NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPTIMIZATION OF MULTI-JUNCTION SOLAR CELLS FOR SPACE APPLICATIONS MODELED WITH SILVACO ATLAS

by

James S. Walsh

June 2018

Thesis Advisor:	Sherif N. Michael
Second Reader:	Matthew A. Porter

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE OPTIMIZATION OF MULTI-JUNCTION SOLAR CELLS FOR SPACE APPLICATIONS MODELED WITH SILVACO ATLAS	**5. FUNDING NUMBERS**
6. AUTHOR(S) James S. Walsh	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Dual junction solar cells are used in space applications for their high efficiency. In this thesis, we model an indium gallium phosphide/gallium arsenide dual-junction solar cell. The solar cell is modeled using Silvaco ATLAS software. Solar cell layer thicknesses and doping concentrations were varied to find optimum efficiency parameters for the solar cell under a variety of radiation conditions. These radiation conditions mimic the damage done at various orbits around Earth for an arbitrary mission length of 12 years. The optimization process resulted in an improved efficiency of 15.1% to 22.4%.

14. SUBJECT TERMS dual junction, solar cell, optimization, NOLH, Silvaco, radiation	**15. NUMBER OF PAGES** 139
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

OPTIMIZATION OF MULTI-JUNCTION SOLAR CELLS FOR SPACE APPLICATIONS MODELED WITH SILVACO ATLAS

James S. Walsh
Lieutenant, United States Navy

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2018

Approved by: Sherif N. Michael
Advisor

Matthew A. Porter
Second Reader

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Dual junction solar cells are used in space applications for their high efficiency. In this thesis, we model an indium gallium phosphide/gallium arsenide dual-junction solar cell. The solar cell is modeled using Silvaco ATLAS software. Solar cell layer thicknesses and doping concentrations were varied to find optimum efficiency parameters for the solar cell under a variety of radiation conditions. These radiation conditions mimic the damage done at various orbits around Earth for an arbitrary mission length of 12 years. The optimization process resulted in an improved efficiency of 15.1% to 22.4%.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AlAs	aluminum arsenide
AlGaAs	aluminum gallium arsenide
AlInP	aluminum indium phosphide
AlP	aluminum phosphide
GaAs	gallium arsenide
GaP	gallium phosphide
Ge	germanium
GEO	geo-synchronous orbit
InAlGaP	indium aluminum gallium phosphide
InGaP	indium gallium phosphide
InP	indium phosphide
IV	current-voltage
LEO	low-earth orbit
NIEL	non-ionizing energy loss
NPS	Naval Postgraduate School

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to show appreciation for the Naval Postgraduate School staff members who helped me in my research efforts: my advisor Professor Sherif Michael, my second reader Mathew Porter, the Research Computing Lab Manager Ray Chatten, the Hamming Systems Architect Bruce Chiarelli, and Professor Tomas Lucas in the Operations Research Department.

I would also like to thank my children, Austin, Ainsley, and Addison, for their support and patience during this time.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. SOLAR CELLS FOR SPACE APPLICATIONS

The ability to provide consistent, reliable power to space based systems is of vital importance. The cost of launching fuel or batteries from Earth into orbit represents a huge financial burden that can hinder any space-based mission. Currently, solar cells, which are much lighter than chemical fuel, are used to provide power to orbiting systems. The cost associated with producing state-of-the-art multi-junction solar cells generally prohibits their use for terrestrial applications; this is not true for space applications. The relatively high cost of putting any cell into space dictates that the additional cost of a multi-junction solar cell relative to a single-junction solar cell is immaterial compared to the savings in weight. Improved efficiency in solar panel design will lead to a lower weight for power generation, a lower surface area exposed to orbital debris, and the ability to utilize equipment with higher power demands.

The nonlinear interaction of solar cell parameters such as doping levels, layer thicknesses, and material composition make finding optimal cell parameters a non-trivial task. Sophisticated computer software, using numerical approximation to solve a nonlinear set of differential equations, which model solar cell operation must be used to simulate the efficiency of a specific solar cell design with specific parameters. Further improvement can be achieved by direct simulation of solar cells with variations of the parameters. Analyzing a large number of simulation results gives a picture of what parameter values lead to the highest efficiency cell at a fraction of the cost of manufacturing myriad solar cells with different characteristics.

For a space application, radiation damage to the cell and its effect on power generation must be considered. In this work, end-of-life efficiency, or the efficiency after a certain amount of time exposed to radiation determined by orbit, is the parameter around which the solar cell is optimized. It is generally assumed that the most efficient cell before being irradiated remains the most efficient cell after irradiation. To the best of the author's

knowledge, this thesis research is the first optimization to account for radiation damage and test that theory.

B. PAST WORK AT NPS

Several theses have examined solar cell optimization at the Naval Postgraduate School (NPS). Panayiotis Michalopoulos [1] successfully modeled several solar cells using Silvaco ATLAS (Silvaco) in 2003. This research consisted mainly of demonstrating good agreement between modeled cells and tested cells, as well as finding theoretical optimal parameters based on the physics driving solar cell efficiency. In 2017, both Raymond Kilway [2] and Silvio Pueschel [3] continued this work. They showed that solar cells could be optimized by simulation, utilizing either a genetic algorithm or nearly orthogonal Latin hypercubes. The latter method proved to be superior and is the method utilized in this thesis.

C. OBJECTIVE

The goal of this research is four fold. Firstly, a solar cell with known parameters and output characteristics is modeled to verify simulation results match real-world data. Secondly, a radiation model is applied to the solar cell to simulate the damage done to the cell while in orbit around the Earth. Thirdly, a set of parameters is determined by using optimization software and tools to maximize cell efficiency at the end of life. Finally, a set of generic tools is created to facilitate duplication of the optimization and modeling methods that can be used with any arbitrary solar cell and with any arbitrary parameters, including those parameters not optimized in this research (such as molar composition).

The solar cell modeled is a dual-junction indium gallium phosphate (InGaP) / gallium arsenide (GaAs) cell fabricated and tested at Ohio State University [4]. A triple-junction or even a five-junction solar cell gives better efficiency and offers more opportunities for optimization, but in-depth data for these advanced cells are not available for research. Meeting the first goal of having a realistic simulation requires a thoroughly documented cell be used.

D. ORGANIZATION

In Chapter II, we cover the physics of semiconductors, solar cells, and radiation damage as well as the theory behind the optimization techniques used. Chapter III is dedicated to methodology: Silvaco Atlas and its many dependent files, radiation modeling, and optimization tools. Results and conclusions are discussed in Chapters IV and V, respectively. Areas of future improvements are discussed both in applicable sections and as a collection of recommended future work in Chapter V.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND THEORY

A. SEMICONDUCTORS

All electronic fundamentals rely on the concept of energetically restricted electrons. The allowed energy levels of electrons can be grouped into bands with bandgap E_g, or large swath of forbidden energy levels, separating them. The band of energies higher than the bandgap is called the conduction band, and the band below is called the valence band. The absence of electrons in allowed states in the valence and are called holes and treated as particles with a positive charge and mass similar to the mass of an electron. A simplified band diagram of a semiconductor is shown in Figure 1. Notice that some electrons already occupy the conduction band. This is because at thermal equilibrium, some electrons naturally have a large enough energy (equal or greater than the bandgap energy) to exist in the conduction band.

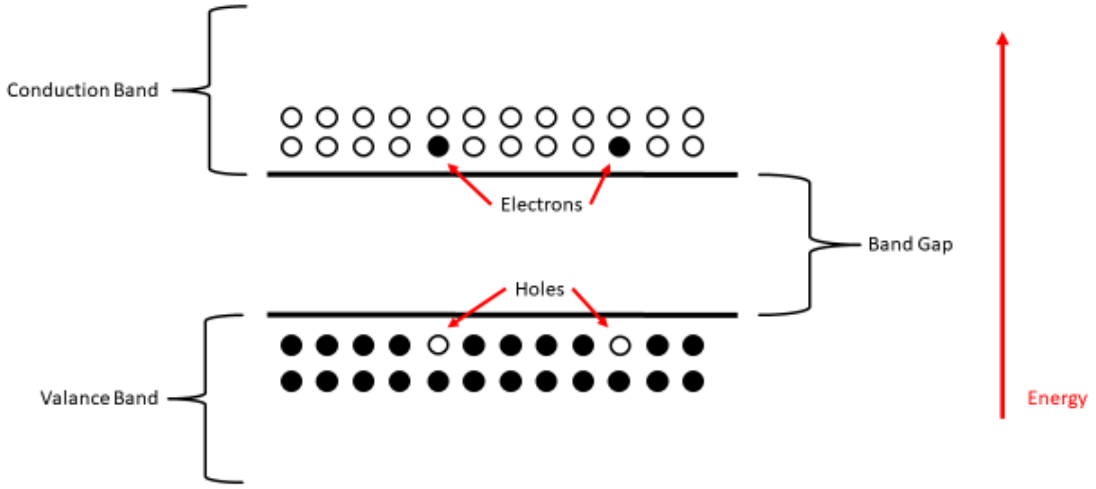


Figure 1. Simplified Bandgap Structure for a Semiconductor Material. Adapted from [5].

In a real semiconductor, energy is not the only parameter that restricts the state of electrons. A momentum **k** also defines the shape of the valence and conduction bands. In a direct bandgap material, the lowest energy point of the conduction band shares a momentum with

the highest energy point on the valence band. In an indirect bandgap material, there is a momentum offset between these points. Both types of bandgap materials are depicted in Figure 2.

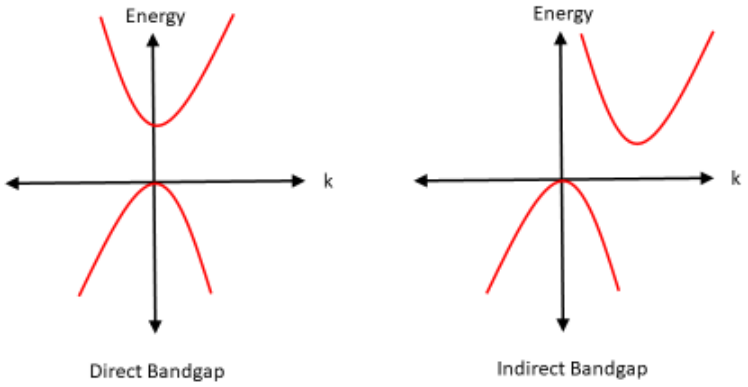


Figure 2. Two-Dimensional Representation of Direct and Indirect Bandgaps

This two dimensional structure is much more complicated in real life as compared to Figure 2, with GaAs, a direct bandgap material, having the structure shown in Figure 3. This complicated band structure gives rise to a highly energy dependent affinity for certain wavelengths (or energy) photons in a material. This property of semiconductors is explored further in Chapter III, Section C.

The curvature of these energy bands in relation to momentum leads to a phenomenon that greater incremental additional energy is required to give the same increase in momentum. To simplify the equations that govern semiconductor behavior, this is accounted for by using an *effective mass* for electrons and holes. This is an additional fraction of the rest mass of an electron, given as a unitless coefficient.

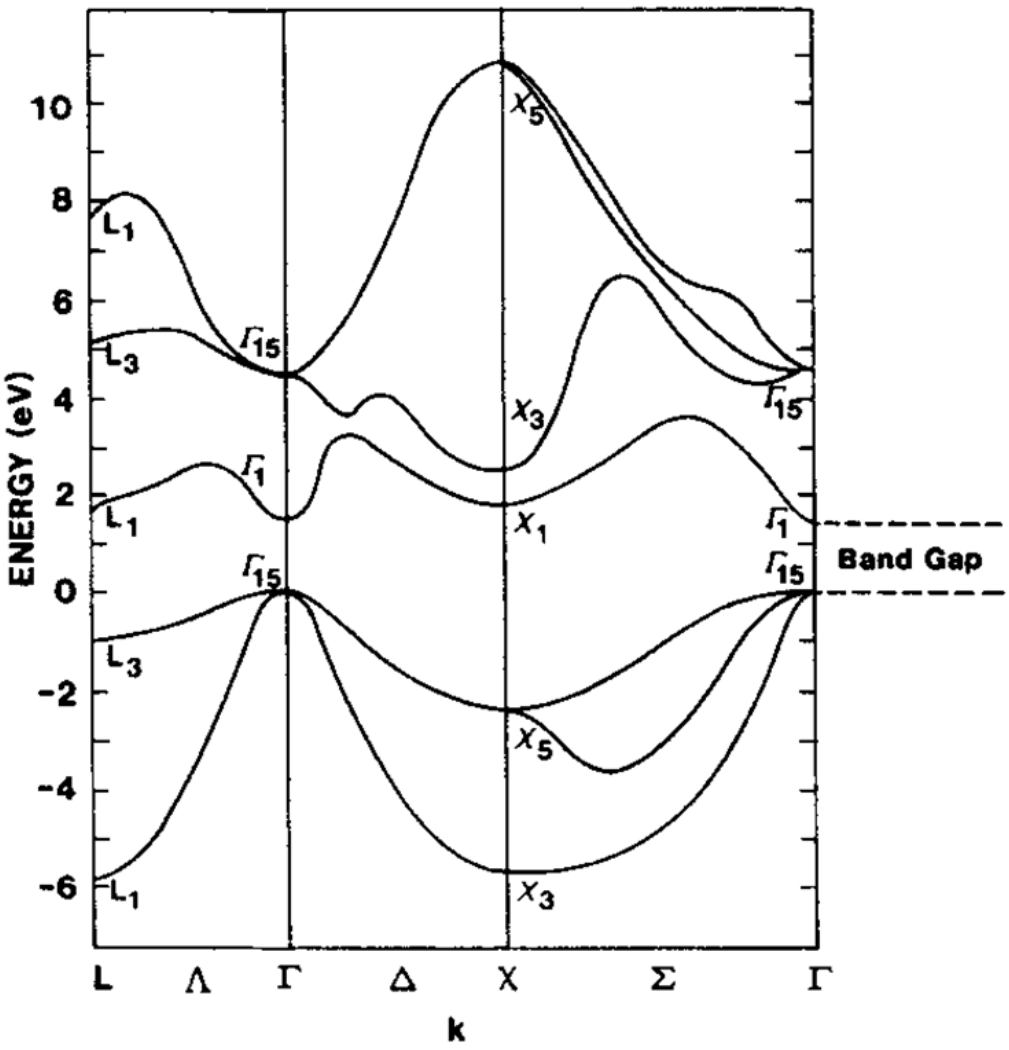


Figure 3. Real Band Diagram for GaAs. Source: [6].

1. Generation Rate and Recombination

An electron can be promoted from the valence band to the conduction band if it receives the correct amount of energy and momentum. This momentum is provided in the form of phonons, which are packets of acoustic momentum traveling though the bonds between atoms in the lattice. These phonons are naturally occurring at all values of momentum, leaving the only consequence of an indirect bandgap being that only a portion of the valence electrons that receive the correct amount of energy are promoted to the conduction band. An intrinsic amount n_i of free, or conduction band, electrons exist in a

pure semiconductor due to energy available due to temperature of the material. Energy added to the material creates more of these free electrons. This energy can be in the form of increased temperature, electric field, or electromagnetic packets of energy called photons. Each promoted electron also leaves behind a hole; a one-dimensional representation of this is shown in Figure 4.

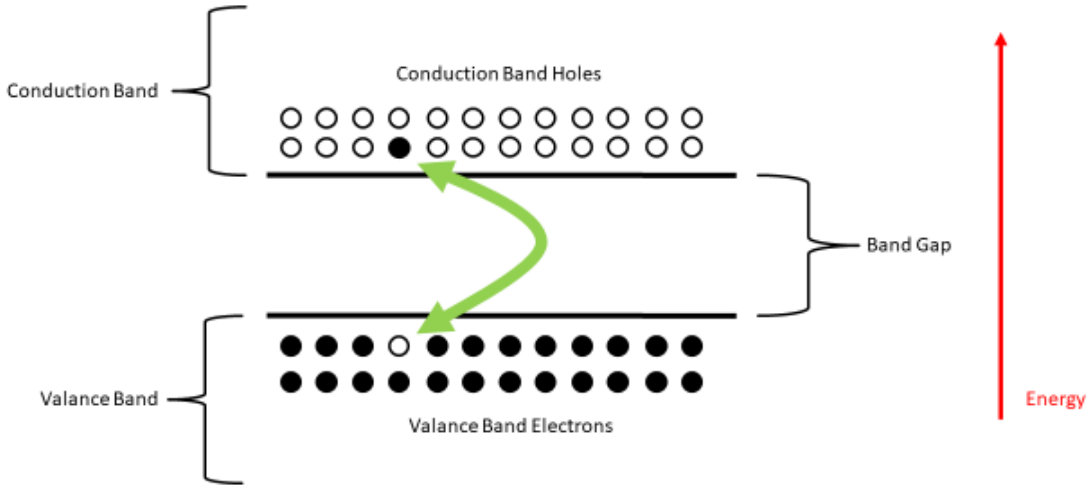


Figure 4. Electron Promoted to Conduction Band. Adapted from [5].

The creation of free electron and hole pairs is not a static event. Electron-hole pairs are continuously created as a function of time, and electrons and holes are continuously combining as electrons lose energy as a function of time. The intrinsic level of free electrons is the steady-state value across time. The generation and recombination of these electron hole pairs is of critical importance for solar cells.

2. Semiconductor Materials and Doping

With the properties of semiconductors established, we now explore which types of materials display these properties. Group IV elements, like silicon and germanium, are natural semiconductors. Group III-Group V compounds, like aluminum arsenide or gallium

arsenide, also display these properties. Group II-Group VI compounds display these properties as well, though none are explored in this thesis.

Doping is the process of deliberately adding impurities to a semiconductor. While these impurities may become interstitial defects in the lattice structure, the more common outcome is that they replace an atom within the lattice. If a dopant is a Group III or Group V element, it does not result in a lattice site with additional or fewer electron bonds with its neighbor atoms, but rather the bond is the same as with the semiconductor, with the extra hole or electron becoming free. For example, adding a Group V element, such as arsenide, to silicon at a level at 10^{16} impurities per cubic centimeter (cm^{-3}), gives additional electrons much more numerous than the intrinsic level of electrons in pure silicon at room temperature $\left(n_i = 10^{10}\,\mathrm{cm}^{-3}\right)$. Adding these two together gives the result of $\sim 10^{16}$ electrons per cm^{-3}; essentially, the concentration of holes or electrons is equal to the concentration of acceptor (Group III) or donor (Group V) dopants. An important consequence of this is that in the example of arsenide doped silicon, the number of holes (also 10^{10} cm^{-3}) does not remain constant but decreases due to the relationship

$$\overline{n}\overline{p} = n_i^2 \tag{1}$$

where $\overline{n}$ and $\overline{p}$ are the thermal equilibrium concentration of free electrons and holes. In this example the electron concentration is 10^{16} cm^{-3}, and the hole concentration is 10^4 cm^{-3}. As almost all free charge carriers are now electrons, which are negatively charged, this is called an *n*-type or *n*-doped semiconductor, while a semiconductor with a Group III (acceptor) type dopant is a *p*-type or *p*-doped (for the positive charge of a hole) semiconductor. The case of an *n*-type doped semiconductor is depicted in Figure 5.

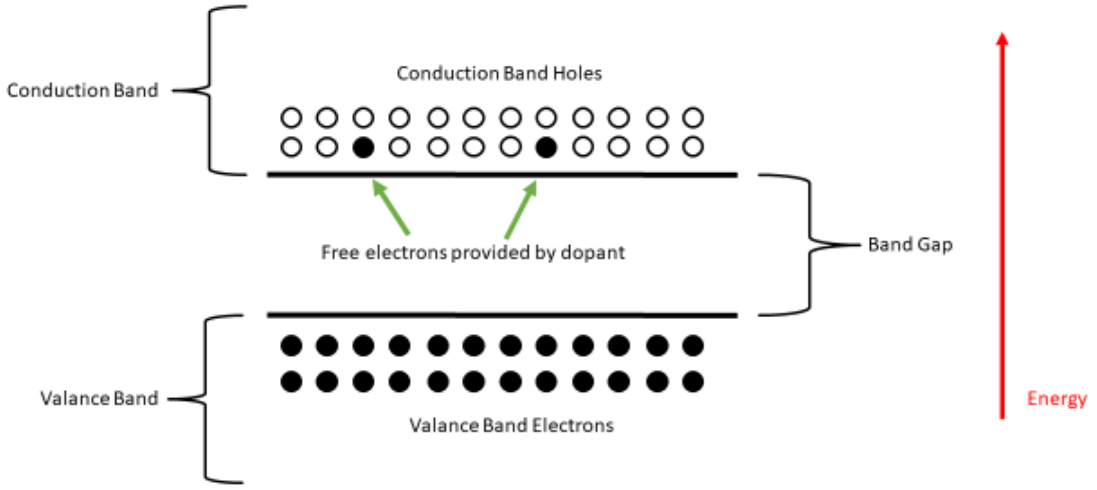


Figure 5. *n*-type Doped Semiconductor. Adapted from [5].

B. PN JUNCTIONS

A doped semiconductor is already a semiconductor device. If contacts are attached to a doped semiconductor and a voltage applied, current flows in the semiconductor as the charge carriers move in response to the applied electric field. Because there are much fewer charge carriers than in a conductor, current does not flow as efficiently as in a wire, meaning it is acting as a resistor. Something more interesting happens if we connect a *p*-doped and *n*-doped semiconductor together. Across the barrier of the junction between the doped regions, also called a diode, a concentration gradient of charge carriers exists. Much like dye dispersing though water, this gradient causes electrons to diffuse from the *n*-doped region to the *p*-doped region and holes to diffuse in the other direction. This movement of charge carriers is called diffusion current. When this occurrs, the previous assumption of space charge neutrality, where the numbers of protons and electrons in any given area are equal, is no longer true. The dopant atoms cannot move from their lattice sites and, thus, contribute to a local electric field. The concept of space charge neutrality is depicted by

$$\bar{n} + N_A = \bar{p} + N_D \tag{2}$$

where N_A is the concentration of acceptor dopants and N_D is the concentration of donor dopants as given in [5]. The relationship shown in Equation (2) remains true for the

semiconductor as a whole but is no longer true locally near the junction. As electrons settle into the holes present on the *p*-doped side of the junction and leave the *n*-doped side, these areas become devoid of charge carriers. This region around the junction, which contains very few carriers, is called the depletion region. While this diffusion is occurring, the field developed across the junction creates a current, called drift current, in the opposite direction of diffusion current. As more and more stationary charged dopants are relieved of their free neutralizing charge carriers, this field becomes stronger and stronger until the magnitude of drift and diffusion currents are equal, at which point no further exchange of charge carriers occurs across the junction [5]. A visual representation of this process is shown in Figure 6.

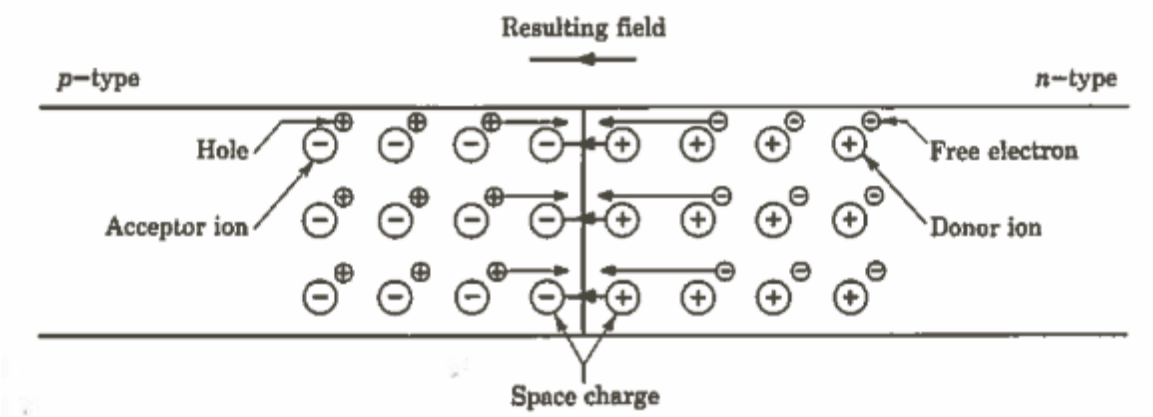


Figure 6. Field Development across a PN Junction. Source: [5].

