Bachelor Thesis

Simulation of the response of Silicon Photomultipliers with Julia

Cedric C. H. Engler

HAW HAMBURG

Cedric Charles Henry Engler

Mat.-Nr.: [redacted]

Simulation of the response of Silicon Photomultipliers with Julia

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang *B.Sc. Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstgutachter: Prof. Dr. Matthias Kronauge
Zweitgutachterin: Prof. Dr. Erika Garutti

Eingereicht am: 27.08.2024

Abstract

This thesis focuses on modeling and simulating the response of Silicon Photomultipliers (SiPMs) and aims to develop a proof-of-concept for using the Julia programming language as the foundation. Key topics covered include the working principle of SiPMs, primary and secondary discharges, and the generation of the output signal. The simulation itself incorporates these features and implements them in a framework. Results are displayed, validated, and benchmarked. A preceding implementation is used as a comparison for the latter. This thesis contributes to the understanding and optimization of SiPMs for low-light and high-light scenarios and enables a systematic study.

Kurzzusammenfassung

Diese Arbeit ist darauf fokussiert, Silizium Photomultiplizierer (SiPMs) zu modellieren und anschließend zu simulieren. Dabei soll ein vielseitiges Framework, geschrieben in Julia, entstehen, welches als Machbarkeitsstudie verwendet wird. Die wichtigsten Punkte sind dabei das grundlgende Funktionsprinzip von SiPMs, sowie die Generation des Ausgangssignal durch primäre und sekundäre Effekte. Das Framework baut darauf auf. Die Ergebnisse werden dargestellt und validiert. Abschließend folgt eine Feststellung der einflussreichsten Parameter durch Benchmarks, wobei vorangegangene Untersuchen zum Vergleich herangezogen werden. Diese Arbeit trägt dazu bei, das Verständnis von SiPMs zu festigen und erlaubt systematische Untersuchungen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Silicon Photomultipliers (SiPMs) are used in various applications, ranging from Time-Of-Flight sensors (such as Light Detection and Ranging, or LiDAR for short) to positron emission tomography, quantum cryptography, and astrophysics [1]. Figure 1.1 shows the application of an SiPM in a LiDAR sensor, where it acts as the detector for reflected light after it has been transmitted from the source and ricocheted back from an object.

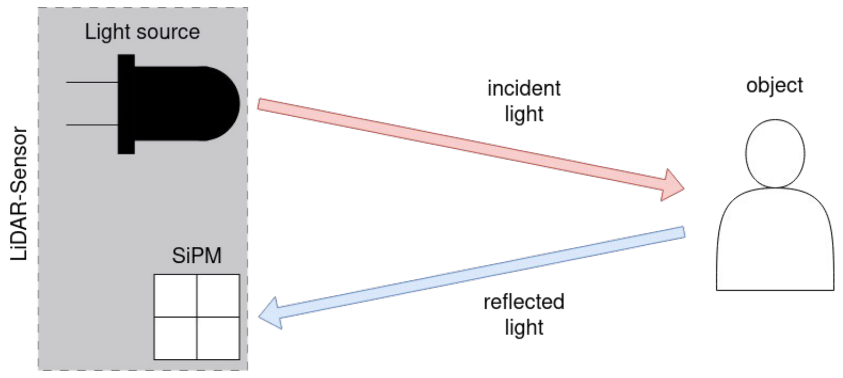


Figure 1.1: Exemplary application for an SiPM in a LiDAR sensor. By measuring the time it takes for the light to get to the SiPM from the source the distance to the object is determined.

Characterizing these SiPMs is conventionally done by illuminating them in a controlled environment with a set light source and extracting parameters from the output signal and the arising charge histogram [2]. By having a program that simulates the statistical nature, one can test and verify SiPMs, without needing physical components and expensive lab experiments. The SUM project (SiPM Unified Model) by a research group from the University of Hamburg's experimental physics department is dedicated to this task, by creating a new and extensive simulation framework. The framework itself can be understood as a reusable set of software libraries, intended to provide the most common components and be user-extendable. There are already numerous simulations available, with SUM being a new approach that aims to solve the most prominent problem: the computational load created by such a program is very high, effectively disabling the study of high light intensities since this seeds a long program runtime. The primary causes for this load are the complex calculations it takes to model the process of photon detection and related operations.

1.2 Thesis Goals

The main objective of this thesis is to develop a proof-of-concept for the new framework, that simulates SiPMs using the Julia programming language. By leveraging the capabilities of Julia, a high-performance and flexible programming language [3], the aim is to create a versatile simulation capable of accurately modeling the response of SiPMs across a wide range of operating conditions and configurations. Using a new language can impose problems regarding compatibility or stability, as well as performance. It is thus important to conduct a study before building a complete simulation. This is done by implementing efficient algorithms that factor in device properties and the statistical models used to describe those properties. More precisely, the thesis objective includes

1. developing a model for physical processes involved in the response of SiPMs to light, including the simulation of noise
2. generating the output signal while retaining full knowledge of the single discharges for every cell, what caused them, and when they occurred
3. integrating the response function to create a charge histogram
4. benchmarking the resulting simulation to access performance details and compare them to preceding research, determining the advances.

Chapter 2

Fundamentals of SiPMs and Parameter Extraction

2.1 Application

The primary task of SiPMs is detecting photons, which makes the aforementioned applications possible. Furthermore, since they combine electronics and optics, they are optoelectronic components, based solemnly on semiconductors. They consist of many single-photon avalanche diodes (SPADs) connected in parallel, with a cathode and anode serving as the common connectors. Each SPAD is called a pixel or microcell and has a quenching resistor (R_q) in series [2], as shown in Fig. 2.1.

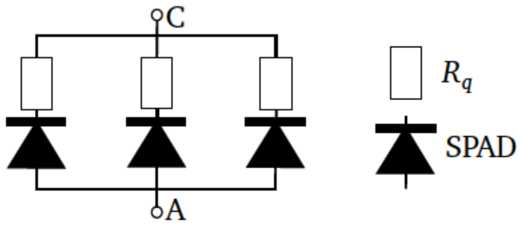


Figure 2.1: Equivalent circuit for an SiPM with three cells [4]

The applied voltage is above the reverse breakdown voltage of the SPAD, since this ensures an operation in Geiger mode. In this mode, a high electric field within the SPAD causes the acceleration of charge carriers after an electron-hole pair is created. This initial creation is caused by a photon that transfers its energy onto the electron, moving it to the conduction band; an electron avalanche is the result [5]. SiPMs possess exceptional versatility with a high sensitivity to single photons, a high dynamic range, and rapid timing properties [1].

2.2 Basis of Light Detection

2.2.1 Photons

A photon (denoted with γ) is the fundamental particle of electromagnetic radiation, possessing both wave-like and particle-like characteristics, also known as the wave-particle duality. It travels through space at the speed of light, carrying energy proportional to its frequency, and interacts with matter through absorption, emission, and scattering processes, exemplifying the quantized nature of electromagnetic fields. This quantization also leads to discrete levels of energy, which are defined as $E_\gamma = h \cdot \nu$, where h is Planck's constant and ν is the frequency [6]. Photons with a wavelength ($\lambda = \frac{c_0}{\nu}$) between 380nm and 750nm are more commonly referred to as light, with c_0 denoting the speed of light in a vacuum.

2.2.2 Photoelectric effect

The photoelectric effect is the extraction of electrons from a semiconductor or metal surface upon exposure to electromagnetic radiation, typically in the form of light (Fig. 2.2). The extraction is based upon the transfer of energy from an incident photon onto an electron within the material, with the threshold being the binding energy of said electron [6]. These released electrons are also referred to as photoelectrons.

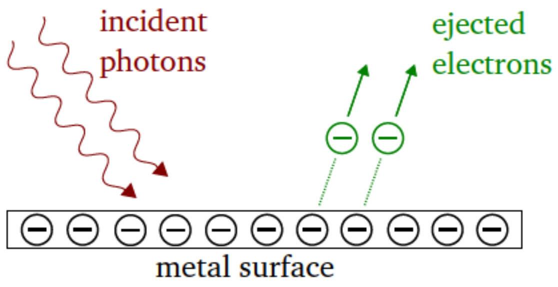


Figure 2.2: Visualization of the photoelectric effect

2.2.3 Photovoltaic effect

The photovoltaic effect, or inner photoelectric effect, is based on the premise that electrons can be ejected from their bonds within a crystal structure by the transfer of energy from a photon onto said electron. If this happens within the p-n junction of a diode sensitive to light (photodiode), an electron-hole pair is created. It then gets separated and drifts into the p-doped (hole) and n-doped (electron) parts. This creates an electric current [6], and the general idea is depicted in Fig. 2.3).

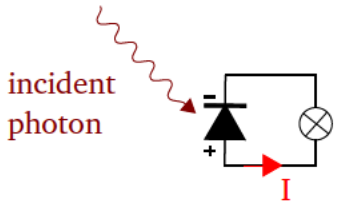


Figure 2.3: Visualization of the photovoltaic effect

2.3 Working principle of SiPMs

The physical realization of a SiPM is based on the principle of a p-n junction, with differently doped silicon profiles as seen in Fig. 2.4. Also observable are the anti-reflective SiO_2 coating and trenches that restrain photons from leaving a cell, as well as the avalanche region. The distribution of the electric field can be observed in Fig. 2.5.

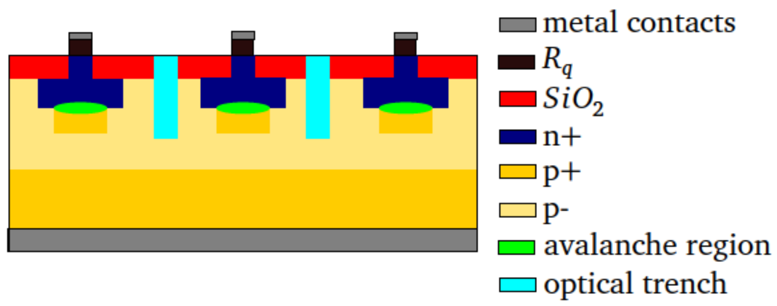


Figure 2.4: Structure of an SiPM with three cells [4]

If a photon that entered silicon transfers its energy onto an electron via absorption, the electron moves to the conduction band from the valence band it was previously in, as dictated by the photoelectric effect. It then becomes a photoelectron (e_{γ}^{-}) and leaves behind a hole in the lattice structure. By applying a negative bias voltage that is higher than the breakdown voltage, a strong electric field within the depletion region of the SiPM is created (Fig. 2.5). This accelerates the hole toward the cathode and the electron toward the anode. This acceleration adds to the kinetic energy of the electron, enabling it to ionize other electron-hole pairs by impact. This process is then reiterated with the newly freed electrons, effectively creating an avalanche, and the detection of one photon is thus amplified [7]. A graphical representation is given in Fig. 2.6.

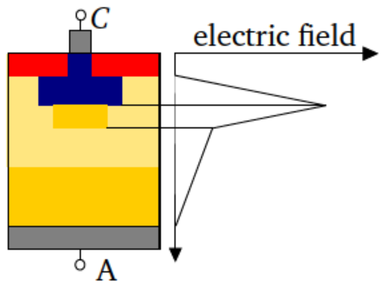


Figure 2.5: Visualization of the electric field within an SiPM. An explanation for the meaning of the colors can be found in Fig. 2.4 [8]

This closely resembles the detection of radiation with a Geiger counter, and this kind of discharge is thus more commonly known as a Geiger discharge. To stop the avalanche, a quenching resistor R_q has to be connected in series, reducing the voltage drop over the SiPM to be lower than the breakdown voltage, lowering the electric field, and ending the discharge [2]. If the quenching resistor were not present, the current flow would only be limited by the resistance of the wires, leading to a much higher current and possible damage to the materials.

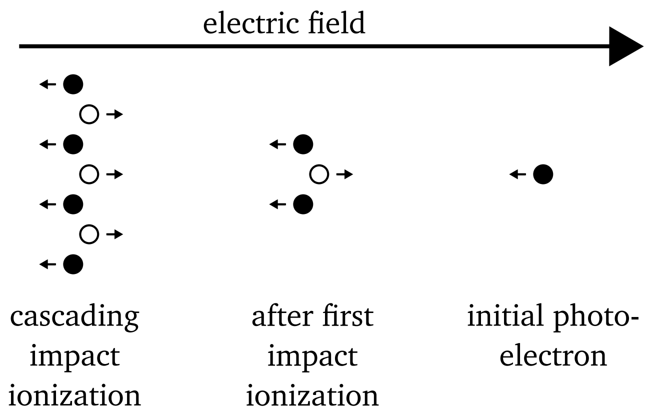


Figure 2.6: Depiction of the reiteration of an avalanche through impact ionization, after an initial electron-hole pair was created (displayed without the first hole). Black dots are electrons and white dots are holes [9].

Electrically this principle can be described by a capacitance C_d, which denotes the junction capacitance, in parallel with a resistor R_s and a switch. The capacitance is charged via the applied bias voltage and the quenching resistor, as seen in Fig. 2.7. The start of an avalanche by an impinging photon is modeled by closing the switch in the circuit, which leads to the discharge of the capacitor and thus a current flow through R_s. The capacitor then has to be recharged for the next detection. Until the recharge voltage reaches the breakdown voltage, a pixel cannot discharge again and is thus "blind" to impinging photons. This denotes the Single Photon Timing Resolution (SPTR) [8] and will be referred to as the blind time.

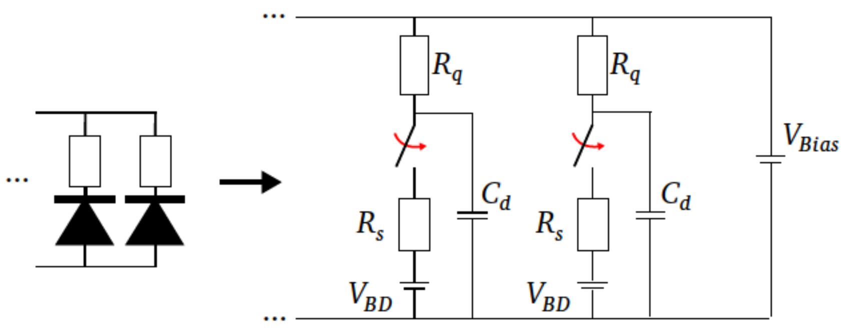


Figure 2.7: Equivalent circuit for one cell with the connected bias voltage and the internal breakdown voltage [10]

When N photons hit N pixels at the same, N times the charge is created and the signal scales linear. This can be seen as the sum of N discharges.

