Modelling of Nanosecond time-scale Transient response of GaN LEDs

Project Outline

Zachary Humphreys

ID: 200951438

02/02/2018

A Thesis Submitted in partial fulfilment of the requirements for the degree of:

Master of Physics

Under the supervision of Dr. Joachim Rose

At the

Department of Physics

UNIVERSITY OF LIVERPOOL

Contents

Listings

List of Figures

List of Tables

List of Algorithms

Declaration

I, Zachary Humphreys, hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

Note: As this is the project outline, certain missing information, citations, and other incomplete sections will be labelled in ***bold*** *font to bring attention to it.*

1 Abstract

In this thesis, the light emission properties of Gallium Nitride blue LEDs are modelled and explained, particularly of note being the unusual characteristics of one being switched on and off on nanosecond timescales.
Switching under these conditions require high voltages in both the forwards and reverse biases and effects that can normally be ignored appear to have much larger effects under these conditions and time scales.
Switching LEDs under these time-scales could be of particular use in the calibration of large photo-detectors which focus on detection of very small **approx#ofphotons** bursts of photons (such as Neutrino detection experiments)**findcitationforsuperK**, and so require similarly small, but consistent amounts for calibration.

2 History & Introduction

The very first semiconductor device was the photometer, developed by Von Siemens in 1875 [1](p6) (effectively a primitive solar cell) and hence, from the very inception of semiconductor devices, interaction with light has been closely related. However, it wasn't until 1907 until the electro-luminescence effect was discovered [2][p601] in the form of a point contact with a SiC substrate [2](p608) that emission of light, instead of absorption, was demonstrated. This electro-luminescence was different from radiative emission in both frequency and broadness of spectrum. But it wasn't for yet another 42 years, with the advent of the p-n junction in 1949, that the Light Emitting Diode (LED) as we know it today, was invented. [2](p608) [1](p1).

Unfortunately, due to them being made of Indirect Band-gap semiconductors, they had very poor efficiency, and it's progress remained stagnant with a lack of commercialisation [1] until the discovery of direct band-gap designs of GaAs in 1962 which had a much higher Quantum Efficiency leading to the first semiconductor laser. Soon after, in 1964, indirect bandgap materials also improved with the process of "doping" (addition of impurities) being discovered. [2](p608)

Up until this point, all the discovered LED technology had led to LEDs with spectra in the Infrared to the lower frequency end of the visible spectrum and in 1971, Pankove et. al. reported the first GaN LED and hence, the first LEDs with frequencies in the blue to UV range. [3](p3) However, traditional methods growing crystals of this nature were difficult and would lead to progress in a different technique: Epitaxial growth, where a film of crystal is grown from the surface of another. [1](p8) [4](p4). The only problem was, all of the known materials that would produce blue light had short-comings. Of note, was the epitaxial growth of GaN had a lattice mismatch to its best substrate (Sapphire) of 13.8% , compared with ZnSe (an alternative) on GaAs with a 0.3% mismatch and could thus easily be grown with few defects, leading to high quality crystals. [1](p11,13) [4](p4).

These films were generally deposited by either Molecular beam, or metalorganic vapour phase Epotaxy (MOVPE) but would require temperatures of $\approx$ 1300K and high pressures. As a result of the high rate of defects in GaN at the time ($10^9 cm^{-2}$) made ZnSe generally the more popular choice for scientists ($10^3 cm^{-2}$) which in turn, led to little research in the area. [4]

In 1990, Nakamura developed a technique (low carrier gas flow MOCVD) for producing high quality, uniform GaN using ammonia, but there was still the challenge of successfully doping it, meaning overcoming Mg acceptor dopant producing complexes with the H [4](p5) and finding a suitable Donor n-type. The first GaN pn junction was created in 1991 with an output power of $42\mu W$ and a External Quantum Efficiency of 0.18% at a wavelength of 430nm (far lower than the minimum 1mW for usefulness.)

Only after the discovery of InGaN, and its ideal properties as an active layer in a Double Heterostructure (DH), where it's bandgap was able to be tightly controlled through level of doping with tight control was the first Blue DH LED created, in 1993 which [4](p7), in turn, would lead to the first white

LED with the addition of a white-phosphor [2](p608),

The highest External Quantum efficiencies for LEDs are being achieved by GaInP encased in epoxy with 55%, [5] a massive improvement from the earliest versions of LEDs. In comparison, an oil lamp produces approximately $0.1lmW^{-1}$, an incandesceent bulb produces $16lmW^{-1}$, fluorescent $70lmW^{-1}$, but an LED will produce $300lmW^{-1}$ [4](p3) and the very first pn junction laser diode in visible spectrum of $\approx 0.1lmW^{-1}$ created by Holonyak in 1962. [6](p580)

Due to their high efficiency and ability to be tightly controlled, LEDs have been implemented in many technologies from simple indicator lights, to room illumination, but recently, they've been considered for calibration of Large photo-detection experiments such as Hyper-K and ANTARES neutrino detection experiments, where they aim to detect the Cherenkov radiation from Muon's produced by neutrinos interacting with matter. These interaction create very short pulses of very few photons of approximately blue colour, and thus, a similar pulse of light would be preferable to calibrate such an experiment.**CITATIONS NEEDED**

For this, the use of Blue, GaN-based LEDs pulsed at nano-second timescales has been proposed. However, as the lifetime of charge carriers in these materials is on the order of nanoseconds for InGaN/AlGaN DH LEDs [7], it has generally been considered practically impossible to produce such results, and conventional methods for driving LEDs result in rather low switching frequencies on the order of 1GHz.

To the contrary, it has been demonstrated that such pulses can be achieved experimentally **Rose citation?** and thus, an investigation into the mechanism will be conducted as the objective of this thesis, and whether experimental results can be simulated with current models and understanding.

3 Background Physics and Past Work

3.1 Fundamentals

At the core of this investigation are the physics of charge-carrier transport within semiconductors, and semiconductor junctions.

The simplest type of junction is the p-n, which made of a single crystal of semiconductor, doped with impurities to produce regions with different "majority carries". Donor atoms, which have more valence electrons than the intrinsic semiconductor, when infused into the semiconductor, result in an *n-type* region with a majority carrier of electrons. [6](p570) When these impurities become ionized by depletion of the carriers, an electric field is generated internally. [2](p16) *P-type* semiconductor is produced in much the same way, but with doping with atoms with fewer valence electrons ("acceptor" impurities), leaving "holes" which become the majority carriers in that region, which are simply absences of an electron compared to the intrinsic lattice. These holes act as if they were positive charges when subject to electric and magnetic fields and have the equivalent equations of motion. [6](p206-8)

By having a region of *n-type* and *p-type* semiconductor in contact, a p-n junction is formed, and the excess majority carriers from each attempt to diffuse to produce uniformity across the device, leading to the two majority carriers recombining at the interface to cancel each other out, leaving ionization of the impurities in both regions. The electric fields generated leads to potential barrier preventing complete diffusion and leads to separation of remaining electrons holes, and a depletion region near the potential barrier. [2](p80)

The understanding of the movement of these charge carriers is what is needed to understand the nature of this fast-switching of an LED.

As described above, carriers respond to electric fields. An application of an electric field, or magnetic field, will impart a Lorentz force

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \tag{1}$$

upon these particles, which will lead to an overall Drift current, $\vec{J}_{drift}$. With the addition of the current from the diffusion of these carriers in concentration gradient, $\vec{J}_{dif}$, we find the most widely used approximation to describe carrier transport in semi-conductors: [8](p58)

$$\vec{J}_{tot} = \vec{J}_{drift} + \vec{J}_{dif} \tag{2}$$

$$\vec{J}_n = q(n\mu\vec{E} + D\nabla n) \tag{3}$$

for electrons, where $\vec{J}_n$ is the current density vector, q is the charge on the electron, n is the charge density of electrons, μ is a mobility constant, $\vec{E}$ is the applied electric field, and D is the drift mobility constant, which in a non-degenerate case [in which the doping concentration N_D is $\ll N_C$ than the Number of conduction band states, and hence obey Boltzmann statistics [2](p18)] has the Einstein relation: [8](p58)

$$D = \frac{k_B T}{q}\mu \tag{4}$$

where k_B is the Boltzmann constant, and T is temperature.