The region of ionized dopants at the junction is known as the depletion region. The voltage drop across the depletion region is called the built-in voltage. As an external voltage is applied across the junction, minority carrier (electrons in the *p*-type region and holes in the *n*-type region) concentration vary exponentially with the magnitude of the voltage [5]. This relationship is given by

$$n_p = \bar{n}_p \exp\left(\frac{qV_a}{kT}\right) \tag{3}$$

and

$$p_n = \bar{p}_n \exp\left(\frac{qV_a}{kT}\right). \tag{4}$$

Equations (3) and (4), respectively, describe the minority carrier concentrations, n_p and p_n, as functions of applied voltage V_a, temperature T, and charge q. The constant k is Boltzmann's constant. Because current across a device is proportional to the minority carrier concentration, current in the device rises and falls exponentially with applied voltage. This relationship is given as

$$I = I_0\left(\exp\left(\frac{qV_a}{kT}\right) - 1\right) \tag{5}$$

where I_0 is the reverse-bias steady-state current value. This relationship is shown visually in Figure 7.

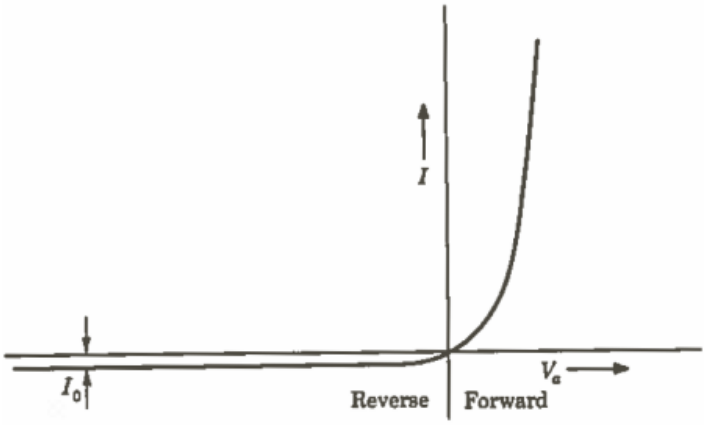


Figure 7. Diode Current as a Function of Voltage. Source: [5].

C. SOLAR CELLS

A solar cell, or photovoltaic device, is a semiconductor device that directly converts photons into electrical current and voltage. The primary source of these photons is the sun itself.

1. Solar Spectrum

The intensity of photons output by the sun is relatively constant as a function of time and varies as a function of wavelength. The shape of this spectrum remains the same at all points in space, though the magnitude of the curve as a whole is shifted up or down with distance from the sun. As photons move away from the sun, they cover a greater and greater surface area, which grows proportionally as the square of the distance from the sun. Given the distance from the sun, every location on Earth or in an earth-bound orbit is essentially the same, so the change in power based on distance is trivial at any point on earth or earth orbit. The relative intensity of photons at each wavelength, where intensity is taken as the number of photons per unit area per unit time, changes as the light moves through a medium such as Earth's atmosphere. Because we are primarily interested in only two regions, space, where there is no atmosphere, and the surface of the earth, with a full atmosphere to interfere with the light spectrum, we give these two spectra special names: AM0 (space) and AM1.5 (the surface of the earth). The relative spectra at each of these levels of atmosphere is shown in Figure 8.

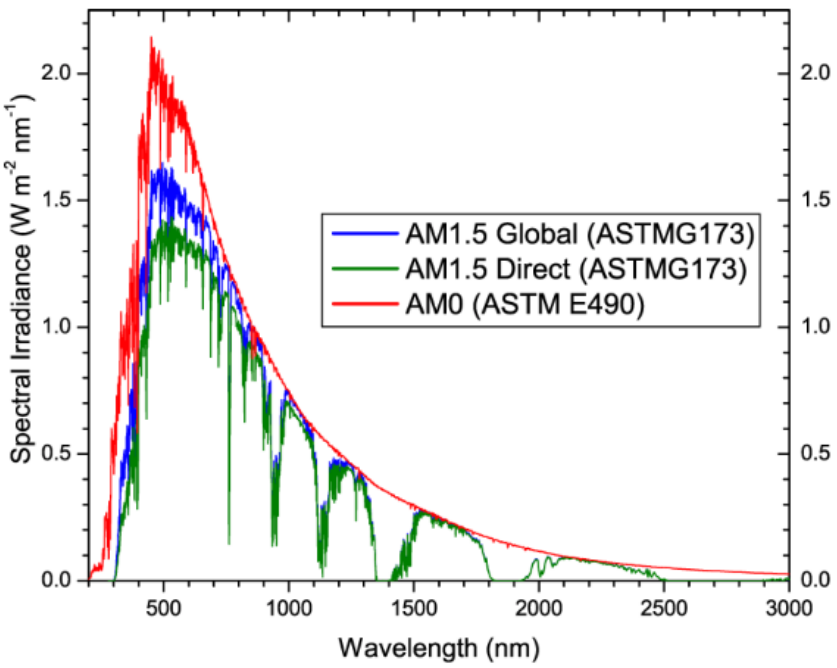


Figure 8. Solar Spectrum at AM0 and AM1.5. Source: [7].

The wavelength of a photon is directly related to the energy of the photon through the well-known relationship

$$\hbar c = \lambda E \,, \tag{6}$$

where $\hbar$ is the reduced Plank's constant, c is the speed of light, E is photon energy, and λ is the wavelength of the photon.

The spectrum is important in designing a solar cell. The closer a semiconductor's bandgap matches a portion of solar spectrum, the more light from those wavelengths is absorbed. It is obvious that few to no photons with energies below the bandgap for a material are absorbed since this would give electrons a forbidden amount of energy. It is less obvious, but equally true, that energies high above the bandgap are also seldom absorbed in a semiconductor.

2. Solar Cell Operation

A solar cell is nothing more than a PN junction that is intended to be exposed to light. The operation of a solar cell is a process in which photons enter into the semiconductor, promoting valence charge carriers into the conduction band. These excess carriers (excess compared to the amount a doped material has without light exposure) are then swept by the built-in field of the junction to contacts at the edge of the semiconductor. This current, called photocurrent, can then power an external circuit. This makes the cell a power source.

With no current flow, the voltage of the cell is the built-in voltage. Conservation of energy dictates that as current increases, voltage must decrease using the same exponential relationship discussed in relation to Equation (5). This relationship is shown visually in Figure 9, where V_{OC} is open circuit voltage and I_{SC} is short circuit current, the maximum possible voltage and current, respectively, a cell can produce. Power is calculated as current multiplied by voltage, meaning the maximum power point occurs at the knee of the current-voltage (IV) curve. A theoretical, but unachievable, maximum power is the intersection of V_{OC} and I_{SC}, which is visually represented by a perfect rectangle. How much of that theoretical rectangle is filled by the actual IV curve is called the fill factor (given as a

percent filled). The efficiency of a cell is given as a ratio of the maximum power point (usually given as watts or milliwatts per square centimeter) to total solar power available in an area (which is constant under ideal weather conditions at about 100 mW per square centimeter on earth and 135 mW per square centimeter in space), given as a percentage. Open circuit voltage is set by the built-in voltage of the material and, thus, by the bandgap. Efficiency is related to short circuit current. Photocurrent is a function of recombination, a situation where excess electrons and holes combine with each other and are no longer available for power generation, which can occur as band-to-band recombination or defect mediated recombination. A visual representation of solar cell current as a function of voltage is shown in Figure 9.

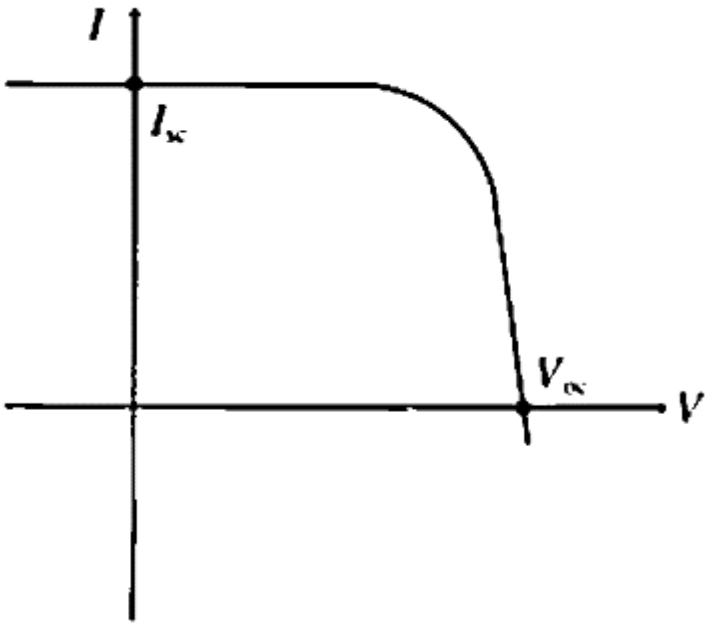


Figure 9. Current-Voltage Relationship of a Solar Cell in Light.
Adapted from: [8].

From these relationships, it is clear that finding the most efficient cell thickness is a balancing act, as a thinner cell requires less time for charge carriers to reach the contacts, reducing recombinations, but also gives photons less distance to create free carriers to begin with.

D. DUAL-JUNCTION SOLAR CELLS

Given the previous discussion of solar cell operation, any single solar cell cannot capture the large spectrum of solar energy to convert to useful electrical energy. The bandgap of a material limits its usefulness to a limited range of the full solar spectrum. To capture more of the light emitted by the sun, additional solar cells are needed, each with a peak affinity for light at a different wavelength. Using an array of these, it is possible to capture almost every part of the spectrum. Of course, to actually gain in efficiency, this needs to be done without increasing the area of the cell; therefore, the cells must be stacked vertically. In such a configuration, one part of the spectrum is absorbed by the top cell and the rest of the photons pass through to the lower cells, and so on. The quantum efficiency of the solar cell modeled is shown in Figure 10. This quantum efficiency is a representation of how well each junction in a multi-junction solar cell absorbs a particular part of the light spectrum. Ideally, these would be two distinct and non-overlapping curves, as is the case in Figure 10.

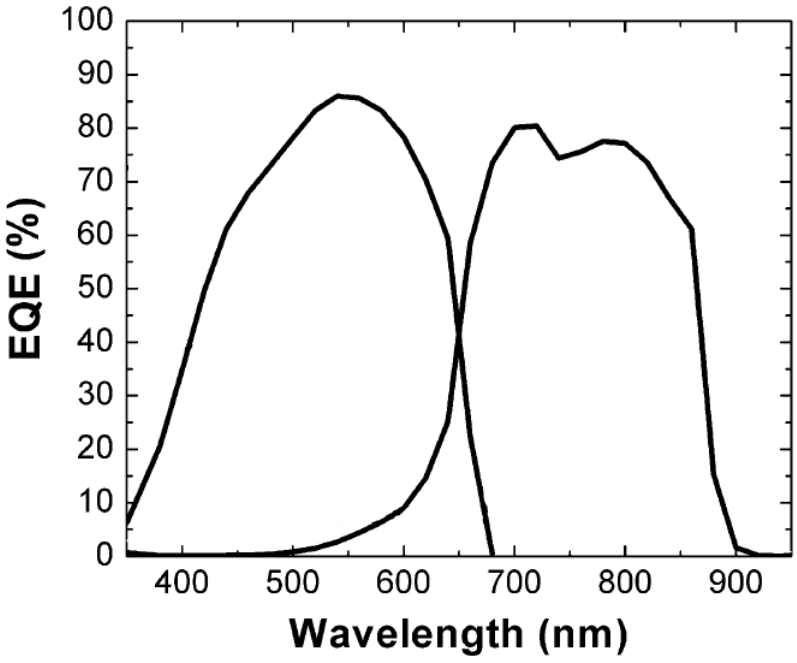


Figure 10. Quantum Efficiency of a Dual-junction Cell. Adapted from [4].

In a tandem (dual-junction) or multi-junction cell, carrier generation works exactly as in a single-junction solar cell. The location of carriers generated, however, takes on a new importance. Cells in a dual-junction configuration are connected in series and current is limited to the smallest current producing element in the series. To minimize wasted current, both cells in a dual-junction cell should produce about the same amount of current. The highest bandgap element in a dual-junction or multi-junction cell should be placed on the top, absorbing only the highest energy photons, and allowing all lower energy photons to pass through to the lower cells to contribute to the photogeneration rate in the lower cells. In Figure 10, the curve on the left represents the top junction, with its peak absorption at a lower wavelength and higher energy.

The component cells being connected in series may limit current through the cell but will drastically raise the voltage at which the cell operates. The open circuit voltage for a multi-junction cell can be estimated by adding the open circuit voltages of the component cells, which themselves are the built-in voltages created across the junctions and mostly a property of the materials used as the semiconductors.

1. Manufacturing

Solar cells can be stacked in one of two ways. First, they can be mechanically stacked, where a complete cell is put on top of another complete cell. Because metallization exists between the cells, they are actually connected in parallel, eliminating the issue of current limitations discussed in the principles of operation for tandem cells. This method is prohibitively expensive and difficult for all but a few test cells to be built.

The second method, and the method used in the cell explored in this thesis, is to grow the cells atop of one another, epitaxially. While this process is much more difficult and expensive than producing a single junction solar cell, its cost lies within the realm of affordability for space applications. In theory, this type of dual-junction cell looks like the cell depicted in Figure 11, but there is a major problem. As shown in Figure 11, simply stacking one cell on top of another inadvertently creates a reverse-biased junction between the two cells. This junction creates an electric field in the opposite direction of the fields created by the component cells and renders the total solar cell nonfunctional. Fortunately,

a special type of PN junction, called a tunnel junction, can be inserted between the cells to alleviate this problem.

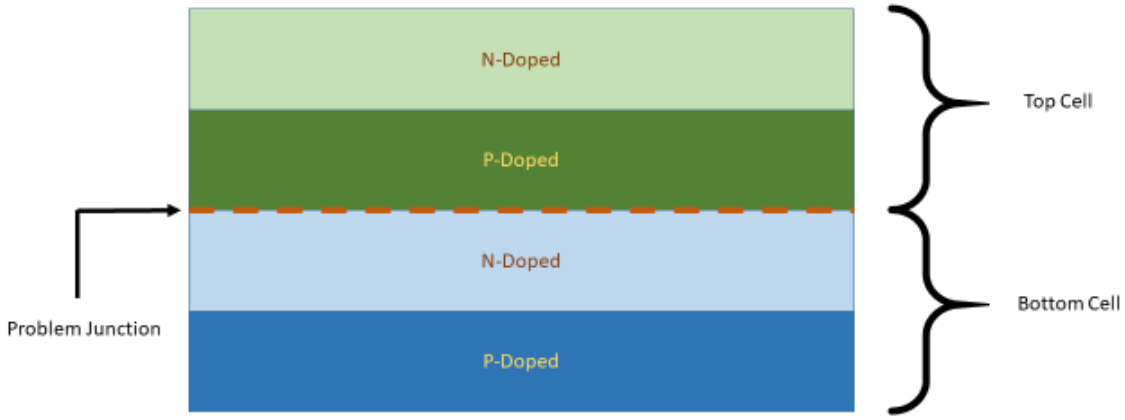


Figure 11. Dual-junction Cell with No Tunnel Junction

2. Tunnel Junctions

A tunnel junction occurs where a PN junction is so heavily doped that it ceases to operate as a normal diode. This situation occurs when both sides of the junction become degenerate, meaning the fermi levels are inside the valence and conductions bands themselves [9]. A full understanding of fermi levels and band physics across a junction is necessary to fully understand the principles of operation of a tunnel junction, but only the characteristics of these junctions are discussed in this thesis. The current-voltage characteristics of a tunnel junction are displayed in Figure 12. When this diode is reverse biased, it does not completely block current but instead acts as a resistor; therefore, when properly biased, current can pass through in the reverse direction of a normal diode [9]. This allows such a diode to be placed between cells in a multi-junction solar cell in the reverse direction of the component PN junctions without destroying the current of the overall cell. This eliminates the accidental creation of a normal PN junction in the reverse direction, as in Figure 13.

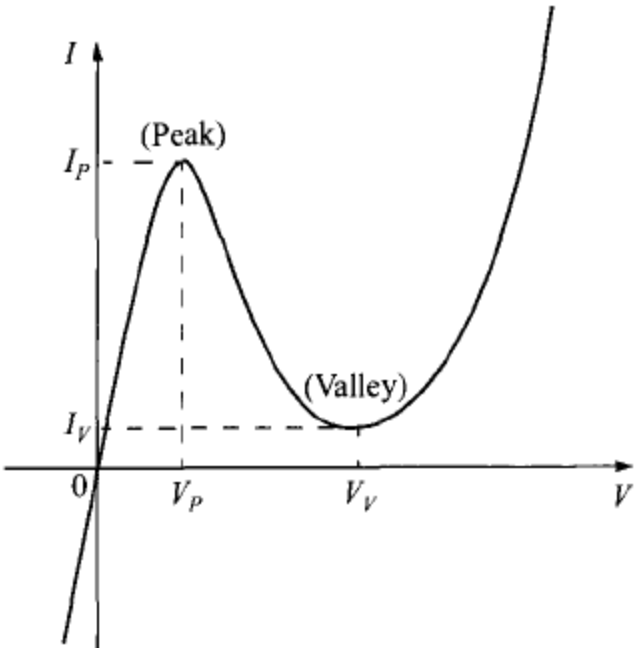


Figure 12. Tunnel Junction Current Voltage Relationship. Source: [9].

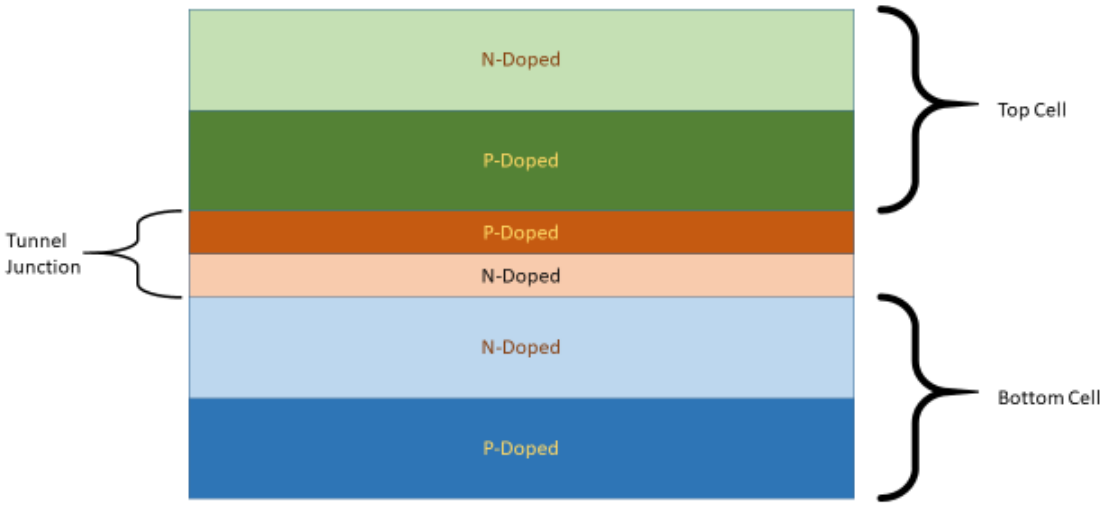


Figure 13. Dual-Junction Solar Cell with Tunnel Junction

E. RADIATION

The space environment is especially grueling for electronic devices. A magnetic field encompasses the earth that traps charged particle radiation (electrons, protons, and heavy ions) in orbit. This area of high radiation is called the Van Allen belt. Proton and

heavy ion radiation is especially damaging, but this can be used to protect electronics. As these heavier particles are so damaging, they quickly lose kinetic energy and stop posing a threat. This process is so rapid and takes place over such a short distance that the top layers of a solar cell, such as antireflective coating, window layers, and contacts, end up shielding the vulnerable layers underneath. Electrons, being equally charged but much less massive than protons or heavy ions, lose energy much more slowly. These particles can travel straight through a solar cell, depositing energy while they do so, but leaving with some energy remaining. While the deposited energy can cause damage in all areas of the cell, in this thesis we focus on damage in the actual solar cell junctions, not in other supporting elements.

In order to have as useful a tool as possible for modeling the space environment for solar cells, radiation flux (incident events per square centimeter per second) are calculated for two orbital paths and as a function of length of time exposed. Stassinopoulos and Raymond [10] present radiation information for two distinct orbits, low-earth orbit (LEO) and geosynchronous orbit (GEO). Low-earth orbit is much lower than GEO, and only extends to about 2,000 km above the earth, while GEO is ~35,800 km from the earth. The flux at LEO is much greater than the flux at GEO due to the density of magnetic field lines closer to the earth [10].

1. Damage Mechanisms

The rate that a charged particle deposits energy in a material as a function of depth into the material is known as stopping power [11]. There are three distinct methods by which energy is deposited [11]. The first mechanism is radiative, when an electron slows down and gives off energy in the form of photons. The second method is ionization, when a fast moving electron excites and imparts enough energy on a bound electron to free it through electric field interactions [11]. The third form is non ionizing energy loss, or NIEL [11]. This occurs when an electron deposits energy directly into the nucleus of an atom through non-elastic collision. If enough energy is given to the nucleus to break the bond between it and its neighbor atoms, the atom is moved out of its crystal lattice site [11].

NIEL deposition is detrimental and creates permanent damage in solar cells. A simple representation of this type of damage is displayed in Figure 14. When an atom is knocked from the lattice, it leaves behind a site vacancy. The atom must still reside somewhere, and since all other sites are occupied, it ends up in a space that is not part of the lattice. This is called an interstitial defect. The vacancy and interstitial defects are known as a Frenkel Pair [11]. Both the vacancy and interstitial defect are spots, which allow a hole and electron to recombine, lowering the photocurrent and efficiency. This mechanism causes a solar cell to become less efficient the longer it is exposed to radiation.

