DEVELOPMENT OF AN AUTOMATED GPIB SYSTEM FOR CHARACTERIZATION OF SIGE AVALANCHE PHOTODIODES

By

David A. Santiago

Bachelor of Science – Electrical Engineering
University of Nevada, Las Vegas
2020

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Engineering – Electrical Engineering

Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2023

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 5, 2023

This thesis prepared by

David A. Santiago

entitled

Development of an Automated GPIB System for Characterization of SiGe Avalanche Photodiodes

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Engineering – Electrical Engineering
Department of Electrical and Computer Engineering

R. Jacob Baker, Ph.D.
Examination Committee Chair

Peter Stubberud, Ph.D.
Examination Committee Member

Yingtao Jiang, Ph.D.
Examination Committee Member

Victor Kwong, Ph.D.
Graduate College Faculty Representative

Alyssa Crittenden, Ph.D.
Vice Provost for Graduate Education & Dean of the Graduate College

Abstract:

Photon counting is a technique in many modern technologies such as medical imaging and LiDAR systems. Photon counting is needed in many aspects of the development process for these technologies from photodetector characterization to operation of the finished product. This thesis will serve to solve the challenge of quantifying photodetector performance using automating test equipment with known software packages such as MATLAB to be able to characterize photodetectors in an efficient manner.

A custom MATLAB program has been developed to interface with various instrumentation using a General-Purpose Interface Bus (GPIB) and a Prologix GPIB-USB controller that will be able to send SCPI commands to conduct experiments on Avalanche Photodiodes for their various properties. The program automatically calculates the estimated completion time to notify the user of when the experiment and data will be complete for data processing and presentation. Testing throughput is significantly advanced through this automated system with a test consisting of 1,000 unique operating points taking only about 2 hours, compared to having a user manually performing the same test in around 8 hours; a decrease of time by a factor of 4. This enables complete characterization and calibration of a typical photodetector in less than a day in a manufacturing environment. The results are processed in a program that compares multiple tests for parameters such as the generated photon counts produced by incident light and normalized count data over a certain area of a photodiode.

Acknowledgements

I would first like to thank God for another beautiful day of life. This is what my family reminds me to this day – my mom, dad, brother, and little sister.

I also want to thank Dr. R. Jacob Baker for lending me the opportunity to join the Baker Research Lab group, for I have gained and appreciated all the practical experience that I have witnessed throughout my studies at UNLV. Angsuman Roy, Gonzalo Arteaga, Abraham Lopez, and all the Baker group, thank you for the wonderful memories.

To Allyana, thank you for helping me through my ups and downs; I promise you and to everyone I love and care for that I will always do my very best.

To my academic colleagues and students, it was a pleasure to meet everyone and assist you and the college with what I have learned from getting to know you all.

Table of Contents

Table of Figures

Chapter 1 Intro to Photon Counting

1.1 What is a photon?

Before introducing what a photon is, one must bring in a trivial question: what do photons do? One may look at an object such as an apple, and light reflects off the object, however, one will never be able to see individual photons, but one experiences what photons do. What photons do is bind electric charges along with magnetic effects together by discrete irreducible processes of photon emission and absorption connected by a continuous process of propagation [Roychoudhuri, 1]. The modern photon was originally theorized by Albert Einstein in 1905 to explain various experimental observations about light such as black body radiation and the photoelectric effect. In the simplest terms, a photon is the smallest possible unit of electromagnetic energy that can be generated or detected. This level of explanation is sufficient for discussions on photon counting techniques and applications. While there are many quantum physics-based descriptions of a photon, an understanding of these is not required for practical photon counting.

1.2 Photon Counting

Photon counting is a method of quantifying light intensity by counting events triggered by photons rather than measuring the current flowing in a photodetector. This is a powerful technique that is a key component of many emerging technologies such as LiDAR and medical imaging. Photons with wavelengths in the visible light range have a greater advantage over other alternatives such as using RADAR due to having shorter wavelengths and can be favored when detecting photons in medical imaging. Figure 1 shows an example of medical imaging of the author's retina using optical coherence tomography, a type of LiDAR imaging which can use photon counting [Mohan, 2], where Figure 1 displays the author's eye.

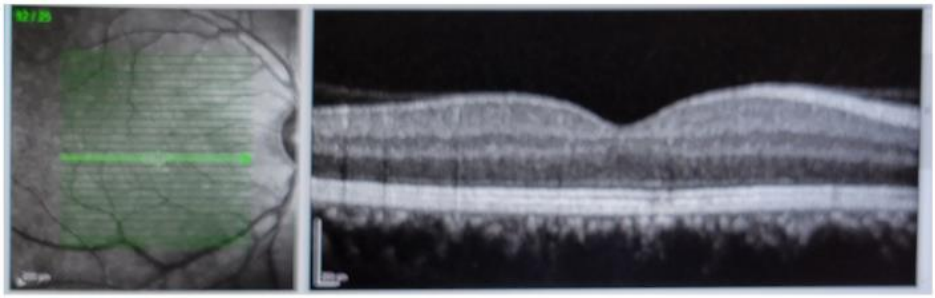


Figure 1 - Imagining of the Author's eye

1.3 Photodetection

The process of detecting photons can be mainly characterized by the absorption of photons in the material with generation of charge carriers [Donati, 4]. These charge carriers can be electrically measured and are proportional to the light irradiance on a photodetector. The most fundamental measurement of the performance of a photodetector is responsivity, typically given in units of amps/watt. Responsivity is the amount of electrical current (amps) produced by a photodetector for a given amount of light power (watts). The photodetector material, structure of the layers, biasing conditions, temperature, and many other environmental and design parameters contribute to the responsivity. Responsivity is accurate when the number of photons hitting the detector is large. For irradiance values that correspond to relatively small numbers of photons, the statistical nature of charge carrier generation along with noise limitations of electronics, requires photon counting techniques to resolve. This will be expanded upon later in the chapter.

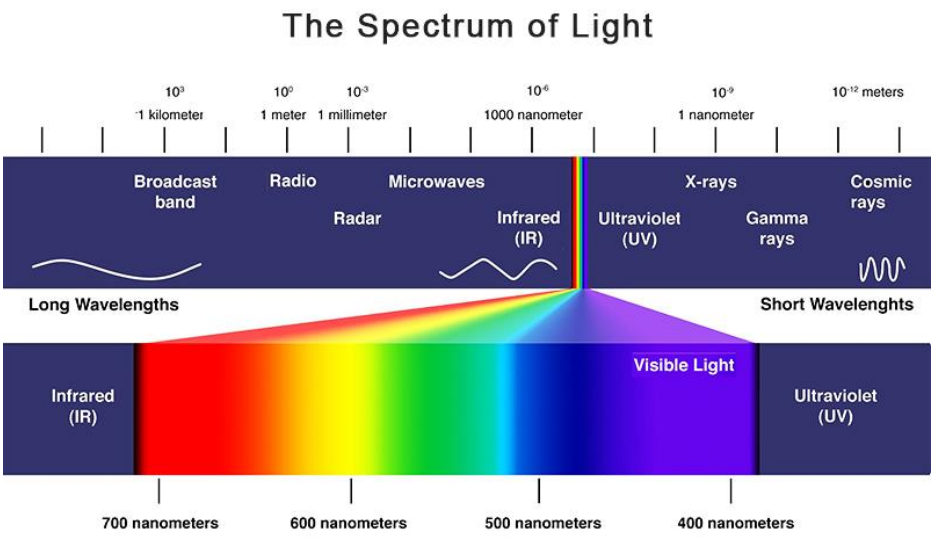


Figure 2 - Spectrum of Light

1.4 Quantum Efficiency

Another merit that is used to characterize photodetectors is the quantum efficiency. Quantum efficiency is the ratio of the number of photocarriers to the number of incident photons [Pearsall, 5]. For example, if the result of a sensor absorbing every photon is that a minority carrier (a hole in semiconductors) reaches the p-n junction of a diode (and therefore pulling an electron), then the quantum efficiency would be 100%. Photocurrent is described as the number of electrons per second, n_e, times the charge of each electron, q, shown in Equation 1.1. This equation can also be arranged to describe the number of electrons per second as a function of current and charge.

$$I_{photocurrent} = q \cdot n_e, \rightarrow n_e = \frac{I_{photocurrent}}{q}\ electrons/sec$$

Equation 1.1 – Describing photocurrent

Optical power is described as the number of photons per second, n_ϕ, times the energy per photon, $E = hf$, where h is Planck's constant, and f is the frequency of a photon, shown in Equation 1.2. This equation can also describe the number of photons in terms of power and energy.