This drift-diffusion equation can be modified to also account for conduction band energy E_C, as well as position dependence through effective mass $m*$ [which results from the parabolic energy-band equation $E(k) = \frac{\hbar^2 k^2}{2m*}$ [2](p13)] resulting in: [9](p21)

$$J_n = -q\mu_n n\nabla V + \mu_n n \nabla E_C + qD_n \nabla n - \frac{3}{2} q D_n n \nabla \ln(m*_C) \tag{5}$$

These current equations come are joined by two other equations of importance: Maxwell's equations (specifically Poisson's equation):

$$\nabla \vec{E} = \frac{\rho(x,y,z)}{\epsilon} \tag{6}$$

(where ρ is the charge density, and ϵ is permittivity) and the continuity equation for time-dependent systems:

$$\frac{\partial n}{\partial t} = G_n - R_n + \frac{1}{q}\nabla \cdot \vec{J}_n \tag{7}$$

In which G_n and R_n are thermal generation and recombination rates respectively. [2](p63) [3](p287) An alternative to this method of modelling transport within the device, is through the semi-classical Boltzmann Transport equation (BTE), describing the movement of the charge carriers through phase space, and in bulk material has the form:

$$\Big(\frac{\partial f(\vec{r},\vec{k},t)}{\partial t}\Big)_C = \frac{\partial f(\vec{r},\vec{k},t)}{\partial t} + \vec{v}\cdot\nabla f(\vec{r},\vec{k},t) + \frac{E}{\hbar}\cdot\nabla_k f(\vec{r},\vec{k},t) \tag{8}$$

Where $\vec{r}$ and $\vec{k}$ are position and phase-space vectors respectively, and $f(\vec{r},\vec{k},t)$ is the particle distribution function. [3](p70)

The evolution of this particle distribution function in time is handled through scattering of these electrons and holes treated as classical point particles:

$$\begin{aligned}\Big(\frac{\partial f(\vec{r},\vec{k},t)}{\partial t}\Big)_C \quad &= \textstyle\int\{f(\vec{r},\vec{k}',t)[1-f(\vec{r},\vec{k},t)]S(\vec{k}',\vec{k}) & (9)\\ &-f(\vec{r},\vec{k},t)[1-f(\vec{r},\vec{k}',t)]S(\vec{k},\vec{k}')\}d\vec{k}' & (10)\end{aligned}$$

Where $S(\vec{k}, \vec{k}')$ is the complete set of scattering rates from $\vec{k}$ to $\vec{k}'$. This scattering rate comprises of many terms, incorporating scattering due to Polar optical phonons, acoustic phonons, impurities, and non-parabolic band scattering amongst others. [3](p72-6)

Due to it being less popular than the Drift-Diffusion model, this method was not explored in much depth.

In the search for existing literature on the topic of nanosecond time-scale physics for LEDs, two papers of note were encountered which modelled LEDs under these conditions. The following subsections detail these two existing pieces of literature.

3.2 Existing Literature: Windisch

In the paper, *Large-Signal-Modulation of High-EfficiencyLight-Emitting Diodes for Optical Communication*, R. Windisch, et. Al [10] produced a model to explain the rapid recombination that happened up to a factor of two-times faster than the radiative recombination lifetime which, in general, is given by:

$$\tau_{rec} = \frac{n}{R_{rec}} \tag{11}$$

where R_{rec} is the radiative recombination rate, which in this case, is given by:[1]

$$R_{rec} = Bn(p_0 + n) \tag{12}$$

$$p = p_0 + n \tag{13}$$

where B is the radiative recombination constant, which is found experimentally, [10](p2) and p_0 is the doping concentration.

In this model, a number of assumptions were made: The model was based on a Double Heterostructure LED (one where two semiconductors materials are grown together, where the different semiconductors have different band-gaps in a "sandwich" like arrangement. [2](p56)), had instant voltage and current switching, and had negligible recombination through means other than radiative.

[1]This is not the only form found in literature for R_{rad}. Alternatives include:$R_{rad} = Bnp \cdot \Big[1 - \exp(-\frac{F_n - F_p}{k_B T})$ [3](p306) which includes Quasi-Fermi level dependancy, and $R_{rad} = B(np - n_0 p_0)$ [3](p289).

The boundary condition was set so at $t = 0$, $n = 0$, and increased in time following:

$$\frac{dn}{dt} = R_{in}(t) - R_{rec}(t) \tag{14}$$

Where R_{in} was simply the injection rate in, defined as:

$$R_{in} = \frac{J}{qw} \tag{15}$$

Where w was the width of the active region of the device. From this, an analytical solution for the carrier density was found: [10](p2,3)

$$n(t) = -\tfrac{p_0}{2} + \tfrac{\omega}{2B}\tanh(\tfrac{\omega}{2}(t + C)) \tag{16}$$

$$\omega = \sqrt{B^2 p_0^2 + 4\tfrac{JB}{qw}} \tag{17}$$

$$C = \tfrac{2}{\omega} arctanh(\tfrac{p_0 B}{\omega}) \tag{18}$$

and found that in the cases that the active region is highly doped, the device does indeed follow the expected Recombination lifetime exactly, but that in the case of low doping, where injected carrier concentration isn't proportionally negligible, the switch time can up to a factor of two faster. [10](p4)

However, this paper neglected the fact the carriers have charge and the associated Electric fields they'd generate, and focused exclusively on the active region of the device for simplicity.

3.3 Existing Literature: Brailovsky & Mitin

The second paper of note was *Fast switching of light-emitting diodes* by A.B. Brailovsky, and V.V.Mitin [7] where the authors produce an analytical model roughly based on the Drift-Diffusion equation. Again, like the previous paper, it assumes step-like instant-switching of the voltage.

The basis of the model is that switching speed may be increased by switching from direct current pumping to controlling the voltage across the LED. [7](p2) By starting with the equation for change in electron concentration around the p-region of the device:

$$\frac{\partial n}{\partial t} = D\frac{\partial^2 n}{\partial t^2} - \frac{n}{\tau} \tag{19}$$

where D is the diffusion coefficient, and τ is the recombination lifetime of the electrons in the p-region. After moving to dimensionless variables, setting the boundary conditions for $n(x,t)$ of $n(0,0) = 0$, $n(0,t) = \Delta n(1 - \exp(-\alpha t))$, and $n(\infty, t) = 0$, then applying a laplace transformation, a solution was found of:

$$n_p(x) = \Delta n e^{-x\sqrt{p+1}} \Big[\frac{1}{p} - \frac{1}{p+\alpha} \Big] \tag{20}$$

where α is a constant determining how fast concentration at the junction changes, and $n_p(x)$ is the Laplace transform of $n(x,t)$. [7](p2) This applies to the switching on of the device, and not to the switching off, and also isn't complete, as it ignores the Electric fields effects like the previous paper and neglects the drift effects in this solution. [7](p3)

From this solution, it was shown that the current in the device in the case of large values of α ($\approx > 10$) resulted in large reverse currents within the device, which could be the reason for switching times faster than the Radiative Recombination lifetime.