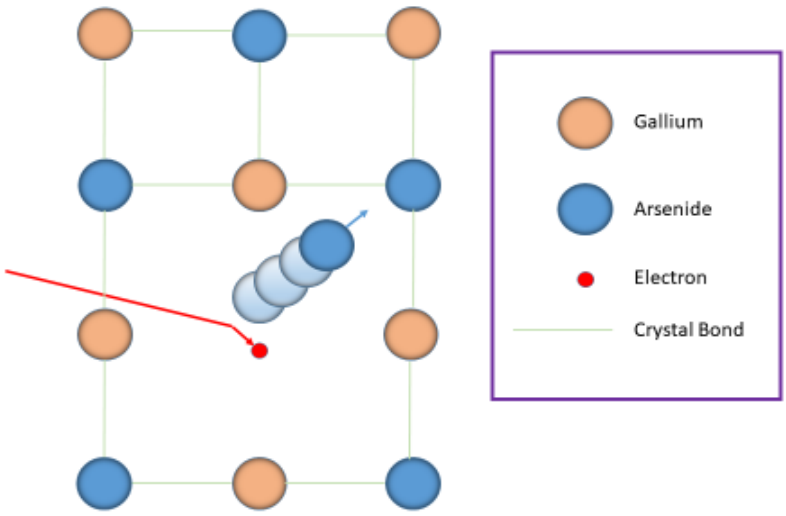


Figure 14. Lattice Damage Caused by Electron Impingement

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY

A. MODELED CELL

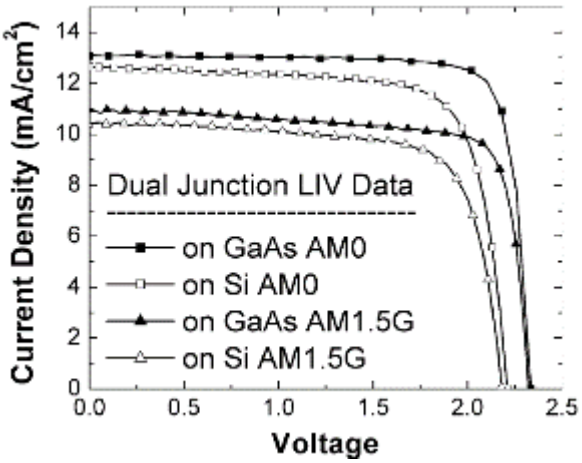
The cell modeled in this work is a dual-junction InGaP-GaAs cell fabricated at Ohio State University [4]. This cell was chosen for several reasons. While the cell is a dual-junction cell, such a cell has all the properties of more advanced multi-junction cells; thus, any techniques for optimizing a dual-junction cell directly translate to a multi-junction solar cell. In addition, an exact profile of the layer thickness, composition, and doping profile in the cell is given in [4] as well as data about the cell performance. Due to the difficulty, expense, and time needed to design and fabricate a solar cell, especially one for space applications, companies who manufacture them carefully guard the exact properties of the design. This data allows for an accurate physics-based model of the cell to be constructed with minimal assumptions.

The top junction of the cell is an InGaP cell (49% indium phosphide (InP), 51% gallium phosphide (GaP)) with an indium aluminum gallium phosphide (InAlGaP) (47% InP, 37.1% aluminum phosphide (AlP), 15.9% GaP) window and back surface field layer. The bottom junction is a GaAs cell with an InGaP (49% InP, 51% GaP) window and an aluminum gallium arsenide (AlGaAs) (70% aluminum arsenide (AlAs), 30% GaAs) back surface field layer. The two junctions are separated by a GaAs tunnel junction. The contact layer on top of the cell is also made of GaAs, as is the bottom buffer layer. The entire solar cell is grown on germanium (Ge) buffer and a silicon germanium (SiGe) substrate. A representation of the cell, not to scale, with all layer thicknesses, composition, doping type, and doping amounts is shown in Figure 15.

p++ GaAs contact layer (1000 Å)	~1x10^{19}
p+ $In_{0.47}(Al_{0.7}Ga_{0.3})_{0.53}P$ window (300 Å)	~2 x10^{18}
p+ $In_{0.49}Ga_{0.51}P$ emitter (500 Å)	~2 x10^{18}
n $In_{0.49}Ga_{0.51}P$ base (5500 Å)	~7 x10^{16}
n+ $In_{0.47}(Al_{0.7}Ga_{0.3})_{0.53}P$ back surface field (300 Å)	~2 x10^{18}
n++ GaAs TJ (250 Å)	~2 x10^{19}
p++ GaAs TJ (250 Å)	~2 x10^{19}
p+ $In_{0.49}Ga_{0.51}P$ window (400 Å)	~3 x10^{18}
p+ GaAs emitter (5000 Å)	~2 x10^{18}
n GaAs base (20,500 Å)	~2 x10^{17}
n+ $Al_{0.7}Ga_{0.3}As$ back surface field (1000 Å)	~2 x10^{18}
n+ GaAs buffer (2000 Å)	~2 x10^{18}
Ge (300Å)	uid
n+ SiGe substrate	~1 x10^{18}

Figure 15. Modeled Cell Profile. Source: [4].

The measured current-voltage relationship under AM0 illumination and the short circuit current, open circuit voltage, fill factor, and efficiency of the fabricated cell are shown in Figure 16. Notice that while several curves are given, we are only focused on the closed box curve for GaAs at AM0 and only interested in the AM0 column for the chart of values for the purposes of this thesis. It is immediately obvious that this cell is not particularly good, with an 18.6% efficiency. This can be attributed to the goal of the team when manufacturing this cell, testing a new substrate. The parameters have not been optimized, the ten percent metal coverage is far higher than the two percent standard coverage, and the antireflective coating present on all solar cells is especially bad on this cell. It reflects about ten percent of the light across all wavelengths [4]. The industry standard is to reflect about two percent. While these factors contribute to an underwhelming efficiency, they are relatively unimportant for optimization. Obviously, the less shadowing and reflected light, the higher the efficiency. Moreover, making those changes raises efficiency linearly and does not change the values of layer thicknesses or doping levels that give an optimum output.



	GaAs	
	AM0	AM1.5G
Jsc (mA/cm²)	13.08	10.9
Voc (V)	2.34	2.32
FF (%)	82.5	79.0
η(%)	18.6	20.0

Figure 16. Experimental Data for Modeled Cell. Adapted from [4].

With the changes made to allow simulation of the cell in Silvaco, there are nine layers available for manipulation. Each layer has a thickness and doping level, totaling 18 parameters to optimize.

B. SILVACO ATLAS

Silvaco ATLAS, or Silvaco, is a program to calculate the solution to the differential equations that govern semiconductor behavior. Because this set of differential equations is nonlinear, there is no method known to analytically solve them as a function of location (in our solar cell, only one dimension, depth into the cell, exists, though Silvaco can handle two and three dimensional semiconductors as well). It is relatively easy, however, to check if a given candidate solution is correct. If a solution is known in one location, then the solution to a location sufficiently nearby should be similar. Silvaco applies this logic throughout the cell to attempt to find solutions at all specified locations. Silvaco iteratively checks candidate solutions using Newton's method to refine the candidate each iteration until the difference between sequential candidates is less than a certain threshold. At this point Silvaco reports this candidate as the correct solution.

Silvaco uses a scripting language to input the design of the semiconductor device to be tested. The rigid nature of the scripting language makes it ideal for programmatically generating scripts with slight variations to be tested for optimizing a design.

1. Mesh

Silvaco attempts to find a solution for the semiconductor equations at specified points and uses the last specified point solution as the starting point for finding a solution for the current location. This means that the distance between any two sequential points must be small enough that the difference in solution values is small. This must be balanced with the fact that each additional solution location requires Silvaco to conduct additional calculations, increasing simulation time. This leads to the conclusion that the test points must be very densely packed where electric field, materials, and doping levels are changing, such as at the junctions. Where these parameters are relatively constant, such as in the bulk of the layers, the primary characteristic affecting charge carrier concentration and current is bulk resistance, which reduces current linearly. In these bulk regions in the layers, test points can be spaced relatively widely in order to reduce simulation time.

Silvaco requires these test points to be specified as a mesh, or grid, of points to check. These points are specified by *x* and *y* coordinates. For the *x* values, since this is really a single-dimension structure and there are no changes from left to right across the cell, only three columns of test points are used, one on the left edge, one in the middle, and one on the right. The *y* values of these points must take into account the location of the junctions. As the layer thicknesses change, the location of the points also changes, requiring dynamically created meshes for each simulation. The profile used for these simulations was very dense around the edge of the layers (where the junctions occur), with a concentration that would give 100 points if it encompassed the entire layer. This gradually was reduced to a medium concentration (a value that would give 50 points if it encompassed the layer) at 20% of the distance from the junction. Finally, this gradually fades to a very light concentration (ten points per layer concentration) at 50% of the distance from the edges of the layer, as shown in Figure 17.

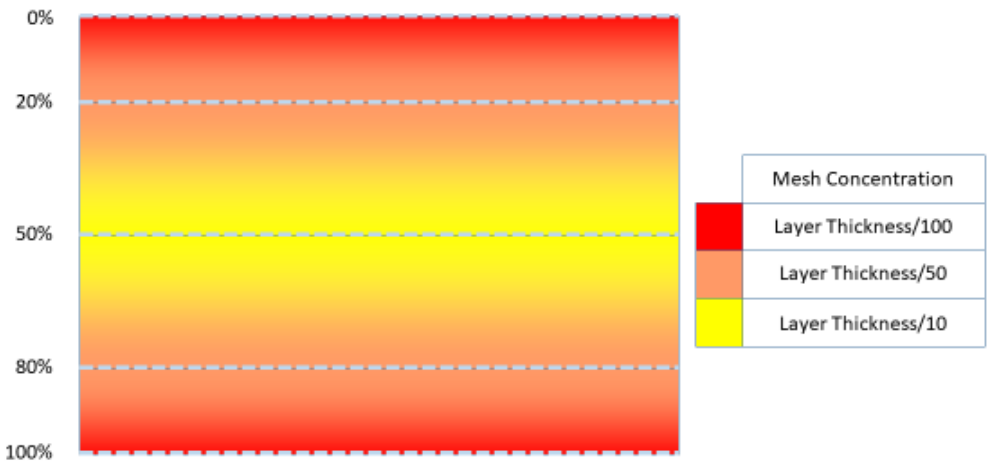


Figure 17. Mesh Density Profile

2. Electrodes

The electrode statements usually only contain the anode at the top of the cell and the cathode at the bottom. For this model, two additional contacts are used, one that covers the top layer of the tunnel junction and encroaches slightly into the bottom layer, and one that connects the bottom layer of the tunnel junction to the InGaP window of the bottom cell, as shown in Figure 18.

3. Contact

Contact statements allow a contact resistance to be specified as a resistance between the contact and the cathode. For the anode, this becomes an in-series resistance and is used to fine tune the simulation to the experimental data. For the tunnel-junction contacts this is a parallel path to the cathode and is set at some arbitrarily large number (10^{18}) to prevent this path from affecting the series path through the tunnel junction for current.

4. Traps

Trap statements allow trap-site concentrations and parameters to be specified. These are treated by Silvaco as dopants, but poor dopants that allow recombination of free electrons and holes. In Section E (Radiation Model) of this chapter, we give more information about the creation of these statements.

5. Beam

The beam statements are used to specify details about the light source. The beam intensity was set to 81% to account for the antireflective coating and metal shadowing in the model cell.

6. Changes from Model Cell

The cell modeled has several changes made to it compared to Figure 15. The top layer, the GaAs contact layer, was removed. This layer is only present beneath the metal contacts and not on the parts of the cell exposed to light. To account for the effect of the shadowing created by the metal contacts, which [4] reported covered 10% of the cell, the intensity of the light beam in the simulation was reduced by 10%. To account for the resistance added by this contact layer and the connection between the layer and the metal contact (which Silvaco models as unrealistically perfect), a resistance was added to the top contact (the anode) in the model. The GaAs tunnel junction layers, layers six and seven, are not something that Silvaco can model consistently. To alleviate this problem without sacrificing the accuracy of the model, these layers were replaced by a sandwich of two perfect contacts with an exposed area of GaAs remaining. This sandwich allows perfect voltage and current transfer across the contacts, and the exposed GaAs doping can be adjusted to simulate the resistance the tunnel junction adds. The downside to this solution is that optimization of the tunnel junction is impossible. Once properties are found to match the test data, that section is set in stone. Even with this restriction, this represents a huge improvement in tunnel-junction modeling. Previous solutions have been to turn the entire area into a metal contact, ignoring the resistance of the junction and allowing light to pass through undisturbed. The resistance sandwich approach leaves GaAs as the material so it absorbs light just like the real junction does. Finally, the Ge and SiGe bottom layers are discarded. Lueck's team was researching a less expensive manufacturing technique of growing this cell on these materials instead of the conventional GaAs [4]. That research is irrelevant to the optimization undertaken in this thesis. Cells were fabricated and tested in [4] with a traditional GaAs substrate, and that is the cell and data used to fine tune and verify the modeled cell. The cell modeled in Silvaco with the changes made is shown in Figure 18.

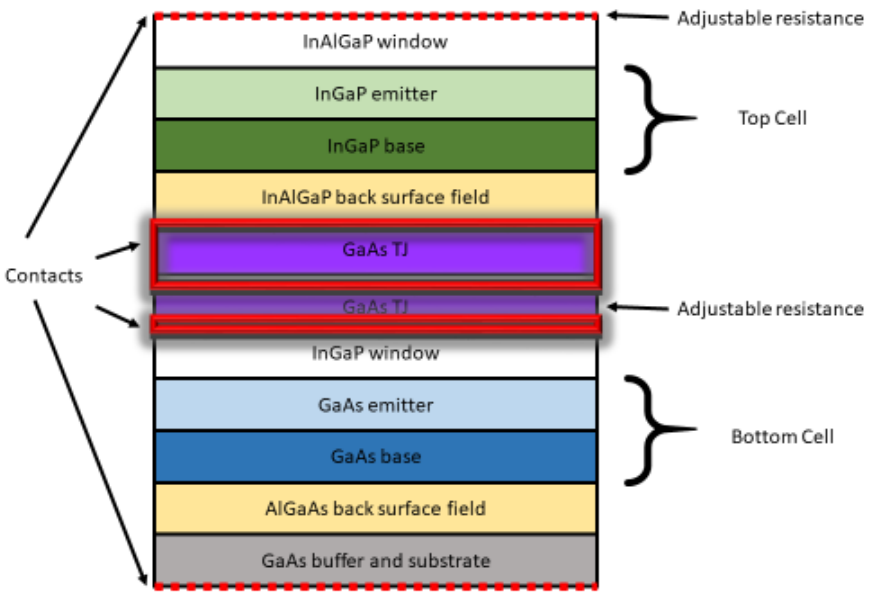


Figure 18. Cell Modeled in Silvaco

C. OPTICAL PARAMETERS

To determine the rate and location of generated electron/hole pairs, it is necessary to have an accurate description of the photon absorption coefficient in materials used in the solar cell. This absorption coefficient is a function of photon energy level (or photon wavelength). For each of the materials used in the solar cell modeled, experimental data was gathered from performing spectroscopic ellipsometry. Spectroscopic ellipsometry provides optical dielectric response at several wavelengths of light. This data was then modeled mathematically using a curve fitting function. While any type of curve fit will work (an infinite series of polynomials, an ordered set of points, etc.), work by Adachi [12], and his team [13] argues for a semi-predictive method using physical models of absorption by expanding about critical points in the electronic structure of a material corresponding to bandgap energies E_0, E_1, and E_2 as well as the split off energies corresponding to these points. A non-dispersive term ε_∞ was also added to account for higher energy levels [12], [13]. Silvaco was found to have satisfactory models for GaAs. Silvaco also comes packaged with an adequate example file for InGaP. For the more complex materials, InAlGaP and AlGaAs, Silvaco does not have satisfactory models that fit experimental data well. Adachi's model was used to generate the index of refraction n

and the extinction coefficient k that Silvaco uses to describe absorption and reflection in each material as a function of photon wavelength. The *n* and *k* values are related to the dielectric function as, respectively,

$$n = \left[\frac{\left(\varepsilon_1^2 + \varepsilon_2^2\right)^{0.5} + \varepsilon_1}{2} \right]^{0.5} \tag{7}$$

and

$$k = \left[\frac{\left(\varepsilon_1^2 + \varepsilon_2^2\right)^{0.5} - \varepsilon_1}{2} \right]^{0.5}, \tag{8}$$

where ε_1 and ε_2 are the real and imaginary parts of the dielectric function [12],[13].

1. Aluminum Gallium Arsenide

For AlGaAs, the dielectric function about E_0 is

$$\varepsilon_2(\omega) = \left[\frac{A}{(\hbar\omega)^2} \right] \left[(\hbar\omega - E_0)^{0.5} H(\chi_0 - 1) + \frac{1}{2}(\hbar\omega - E_0 - \Delta_0)^{0.5} H(\chi_{so} - 1) \right] \tag{9}$$

and

$$\varepsilon_1(\omega) = AE_0^{-1.5} \left\{ f(\chi_0) + \frac{1}{2} \left[\frac{E_0}{E_0 + \Delta_0} \right]^{1.5} f(\chi_{so}) \right\}, \tag{10}$$

where A, E_0 and $E_0 + \Delta_0$ are fitting parameters, $\hbar$ represents the reduced Plank's constant, and ω is the radial frequency of the photon [12]. The parameters $H_{(y)}$, $f_{(\chi)}$, χ_0, and χ_{so} are calculated in [12] as

$$H(y) = \begin{Bmatrix} 1 \text{ for } y \geq 0 \\ 0 \text{ for } y < 0 \end{Bmatrix}, \tag{11}$$

$$f(\chi) = \chi^{-2} \left[2 - (1+\chi)^{0.5} - (1-\chi)^{0.5} H(1-\chi) \right], \tag{12}$$

$$\chi_0 = \frac{\hbar\omega}{E_0}, \tag{13}$$

and

$$\chi_{so} = \frac{\hbar\omega}{E_0 + \Delta_0}. \tag{14}$$

For the E_1 transition,

$$\varepsilon_2(\omega) = \pi B_1 \chi_1^{-2} H(\chi_1 - 1) \tag{15}$$

and

$$\varepsilon_1(\omega) = -B_1 \chi_1^{-2} \ln(1 - \chi_1^2) \tag{16}$$

were used to calculate the dielectric function. In Equation (15) and Equation (16), B_1 is a fitting parameter and χ_1 is either [12]

$$\chi_1 = \frac{\hbar\omega}{E_1} \tag{17}$$

or

$$\chi_1 = \frac{\hbar\omega + j\Gamma}{E_0}. \tag{18}$$

In Equation (18), Γ is a broadening parameter. For the E_2 transitions the dielectric function is [12]

$$\varepsilon_2(\omega) = \frac{C\chi_2\gamma}{\left(1 - \chi_2^2\right)^2 + \chi_2^2\gamma^2} \tag{19}$$

and

$$\varepsilon_1(\omega) = \frac{C\left(1 - \chi_2^2\right)}{\left(1 - \chi_2^2\right)^2 + \chi_2^2\gamma^2}. \tag{20}$$

For Equations (19) and (20), C and γ are fitting parameters and χ_2 is calculated as [12]

$$\chi_2 = \frac{\hbar\omega}{E_2}. \tag{21}$$

There is an indirect gap transition to account for,

$$\varepsilon_2(\omega) = \frac{D}{(\hbar\omega)^2}\left(\hbar\omega - E_g^{ID} + \hbar\omega_q\right)^2 H\left(1 - \chi_g\right) H\left(1 - \chi_c\right) \tag{22}$$

where D is a fitting parameter, E_g^{ID} is the indirect band gap, and $\hbar\omega_q$ is phonon energy (taken to be 0; in general, phonons contribute a great amount of momentum while contributing negligible energy) [12]. Adachi solves for χ_g and χ_c with

$$\chi_c = \frac{\hbar\omega}{E_c} \tag{23}$$

and

$$\chi_g = \frac{\hbar\omega}{E_g^{ID} - \hbar\omega_q}. \tag{24}$$

The dielectric function ε for AlGaAs is the sum of all the ε_2 terms multiplied by the imaginary number *j*, added to the sum of all ε_1 terms and ε_∞ [12]. The real and imaginary parts of ε are then ε_1 and ε_2, respectively [12].

The parameters used to fit these equations to experimental data are presented in Table 1. Adachi gives many values for each of these parameters to account for different molar concentrations for aluminum and gallium. The information presented in Table 1 and used in the solar cell model are for 0.7 Aluminum and 0.3 Gallium [12].

Plots of the real and imaginary parts of the dielectric constant from experimental data, the Adachi model, and what already exists in Silvaco are displayed in Figure 19 through Figure 24.

Table 1. Fitting Parameters for AlGaAs. Adapted from [12].

Parameter	Value	Units
E_0	2.42	eV
$E_0 + \Delta_0$	2.73	eV
E_1	3.43	eV
E_2	4.7	eV
E_g^{ID}	2.03	eV
A	23.30	$eV^{1.5}$
B_1	5.41	no units
Γ	0.12	eV
C	1.76	no units
γ	0.103	no units
D	8.1	no units
ε_∞	-0.3	no units

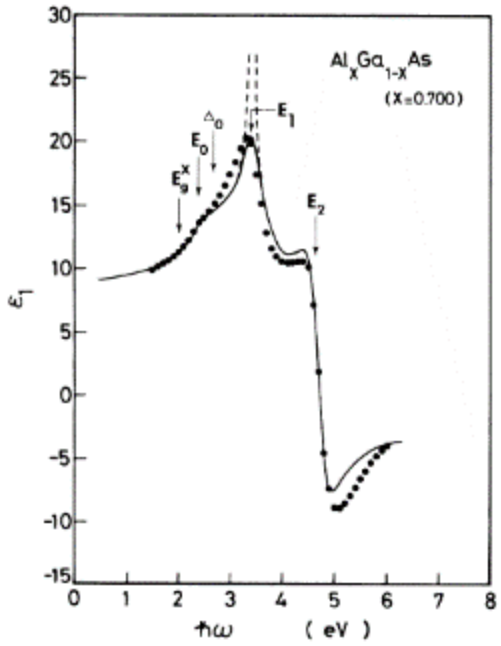


Figure 19. Experimental Data of Real Part of Dielectric Constant for AlGaAs. Source: [12].

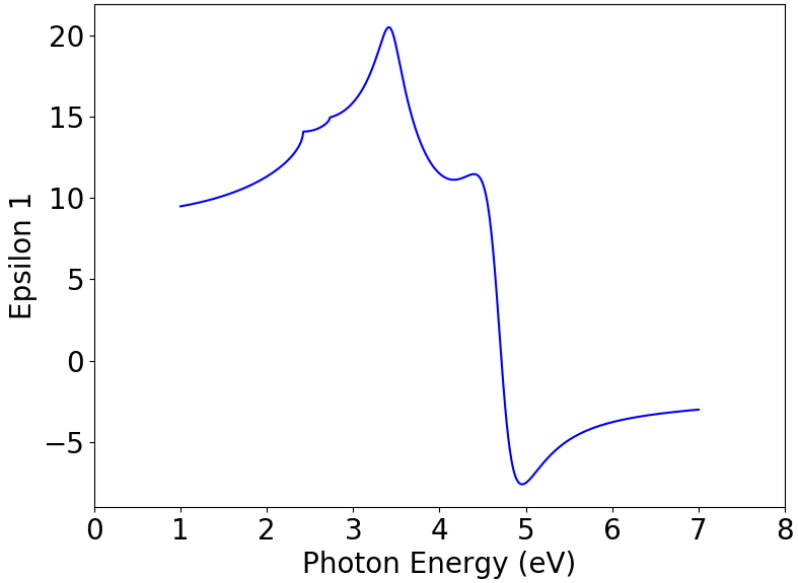


Figure 20. Real Part of Dielectric Constant for AlGaAs Created from Adachi Model. Adapted from [12].