$$P_{optical} = n_\phi \cdot hf \rightarrow n_\phi = \frac{P_{optical}}{hf}\ photons/sec$$

Equation 1.2 – Optical Power Equation

Combining both Equations 1.1 and 1.2, the quantum efficiency, n_Q, can be described by Equation 1.3, where $I_{photocurrent}$ and $P_{optical}$ are values that can be measured and produced in a laboratory setting.

$$Quantum\ Efficiency = n_Q = \frac{n_e}{n_\phi} = \frac{I_{photocurrent} \cdot hf}{P_{optical} \cdot q}$$

Equation 1.3 – Quantum Efficiency

1.5 Photon Counting Detectors

Photon counting begins with a suitable photodetector. All photon counting detectors use some form of electron multiplication, such as avalanching and/or breakdown effects to amplify the photocurrent from a single photon. Photon counting detectors can be divided into two broad categories: semiconductor and vacuum tube based. Vacuum tube-based detectors such as photomultiplier tubes are only used in niche applications and will not be covered in this thesis. A few examples of semiconductor photodetectors are the PN photodiode, Schottky photodiode, and the avalanche photodiode. The most common photon counting detector is the avalanche photodiode (APD). An APD is a photodiode with an internal structure optimized for electron

multiplication and is typically biased at relatively high voltages such that a single photoelectron causes an "avalanche" of electrons due to impact ionization. This device can therefore be thought of as a photodiode with gain. Combining hundreds of Geiger-mode APD unit cells (Geiger-mode APD combined with a series quench resistor) in parallel creates a silicon photomultiplier (SiPM) also known as a multi-pixel photon counters (MPPC). The APD will be discussed in more detail later in this thesis, along with the structures used for experimentation.

1.6 Motivation of Thesis

General photon counting in a laboratory setting will require instrumentation such as power supplies, generators, analyzers, and active personnel that conducts the experiments. Depending on the equipment used, this may be a tedious procedure and would require extensive amounts of user experience to manage and gather precise results. One form of conducting experiments that is accessible to the user is the use of a communication system that is already in place in most laboratory equipment, a General-Purpose Interface Bus (GPIB) system. Using this system may require the use of purchasing special subscription-based licenses and controllers such as National Instrument's LabVIEW and GPIB control devices, however, the cost to maintain a 1st party communication system for a multi-year project that only requires basic data retrieval would be cost-ineffective and add unnecessary complexity to operate.

One solution of having a perpetual low-cost and efficient manner of using GPIB is by using a 3rd party controller that is relatively cheaper than a 1st party controller and by utilizing MATLAB, which is an excellent program for data processing and presentation. This thesis will discuss the instrumentation used, investigate the process of setting up an automated system with GPIB, and to analyze select size and shaped APDs for their characteristics.

1.7 Context of Thesis

This thesis will serve as a reference and guideline for experimentation and future research on APDs. The first two chapters serve as an introduction to photon counting and provide background information on what the user would find helpful during testing. Chapter 3 discusses the current experimentation system in place and explains why an alternative system would be preferred. Chapter 4 provides the context behind GPIB that is deemed practically beneficial for not just this thesis, but also for other research work with instrumentation that works with GPIB. Chapter 5 goes into depth on the integration of MATLAB and GPIB to make a powerful program that is able to automate GPIB-configurable devices and retrieve data, and Chapter 6 goes on to discuss the analysis of data produced and processed via MATLAB. The final portion of this thesis contains a conclusion chapter summarizing the work

Chapter 2 Background information of Photodiodes

2.1 Semiconductors

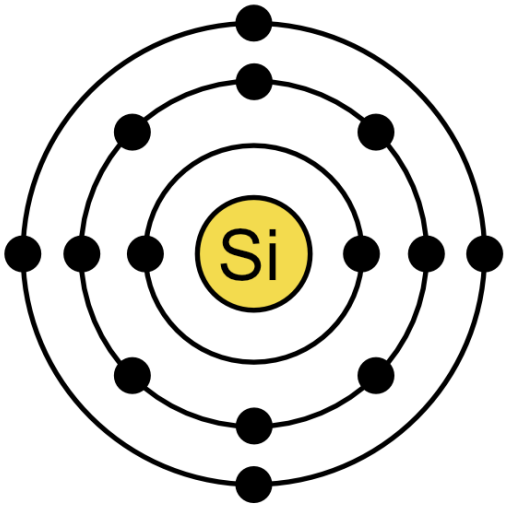


Figure 3 – Bohr Structure of Silicon atom

An elementary structure of a photodiode begins with silicon, an element with an equal number of 14 protons, electrons, and neutrons. The Bohr model electron structure of an atom follows a stable electron shell pattern where the innermost shell contains 2 electrons, the second shell contains 8 electrons, and the third shell contains 8 electrons, where an atom will either want to gain or lose electrons depending on the number of valence electrons in the outermost shell to achieve stability. Silicon fills these shells where the valence shell only has 4 electrons, shown in Figure 3 , giving it the ability to either gain or lose an electron. In its pure form, silicon forms covalent bonds with other silicon atoms and creates a structure where each silicon is sharing a bond with four neighboring silicon atoms, shown in Figure 4.

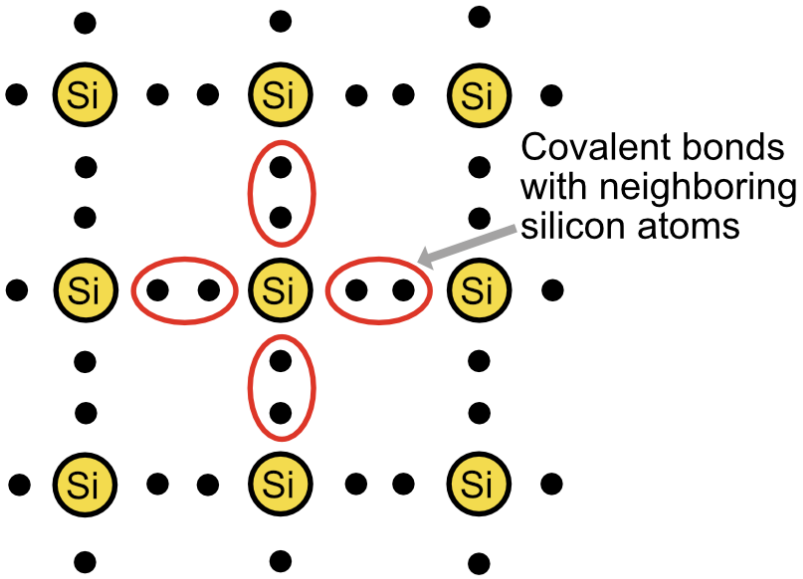


Figure 4 – Covalent bonding of silicon atoms

Silicon on its own will not conduct a current and ideally has no free moving electrons. To make it into a semiconductor, silicon (which has 4 valence electrons) is doped with either donor atoms (elements found under Group V with an extra electron) such as phosphorus or acceptor atoms (elements found under Group III with one less electron) such as boron. Doping silicon with donor atoms will introduce an extra mobile electron that is able to freely move around and create an n-type semiconductor, shown in Figure 5. Similarly, doping silicon with acceptor atoms will create a space where an electron can jump towards, and a mobile "hole" appears to move and create a p-type semiconductor, shown in Figure 6. Although n-type and p-type semiconductors can freely move charge, they are both neutrally charged as the semiconductors will not conduct a current until a potential difference is introduced.