[7](p2)

4 Method

In order to explore the behaviour of the light emitting diode at short time-scales, two simulations were created to explore how different interpretations of the device would affect the results: A "macroscopic" simulation treating the device as a singular object, and a second "microscopic" simulation that took into account of internals of the device. The theoretical foundation for carrier transport in both device simulations was the Drift-Diffusion model due to its common usage, and simpler implementation than competing models, such as the Semi-classical Boltzmann Transport Equations that has its basis in the scattering of classical point-like holes and electrons from one set of phase-space coordinates to another. [3][p70-6]

The goal of both simulations was to produce plots of the full width half maximum (σ_{FWHM}) as a function of the number of photons in the pulse, to compare with the real device output.

Experimentally, voltage across the diode was ramped up and down over the period of several nanoseconds, with the forwards bias injecting charges, and the reverse bias pulling any remaining charges that hadn't combined, out of the device. In both models, to match the experimental conditions, the ap-

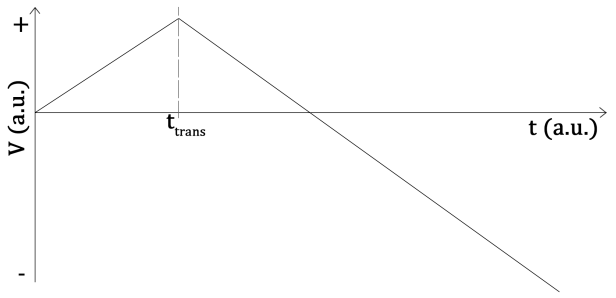

Figure 1: Graph of the approximate shape of the voltage ramp up and down used experimentally.

plied voltage to the diode was modelled as a tent map with the equations:

$$V(t) = \begin{cases} \alpha t & t < t_{trans} \\ \alpha(2t_{trans} - t) & t \geq t_{trans} \end{cases} \tag{21}$$

where V is the bias voltage, t is the time, t_{trans} is the transition time from voltage ramp up to voltage ramp down, and α is the voltage ramp scale (found experimentally to be $4Vns^{-1}$). This is an approximation to the experiment as it is not possible to switch instantly from forwards to backwards, with a more realistic model expecting some rounding to the graph graph during switching on, and switching from forwards to backwards.

4.1 "Macroscopic" Model

4.1.1 Conceptual Overview

In this simulation, the device was modelled as a single, complete object where charges would be injected, and photons ejected as a result of holes and electrons in the device combining (fig.2). This interpretation would be similar to treating the device to that of a leaky capacitor under Direct Current operation. This is a not-entirely unrealistic as "real" LEDs do have some small level of capacitance in the form of charge storage and depletion layer capacitance.**[CITE: VALEDAR,p30]** This model also means that opposite charges can interact immediately once inside, as the device is treated as so small that the charges are injected in the same place.

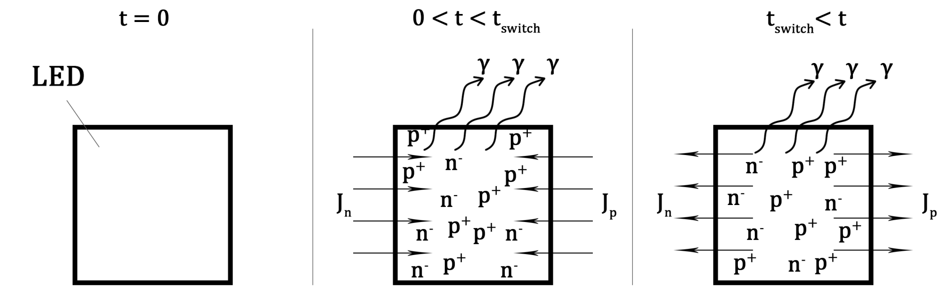

Figure 2: This figure displays the concept behind the macroscopic model where after time, $t = 0$, electrons and holes (n^- and p^+ respectively) are injected by currents J_n and J_p until the switch time, where upon the currents are reversed until all the charges have either combined to create photons (γ), or have been removed.

4.1.2 Calculating number of charges

The core calculation for number of charges in the device was done in a discrete time-step manner thus:

$$n(t) = n(t-1) + \left.\frac{dn}{dt}\right|_{n=n(t)} \quad (22)$$

$$\hookrightarrow \quad n(t) = n(t-1) + n_J(t) - n_R(t) \quad (23)$$

where $n(t)$ is the number of charges at time t, $n_J(t)$ is number of charges injected at time t, and $n_R(t)$ is the number of charges recombined at time t, with the boundary conditions, $n(0), n(t_{end}) = 0$ with the approximation that Thermal Generation and Recombination were negligible.

As both negative and positive charges would be injected at the same rate in this model, it is convenient to work in just one of them (in this case, negative) to simplify certain calculations.

To calculate the number of injected charges, the approximation was taken that the LED itself was an Ohmic device, with resistance of 5Ω, and using Ohms law:

$$I = \frac{V}{R} = \frac{qn_J}{t} \quad (24)$$

$$\hookrightarrow \quad n_J = \frac{V}{qR}t \quad (25)$$

where V is the external applied Voltage, R is the resistance, q is the charge on the electron, and t is the amount of time the current is being applied for.

The number of charges lost to Radiative Recombination can be found from the Radiative recombination rate equation given by: [3][p289,306] [2][p614]

$$R_{rad} = Bnp \tag{26}$$

$$\hookrightarrow \quad R_{rad} = Bn^2\Big|_{p=n} = n_R \tag{27}$$

where B is the bulk recombination rate constant.

4.1.3 Simulation Algorithms

The method for finding the number of photons, and FWHM of the associated pulse, was done by running the device simulation over a range of transition times, incrementing the transition time by a small step each loop:

```
1 Create arrays to store FWHM and Number of photons;
2 for $t_{trans} = 0$ to $t_{trans-max}$ do
3     Carry out the simulation at $t_{trans}$;
4     Save the resulting FWHM and Number of photons to arrays;
5 end
6 Print the arrays to file;
```

Algorithm 1: Macroscopic outer loop

In the program, the size of the transition step, the maximum transition step, and the simulation step can all be determined at run-time via console input at the start.

The algorithm for the nested simulation step of the outer loop is defined by Algorithm 2.

```
1  Create array for number of photons and current time;
2  Create Cumulative Radiation variable and set to 0;
3  while $n \geq 0$ do
4      Increment simulation at $t$ by $t_step$;
5      Add time onto end of the time array;
6      Find V at $t$ from tent map;
7      Find $n_J$ from V ;
8      Add $n_J$ to $n$;
9      Find $n_R$ (and hence number of photons);
10     Take $n_R$ from $n$;
11     Add $n_R$ onto end of photon array;
12     Add $n_R$ to cumulative photons.;
13 end
14 Use arrays to find FWHM;
15 Send total photons and FWHM to outer loop.;
```

Algorithm 2: Macroscopic Inner simulation loop

From both of these algorithms, it's clear that the smaller the time step for both the transition time and inner simulation time, the higher the resolution of the data, so generally, the step for the simulation was on the order of 10^{-3} times the transition time.

To find the FWHM of the pulse, Algorithm 3 was used. From this method, the error on the FWHM is no more than the width of two bins.

```
1 Find peak photon count bin in the photon array;
2 Find first bin to fall below half this value before and after peak bin;
3 Use bin numbers to find associated times in the time array;
4 Subtract the time from the after-peak bin from the before-peak bin;
```

Algorithm 3: Finding photon Pulse Full Width Half Maximum

The resulting data generated by this simulation was then outputted to a .csv file where it could be analysed.