2.3.1 Response Signal

In the context of SiPMs, the Geiger discharge that follows the detection of a photon is more commonly referred to as a primary discharge, since this is the underlying principle for the detection of these and produces the desired signal. Furthermore, the charge created through the avalanche process is always the same for one detected photon. The flow of charge is known as current, and the height of an output pulse is always the same for one detected photon, resulting in discrete steps if multiple are detected at the same time [7]. Since all pixels have the same voltage applied, two (or more) photons hitting different cells at the same time each cause a discharge with the same height. The combined output signal is then an addition of these. The output pulse is depicted in Fig. 2.8, where the height is one in an arbitrary unit [a.u.] for one discharge and two for two discharges. This arbitrary unit depicts a gain of one and denotes the normalization of the resulting current pulse to a height of one.

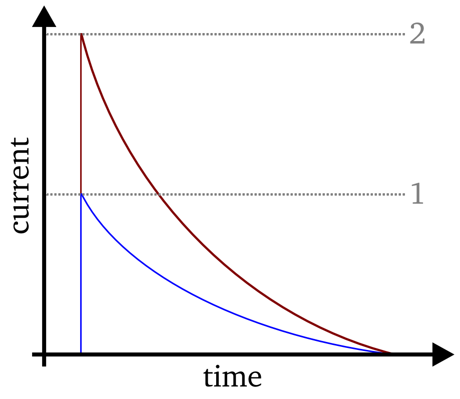


Figure 2.8: Current pulse for a Geiger discharge, with one detected photon (lower curve) and two instantaneously detected photons (upper curve)

The mean number of photons wished to be sent onto the SiPM for a simulation is smeared by a Poisson distribution to make up for the variation of the light source. The time of arrival on the other hand is represented by a Normal distribution, which depicts the sending of light from an LED [11].

2.3.2 Dark Counts

Silicon can eject electrons through the influence of thermal energy, which leads to the triggering of an avalanche without a photon being present [1]. This is known as a dark count, and it is highly dependent on the temperature. The dark counts are normally depicted by the aperiodic specification of the rate $f_{DCR} = \frac{counts}{s}$ and referred to as the Dark Count Rate (or DCR for short). A discharge from DCR is indistinguishable from a photon-induced discharge, which leads to both being primary discharges and the pulse form for DCR is the same as the one for a Geiger discharge from light, observable in Fig. 2.9. It is noted, that for this instance all pixels contribute to the DCR, and a distinction between different contributions of cells to the overall DCR is not considered.

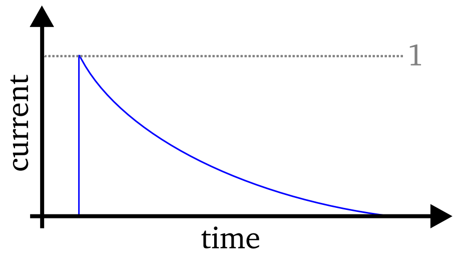


Figure 2.9: Current pulse for a DCR discharge

To calculate the number of DCR-induced discharges that happened during the simulation, first, f_{DCR} is multiplied by the observed time (referenced in Chapter 3.3.1), and the obtained number is fed into a Poisson distribution to account for fluctuations. Furthermore, DCR can occur at any time and thus a Uniform distribution over the observed time covers the time of occurrence best [11].

2.3.3 Crosstalk

During the avalanche multiplication, photons can be emitted from accelerated electrons in the high field region (Fig. 2.10), which in return cause avalanches in neighboring cells or the inactive region of the same cell [1]. There are two different types of this optical effect taken into consideration, namely prompt and delayed crosstalk. The keywords prompt and delayed declare the timing difference of the effect since prompt crosstalk happens nearly instantaneously and is very close in time to the original discharge. For this, the photon hits directly in the depletion region and starts an avalanche, while for the delayed crosstalk a photon impinges into the non-depleted area. From there it has to diffuse into the multiplication area, which causes a timely delay. Since for both types, the photon does not leave the SiPM, they are differentiated from external crosstalk, which happens when the emitted photon exits the component and gets reflected by surrounding materials (e.g. a protective window) [1].

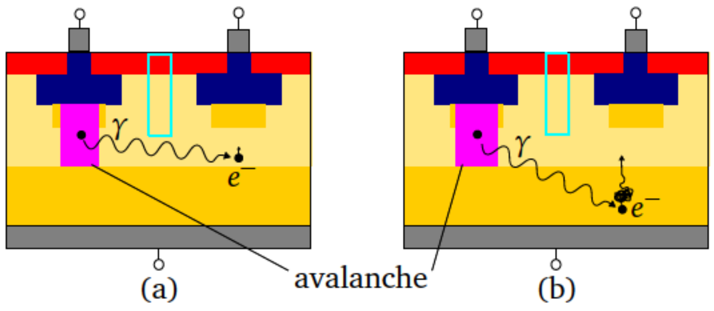


Figure 2.10: Overview of the different causes for crosstalk, (a) depicts prompt and (b) delayed crosstalk if no trenches to block photons were present [4]. The legend can be found in Fig. 2.4.

The combined height of a discharge that causes prompt crosstalk is the initial discharge height of the parent, plus one complete discharge with the height of 1 in [a.u.]. This can be observed in Fig. 2.11 (a). Graph (b) of the same figure shows the height of a single pulse with delayed crosstalk, where the total height is reduced by the subsiding of the initial discharge.

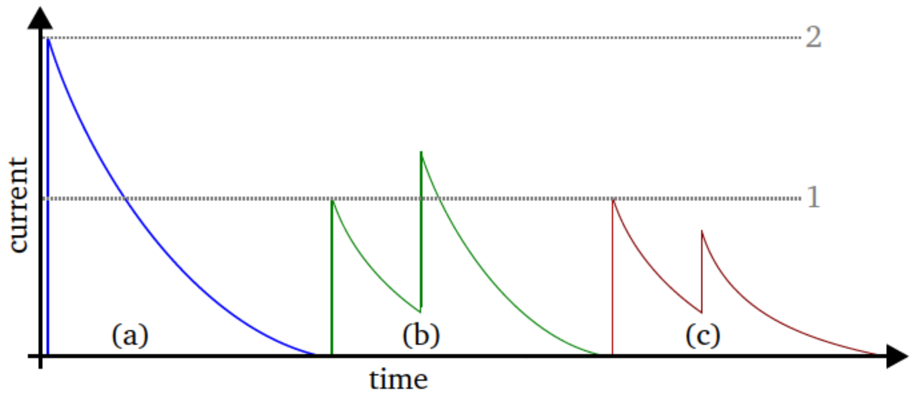


Figure 2.11: Current pulse for a Geiger discharge with (a) prompt and (b) delayed crosstalk, as well as (c) afterpulse

For the simulation only prompt (PXT) and delayed crosstalk (DXT) are considered, each with a unique probability (P_{PXT} and P_{DXT}), which enables the study of both effects independently. To determine the sum of all crosstalk incidents, a Binomial distribution for each effect is assumed [11], with the number of trials being the number of discharges that happened before and the specific probability being the success rate. The Binomial distribution directly gives the number of occurred crosstalk, instead of having to loop over all Geiger discharges for this determination, speeding up the process.

For the delayed crosstalk an accompanying Exponential distribution is introduced to represent the time delay [11]. This model is chosen since it is more plausible that the delayed crosstalk happens shortly after the discharge causing it, but it could occur later on, due to the diffusion. Prompt crosstalk occurs at the same time as the parent discharge and thus has no time delay.

2.3.4 Afterpulses

Producing pure silicon is hard to achieve, and defects in the crystal structure are very common. These defects can trap electrons during the avalanche multiplication, which are then released during the recharge phase of the SPAD. This observable effect is called afterpulsing (AP), a type of correlated noise with an amplitude lower than the preceding primary discharge. The probability for an afterpulse to occur (P_{AP}) is dependent on the number of defects with their related release time and the recharge time [1]. In order to calculate the total number of afterpulses per event, again a Binomial distribution is assumed over all previous discharges [11], as well as an Exponential distribution for the time of occurrence [2]. This Exponential release best matches the characteristics of the release time.

The height of the afterpulse is limited by the recharging of the cell after an initial discharge, and since afterpulsing happens in the same cell as the parent discharge, the amplitude is lower than the original, as seen in Fig. 2.11 (c).

2.3.5 Summary of effects

A summary of the effects that are implemented in the simulation can be found in table 2.1. Given is the reason for each occurred effect, together with their respective distributions for the total number and their timestamp.

	Signal	Noise			
Effect	e_{γ}^{-}	DCR	PXT	DXT	AP
Reason	Detected light	Thermal Energy	Secondary photons	Secondary photons	Trapped electrons
Distribution (Number)	Poisson	Poisson	Binomial	Binomial	Binomial
Distribution (Time)	Normal	Uniform	-	Exponential	Exponential

Table 2.1: Extract from table A.1 to summarize the primary and secondary effects used in the simulation

All of the mentioned distributions are the starting point and reflect the current position of research. If future studies figure out different representations for these physical effects, they can be easily adapted with little changes to the code.

2.3.6 Further Parameters

The following parameters are usually used to describe an SiPM but are left out (or set to 1) in the scope of this thesis [7] [8] [12].

- Quantum Efficiency $\eta(\lambda)$
  The quantum efficiency is the probability that a photon creates an electron-hole pair within the depletion region of the SiPM. Since the penetration depth of an electromagnetic wave - and thus a photon - is relative to its frequency, η can be expressed as a function of the wavelength λ. The assumed value is $\eta(\lambda) = 1.00$.

- Avalanche Initiation Probability $\epsilon(V)$
  $\epsilon(V)$ is the probability that a Geiger discharge occurs in response to the creation of an electron-hole-pair. Since a stronger electric field leads to a higher avalanche probability this parameter hinges on the applied voltage. The assumed value is $\epsilon(V) = 1.00$.

- Photon Detection Efficiency PDE(λ, V)
  The probability of a photon hitting the SiPM and being detected. This

parameter depends on the quantum efficiency and the avalanche initiation probability, as well as the active-to-inactive area, or fill factor, F. $PDE(\lambda, V) = \eta(\lambda) \cdot \epsilon(V) \cdot F$. The assumed value is $PDE(\lambda, V) = 1.00$.

- Bias Voltage V_{bd}
  For low reverse bias voltages, the p-n junction blocks most of the current flow, and only a small leakage current occurs. If the voltage reaches the breakdown voltage, a Geiger discharge occurs and the current rises steeply. This parameter is mentioned before, but not implemented.

- Turnoff Voltage V_{off}
  The reverse bias voltage at which the Geiger discharge comes to a standstill. When the current rises, the voltage drop over the quenching resistor also rises, leaving less voltage for the pixel. The current is then limited and the discharge ends.

- Gain G
  The gain is a measurement of how much charge is generated by a single photon, effectively indicating the amplification. It is dependent on the internal capacitance and the overvoltage.

Chapter 3

Simulation Methodology

The developed code is available on a CD and can be examined by contacting the first supervisor ("Erstgutachter") of this thesis.

3.1 Preceding research

Foregoing research for an SiPM simulation was already conducted by Jack Rolph with help from the research group, and he created the program "lightsimtastic". A link to the repository can be found in [13]. Lightsimtastic is written in Python and serves as a comparing tool for an existing implementation with differences arising not only from the code side of view but also from the usability. Furthermore, this thesis tries to build a combined simulation for low-light and high-light intensities to properly handle saturation effects, which lightsimtastic is not capable of. On top of that, Julia instead of Python is used to speed up calculations and a comparison between both programs is conducted in Chapter 5.3.

3.2 Overview of the Julia Programming Language

Choosing a programming language can significantly impact the efficiency and effectiveness of data analysis and computational tasks. Julia and Python are two prominent languages used in scientific computing, offering distinct advantages and drawbacks. Table 3.1 offers a comparison between the features of interest.

Feature	Julia	Python
Speed	Fast and close to C (JIT compilation)	Slower (interpreted language)
Libraries	Growing ecosystem (10,000+ packages)	Extensive collection (137,000+ packages)
Computing Requisition	Excellent for numerical/ scientific computing (efficient machine code)	Slower for heavy computations (unefficient line-by-line translation)
Composability	Very good	Very good
Use Cases	Machine learning and scientific computing	General-purpose (web development, machine learning, automation etc.)

Table 3.1: Comparison between Julia and Python [3] [14]

Despite the drawback of having a limited number of available packages, the advantages of Julia's speed and its suitability for scientific computation make it an increasingly attractive option for a wide range of applications [3]. Since it is fairly easy to learn and has a clear syntax, it serves as the language of choice for this thesis.

3.3 Implementation

The goal of this thesis is to simulate the behavior of SiPMs for primary and secondary discharges, which is done by taking known physical properties and developing new algorithms that model them. Additional functionalities, which display the output signal and its respective spectrum are also implemented. The steps it takes and the novel ideas distinguishing this approach from the preceding implementation are given in the order they are processed in the simulation. These steps can be observed in Fig. 3.1 and they are further broken down to explain the structural process. The functional dependencies of the implemented methods can be found in the appendix in Fig. A.1. The resulting folder and file structure are depicted in Fig. A.2 and Fig. A.3 respectively. An example file included in the framework provides a quick setup for new users.