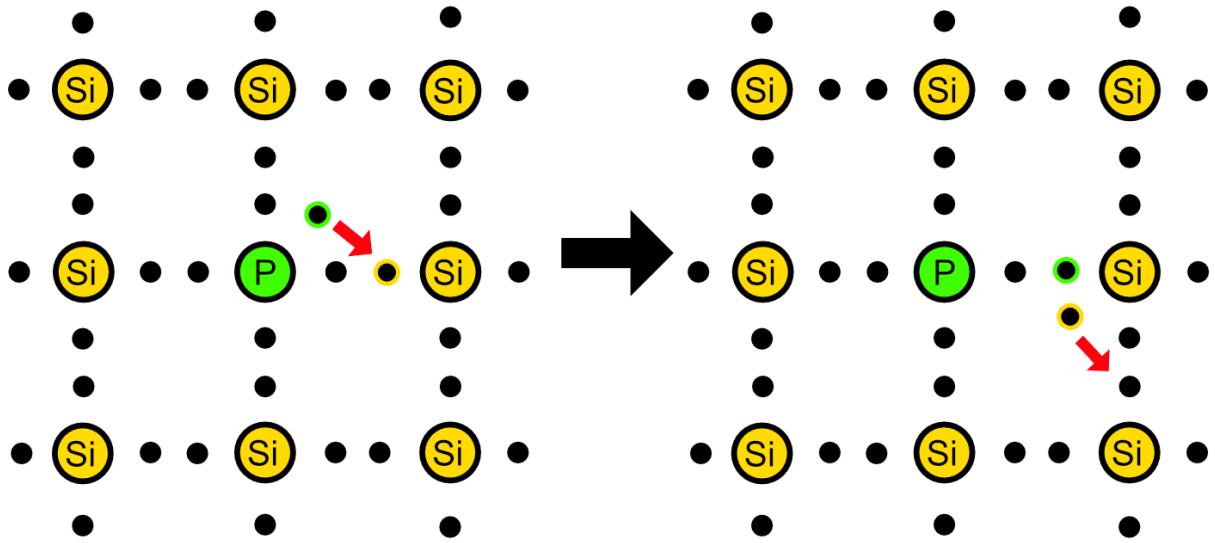


Figure 5 – Movement of mobile free electron through n-type semiconductor

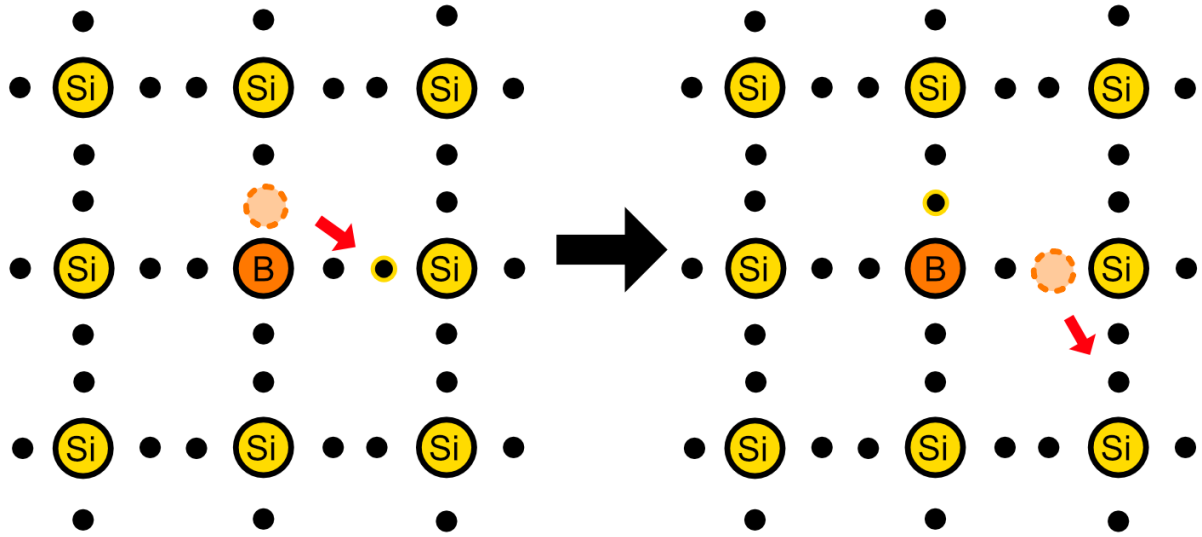


Figure 6 – Movement of “hole” through p-type semiconductor

2.2 The Diode

A fundamental device that appears in practice either as a useful technology or a parasitic is the diode. A diode is formed when p-type and n-type semiconductors neighbor each other and

create a PN junction. The PN junction consists of a region where the free mobile electrons from the n-type material move across the junction and leave behind a positive charge, and similarly the holes of the p-type material move across and leave behind a negative charge. Consequently, this creates a neutral region that is depleted of free roaming carrying electrons called the depletion region. The fixed atoms on each side of the junction will generate a force that will equalize the charge distribution in the diode and give rise to a parasitic capacitance [Baker, 8]. The p-type material will become the anode of the diode and the n-type material will be the cathode, shown in Figure 7. In the depletion region, the n-type side will have a net positive charge and the p-type will have a net negative charge, creating an electric field in the depletion region and therefore a junction voltage is created.

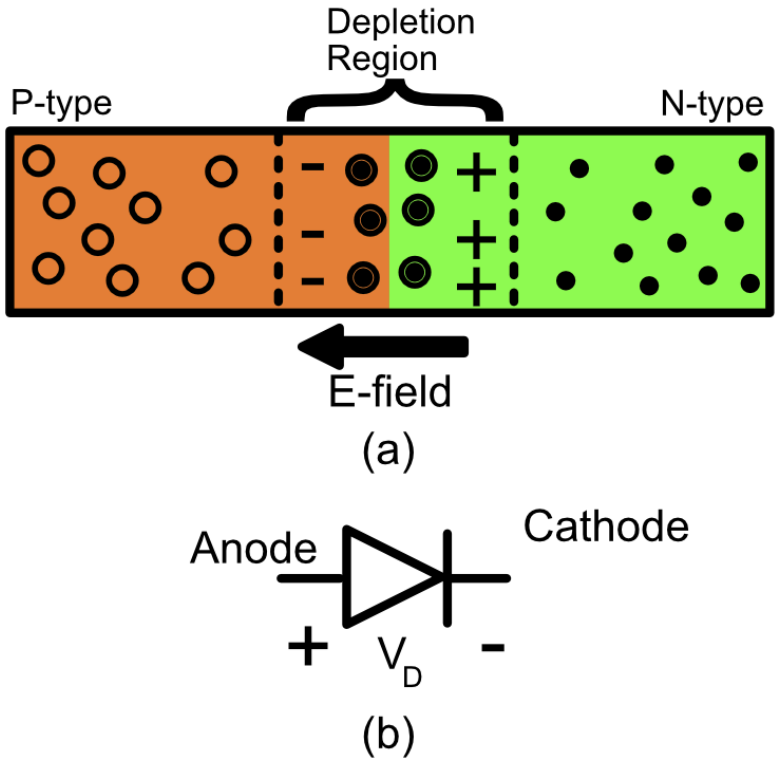


Figure 7 – PN diode (a) showing depletion region (b) schematic symbol

If the PN diode is forward biased (a positive potential difference between the anode and cathode), the electrons on the n-type side will move towards the p-type side and vice versa for the holes. The applied negative voltage on the n-type side assists with electron diffusion and therefore the length of the depletion region shortens. Conversely, in reverse bias operation, the diode will experience a negative external voltage which will prevent the electron and hole carriers from crossing the junction, therefore widening the depletion region (and lowering the depletion region capacitance) and conduct a negligible leakage current until a very high voltage punches through the diode. This punch-through voltage is also known as the breakdown voltage of a diode, shown in Figure 8. All APDs described in this thesis are only tested in the reverse bias region including breakdown and beyond.

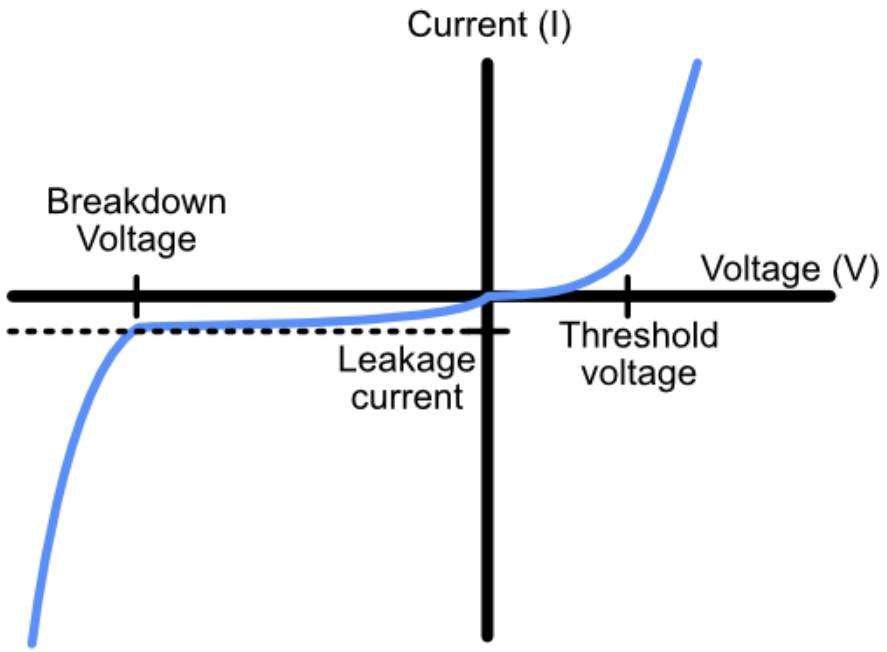


Figure 8 – I-V curve of a diode

2.3 Photodiodes

The structure of a photodiode is a PN junction where the photodiode's semiconductor material is designed to be more responsive to light. When light is applied to a photodiode, the photons from the light interact with silicon atoms and release an electron if the photon contains enough energy to knock out a valence electron from the silicon atom, which results in an electron-hole pair. If photons are absorbed in either the P or N regions, the electron-hole pair would recombine in the material as heat, however, if photons are absorbed in the depletion region where there are no free electrons, the electron-hole pair that would move to opposite sides of the PN junction due to the electric field inside the depletion region and generates current. More simply, photocurrent is current generated due to incident photons applied onto a diode.