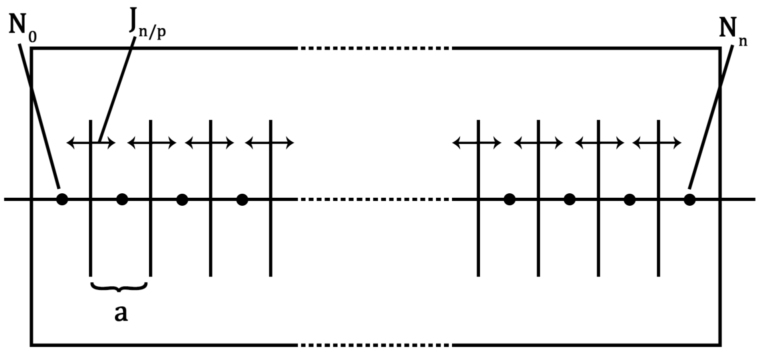

Figure 3: Visual representation of Microscopic model's segmented device. N_x is node at position x, a is the node width, and $J_{n/p}$ is the current density between two nodes for either electrons(n), or holes(p).

4.2 "Microscopic" Model

4.2.1 Conceptual Overview

This model was produced to compare the results with the Macroscopic interpretation of the device. It centred around numerically solving the charge Continuity Equations through calculating both the Drift-Diffusion and Poisson's equations: [9][p21]

$$J_n = -q\mu_n n\nabla V + \mu_n n\nabla E_c + qD_n\nabla n \tag{28}$$

$$J_p = -q\mu_p p\nabla V + \mu_p p\nabla E_v - qD_p\nabla p \tag{29}$$

$$\nabla \cdot \vec{D} = \rho \tag{30}$$

where the first and second equations are the drift-diffusion equations with the addition the second terms to account for spacial variation of band structure, and the third equation is Gauss' law.

This was done by dividing the device into segments. A visual representation can be found in Figure 3.

Each "node" along the device has a set width, a, and contain several variables and constants: The charge densities for electrons and holes (n and p respectively), and the potential at that point V being the variables, and with constants determining doping of acceptor atoms N_a, doping of donor atoms

N_d, and the Conduction and valence band relative energies, E_c and E_v respectively.

The device would be constructed as a "string" of these Nodes, with a set length, and area perpendicular to the current, which would limit it strictly to cuboid shaped devices.

Due the increased complexity of this model, there are more initial conditions to be considered, such as how many nodes there are, the specifications of each of the nodes, the overall dimensions and parameters of the device, etc. so the program running this simulation was designed to have similarly broad functionality. A command-line interface was built to allow a user to do the following tasks: load and save a "device" to file, bring a device to equillibrium, run the simulation on a device for a set of conditions, and run the a range of simulations for a range of conditions to calculate a range of photon pulses for different transition times to find the how the FWHM varied with number of photons in a pulse.

4.2.2 Converting the equations to numerical form

In order to do calculation using the Drift-Diffusion and Poisson's equations, they first had to be discretised. This could be achieved through the use of Taylor expansion's. [11] [12]

Using the Taylor expansions of:

$$f(x+\delta x) = f(x) + \delta x \frac{df}{dx}\bigg|_{x=x+\delta x} + (\delta x)^2 \frac{d^2 f}{dx^2}\bigg|_{x=x+\delta x} + ... \quad (31)$$

$$f(x-\delta x) = f(x) - \delta x \frac{df}{dx}\bigg|_{x=x-\delta x} + (\delta x)^2 \frac{d^2 f}{dx^2}\bigg|_{x=x-\delta x} - ... \quad (32)$$

$$(33)$$

it was possible to find approximations for both first a second derivatives by using:

$$\frac{df}{dx}\bigg|_{x+\delta x/2} = \frac{f(x+\delta x)-f(x-\delta x)}{\delta x} \quad (34)$$

$$\frac{d^2 f}{dx^2}\bigg|_{x_i} = \frac{f(x+\delta x)+f(x-\delta x)-2f(x_i)}{(\delta x)^2} \quad (35)$$

Using these approximations, it was possible use Gauss' law to find the potential at each node point using: [12]

$$\nabla \cdot \vec{D} = \rho \tag{36}$$

$$\nabla^2 \cdot V = \frac{\rho}{\epsilon_r \epsilon_0} \tag{37}$$

$$\nabla^2 \cdot V = \frac{V_{i+1} + V_{i-1} - 2V_i}{a^2} \tag{38}$$

$$\hookrightarrow \quad \frac{V_{i+1} + V_{i-1} - 2V_i}{a^2} = \frac{\rho}{\epsilon_r \epsilon_0} \tag{39}$$

$$\frac{V_{i+1} + V_{i-1} - 2V_i}{a^2} = \frac{q}{\epsilon_r \epsilon_0}(p + N_D - n - N_A) \tag{40}$$

Where i indicates the node index. Using the same method, current densities J_n and J_p can be calculated thus:

$$J_n = -q\mu_n(\frac{n_{i+1} + n_i}{2})(\frac{V_{i+1} - V_i}{a}) \tag{41}$$

$$+ \quad \mu_n(\frac{n_{i+1} + n_i}{2})(\frac{E_{c(i+1)} - E_{c(i)}}{a}) + qD_n(\frac{n_{i+1} - n_i}{a}) \tag{42}$$

$$J_p = -q\mu_p(\frac{p_{i+1} + p_i}{2})(\frac{V_{i+1} - V_i}{a}) \tag{43}$$

$$+ \quad \mu_p(\frac{p_{i+1} + p_i}{2})(\frac{E_{v(i+1)} - E_{v(i)}}{a}) - qD_p(\frac{p_{i+1} - p_i}{a}) \tag{44}$$

$$\tag{45}$$

These three equations form the basis for the equilibrium and simulation abilities of the program.

4.2.3 Program functionality: Saving & Loading

To allow use of user defined devices in the simulation process, a method of inputting the data rather than having it hard coded was necessary. To achieve this, a loading and saving method was designed.

A specific *.csv* file format was developed that could be read by the program. An overview of the specifics of the format can be found in the **APPENDIX: TODO** to allow for the reader to create their own. This file would contain all the necessary information to create a device in the program, including: device length, area, resistance, number of nodes, and each of the individual node's data. To enable ease of reading for the program, a second internal file-type was created based on this file called a "doesn't include commas" file that was, in essence, this specifically structured .csv file type with all commas removed. This .dic file was easier to read and less error prone on reading, and would be automatically generated by the program.

The program could also save the current device in memory to a .csv file in the correct format to be transported, permanently recorded, or opened in an external spreadsheet or text editor program. This makes editing or altering device data far easier for the user.

4.2.4 Program functionality: Equillibrium

Once a device has been loaded into the program, the user is able to bring the device to "equilibrium". This is done by running the continuity equation simulation on the device with no external input until a minimum total internal current is reached (defined by the user, recommended $\approx 1cm^{-3}s^{-1}$ or less) where total internal current is defined by:

$$J_{tot} = \sum_{i=1}^{n} J_i \tag{46}$$

$$J_i = J_{n(i)} + J_{p(i)} \tag{47}$$

This step is important to ensure the calculations have the intended behaviour and that running a simulation cycle doesn't result in excess photons being released from the device not already being in steady-state equilibrium. The following algorithm was used to achieve this: As running this algorithm at

```
1 while $J_{tot} < tolerance$ do
2     Calculate V at each node;
3     Calculate $J_n$ between each node and transfer charge density;
4     Calculate $J_p$ between each node and transfer charge density;
5     Cancel any charges that would recombine;
6     $J_{tot} = \sum J_{n(i)} + J_{p(i)}$
7 end
```

Algorithm 4: Algorithm for bringed device to equillibrium

standard J values would be very slow, a user-defined "Approximation" multiplier is used to speed this process up (typically a value from 10^3 to 10^6 is recommended.) This multiplier simply takes what the exchange current J would be, and multiplies it by this value. Very large values has undefined behaviour, and in such cases, reloading the device from file is necessary, but without it, this process could take many millions of iterations.

Though not used in the final program, the code has debug functionality to print device status to file and console whilst this is happening, as well as print J_tot to file for each iteration.