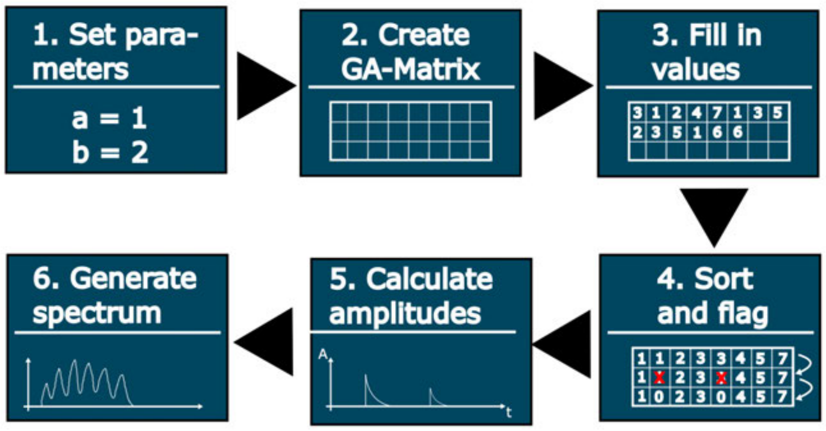


Figure 3.1: Structural overview of the simulation

In the real world, the output signal is created by shooting pulsed light at the SiPM in a controlled environment. In this simulation, the SiPM is described by an NxM matrix of adjacent cells with progressing numbering, creating a unique ID for each cell. This allows the tracking of discharges over all cells, a distinctive feature. An example of a simple 3x3 matrix is given in Fig. 3.2

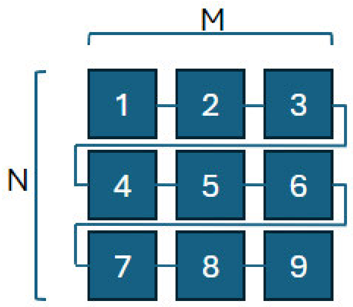


Figure 3.2: Structural representation of the SIPM within the simulation. Each number is a unique cell ID. The connection between the cells represents the parallel connection from the circuit, as introduced before.

Each arriving pulse is called an event and for each simulated one, photon-induced and resulting secondary discharges are created, along with dark counts. All discharges from one event are stored in a Geiger Array (GA), an array of a custom struct, which holds for each discharge the cell ID, the timestamp, the source, as well as the amplitude (Fig. 3.3 (a)). Since normally more than one event is simulated, all arrays are combined to form the GA-matrix, as seen in Fig. 3.3 (b). This effectively stores all data in one feasible object for easier data handling, retaining all information in the memory during the runtime.

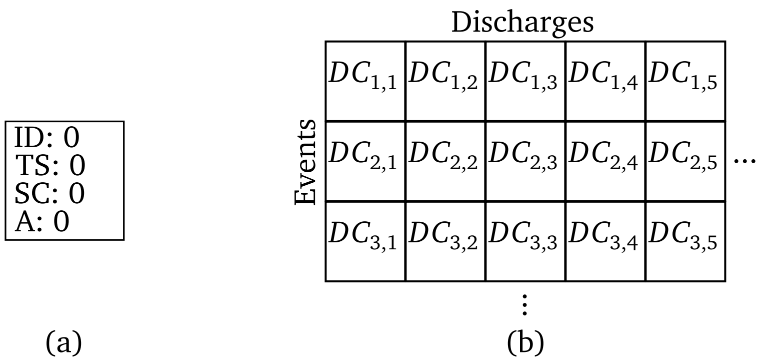

ID: 0
TS: 0
SC: 0
A: 0
(a)
Discharges
Events
$DC_{1,1}$ $DC_{1,2}$ $DC_{1,3}$ $DC_{1,4}$ $DC_{1,5}$
$DC_{2,1}$ $DC_{2,2}$ $DC_{2,3}$ $DC_{2,4}$ $DC_{2,5}$ …
$DC_{3,1}$ $DC_{3,2}$ $DC_{3,3}$ $DC_{3,4}$ $DC_{3,5}$
⋮
(b)

Figure 3.3: Visual representation of the internal handling of discharges with (a) a single entry of the custom struct and (b) the resulting GA-matrix. DC is an abbreviation for discharge, with the indices denoting the event and discharge numbers.

3.3.1 Setting parameters

Setting necessary parameters is done by assigning values to variables before functions are called and a table holding all initial values is provided in the appendix (A.1). Afterward, the program code is executed, which can be seen in Fig. 3.4

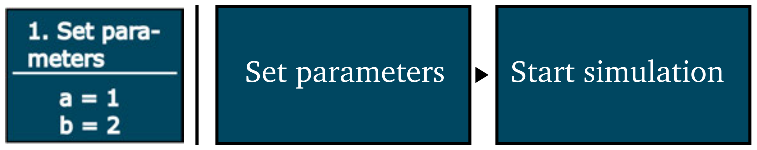


Figure 3.4: Overview of the steps for the initialization

The simulation is based on a real-life timescale to account for the time constraints set by the copied physical processes. An explanation of this timescale can be found in Fig. 3.5. The primarily used points in time are the simulation start and rising signal, as well as the integration start and length; the integration is explained in Section 3.3.6, where the output signal is integrated to gain the charge. The rising signal timestamp denotes the arrival of photons on the SiPM and serves as a point of reference to the simulation start and integration start. To integrate over the complete pulse and start the simulation before, both times have a negative sign. The integration end is calculated by adding the integration length to the integration start and is thus positive. The overall observed time starts at the simulation start and ends with the integration end, a time difference used to calculate the number of occurred DCR discharges.

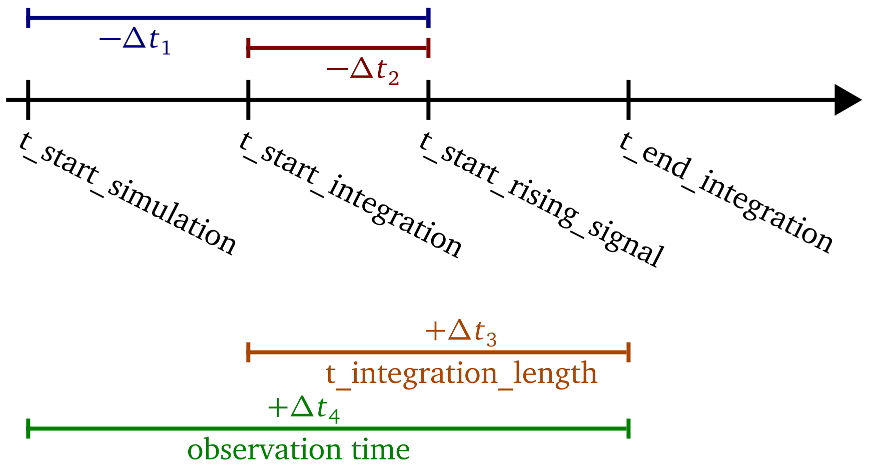


Figure 3.5: Explanation of the timescale used for the simulation, adapted from [13]

3.3.2 Creating the GA-matrix

Once the simulation is running, it is necessary to calculate the size of the resulting GA-matrix and initialize it without any discharges. For this, the number of photo-electrons from photons and DCR is calculated. For the photons a mean is smeared with a Poisson distribution and randomly sampled for each event, whereas the mean for the DCR is obtained by multiplying the observed time with the respective rate; the resulting number is also smeared with a Poisson distribution and randomly sampled for each event. To acquire the number of prompt and delayed crosstalk, as well as afterpulses, the preceding primary discharges are used as the number of trials for three Binomial distributions, with the rate of success being the probability for each effect to occur. It is also possible to calculate the secondaries of secondaries, providing more accurate calculations and a unique feature.

For each event, the total number of discharges is summed up and an empty matrix with the dimensions *number of events* by *maximum of all accumulated discharges from every event* is initialized. This procedure is displayed in Fig. 3.6. Each entry is one struct (Fig. 3.2) creating a three-dimensional matrix.

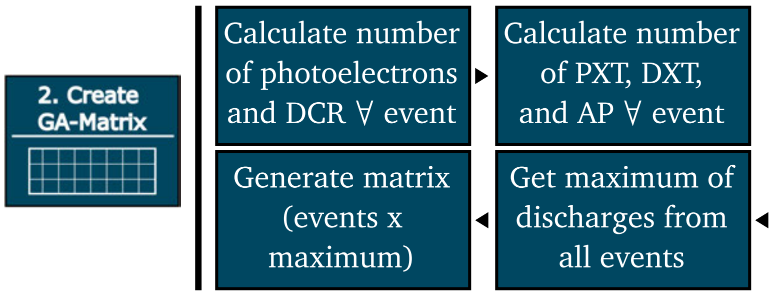

2. Create GA-Matrix
Calculate number of photoelectrons and DCR ∀ event
Calculate number of PXT, DXT, and AP ∀ event
Get maximum of discharges from all events
Generate matrix (events x maximum)

Figure 3.6: Steps that lead to the creation of the matrix

A numerical example, as depicted in figure 3.7 and the subsequent figures, has five total discharges. The ID is the unique cell ID in which the respective discharge happened, TS denotes the time of occurrence, and SC is an abbreviation for the source of the discharge. The letter A represents the amplitude of the discharge, which is calculated at a later step.

Discharge	#1	#2	#3	#4	#5
Matrix	ID: 0 TS: 0 SC: 0 A: 0	ID: 0 TS: 0 SC: 0 A: 0	ID: 0 TS: 0 SC: 0 A: 0	ID: 0 TS: 0 SC: 0 A: 0	ID: 0 TS: 0 SC: 0 A: 0

Figure 3.7: Depiction of the empty matrix for one event with five cumulated discharges. ID is the unique number of the fired cell, TS is the timestamp of occurrence (currently in nanoseconds), SC is the source, and A the amplitude

3.3.3 Matrix filling

After initializing, the matrix needs to be filled with actual discharges, which is done for each effect separately and can be observed in Fig. 3.8, which depicts the steps leading to a filled matrix. Having this matrix provides a novel implementation of the Geiger-Array, keeping the information of already calculated events available. Furthermore, parallel threads are used to speed up loops.

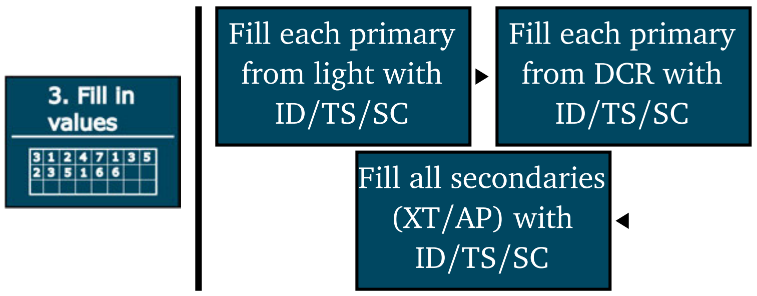

3. Fill in values
Fill each primary from light with ID/TS/SC
Fill each primary from DCR with ID/TS/SC
Fill all secondaries (XT/AP) with ID/TS/SC

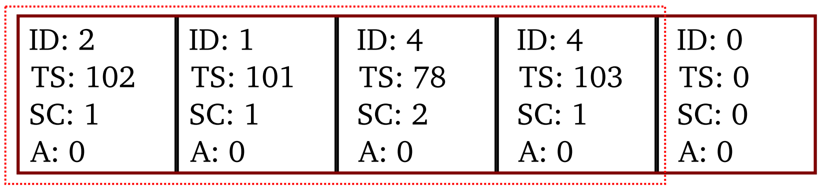
Figure 3.8: General overview of filling the matrix

Since it is already known how many discharges of each effect occurred, the program loops over all of them, starting with the primaries. For each discharge, a random ID is selected, as well as a random timestamp from the respective distribution. Furthermore, the source is saved alongside the other information and an overview of the numeral representations is depicted in table 3.2.

Discharge		Source
Initialization		0
Primary	Photon	1
	DCR	2
Secondary	PXT	3
	DXT	4
	AP	5

Table 3.2: Overview of the possible numeral representation for the source of a discharge

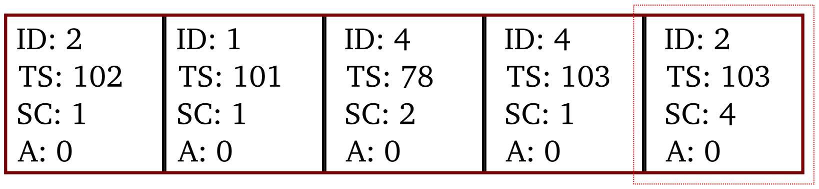
The resulting matrix after the primary effects are filled in, is displayed in Fig. 3.9. This example has five total discharges with three being primaries from light, one primary from DCR, and one secondary from delayed crosstalk.



ID: 2	ID: 1	ID: 4	ID: 4	ID: 0
TS: 102	TS: 101	TS: 78	TS: 103	TS: 0
SC: 1	SC: 1	SC: 2	SC: 1	SC: 0
A: 0	A: 0	A: 0	A: 0	A: 0

Figure 3.9: Visualization of the matrix after filling in four primary discharges, extending Fig. 3.7

Once all the primary discharges are saved, the necessary data for the secondary discharges are provided by randomly selecting the ID of a discharge that already happened, providing the neighboring cells (for crosstalk), randomly selecting a neighbor, and saving this information in the matrix. For delayed crosstalk and afterpulses, an Exponential distribution for the timestamp is introduced, which denotes the delayed nature of both effects. The completely filled matrix can be observed in Fig. 3.10.



ID: 2	ID: 1	ID: 4	ID: 4	ID: 2
TS: 102	TS: 101	TS: 78	TS: 103	TS: 103
SC: 1	SC: 1	SC: 2	SC: 1	SC: 4
A: 0	A: 0	A: 0	A: 0	A: 0

Figure 3.10: Visualization of the matrix after filling in the remaining secondary discharge in the fifth entry of the matrix, extending Fig. 3.9 and Fig. 3.7

3.3.3.1 Obtaining neighboring cells

Concerning the size of the simulated SiPM, the neighbors of a cell are calculated by determining the position of that specific cell and assessing the cells above, below, left, and right. The outcome of this algorithm is visualized by an example in Fig. 3.11.