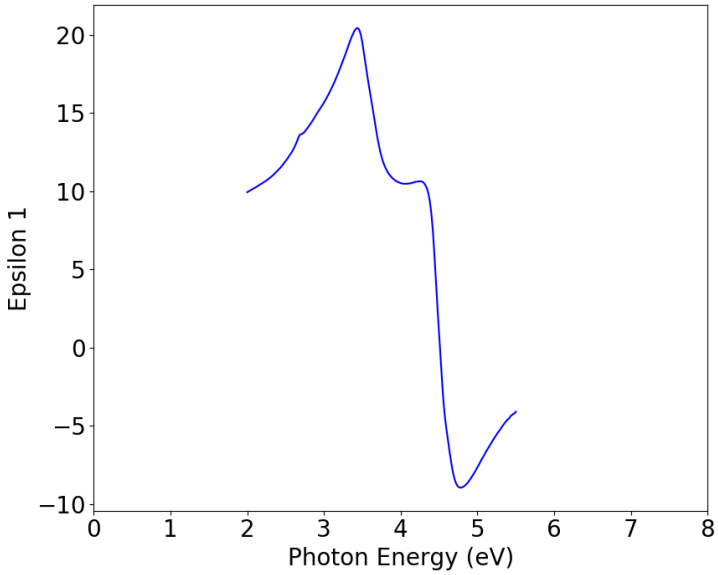


Figure 21. Real Part of Dielectric Constant for AlGaAs Existing in Silvaco

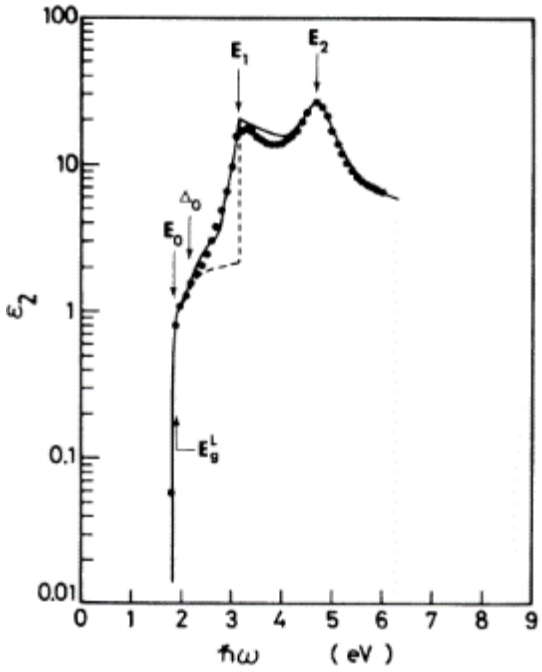


Figure 22. Experimental Data of Imaginary Part of Dielectric Constant for AlGaAs. Source: [12].

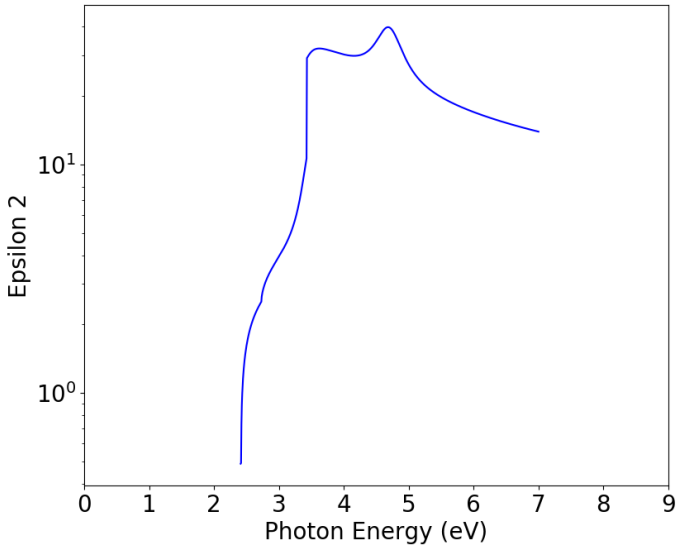


Figure 23. Imaginary Part of Dielectric Constant for AlGaAs Created from Adachi model. Adapted from [12].

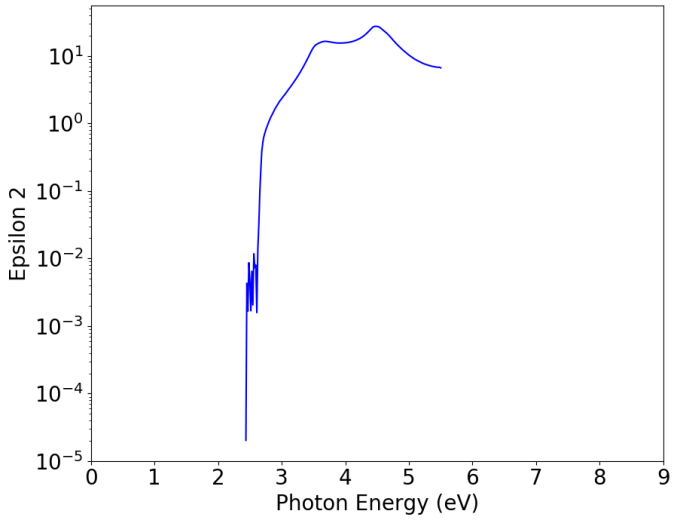


Figure 24. Imaginary Part of Dielectric Constant for AlGaAs Existing in Silvaco

2. Indium Aluminum Gallium Phosphide

Adachi et al. [13] also determined a fitting function for the dielectric constant for InAlGaP. This model has fewer equations because the real and imaginary parts are not separated; rather the functions are complex, combining real and imaginary parts. As with

AlGaAs, critical points corresponding to distinct energy levels are expanded about. Adachi uses [13]

$$\varepsilon_0(E) = AE_0^{-1.5}\left[f(\chi_0) + \frac{1}{2}\left(\frac{E_0}{E_0+\Delta_0}\right)^{1.5} f(\chi_{s.o.})\right], \tag{25}$$

$$f(\chi_0) = \chi_0^{-2}\left[2-(1+\chi_0)^{0.5}-(1-\chi_0)^{0.5}\right], \tag{26}$$

$$f(\chi_{s.o.}) = \chi_{s.o.}^{-2}\left[2-(1+\chi_{s.o.})^{0.5}-(1-\chi_{s.o.})^{0.5}\right], \tag{27}$$

$$\chi_0 = \frac{E+j\Gamma_0}{E_0}, \tag{28}$$

and

$$\chi_{s.o.} = \frac{E+j\Gamma_0}{E_0+\Delta_0} \tag{29}$$

to find the E_0 transitions. In these equations, A, E_0, Γ_0, and $E_0+\Delta_0$ are fitting parameters and E is the energy level of the photon [13]. The E_1 transition is found from

$$\varepsilon_1(E) = -B_1\chi_1^{-2}\ln\left(1-\chi_1^{-2}\right) \tag{30}$$

and

$$\chi_1 = \frac{E+j\Gamma_1}{E_1}, \tag{31}$$

where parameters B_1, E_1, and Γ_1 are for fit [13]. Adachi uses [13]

$$\varepsilon_2(E) = \frac{CE_2^2}{E_2^2-E^2-jE\Gamma_2} \tag{32}$$

and

$$\varepsilon_{2\delta}(E) = \frac{C_\delta(E_2+\delta)^2}{(E_2+\delta)^2-E^2-jE\Gamma_\delta} \tag{33}$$

to solve for the E_2 transitions. For this energy level, C, C_δ, E_2, $E_2+\delta$, Γ_2, and Γ_δ are fitting parameters.

The dielectric constant for InAlGaP is the sum of ε_0, ε_1, ε_2, ε_δ, and nondispersive term ε_∞ [13]. Fitting parameters for these equations are given in Table 2. Once again, Adachi list several values for different molar fractions. Those presented in Table 2 and used in the solar cell model are for 0.375 aluminum, 0.125 gallium, and 0.5 indium[13].

Table 2. Fitting Parameters for InAlGaP. Adapted from [13].

Parameter	Value	Units
E_0	2.38	eV
$E_0 + \Delta_0$	2.45	eV
A	11	$eV^{1.5}$
Γ_0	0.03	eV
E_1	3.6	eV
B_1	4.4	no units
Γ_1	0.26	eV
E_2	4.85	eV
C	1.7	no units
Γ_2	0.82	eV
$E_0 + \delta$	5.02	eV
C_δ	0.5	no units
Γ_δ	0.7	eV
ε_∞	0.4	no units

Plots of the dielectric constant for InAlGaP as a function of photon energy for experimentally determined values, the values given by the Adachi model, and the built in files Silvaco uses if no user defined data is input are contained in **Error! Reference source not found.** through Figure 30. It is obvious that while the shapes are mostly correct, the values natively assumed by Silvaco are not close to being correct.

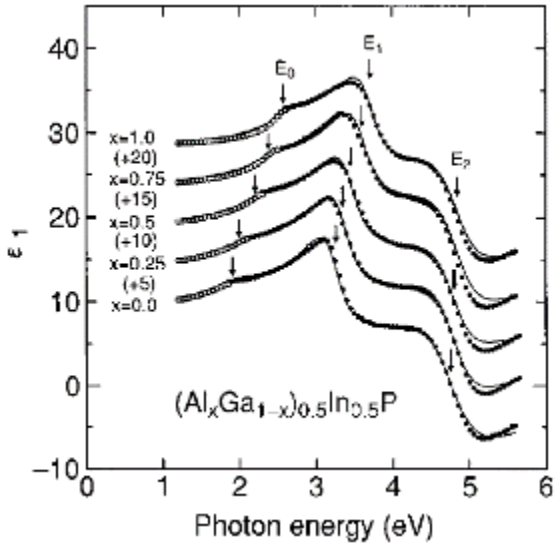


Figure 25. Experimental Data of Real Part of Dielectric Constant for InAlGaP. Source: [13].

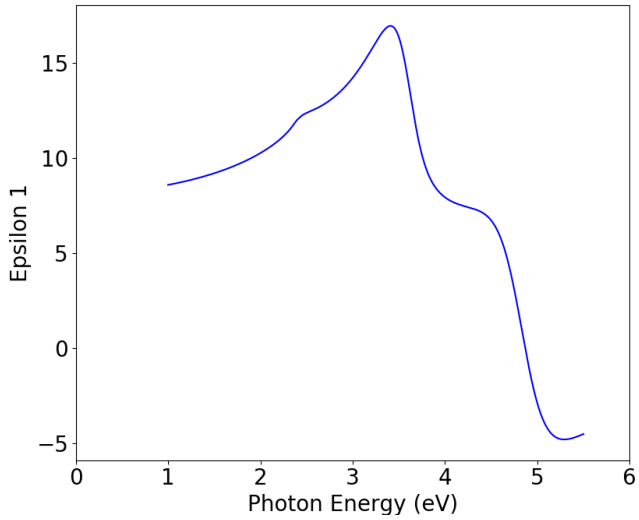


Figure 26. Real Part of Dielectric Constant for InAlGaP Created from Adachi model. Adapted from [13].

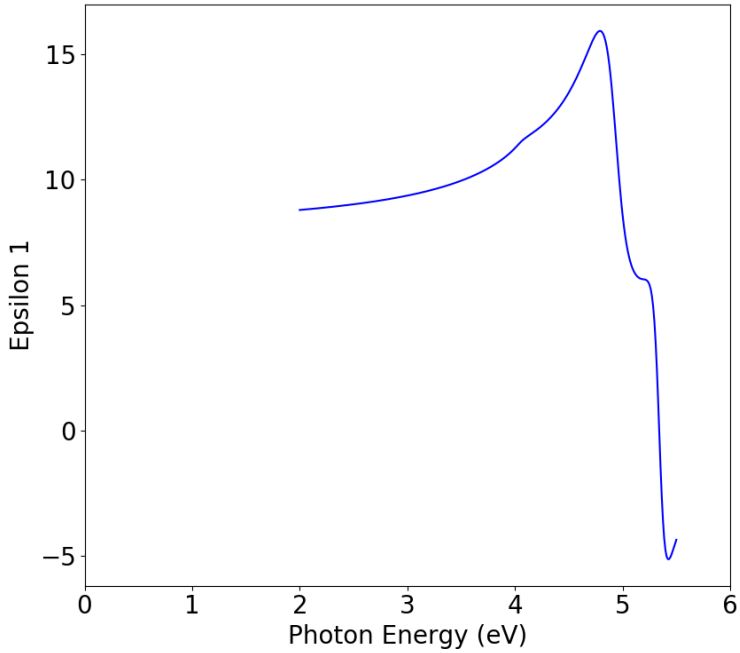


Figure 27. Real Part of Dielectric Constant for InAlGaP Existing in Silvaco

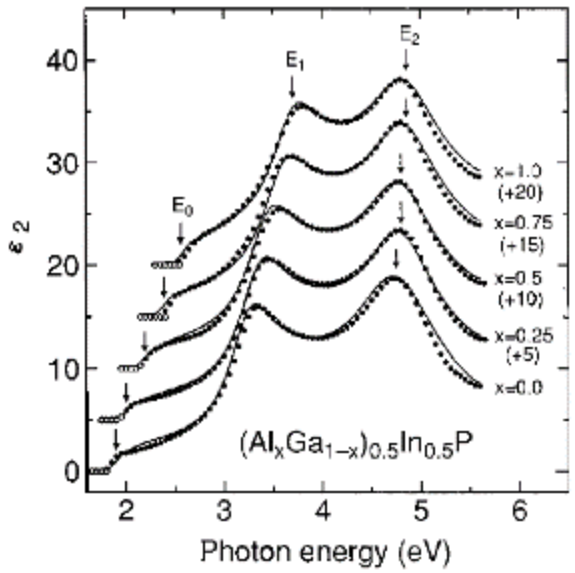


Figure 28. Experimental Data of Imaginary Part of Dielectric Constant for InAlGaP. Source: [13].

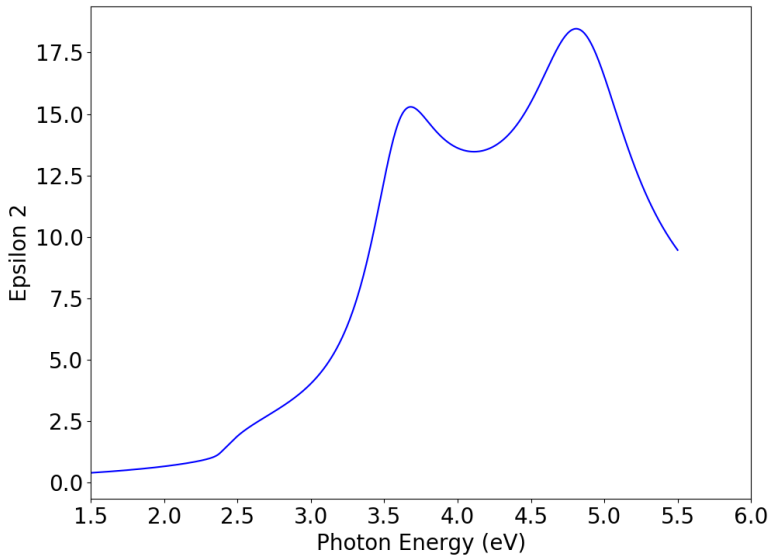


Figure 29. Imaginary Part of Dielectric Constant for InAlGaP created from Adachi Model. Adapted from [13].

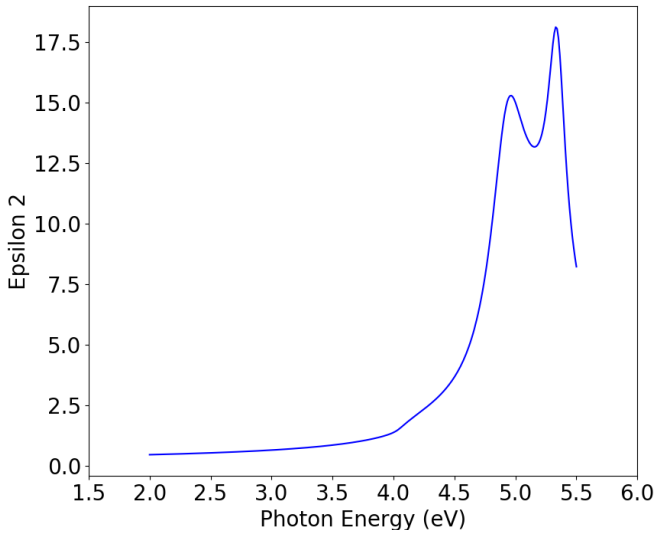


Figure 30. Imaginary Part of Dielectric Constant for AlGaAs Existing in Silvaco

D. MOBILITY

The ability for an electron or hole to move through a material is a function of both the alloy mole fraction as well as the doping level of the material. Silvaco calculates the mobility for electrons and holes (μ_n and μ_p, respectively) as a function of doping for binary materials but does not have built in models for mobility dependence upon doping concentration for tertiary or quaternary materials. While mobility has a fairly small effect on overall efficiency, there is no good reason for ignoring the contribution doping makes to mobility for tertiary or quaternary layers. To this end, the cell in question was modeled with mobilities for InGaP, AlGaAs, and InAlGaP which take into account alloy mole fraction and doping concentration. The equations to calculate these values come mostly from work performed by Sutherland and Hauser [14] in 1977. Since that time, further research has refined their findings, and those advances have been incorporated into their calculation method for this model.

Sutherland and Houser assume a doping dependent function for the binaries constituting a material are already known; i.e., for AlGaAs, the doping dependent mobilities for both AlAs and GaAs are known, where AlGaAs is a blend of these two binaries. To find these baseline binary values, the Caughy-Thomas model was used to

model the dependence of mobility on doping and temperature [15]. Electron and hole mobilities in [15] are calculated from

$$\mu_n = \mu_{1n}\left(\frac{T}{300K}\right)^{\alpha_n} + \frac{\mu_{2n}\left(\frac{T}{300K}\right)^{\beta_n} - \mu_{1n}\left(\frac{T}{300K}\right)^{\alpha_n}}{1+\left(\frac{T}{300K}\right)^{\gamma_n}\left(\frac{N}{N_{CRITn}}\right)^{\delta_n}} \tag{34}$$

and

$$\mu_p = \mu_{1p}\left(\frac{T}{300K}\right)^{\alpha_p} + \frac{\mu_{2p}\left(\frac{T}{300K}\right)^{\beta_p} - \mu_{1p}\left(\frac{T}{300K}\right)^{\alpha_p}}{1+\left(\frac{T}{300K}\right)^{\gamma_p}\left(\frac{N}{N_{CRITp}}\right)^{\delta_p}}, \tag{35}$$

where T is temperature in Kelvin, N is doping concentration, and μ_1, μ_2, N_{CRIT} as well as exponents α, β, γ, and δ are constants of the material [15]. The values of these parameters for the materials used in the solar cell being modeled are listed in Table 3 through Table 6.

Table 3. Gallium Arsenide Mobility Modeling Constants. Adapted from [15].

Parameter	Value	Units
μ_{1n}	500	$\frac{\text{cm}^2}{(V)(s)}$
μ_{1p}	20	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2n}	9400	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2p}	491.5	$\frac{\text{cm}^2}{(V)(s)}$
α_n	0	no unit
α_p	0	no unit
β_n	−2.1	no unit
β_p	−2.2	no unit
γ_n	−1.182	no unit
γ_p	−1.14	no unit
δ_n	0.394	no unit
δ_p	0.38	no unit
N_{CRITn}	6.0×10^{16}	cm^{-3}
N_{CRITp}	1.48×10^{17}	cm^{-3}

Table 4. Aluminum Arsenide Mobility Modeling Constants. Adapted from [15].

Parameter	Value	Units
μ_{1n}	10	$\frac{\text{cm}^2}{(V)(s)}$
μ_{1p}	10	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2n}	400	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2p}	200	$\frac{\text{cm}^2}{(V)(s)}$
α_n	0	no unit
α_p	0	no unit
β_n	−2.1	no unit
β_p	−2.24	no unit
γ_n	−3	no unit
γ_p	−1.464	no unit
δ_n	1	no unit
δ_p	0.488	no unit
N_{CRITn}	5.46×10^{17}	cm^{-3}
N_{CRITp}	3.48×10^{17}	cm^{-3}

Table 5. Gallium Phosphide Mobility Modeling Constants. Adapted from [15].

Parameter	Value	Units
μ_{1n}	10	$\frac{\text{cm}^2}{(V)(s)}$
μ_{1p}	10	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2n}	152	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2p}	147	$\frac{\text{cm}^2}{(V)(s)}$
α_n	0	no unit
α_p	0	no unit
β_n	−1.6	no unit
β_p	−1.98	no unit
γ_n	−0.568	no unit
γ_p	0	no unit
δ_n	0.8	no unit
δ_p	0.85	no unit
N_{CRITn}	4.4×10^{18}	cm^{-3}
N_{CRITp}	10^{18}	cm^{-3}

Table 6. Indium Phosphide Mobility Modeling Constants. Adapted from [15].

Parameter	Value	Units
μ_{1n}	400	$\frac{\text{cm}^2}{(V)(s)}$
μ_{1p}	10	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2n}	5200	$\frac{\text{cm}^2}{(V)(s)}$
μ_{2p}	170	$\frac{\text{cm}^2}{(V)(s)}$
α_n	0	no unit
α_p	0	no unit
β_n	−2	no unit
β_p	−2	no unit
γ_n	−1.5275	no unit
γ_p	−1.86	no unit
δ_n	0.47	no unit
δ_p	0.62	no unit
N_{CRITn}	3×10^{17}	cm^{-3}
N_{CRITp}	4.87×10^{17}	cm^{-3}

No information exists to model AlP. Static values of 60 $cm^2/V{\cdot}s$ for μ_n and 450 $cm^2/V{\cdot}s$ for μ_p are assumed based on data available in [16]. This information is not sourced and is only used due to the absence of better available information. As more research into this material is done, this should be updated for a more accurate model.

Sutherland and Hauser argue that to account for composition, hole mobility be calculated using [14]

$$\mu_p\left(N,C\right)=\frac{\mu_{2p}\left(m_{p2}^*\right)^{1.5}\left(\frac{1}{\varepsilon_{h2}}-\frac{1}{\varepsilon_{l2}}\right)}{\left(m_p^*\right)^{1.5}\left(\frac{1}{\varepsilon_h}-\frac{1}{\varepsilon_l}\right)}. \tag{36}$$

In Equation (36), μ_{2p} is the doping dependent mobility of the second material (GaAs in AlGaAs) given by Equation (35), N is the doping concentration, C is the molar fraction of the first material (AlAs in AlGaAs), m_{p2}^* is the effective mass of holes in the second material, and ε_{h2} and ε_{l2} are static dielectric constants of material 2 [14]. The parameter m_p^* is an effective mass for holes in the composite material, which has a value calculated by [14] as

$$\frac{1}{m^*}=\frac{C}{m_1^*}+\frac{1-C}{m_2^*}. \tag{37}$$

Both ε_h and ε_l are dielectric constants that are functions of composition with values calculated by [14] as

$$\varepsilon=\frac{1+2\left[C\left(\frac{\varepsilon_1-1}{\varepsilon_1+2}\right)+\left(1-C\right)\left(\frac{\varepsilon_2-1}{\varepsilon_2+2}\right)\right]}{1-C\left(\frac{\varepsilon_1-1}{\varepsilon_1+2}\right)-\left(1-C\right)\left(\frac{\varepsilon_2-1}{\varepsilon_2+2}\right)}. \tag{38}$$

Holes have both a heavy and light effective mass. The combined effective mass for the heavy (m_{hp}^*) and light (m_{lp}^*) holes used in Equation (37) is found from [15]

$$m_p^*=\left(m_{lp}^{*1.5}+m_{hp}^{*1.5}\right)^{\frac{2}{3}}. \tag{39}$$

The equation Sutherland and Hauser give for electron mobility is [14]

$$\mu_n(N,C) = \mu_d R_d + \mu_i (1 - R_d) . \tag{40}$$

Electron mobility is split between a direct μ_d and an indirect μ_i term. These terms are calculated in [14] using

$$\mu_d(N,C) = \frac{\mu_{2n}\left(m^*_{nd2}\right)^{1.5}\left(\frac{1}{\varepsilon_{h2}} - \frac{1}{\varepsilon_{l2}}\right)}{\left(m^*_{nd}\right)^{1.5}\left(\frac{1}{\varepsilon_h} - \frac{1}{\varepsilon_l}\right)} \tag{41}$$

and

$$\mu_i(N,C) = \frac{\mu_{1n}\left(m^*_{ni2}\right)^{1.5}\left(\frac{1}{\varepsilon_{h1}} - \frac{1}{\varepsilon_{l1}}\right)}{\left(m^*_{ni}\right)^{1.5}\left(\frac{1}{\varepsilon_h} - \frac{1}{\varepsilon_l}\right)} . \tag{42}$$

In Equations (41) and (42), m^*_{nd} and m^*_{ni} are the direct and indirect effective masses, respectively, of electrons. All other terms are analogous and calculated the same way as in Equation (36), with a subscript of 1 indicating the parameter is a property of the first material and a subscript of 2 denoting that the parameter is a property of the second material [14]. The ratio between these terms R_d is calculated with [14]

$$R_d = \frac{1}{1 + \left(\frac{m^*_{ni}}{m^*_{nd}}\right)^{1.5} \exp\left(\frac{E_{gd} - E_{gi}}{kT}\right)} . \tag{43}$$

The parameter T in Equation (43) is temperature in Kelvin, k is Boltzmann's constant, and E_{gd} and E_{gi}, are, respectively, the direct and indirect bandgaps [14]. Vurgaftman et al. [17] has found

$$E_g = (1 - C) E_{g1} + C E_{g2} - C(1 - C)\beta \tag{44}$$

to be the correct calculation of composite bandgap. The bowing parameter β was used to give a more accurate model than a linear interpolation between the two materials' bandgaps (both direct and indirect) [17].