Operating a photodiode in reverse bias mode, as the reverse bias voltage increases, the depletion region continues to widen. This consequently will result in a larger area of absorption for photons and generate a larger amount of photocurrent. The direction of the photocurrent is dictated by the movement of photogenerated holes in the depletion region, where the holes (positive charges) move with the electric field (direction from n-type to p-type). This direction is the same as the leakage current through the photodiode. At a constant reverse bias voltage (and therefore constant leakage current), the amount of photocurrent flow is proportional to the intensity of the photons applied to the photodiode [9], shown in Figure 9.

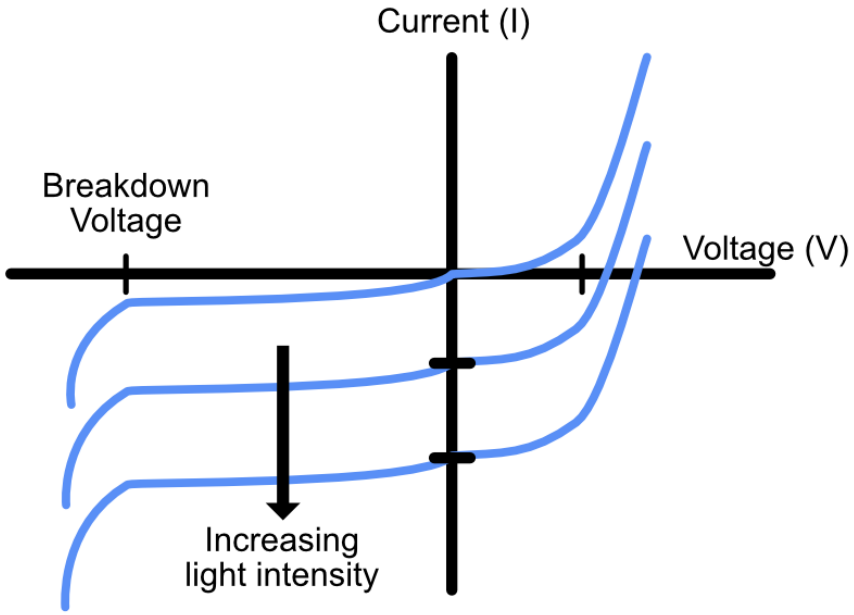


Figure 9 – Family of photodiode IV curves with increasing light intensity.

2.4 Avalanche Photodiodes

The structure of all photodiodes stem from the elementary PN junction design. From here, devices such as the PIN photodiode, the Schottky photodiode, and the avalanche photodiode are designed to meet certain specifications and performance modes. The avalanche photodiode (APD) is a high-speed, high-sensitivity photodiode that internally multiplies photocurrent when reverse voltage is applied. The internal multiplication means that the APD has a high sensitivity to photons and therefore can perform at low light intensities [Hamamatsu, 10].

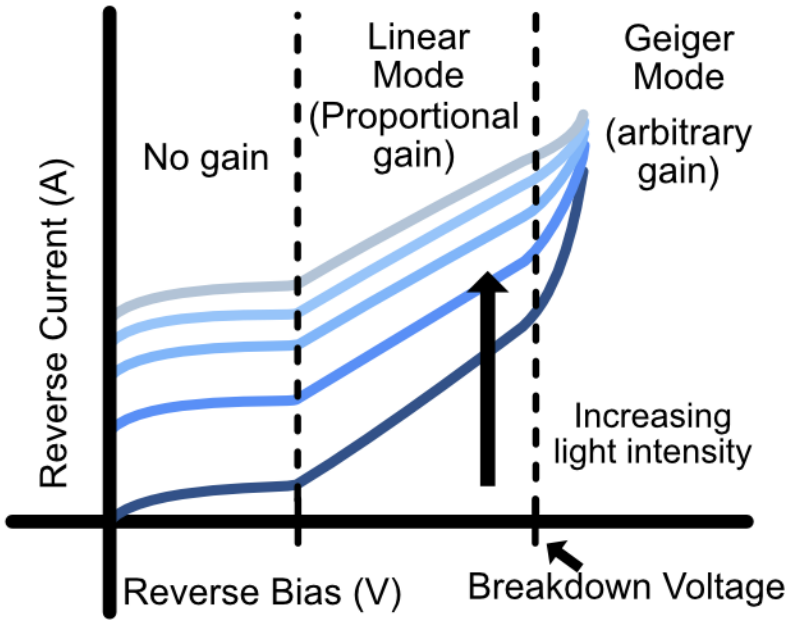


Figure 10 – Photodiode I-V curve rotated to show reverse bias region in the right quadrant.

As an APD operates at relatively low reverse bias voltages, it performs similarly to a general PN photodiode and is said to be operating in photoconductive mode, where the depletion region increases with increasing reverse bias voltage. As the reverse bias voltage increases and nears the breakdown voltage, the APD begins to operate in linear mode, where the avalanche gain of an APD can vary from around 10 to 1000 and can be amplified with sensitive electronics and used to trigger photon counting instrumentation. This region of APD operation below its breakdown voltage is known as the linear region because there is a proportional output to incident light. The effects of noise within the signal chain can make the single photon level output from an APD difficult to resolve. Detecting 10 to 1000 electrons is often as hard as detecting a single photogenerated electron. Another method of APD operation is Geiger-mode, where the APD is biased beyond its breakdown voltage. The three main regions that the APD can operate are shown in Figure 10, where region 1 is the photoconductive mode with reverse

biased applied and a gain of 1, region 2 is linear-mode where typical APDs are operated with gains of 10-1000, and region 3 is Geiger-mode where the APD is biased beyond breakdown.

2.5 Geiger-Mode APD

When operated in Geiger-mode, a current limiting resistor, called a quenching resistor is needed to prevent damage due to excess current flow from breakdown events. The APD breaks down, current flows through the quenching resistor and drops the voltage across APD until the breakdown is stopped. This process repeats and the output from a Geiger-mode APD is a train of short pulses with a mostly fixed amplitude determined by the bias voltage beyond breakdown and the value of the quenching resistor.

The concept of biasing "beyond breakdown" often causes confusion with the assumption being that a device in breakdown has a permanent short circuit current flowing. Even with a quenching resistor, the intuitive understanding of breakdown would lead to the assumption that the Geiger-mode APD would toggle between the two states as fast as the quench resistor and photodiode capacitance would allow. However, breakdown is a probabilistic process and the probability of breakdown increases with incident light on a Geiger-mode APD. This is the reason why an APD biased beyond breakdown outputs a pulse rate that is proportional to the light intensity. There are also some breakdown pulses even without light and this is known as the "dark count". If the dark count is sufficiently low, then a small change in count rate due to a

single photon can be detected. Even if the dark count is high for a detector, a single photon can still be resolved if the same experiment is repeated thousands of times and averaged.

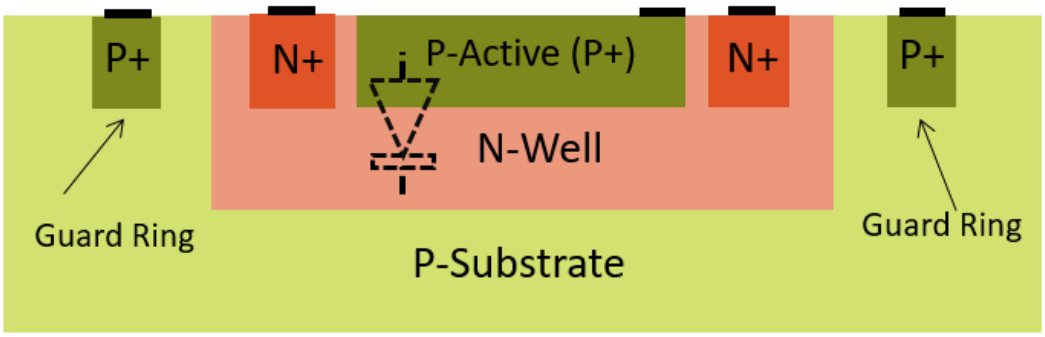


Figure 11 – APD cross section

A Geiger-mode APD can be easily created from the layers available in standard CMOS processes with no modifications. The cross-section of an elementary Si CMOS APD is shown in Figure 11. The PN junction is formed from the N-well used for PMOS transistors and a square or circle of P+ active material usually used for source/drain terminals. Metal contacts to the N-well and the P+ are then used to connect the APD to either bond pads or other circuitry on the chip. It is important to note that if the CMOS process is silicided, a silicide block layer must be used in layout to cover the P+ region. Otherwise, silicide deposited on the photoactive region will block light from hitting the junction.