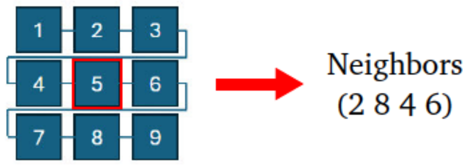


Figure 3.11: Exemplary neighbors of cell no. 5 in a 3x3 formation. Cells 2, 8, 4, and 6 are returned as neighbors

Important to note is, that border elements also have four neighbors, but instead of the cell ID (when there is no neighbor) zero is saved. If the rand()-function selects a discharge with zero as the cell ID for crosstalk, the information about the potential secondary effect is stored, but omitted when amplitudes and the spectrum are calculated. This is to track crosstalk that exits the SiPM and cannot enter again.

3.3.4 Sorting and flagging

Since each cell that is fired possesses a blind time during which it can not be fired again, and random timestamps are generated, it is necessary to flag every implausible discharge. First, all discharges are sorted with a custom sorting function, which reorders each Geiger-Array by (a) the cell ID and (b) the timestamps of each ID. Introducing a custom sorting function is necessary since the array-to-sort is also made from a custom struct, which is unique to this simulation. The sorting process guarantees the timely dependency of each cell is kept in order. Afterward, each source of a discharge that occurred within the specified blind time from the previous discharge is reset to "0", effectively flagging all implausible discharges. A brief overview is given in Fig. 3.12.

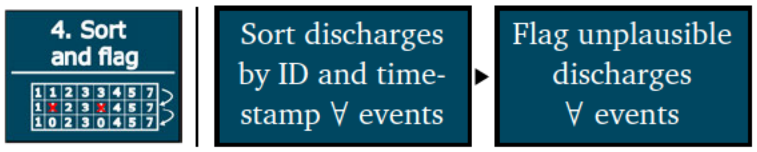
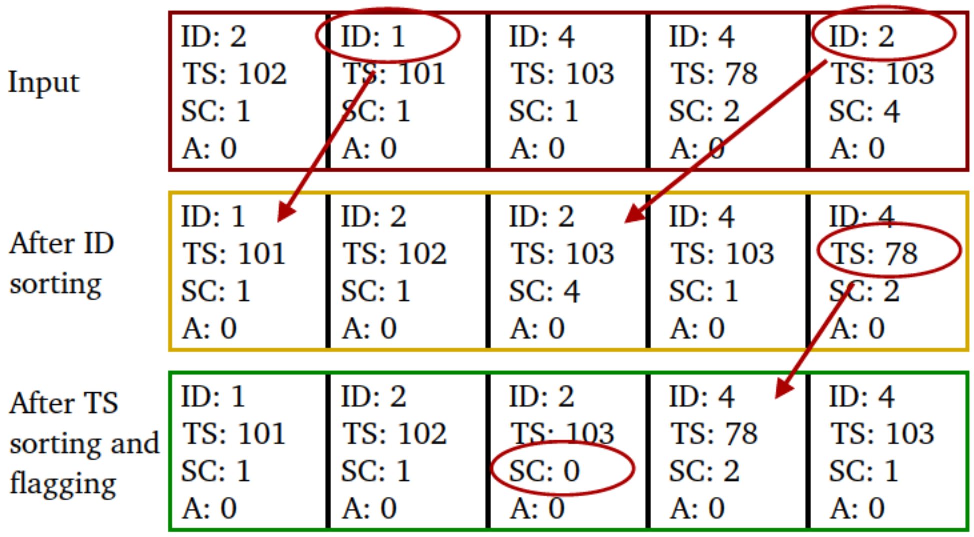

4. Sort and flag
Sort discharges by ID and time-stamp ∀ events
Flag unplausible discharges ∀ events

Figure 3.12: Overview of the computational steps for sorting and flagging the matrix after filling

Respectively, the algorithm steps are more accurately depicted in Fig. 3.13.



Input	ID: 2 TS: 102 SC: 1 A: 0	ID: 1 TS: 101 SC: 1 A: 0	ID: 4 TS: 103 SC: 1 A: 0	ID: 4 TS: 78 SC: 2 A: 0	ID: 2 TS: 103 SC: 4 A: 0
After ID sorting	ID: 1 TS: 101 SC: 1 A: 0	ID: 2 TS: 102 SC: 1 A: 0	ID: 2 TS: 103 SC: 4 A: 0	ID: 4 TS: 103 SC: 1 A: 0	ID: 4 TS: 78 SC: 2 A: 0
After TS sorting and flagging	ID: 1 TS: 101 SC: 1 A: 0	ID: 2 TS: 102 SC: 1 A: 0	ID: 2 TS: 103 SC: 0 A: 0	ID: 4 TS: 78 SC: 2 A: 0	ID: 4 TS: 103 SC: 1 A: 0

Figure 3.13: Matrix after sorting and flagging (with 2ns blind time). ID is the unique number of the fired cell, TS is the timestamp of occurrence, SC is the source, and A the amplitude.

Conceptually this sorting and flagging algorithm holds the potential for a simplification, where the implausible discharges are directly omitted without saving them in the matrix. By doing this, the simulation would lose information, and the crosstalk that exits the SiPM would not be traceable, leading to a bias in the possibility of crosstalk. Furthermore, the complexity of the algorithm would increase by adding code that tracks where each new discharge should be saved.

3.3.5 Calculating amplitudes

At this point all discharges are available and the actual amplitudes can be calculated, which is done for each event separately and again uses parallelization. The algorithm loops through all discharges with an ID greater than zero, setting the amplitude to $A_i = 1.0$, where the index i denotes the discharge, starting at one. This process is displayed in Fig. 3.14.

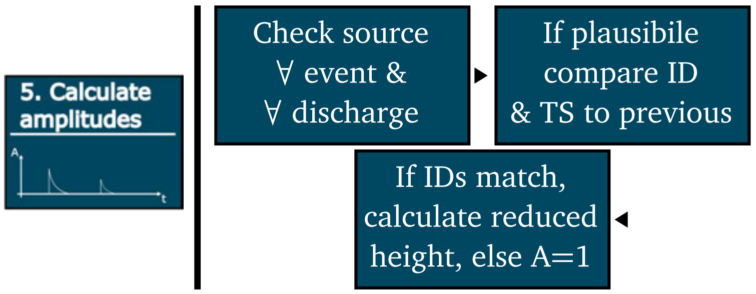

5. Calculate amplitudes
A
t
Check source ∀ event & ∀ discharge
If plausibile compare ID & TS to previous
If IDs match, calculate reduced height, else A=1

Figure 3.14: General overview of calculating the amplitudes for each discharge

If the considered discharge happened in a cell that has been triggered before, the amplitude is reduced by considering the recharge time, which is adapted from [5] and given by

$$A_i = 1 - e^{-\Delta t/\tau_s} \tag{3.1}$$

It is derived from the recharging of the capacitance after discharging. Δt is the time difference between the last timestamp and the current timestamp, and τ_s is a time constant. Usually, two different effects influence the recharging, which are the recharging current of the pixel (slow component, τ_s) and the capacitive coupling of the quenching resistor (fast component, τ_f). An additional proportion factor r_f denotes the share of the fast component in the overall recharge function [1] [11]. For a simple recharge function, the separation is not necessary, but the waveform (or transient)

$$I_1(t) = \left(\frac{1-r_f}{\tau_s} \cdot e^{-t/\tau_s} + \frac{r_f}{\tau_f} \cdot e^{-t/\tau_f} \right) \cdot \Theta(t) \tag{3.2}$$

takes this into consideration (adapted from [11]), with $\Theta(t)$ denoting the Heaviside step function. This equation is used to display the discharges with their respective height and to obtain the charge spectrum. Since this is additional information, the step itself is satisfied by the calculation of amplitudes. Once the height is determined, the amplitude is stored in the matrix. When displaying the amplitudes, t is shifted exactly by the time of occurrence, resulting in

$$I'(t) = I_1(t - t_{s,i}) \tag{3.3}$$

where $t_{s,i}$ is the respective timestamp [11].

3.3.6 Generating the spectrum

To conclude the functionality of the framework and determine the spectrum, all discharges are integrated in the specified integration times. It is noted, that 'spectrum' relates to the displaying of how often a specific charge occurred. Fig. 3.15 shows the process of generating this spectrum.

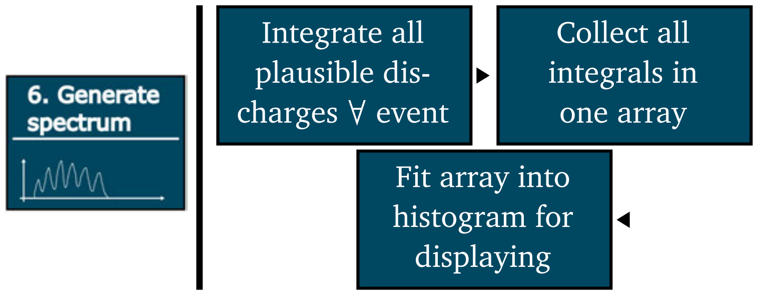

6. Generate spectrum
Integrate all plausible discharges ∀ event
Collect all integrals in one array
Fit array into histogram for displaying

Figure 3.15: General overview of the spectrum generation

The output signal provided with the waveform (equation 3.2) for each discharge is a current pulse and the charge of a current can be obtained by

$$Q(t) = \int I(t)\,dt \tag{3.4}$$

For one discharge the equation is slightly altered to account for the amplitude, the time of occurrence, and the integration time, which resides in

$$Q_i = A_i \cdot \int_{t_0}^{t_{end}} I(t - t_{s,i})\,dt \tag{3.5}$$

and can be gleaned in [11]. Summing the charges of all current pulses from the discharges within the specified integration time for one event gives the total charge Q' that happened during that event. Collecting the values for Q' for all events in a histogram produces the simulated charge spectrum.

Chapter 4

Simulation Results

4.1 Validation and Verification

To ensure the accuracy and reliability of the simulation, it is crucial to compare the results of each task that is performed with expected values.

4.1.1 Unit tests

To test the functions, multiple test files are included in the framework, which incorporates this functionality. Tests are performed for each function, using the @test macro from the Test.jl package. This macro evaluates an expression (e.g. $2 + 2 == 4$) and returns whether this test condition is true. For example, a test for the function that sorts and flags unplausible discharges includes checking if an event is sorted correctly and if sources are flagged if necessary. Other function checks include the creation of the matrix itself, checking for primary and secondary discharges, as well as the return of the neighbors to a provided ID.

The development of these tests is done systematically. First, test scenarios are created, effectively simulating user input for normal use cases. Additionally, worst-case and error scenarios are implemented to test the boundaries of the simulation. For each trial the expected results are assessed and tested against the real function output, and, if necessary, the function is altered to disclose the expected behavior. A complete list of the tested functions can be found in the appendix in table A.2, where it is noted that only necessary functions are tested and marked with "Tested".

To test the physical accuracy, visual assessments were done. In these the produced GA matrix was meticulously investigated by hand, going through every discharge in a dual-control manner. A visual representation of the variation of the cell IDs can be found in Fig. A.4. Furthermore, a separate function is implemented, that takes all testable variables as input and applies a plausibility check. This can already detect some errors but does not hinder the user from inputting wrong values.

4.1.2 Distributions

In order to prove the mathematical accuracy of the used distributions, testing functions are created, which take the distribution as input and either extract samples from it or, if provided, use the values created during the simulation. To gain a better insight, the moments as depicted in table 4.1, are calculated, as well as the Probability Density Function or PDF. The PDF is further referenced as "Line Plot".

Distribution	Mean μ	Variance σ^2
Normal(μ, σ^2)	μ	σ^2
Poisson(λ)	λ	λ
Exponential(λ)	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Uniform(a, b)	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Binomial(n, p)	$n \cdot p$	$n \cdot p \cdot (1-p)$

Standard Error	$\frac{\sqrt{\sigma^2}}{\sqrt{n}}$	$\sigma^2 \cdot \sqrt{\frac{2}{n-1}}$

Table 4.1: Mean and variance for each implemented distribution. The calculation of the standard error (SE) is adapted from [15]

An example of the Exponential distribution is given in Fig. 4.1 and the other distributions can be found in the appendix (A.4). Since these tests are done while running the simulation, they are implemented as a debug feature in the framework and can be switched off if desired.

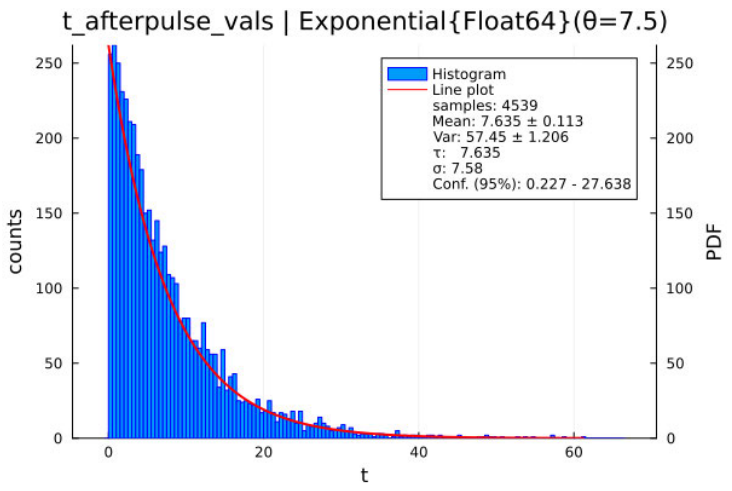


Figure 4.1: Exemplary plot for the distribution of timestamps from afterpulses with an exponential scattering. The "vals" in the title denotes the usage of values generated during a simulation as opposed to external sampling.

4.2 Overview of the Simulated Data

4.2.1 Output signal

Implemented is an option to plot every amplitude that was calculated for a simulation, which equals the output signal of the SiPM. To display the general idea, one event is simulated with the settings found in Fig. A.1. The gained knowledge can be transferred to the simulation of more events, but it is then harder to distinguish all discharges and break down the output signal. The general pulse form for each discharge can be found in Fig. 4.6.