4.2.5 Program functionality: Single simulation

Using many of the same principles as the "Macroscopic" device model program and its algorithms, this command takes the device loaded and injects charges following the voltage tent map, and Ohmic D.C. modelling. It uses exactly the same code for finding FWHM of the pulse, though due to fundamental differences in design, the algorithms for finding the number of photons released from recombination are slightly different.

From device equilibrium where:

$$n_R(t=0) = 0 \tag{48}$$

$$J_{tot}(t=0) \approx 0 \tag{49}$$

$$\hookrightarrow \quad \left.\frac{dn}{dt}\right|_{t=0} \approx 0 \tag{50}$$

and same for holes, injection of electrons from one end and holes from the other can begin.

This is done in the same time-step like manner of the Macroscopic device, however the algorithm is adjusted to:

```
1  while $n_R \geq 0$ do
2      Add $t_step$ to $t$;
3      Find V at $t$;
4      Find $J_{input}$ from V;
5      Find $V_i$ for each node;
6      Propagate $n$ and $p$ through device with $J_n$ and $J_p$;
7      Find $n_R$ at each node and remove respective charge densities;
8      Save $n_R$ to histogram bin arrays;
9      Save respective $t$ to corresponding array;
10 end
11 Find FWHM using Algorithm 3;
```

Algorithm 5: Segmented device single simulation run algorithm.

In the case where the external bias Voltage is reversed, so will be the J_{input} current density, pulling that charges from that end of the device where

$$J_{input} = \frac{V}{AR} \tag{51}$$

where A is the area perpendicular to the direction of current.

4.2.6 Program functionality: Full simulation

This command treats the single simulation as the Macro model would treat the "inner loop" algorithm (see Algorithm 2) and in this respect, the Full simulation command is much like the "outer loop" algorithm (see Algorithm 1).

The same basis of incrementing the transition time in a stepwise fashion to find the associated FWHM and number of photons until reaching a maximum transition time is employed. However, due to the device being left in a state of none-equilibrium between each run, it has to be reloaded from file between each iteration. The algorithm is as follows:

```
1 for $t_{trans} = 0$ to $t_{trans} = t_{transMax}$ do
2     Load device in equilibrium;
3     Single iteration simulation (Algorithm 5);
4     Log both FWHM and $n_R$;
5 end
6 Save FWHM and $n_R$ arrays to file;
```

Algorithm 6: Segmented device full ranged simulation algorithm.

in which the transition time maximum, step, and simulation time-step are defined by the user on command call.

5 Results

5.1 Macro Simulation

The simulation was run for a device of Volume $= 1 \cdot 10^{-11} cm^3$ and constants for GaN of Radiative Recombination $B = 2.4 \cdot 10^{-17} m^3 s^{-1}$, an estimated experimental External Quantum Efficiency of 1%, and Direct Current resistance of 5Ω. The Voltage ramp up-down constant was taken experimentally as $4Vns^{-1}$. The simulation time step was $10^{-12}s$, with ramp up to down switching time increased in steps of $5 \cdot 10^{-10}s$ until $5 \cdot 10^{-9}s$. Error on FWHM values are two times a simulation time step from how its value is found, and the error on the number of photons is $\sqrt{N}$ where N is the bin's number of photons. For almost all plots, the error bars are too small to be visible.

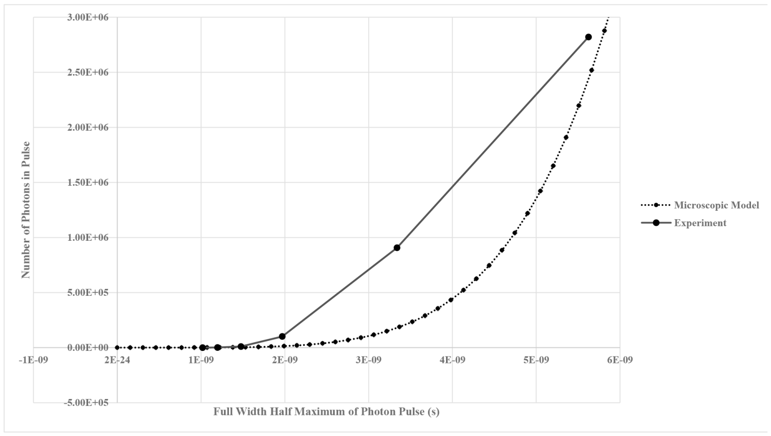

Figure 4: Data from Macroscopic model with Experimental Data plotted alongside.

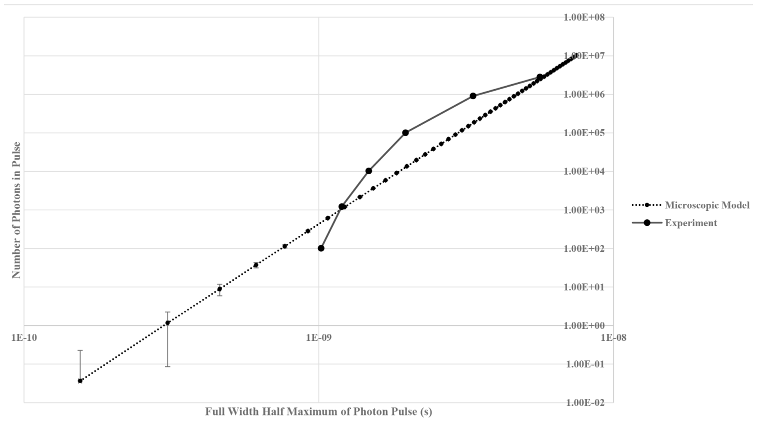

Figure 5: Log-log plot of Microscopic model data.

5.2 Micro Simulation

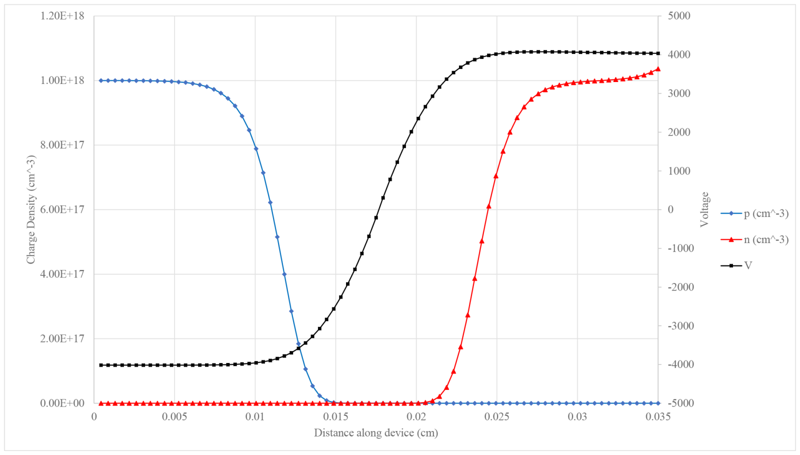

Figure 6: Equilibrium Data plot of charge densities and potentials for a simulated pn device.

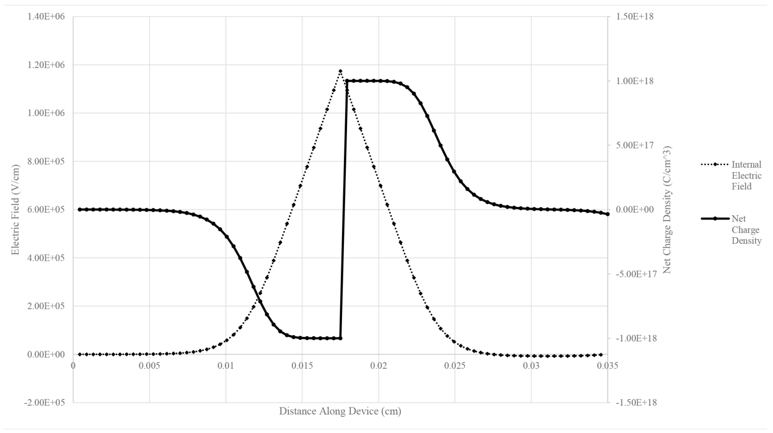

Figure 7: Equilibrium Data plot of net charge densities and Internal Electric Field for a simulated pn device.