The APDs formed in CMOS processes can be operated in either linear or Geiger mode. However, the linear multiplication region is very small due to the sharp breakdown characteristics of the heavily doped CMOS process layers. As a result, it is difficult to bias the APD in the linear region. This is why Geiger mode operation is preferred for a CMOS APD. Many variations in terms of APD size, shape, and guarding can be done to optimize the

performance and tailor the breakdown characteristics of the Geiger-mode CMOS APD. For example, using a circular structure instead of a square structure can reduce breakdown effects at the edges leading to reduced dark count rates.

2.6 SiGe APDs

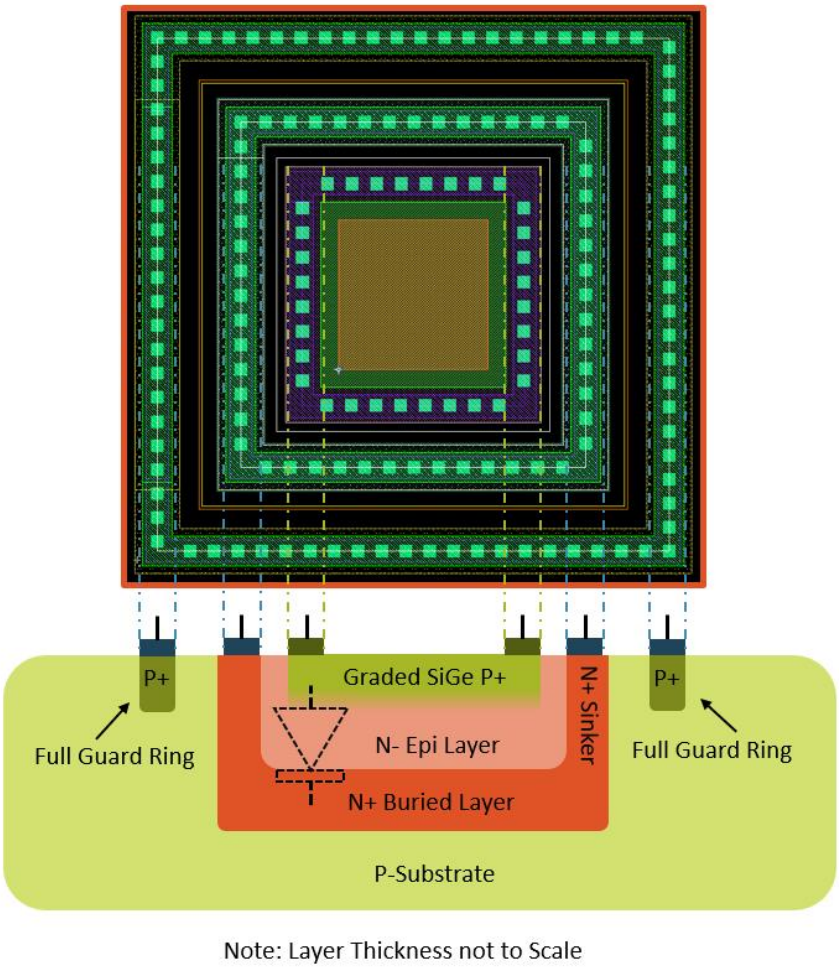


Figure 12 – SiGe APD cross section and layout

A variant on the Si APD formed in CMOS processes is the SiGe APD formed using layers found in SiGe BiCMOS processes. SiGe BiCMOS processes use a graded heterojunction SiGe base layer to create high heterojunction bipolar transistors (HBT). The SiGe base layer can be exposed to light by omitting the polysilicon emitter. The SiGe base layer then forms the anode

of the APD. The deep N-well originally used for the collector is then repurposed as the cathode of the APD. The basic structure is then improved with uniform contacts to the anode and cathode regions. These SiGe APDs can be shaped, resized, and have guard rings added just like their Si CMOS counterparts. A representative cross-section of a SiGe APD is shown in Figure 12. The main focus of this thesis is testing these SiGe APD structures to determine what parameters make an optimal SiGe APD.

The SiGe APD structures can be broadly classified into two shapes, circular and square. Among these shapes, there are APDs with sizes (defined as width of square or diameter of circle) of 5 µm, 24 µm, and 50 µm. Examples of these structures are shown in the figures below. The layouts are a part of a broad research investigation of SiGe APDs [11], where the devices chosen function properly and will be tested using the system developed in this thesis.