Fig. 4.2 (a) shows the simplest SiPM layout possible, consisting of one single cell. This simplification is done to show the behavior of each cell for multiple discharges. Larger arrays have the same behavior for each cell, except for added crosstalk, which can not occur in a single-cell array since there are no other cells that photons produced during the avalanche can travel to. Going from left to right, the reasons for the discharges are DCR, photon, AP, and DCR. In one cell the maximum amplitude can never exceed 1 [a.u.]. Moving one step further, Fig. 4.2 (b) demonstrates the behavior for four cells, with the reasons for the discharges being DCR, AP, two photons and PXT, DCR and

PXT, and DCR. Noticeable is the change in the maximum height, which is due to the overlay of multiple cells firing at the same time. This is observable at 100ns, where two photons hit the SiPM at the same time and one discharge also causes prompt crosstalk.

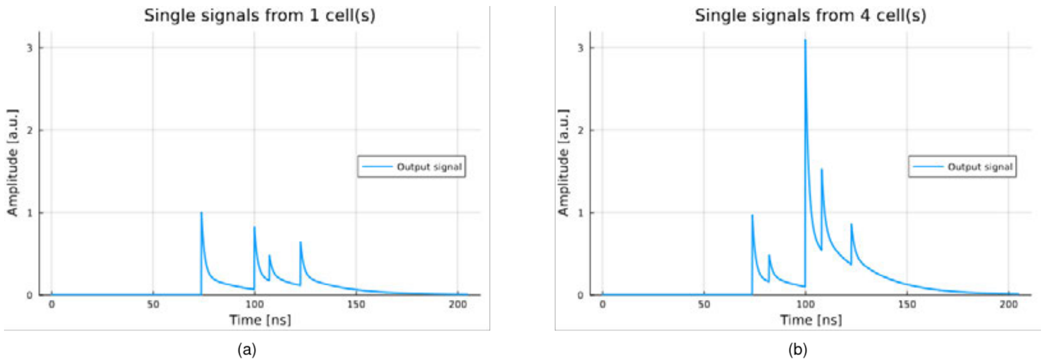


Figure 4.2: Depiction of the output signal from (a) one cell and (b) four cells

In Fig. 4.3 the process of amplitude calculation in accordance with the recharging of the cell can be observed. The orange line depicts the recharging (in percent) from the time of discharge until the theoretical end of the simulation. The blue line is again the complete output signal, which is built from two discharges. The first, with a height of 1.0 is from DCR, and the second is from a photon, but with a lowered amplitude. In this example, the capacitance has been charged to 72.4% and the second discharge is added on top.

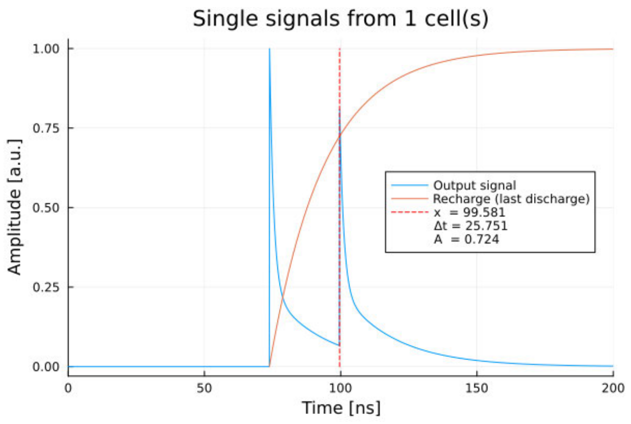


Figure 4.3: Example discharges from one cell in combination with the recharging of the cell. The height is adjusted to an amplitude of 1 a.u.

4.2.2 Spectra

The final implemented functionality is the calculation of the charge spectrum of an SiPM, which depicts the amount of charge against their respective number of occurrences. To generate suitable spectra, some variables were altered and the comparison of spectra is done with a separately implemented function that only has this task. The spectrum itself is the accumulation of measured charge per event and the sorting into bins; it is thus a histogram. To get the necessary information, each current pulse is integrated with the respective integration time and the charges for each event are summed up, as explained in section 3.3.6. An example of a spectrum without any effects, except for arriving photons, is depicted by the blue graph in Fig. 4.4. The resulting spectrum resembles the initial Poisson distribution.

The next step is to gradually add effects other than light and determine the correctness. The orange line plot in Fig. 4.4 is showing the initial plot with added DCR.

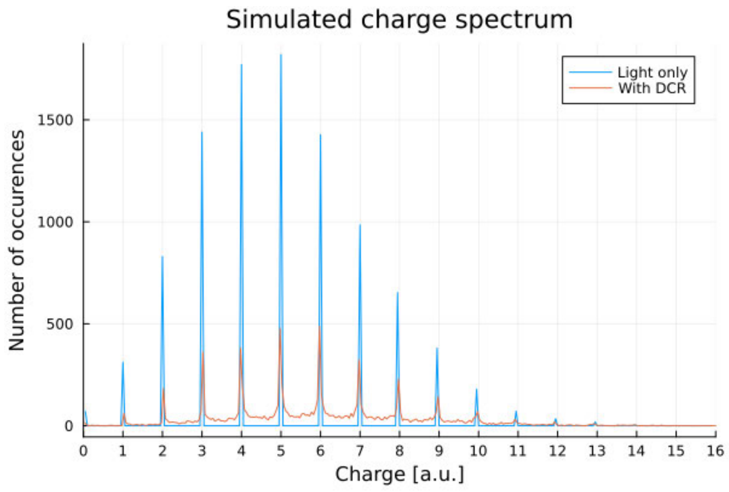


Figure 4.4: Depiction of a charge spectrum that has only primary discharges from light (blue graph) and light combined with DCR (orange graph)

Instead of the integrated output pulses, it is also possible to display the discharges that happened during each event, making it easier to verify the results. A spectrum featuring this context is depicted in Fig. 4.5.

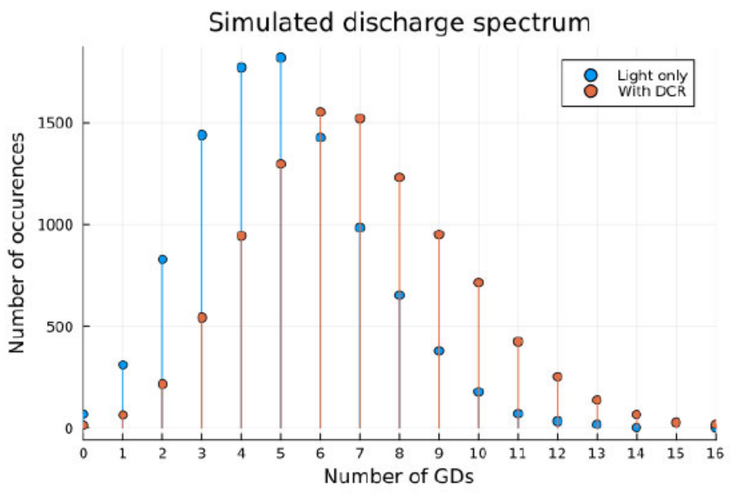


Figure 4.5: Depiction of a discharge spectrum that has only primary discharges (light and DCR). As opposed to fig. 4.4, the total number of occurred Geiger discharges per event are tracked instead of the amount of charge.

It is important to note, that the mean and variance of the combined distribution can be calculated by adding the mean and the variance of each distribution (light and DCR; both Poisson) together. The RMS (Root-Mean-Square) on the other hand is calculated by (adapted from [16])

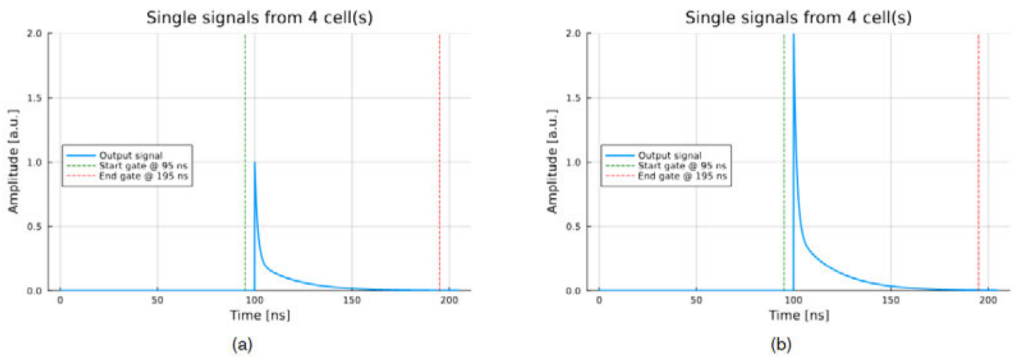
$$RMS_{total} = \sqrt{(RMS_1)^2 + (RMS_2)^2} \tag{4.1}$$

The calculated results are presented in table 4.2 with a noticeable closeness of the calculated data and the expected results. Using the charge spectrum instead of the discharge spectrum is error-prone since it happens that the discharges from DCR are not completely integrated due to the uniform timestamp distribution, leading to wrong results.

Moment	DCR Poisson(1.95)	Light Poisson(5)	Result (expected)	Result (from data)
Mean	1.940	4.992	6.932	6.932
Variance	1.920	4.957	6.877	6.983
RMS	2.622	2.227	2.622	2.643

Table 4.2: Results of the charge spectrum with added DCR

Fig. 4.6 shows each possible discharge, as summarized in table 2.1.



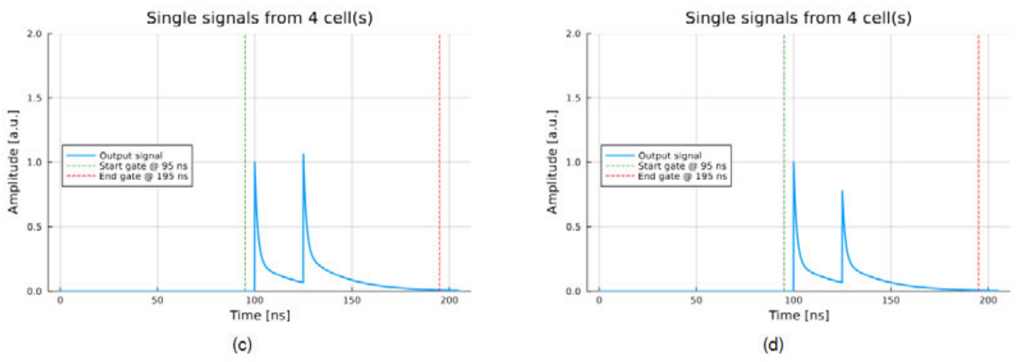


Figure 4.6: Comparison of the secondary effects with the integration time. Subfigure (a) depicts a single discharge (light/DCR), whereas (b) shows a discharge with prompt crosstalk. Subfigure (c) includes delayed crosstalk and (d) models an added afterpulse, as introduced in Fig. 2.8 and Fig. 2.11.

The expected results needed to verify the spectra for each effect are deductible from their respective distribution and the absolute timescale, combined with the integration time. Prompt crosstalk (Fig. 4.6 (b)) happens at the same time as the initial primary discharges caused by light and thus only the peaks should move, but it is still a discrete spectrum. Delayed crosstalk (Fig. 4.6 (c)) causes a lower height to the initial spectrum, with a broader base on the left side of the peaks. This is due to the Exponential distribution that determines when the prompt crosstalk happens; a later discharge means less charge is calculated for the integral since the integration limit is fixed. Afterpulses (Fig. 4.6 (d)) have a similar effect as delayed crosstalk, but the time constant is smaller, leading to a broader base on the right side of the peaks. Example spectra are included in the appendix, in A.5, A.6, and A.7 respectively.

The combined spectrum from light and DCR is easy to verify, but this is not the case for spectra that are made from more than two effects, especially since the code calculates the number of secondary effects for each effect based on the number of primary discharges. For afterpulses and crosstalk, this provides a Binomial distribution, which itself is already dependent on the Poisson distribution. Since a verification of this is not trivial, it is waived and this topic is open for an outlook on future enhancements of the SUM project. Further problems occurred with the displaying of the spectrum, where the number of bins of the histogram had an inexplicable effect on the data.

Chapter 5

Benchmarks

To gain a better insight into the functionality and the limits of the simulation, multiple benchmarks are done to assess performance details and compare them with the preceding implementation. Table 5.1 shows the conducted benchmarks, where it is noted that "transients" denotes the waveform of the output signal. Calculating the transients is additional information, included in the third benchmark.

#	Benchmark	Julia	Python
1	Mean number of photons (without transients)	X	
2	Number of cores	X	
3	Mean number of photons (with transients)	X	X

Table 5.1: Overview of the done benchmarks

Visual benchmarks that show the output signal and the resulting spectra have been done before and will not be mentioned in this chapter again. The key performance indicator is the overall runtime and additional measurements for the number of allocations and the used memory during a simulation are listed in the appendix while being referenced in the text. After the benchmarks are discussed, a comparison between Julia and Python is done, followed by a quick bottleneck analysis.

5.1 Mean number of photons

The benchmark for the mean number of photons (and the number of events) is the key criterion for the measurements since the goal of this thesis is to create a fast simulation for SiPMs. To verify this, the mean number of photons, as well as the number of events is changed, ranging from 1 to 20,000 and 1 to 10,000 respectively. Even higher numbers are possible, but the processed amount of data causes OutOfMemory() errors on the test machine, meaning there is insufficient RAM. The number of cells is fixed to 10,000 and this benchmark is done for the first five steps (in reference to Fig. 3.1), providing measurements for the generation and calculation of discharges including the amplitude. The resulting graphs are depicted in Fig. 5.1 and 5.2. Additional plots depicting the number of allocations and memory usage can be found in the appendix in figures A.12 and A.13. Table 5.2 holds the maximum values of all three benchmarks.

