no current reading, then either there is no proper contact or an oxide is present, which is an insulator.

Resistance Test

The resistance test was a quick measurement of the resistance on the surface of the silicon wafer. We used a handheld multimeter, touching the probes to different locations on the surface of the wafer. This test verifies some type of change in the surface resistance of the wafer.

Chapter 3

Results

3.1 Computational Results

The numerical model of dopant diffusion in semiconductors was completed. The results from the impurity calculation section were compared to the empirical plots by S. M. Sze and M. K. Lee [1]. The plot generated by the program also matched the published results for similar starting conditions, verifying that the impurity calculations were accurate (see Figure 2.3).

Unlike the plot produced by Sze and Lee, the program-generated plot displays the doping level regions. This makes it easier for the BYU–Idaho Semiconductor Research group to verify the characterization of the initial wafer being analyzed.

Next, the diffusion process calculations were verified by inputting identical parameters while solving for different unknown variables. For example, if the user specified that a silicon wafer was heated to 1000°C for 1800 seconds and reached a depth of 1 micrometer, the program calculated the resulting dopant profile according to those parameters.

To validate the program, the same parameters were tested again, but the pre-

viously calculated value was treated as a known parameter and a different variable was designated as the unknown. After cycling through each variable as the unknown (temperature, time, depth, and dopant concentration), the calculated results remained consistent. This confirmed that the diffusion calculations were internally consistent and indicated that no computational errors were present in the program.

3.2 Experimental Results

Comparing the program's diffusion calculations with experimental results requires empirically doped silicon wafers. However, by the time the program and experimental work were completed, a full comparison between the two was not possible within our time frame. Thus, this analysis remains pending.

The experimental work introduced phosphorus impurities into an initially normal-doped p-type 2" silicon wafer through impurity diffusion. Three tests were performed on the treated wafer to determine the effectiveness of the doping and diffusion process executed by the BYU-Idaho Semiconductor Research group. The results are listed in Table 3.1. The reference wafer was an initially normal-doped (1-10 Ω·cm) p-type 2" silicon wafer. The test wafer was a heavily doped ($\leq$ 0.005 Ω·cm) p-type 2" silicon wafer, with n-type spin-on dopant driven into it.

Test	Reference Wafer	Tested Wafer
Hot-Point Test:	slight -V with firm contact	$\sim 3 \pm 1$ mA
Four-Point Test:	0.2-0.005 mA and no V reading	18 ± 1 mA and 45 ± 5 mV
Resistance Test:	$\sim$ 300 kΩ (unstable)	30 ± 1 Ω

Table 3.1 Silicon Wafer Characterization Test Results

The hot-point probe test showed that the tested wafer was no longer p-type but n-type due to the sign change in the voltage reading. The four-point probe test determined the voltage and current passing through the silicon wafer. With these results, the bulk resistivity ($\Omega * cm$)was calculated using equation 2.28, with the wafer thickness, d, substituted in for x_j:

$$\rho_B = (4.532\frac{V}{I})d = (11.33\Omega)(279 \times 10^{-4} cm) = 0.316 \tag{3.1}$$

This measurement showed a difference in the bulk resistance of the diffused layers. This measurement indicated that impurities had diffused into the silicon wafer during the drive-in process. Using this resistivity value, the final dopant concentration was determined by looking at Figure 2.5. The dopant concentration obtained from the figure and the bulk resistivity is roughly 10^{16} atoms/cm^3, meaning it is normally doped.

Lastly, the resistance test also indicated a change in resistance at the surface compared to the reference wafer.

Even with the results listed, there were still many uncertainties and assumptions made. During our measurement tests, we experienced very unstable readings. The listed values in the table were either an average of the measured values (indicated by "$\sim$"), a range of values, or what was accurately measured to the best of our capabilities. To add to that, the four-point probe that was used was student made. One of the probes was not flush with the rest, making it difficult to get full contact without damaging the wafer. Next, the tests for the reference wafer were not performed on the same wafer that was doped due to a lack of foresight. To account for that, we measured a similar wafer to the treated doped wafer in order to obtain some type of comparison. Lastly, the large time gap between the drive-in and testing could have affected the recorded results. Thus, more research and planning are needed to fully

verify the doping process of a silicon wafer.

Overall, the computational and experimental work produced significant steps towards solidifying the fabrication process of a functioning semiconductor. The computational work produced accurate resistivity and dopant concentration plots. The experimental work successfully doped a heavily doped p-type silicon wafer to a normally doped n-type. The computational and experimental work presented here is only the beginning of future work for the BYU-Idaho Semiconductor Research group.

Chapter 4

Conclusion and Future Work

This project demonstrated the ability to model and execute the diffusion process in a silicon wafer. The numerical model of dopant diffusion successfully calculates the initial dopant concentration of a wafer and then its final state after the drive-in process. The python program is adaptable, allowing the user to define which variable to solve for during the drive-in process, whether that is temperature, time, depth, or dopant concentration. This enables more versatility and estimation for future research needs. The program was tested alongside the empirical work in *Semiconductor Devices: Physics and Technology* by Sze and Lee, and aligns with their published work[1].

The experimental work successfully diffused impurities into a silicon wafer, changing a heavily doped p-type wafer to a normally doped n-type. Thus resulting in a diode junction. More silicon wafers need to be treated with the dopant solution to ensure the accuracy of doping process. Furthermore, better techniques or instruments are needed to improve characterization tests on silicon wafers. Overall, the performed experimental work was a significant step towards fine-tuning the doping process and eventually fabricating a MOSFET for the BYU-Idaho Semiconductor Research group.

Bibliography

[1] S. Sze and M. K. Lee, *Semiconductor Devices: Physics and Technology* (Wiley Global Education, 2012).

[2] S. W. Jones, "Diffusion in Silicon," In , (2008).

[3] National Institute of Standards and Technology, https://www.nist.gov/semiconductors, 2026-02-21.

[4] G. N. Felder and K. M. Felder, in *Modern Physics* (Cambridge University Press, 2022), Chap. 11 Solids.

[5] A. Grove, *Physics and Technology of Semiconductor Devices* (New York, Wiley, 1967), Chap. 3, Solid-State Diffusion.