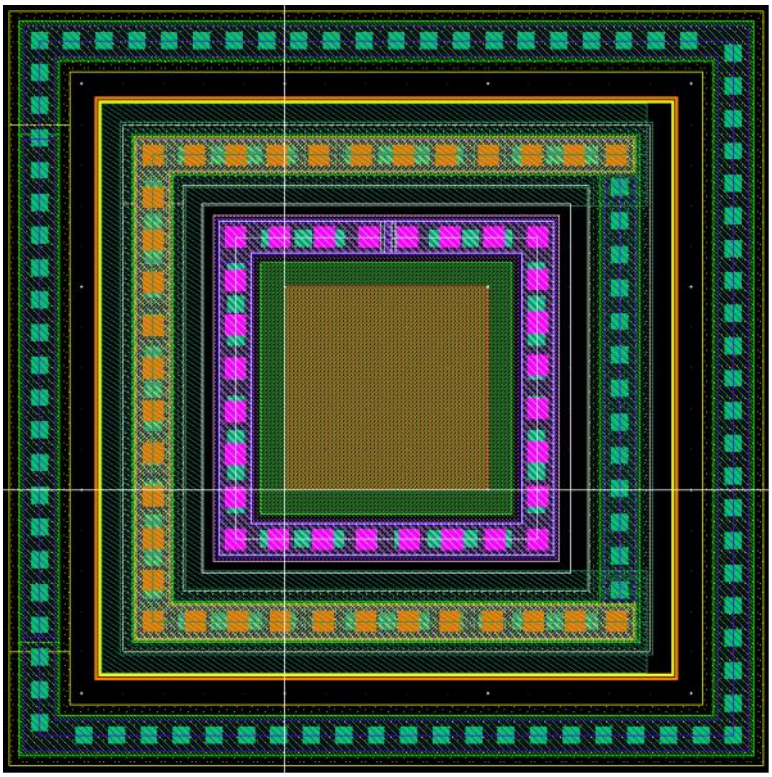


Figure 13 – 5µm Square SiGe APD

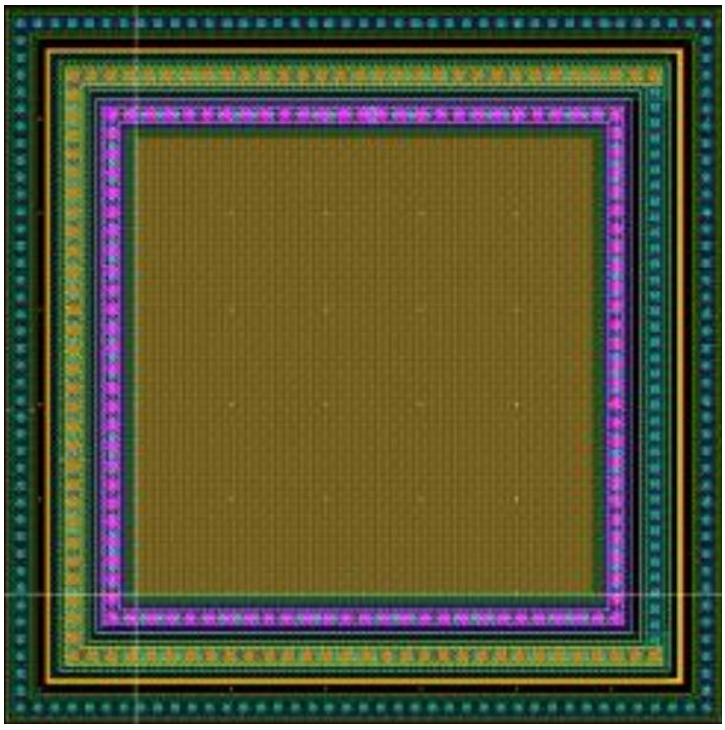


Figure 14 – 24μm Square SiGe APD

Figure 15 – 50μm Square SiGe APD

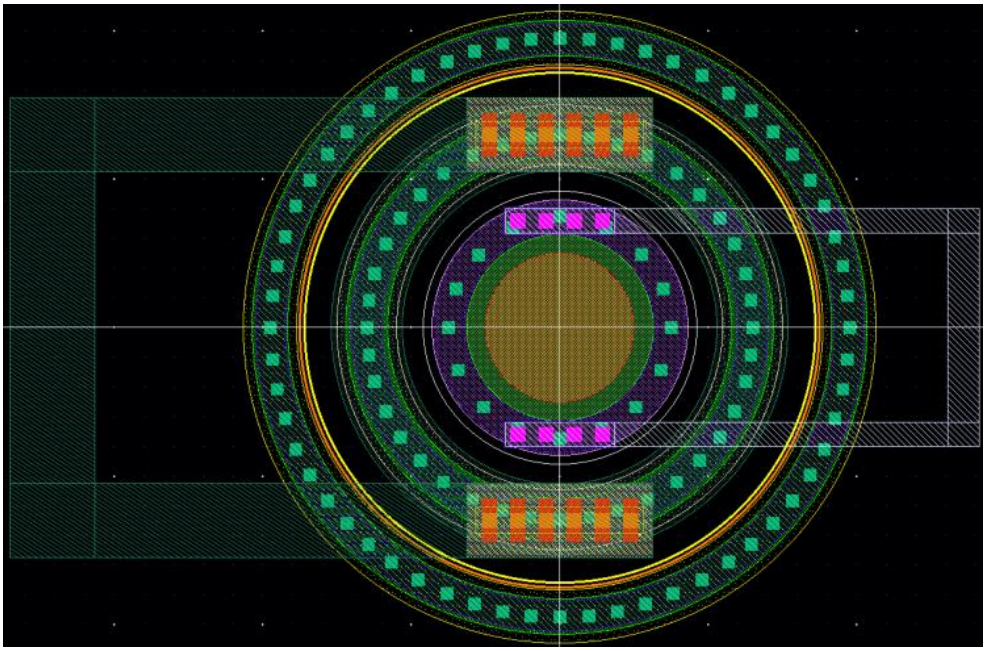


Figure 16 – 5µm Circle SiGe APD

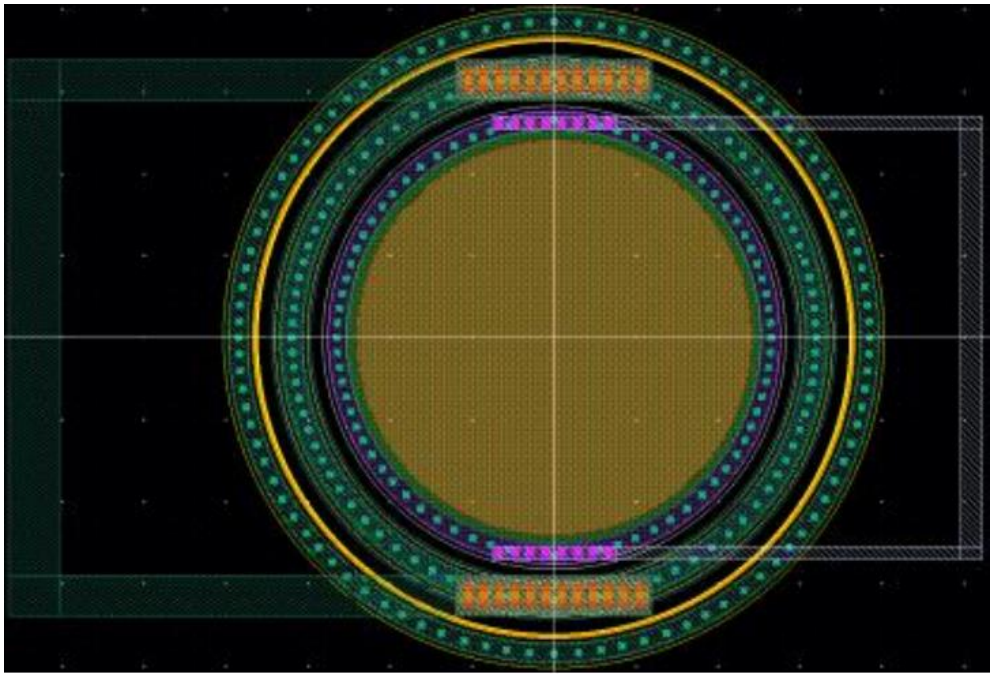


Figure 17 – 24µm Circle SiGe APD

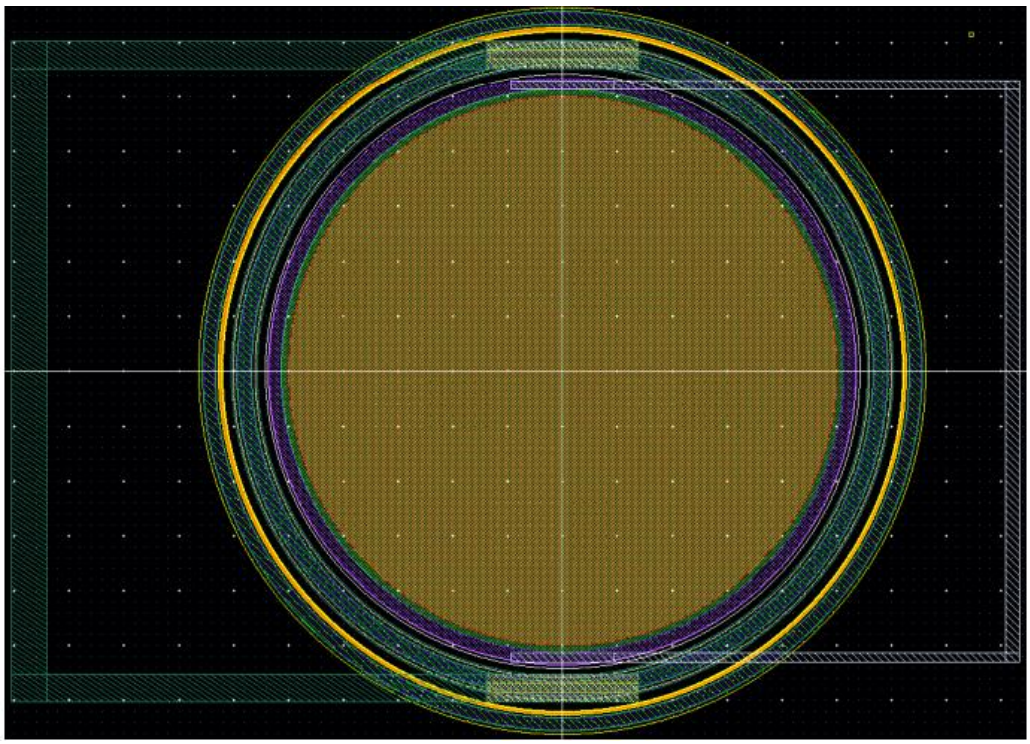


Figure 18 – 50μm Circle SiGe APD

Chapter 3 Understanding Photon Counting Instrumentation

3.1 Photon Counting Experimentation Setup

For this thesis, there are a few metrics to discuss before proceeding to conduct photon counting tests. When performing a single test, a laser diode at a specific wavelength of visible light is applied to an avalanche photodiode (APD). The circuit under test is an APD in series with a quenching resistor and a termination resistor, where the quenching resistor will limit the current coming from a voltage supply, shown in Figure 19 in a schematic. The APD is configured with the cathode connected to the quenching resistor and the anode connected to the termination resistor. The APD is inserted into an integrating sphere that contains the laser diode that will cause the inside of the sphere to equally diffuse light onto the APD. The anode of the APD is connected to a low-noise amplifier that is used to amplify short current pulses and convert it into a measurable voltage that will feed into the SR430 photon counter.