Quaternary mobilities are calculated in the same manner, with the first and second material being the tertiary composites that are blended together. For example, for InAlGaP, first aluminum indium phosphide (AlInP) is solved for, followed by InGaP. These two blends are then the first and second material used in Equations (36), (41), and (42). Equation (44) was still used to calculate the composite bandgaps of the quaternary.

The base material properties are given in Table 7 through Table 11. Due to the large number of materials and the large number of parameters needed for each material, numerous sources were used solely for these reference values. These sources include a book on III-V compound semiconductors by Madelung et al. [18], limited research conducted by Saliev [19] on the properties of AlP, and the NSM database [20], which contains various semiconductor properties.

Table 7. Mobility Parameters for GaAs

Parameter	Value	Units	Source
m_{nd}^*	0.067	no unit	[17]
m_{ni}^*	0.85	no unit	[17]
m_{lp}^*	0.082	no unit	[20]
m_{hp}^*	0.51	no unit	[20]
ε_h	10.89	no unit	[20]
ε_l	13.2	no unit	[18]
E_{gd}	1.1519	eV	[17]
E_{gi}	1.981	eV	[17]

Table 8. Mobility Parameters for AlAs

Parameter	Value	Units	Source
m_{nd}^{*}	0.15	no unit	[17]
m_{ni}^{*}	0.19	no unit	[18]
m_{lp}^{*}	0.16	no unit	[18]
m_{hp}^{*}	0.81	no unit	[18]
ε_{h}	8.16	no unit	[18]
ε_{l}	12	no unit	[18]
E_{gd}	3.099	eV	[17]
E_{gi}	2.24	eV	[17]

Table 9. Mobility Parameters for AlP

Parameter	Value	Units	Source
m_{nd}^{*}	0.22	no unit	[17]
m_{ni}^{*}	0.793	no unit	[19]
m_{p}^{*}	0.7	no unit	[16]
ε_{h}	8.06	no unit	[18]
ε_{l}	9.8	no unit	[16]
E_{gd}	3.63	eV	[17]
E_{gi}	2.52	eV	[17]

Table 10. Mobility Parameters for GaP

Parameter	Value	Units	Source
m_{nd}^{*}	0.13	no unit	[17]
m_{ni}^{*}	1.12	no unit	[20]
m_{lp}^{*}	0.14	no unit	[20]
m_{hp}^{*}	0.79	no unit	[20]
ε_{h}	9.11	no unit	[20]
ε_{l}	11.1	no unit	[16]
E_{gd}	2.87	eV	[17]
E_{gi}	2.35	eV	[17]

Table 11. Mobility Parameters for InP

Parameter	Value	Units	Source
m_{nd}^{*}	0.0795	no unit	[17]
m_{ni}^{*}	0.88	no unit	[17]
m_{lp}^{*}	0.089	no unit	[20]
m_{hp}^{*}	0.6	no unit	[20]
ε_{h}	9.61	no unit	[20]
ε_{l}	12.5	no unit	[20]
E_{gd}	1.4236	eV	[20]
E_{gi}	2.273	eV	[17]

Bowing parameters for the tertiary and quaternary materials are presented in Table 12. The parameter C in this table refers to molar concentration.

Table 12. Bowing Parameters for Alloy Bandgaps. Adapted from [17].

Material	Bandgap	Bowing Parameter
AlGaAs	E_{gd}	$1.31C - .127$
	E_{gi}	0.055
AlInP	E_{gd}	−0.48
	E_{gi}	0.38
InGaP	E_{gd}	0.65
	E_{gi}	0.2
InAlGaP	E_{gd}	0.18
	E_{gi}	E_{gi} remains constant with a value equal to the value of E_{gd} at C = 0.55

E. RADIATION MODEL

A radiation model was integrated into the Silvaco scripts using a radiation modeling tool [21] created at NPS. This tool takes all relevant data discussed in the subsections of this section and generates displacement trap density profiles for Silvaco to reference. As each atom in each material has both a profile for the interstitial dislocation and the vacancy dislocation, many files are created. To keep the number of files manageable, only the solar cell regions have these traps taken into account. This results in ten files being created per simulation, two each for the indium, gallium and phosphide in the top InGaP cell and two

each for the gallium and arsenide in the GaAs bottom cell. This tool works by combining data specified by the user with information from the ESTAR electron stopping power database [22] to calculate the stopping power at every point within the entire cell. From this, an energy profile of the electron radiation was generated for each point in the cell. In the solar cell regions, the profile was used in conjunction with data from the SR-NIEL online database [23] to calculate the NIEL in these regions. The NIEL values were then used to create the trap density files for Silvaco.

1. Density

Each binary material in the cell has a density the radiation modeling tool uses to calculate stopping power in that region. The density of materials is commonly available; the exact values used are displayed in Table 13. A linear interpolation of density is then calculated using the molar concentrations of the binaries in the tertiary or quaternary material.

Table 13. Material Densities Used

Material	Density (grams/cm^3)
InP	4.81
GaP	4.14
AlP	2.85
GaAs	5.32
AlAs	3.71

2. Materials

This input requires the names of the unique materials used in the cell in a format that give the atomic symbol of the element and its relative weight in the material. For example, InAlGaP is specified as “In_0.235_Al_0.1855_Ga_0.0759_P_0.5”.

3. Displacement Energy Threshold

The threshold energy required to displace an atom from its lattice site is specified here. Values for the energy required to displace a gallium or arsenide atom from GaAs were found by D. Pons et al. [24] to be 10.0 eV and 15.5 eV, respectively. Threshold energies for AlGaAs were found by K. Gartner [25] to be 23.0 eV, 14.5 eV, and 15.5 eV. For InGaP, Y. Okuno's team [26] conducted a study that placed the displacement threshold energies at 4.0 eV, 10.0 eV, and 9.0 eV. No information exists for InAlGaP, so energies near the values for the individual atoms in the other materials was used. The energies used were 4.0 eV, 20.0 eV, 10.0 eV, and 9.0 eV. The accuracy of these estimates are of little consequence as they are used to calculate the NIEL in an area where the displacement damage is not modeled and only used to obtain an accurate energy profile. As NIEL is an extremely small portion of the energy lost by an electron compared to radiative and ionizing loss, any reasonable numbers can be used.

4. Trap Type, Energy, and Capture Cross-section

These parameters, like the parameters for mobility, were available for GaAs, but a complete list of these values does not exist for InGaP, the other material for which traps were calculated. Schultz and Lilienfeld [27], [28] calculated energy levels for gallium and arsenide interstitial and vacancy defects in GaAs as well as for InP and GaP using ab-initio simulation [27], [28]. The values for InP and GaP were used to estimate the energy levels for the traps in InGaP. Values used for the trap energy levels are contained in Table 14. Energies are given in eV, I denotes the interstitial trap energy, and V indicates the vacancy trap energy. The type of trap, acceptor or donor, was determined by looking at the location of the energy level. Those closest to the conduction band were considered acceptors, and those near the valence band were considered donors. The capture cross sections for the gallium and arsenide interstitial defects in GaAs and the phosphide interstitial defect in InGaP were also found in [27] and [28] and are shown in Table 15. All other capture cross sections were defined to be 10^{14} cm^2 as a best guess (a value in the range of those found for materials for which values exist).

Table 14. Trap Energy Levels. Adapted from [27], [28].

GaAs				InGaP					
Ga		As		In		Ga		P	
I	V	I	V	I	V	I	V	I	V
0.7	0.27	0.35	0.36	1.2	0.5	1.2	0.5	0.89	1

Table 15. Trap Capture Cross-Sections. Adapted from [27], [28].

Interstitial	Capture Cross-section (cm^2)
Gallium in GaAs	1.9×10^{-12}
Arsenide in GaAs	6.2×10^{-15}
Phosphide in InGaP	4×10^{-14}

5. Radiation Initial Energy

Electron radiation energy exists at a wide array of values in space. It is common practice, however, to use a value of 1.0 MeV as a representative value for testing space application electronics. The radiation modeling tool only allows for a single energy value, making 1.0 MeV the obvious choice. Displacement damage, the damage mechanism of interest, is highly energy dependent; future work can improve the radiation modeling tool to account for the entire range of electron radiation energy values to give a more accurate result.

6. Radiation Flux

To find the total radiation flux, charts presented in [10], specifically the charts based on the AE8 data on electron radiation developed by NASA, were integrated across all energies and averaged across all orbit inclinations. This has the effect of giving a representative flux at a given distance from Earth as a delta function of one energy

(1.0 MeV) instead of the actual distribution. These charts are presented as Figure 31 and Figure 32. Values of 1.2×10^{11} cm^2 s^{-1} and 2.5×10^{9} cm^2 s^{-1} were calculated for LEO and GEO, respectively.

Trapped Electron Fluxes, LEO, Solar Minimum

E(>MEV)	300 KM INCLINATION			500 KM INCLINATION		
	28.5 DEG	60 DEG	90 DEG	28.5 DEG	60 DEG	90 DEG
0.04	2.973E+08	3.203E+09	2.971E+09	5.153E+09	9.171E+09	7.876E+09
0.07	2.351E+08	2.391E+09	2.257E+09	4.082E+09	7.007E+09	6.066E+09
0.10	1.861E+08	1.795E+09	1.730E+09	3.236E+09	5.382E+09	4.712E+09
0.20	5.629E+07	6.779E+08	7.424E+08	9.975E+08	1.908E+09	1.816E+09
0.30	2.227E+07	3.631E+08	4.262E+08	3.969E+08	9.484E+08	9.514E+08
0.40	1.144E+07	2.384E+08	2.849E+08	2.017E+08	5.695E+08	6.029E+08
0.50	5.897E+06	1.616E+08	1.950E+08	1.030E+08	3.807E+08	3.944E+08
0.60	3.985E+06	1.283E+08	1.526E+08	6.850E+07	2.917E+08	3.007E+08
0.70	2.701E+06	1.027E+08	1.204E+08	4.574E+07	2.258E+08	2.315E+08
0.80	1.948E+06	8.399E+07	9.744E+07	3.268E+07	1.813E+08	1.845E+08
0.90	1.494E+06	7.001E+07	8.051E+07	2.494E+07	1.504E+08	1.515E+08
1.00	1.147E+06	5.850E+07	6.669E+07	1.904E+07	1.252E+08	1.248E+08
1.25	7.213E+05	3.857E+07	4.262E+07	1.179E+07	8.119E+07	7.931E+07
1.50	4.549E+05	2.554E+07	2.742E+07	7.310E+06	5.292E+07	5.076E+07
1.75	3.051E+05	1.747E+07	1.828E+07	4.870E+06	3.561E+07	3.371E+07
2.00	2.053E+05	1.199E+07	1.224E+07	3.250E+06	2.407E+07	2.250E+07
2.25	1.392E+05	8.275E+06	8.289E+06	2.194E+06	1.644E+07	1.516E+07
2.50	9.419E+04	5.725E+06	5.637E+06	1.484E+06	1.127E+07	1.026E+07
2.75	3.788E+04	3.899E+06	3.751E+06	5.934E+05	7.373E+06	6.610E+06
3.00	1.521E+04	2.695E+06	2.529E+06	2.405E+05	4.956E+06	4.361E+06
3.25	4.850E+03	1.856E+06	1.695E+06	7.591E+04	3.324E+06	2.862E+06
3.50	1.357E+03	1.292E+06	1.148E+06	2.394E+04	2.274E+06	1.914E+06
3.75	3.874E+02	8.495E+05	7.316E+05	7.263E+03	1.474E+06	1.206E+06
4.00	0.000E+00	5.650E+05	4.726E+05	8.860E+02	9.693E+05	7.712E+05
4.50	0.000E+00	2.066E+05	1.643E+05	0.000E+00	3.493E+05	2.633E+05
5.00	0.000E+00	6.828E+04	5.129E+04	0.000E+00	1.143E+05	7.979E+04
5.50	0.000E+00	1.572E+04	1.188E+04	0.000E+00	2.659E+04	1.751E+04
6.00	0.000E+00	2.858E+03	1.970E+03	0.000E+00	3.923E+03	2.470E+03
6.50	0.000E+00	0.000E+00	0.000E+00	0.000E+00	1.235E+02	6.052E+01
7.00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00

Figure 31. Trapped Electron Fluxes. Source: [10].

Geostationary Electron Fluxes

E(>MEV)	70 DEG W	160 DEG W
0.04	3.775E+07	4.643E+07
0.07	3.023E+07	3.847E+07
0.10	2.421E+07	3.188E+07
0.20	1.145E+07	1.587E+07
0.30	5.944E+06	8.575E+06
0.40	3.383E+06	5.044E+06
0.50	1.925E+06	2.967E+06
0.60	1.224E+06	2.048E+06
0.70	7.788E+05	1.414E+06
0.80	5.290E+05	9.879E+05
0.90	3.838E+05	6.983E+05
1.00	2.784E+05	4.935E+05
1.25	1.338E+05	2.475E+05
1.50	6.435E+04	1.242E+05
1.75	3.497E+04	7.171E+04
2.00	1.900E+04	4.142E+04
2.25	9.313E+03	2.128E+04
2.50	4.653E+03	1.093E+04
2.75	2.816E+03	6.494E+03
3.00	1.737E+03	3.858E+03
3.25	1.118E+03	2.484E+03
3.50	7.196E+02	1.600E+03
3.75	4.260E+02	8.527E+02
4.00	2.522E+02	4.546E+02
4.50	6.825E+01	1.187E+02
5.00	1.673E+00	4.519E+00
5.50	0.000E+00	0.000E+00
6.00	0.000E+00	0.000E+00
6.50	0.000E+00	0.000E+00
7.00	0.000E+00	0.000E+00

Figure 32. Geostationary Electron Fluxes. Source: [10].

F. OPTIMIZATION

The method of genetic algorithm optimization is an iterative process. This iterative property means that to conduct thousands of simulations takes a large amount of time. Instead of this, the optimization of this cell was done using the nearly orthogonal Latin

hypercube tool created by Sanchez [29]. This tool predetermines any number of test parameters that are uniformly spaced across all dimensions to equally sample all areas of the design space. These predetermined points have the property of orthogonality, meaning that interactions between the parameters can be determined after the simulations are run.

The tool requires a high and low boundary condition for every parameter. In order to ensure the optimum falls within these boundaries, a broad range was chosen. For each thickness, the minimum value was 20% the original value in the cell presented by [4] and the maximum was double the initial value. The doping concentrations for each layer were given a minimum value of 10^{16} cm^{-3} and a maximum of $3.2\times10^{19}\,cm^{-3}$, scaled logarithmically. For the number of points to test, 2056 was chosen somewhat arbitrarily, based on this being a large number to partially fill the design space and a multiple of 257, the number of points the tool provides for a single rotation of the variables.

From this set of 2056 test candidates, an equal number of Silvaco files were created. These files, as well as the supporting optical files, were sent to the Hamming Supercomputer. The Hamming allows multiple simultaneous processes to run in parallel. Running 15 simulations at a time (limited by the number of Silvaco licenses available at NPS), the total time to run all simulations was under three hours.

The resulting log files were then downloaded from the Hamming and parsed to compile the output data for all of the simulations. Some simulations resulted in very low efficiencies, either negative or well below 1%. These occurred when the simulation did not work and Silvaco returned partial results from the top cell of the dual-junction only. As the number of these nonworking cells was small compared to the number of cells with valid results (<1% of the test simulations), these results were purged without much investigation into the cause of the problem.

The results were then input into JMP, a statistical analysis tool. This tool was used to fit each parameter to a quadratic model vs efficiency. JMP then created a predicted optimum set of values for the parameters. A screenshot of JMP being used to predict an optimum cell is presented in Figure 33. In Figure 33, the red numbers are the predicted optimum values, thicknesses in microns and doping concentrations in dopants per cubic centimeter.

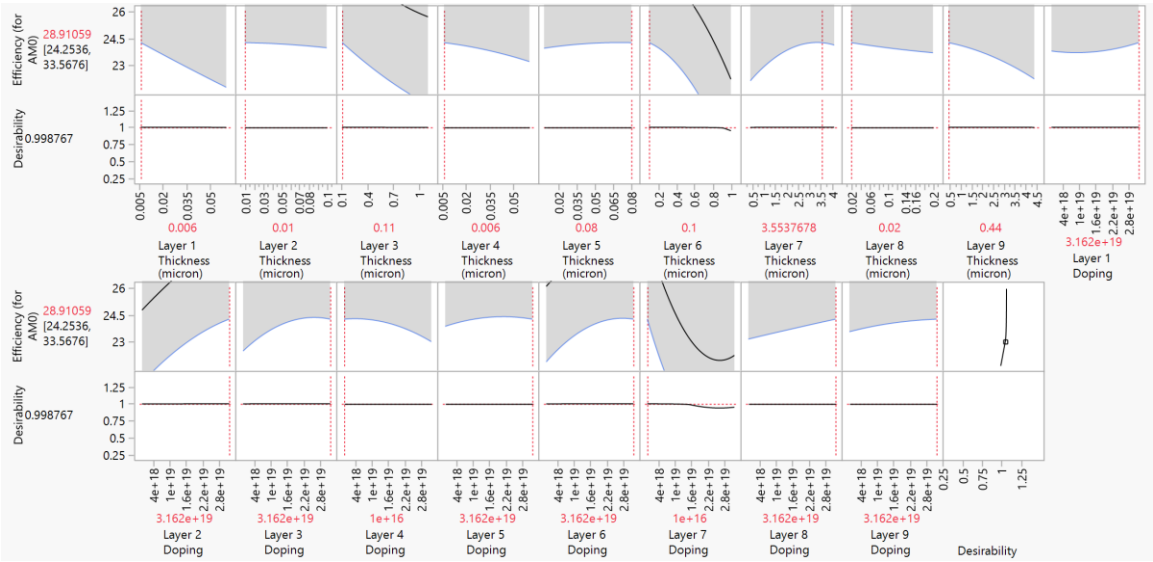


Figure 33. Predicted Optimum Example

This predicted optimum cell was then simulated in Silvaco. The result of the first optimization run was always a much worse cell than the initial model. Initial optimization runs showed the values predicted are all edge values, either the maximum or minimum possible value for the parameter. This was taken to mean the initial range was too large for the number of points tested. The predicted optimum values were used then to shrink the range of each parameter, halving or nearly halving it in the direction of the prediction. With these as the new highest and lowest boundary values, another set of 2056 test points was created, and the process repeated.

The second run always (for every radiation condition) returned a cell that improved on the model cell but with many parameters still pegged high or low. At this point, a further shrinking of the test space was done. This was done in the same manner as before for boundary predicted optimums and in a narrow range about the predicted optimum for predictions that were not on a boundary. The results of this third set of simulations resulted in a cell slightly improved from the second set. Further iterations of this process would theoretically continue to approach a true optimum.

IV. RESULTS

A. INITIAL CELL MODELING

To verify the accuracy of the model, the values for the two degrees of freedom, anode contact resistance and the bottom tunnel junction doping concentration (used to control tunnel junction resistance), were found. A trial and error method was used until the Silvaco results matched the experimental data within 5%. The values found for these parameters were unique to this cell, and should not be used in any other. The anode resistance value was found to be 2e9 $\Omega \bullet cm$, while 1e11 cm^{-3} was found to be the doping concentration that gave a resistance for the tunnel junction that resulted in a matching of simulated output parameters with experimental data. The closeness of fit to the experimental data is presented in Table 16 and Figure 34.

Table 16. Results of Model versus Experimental Data. Adapted from [4].

	Experimental	Model	Difference (%)
Jsc (mA/cm^2)	13.08	12.53	-4.2
Voc (V)	2.34	2.38	1.7
FF (%)	82.5	86.58	4.9
η (%)	18.6	19.3	3.8

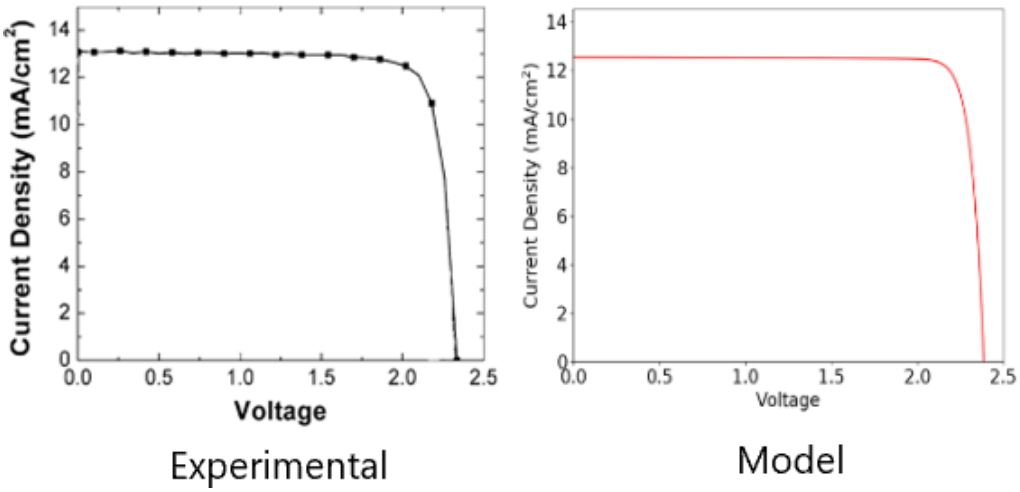


Figure 34. Results of Model versus Experimental Current-Voltage Curves. Adapted from [4].

B. PRE-IRRADIATION OPTIMIZATION

Given the process outlined in the optimization section of methodology, the first pass at optimization resulted in a reduction in efficiency to 17.78% from an initial value of 19.34%. The second optimization resulted in an improvement to 19.43%, and a final pass gave an efficiency of 22.73%, an improvement from the base model of 3.39%. The optimum parameter values for this optimized cell are contained in Table 17.

Table 17. Pre Irradiation Optimal Cell Parameters

Layer 1: InAlGaP	0.006 microns	10^{19} cm^{-3}
Layer 2: InGaP	0.1 microns	$2.09\times10^{19} cm^{-3}$
Layer 3: InGaP	0.3 microns	10^{19} cm^{-3}
Layer 4: InAlGaP	0.006 microns	$3.16\times10^{18} cm^{-3}$
Layer 7: InGaP	0.008 microns	$3.16\times10^{19} cm^{-3}$
Layer 8: GaAs	0.7 microns	10^{18} cm^{-3}
Layer 9: GaAs	2.5 microns	$3.16\times10^{17} cm^{-3}$
Layer 10: AlGaAs	0.06 microns	$3.16\times10^{18} cm^{-3}$
Layer 11: GaAs	0.8 microns	$3.16\times10^{18} cm^{-3}$

C. LOW EARTH ORBIT PERFORMANCE

For low earth orbit characteristics, a mission length of 12 years was chosen. The non-optimized model cell was then simulated with the radiation damage it would sustain at this orbit for this length of time. Predictably, efficiency dropped to 17.57%.