[6] R. Brent, *Algorithms for minimization without derivatives* (Prentice-Hall, 1973).

[7] D. Caughey and R. Thomas, "Carrier mobilities in silicon empirically related to doping and field," Proceedings of the IEEE (1967).

[8] KemLab Photoresist, https://www.kemlab.com/kl6000-resist, 2026-03-07.

[9] R. M. Corless and D. J. Jeffrey, *The Princeton Companion to Applied Mathematics* (Princeton University Press, 2015), Chap. III.17, The Lambert W Function.

[10] SciPy v1.17.0 Manual, https://docs.scipy.org/doc/scipy-1.17.0/reference/generated/scipy.special.lambertw.html, 2026-02-14.

[11] ProjectsInFlight, https://youtu.be/1dFj-tGn8DI?si=eLipgh8TGeoz1oyL, 2024-09-21.

[12] University of Illinois Department of Electrical and Computer Engineering, *ECE444: Theory and Fabrication of Integrated Circuits*, 2009.

Appendix A

Main Code

Contained below is the code to the dopant diffusion program. It implements object-oriented programming to make the program more versatile and easy to use.

This appendix contains the main file, which calls and executes the attached libraries and functions in order for the program to run. The other files each perform a specific function for the program, whether that is user interface, creating plots, or performing calculations. The accompanying files for dop_main.py are:

- dop_userinput.py, which contains majority of the text, inputs, and outputs between the user and the program.
- dop_initialc .py calculates the initial concentration of silicon wafer.
- dop_diffusion .py calculates the unknown variable from the user during the diffusion process.
- dop_plots.py creates plots from calculated results.

Further details about each file are found in their respective appendix.

```
#################################################################
# File Name: dop_main.py
```

```
# Desription: The "dop" collection of files intends to create a
# user-friendly set of files used to calculate the diffusion of
# impurities in a silicon wafer, the impurity
# concentration, and characterize the wafer's electrical
# properties.
#
# This is the main file that calls other classes to
# properly run the program. Other files include:
# - dop_userinput.py
# - dop_initialc.py
# - dop_diffusion.py
# - dop_plots.py
#
# Overview:
# - Import Libraries and Classes
# - Define Classes
# - User Inputs:
#       - Wafer Input
#       - Diffusion Agent Input
# - Impurity Calculation in Initial Wafer
# - Plot Resisivity vs. Dopant Concentration Plot
# - User Inputes:
#       - Known and Unknown Variables
# - Diffusion Calculation
# - Plot Dopant Density vs. Depth at various time intervals
#
# Author: Makelle Forson (win22012@byui.edu)
# Date: 31 May 2025 (Updated 28 March 2026)
##################################################################
# Import Libraries and Classes #
from dop_userinput import UserInput              # File to collect user
```

```
    inputs
from dop_initialc import InitialC                # File for calc'ing
    initial dopant concentration
from dop_diffusion import Diffusion              # File for calculating
    diffusion
from dop_plots import DopePlots                  # File for plots

# Define Classes #
UI = UserInput()


#### User Inputs ####
# Wafer Input #
# dim (int), whole (boolean), extrin (boolean), type (), rho(float), d
    (), d_tol ()
dim,whole,extrin,init_type,rho,d,d_tol = UI.user_input_wafer()


# Diffustion Agent Input #
dop_type, Mol, dop_agent = UI.user_input_dop_agent()    # dop_agent is a
     string ("boron" or "phosphorus")


# Calculate Impurity Concentraion in Initial Wafer #
INC = InitialC(init_type)                # Initialize variables/inputs.
init_C = INC.estimateDoping(rho)         # Perform the calculation and
    set it equal to init_c.
INC.print_dop_calc(init_C)               # Print results and its doping
    level to user.
DP = DopePlots()
DP.plot_rho_vs_conc(init_C,rho)          # Plots changing resistivity vs
```

```
    dopant concentration


# Diffusion Input of unknown and known variables #
unknown,knowns = UI.user_input_diff_unknown()


# Diffusion #
DF = Diffusion(Mol,dop_agent,d,init_C)  # Needs mol, dop_agent, wafer
    thickness, and inital dopant concentration inputs
if unknown == 1:          # Unknown temp to calculate
        T,S,D = DF.calc_temp(knowns)
        print(f"Temperature = {T:.2f} degrees C")
        t = knowns[0]     # Desired Time from user
        x = knowns[1]     # Desired depth from user in cm
        C = knowns[2]     # Desired dopant concentration from user
elif unknown == 2:        # Unknown time to calculate
        t,S,D = DF.calc_time(knowns)
        print(f"Time = {t:.2f} seconds, or {t/60:.2f} minutes")
        T = knowns[0]     # Desired Temperature from user in K
        x = knowns[1]     # Desired depth from user in cm
        C = knowns[2]     # Desired dopant concentration from user
elif unknown == 3:        # Unknown depth to calculate
        x,S,D = DF.calc_depth(knowns)
        x = x*1e4         # convert to microns
        print(f"Depth = {x:.4f} microns")
        x = x*1e-4        # convert back to cm
        T = knowns[0]     # Desired Temperature from user in K
        t = knowns[1]     # Desired time from user in s
        C = knowns[2]     # Desired dopant concentration from user
else:                     # Unknown Depth Profile to calculate
```

```
        C,S,D = DF.calc_depth_profile(knowns)
        T = knowns[0]    # Desired Temperature from user in K
        t = knowns[1]    # Desired time from user in s
        x = knowns[2]    # Desired depth from user in cm

# Calculates the final concentration of the wafer, taking into account
    the
# initial concentration of impurities. It also changes the type if
    needed.
C_f, new_type = DF.calc_final_C(C,init_type)
print(f"\nInitial Concentration = {init_C:.3e}")
print(f"New Concentration = {C:.3e}")
print(f"Final dopant concentration at that time and depth is {C_f:.3e}
    atoms/cm^3, and the type is {new_type}")

# Plots Depth Profile vs Depth with various time intervals #
# It also includes the initial dopant concentration of the wafer.
# That is why S and D are inputs.
DP.plot_C_vs_x(Mol,dop_agent,d,init_C,T,t,x,C_f,init_type,new_type)
```