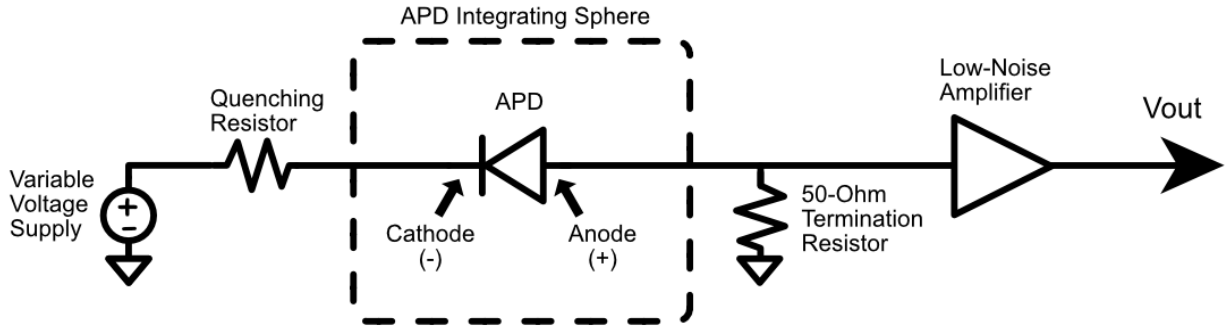


Figure 19 – APD Test Setup (schematic)

A PCB board was designed by a colleague, seen in Figure 20, where the variable voltage supply feeds into a BNC connector that is on the cathode side, and the output is the SMA

connector on the anode side. The APD is on the bottom side of the PCB board, shown in Figure 21.

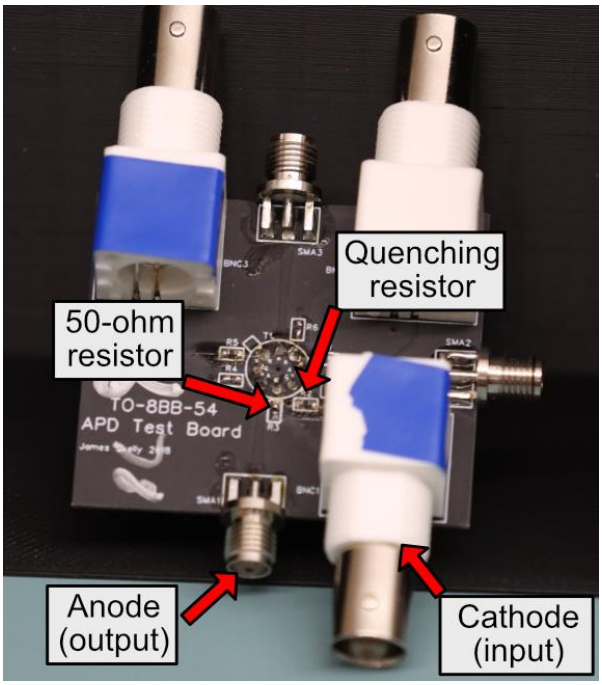


Figure 20 – APD Test Board (top view)

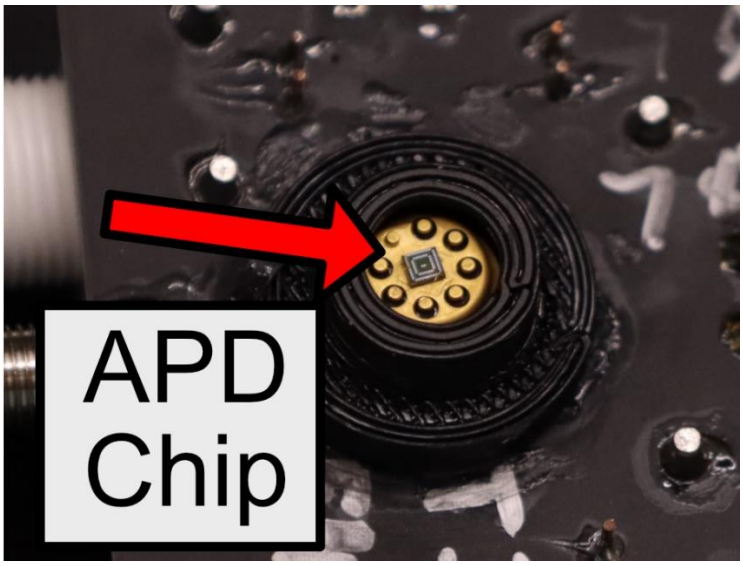


Figure 21 – APD Test Board (bottom view)

The SMA output feeds into a Mini-Circuits ZFL-500LN-BNC+ amplifier, shown in Figure 22, which is powered with an external voltage supply of 15V, has a bandwidth up to 500MHz and amplifies the incoming signal by around 28dB (linear gain of 25V/V).

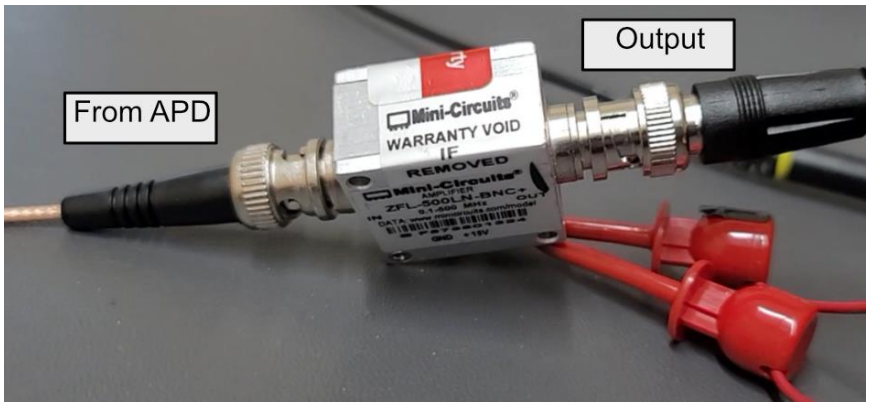


Figure 22 – Mini-Circuits ZFL-500LN-BNC+ Low Noise Amplifier (LNA)

The laser diode is attached to the integrating sphere, where the laser is enabled using a current source, shown in Figure 23.

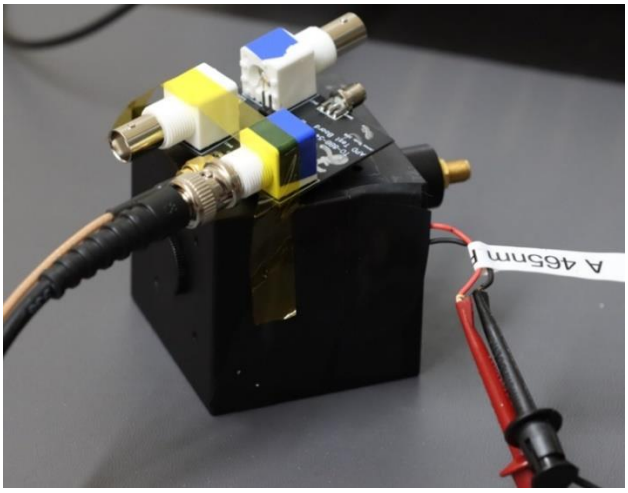


Figure 23 – Integrating Sphere

The setup explained in this section is the minimum requirement to proceed with testing. The following sections and chapters will discuss how tests are conducted and the values obtained.