The first, second, and third optimization passes resulted in efficiencies of 12.33%, 21.08%, and 21.50%, respectively. This is a total gain of 3.93% in efficiency. The parameters for the optimum cell are shown in Table 18.

Table 18. Low Earth Orbit Optimum Cell

Layer 1: InAlGaP	0.006 microns	$3.16\times10^{18}\,cm^{-3}$
Layer 2: InGaP	0.025 microns	$2.57\times10^{19}\,cm^{-3}$
Layer 3: InGaP	0.35 microns	$1.51\times10^{19}\,cm^{-3}$
Layer 4: InAlGaP	0.006 microns	$10^{18}\ cm^{-3}$
Layer 7: InGaP	0.02 microns	$3.16\times10^{18}\,cm^{-3}$
Layer 8: GaAs	0.8 microns	$10^{18}\ cm^{-3}$
Layer 9: GaAs	3.2 microns	$6.31\times10^{17}\,cm^{-3}$
Layer 10: AlGaAs	0.05 microns	$3.16\times10^{18}\,cm^{-3}$
Layer 11: GaAs	0.8 microns	$3.16\times10^{19}\,cm^{-3}$

D. GEOSYNCHRONOUS ORBIT PERFORMANCE

For the geosynchronous data, again a mission length of 12 years was used. The lower radiation flux of a geosynchronous orbit resulted in efficiencies that mimicked the pre-irradiation cell very closely. The initial model had an efficiency of 19.28% (slightly lower than the pre irradiation efficiency of 19.34%). The values of efficiency after the first, second, and final passes were 14.37%, 21.84%, and 22.19%, respectively. This final optimum efficiency is slightly lower than the non-irradiated optimum efficiency and much greater than the low earth orbit optimum efficiency, exactly as expected. Parameters found to be optimum for this solar cell at this orbit are contained in Table 19.

Table 19. Geo Synchronous Optimum Parameters

Layer 1: InAlGaP	0.006 microns	$1.58\times10^{18}\,cm^{-3}$
Layer 2: InGaP	0.05 microns	$3.16\times10^{19}\,cm^{-3}$
Layer 3: InGaP	0.31 microns	$2.09\times10^{19}\,cm^{-3}$
Layer 4: InAlGaP	0.015 microns	$5.01\times10^{17}\,cm^{-3}$
Layer 7: InGaP	0.02 microns	$3.16\times10^{19}\,cm^{-3}$
Layer 8: GaAs	0.45 microns	$10^{19}\ cm^{-3}$
Layer 9: GaAs	1.9 microns	$3.16\times10^{17}\,cm^{-3}$
Layer 10: AlGaAs	0.07 microns	$5.01\times10^{18}\,cm^{-3}$
Layer 11: GaAs	0.44 microns	$1.58\times10^{18}\,cm^{-3}$

E. CHANGES IN OPTIMUM PARAMETERS WITH RADIATION DAMAGE

The optimum cell pre irradiation was also tested with the damage equivalent to 12 years in a low earth orbit, with an end of life efficiency of 21.492%, nearly identical but slightly lower than this optimum cell (taken to three decimal places, it has an efficiency of 21.497%). This cell's parameters, with no radiation taken into account, result in an efficiency of 22.42%, smaller than the optimum for a non-irradiated cell.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND FUTURE WORK

This work started with four goals. The first goal was to verify modeling capability by matching a real world solar cell to a model cell with the same parameters. With these models, a cell was simulated using Silvaco ATLAS, matching real world data within 5% along four separate outputs: short circuit current, open circuit voltage, fill factor, and efficiency. While this solution allowed this model to be validated, it is obviously better to have a full model for the tunnel junction. The resistance value used was accurate only for this cell. A full tunnel junction model will not only allow optimization of this component but also allow new dimensions of optimization, such as adding or changing whole solar cell junctions.

The second goal was to implement a radiation model to test end-of-life efficiency. This model, and its integration into simulations, was a resounding success. Data matched exactly with predicted outcomes, with radiation damage reducing cell efficiency. This is closely tied into the third goal of optimizing a solar cell for end-of-life performance. Not only was this done, but based on this model, it was shown that the cell with the highest efficiency at beginning of life is not the cell that has the highest efficiency at the end of life. This unintuitive result should be immediately validated by further research. If this conclusion holds true across many different designs of solar cells, manufacturing of space application solar cells should be modified to account for this.

The last goal, the goal of building a tool kit capable of automating this process across a dimensions of optimization used in this thesis as well as future potential dimensions of optimization has also been successful. The tools created are capable of determining an optimum cell under any radiation conditions within a day. The tools used were made overly generic, allowing optimization of molar compositions of the materials used or, with few changes, optimization of a completely new cell design.

Further research should focus on four areas. The most obvious would be extending the parameters optimized to include new vectors, such as molar concentration of the materials in the cell. The second sphere of research would be to conduct fundamental

physics research on exotic semiconductor materials. Many values for the radiation model, and some values for the mobility model, were assumed values because research does not currently exist on the actual values. The third major research area would be model improvement. A more accurate tunnel-junction model was discussed, but it is not the only opportunity for improvement. The cell modeled for this thesis was essentially a one-dimension cell; a more realistic, but much more computationally expensive, model would be a two-dimension cell with a real top contact, or even a three-dimension model. This would show the effects of current moving longer distances to get to the contacts. The last focus area for further research is to actually manufacture the cells designed in this work and perform real-world testing on them to validate the predicted results.

APPENDIX A. SAMPLE SILVACO SCRIPT WITH RADIATION

```
go atlas simflags = "-P 8"

mesh

x.mesh location   = 0.000000 spacing  = 0.333333
x.mesh location   = 1.000000 spacing  = 0.333333

y.mesh location   = 0.000000 spacing = 0.003930
y.mesh location   = 0.031440 spacing = 0.000786
y.mesh location   = 0.039300 spacing = 0.000393
y.mesh location   = 0.056060 spacing = 0.001676
y.mesh location   = 0.081200 spacing = 0.008380
y.mesh location   = 0.106340 spacing = 0.001676
y.mesh location   = 0.123100 spacing = 0.000838
y.mesh location   = 0.167520 spacing = 0.004442
y.mesh location   = 0.234150 spacing = 0.022210
y.mesh location   = 0.300780 spacing = 0.004442
y.mesh location   = 0.345200 spacing = 0.002221
y.mesh location   = 0.353780 spacing = 0.000858
y.mesh location   = 0.366650 spacing = 0.004290
y.mesh location   = 0.379520 spacing = 0.000858
y.mesh location   = 0.388100 spacing = 0.000429
y.mesh location   = 0.393100 spacing = 0.000500
y.mesh location   = 0.400600 spacing = 0.002500
y.mesh location   = 0.408100 spacing = 0.000500
y.mesh location   = 0.413100 spacing = 0.000250
y.mesh location   = 0.418100 spacing = 0.000500
y.mesh location   = 0.425600 spacing = 0.002500
y.mesh location   = 0.433100 spacing = 0.000500
y.mesh location   = 0.438100 spacing = 0.000250
y.mesh location   = 0.443020 spacing = 0.000492
y.mesh location   = 0.450400 spacing = 0.002460
y.mesh location   = 0.457780 spacing = 0.000492
y.mesh location   = 0.462700 spacing = 0.000246
y.mesh location   = 0.538240 spacing = 0.007554
y.mesh location   = 0.651550 spacing = 0.037770
y.mesh location   = 0.764860 spacing = 0.007554
y.mesh location   = 0.840400 spacing = 0.003777
y.mesh location   = 1.424000 spacing = 0.058360
y.mesh location   = 2.299400 spacing = 0.291800
y.mesh location   = 3.174800 spacing = 0.058360
y.mesh location   = 3.758400 spacing = 0.029180
```

y.mesh location = 3.792640 spacing = 0.003424
y.mesh location = 3.844000 spacing = 0.017120
y.mesh location = 3.895360 spacing = 0.003424
y.mesh location = 3.929600 spacing = 0.001712
y.mesh location = 4.039260 spacing = 0.010966
y.mesh location = 4.477900 spacing = 0.054830

region num = 1 material = InAlGaP x.min = 0.0 x.max = 1.0 y.min = 0.000000 y.max = 0.039300 x.comp = 0.371 y.comp = 0.159
region num = 2 material = InGaP x.min = 0.0 x.max = 1.0 y.min = 0.039300 y.max = 0.123100 x.comp = 0.49
region num = 3 material = InGaP x.min = 0.0 x.max = 1.0 y.min = 0.123100 y.max = 0.345200 x.comp = 0.49
region num = 4 material = InAlGaP x.min = 0.0 x.max = 1.0 y.min = 0.345200 y.max = 0.388100 x.comp = 0.371 y.comp = 0.159
region num = 5 material = GaAs x.min = 0.0 x.max = 1.0 y.min = 0.388100 y.max = 0.413100
region num = 6 material = GaAs x.min = 0.0 x.max = 1.0 y.min = 0.413100 y.max = 0.438100
region num = 7 material = InGaP x.min = 0.0 x.max = 1.0 y.min = 0.438100 y.max = 0.462700 x.comp = 0.49
region num = 8 material = GaAs x.min = 0.0 x.max = 1.0 y.min = 0.462700 y.max = 0.840400
region num = 9 material = GaAs x.min = 0.0 x.max = 1.0 y.min = 0.840400 y.max = 3.758400
region num = 10 material = AlGaAs x.min = 0.0 x.max = 1.0 y.min = 3.758400 y.max = 3.929600 x.comp = 0.7
region num = 11 material = GaAs x.min = 0.0 x.max = 1.0 y.min = 3.929600 y.max = 4.477900

electrode name = anode top
electrode name = cathode bottom
electrode name = TJ_Top material = GaAs x.min = 0 x.max = 1.0 y.min = 0.388100 y.max = 0.415600
electrode name = TJ_Bottom material = GaAs x.min = 0 x.max = 1.0 y.min = 0.435600 y.max = 0.438100

doping p.type uniform concentration = 8.737758e+17 region = 1
doping p.type uniform concentration = 7.009709e+17 region = 2
doping n.type uniform concentration = 1.655008e+16 region = 3
doping n.type uniform concentration = 1.363955e+17 region = 4
doping n.type uniform concentration = 2e19 region = 5
doping p.type uniform concentration = 1e11 region = 6
doping p.type uniform concentration = 1.851825e+19 region = 7
doping p.type uniform concentration = 6.378227e+17 region = 8

doping n.type uniform concentration = 2.119337e+17 region = 9
doping n.type uniform concentration = 1.547035e+17 region = 10
doping n.type uniform concentration = 1.154782e+19 region = 11

material mun = 51.239071 mup = 70.917663 region = 1
material mun = 1837.857246 mup = 72.481755 region = 2
material mun = 3332.813147 mup = 136.447213 region = 3
material mun = 51.731188 mup = 112.499094 region = 4
material mun = 793.005146 mup = 22.527853 region = 7
material mun = 258.726820 mup = 85.157429 region = 10

material mat = InAlGaP index.file = AlGaInP.nk
material mat = InGaP index.file = InGaP_ex.nk
material mat = AlGaAs index.file = AlGaAs.nk

material material=InGaP affinity = 4.08
material material=AlGaAs affinity = 3.54

contact name = TJ_Top resistance = 1e18
contact name = TJ_Bottom resistance = 1e18
contact name = anode resistance = 2e9

trap acceptor e.level = 1.200000 degen = 1 sign = 1.000000e-14 sigp =1.000000e-14 f.density = 20_1246_2_In_Vprofile.lib
trap donor e.level = 0.500000 degen = 1 sign = 1.000000e-14 sigp =1.000000e-14 f.density = 20_1246_2_In_Iprofile.lib
trap acceptor e.level = 1.200000 degen = 1 sign = 1.000000e-14 sigp =1.000000e-14 f.density = 20_1246_2_Ga_Vprofile.lib
trap donor e.level = 0.500000 degen = 1 sign = 1.000000e-14 sigp =1.000000e-14 f.density = 20_1246_2_Ga_Iprofile.lib
trap acceptor e.level = 0.890000 degen = 1 sign = 4.000000e-14 sigp =4.000000e-14 f.density = 20_1246_2_P_Vprofile.lib
trap acceptor e.level = 1.000000 degen = 1 sign = 1.000000e-14 sigp =1.000000e-14 f.density = 20_1246_2_P_Iprofile.lib
trap acceptor e.level = 0.700000 degen = 1 sign = 1.900000e-12 sigp =1.900000e-12 f.density = 20_1246_6_Ga_Vprofile.lib
trap donor e.level = 0.270000 degen = 1 sign = 1.000000e-14 sigp =1.000000e-14 f.density = 20_1246_6_Ga_Iprofile.lib
trap acceptor e.level = 0.350000 degen = 1 sign = 6.200000e-15 sigp =6.200000e-15 f.density = 20_1246_6_As_Vprofile.lib
trap donor e.level = 0.360000 degen = 1 sign = 1.000000e-14 sigp =1.000000e-14 f.density = 20_1246_6_As_Iprofile.lib

models region = 5 analytic
models region = 6 analytic

```
models region = 8       analytic
models region = 9       analytic
models region = 11      analytic

models print srh fermi optr auger bgn temp = 300.0

method newton itlimit = 50

beam num = 1 x.origin = 0.50 y.origin = -1  angle = 90 am0 wavel.start = 0.28 wavel.end
= 3.5 wavel.num = 500 reflect = 1

output con.band val.band opt.intens

solve init
solve b1 = 0.01
solve b1 = 0.10
solve b1 = 0.81

log outfile = testcell_20_1246.log
solve name = anode vanode = 0.000 vfinal = 0.100 vstep = 0.030
solve name = anode vanode = 0.100 vfinal = 1.500 vstep = 0.300
solve name = anode vanode = 1.500 vfinal = 2.800 vstep = 0.010
log off

quit
```

APPENDIX B. SAMPLE TRAP PROFILE FILE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <malloc.h>
#include <string.h>
#include <template.h>