Appendix B

User Input Code

This file's main purpose is to contain the text, inputs, and (almost all) outputs between the user and the program. This makes it easy to navigate the program in order to find particular prompts or outputs. Then, each input that is received from the user is directed back to the main file, dop_main.py, to be used by other attached files.

```
#
    ##############################################################################

# File Name: dop_userinput.py
# Description: This file is a subpart of dop_main.py. It's purpose is to
# interact with the user, whether by asking for data or outputting
    calculated
# results.
#
# Overview:
# - Import Libraries and Classes
# - Functions:
#       - cor_dim: Correct Dimensions From the User
```

```
#       - whole_wafer: Whole or Half Wafer
#       - extrinsic: Extrinsics or Intrinsic wafer
#       - user_input_wafer: Initial Conditions of Wafer
#       - user_input_dop_agent: Doping Agent Information
#       - user_input_diffusion: Diffusion Parameters
#
# Author: Makelle Forson (win22012@byui.edu)
# Date: 21 May 2025 (Updated 28 March 2026)
#
   ##############################################################################

# Import Class #
from dop_diffusion import Diffusion
import numpy as np

# Class Declaration #
class UserInput:

############## Correct Dimensions ###################
        # Function to check with user if dimensions are correct #
        def cor_dim(self,cordim,dim,whole):
                if whole == True:       # New Dimensions for whole wafer
                        if cordim in ["yes", "y"]:
                                return dim
                        elif cordim in ["no","n"]:
                                dim = input("Please enter correct
                                    diameter (inches): ")
                                return dim
                        else:
                                print("\n\nInvalid input. Please answer
                                    'yes' or 'no'.\n\n")
```

```
            else:
                if cordim in ["yes","y"]:
                    return dim
                elif cordim in ["no","n"]:
                    xdim = input("Please enter correct
                        dimensions.\nx dimension (cm): ")
                    ydim = input("y dimension (cm): ")
                    zdim = input("z dimension (cm): ")
                    dim = np.array([xdim,ydim,zdim])
                    return dim
                else:
                    print("\nInvalid input. Please answer '
                        yes' or 'no'.\n")

############## Correct Dopant ##################
        # Checks with user if diffusion agent info is correct #
        def cor_dop_agent(self, dop_type, M):
            print(f"You entered the type as {dop_type} and its
                molarity as {M}.")
            cor_dop = input("Is that correct? ").strip().lower()
            if cor_dop in ["y","yes"]:
                return dop_type, M
            elif cor_dop in ["n","no"]:
                dop_type = input("Is it a 'N' or 'P' type
                    diffusion agent? ")
                M = float(input("What is it's molarity (mol/L)?
                    "))
                return dop_type, M
            else:
```

```
                print("\n\nInvalid input. Please enter 'yes' or
                    'no'.\n\n")
                return self.user_input_dop_agent()

############## Wafer Dimensions ###################
        # Check if wafer is whole or partial, along with its dimensions
            #
        def whole_wafer(self,whole):
                if whole in ["yes", "y"]:
                        whole = True
                        dim = input("Diameter of silicon wafer (inches):
                            ")
                        return dim      # outputs user entry as dim from
                             function
                elif whole in ["no", "n"]:
                        whole = False
                        print("Assuming the wafer is square/rectangular,
                            please provide its dimensions:")
                        xdim = input("x dimension (cm):")
                        ydim = input("y dimension (cm):")
                        zdim = input("z dimesnion (cm):")
                        dim = np.array([xdim,ydim,zdim])          # puts x
                            ,y,z dimensions in array
                        return dim
                else:
                        print("\n\n Invalid input. Please answer 'yes'
                            or 'no'.\n\n")
                        return self.whole_wafer(input("Is the wafer
                            whole (Y/N):").strip().lower())

############## Extrinsic/Intrinsic ###################
```

```
        # Check if the wafer is extrinsic and determines its type #
        def extrinsic(self,extrin):
                if extrin in ["yes","y"]:
                        extrin = True
                        type = input("P-or-N-type-wafer?-").strip().
                            lower() # .lower() converts to lowercase
                        if type in ["p","p-type","p-type"]:
                                rho = float(input("Please-enter-the-
                                    resistiity-of-the-wafer-(ohm*cm)-at-
                                    300-K:-"))       # Resistivity from
                                    user (ohm*cm)
                                return "p",rho
                        else:
                                rho = float(input("Please-enter-the-
                                    resistivity-of-the-wafer-(ohm*cm)-at
                                    -300-K:-"))      # Resistivity from
                                    user (ohm*cm)
                                return "n",rho
                elif extrin in ["no","n"]:
                        extrin = False
                        rho = 2.3e5      # intrinsic wafer resistivity
                            from Sze (ohm*cm)
                        return "intrinsic", rho
                else:
                        print("\n\nInvalid-input.-Please-answer-'yes'-or
                            -'no'.\n\n")
                        return self.extrinsic(input("Extrinsic-wafer?-")
                            .strip().lower())

############## Initial Conditions ###################
        # Wafer Initial Conditions from user#
```

```
        def user_input_wafer(self):
                print("Please enter data about wafer:")