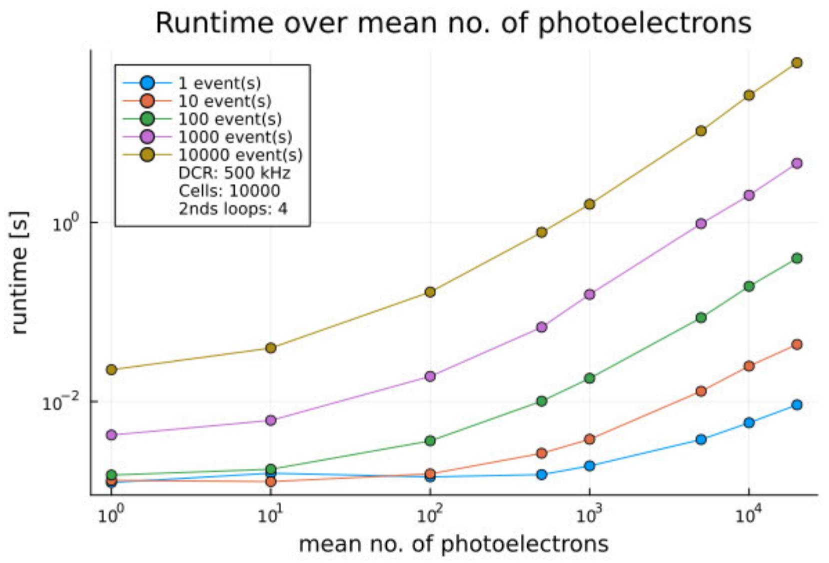


Figure 5.1: Runtime for different numbers of events following a rising mean number of photons

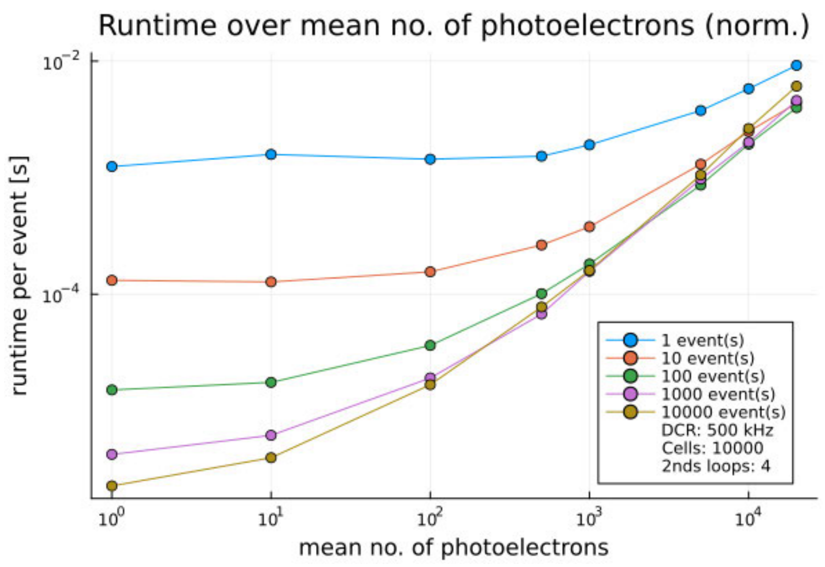


Figure 5.2: Runtime for different numbers of events following a rising mean number of photons, normalized to one event

Events	Max. runtime [s]	Max. Allocations	Max. memory [GB]
1	0.009	$5.86 \cdot 10^4$	0.006
10	0.043	$5.75 \cdot 10^5$	0.057
100	0.398	$5.76 \cdot 10^6$	0.569
1000	4.584	$6.73 \cdot 10^7$	5.842
10000	60.849	$7.65 \cdot 10^8$	59.983

Table 5.2: Maximum values for a mean number of photons of 20.000

Discernible is the linear scaling (mean $>$ 1,000) of the runtime, number of allocations, and memory usage, which poses a threat to the simulation for more than 10,000 events. These findings can figuratively be used to draw a paramount conclusion: the more events and the more discharges occur per event (whereas the reason for a discharge is insignificant), the higher the runtime, with a linear rise.

5.2 Number of cores

To determine the influence of using multiple cores for parallelization of the code, the first benchmark (mean number of photons) is reiterated. To allow a steady comparison, the number of cores is varied from four to one, with a fixed number of events (1000 and 10000). Fig. 5.3 shows the resulting graphs.

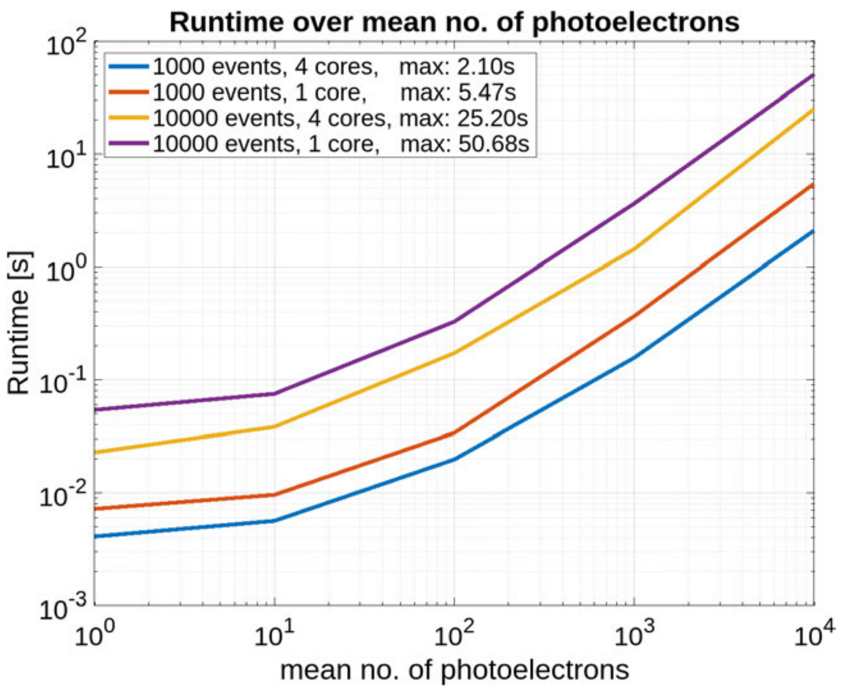


Figure 5.3: Runtime comparison between 1 and 4 cores

Comparing the value for a mean number of photoelectrons of 10,000, the speed gain by using four cores instead of one is 61% for 1000 events and 50% for 10,000 events. This implies, that using multi-core processors speeds up the simulation even further and can be brought in if one needs excessive speeds. For a normal usage single-core (or dual-core) computations are fast enough.

5.3 Julia vs. Python

As introduced in Chapter 3.1, a comparison between SUM and lightsimtastic is made, demonstrating its capabilities and contrasting the different approaches for Julia and Python respectively. This comparison is solemnly based on operation speed since it is the prevailing method of assessment and the main focus of this thesis, including the calculation of transients. Fig. 5.4 holds the result from this comparison, which is done for 100 events.

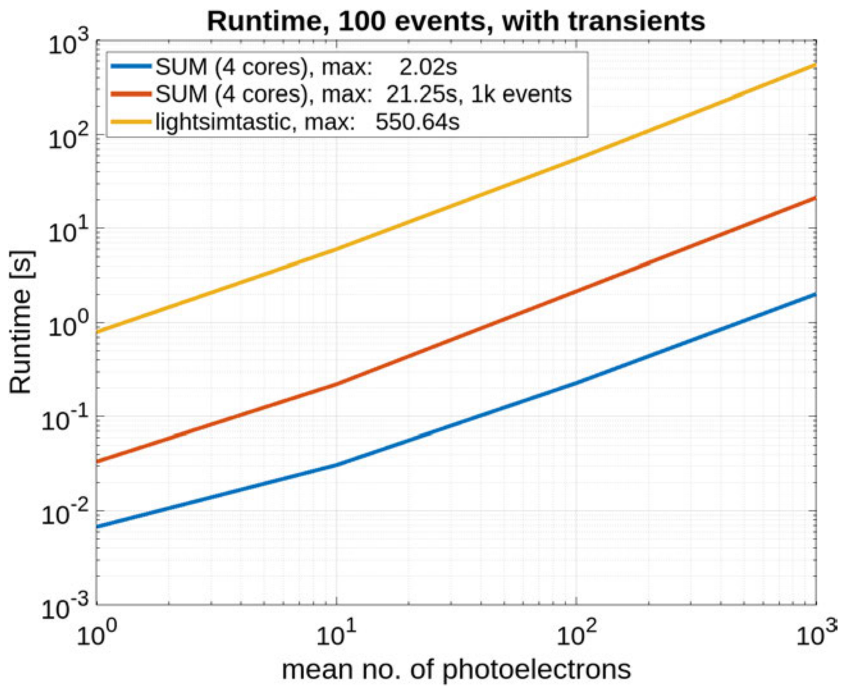


Figure 5.4: Runtime comparison between Julia (SUM) and Python (lightsimtastic) for 100 events with transient calculation.

Considering the maximum values for 100 events (2.02s and 550.64s), it can be remarked that SUM is approximately 272 times faster than lightsimtastic. To demonstrate how big this margin is, the same measurement is done for SUM with 1,000 events and it is still faster by approximately 26 times. Since it takes lightsimtastic 550 seconds to run once for 100 events, further research has not been conducted.

The huge timing differences can be explained by the compilation differences between Julia and Python. The latter is interpreted instead of compiled, which means the code is converted into bytecode and then run on a virtual machine, whereas Julia uses JIT compiling. This allows Julia to precompile the code into machine-readable code and then run it directly on the processor. This approach usually is much faster and therefore the results are as expected; it is important to also mention, that Python can be highly optimized for speed but this is not part of this thesis. Furthermore, SUM calculates secondary effects four times instead of once like lightsimtastic.

5.4 Bottleneck Analysis

To understand the bottleneck of the simulation, a timing analysis is done for all steps, excluding the calculation of the spectrum and the setting of variables. In Fig. 5.5 the contribution of each part to the overall runtime is presented. Since the runtime of setting variables is constant, this step is omitted for the graphic. Comparing the proportions, bottlenecks that slow the whole simulation down occur in the filling of values and sorting and flagging steps, since these have the greatest overall impact (87% combined). This is as expected since these two parts are computationally the heaviest; for each event, random values for ID and the timestamp are sourced and saved in the matrix with additional information. This takes longer, the more events are simulated.

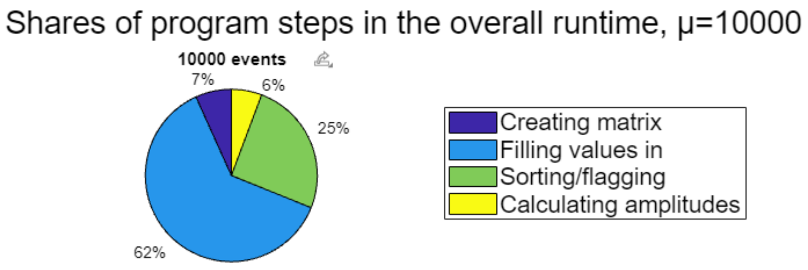


Figure 5.5: Percentage shares of each step to the overall runtime

Chapter 6

Conclusion

As explained in Chapter 3, this thesis implements new algorithms that model the main physical processes of detecting light with an SiPM, combined with the influence of correlated noise. The presented simulation in its current state is capable of simulating the output signal of an SiPM, by calculating primary and secondary discharges with their respective cell ID, timestamp, source, and amplitude. This approach differs vastly from lightsimtastic, since it is now possible to track each discharge and precisely tell what happened when. Additionally, the SiPM is now used as a reference object with a distinct geometry, which is a novelty and effectively enables this tracking. Considering the thesis goals, this represents the fulfillment of the first and second objectives. Furthermore, performance predictions for SiPMs for different light intensities can be made, since it is possible to alter the light source via a distribution and set the mean number of photoelectrons. It is also possible to study secondary effects, by using their respective probabilities paired with a distribution for the occurrence. These predictions are also possible with lightsimtastic, but the new implementation simplifies this process. By also surpassing the runtime (2.02s against 550.64s for 100 events with a mean number of photoelectrons of 1,000), the study of high-light intensities is specifically facilitated. This speed gain is made possible by the usage of thread parallelization, a feature that is directly built into the programming language. Julia also offers fast vector operations and its JIT compilation leverages the impact on the runtime even more. It is thus possible to conclude, that the fourth goal is also accomplished since the usability is given.

It is important to note the limitations of the implemented functions to figure out where future enhancements could be introduced. Currently, the runtime scales linearly, which limits the number of calculable events, and aiming for a faster simulation could be a priority. Besides this, more parameters defining the SiPM should be implemented (section 2.3.6) together with electronics noise, and a picosecond timescale could help with more exact tracking. Furthermore, different statistical approaches could better capture real-world characteristics. Two remaining issues are the calculation and binning of the spectrum, which could not be resolved in the scope of this thesis. The third goal is thus only partially fulfilled.

The contribution to current research is, that this thesis and its resulting program serve as the foundation for the SiPM Unified Model (SUM) research directive, proving the use of Julia as a basis for a novel SiPM simulation. Having a simulation that accurately provides the output signals and spectra of an SiPM, allows a deeper study of these components, without having to physically be in a lab and measure data. The subordinate goal of providing a proof-of-concept for the use of Julia as a new approach to the simulation of the response of SiPMs is therefore given.

Bibliography

[1] S. Gundacker and A. Heering, "The silicon photomultiplier: Fundamentals and applications of a modern solid-state photon detector," *Physics in Medicine & Biology*, vol. 65, 2020, 17TR01. DOI: https://doi.org/10.1088/1361-6560/ab7b2d.

[2] F. Acerbi and S. Gundacker, "Understanding and simulating sipms," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 926, pp. 16–35, 2019. DOI: https://doi.org/10.1016/j.nima.2018.11.118.

[3] F. Sekan, *Pros and cons of julia in data science*, Website, opened on 01.07.24, 2023. [Online]. Available: https://medium.com/@filip.sekan/pros-and-cons-of-julia-in-data-science-3f386cb71757.

[4] S. Piatek, *What is an sipm and how does it work?* Website, opened on 12.07.2024, 2016. [Online]. Available: https://hub.hamamatsu.com/us/en/technical-notes/mppc-sipms/what-is-an-SiPM-and-how-does-it-work.html.