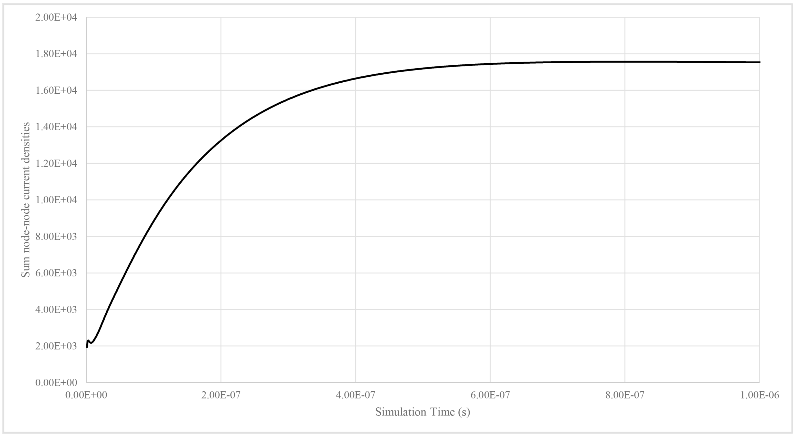

Figure 8: Total current density between all nodes in the device for Instant switch on of 0.1 Amps Direct Current for $1\mu s$*.*

6 Discussion

In this section, the results of the modelling compared to real-world results will be discussed. Topics of interest will be: Does the model predict the $N_\gamma \propto t^3$ results instead of the linear response predicted by Windisch's instantaneous current model ($N_\gamma \propto t^2$) or does it manage the closer to experimental result of t^4?

Discuss the model's simulated capacitance and whether it's similar to that measured experimentally, or predicted by **Veledar**.

Did the model produce any temperature dependency graphs for photon emission, and if so, how do they again compare to experimental results?

7 Conclusion

In this section talk about the limits of the model (only 1D, inaccuracies with reality, etc) and discuss known ways to improve it. Discuss reasons for them not already being implemented (lack of knowledge about device[read: band structure, dimension, composition], more accurate models much more time consuming to build, etc...)

8 References

Format requested: "Last name, First initial. (Year published). Article title. Journal, Volume (Issue)" **How do I get it in that form ↓ ? Im aware a couple of these haven't worked properly from Bibtex, I'll find a fix later.**

References

[1] B. Gil, *Group III Nitride Semiconductor Compounds Physics and Applications.*

[2] K. N. K. Sze S.M., *Physics of Semiconductor Devices.*

[3] J. Piprek, *Nitride Semiconductor Devices.*

[4] S. Nakamura, *Background Story of the Invention of Efficient Blue InGaN Light Emitting Diodes.*

[5] M. R. Krames, M. Ochiai-Holcomb, G. E. Hfler, C. Carter-Coman, E. I. Chen, I.-H. Tan, P. Grillot, N. F. Gardner, H. C. Chui, J.-W. Huang, S. A. Stockman, F. A. Kish, M. G. Craford, T. S. Tan, C. P. Kocot, M. Hueschen, J. Posselt, B. Loh, G. Sasser, and D. Collins, "High-power truncated-inverted-pyramid (alxga1x)0.5in0.5p/gap light-emitting diodes exhibiting ¿50% external quantum efficiency," *Applied Physics Letters*, vol. 75, no. 16, pp. 2365–2367, 1999.

[6] C. Kittel, *Introduction to Solid State Physics.*

[7] A. B. Brailovsky and V. Mitin, "Fast switching of light-emitting diodes," vol. 44, pp. 713–718, 04 2000.

[8] S. S.M., *High-Speed Semiconductor Devices.* Wiley.

[9]

[10] R. Windisch, A. Knobloch, M. Kuijk, C. Rooman, B. Dutta, P. Kiesel, G. Borghs, G. H. Dohler, and P. Heremans, "Large-signal-modulation of high-efficiency light-emitting diodes for optical communication," *IEEE Journal of Quantum Electronics*, vol. 36, pp. 1445–1453, Dec 2000.

[11] B. Nkonga, “Numerical methods for pde: Finite differences and finite volumes,” 2009.

[12] G. Klimeck, 2012.

9 Appendix

The following sections detail an overview of the most important segments of code and its functionality. The complete source code for the project can be found online at `https://github.com/donmegamuffin/Thesis`.[2] Much of the following code has had comments removed or has been slightly altered for clarity within the code snippets.

9.1 Code Overview: General

The following subsections are the explanations of general code used across both simulations.

9.1.1 FWHM calculation methods

The calculation of the FWHM is split across 4 methods for readability and reuse. The algorithm for finding the FWHM is found in algorithm 3. Each step is handled by a separate method.

The first method simply handles finding the peak value of all the bins of a std::vector array, and returns its index:

```cpp
int Device_1D::findPeakBin(std::vector<double> RadVector)
{
  double PeakValue = 0;
  int PeakValueAssociatedIndex = 0;
  for (std::size_t i = 0; i < RadVector.size(); i++)
  {
    if (RadVector[i] > PeakValue)
    {
      PeakValue = RadVector[i];
      PeakValueAssociatedIndex = i;
    }
  }
  return PeakValueAssociatedIndex;
}
```

Listing 1: Method for finding histogram peak bin

It simply takes in a general vector of values (with the assumption the bins follow a single-peak curve), and iterates through each value, checks if it's higher

[2] As of writing on 24/03/2018.

than the current maximum value, and if so, sets the new peak value and associated index. It finally returns the peak with the maximum value.

To find the width of the curve, finding the points at which curve reaches half the peak is necessary. However, as the array is discrete, this has to be approximated to the nearest bin, giving the error on this value a maximum of $\pm 2 \cdot t_{step}$, because each bin is calculated over a single time step.

This is handled by the following two functions that have similar functionality. The first one finds the bin that falls below the half maximum, by starting at the peak value index in the vector, and working backwards:

```cpp
int Device_1D::FWHMfindFirstBinBelow(std::vector<double> RadVector,
        int PeakIndex)
{
  int Index = 0;
  for (std::size_t i = PeakIndex; i >= 0; i--)
  {
    if (RadVector[i] < (RadVector[PeakIndex] / 2))
    {
      Index = i;
      break;
    }
  }
  return Index;
}
```

Listing 2: Method for finding first bin to fall below half peak before peak

The second method does the same, but finding the first bin after the peak, starting from the peak and incrementing the bin index until it's found.

```cpp
int Device_1D::FWHMfindFirstBinAbove(std::vector<double> RadVector,
        int PeakIndex)
{
  int Index = 0;
  for (std::size_t i = PeakIndex; i < RadVector.size(); i++)
  {
    if (RadVector[i] < (RadVector[PeakIndex] / 2))
    {
      Index = i;
      break;
    }
  }
  return Index;
}
```

Listing 3: Method for finding first bin to fall below half peak after peak

This function puts the previous functions together to find the photon emission pulse bin's FWHM indices, and calculates the time by finding the bin's associated times by using its associated time vector:

```
double Device_1D::calculateFWHM(std::vector<double> RadVector,
        std::vector<double> TimeVector)
{
  //Get the peak of the curve
  int PeakBin = findPeakBin(RadVector);
  //Get the first bin below half max
  int LowBin = FWHMfindFirstBinBelow(RadVector, PeakBin);
  //Get the first bin after half max
  int HighBin = FWHMfindFirstBinAbove(RadVector, PeakBin);
  //Return the difference in time between the associated time vector bins
  return (TimeVector[HighBin] - TimeVector[LowBin]);
}
```

Listing 4: Method for calculating FWHM from an std::vector of values

These functions are implemented in both simulations as the method for finding FWHM of the photon pulses.