                whole = input("Is the wafer whole? (Y/N):").strip().
                    lower()
                dim = self.whole_wafer(whole)
                    # function returns dimensions as one value (whole
                    wafer) or an array (partial wafer)
                print("The wafer dimensions are:",dim)
                    # Prints dimensions
                cordim = input("Is that correct?")
                dim = self.cor_dim(cordim,dim,whole)
                    # Function to change data if needed

                extrin = input("Extrinsic wafer?").strip().lower()
                    # .strip() removes leading and trailing characters
                    from a string (like whitespace)
                type,rho = self.extrinsic(extrin)
                            # Function returns dopant type and its
                    resistivity (ohm*cm)
                d = float(input("Thickness of wafer (microns):")) *
                    10**(-4)    # convert to cm
                d_tol = float(input("Thickness tolerance (microns):"))
                    * 10**(-4)        # convert to cm

                print("\nThank you for your input.\n")
                return dim,whole,extrin,type,rho,d,d_tol          # return
                     statement so we can store the values given by the
                    user.

############## Doping Agent ###################
        # Doping Agent information from user #
```

```
    def user_input_dop_agent(self):
            dop_type = input("What type of diffusion agent are you
                using, 'N' or 'P' type?").strip().lower()
            # From user, get the molarity of impurity element
                depending on type.
            if dop_type in ["p","p-type","p type","ptype"]:
                    M = float(input("For p-type diffusion (boron
                        impurities), what is the molarity of boron
                        in the diffusion agent (mol/L)?"))
                    dop_type, M = self.cor_dop_agent(dop_type, M)
                        # Function to check with user if info is
                        correct
            elif dop_type in ["n","n-type","n type","ntype"]:
                    M = float(input("For n-type diffusion (
                        phosphorous impurities), what is the
                        molarity of phosphorus in the diffusion
                        agent (mol/L)?"))
                    dop_type, M = self.cor_dop_agent(dop_type, M)
                        # Function to check if info is correct with
                        user
            else:
                    print("\n\nInvalid input. Please enter 'n' or 'p
                        '.\n\n")
                    return self.user_input_dop_agent()

            if dop_type in ["p","p-type","p type","ptype"]:
                    dop_agent = "boron"
            else:
                    dop_agent = "phosphorus"

            print("\nThank you for your input.\n")
```

```
                return dop_type, M, dop_agent   # return diffusion agent
                    type and its molarity


############## Diffusion Parameters ###################
        # Diffusion parameters from user #
        def user_input_diff_unknown(self):
                # Specify possible known/unknown variables to user
                print("To characterize the diffusion process, there are
                    four crucial \
variables to consider: baking temperature (T, degree C), time (t, s), \
diffusion depth (x, microns), and dopant concentration (C, \
atoms/cm^3). One will be the unknown variable.")
                print("To specify the unknown variable to solve, please
                    indicate with \
the following number:")
                print("temperature = 1")
                print("time = 2")
                print("diffusion depth = 3")
                print("dopant concentration = 4")
                # Ask user what to solve for.
                unknown = float(input("Which variable is the unknown? ")
                    )
                # Now get other known variables from user.
                print("\nThank you. Please provide info for the known
                    variables.\n")
                if unknown == 1:          # Unknown Temperature
                        t = self.get_time()      # seconds
                        depth = self.get_depth()          # cm
                        C = self.get_C()          # atoms/cm^3
                        knowns = np.array([t,depth,C])
```

```
                return unknown,knowns
            elif unknown == 2:      # Unknown Time
                T = self.get_temp()
                depth = self.get_depth()
                C = self.get_C()
                knowns = np.array([T,depth,C])
                return unknown,knowns
            elif unknown == 3:      # Unknown Depth (x)
                T = self.get_temp()
                t = self.get_time()
                C = self.get_C()
                knowns = np.array([T,t,C])
                return unknown,knowns
            else:                   # Unkown Concentration
                T= self.get_temp()
                t = self.get_time()
                depth = self.get_depth()
                knowns = np.array([T,t,depth])
                return unknown,knowns

############## Temperature Input ###################
        # Get Temperature from User #
        # Function is call in diffusion file
        def get_temp(self):
            T = float(input("Temperature wafer is baked at (C):"))
            T += 273.15     # Convert to Kelvin
            return T

############## Depth Input ###################
        # Get Time from User #
        # Funciton is called in diffusion file
```

```
        def get_time(self):
                t = float(input("Time wafer will be baked (s): "))
                return t

############## Depth Input ###################
        # Get Depth from User #
        # Function is called in diffusion file
        def get_depth(self):
                x = float(input("Desired depth of dopant (microns): "))
                x = x*10**-4     # Convert microns to cm
                return x

############## Dopant Concentration Input ###################
        # Get Concentration from User #
        # Function is called in diffusion file
        def get_C(self):
                C = float(input("What is the desired dopant
                    concentration (atoms/cm^3)? Please give input in '1
                    e14' format: "))
                return C
```