[5] R. Klanner, "Simulation of the response of sipms; part ii: With saturation effects," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 1059, 2024, 169018. DOI: https://doi.org/10.1016/j.nima.2023.169018.

[6] H. Körner and W. Zinth, *Physics III: Optics, quantum phenomena, and atomic structures ['Physik III: Optik, Quantenphänomene und Aufbau der Atome']*. Oldenbourg Verlag GmbH, original language: German, 1994, ISBN: 3-486-39951-9.

[7] onsemi, "Introduction to the silicon photomultiplier (sipm)," onsemi, Tech. Rep., 2023, AND9770/D. [Online]. Available: https://www.onsemi.com/pub/Collateral/AND9770-D.PDF.

[8] E. Garutti, "Silicon photomultipliers," in *Handbook of Particle Detection and Imaging*, I. Fleck, M. Titov, C. Grupen, and I. Buvat, Eds. Cham: Springer International Publishing, 2020, pp. 1–21, ISBN: 978-3-319-47999-6. DOI: 10.1007/978-3-319-47999-6_48-1.

[9] R. Quimby, *Photonics and lasers*, Website, opened on 12.07.2024, 2006. [Online]. Available: https://www.globalspec.com/reference/21446/160210/chapter-14-4-1-avalanche-photodiode.

[10] S. Stornelli et al., "Silicon photomultiplier sensor interface based on a discrete second generation voltage conveyor," *Sensors*, 2020. DOI: 10.3390/s20072042.

[11] E. Garutti, R. Klanner, J. Rolph, and J. Schwandt, "Simulation of the response of sipms; part i: Without saturation effects," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 1019, 2021, 165853. DOI: https://doi.org/10.1016/j.nima.2021.165853.

[12] K. Dransfeld and P. Kienle, "Electromagnetic waves ['elektromagnetische wellen']," in *Physics II: Electrodynamics and special theory of relativity ['Physik II: Elektrodynamik und spezielle Relativitätstheorie'].* Oldenbourg Wissenschaftsverlag GmbH, original language: German, 2002, ch. 10, ISBN: 3-486-24053-6.

[13] J. Rolph, *A silicon photomultiplier simulation package written in python*, Repository, opened on 01.07.2024. [Online]. Available: https://gitlab.desy.de/jack.rolph/lightsimtastic.

[14] K. Pykes, *Julia vs python - which should you learn?* Website, opened on 01.07.2024, 2022. [Online]. Available: https://www.datacamp.com/blog/julia-vs-python-which-to-learn.

[15] S. Ahn and J. Fessler, "Standard errors of mean, variance, and standard deviation estimators," 2003, Website, opened on 01.07.2024. [Online]. Available: https://web.eecs.umich.edu/~fessler/papers/files/tr/stderr.pdf.

[16] A. A. Kostina, P. M. Tzvetkov, and A. N. Serov, "Investigation of the method of rms measurement based on moving averaging," in *2020 55th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, 2020, pp. 235–238. DOI: 10.1109/ICEST49890.2020.9232779.

Acronyms and Abbreviations

P_{AP}	Probability for afterpulse to occur.
P_{DXT}	Probability for delayed crosstalk to occur.
P_{PXT}	Probability for prompt crosstalk to occur.
A	Amplitude.
AP	Afterpulse.
DC	Discharge.
DCR	Dark Count Rate.
DXT	Delayed Crosstalk.
G	Gain.
GA	Geiger array.
ID	Identification.
JIT	Just-in-time [compilation].
LiDAR	Light Detection and Ranging.
PDE	Photon Detection Efficiency.
PDF	Probability Density Function.
PXT	Prompt Crosstalk.
RAM	Random Access Memory.
RMS	Root-Mean-Square.
SC	Source.
SE	Standard Error.
SiPM	Silicon Photomultiplier.
SPAD	Single-Photon Avalanche Diode.
SPTR	Single Photon Timing Resolution.
SUM	SiPM Unified Model.
TS	Timestamp.

Appendix A

Figures and Tables

A.1 Parameters

Group	Parameter name	Description	value/unit
General	n_events	Total number of events	1
	n_rows	Number of pixel rows	10
	n_rows	Number of pixel columns	10
	n_cells	Total number of cells	100
	mu_light	Mean number of photons	10
	dcr	Dark count rate	100 kHz
	seed	Seed for the RNG	1234
	r_fast	Proportion of fast component	0.2
Times	t_start_rising_signal	Arrival time of photons	100 ns
	t_integration_length	Length of integration	100 ns
	t_start_integration	Start of integration (in respect to rising signal)	-5 ns
	t_start_simulation	Start of simulation (in respect to rising signal)	-100 ns
	t_var_photons	Variance of arrival time	1 ns
	t_pulselength	Cell "blind" time	2 ns
	tau_delayed_xt	Time constant DXT	25 ns
	tau_afterpulse	Time constant AP	7.5 ns
	tau_fast	Time constant fast component	20 ns
	tau_slow	Time constant slow component	1.5 ns
Secondary effects	p_prompt_xt	Prompt crosstalk probability	0.2
	p_delayed_xt	Delayed crosstalk probability	0.1
	p_ap	Afterpulse probability	0.05
	n_xt_loops	Number of secondary calculations	4
Distributions	df_n_light	Number of photons	Poisson
	df_n_dcr	Number of DCR	Poisson
	df_t_light	Photon arrival time	Normal
	df_t_dcr	Time spreading DCR	Uniform
	df_n_pxt	Number of PXT	Binomial
	df_t_pxt	Time spreading of PXT	undef
	df_n_dxt	Number of DXT	Binomial
	df_t_dxt	Time spreading of DXT	Exponential
	df_n_ap	Numbers of AP	Binomial
	df_t_ap	Time spreading of AP	Exponential

Table A.1: Overview of the input parameters

A.2 File Structure

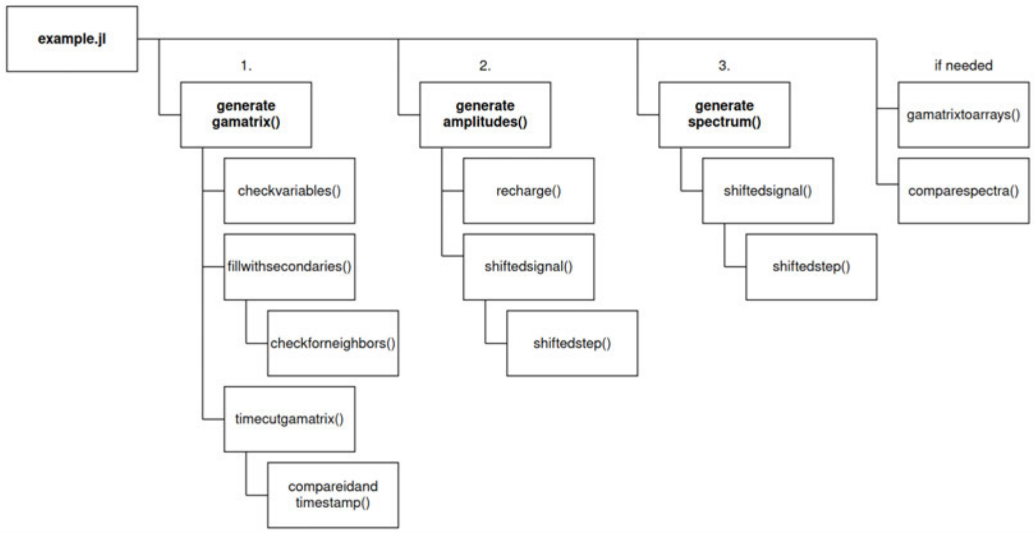


Figure A.1: Functional dependencies of the example.jl file (in SUM/example)

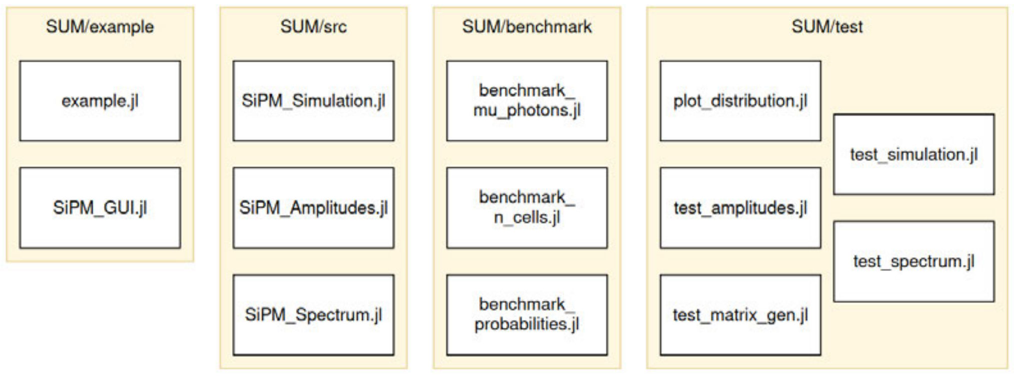


Figure A.2: Folder structure for the SUM-package

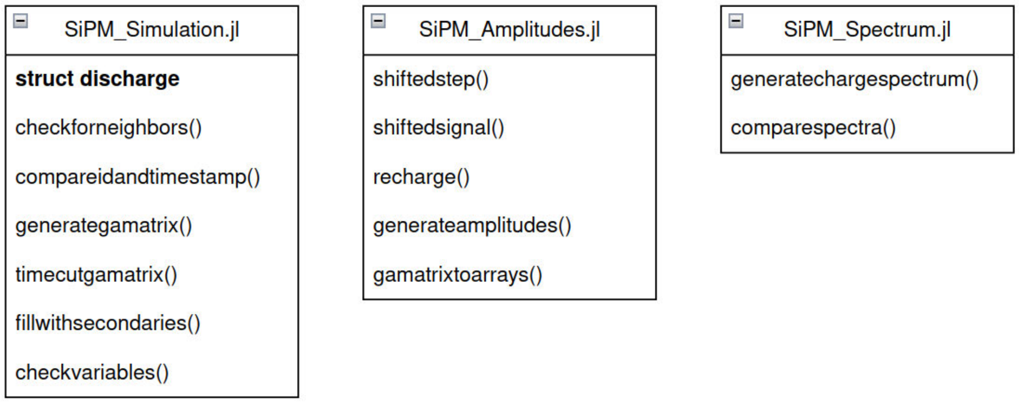


Figure A.3: File structure containing the implemented functions and the custom struct in the SUM/src folder

A.3 Tests and Data

Function	Status
checkforneighbors()	Tested
timecutgeigerarraymatrix()	Tested
generategamatrix()	Tested
fillwithsecondaries()	Tested
generateamplitudes()	Tested
generatechargespectrum()	Tested
compareidandtimestamp()	Tested
shiftedsignal()	Tested
shiftedstep()	Tested
recharge()	Tested
checkvariables()	Tested
gamatrixtoarrays()	Untested
comparespectra()	Untested

Table A.2: Overview of the done tests. The last two functions are untested, since they are optional and do not contribute to the actual calculations

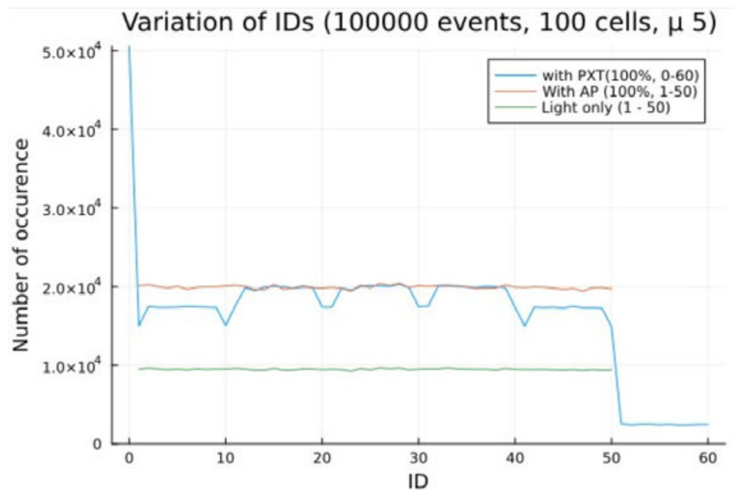


Figure A.4: Variation of the cell ID, if an SiPM (10x10 cells) is only illuminated on the first 50 cells. Afterpulsing happens in the same cells and thus the occured IDs only double, whereas for prompt cross-talk the bordering cells also occur. Delayed cross-talk discloses the same behavior as prompt cross-talk.