int acc_trap(double x,double y,double z,double *density)
{
if(y<0.058900)
{
*density=0;
}
else if( (y >= 0.058900) && (y <= 0.843400) )
{
*density = 8.291865e-02*pow(y,2.0)+-1.773639e+07*y+6.246270e+10;
}
else if( y > 0.843400)
{
*density=0;
}
return(0);
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SILVACO SCRIPT GENERATOR

```
# -*- coding: utf-8 -*-
"""
Silvaco scripting

@author: jswalsh
"""

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import sys
from mobility import mobility_triplet,mobility_quad
import xlrd
import MultilayerNIELCalculator


# User Inputs

_Name = "Check"  # Model or testcell or Check
Batch = 22
Radiation = 0 # 0 or 1
Orbit = "LEO" # "LEO" or "GEO"
MissionLength = 12 # Length in years

# Convert from Logspace to a number

def _Doping(Power):
    a = Power % 1
    b = Power - a
    c = np.power(10.0,a)
    string = "%fe%d"%(c,b)
    doping = float(string)
    return doping

# Radiation Model

def _testdpar(Orbit,length_in_years):
    if Orbit == 'LEO':
        flux = 1.2e11
    elif Orbit == 'GEO':
        flux = 2.5e9
    else:
```

```
        print("Orbit not recognized")
        exit()
    testdpar = flux*length_in_years
    return testdpar

# Display Parameters

if _Name == 'testcell':
    View = 0
else:
    View = 1


# Computer parameters

if _Name == 'testcell':
    Cores = 8
else:
    Cores = 2

# Quality of life Parameters

V1 = .1
V2 = 1.5
Voc = 2.8
Vstep1 = .03
Vstep2 = .3
Vstep3 = .01

beam1 = .01
beam2 = .1

heavy = 100   # Mesh thickness
medium = 50
light = 10

# Static Parameters

Layer                                                                                    =
["InAlGaP","InGaP","InGaP","InAlGaP","InGaP","GaAs","GaAs","AlGaAs","GaAs"]
TJ_top_Thickness = float(.025)
TJ_top_Doping    = "2e19"
TJ_bot_Thickness = float(.025)
TJ_bot_Doping    = "1e11"
Temp = 300
```

```
Width = 1
Reflectivity = .19
RHeader = 2 # Number of rows for header in Parameters_x.xlsx
CHeader = 1 # Number of rows for header in Parameters_x.xlsx

# Setting up the optimization parameters

if _Name == "testcell":
   xl_workbook = xlrd.open_workbook("C:\\Users\\LT Walsh\\Documents\\Python
Scripts\\NOHL\\Parameters_%d.xlsx"%(Batch)) # Open the workbook
   xl_sheet = xl_workbook.sheet_by_index(0) # Using NOLH for up to 22 factors
if _Name == "Model":
   xl_workbook = xlrd.open_workbook("C:\\Users\\LT Walsh\\Documents\\Python
Scripts\\NOHL\\Model.xlsx") # Open the workbook
   xl_sheet = xl_workbook.sheet_by_index(0)
if _Name == "Check":
   xl_workbook = xlrd.open_workbook("C:\\Users\\LT Walsh\\Documents\\Python
Scripts\\NOHL\\Checkcell.xlsx") # Open the workbook
   xl_sheet = xl_workbook.sheet_by_index(0)

Runs = xl_sheet.nrows - RHeader   # Number of rows

Thickness = np.zeros((Runs,len(Layer)))
Doping = np.zeros((Runs,len(Layer)))

for row_idx in range(RHeader,Runs+RHeader):    # Offset for header
   for col_idx in range(CHeader,xl_sheet.ncols):  # Offset for blank column
      cell_obj = xl_sheet.cell(row_idx, col_idx)  # Get the cell
      if col_idx in range(CHeader,CHeader+len(Layer)):
         Thickness[row_idx-RHeader,col_idx-CHeader] = float(cell_obj.value)
      elif col_idx in range(CHeader+len(Layer),CHeader+(2*len(Layer))):
         Doping[row_idx-RHeader,col_idx-(CHeader+len(Layer))]                    =
_Doping(cell_obj.value)
Mobilityp = np.zeros((Runs,len(Layer)))
Mobilityn = np.zeros((Runs,len(Layer)))
for j in range(Runs):
   for i in range(len(Layer)):
      if Layer[i] == "InAlGaP":
         mat_1 = "AlP"
         mat_2 = "GaP"
         mat_3 = "InP"
         triplet_name_1 = "AlInP"
         triplet_name_2 = "GaInP"
         quad_name = "AlGaInP"
         C_inner = .7
```

```
            C_outer = .5
            N = float(Doping[j,i])
            Mobilityn[j,i],Mobilityp[j,i] =
mobility_quad(mat_1,mat_2,mat_3,triplet_name_1,triplet_name_2,quad_name,C_inner,
C_outer,Temp,N)
        elif Layer[i] == "InGaP":
            C = .51
            N = float(Doping[j,i])
            Mobilityn[j,i],Mobilityp[j,i] = mobility_triplet('GaP','InP','GaInP',C,Temp,N)
        elif Layer[i] == "AlGaAs":
            C = .7
            N = float(Doping[j,i])
            Mobilityn[j,i],Mobilityp[j,i] = mobility_triplet('AlAs','GaAs','AlGaAs',C,Temp,N)

# Radiation Model
if Radiation == 1:

    defaultcs = 1e-14

    densInP = 4.81
    densGaP = 4.14
    densAlP = 2.85
    densGaAs = 5.32
    densAlAs = 3.71

    material1 = "In_0.235_Al_0.1855_Ga_0.0759_P_0.5"
    material2 = "In_0.245_Ga_0.255_P_0.5"
    material3 = "Ga_0.5_As_0.5"
    material4 = "Al_0.35_Ga_0.15_As_0.5"

    density1 = 0.47*densInP+0.371*densAlP+0.159*densGaP
    density2 = 0.49*densInP+0.51*densGaP
    density3 = densGaAs
    density4 = 0.7*densAlAs+0.3*densGaAs

    Td1 = ['4','20','10','9']
    Td2 = ['4','10','9']
    Td3 = ['10','15.5']
    Td4 = ['23','14.5','15.5']

    traptype1 = [['A','D'],['A','A'],['A','D']] # not used
    traptype2 = [['A','D'],['A','D'],['A','A']]
    traptype3 = [['A','D'],['A','D']]
    traptype4 = [['A','D'],['A','A'],['A','D']] # not used
```

```
    trapener1 = [[1,1],[1,1],[1,1]] # not used
    trapener2 = [[1.2,.5],[1.2,.5],[.89,1]]
    trapener3 = [[.7,.27],[.35,.36]]
    trapener4 = [[1,1],[1,1],[1,1]] # not used

    trapcapn1 = [[defaultcs,defaultcs],[defaultcs,defaultcs],[defaultcs,defaultcs]] # not used
    trapcapn2 = [[defaultcs,defaultcs],[defaultcs,defaultcs],[4e-14,defaultcs]]
    trapcapn3 = [[1.9e-12,defaultcs],[6.2e-15,defaultcs]]
    trapcapn4 = [[defaultcs,defaultcs],[defaultcs,defaultcs],[defaultcs,defaultcs]] # not used

    trapcapp1 = [[defaultcs,defaultcs],[defaultcs,defaultcs],[defaultcs,defaultcs]] # not used
    trapcapp2 = [[defaultcs,defaultcs],[defaultcs,defaultcs],[4e-14,defaultcs]]
    trapcapp3 = [[1.9e-12,defaultcs],[6.2e-15,defaultcs]]
    trapcapp4 = [[defaultcs,defaultcs],[defaultcs,defaultcs],[defaultcs,defaultcs]] # not used

    matdata = [[material1,density1,Td1,traptype1,trapener1,trapcapn1,trapcapp1],\
            [material2,density2,Td2,traptype2,trapener2,trapcapn2,trapcapp2],\
            [material3,density3,Td3,traptype3,trapener3,trapcapn3,trapcapp3],\
            [material4,density4,Td4,traptype4,trapener4,trapcapn4,trapcapp4]]

    mutocmconv = 1e-4

    numatom = [0,3,2,0] # 0 and 3 collumns for layers not simulated
    stackmat = [0,1,0,2,1,2,3,2]
    stackcout = [False,True,False,False,False,True,False,False]



    Erad = 1
    testdpar = _testdpar(Orbit,MissionLength)

    stackSPdat                                                                    =
MultilayerNIELCalculator.fetch_stack_material_SPdat(matdata,float(Erad))

# Deckbuild creation

for j in range(Runs):
    if _Name == "testcell":
        Name = "testcell" + "_%d_%d" %(Batch,j)
        directory          =          'C:\\Users\\LT          Walsh\\Documents\\Python
Scripts\\Deckbuilds\\Batch_%d\\'%(Batch)
        Script = open(directory + '%s.in'%(Name), "w")
    if _Name == "Model":
        Name = "Model_Cell"
```

```
        directory = 'C:\\Users\\LT Walsh\\Documents\\Python Scripts\\Deckbuilds\\Model_Cell\\'
        Script = open(directory + '%s.in'%(Name), "w")
    if _Name == "Check":
        Name = "Check_Cell"
        directory = 'C:\\Users\\LT Walsh\\Documents\\Python Scripts\\Deckbuilds\\Check_Cell\\'
        Script = open(directory + '%s.in'%(Name), "w")
    string = "go atlas simflags = \"-P %d\"\n\n" % (Cores)
    Script.write(string)

    # Radiation Model

    if Radiation == 1:
        stackthick = [Thickness[j,0],Thickness[j,1]+Thickness[j,2],Thickness[j,3],TJ_top_Thickness+TJ_bot_Thickness,Thickness[j,4],Thickness[j,5]+Thickness[j,6],Thickness[j,7],Thickness[j,8]]
        stackdata = []
        x0 = 0
        for i, thick in enumerate(stackthick):
            stackdata.append([stackmat[i],mutocmconv*thick,x0,stackcout[i]])
            x0 = x0 + mutocmconv*thick
        output_data = MultilayerNIELCalculator.calc_multilayer_profiles(stackdata,matdata,stackSPdat,Erad,testdpar)

MultilayerNIELCalculator.gen_lib_files(stackdata,matdata,output_data,j,Batch,directory)
    # Setting up the mesh

    string = "mesh\n\n"
    Script.write(string)
    string = "x.mesh location   = %f spacing  = %f\n" % (0,Width/3)
    Script.write(string)
    string = "x.mesh location   = %f spacing  = %f\n\n" % (Width,Width/3)
    Script.write(string)

    depth = 0
    for i in range(len(Layer)):
        if i == 0:
            string = "y.mesh location   = %f spacing = %f\n" %(depth,Thickness[j,i]/light)
            Script.write(string)
            string = "y.mesh location = %f spacing = %f\n" %(depth+.8*Thickness[j,i],Thickness[j,i]/medium)
            Script.write(string)
```

```
        string = "y.mesh location = %f spacing = %f\n"
%(depth+Thickness[j,i],Thickness[j,i]/heavy)
        Script.write(string)
    elif i == 8:
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.2*Thickness[j,i],Thickness[j,i]/medium)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n\n"
%(depth+Thickness[j,i],Thickness[j,i]/light)
        Script.write(string)
    elif i == 4:
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.2*TJ_top_Thickness,TJ_top_Thickness/medium)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.5*TJ_top_Thickness,TJ_top_Thickness/light)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.8*TJ_top_Thickness,TJ_top_Thickness/medium)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+TJ_top_Thickness,TJ_top_Thickness/heavy)
        Script.write(string)
        depth = depth + TJ_top_Thickness
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.2*TJ_bot_Thickness,TJ_bot_Thickness/medium)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.5*TJ_bot_Thickness,TJ_bot_Thickness/light)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.8*TJ_bot_Thickness,TJ_bot_Thickness/medium)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+TJ_bot_Thickness,TJ_bot_Thickness/heavy)
        Script.write(string)
        depth = depth + TJ_bot_Thickness
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.2*Thickness[j,i],Thickness[j,i]/medium)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.5*Thickness[j,i],Thickness[j,i]/light)
        Script.write(string)
        string = "y.mesh location = %f spacing = %f\n"
%(depth+.8*Thickness[j,i],Thickness[j,i]/medium)
```

```
            Script.write(string)
            string = "y.mesh location = %f spacing = %f\n"
%(depth+Thickness[j,i],Thickness[j,i]/heavy)
            Script.write(string)
        else:
            string = "y.mesh location = %f spacing = %f\n"
%(depth+.2*Thickness[j,i],Thickness[j,i]/medium)
            Script.write(string)
            string = "y.mesh location = %f spacing = %f\n"
%(depth+.5*Thickness[j,i],Thickness[j,i]/light)
            Script.write(string)
            string = "y.mesh location = %f spacing = %f\n"
%(depth+.8*Thickness[j,i],Thickness[j,i]/medium)
            Script.write(string)
            string = "y.mesh location = %f spacing = %f\n"
%(depth+Thickness[j,i],Thickness[j,i]/heavy)
            Script.write(string)
        depth = depth + Thickness[j,i]

    # Setting up region statements

    depth = 0
    string = "region num = 1     material = InAlGaP x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f x.comp = 0.371 y.comp = 0.159\n" %(Width, depth, depth+Thickness[j,0])
    Script.write(string)
    depth = depth+Thickness[j,0]
    string = "region num = 2     material = InGaP    x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f x.comp = 0.49\n" %(Width, depth, depth+Thickness[j,1])
    Script.write(string)
    depth = depth+Thickness[j,1]
    string = "region num = 3     material = InGaP    x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f x.comp = 0.49\n" %(Width, depth, depth+Thickness[j,2])
    Script.write(string)
    depth = depth+Thickness[j,2]
    string = "region num = 4     material = InAlGaP x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f x.comp = 0.371 y.comp = 0.159\n" %(Width, depth, depth+Thickness[j,3])
    Script.write(string)
    depth = depth+Thickness[j,3]
    string = "region num = 5     material = GaAs     x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f\n" %(Width, depth, depth + TJ_top_Thickness)
    Script.write(string)
    depth = depth + TJ_top_Thickness
    string = "region num = 6     material = GaAs     x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f\n" %(Width, depth, depth + TJ_bot_Thickness)
    Script.write(string)
```

```
    depth = depth + TJ_bot_Thickness
    string = "region num = 7     material = InGaP    x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f x.comp = 0.49\n" %(Width, depth, depth+Thickness[j,4])
    Script.write(string)
    depth = depth+Thickness[j,4]
    string = "region num = 8     material = GaAs     x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f\n" %(Width, depth, depth+Thickness[j,5])
    Script.write(string)
    depth = depth+Thickness[j,5]
    string = "region num = 9     material = GaAs     x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f\n" %(Width, depth, depth+Thickness[j,6])
    Script.write(string)
    depth = depth+Thickness[j,6]
    string = "region num = 10    material = AlGaAs  x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f x.comp = 0.7\n" %(Width, depth, depth+Thickness[j,7])
    Script.write(string)
    depth = depth+Thickness[j,7]
    string = "region num = 11    material = GaAs     x.min = 0.0 x.max = %.1f y.min = %f
y.max = %f\n" %(Width, depth, depth+Thickness[j,8])
    Script.write(string)

    # Setting up electrodes

    depth1 = Thickness[j,0]+Thickness[j,1]+Thickness[j,2]+Thickness[j,3]
    depth2 = depth1 + TJ_top_Thickness + (TJ_bot_Thickness*.1)
    depth3 = depth2 + (TJ_bot_Thickness*.8)
    depth4 = depth3 + (TJ_bot_Thickness*.1)
    string = "\nelectrode name = anode top\n"
    Script.write(string)
    string = "electrode name = cathode bottom\n"
    Script.write(string)
    string = "electrode name = TJ_Top    material = GaAs x.min = 0 x.max = %.1f y.min =
%f y.max = %f\n" %(Width, depth1, depth2)
    Script.write(string)
    string = "electrode name = TJ_Bottom material = GaAs x.min = 0 x.max = %.1f y.min
= %f y.max = %f\n\n" %(Width, depth3, depth4)
    Script.write(string)

    # Setting up doping profile

    string = "doping p.type uniform concentration = %s region = 1\n" %(Doping[j,0])
    Script.write(string)
    string = "doping p.type uniform concentration = %s region = 2\n" %(Doping[j,1])
    Script.write(string)
    string = "doping n.type uniform concentration = %s region = 3\n" %(Doping[j,2])
```

```
    Script.write(string)
    string = "doping n.type uniform concentration = %s region = 4\n" %(Doping[j,3])
    Script.write(string)
    string = "doping n.type uniform concentration = %s region = 5\n" %(TJ_top_Doping)
    Script.write(string)
    string = "doping p.type uniform concentration = %s region = 6\n" %(TJ_bot_Doping)
    Script.write(string)
    string = "doping p.type uniform concentration = %s region = 7\n" %(Doping[j,4])
    Script.write(string)
    string = "doping p.type uniform concentration = %s region = 8\n" %(Doping[j,5])
    Script.write(string)
    string = "doping n.type uniform concentration = %s region = 9\n" %(Doping[j,6])
    Script.write(string)
    string = "doping n.type uniform concentration = %s region = 10\n" %(Doping[j,7])
    Script.write(string)
    string = "doping n.type uniform concentration = %s region = 11\n\n" %(Doping[j,8])
    Script.write(string)

    # Material Statements

    string = "material mun = %f          mup = %f       region              =              1\n"
%(Mobilityn[j,0],Mobilityp[j,0])
    Script.write(string)
    string = "material mun = %f          mup = %f       region              =              2\n"
%(Mobilityn[j,1],Mobilityp[j,1])
    Script.write(string)
    string = "material mun = %f          mup = %f       region              =              3\n"
%(Mobilityn[j,2],Mobilityp[j,2])
    Script.write(string)
    string = "material mun = %f          mup = %f       region              =              4\n"
%(Mobilityn[j,3],Mobilityp[j,3])
    Script.write(string)
    string = "material mun = %f          mup = %f       region              =              7\n"
%(Mobilityn[j,4],Mobilityp[j,4])
    Script.write(string)
    string = "material mun = %f          mup = %f       region            =            10\n\n"
%(Mobilityn[j,7],Mobilityp[j,7])
    Script.write(string)

    string = "material mat = InAlGaP  index.file = AlGaInP.nk\n"
    Script.write(string)
    string = "material mat = InGaP    index.file = InGaP_ex.nk\n"
    Script.write(string)
    string = "material mat = AlGaAs   index.file = AlGaAs.nk\n\n"
    Script.write(string)
```

```
   string = "material material=InGaP   affinity = 4.08\n"
   Script.write(string)
   string = "material material=AlGaAs  affinity = 3.54\n\n"
   Script.write(string)
   string = "contact name = TJ_Top resistance    = 1e18\n"
   Script.write(string)
   string = "contact name = TJ_Bottom resistance = 1e18\n"
   Script.write(string)
   string = "contact name = anode resistance     = 2e9\n\n"
   Script.write(string)

   # Radiation model

   if Radiation == 1:
      for i in range(len(stackmat)):
         if stackcout[i]:
            matinput, atomoutput, stoichoutput =
MultilayerNIELCalculator.matinterp(matdata[stackdata[i][0]][0])
            for k in range(len(matdata[stackmat[i]][3])):
               if matdata[stackmat[i]][3][k][0] == 'A':
                  string = "trap acceptor e.level = %f degen = 1 sign = %e sigp =%e f.density
=                                                    %d_%d_%d_%s_Vprofile.lib\n"
%(matdata[stackmat[i]][4][k][0],matdata[stackmat[i]][5][k][0],matdata[stackmat[i]][6][k]
[0],Batch,j,i+1,atomoutput[k])
               elif matdata[stackmat[i]][3][k][0] == 'D':
                  string = "trap donor    e.level = %f degen = 1 sign = %e sigp =%e f.density
=                                                    %d_%d_%d_%s_Vprofile.lib\n"
%(matdata[stackmat[i]][4][k][0],matdata[stackmat[i]][5][k][0],matdata[stackmat[i]][6][k]
[0],Batch,j,i+1,atomoutput[k])
               Script.write(string)
               if matdata[stackmat[i]][3][k][1] == 'A':
                  string = "trap acceptor e.level = %f degen = 1 sign = %e sigp =%e f.density
=                                                     %d_%d_%d_%s_Iprofile.lib\n"
%(matdata[stackmat[i]][4][k][1],matdata[stackmat[i]][5][k][1],matdata[stackmat[i]][6][k]
[1],Batch,j,i+1,atomoutput[k])
               elif matdata[stackmat[i]][3][k][1] == 'D':
                  string = "trap donor    e.level = %f degen = 1 sign = %e sigp =%e f.density
=                                                    %d_%d_%d_%s_Iprofile.lib\n"
%(matdata[stackmat[i]][4][k][1],matdata[stackmat[i]][5][k][1],matdata[stackmat[i]][6][k]
[1],Batch,j,i+1,atomoutput[k])
               Script.write(string)
      string = "\n"
      Script.write(string)

   # Model Statements
```

```
    for i in range(len(Layer)+2):
        if i in [4,5,7,8,10]:
            string = "models region = %d\t analytic\n" %(i+1)
            Script.write(string)
    string = "\nmodels print srh fermi optr auger bgn temp = %.1f\n\n" %(Temp)
    Script.write(string)

    # Method Statements

    string = "method newton itlimit = 50\n\n"
    Script.write(string)

    # Setting up the beam

    string = "beam num = 1 x.origin = %.2f y.origin = -1  angle = 90 am0 wavel.start = 0.28
wavel.end = 3.5 wavel.num = 500 reflect = 1\n\n" %(Width/2)
    Script.write(string)

    # Setting up the outputs

    string = "output con.band val.band opt.intens\n\n"
    Script.write(string)

    # Solving & Outputs

    string = "solve init\n"
    Script.write(string)
    string = "solve b1 = %.2f\n" %(beam1)
    Script.write(string)
    string = "solve b1 = %.2f\n" %(beam2)
    Script.write(string)
    string = "solve b1 = %.2f\n\n" %(1-Reflectivity)
    Script.write(string)
    string = "log outfile = %s.log\n" %(Name)
    Script.write(string)
    string = "solve name = anode vanode = %.3f vfinal = %.3f vstep = %.3f\n"
%(0,V1,Vstep1)
    Script.write(string)
    string = "solve name = anode vanode = %.3f vfinal = %.3f vstep = %.3f\n"
%(V1,V2,Vstep2)
    Script.write(string)
    string = "solve name = anode vanode = %.3f vfinal = %.3f vstep = %.3f\n"
%(V2,Voc,Vstep3)
    Script.write(string)
```

```
string = "log off\n\n"
Script.write(string)
if View == 1:
   string = "save outfile = %s.str\n\n" %(Name)
   Script.write(string)
   string = "tonyplot %s.log -set Display.set\n\n" %(Name)
   Script.write(string)
   string = "tonyplot %s.str\n\n" %(Name)
   Script.write(string)
string = "quit"
Script.write(string)

Script.close()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. ALGAAS OPTICAL FILE GENERATOR

```
# -*- coding: utf-8 -*-
"""
AlGaAs MDF calculation
From Adachi et al

@author: jswalsh
"""

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import sys




# Parameter definitions

E0        = 2.42            # E0 (Bandgap) transition energy
E0_Delta0 = 2.73              # E0 + delta (VB split off) energy
A         = 23.2            # Direct transition strength (matrix element)

E1        = 3.43            # E1 (Bandgap) transition energy
EgID      = 2.03              # Indirect transition mechanism
B1        = 5.41            # Direct transition strength (matrix element)
B11       = 9.55             # Direct transition strength (matrix element)
Gamma1  = .12                 # E1, E1 + delta broadening energy

E2        = 4.7             # E2 (Bandgap) transition energy
C         = 1.76            # nondimentional strength parameter
gamma     = .103              # nondimentional broadening parameter

D         = 8.1             # Indirect-transition strength parameter
Einf      = -.3             # nondispersive term
Eq        = 0               # phonon energy


Name      = "AlGaAs"
Estart    = 1               # starting energy in eV
```

```
Estop    = 7              # ending energy in eV
Points   = 1000             # number of points to use in file

Energy   = np.linspace(Estart,Estop,Points)
Lamda    = 1.24/Energy

PlotAll  = 0              # Set to 1 to plot subsections of Epsilon


# Epsilon calculations

# Epsilon 2_0

Chi_0 = Energy/E0
Chi_so = Energy/E0_Delta0

Epsilon_2_0a = np.zeros(Points)
Epsilon_2_0b = np.zeros(Points)
for i in range(Points):
   if Chi_0[i]>=1:
      Epsilon_2_0a[i] = np.power(Energy[i]-E0,.5)
   else:
      Epsilon_2_0a[i] = 0
   if Chi_so[i]>=1:
      Epsilon_2_0b[i] = np.power(Energy[i]-E0_Delta0,.5)
   else:
      Epsilon_2_0b[i] = 0
Epsilon_2_0 = (A/np.power(Energy,2))*(Epsilon_2_0a + .5*Epsilon_2_0b)

# Epsilon 1_0

f_Chi_0 = np.zeros(Points)
f_Chi_so = np.zeros(Points)

for i in range(Points):
   if 1>=Chi_0[i]:
      f_Chi_0[i]                    =                    (2-np.power(1+Chi_0[i],.5)-np.power(1-
Chi_0[i],.5))/np.power(Chi_0[i],2)
   else:
      f_Chi_0[i] = (2-np.power(1+Chi_0[i],.5))/np.power(Chi_0[i],2)
   if 1>=Chi_so[i]:
      f_Chi_so[i]                   =                   (2-np.power(1+Chi_so[i],.5)-np.power(1-
Chi_so[i],.5))/np.power(Chi_so[i],2)
   else:
      f_Chi_so[i] = (2-np.power(1+Chi_so[i],.5))/np.power(Chi_so[i],2)
```

```
Epsilon_1_0 = (A/np.power(E0,1.5))*(f_Chi_0 + .5*np.power(E0/E0_Delta0,1.5)*f_Chi_so)

# Epsilon 2_1

Chi_1 = Energy/E1
Epsilon_2_1 = np.zeros(Points)
for i in range(Points):
   if Energy[i]<E1:
      Epsilon_2_1[i] = 0
   else:
      Epsilon_2_1[i] = np.pi*B1/np.power(Chi_1[i],2)


# Epsilon 1_1

Chi_1_1 = (Energy + 1j*Gamma1)/E1
Epsilon_1_1 = (-B1/np.power(Chi_1_1,2))*np.log(1-np.power(Chi_1_1,2))

# Epsilon 2_2

Chi_2 = Energy/E2
Epsilon_2_2 = C*Chi_2*gamma/(np.power(1-np.power(Chi_2,2),2)+np.power(Chi_2*gamma,2))

# Epsilon 1_2

Epsilon_1_2 = C*(1-np.power(Chi_2,2))/(np.power(1-np.power(Chi_2,2),2)+np.power(Chi_2*gamma,2))

# Epsilon 2 indirect

Chi_c = Energy/E1
Chi_g = Energy/(EgID - Eq)

Epsilon_2_ind = np.zeros(Points)
for i in range(Points):
   if 1>=Chi_g[i] or 1>=Chi_c[i] or Energy[i]<2.41:
      Epsilon_2_ind[i] = 0
   else:
      Epsilon_2_ind[i] = (D/np.power(Energy[i],2))*np.power(Energy[i]-EgID-Eq,2)

# Epsilon calculations
```

```
Epsilon_r   = Epsilon_1_0 + Epsilon_1_1 + Epsilon_1_2 + Einf
Epsilon_i   = Epsilon_2_0 + Epsilon_2_1 + Epsilon_2_2 + Epsilon_2_ind
Epsilon     = Epsilon_r + 1j*Epsilon_i
Epsilon_r   = np.real(Epsilon)
Epsilon_i   = np.imag(Epsilon)
for i in range(Points):
   if Energy[i]<=2.41:
      Epsilon_i[i] = 0
Epsilon     = Epsilon_r + 1j*Epsilon_i

# n and k calculations

e1 = np.real(Epsilon)
e2 = np.imag(Epsilon)
na = np.power(e1,2)
nb = np.power(e2,2)
nc = np.power(na + nb, .5)
nd = (nc + e1)/2
n  = np.power(nd,.5)

kd = (nc - e1)/2
k  = np.power(kd,.5)


#Plot the figure. Based on spyder settings, plt pops out a separate
#plot window.  The baseline plot settings will plot the plot inline in the
#console.  You can change these settings in the Tools > Preferences menu
plt.figure(1)
plt.clf()
#plt.plot(Energy,np.real(Epsilon),'b-',label="Epsilon Real")
plt.xlim(0,9)
plt.semilogy(Energy,np.imag(Epsilon),'g-')
plt.xlabel("Energy (eV)")
plt.ylabel("Epsilon 2")
plt.show()

plt.figure(10)
plt.clf()
#plt.plot(Energy,np.real(Epsilon),'b-',label="Epsilon Real")
plt.xlim(0,8)
plt.plot(Energy,np.real(Epsilon),'b-')
plt.xlabel("Energy (eV)")
plt.ylabel("Epsilon 1")
plt.show()
```

```
if PlotAll:
    plt.figure(3)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,Epsilon_2_0,'b-',label="Epsilon 2(0)")
    plt.xlabel("Energy (eV)")
    plt.ylabel("Epsilon 2(0)")
    plt.show()

    plt.figure(4)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,Epsilon_1_0,'b-',label="Epsilon 1(0)")
    plt.xlabel("Energy (eV)")
    plt.ylabel("Epsilon 1(0)")
    plt.show()

    plt.figure(5)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,Epsilon_2_1,'b-',label="Epsilon 2(1)")
    plt.xlabel("Energy (eV)")
    plt.ylabel("Epsilon 2(1)")
    plt.show()

    plt.figure(6)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,np.real(Epsilon_1_1),'b-',label="Epsilon 1(1) (real)")
    plt.plot(Energy,np.imag(Epsilon_1_1),'b-',label="Epsilon 1(1) (imaginary)")
    plt.xlabel("Energy (eV)")
    plt.ylabel("Epsilon 1(1)")
    plt.legend()
    plt.show()

    plt.figure(7)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,Epsilon_2_2,'b-',label="Epsilon 2(2)")
    plt.xlabel("Energy (eV)")
    plt.ylabel("Epsilon 2(2)")
    plt.show()

    plt.figure(8)
    plt.clf()
```

```
    plt.title(Name)
    plt.plot(Energy,Epsilon_1_2,'b-',label="Epsilon 1(2)")
    plt.xlabel("Energy (eV)")
    plt.ylabel("Epsilon 1(2)")
    plt.show()

    plt.figure(9)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,Epsilon_2_ind,'b-',label="Epsilon 2(Indirect)")
    plt.xlabel("Energy (eV)")
    plt.ylabel("Epsilon 2(Indirect)")
    plt.show()


plt.figure(2)
plt.clf()
plt.plot(Energy,n,'b-',label="n")
plt.plot(Energy,k,'g-',label="k")
plt.xlabel("Energy (eV)")
plt.ylabel("n,k")
plt.legend()
plt.show()

# Create .nk file

Properties = open(Name + ".nk", "w")
string = "%d\n" %(Points-1)
Properties.write(string)
for i in range(Points):
    string = "%f\t%f\t%f \n" % (Lamda[i],n[i],k[i])
    Properties.write(string)
Properties.close()
```

APPENDIX E. INALGAP OPTICAL FILE GENERATOR

```
# -*- coding: utf-8 -*-
"""
AlGaInP MDF calculation
From Adachi et al

@author: jswalsh
"""

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import sys




# Parameter definitions

E0       = 2.38            # E0 (Bandgap)transition energy
E0_Delta0 = 2.45           # E0 + delta (VB split off) energy
A        = 11              # Direct transition strength (matrix element)
Gamma0    = 0.03           # E0, E+del0 broadening energy
E1       = 3.60            # E1 (Bandgap) transition energy
B1       = 4.4             # Direct transition strength (matrix element)
B1x      = 1.8             # 2D exciton strength parameter
Gamma1    = .26            # E1, E1 + delta broadening energy
E2       = 4.85            # E2 (Bandgap)transition energy
C2       = 1.7             # nondimentional strength parameter
Gamma2    = .82            # nondimentional broadening parameter
Gamma3    = .7             # E2 + delta broadening energy
E2_Delta2 = 5.02           # E2 + delta (VB split off) energy
C2_Delta2 = .5             # nondimentional strength parameter
Einf     = .4              # nondispersive term

Name     = "AlGaInP"
Estart    = 1              # starting energy in eV
Estop     = 5.5            # ending energy in eV
Points    = 1000           # number of points to use in file
```