Appendix C

Initial Concentration Code

The dop_initialc .py file is focused on calculating the initial dopant concentration of the silicon wafer. It takes values from the main file, dop_main.py, calculates the result, then directly outputs the result to the user.

```
#
    ##############################################################################

# File Name: dop_initialc.py
# Description: This file is a subset of the dop_main.py file. It takes
    the
# resistivity of the original wafer, whether instrinic or extrinsic, and
# calculates the dopant level.
#
# Overview:
# - Import Libraries and define class
# - Initial Conditions
# - Carrier Type Determination
# - Calulations:
#       - Mobility
#       - Resistiity
```

```
#       - Doping
# - Output Result to User
#
# Author: Makelle Forson (win22012@byui.edu)
# Date: 31 May 2025 (Updated 28 March 2026)
# Resources:
# - "Carrier Mobilities in Silicon Empirically Related to Doping and
    Field,"
#       D. M. Caughey and R. E. Thomas
#
   ##############################################################################

# Import Libraries #
import numpy as np
from scipy.optimize import root_scalar

class InitialC:
        # Initial Conditions #
        def __init__(self, wafer_type):
                self.type = wafer_type.lower()        # "intrinsic", "n",
                    or "p"
                self.q = 1.6e-19                      # C
                self.T = 300                          # K
                self.n_i = 1.45e10                    #  c m    (Sze &
                   Lee Appendix G value at 300 K)
                                                  # n_i value also matches
                                                       Misiakos' 1993 work
                                                      .
                # Mobility values for intrinsic silicon (from Sze & Lee)
                if self.type == "intrinsic":
```

```
                self.mu_n = 1450.0                  # electrons (
                    cm/V s)
                self.mu_p = 505.0                   # holes     (
                    cm/V s)
            elif self.type == "n":                  # n-type background
                (electrons dominant)
                self.mu_max, self.mu_min, self.alpha, self.N_ref
                    = 1330, 65, 0.72, 8.5e16
            else:                                   # p-type background
                (holes dominant)
                self.mu_max, self.mu_min, self.alpha, self.N_ref
                    = 495, 47.7, 0.76, 6.3e16

    # Extrinsic Mobility #
    def mu_calc(self, N):
            if self.type == "intrinsic":
                return None                 # not used for intrinsic
            mu = self.mu_min + (self.mu_max - self.mu_min) / (1 + (N
                / self.N_ref)**self.alpha)
            return mu

    # Resistivity #
    def rho_calc(self, N_net):
            """N_net = net doping = n - p (positive = n-type
                background)"""
            if self.type == "intrinsic":
                    # Full two-carrier formula for near-intrinsic
                        material
                    n = (N_net + np.sqrt(N_net**2 + 4*self.n_i**2))
                        / 2     # electrons
                    p = self.n_i**2 / n     # holess
```

```
                sigma = self.q * (n * self.mu_n + p * self.mu_p)
                                # conductivity
                rho = 1.0 /sigma
        else:
                # Extrinsic - Caughey-Thomas Empirical
                    Calculation
                mu = self.mu_calc(abs(N_net))
                rho = 1.0 / (self.q * mu * abs(N_net))

        return rho      # returns resistivity (ohm*cm)

# Root-Finding #
def root(self, N_net):
        return self.rho_calc(N_net) - self.rho_target

def estimateDoping(self, rho_target):
        self.rho_target = rho_target

        if self.type == "intrinsic":
                # Calculate the EXACT theoretical intrinsic
                    resistivity from the model
                rho_i = 1.0 / (self.q * self.n_i * (self.mu_n +
                    self.mu_p))
                #print(f"DEBUG: Theoretical intrinsic    = {
                    rho_i:.1e} ohm cm")
                #print(f"DEBUG: User entered    = {rho_target:.1
                    e} ohm cm")

                # If the users measured rho is close to
                    intrinsic, return zero net doping
```

```
            if abs(rho_target - rho_i) < 10000:   # generous
                10 ,000 tolerance
                    return 0.0   # net doping is zero

            # Otherwise treat it as very lightly doped
            bracket = [-1e16, 1e16]
    else:
            bracket = [1e11, 1e22]

    fa = self.root(bracket[0])
    fb = self.root(bracket[1])

    if fa * fb > 0:
            raise ValueError(
                    f"Function does not change sign over
                        bracket [{bracket[0]:.1e}, {bracket
                        [1]:.1e}]. "
                    f"f(a) = {fa:.2e}, f(b) = {fb:.2e}. "
                    f"Check your measured resistivity (
                        entered {rho_target:.1e} ohm cm)." )

    result = root_scalar(self.root, bracket=bracket, method=
        "brentq", xtol=1e-12)

    if result.converged:
            dop_conc = abs(result.root)
            return dop_conc
    else:
            raise RuntimeError("Root finding did not
                converge.")
```

```
# Print #
    def print_dop_calc(self, dop_conc):
            if self.type == "intrinsic":
                    level = "intrinsic (very lightly doped)"
                    print(f"The wafer is intrinsic at {self.T} K.\n"
                            f"Estimated net doping = {dop_conc:.3e}
                                atoms/cm "
                            f"(essentially zero for practical
                                purposes).\n")
                    return

            # Extrinsic cases
            if dop_conc < 1e14:
                    level = "intrinsicly doped"
            elif dop_conc < 1e16:
                    level = "lightly doped"
            elif dop_conc < 1e18:
                    level = "normally doped"
            elif dop_conc < 1e20:
                    level = "heavily doped"
            else:
                    level = "degenerately doped"

            print(f"The estimated doping concentration at {self.T} K
                is {dop_conc:.3e} atoms/cm "
                    f"({self.type}-type background)    wafer is {
                        level}.\n")
```