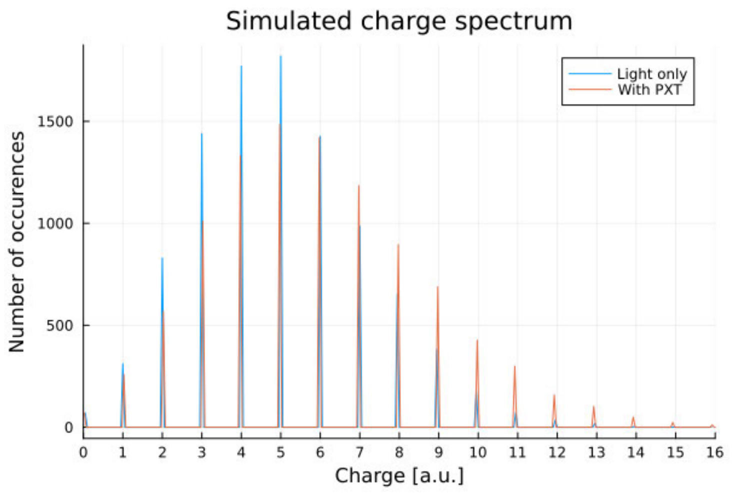


Figure A.5: Spectrum with prompt cross-talk (p_pxt = 20%)

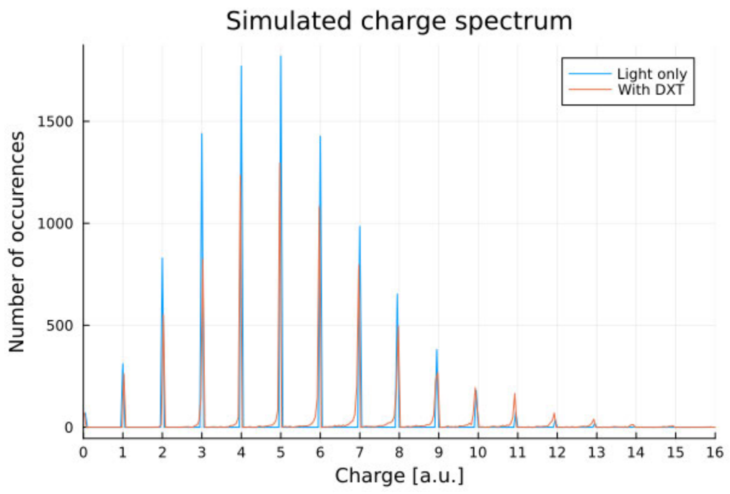


Figure A.6: Spectrum with delayed crosstalk (p_dxt = 20%)

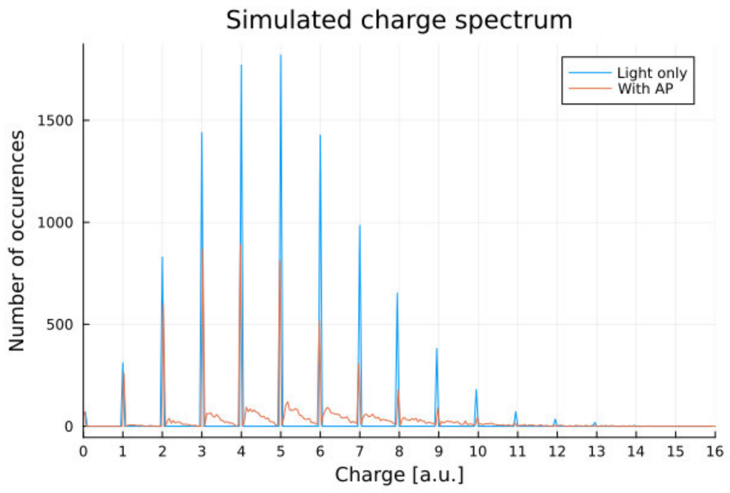


Figure A.7: Spectrum with afterpulsing (p_ap = 20%)

A.4 Distributions

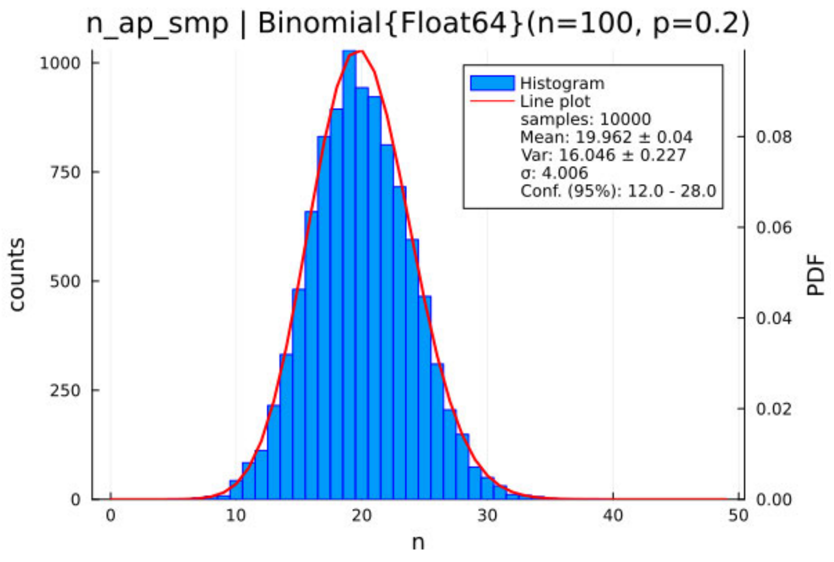


Figure A.8: Binomial distribution

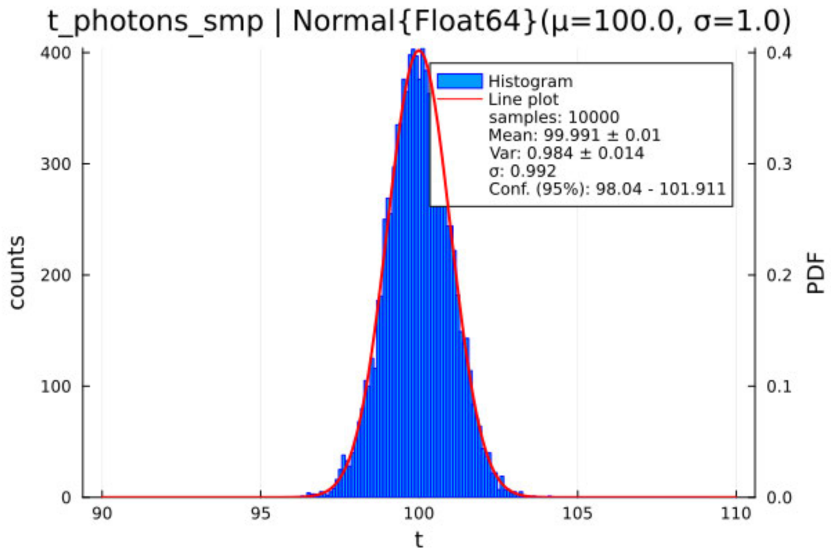


Figure A.9: Normal distribution

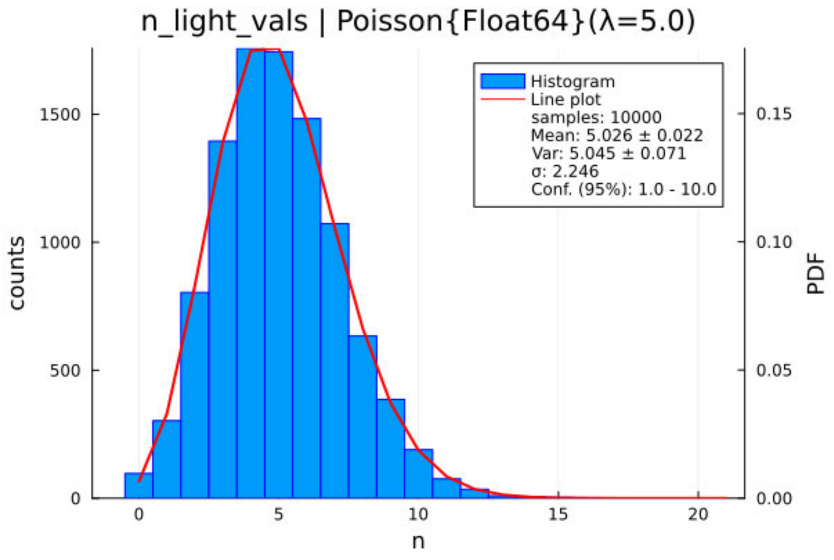


Figure A.10: Poisson distribution

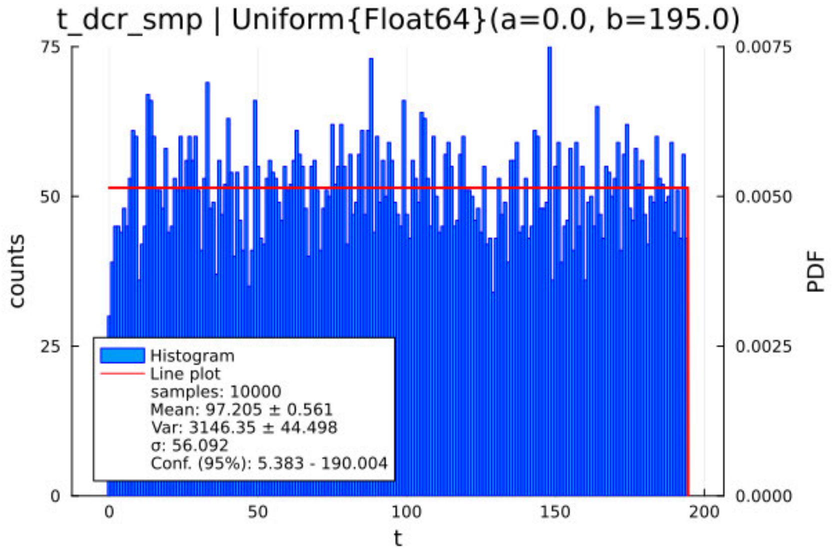
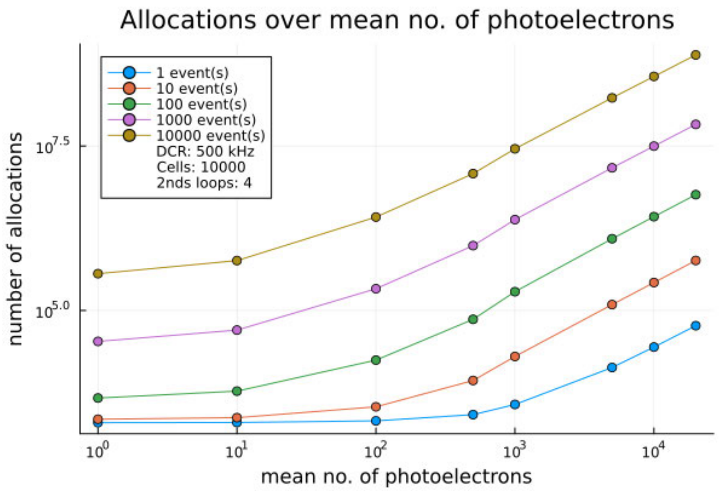
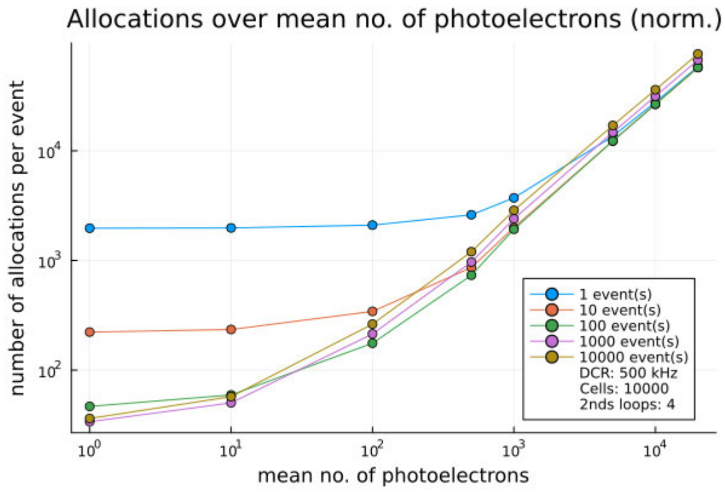


Figure A.11: Uniform distribution

A.5 Benchmarks

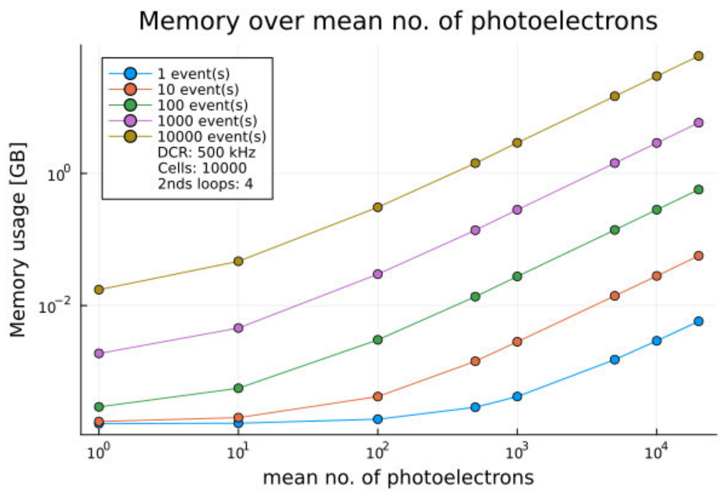


(a) raw data

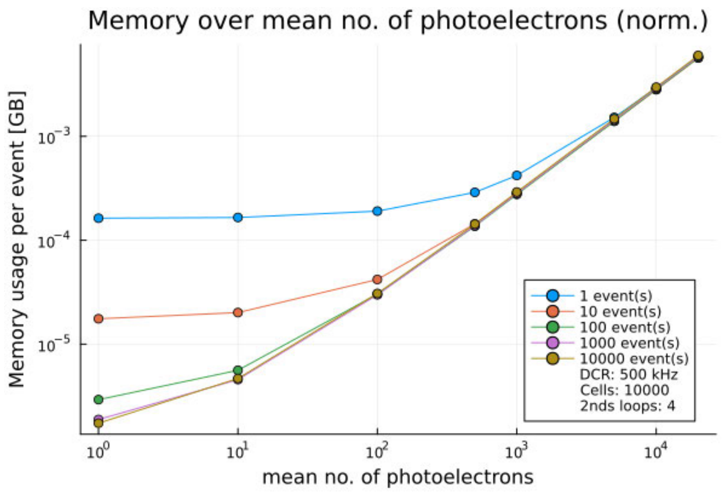


(b) normalized

Figure A.12: Allocations for different number of events in accordance with a rising mean number of photons, with a. raw data and b. normalized to one event



(a) raw data



(b) normalized

Figure A.13: Memory usage for different number of events in accordance with a rising mean number of photons, with a. raw data and b. normalized to one event

Note of Thanks

I want to personally thank everyone who accompanied me on this journey of writing my bachelor thesis, including everyone who works at the UHH lab and is involved in SiPM research. I want to thank my supervisors Prof. Dr. Matthias Kronauge and Prof. Dr. Erika Garutti for their support. Furthermore, I want to single out Dr. Massimilano Antonello, who supported me with all his knowledge and spent long days helping me with the simulation. Lastly, Julian and Leevie Kruse, two fellow students who accompanied me during my studies, helped me by proofreading this thesis.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den ______________________________

Unterschrift ________________________________