9.1.2 External Voltage ramp up/down

Both simulations have the same implementation for the calculation for the external voltage being applied to the device. It applies the tent-map implementation of external voltage ramp up and down with the equation:

$$V(t) = \begin{cases} \alpha t & t < t_{trans} \\ \alpha(2t_{trans} - t) & t \geq t_{trans} \end{cases} \tag{52}$$

As parameters, it takes both the time the simulation is at (*time*) and the point at which the simulation's voltage changes from ramp-up to ramp-down (*transition_time*), and returns the value $V(t)$ at that point.

```
double Device_1D::inputV(double time, double transition_time)
{
  const double Vramp = 4e9;
  if (time < transition_time)
  return Vramp * time;
  else
  return Vramp * ((2 * transition_time) - time);
}
```

The constant on line 3 is taken from experimental apparatus and was $\approx 4 \cdot 10^9 Vs^{-1}$. The two return statements on line 5 and 7 are simply the function before and after the transition time.

9.2 Code Overview: Micro

The following subsections are the explanations of code used within the microscopic, segmented simulation.

9.2.1 Current density calculation methods

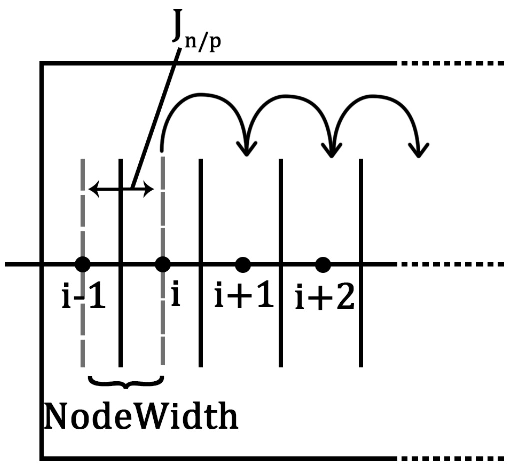

Figure 9: Diagram of calculateJnLEC method's functionality.

This section contains the information on how current was calculated within the device.

Four functions were written, all with very similar structures but with alterations dependent on whether the order was starting from the left, or from the right, and whether it was done for the holes or electrons: JnL, JnR, JpL, and JpR.

For this example, JnL will be used to demonstrate the structure, with its depiction shown in Figure 9. The code worked by taking the device's std::vector¡Node¿s and manipulating the electron (or hole) densities by applying:

$$J_n = -q\mu_n n\nabla V + \mu_n n\nabla E_c + qD_n\nabla n \tag{53}$$

$$\hookrightarrow \quad J_n = -q\mu_n(\frac{n_{i+1}+n_i}{2})(\frac{V_{i+1}-V_i}{a}) \tag{54}$$

$$+ \quad \mu_n(\frac{n_{i+1}+n_i}{2})(\frac{E_{c(i+1)}-E_{c(i)}}{a}) + qD_n(\frac{n_{i+1}-n_i}{a}) \tag{55}$$

$$\tag{56}$$

starting from the second node, to the last in that order, calculating the current between the node, and the that of the one to its left.

Line 3 creates a variable to store the absolute value of the current between the two nodes and adds it to a cumulative value to be returned once the function is complete. This return is used in parts of the program to guage whether the device is in equilibrium or not.

Lines 11 and 12 simply find the current and transfer the difference from one to other with correct dependence depending on the direction of the current density.

The exchange scale multiplies this transfer which results in significantly fewer loops required where time is not a factor in the calculation. As current density is charge transfer per unit time, it also works as a crude method of integrating the current calculation over a time period, treating it as constant current.

By using this approximation method, it is only valid for relatively small approximations where $J_{nL}t \ll n$ and is the equivalent of doing:

$$\sum_{i=1}^{x} \frac{dn}{dt}\Big|_{t=t_i} = \lim_{n''(t)\to 0} x \cdot \frac{dn}{dt}\Big|_{t=t_1} \tag{57}$$

and is assumed that the rate in change in current transfer is smooth and small as to allow for this approximation. Because, in this approximation, $\frac{dn}{dt}$ is only needed to be calculated once, this is far faster for use in the equilibrium code by applying an approximation factor.

```
1  double Device_1D::calculateJnLEC(double exchangeScale)
2  {
3    double JnL_cum = 0;
4    for (std::size_t i = 1; i < nAry.size(); i++)
5    {
6      double dn = (nAry[i].n - nAry[i - 1].n) / nodeWidth;
7      double dV = (nAry[i].V - nAry[i - 1].V) / nodeWidth;
8      double dEc = (nAry[i].Ec - nAry[i - 1].Ev) / nodeWidth;
```

```
    double n = (nAry[i].n + nAry[i - 1].n) / 2;
    double JnL = (-q * mu*n*dV) - (mu*n*dEc) + (q*D*dn);
    nAry[i].n = nAry[i].n - (JnL*exchangeScale);
    nAry[i - 1].n = nAry[i - 1].n + (JnL*exchangeScale);
    JnL_cum += abs(JnL);
  }
  return JnL_cum;
}
```

Listing 5: Example of the current calculation algorithm

9.2.2 Voltage calculation method

This consists of a single function which cycles through each of the nodes in the device and does the following calculation following from the derivation shown in section 4.2.2:

$$\frac{V_{i+1} + V_{i-1} - 2V_i}{a^2} = \frac{q}{\epsilon_r \epsilon_0}(p + N_D - n - N_A) \tag{58}$$

$$\hookrightarrow \quad V_i = \tfrac{1}{2}[\frac{q}{\epsilon_r \epsilon_0}(p + N_D - n - N_A) + V_{i+1} + V_{i-1}] \tag{59}$$

with the cap-ends of the device having slightly modified calculations, for first node of the device:

$$V_i = \frac{1}{2}[\frac{q}{\epsilon_r \epsilon_0}(p + N_D - n - N_A)] + V_{i+1} \tag{60}$$

and last node of the device:

$$V_i = \frac{1}{2}[\frac{q}{\epsilon_r \epsilon_0}(p + N_D - n - N_A)] + V_{i-1} \tag{61}$$

```
void Device_1D::calculateVoltages()
{
  //Loop through each of the nodes
  for (std::size_t i = 0; i < nAry.size(); i++)
  {
    double a2 = pow(nodeWidth, 2);
    double C = (q*a2) / e_0;
    double netCharge = nAry[i].p + nAry[i].Nd - nAry[i].n - nAry[i].Na;
    //If node is at a cap position change calculation
    if (i == 0)
    {
      nAry[i].V = (C*(netCharge)+ nAry[i + 1].V);
    }
    else if (i == (nAry.size() - 1))
    {
            nAry[i].V = (C*(netCharge)+nAry[i - 1].V);
        }
        else  //Do normal calculation
        {
            nAry[i].V = 0.5*(C*(netCharge)+nAry[i - 1].V + nAry[i + 1].V);
        }
    }
}
```

Listing 6: Voltage calculation method

The first **if** statement implement the calculation if it's doing it for the first node, and the **else if** statement implements the end-node calculation. The first lines of the for loop are there to break down the calculation's constants for a node and make later calculations more readable.