Appendix D

Diffusion Code

This code contains majority of the program's calculations. The equations used in this code came from Sze's work[1].

```
##########################################################################
# File Name: dop_diffusion.py
# Description: This file is part of dop_main.py that calculates the
# unkown variable from the user during the diffusion process. This
# unknown varialbe could be temperature, time, depth, or dopant
# concentration. The result is returned to the user.
#
# Overview:
# - Import Libraries and Define Class
# - Checks Dopant Agent Used
# - Calculations:
#       - Initial Surface Concentration
#       - Dopant Atoms per Area
#       - Diffusivity
#       - Temperature
#       - Time
#       - Depth
```

```
#       - Dopant Concentration
#
# Author: Makelle Forson (win22012@byui.edu)
# Date: 10 June 2025 (Updated 3 April 2026)
# Resources: "Semiconductor Devices: Physics and Technology," by S. M.
    Sze and M. K. Lee
##########################################################################
# Import Libraries #
import numpy as np
from matplotlib import pyplot as plt
from scipy.constants import N_A,e,k      # avogadro, elementary charge,
    boltzmann (J/K)
from scipy.special import lambertw       # Used to calc T and t
from scipy.optimize import fsolve        # Used to calc T

class Diffusion:
        # Initial Conditions #
        def __init__(self,mol,dop_agent,d,init_C):
                self.mol = mol                  # Molarity from user (
                    mol/L)
                self.dop_agent = dop_agent      # dopant agent from user
                     (string)
                self.d = d                      # wafer thickness (cm)
                self.film_thick = 1e-5          # Average film thickness
                     for spin-on dopant (cm)
                self.depth_lim = self.d         # diffusion depth limit
                    (cm)
                self.k = 8.617e-5               # Boltzmann constant (eV
                    /K)
                self.init_C = init_C            # initial Concentration
                    from user (atoms/cm^3)
```

```
                # Arrays #
                self.depth = np.array([])       # (cm)
                self.time = np.array([])        # (s)

        ### Don't need this function anymore ###
        # Function to check the dopant agent and assign the correct D_0
            and E_a
        # for diffusion through silicon dioxide (SiO2).
#       def check_agent(self):
#               if self.dop_agent == "boron":
#                       D_0 = 3e-4              # Boron Diffusion
    coefficient at infinite temp. (cm^2/s)
#                       E_a = 3.53              # Activation energy for
    boron (eV)
#
#               else:
#                       D_0 = 0.19              # Phos. Diffusion coeff
    at infinite temp (cm^2/s).
#                       E_a = 4.03              # Activation energy for
    phosphorus (eV)
#
#               return D_0,E_a

        # Initial Surface Concentration Calculation (atoms/cm^3) #
        def C_S_calc(self,S,D,t):
                C_S = S / np.sqrt(np.pi * D * t)
                return C_S              # Tested and it works!

        # Dopant atoms per area calculation (atoms/cm^2) #
        def S_calc(self):
```

```
                S = self.mol * N_A * self.film_thick / 1000      #
                    Calculates the 'dose' (atoms/cm^2)
                return S         # Tested and it works!!

        # Calculate the Diffusivity (Fair model) - uses LOCAL
            concentration
        def D_calc(self, T, local_p):
                """local_p = local dopant concentration C(x,t) in atoms/
                    cm
^^^^^^^^ ^^^^^^^^(For analytical solutions we use C_S as the
    representative value)"""
                self.n_i = 5.29e19 * (T/300)**2.54 * np.exp(-6726/T)

                if self.dop_agent == "boron":
                        D0    = 0.037 * np.exp(-3.46 / (self.k * T))
                        Dplus = 0.76  * np.exp(-3.46 / (self.k * T))
                        D = D0 + Dplus * (local_p / self.n_i)
                else:  # phosphorus
                        D0  = 3.85 * np.exp(-3.66 / (self.k * T))
                        Dmm = 44.2 * np.exp(-4.37 / (self.k * T))
                        D = D0 + Dmm * (local_p / self.n_i)**2

                return D

        # Temperature Calculation #
        def calc_temp(self, knowns):
                t = knowns[0]       # time (s)
                x = knowns[1]       # depth (cm)
                C = knowns[2]       # target concentration at (x,t)  (
                    atoms/cm )
```

```
S = self.S_calc()  # dose (atoms/cm )

# Compute effective diffusivity D_eff from the Gaussian
    profile
u = (-C**2 * x**2 * np.pi) / (2 * S**2)

# Safeguard: clamp u so it stays inside the real domain
    of W_{-1}
u_min = -1.0 / np.e
if u < u_min:
        print(f"WARNING: u = {u:.3e} is below -1/e. "
              f"Requested (x,t,C) is physically
                  impossible for this dose.")
        u = u_min * 0.999999  # tiny nudge so Lambert W
            still works

w = lambertw(u, k=-1).real          # .real forces real
    output

D_eff = -x**2 / (2 * t * w)         # now guaranteed
    real

# Solve for T such that D_model(T, local concentration =
    C) == D_eff
def D_model(T):
        print(f"In D_model, self.dop_agent is {self.
            dop_agent}")
        ni = 5.29e19 * (T/300)**2.54 * np.exp(-6726/T)
        if self.dop_agent == "boron":
                D0     = 0.037 * np.exp(-3.46 / (self.k *
                    T))
```

```
                    Dplus = 0.76   * np.exp(-3.46 / (self.k *
                        T))
                    return D0 + Dplus * (C / ni)     # use
                        local C, not S!
                else:  # phosphorus
                    D0   = 3.85 * np.exp(-3.66 / (self.k * T)
                        )
                    Dmm = 44.2 * np.exp(-4.37 / (self.k * T)
                        )
                    return D0 + Dmm * (C / ni)**2

            def objective(T):
                return D_model(T) - D_eff

            # Solve
            T_guess = 1273.0   # 1000 C      good starting point
            T_sol = fsolve(objective, T_guess, xtol=1e-10)[0]
            return T_sol, S, D_eff


    # Time Calculation
    def calc_time(self, knowns):
            T = knowns[0]   # temp (K)
            x = knowns[1]   # depth (cm)
            C = knowns[2]   # target concentration at (x,t)  (atoms/
                cm )

            S = self.S_calc()

            # First get approximate surface concentration using a
                dummy D
```

```
            dummy_D = 1e-12
            C_S = self.C_S_calc(S, dummy_D, 1.0)    # dummy time=1 s

            # Now compute real D using surface concentration as
                representative local_p
            D = self.D_calc(T, C_S)

            # Re-calculate real C_S with correct D
            C_S = self.C_S_calc(S, D, 1.0)    # still dummy time for
                C_S

            u = (-np.pi * C**2 * x**2) / (2 * S**2)
            w = lambertw(u, k=1).real
            t = -x**2 / (2 * D * w)

            return t, S, D

    # Depth Calculation
    def calc_depth(self, knowns):
            T = knowns[0]    # temp (K)
            t = knowns[1]    # time (s)
            C = knowns[2]    # target concentration at (x,t)  (atoms/
                cm )

            S = self.S_calc()

            # First get approximate surface concentration
            dummy_D = 1e-12
            C_S = self.C_S_calc(S, dummy_D, t)

            # Now compute real D using surface concentration
```

```
            D = self.D_calc(T, C_S)

            # Re-calculate real C_S with correct D
            C_S = self.C_S_calc(S, D, t)

            x = 2 * (-D * t * np.log(C / C_S))**(1/2)

            return x, S, D   # returns x in cm

    # Dopant Concentration Calculation
    def calc_depth_profile(self, knowns):
            T = knowns[0]    # Temp (K)
            t = knowns[1]    # time (s)
            x = knowns[2]    # depth (cm)

            S = self.S_calc()

            # First get approximate surface concentration
            dummy_D = 1e-12
            C_S = self.C_S_calc(S, dummy_D, t)

            # Now compute real D using surface concentration
            D = self.D_calc(T, C_S)

            # Re-calculate real C_S with correct D
            C_S = self.C_S_calc(S, D, t)

            self.C = C_S * np.exp(-x**2 / (4 * D * t))
            return self.C, S, D   # (atoms/cm )

    def calc_final_C(self,C,init_type):
```

```
# Final Concentration Calculation
# If type and doping agent are the same
print(f"Dope agent is {self.dop_agent}")
if self.dop_agent == "phosphorus":
        dop_agent_type = "n"
else:   # boron
        dop_agent_type = "p"

#print(f"The dop agent type is {dop_agent_type}")

if init_type == dop_agent_type:
        C_f = self.init_C + C
        new_type = init_type
# If wafer was intrinsic
elif init_type == "intrinsic":
        C_f = C
        new_type = dop_agent_type
        #print(f"Thus the new type is {new_type}")
# If type and doping agent are different
else:
        #print(f"The initial dopant and new dopant have
            different types. Initial type = {init_type
            }")
        C_f = self.init_C - C
        # If sign change, then it also chang4es what
            type of doping is
        # predominant.
        if C_f < 0:
                if init_type == "n":
                        new_type = "p"
                else:
```

```
                    new_type = "n"
            else:
                new_type = init_type
            #print(f"The new type is {new_type}.")
    return abs(C_f),new_type
```