```
Energy    = np.linspace(Estart,Estop,Points)
Lamda     = 1.24/Energy

PlotAll   = 0              # Set to 1 to plot subsections of Epsilon



# Epsilon and Exciton calculations function


def Chi(Energy, Gamma, Base_Energy_Level):
   Chi = (Energy + 1j*Gamma)/(Base_Energy_Level)
   return Chi

def f_Chi(Chi):
   f_Chi = (1/np.power(Chi,2))*(2-np.power(1+Chi,.5)-np.power(1-Chi,.5))
   return f_Chi


def f_Eps_1(B1,Chi_1):
   Eps_1 = -B1/np.power(Chi_1,2)*np.log(1-np.power(Chi_1,2))
   return Eps_1

def f_Exciton_1(B1x,E1,Energy,Gamma):
   Exciton_1 = B1x/(E1 - Energy -1j*Gamma)
   return Exciton_1

def f_Eps_2(C,Energy,Eg,Gamma):
   Eps_2 = C*np.power(Eg,2)/(np.power(Eg,2)-np.power(Energy,2)-1j*Energy*Gamma)
   return Eps_2



# Epsilon calculations

Eps_0 = A / np.power(E0,1.5) * (f_Chi(Chi(Energy,Gamma0, E0)) + .5*
np.power(E0/E0_Delta0,1.5)*f_Chi(Chi(Energy,Gamma0,E0_Delta0)))
Eps_1 = f_Eps_1(B1,Chi(Energy, Gamma1, E1))
Exciton_1 = f_Exciton_1(B1x,E1,Energy,Gamma1)
Eps_2 = f_Eps_2(C2,Energy,E2,Gamma2)
Eps_2_Split = f_Eps_2(C2_Delta2,Energy,E2_Delta2,Gamma3)
```

```
Epsilon = Eps_0 + Eps_1 + Exciton_1 + Eps_2 + Einf


# n and k calculations

e1 = np.real(Epsilon)
e2 = np.imag(Epsilon)
na = np.power(e1,2)
nb = np.power(e2,2)
nc = np.power(na + nb, .5)
nd = (nc + e1)/2
n  = np.power(nd,.5)

kd = (nc - e1)/2
k  = np.power(kd,.5)

#Plot the figure. Based on spyder settings, plt pops out a separate
#plot window.  The baseline plot settings will plot the plot inline in the
#console.  You can change these settings in the Tools > Preferences menu


plt.figure(1)
plt.clf()
#plt.plot(Energy,np.real(Epsilon),'b-',label="Epsilon Real")
plt.xlim(0,6.5)
plt.plot(Energy,np.imag(Epsilon),'b-')
plt.xlabel("Energy (eV)")
plt.ylabel("Epsilon 2")
plt.show()

#plt.figure(10)
#plt.clf()
##plt.plot(Energy,np.real(Epsilon),'b-',label="Epsilon Real")
#plt.xlim(0,6)
#plt.plot(Energy,np.real(Epsilon),'b-')
#plt.xlabel("Energy (eV)")
#plt.ylabel("Epsilon 1")
#plt.show()

if PlotAll:
    plt.figure(3)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,np.real(Eps_0),'b-',label="Epsilon 0 Real")
```

```
    plt.plot(Energy,np.imag(Eps_0),'g-',label="Epsilon 0 Imaginary")
    plt.xlabel("Photon Energy (eV)")
    plt.ylabel("Epsilon 0")
    plt.legend()
    plt.show()

    plt.figure(4)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,np.real(Eps_1),'b-',label="Eps 1 Real")
    plt.plot(Energy,np.imag(Eps_1),'g-',label="Epsilon 1 Imaginary")
    plt.xlabel("Photon Energy (eV)")
    plt.ylabel("Epsilon 1")
    plt.legend()
    plt.show()

    plt.figure(5)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,np.real(Exciton_1),'b-',label="Exciton 1 Real")
    plt.plot(Energy,np.imag(Exciton_1),'g-',label="Exciton 1 Imaginary")
    plt.xlabel("Photon Energy (eV)")
    plt.ylabel("Exciton 1")
    plt.legend()
    plt.show()

    plt.figure(6)
    plt.clf()
    plt.title(Name)
    plt.plot(Energy,np.real(Eps_2),'b-',label="Eps 2 Real")
    plt.plot(Energy,np.imag(Eps_2),'g-',label="Epsilon 2 Imaginary")
    plt.xlabel("Photon Energy (eV)")
    plt.ylabel("Epsilon 2")
    plt.legend()
    plt.show()



#plt.figure(2)
#plt.clf()
#plt.title(Name)
#plt.plot(Energy,n,'b-',label="n")
#plt.plot(Energy,k,'g-',label="k")
#plt.xlabel("Photon Energy (eV)")
#plt.ylabel("n,k")
```

```
#plt.legend()
#plt.show()

# Create .nk file

Properties = open(Name + ".nk", "w")
string = "%d\n" %(Points-1)
Properties.write(string)
for i in range(Points):
    string = "%f\t%f\t%f \n" % (Lamda[i],n[i],k[i])
    Properties.write(string)
Properties.close()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. MOBILITY CALCULATOR

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 30 09:54:27 2018

Electron and Hole mobility

@author: LT Walsh
"""

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import sys

# Function definitions

def f(mu1, mu2, T, N, Ncrit, alpha, beta, gamma, delta):
    f1 = mu1*np.power(T/300,alpha)
    f2 = mu2*np.power(T/300,beta)
    f3 = np.power(T/300,gamma)
    f4 = np.power(N/Ncrit,delta)
    f5 = 1 + f3*f4
    f6 = (f2-f1)/f5
    f  = f1 + f6
    return f

def mu(f, mdom, mnum, eldom, elnum, ehdom, ehnum):
    mu1 = (1/ehnum) - (1/elnum)
    mu2 = (1/ehdom) - (1/eldom)
    mu3 = f*np.power(mnum,1.5)*mu1
    mu4 = np.power(mdom,1.5)*mu2
    mu  = mu3/mu4
    return mu

def _Rd(mci, mcd, Egi, Egd, T):
    k = 8.617/np.power(10,5) #Boltzman Constant
    R1 = np.power(mci/mcd,1.5)
    R2 = np.exp((Egd-Egi)/(k*T))
    Rd = 1/(1+(R1*R2))
    return Rd

def mass(m1, m2, C):
```

```
    ma = C/m1
    mb = (1-C)/m2
    mass  = 1/(ma +mb)
    return mass

def eps(e1, e2, C):
    eps1 = (e1-1)/(e1+2)
    eps2 = (e2-1)/(e2+2)
    C2   = 1-C
    eps3 = C*eps1
    eps4 = C2*eps2
    eps5 = 1 + (2 * (eps3 + eps4))
    eps6 = 1 - eps3 - eps4
    eps  = eps5/eps6
    return eps

def _Eg_triplet(C,Eg1,Eg2,Bow_m,Bow_b):
    Bow = (Bow_m*C) + Bow_b
    Eg = (C*Eg1) + ((1-C)*Eg2) + (np.power(C,2)*Bow) - (C*Bow)
    return Eg


def mobility_triplet(material_1,material_2,triplet_name,C,T,N,quad_set=0):

    # Material 1 parameters
    if material_1 == 'AlP':

        fh = open("mobility_binaries.txt", "r")
        string = " "
        counter = 0
        while string != material_1:
            string = fh.readline()
            string = string.strip()
            counter += 1
            if counter>200:
                return 0,0

        m_star_n1d = float(fh.readline())
        m_star_n1i = float(fh.readline())
        m_star_p1  = float(fh.readline())
        eps_h1     = float(fh.readline())
        eps_l1     = float(fh.readline())
        Egd_1      = float(fh.readline())
        Egi_1      = float(fh.readline())
        fh.close()
```

```
else:

    fh = open("mobility_binaries.txt", "r")
    string = " "
    counter = 0
    while string != material_1:
        string = fh.readline()
        string = string.strip()
        counter += 1
        if counter>200:
            return 0,0

    mu1_p1     = float(fh.readline())
    mu2_p1     = float(fh.readline())
    Ncrit_p1b  = float(fh.readline())
    Ncrit_p1p  = float(fh.readline())
    Ncrit_p1   = Ncrit_p1b*np.power(10,Ncrit_p1p)
    alpha_p1   = float(fh.readline())
    beta_p1    = float(fh.readline())
    gamma_p1   = float(fh.readline())
    delta_p1   = float(fh.readline())
    mu1_n1     = float(fh.readline())
    mu2_n1     = float(fh.readline())
    Ncrit_n1b  = float(fh.readline())
    Ncrit_n1p  = float(fh.readline())
    Ncrit_n1   = Ncrit_n1b*np.power(10,Ncrit_n1p)
    alpha_n1   = float(fh.readline())
    beta_n1    = float(fh.readline())
    gamma_n1   = float(fh.readline())
    delta_n1   = float(fh.readline())
    m_star_n1d = float(fh.readline())
    m_star_n1i = float(fh.readline())
    m_star_p1  = float(fh.readline())
    eps_h1     = float(fh.readline())
    eps_l1     = float(fh.readline())
    Egd_1      = float(fh.readline())
    Egi_1      = float(fh.readline())
    fh.close()

# Material 2 parameters

if material_2 == 'AlP':

    fh = open("mobility_binaries.txt", "r")
```

```
    string = " "
    counter = 0
    while string != material_2:
        string = fh.readline()
        string = string.strip()
        counter += 1
        if counter>200:
            return 0,0

    m_star_n2d = float(fh.readline())
    m_star_n2i = float(fh.readline())
    m_star_p2  = float(fh.readline())
    eps_h2     = float(fh.readline())
    eps_l2     = float(fh.readline())
    Egd_2      = float(fh.readline())
    Egi_2      = float(fh.readline())
    fh.close()

else:

    fh = open("mobility_binaries.txt", "r")
    string = " "
    counter = 0
    while string != material_2:
        string = fh.readline()
        string = string.strip()
        counter += 1
        if counter>200:
            return 0,0

    mu1_p2     = float(fh.readline())
    mu2_p2     = float(fh.readline())
    Ncrit_p2b  = float(fh.readline())
    Ncrit_p2p  = float(fh.readline())
    Ncrit_p2   = Ncrit_p2b*np.power(10,Ncrit_p2p)
    alpha_p2   = float(fh.readline())
    beta_p2    = float(fh.readline())
    gamma_p2   = float(fh.readline())
    delta_p2   = float(fh.readline())
    mu1_n2     = float(fh.readline())
    mu2_n2     = float(fh.readline())
    Ncrit_n2b  = float(fh.readline())
    Ncrit_n2p  = float(fh.readline())
    Ncrit_n2   = Ncrit_n2b*np.power(10,Ncrit_n2p)
    alpha_n2   = float(fh.readline())
```

```
    beta_n2    = float(fh.readline())
    gamma_n2   = float(fh.readline())
    delta_n2   = float(fh.readline())
    m_star_n2d = float(fh.readline())
    m_star_n2i = float(fh.readline())
    m_star_p2  = float(fh.readline())
    eps_h2     = float(fh.readline())
    eps_l2     = float(fh.readline())
    Egd_2      = float(fh.readline())
    Egi_2      = float(fh.readline())
    fh.close()

# Triplet bowing parameters

fh = open("mobility_triplet.txt", "r")
string = " "
counter = 0
while string != triplet_name:
    string = fh.readline()
    string = string.strip()
    counter += 1
    if counter>200:
        return 0,0
Bow_md     = float(fh.readline())
Bow_bd     = float(fh.readline())
Bow_mi     = float(fh.readline())
Bow_bi     = float(fh.readline())

# hole mobility

m_star_p = mass(m_star_p1,m_star_p2,C)
if material_2 == 'AlP':
    fp2 = 450
else:
    fp2 = f(mu1_p2, mu2_p2, T, N, Ncrit_p2, alpha_p2, beta_p2, gamma_p2, delta_p2)
eps_h = eps(eps_h1,eps_h2,C)
eps_l = eps(eps_l1,eps_l2,C)

Mu_p = mu(fp2, m_star_p, m_star_p2, eps_l, eps_l2, eps_h, eps_h2)

# electron mobility

Egi = _Eg_triplet(C,Egi_1,Egi_2,Bow_mi,Bow_bi)
Egd = _Eg_triplet(C,Egd_1,Egd_2,Bow_md,Bow_bd)
if material_1 == 'AlP':
```

```
        fn1 = 60
    else:
        fn1 = f(mu1_n1, mu2_n1, T, N, Ncrit_n1, alpha_n1, beta_n1, gamma_n1, delta_n1)
    if material_2 == 'AlP':
        fn2 = 60
    else:
        fn2 = f(mu1_n2, mu2_n2, T, N, Ncrit_n2, alpha_n2, beta_n2, gamma_n2, delta_n2)
    m_star_nd = mass(m_star_n1d,m_star_n2d,C)
    m_star_ni = mass(m_star_n1i,m_star_n2i,C)

    Mu_nd = mu(fn2, m_star_nd, m_star_n2d, eps_l, eps_l2, eps_h, eps_h2)
    Mu_ni = mu(fn1, m_star_ni, m_star_n1i, eps_l, eps_l1, eps_h, eps_h1)
    Rd = _Rd(m_star_ni, m_star_nd, Egi, Egd, T)

    Mu_n = (Mu_nd*Rd)+(Mu_ni*(1-Rd))

    if quad_set != 0:
        return Mu_p,Mu_n,m_star_p,m_star_nd,m_star_ni,eps_h,eps_l,Egd,Egi
    else:
        return Mu_n,Mu_p


# For (Al.7Ga.3).5In.5P:
# mat_1 = AlP
# mat_2 = GaP
# mat_3 = InP
# triplet_name_1 = AlInP
# triplet_name_2 = GaInP
# C_inner = the concentration of Al in the (AlGa) compound, .7
# C_outer = the concentration of the (AlGa) compound, .5

def
mobility_quad(mat_1,mat_2,mat_3,triplet_name_1,triplet_name_2,quad_name,C_inner,
C_outer,T,N):

fp1,fn1,m_star_p1,m_star_n1d,m_star_n1i,eps_h1,eps_l1,Egd_1,Egi_1=mobility_triplet(
mat_1,mat_3,triplet_name_1,C_outer,T,N,1)

fp2,fn2,m_star_p2,m_star_n2d,m_star_n2i,eps_h2,eps_l2,Egd_2,Egi_2=mobility_triplet(
mat_2,mat_3,triplet_name_2,C_outer,T,N,1)

    # Quad bowing parameters
    if quad_name == 'AlGaInP':
        fh = open("mobility_quad.txt","r")
        string = " "
```

```
        counter = 0
        while string != quad_name:
            string = fh.readline()
            string = string.strip()
            counter += 1
            if counter>200:
                return 0,0
        Bow_md = float(fh.readline())
        Bow_bd = float(fh.readline())
    else:
        fh = open("mobility_quad.txt", "r")
        string = " "
        counter = 0
        while string != quad_name:
            string = fh.readline()
            string = string.strip()
            counter += 1
            if counter>200:
                return 0,0
        Bow_md     = float(fh.readline())
        Bow_bd     = float(fh.readline())
        Bow_mi     = float(fh.readline())
        Bow_bi     = float(fh.readline())

    # hole mobility

    m_star_p = mass(m_star_p1,m_star_p2,C_inner)
    eps_h = eps(eps_h1,eps_h2,C_inner)
    eps_l = eps(eps_l1,eps_l2,C_inner)

    Mu_p = mu(fp2, m_star_p, m_star_p2, eps_l, eps_l2, eps_h, eps_h2)

    # electron mobility
    if quad_name == 'AlGaInP':
        Egi = _Eg_triplet(.55,Egd_1,Egd_2,Bow_md,Bow_bd) # .55 determined to be
crossover, slope is 0
    else:
        Egi = _Eg_triplet(C_inner,Egi_1,Egi_2,Bow_mi,Bow_bi)
    Egd = _Eg_triplet(C_inner,Egd_1,Egd_2,Bow_md,Bow_bd)
    m_star_nd = mass(m_star_n1d,m_star_n2d,C_inner)
    m_star_ni = mass(m_star_n1i,m_star_n2i,C_inner)

    Mu_nd = mu(fn2, m_star_nd, m_star_n2d, eps_l, eps_l2, eps_h, eps_h2)
    Mu_ni = mu(fn1, m_star_ni, m_star_n1i, eps_l, eps_l1, eps_h, eps_h1)
    Rd = _Rd(m_star_ni, m_star_nd, Egi, Egd, T)
```

```
Mu_n = (Mu_nd*Rd)+(Mu_ni*(1-Rd))

return Mu_n,Mu_p
```

APPENDIX G. MOBILITY CALCULATOR BINARY DATABASE

name
mu1_p
mu2_p
Ncrit_p base
Ncrit_p power
alpha_p
beta_p
gamma_p
delta_p
mu1_n
mu2_n
Ncrit_n base
Ncrit_n power
alpha_n
beta_n
gamma_n
delta_n
electron effective mass direct
electron effective mass indirect
hole effective mass
epsilon high
epsilon low
Bandgap direct (gamma)
Bandgap indirect (x)

GaAs
20
491.5
1.48
17
0
-2.2
-1.14
.38
500
9400
6
16
0
-2.1

-1.182
.394
.067
.85
.53
10.89
13.2
1.519
1.981

AlAs
10
200
3.48
17
0
-2.24
-1.464
.488
10
400
5.46
17
0
-2.1
-3.0
1.0
.15
.19
.80
8.16
12
3.099
2.24

AlP
.22
.793
.7
8.06
9.8
3.63
2.52

GaP

10
147
1
18
0
-1.98
0
.85
10
152
4.4
18
0
-1.6
-.568
.8
.13
1.12
.83
9.11
11.1
2.87
2.35

InP
10
170
4.87
17
0
-2
-1.86
.62
400
5200
3.0
17
0
-2
-1.5275
.47
.0795
.88
.62
9.61

12.5
1.4236
2.273

APPENDIX H. MOBILITY CALCULATOR TERTIARY DATABASE

name
Bowing parameter slope (m value) direct
Bowing parameter constant (b value) direct
Bowing parameter slope (m value) indirect
Bowing parameter constant (b value) indirect

AlGaAs
1.31
-.127
0
.055

AlInP
0
-.48
0
.38

GaInP
0
.65
0
.20

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I. MOBILITY CALCULATOR QUATERNARY DATABASE

name
Bowing parameter slope (m value) direct
Bowing parameter constant (b value) direct
Bowing parameter slope (m value) indirect
Bowing parameter constant (b value) indirect

AlGaInP
0
.18
0

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] P. Michalopoulos, "A novel approach for the development and optimization of state-of-the-art photovoltaic devices using Silvaco," Naval Postgraduate School, Monterey, CA, 2003.

[2] R. Kilway, "Five-junction solar cell optimization using Silvaco ATLAS," Naval Postgraduate School, Monterey, CA, 2017.

[3] S. Pueschel, "Optimization of an advanced multi-junction solar-cell design for space enviroments (AM0) using nearly orthoganal Latin hypercubes," Naval Postgraduate School, Monterey, CA, 2017.

[4] M. R. Lueck, C. L. Andre, A. J. Pitera, M. L. Lee, A. Fitzgerald and S. A. Ringel, "Dual-junction GaInP/GaAs solar cells grown on metamorphic SiGe/Si substrates with high open circuit voltage," *IEEE Electron Device Letters,* vol. 27, no. 3, pp. 142-144, 2006.

[5] J. Lindmayer and C. Wrigley, Fundamentals of semiconductor devices, New York, NY: Van Nostrand Reinhold Company, 1965.

[6] "Energy bands in crystals," what-when-how, [Online]. Available: http://what-when-how.com/electronic-properties-of-materials/energy-bands-in-crystals-fundamentals-of-electron-theory-part-3/. [Accessed 29 May 2018].

[7] C. Honsberg and S. Bowden, "Standard Solar Spectra," PV Education, 2008. [Online]. Available: https://www.pveducation.org/pvcdrom/appendices/standard-solar-spectra

[8] C. Hu and R. White, Solar cells from basic to advanced systems, USA: McGraw-Hill, Inc., 1983.

[9] S. M. Sze and K. Ng, Pysics of semiconductor devices, Hoboken, NJ: Wiley-interscience, 2007.

[10] E. G. Stassinopoulos and J. P. Raymond, "The space radiation environment for electronics," *Proceedings of the IEEE,* vol. 76, no. 11, pp. 1423-1442, 1988.

[11] G. Was, Fundamentals of radiation materials science, New York, NY: Springer, 2007.

[12] S. Adachi, "Optical properties of AlxGa1-xAs alloys," *Physical Review B,* vol. 38, no. 17, pp. 12345-12352, 1988.

[13] S. Ozaki, S. Adachi, M. Sato and K. Ohtsuka, "Ellipsometric and thermoreflectance spectra of (AlxGa1-x)0.5In0.5P alloys," *Journal of Applied Physics*, vol. 79, pp. 439-445, 1996.

[14] J. E. Sutherland and J. R. Hauser, "A Computer Analysis of Heterojunction and Graded Composition Solar Cells," *IEEE Transactions on Electron Devices*, vol. 24, no. 4, pp. 363-372, 1977.

[15] Silvaco, Inc., Atlas User's Manual, Santa Clara, CA, 2017.

[16] Rensselaer Polytechnic Institute, "Room temperature properties of semiconducters: III-V phosphides," [Online]. Available: https://www.ecse.rpi.edu/~schubert/Educational-resources/Materials-Semiconductors-III-V-phosphides.pdf

[17] I. Vurgaftman, J. R. Meyer and L. R. Ram-Mohan, "Band parameters for III-V semiconductors and their alloys," *Journal of Applied Physics*, vol. 89, pp. 5815-5875, 2001.

[18] O. Madelung, U. Rossler and M. Schulz, Eds., Group IV Elements, IV-IV and III-V Compounds. Part a- Lattice Properties, Springer, Berlin, Heidelberg.

[19] A. Saliev, "Electronic properties of aluminum phosphide (AlP)," 2013.

[20] "Semiconductors on NSM," NSM Archives, [Online]. Available: http://www.ioffe.ru/SVA/NSM/Semicond/

[21] M. Porter, private communication, *NIEL calculator*, Monterey, CA, 2018.

[22] National Institute of Standards and Technology, "Stopping power and range tables for electrons," [Online]. Available: https://physics.nist.gov/PhysRefData/Star/Text/ESTAR.html. [Accessed 2018].

[23] M. J. Boschini, P. G. Rancoita and M. Tacconi, "SR-NIEL Calculator: Screened Relativistic (SR) Treatment for tclculating the displacement damage and nuclear stopping powers for electrons, protons, light- and heavy- ions in materials (version 3.9.5)," [Online]. Available: http://www.sr-niel.org/. [Accessed 2018].

[24] D. Pons, P. M. Mooney and J. C. Bourgoin, "Energy dependence of deep level introduction in electron irradiated GaAs," *Journal of Applied Physics*, vol. 51, pp. 2038-2042, 1980.

[25] K. Gartner, "MD simulation of ion implantation damage in AlGaAs: I. Dispacement energies," *Nuclear Instruments and Methods in Physics Research B,* vol. 252, pp. 190-196, 2006.

[26] Y. Okuno, S. Okuda, M. Akiyoshi, T. Oka, M. Harumoto, K. Omura, S. Kawakita, M. Imaizumi, S. R. Messenger, K. H. Lee and M. Yamaguchi, "Radiation degradation prediction for InGaP solar cells by using appropriate estimation method for displacement threshold energy," *Journal of Applied Physics,* vol. 122, pp. 114901-1 - 114901-7, 2017.

[27] P. Schultz and O. Lilienfeld, "Simple intrinsic defects in gallium arsenide," *Modelling Simul. Mater. Sci. Eng.,* vol. 17, pp. 1-35, 2009.

[28] P. Schultz, "First principles defect chemistry for modeling irradiated GaAs and III-V's," Sandia National Laboratories, Albuquerque, NM, 2011.

[29] S. M. Sanchez, *NOLHdesigns spreadsheet.* Available online via http://harvest.nps.edu/. [Accessed 2018]

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California