9.2.3 Charge injection Method

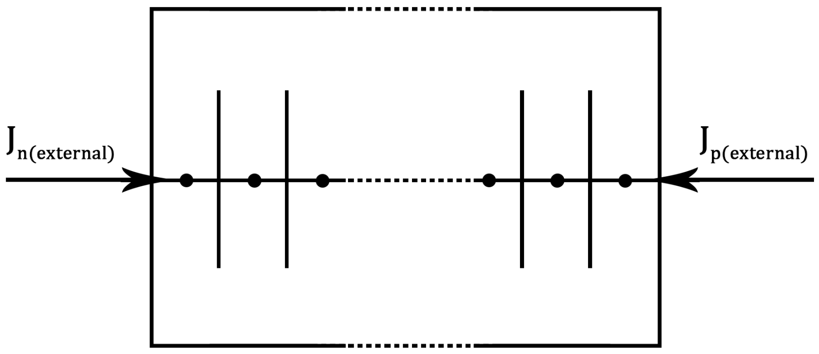

Figure 10: Visualisation of charge injection method

This method handles the injection of charges at the end nodes of the device. A visualisation of what the code is doing can be seen in figure 10 and simply injects these charges into the end nodes of the device. It has two parameters: Current density ($J_{external}$) and the Injection Duration (using the t_{step} for the simulation). The total density of charges to be added to the ends in a single time step of the device will simply be $\frac{dn}{dt} = J_{external} \cdot t$.

The $J_{external}$ used in the parameter for the simulation is the current density calculated assuming Ohm's Law is correct:

$$J = \frac{I}{A} = \frac{V}{A \cdot R} \tag{62}$$

```
void Device_1D::injectCharges(double CurrentDensity, double injectionDuration)
{
  //Inject n in left, p on right.
  nAry[0].n += (CurrentDensity*injectionDuration);
  nAry[nAry.size() - 1].p += (CurrentDensity*injectionDuration);
  return;
}
```

Listing 7: Charge injection method

9.2.4 Simulation inner loop method

This section covers the details of how the inner loop is handled. There are two *overloads* for this function but it will be demonstrated with the variant used for the full simulation rather than the single-use simulation which has minor changes (e.g. outputs a full Debug file of all the data, etc.).

It requires references to two external **double** values as Output parameters to pass data calculated within this inner loop for both the FWHM and total photons calculated by the simulation. The last two parameters are the simulation step time and the time the voltage transitions from ramping up to ramping down.

```
void Device_1D::simulateDevice(double & outFWHM,
      double & outRrad, double t_step, double t_trans)
{
  double t_now = 0; //current device simulation time
  std::vector<double> t_Vec;  //Stores all the times
  double Rrad_now;     //Single Loop number of photons
  std::vector<double> Rrad_Vec; //Current device radiation
  double Rrad_cum = 0;  //Cumulative radiation
  double inV = 0;      //Input voltage
  double J = 0;      //J

  //Main function loop
  do
  {
    t_now += t_step;
    t_Vec.emplace_back(t_now);

    inV = inputV(t_now, t_trans);
    J = inV / (A * R);

21      injectCharges(J, t_step);
22      calculateVoltages();
23      calculateJpREV(t_step);
24      calculateJnLEC(t_step);
25
26      Rrad_now = calcRadRecombine(t_step);
27      Rrad_Vec.emplace_back(Rrad_now);
28      Rrad_cum += Rrad_now;
29    } while (Rrad_now>0);
30
31    outFWHM = GetRadFWHM(Rrad_Vec, t_Vec);
32    outRrad = Rrad_cum;
33  }
```

The simulation is contained in a **do....while** function as to allow it to run always run a single time, and then loop as long as the Radiative recombination is above zero, automatically terminating the loop once all photons have been emitted. This way of doing it meant that it works universally for any length of transition time.

In line 18, the "inputV(...)" function, is the function used to calculate the voltage at a point in time, calculated from the tent map discussed in section 4.2.2 which returns the external Voltage applied. The following lines calculate the external current, inject the charges, and then calculate the internal current.

Line 26 calculates the radiative recombination across every node using:

$$R_{rad} = Bnp \cdot t_{step} \tag{63}$$

where R_{rad} is the number of photons emitted during the simulation time-step, which is then put at the end of a Vector to be stored (so it can be user later to output to file in the other overload) and then added to the total cumulative photons in line 28.

At the end of the simulation loop, the FWHM is calculated in line 31 and passed to the Output parameter *outFWHM* and the cumulative number of photons passed to *outRad*.

9.2.5 Full ranged simulation (outer loop) method

The following code is the full implementation of the algorithm 6 found in section 4.2.6. Lines 5 to 10 hold data for later use, whilst lines 16 to 28 are the actual implementation of the algorithm with 26 ad 27 putting the FWHM and associated cumulative photon counts of the loop onto the data storage std::vectors in 6 and 7. These two std::vectors are then written to a .csv file through lines 30 onwards and then closing the file at the end.

```cpp
1  void Device_1D::fullSim(std::string eqmFileName,
2            double timeStep, double transStep, double transMax)
3  {
4    //Simulation Data arrays for output
5    std::vector<double> t_trans_Vec;
6    std::vector<double> FWHM_Vec;
7    std::vector<double> Rcum_Vec;
8
9    double FWHM_now = 0;  //Holds the loop FWHM
10   double Rcum_now = 0;  //Holds the loop Rcum
11
12   //Open filestream for printing results
13   std::ofstream resultsFile;
14   resultsFile.open(eqmFileName + "_Rad_results.csv",'w');
15
16   for (int i = 0; i*transStep < transMax; i++)
17   {
18     //Set up device
19     loadState(eqmFileName);
20
21     //Add the transition time to trans vector
22     t_trans_Vec.emplace_back(i*transStep);
23
24     simulateDevice(FWHM_now, Rcum_now, timeStep, i*transStep);
25
26     FWHM_Vec.emplace_back(FWHM_now);
27     Rcum_Vec.emplace_back(Rcum_now);
28   }
29   //Print Results to file
30   for (auto a : t_trans_Vec)
31   {
32     resultsFile << a << ",";
33   }
34   resultsFile << std::endl;
35   for (auto a : FWHM_Vec)
36   {
37     resultsFile << a << ",";
38   }
39   resultsFile << std::endl;
40   for (auto a : Rcum_Vec)
41   {
42     resultsFile << a << ",";
43   }
44   resultsFile << std::endl;
45
46   resultsFile.close();
47 }
```

Listing 8: Full outer simulation loop method

9.2.6 Equilibrium method

The following code is used for bringing a loaded device to equilibrium. It follows the principles of algorithm 4 in section 4.2.4.

As parameters it takes the a user-defined ”Tolerance”, an ”exchange scale” (approximation factor), an output stream to write to a file, and a bool for defining whether it should write to console or not.

The ”Tolerance” value is used in two ways: first to define what is considered ”equilibrium” in terms of total current density maximum across the device, and as the point at which the simulation may reach convergence in the case it never goes below tolerance value. This is done on line 8, where the first part of the **if** statement checks for the first condition, and the second part checks the previous loop’s total current isn’t closer than a limit (of $0.1 \cdot Tolerance$)to the current loop’s. Lines 4 and 5 are used to hold those two loop total current densities.

```
1  void Device_1D::bringToEqm(double Tolerance, double exchangeScale,
2  std::ostream &fileOutput, bool bPrintConsole)
3  {
4    double Jtot = 1e13;
5    double J_prev = 1e8;
6    //Whilst the current is higher than the current equilibrium limit
7    //Loop until Jtot is below user defined limit
8    while (Jtot > Tolerance && abs(Jtot - J_prev)>(0.1*Tolerance))
9    {
10     J_prev = Jtot;
11     Jtot = calculateJnREC(exchangeScale) + calculateJpLEV(exchangeScale);
12     calculateVoltages();
13     cancelCharges();
14     //OPTIONAL : Print the current found to console
15     if (bPrintConsole)
16     {
17       std::cout << "\r" << Jtot << "                    ";
18     }
19     //Print current to file
20     fileOutput << Jtot << std::endl;
21   }
22   if (bPrintConsole) { std::cout << "\n Equilibrium calculation finished." << std::endl; }
23   return;
24 }
```

Lines 15 to 18 and line 22 only run if the bool in the parameter is set to true, and simply keeps the user up to date in the console.