Appendix E

Plotting Code

The last file in the dopant diffusion program produces plots based on the values collected and calculated.

```
##########################################################################
# File Name: dop_plots.py
# Description: This file goes with the dop_main.py file. It
# contains code to plot various plots for the diffusing
# calculations.
#
# Overview:
# - Initial Conditions
# - Dopant Concentration (C) vs Depth (x)
# - Resistivity (rho) vs Dopant Concentration (C)
#
# Author: Makelle Forson
# Date: 7 July 2025 (Updated 8 April 2026)
##########################################################################
# Libraries #
import numpy as np
from matplotlib import pyplot as plt
```

```
from dop_initialc import InitialC
from dop_diffusion import Diffusion

class DopePlots:

        # Initial Conditions #
        def __init__(self):
                pass
        # Plot Depth Profile (C) vs depth at various times #
        def plot_C_vs_x(self,mol,dop_agent,d,init_C,T,t,x,C_f,init_type,
            new_type):
                # x from dop_main is in cm
                DF = Diffusion(mol,dop_agent,d,init_C)

                S = DF.S_calc()          # atoms per area (cm^-2)
                D = DF.D_calc(T,S)       # Diffusivity (cm^2/s)

                time = [.1,60,225,450,900,1800,3600,7200]        # time (
                    s)
                depth_range = np.linspace(0,1,1000)      # Goes to a
                    depth of 1 microns (microns)
                init_C = [init_C]*len(depth_range)               #
                    Initial dopant concentration times length of depth
                    range for plotting

                plt.figure()
                # Loop through depth_range with different times,
                    calculating depth profile
                for t in time:
                        C_S = DF.C_S_calc(S,D,t)
```

```
                    C = C_S * np.exp(-(depth_range*1e-4)**2 / 4 / D
                        / t)
                    # Plot depth in microns
                    plt.plot(depth_range,C, label = f"t = {t/60:.2f}
                        minutes")

                plt.plot(depth_range,init_C, "--", label = f"Initial
                    Concentration, type: {init_type}")
#               plt.plot(x*1e4,C_f, "+", label = f"Final Concentration,
    type: {new_type}")      # x (cm) was converted to microns for
    plotting
                plt.title(f"Dopant Density at {T:.0f} K")
                plt.xlabel("Depth (microns)")
                plt.ylabel("Dopant Density (atoms/cm^3)")
#               plt.xlim([np.min(depth_range),np.max(depth_range)])
                plt.ylim([1e12,2*1e23])
                plt.legend()
                plt.grid(True, which = "both", ls = "-", alpha = .65)
                plt.yscale("log")
                # Horizontal Shading for Doping Levels #
                plt.axhspan(0,1*10e13, color = "orchid", alpha = .2,
                    label = "Intrinsic")                    #
                    Intrinsic
                plt.axhspan(1*10e13,1*10e15, color = "darkorchid", alpha
                    = .2, label = "Light Doping")           # Light
                    Doping
                plt.axhspan(1*10e15,1*10e17, color = "darkslateblue",
                    alpha = .2, label = "Normal Doping")    # Normal
                    Doping
                plt.axhspan(1*10e17,1*10e19, color = "blue", alpha = .2,
                     label = "Heavy Doping")                # Heavy
```

```
            Doping
        plt.axhspan(1*10e19,10e29, color = "darkblue", alpha =
            .2, label = "Degenerate Doping") # Degenerate Doping
        plt.show()


# Plot Resistivity vs Dopant Concentration for P and N type
    Dopings #
# This corresponds to the dop_hall_effect.py file where it looks
    at the wafer before being doped
def plot_rho_vs_conc(self,init_conc,init_rho):
        # resistivity values are based off of plot in Bro John's
            book
        rho_values = np.logspace(-3,4,200)      # .001 to 10000
            ohm*cm (log scale)
        plt.figure()

        for dop_type in ["p","n"]:              # Cycle through
            both types of dopant
                dop_conc = []                   # dopant
                    concentration with each rho
                for rho in rho_values:
                        INC = InitialC(dop_type)
                        try:
                                N = INC.estimateDoping(rho)
                                dop_conc.append(N)
                        except Exception as e:
                                dop_conc.append(np.nan)
                                    # skip values that fail to
                                    converge
```

```
            dop_conc = np.array(dop_conc)
            plt.loglog(dop_conc,rho_values, label = f"{
                dop_type}-type silicon")

    # Plotting Details
    plt.plot(init_conc,init_rho,'+',label = "Initial Wafer")
                # Wafer's initial resistivity
    plt.xlabel("Dopant Concentraion (atoms/cm^3)")
    plt.ylabel("Resistivity (ohm*cm)")
    plt.title("Resistivity vs Doping Concentration at 300 K"
        )
    plt.grid(True, which = "both", ls = "-", alpha = .65)
    plt.xlim(10e10,5*10e19)
    # Vertical shading to show doping levels
    plt.axvspan(0,1*10e13, color = "orchid", alpha = .2,
        label = "Intrinsic")        # Intrinsic
    plt.axvspan(1*10e13,1*10e15, color = "darkorchid", alpha
         = .2, label = "Light Doping")  # Light Doping
    plt.axvspan(1*10e15,1*10e17, color = "darkslateblue",
        alpha = .2, label = "Normal Doping")       # Normal
        Doping
    plt.axvspan(1*10e17,1*10e19, color = "blue", alpha = .2,
         label = "Heavy Doping")          # Heavy Doping
    plt.axvspan(1*10e19,5*10e19, color = "darkblue", alpha =
         .2, label = "Degenerate Doping")        # Degenerate
         Doping
    plt.legend()
    plt.show()
```

Appendix F

Spin-On Dopant SOP

This Standard Operating Procedure (SOP) provides details, safety considerations, equipment, and the method needed to procure n-type spin-on dopant solution. This solution is needed to dope a silicon wafer through diffusion.

BYU IDAHO	Production of Phosphorus Dopant Mixture for Semiconductor Wafers	SOP #	2025 - Seare
		Revision #	4
		Implementation Date	3/27/2025
Page #	1 of 3	Last Reviewed/Update Date	3/26/2025
SOP Owner	Bryan Seare	Approval	

Standard Operating Procedure

1. Purpose

Synthesis of a solution containing glass polymer imbued with phosphorus (spin-on dopant) to infuse into silicon semiconductor wafers. The shelf life of the resulting solution is only appx. 3 months.

2. Scope

This SOP is intended **for chemists to use** aiding the semiconductor club/group connected to the Physics Department. **No physics student should ever attempt this procedure,** unless they have the chemical experience(s) defined in personnel qualifications (section 4, below).

3. Definitions

TEOS – Tetraethyl orthosilicate

4. Personnel Qualifications

Primary Contacts for this SOP: Bryan Seare: sea17015@byui.edu

Personnel performing this SOP should be supervised by a faculty member. Completion of at least one semester of organic chemistry lab is **required**, 2 semesters is preferred.

5. Safety Considerations

- Perform this procedure and handle TEOS under a fume hood **only**, as inhalation of TEOS fumes is highly toxic and causes severe eye irritation. No open flames should be present in the lab.
- Wearing gloves is **required** while handling 85% Phosphoric acid. Disregarding glove use may result in severe burns.
- **Only** handle synthesized solution using gloves, as it may still contain acid. The synthesized solution may cause glass polymer to form on skin, the effects of which are not known.
- Read all safety data sheets for all chemicals, prior to lab work.
- Wear standard PPE, including a lab coat, rubber gloves, and safety goggles at all times (glasses do not count!).
- Use of a face shield is **required** when handling TEOS.

6. Equipment and Supplies

- 85% Phosphoric acid
- Tetraethyl orthosilicate
- Pure food grade ethanol – 90% or better purity
- Deionized water
- Reflux condenser
- 1 10-50 mL round-bottom flask
- Plastic conical joint clip (one that fits the condenser joint and round-bottom flask.)
- Heating mantle
- Magnetic stir bar

- Stand with 2 clamps
- 1000 microliter micropipette
- Micropipette tips
- 2 rubber tubes
- 3 small plastic storage bottles with caps
- 3 50mL beakers
- 1 300mL beaker
- 1 10mL graduated cylinder
- about 3 plastic pipettes
- 30 minute timer (or your phone)
- Glass joint grease

7. Interferences

If the solution is refluxed for too long, a gel could form. The solution is unusable as a spin-on dopant at this point and should be safely disposed of.

8. Procedure.

1. in a fume hood, carefully pour about 3 mL of TEOS into a small plastic storage bottle that can be capped. Cap this storage bottle.

2. Set up a heating mantle in the fume hood, leaving the heat off. Clamp the round-bottom flask to a stand in the fume hood, letting the flask rest on the heating mantle. Insert the magnetic stir rod into the round-bottom flask.

3. In the fume hood, attach 2 rubber tubes to the reflux condenser. Attach the lower tube to the water faucet in the fume hood and place the top tube end into the water drain. Clamp the reflux condenser to the stand about 5 inches above the round-bottom flask. Apply a thin film of vacuum grease to the ground glass joint of the condenser.

4. Steps 5 -14 must be done **only** in the fume hood.

5. Using a graduated cylinder, add 4mL of food grade ethanol to the round-bottom flask.

6. Using a micropipette, add 1mL deionized water, 0.5 mL 85% phosphoric acid, and 1 mL of TEOS (TEOS from the plastic bottle in step 1) all to the round-bottom flask. Use new tips for each transfer. Discard used tips into a plastic container and cap the container.

- If a spill of TEOS occurs, contact the lab supervisor.

7. Carefully turn on water to the reflux condenser and increase the flow until a medium-high water flow is obtained. Eliminate any bubbles that appear.

8.Carefully lower and attach the reflux condenser to the round-bottom flask. Using a plastic clip, secure the condenser to the round bottom flask. The reflux setup is now complete.

9. Turn on spin setting on the heating mantle so the spin bar rotates 2-3 times per second.

BYU IDAHO	Production of Phosphorus Dopant Mixture for Semiconductor Wafers	SOP #	2025 - Seare
		Revision #	4
		Implementation Date	3/27/2025
Page #	3 of 3	Last Reviewed/Update Date	3/26/2025
SOP Owner	Bryan Seare	Approval	

10. Turn on the heating mantle to a medium heat and wait until the solution just starts to boil to start reaction/reflux. If the solution doesn't boil, adjust temperature until it does.

11. Using a timer, lightly boil the solution for no more than 30 minutes. During this period, do not increase the heat setting of the mantle. Ideally it should be just hot enough to keep the solution boiling.

12. Turn off the heating mantle and raise the reflux setup off the heating mantle. Let the setup sit for 10 minutes, allowing the solution to cool.

13. Stop water flow to the reflux condenser. Disconnect the reflux condenser from the round-bottom flask.

14. Using a plastic pipette, carefully extract the boiled solution into a small plastic bottle. Cap the plastic bottle.

15. Rinse all glassware that encountered TEOS with ethanol in a fume hood (denatured or food grade), discarding all used ethanol into a 300mL waste beaker. This is done to attempt to stop any glass formation on the walls of the round-bottom flask and reflux condenser. Safely dispose of waste contents in a proper waste storage container.

16. Wash/rinse all glassware with acetone/soap. Discard the micropipette tips and plastic pipettes in a manner instructed by the lab supervisor. Clean up any chemical spills as needed. Wipe off all shelves, including the fume hood.