3.2 Testing with General Purpose Instruments

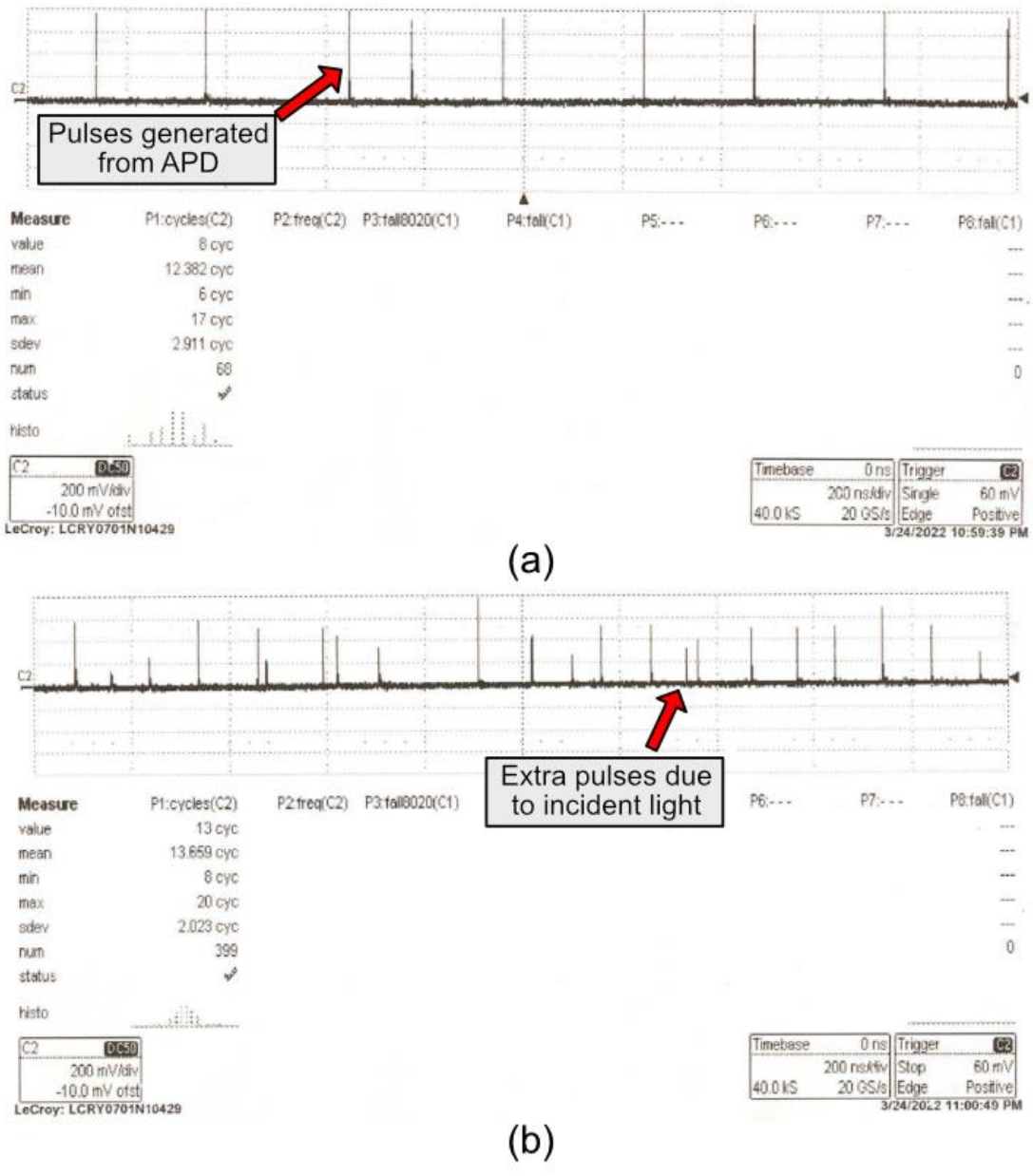


Figure 24 – Observing pulses with LeCroy Oscilloscope (a) Dark Counts (b) Light Counts

It is possible to perform photon counting with general purpose laboratory instruments such as oscilloscopes or pulse detectors. One such way to test the APD for current pulses is by probing the output and viewing the output on an oscilloscope and making observations. By following Figure 19, setting the voltage supply to a voltage higher than the breakdown voltage of the APD and connecting the output to an oscilloscope, the output measured on an oscilloscope can be seen in Figure 24 showing noticeable voltage pulses. With the laser diode disabled, this measured output will be considered pulses that are generated under the absence of light, or dark counts, shown in Figure 24(a). Enabling the laser diode will cause the APD to generate more pulses, indicating that by applying light onto the photodiode will generate more counts, which will be defined as light counts, shown in Figure 24(b). If one were to manually count the number of pulses generated from the dark and light counts, the difference between these two measured outputs will be the extra counts produced due to applying incident photons to the APD, which are the photon counts. Although it is possible to manually experiment with an oscilloscope, it would be deemed impractical to experimentally measure results due to user errors of managing the oscilloscope and discriminating on what can be considered a pulse.

3.3 Survey of Dedicated Photon Counting Instruments

Conducting photon counting testing with everyday lab instruments is a feasible option, however, it would be deemed a tedious task to processing incoming signals for specific measurements such as peak voltages and average photon counts over a given time frame. A dedicated photon counting instrument can discriminate between noise and voltage pulses, and store individual counts into a dedicated time bin. The discriminator voltage of a photon counter is defined as the level at which a voltage pulse can be distinguished as a define count. Accompanying this discriminator voltage is the discriminator slope, where the photon counter

will define a count based on the direction of the voltage pulse. Figure 25 shows an example of how a photon counter would discriminate between a pulse being considered a count, where Figure 25(a) uses a positive discriminator slope to count positive voltage pulses and similarly for negative voltage pulses in Figure 25(b).

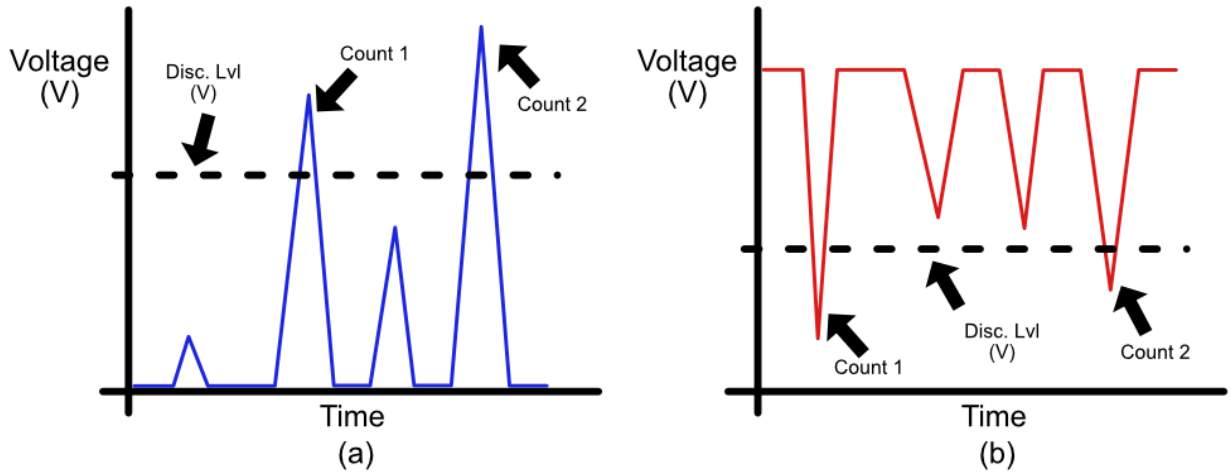


Figure 25 – Discrimination of a count (a) Positive slope (b) Negative slope

A time bin is the time that a bin is exposed to collecting data. While exposing a bin, all photon counters will have a maximum count per bin before the photon counter becomes saturated. A record is a collection of all time bins that sums the count data of all bins. If the maximum bin count has not been met, a test may be running where multiple records may be recorded and summed together. Depending on the number of records, the rate at which each record is triggered, and the time bin size, an accurate count of photons may be collected and analyzed using the photon counter's data processor. The following section will discuss the operation of the SR430.

3.4 Stanford Research Systems SR430

The main photon counter that has been used extensively throughout this thesis is the SR430, featured in Figure 26. The added benefits of this photon counter are the upgraded user interface and data processing options. One of the foundations of the SR430 is the use of time bins that collect counts. The minimum time bin width of the SR430 is 5ns, however, the maximum count rate is 100MHz, or one count per 10ns. The next largest time bin is 40ns, therefore, each bin can record up to 4 counts. The next time bin is 80ns, and each consecutive time bin doubles, up to 10.486ms. For this thesis, a time bin of 160ns is used, therefore 16 counts can be recorded in each bin. A record is a collection of time bins, where there is no dead time between bins and all counts can be collected. The number of bins that can be used in a record range from 1024 to 16,352, in increments of 1024. The length of each record in terms of time is determined by multiplying the time bin width by the number of bins in a record. The SR430 has a feature that can scan a record multiple times and sum all counts into each bin. A scan can accumulate a range of records between 1 and 64k.

Figure 26 – The SR430 Photon Counter

Shown in Figure 27 is the SR430's process of collecting counts on a time diagram [Stanford Research Systems, p.12]. The time it takes to do a scan with multiple records will change as a function of the length of bins and number of bins used, plus accumulation and processing time to add all counts. Equation 3.1 can be used to mathematically solve for the busy time it takes to process one record, T_{busy} , where T_{bin} is the time bin and N is the number of bins per record [Stanford Research Systems, p.11]. A record is triggered using a trigger pulse, which can be generated using a square pulse generator that serves as a clock. The frequency of this square pulse is the reciprocal of the busy time, and at each trigger a new record will be initiated. If the trigger pulse frequency is higher than the maximum frequency calculated, the SR430 will throw a trigger rate error, which indicates that the SR430 detected a trigger pulse while the record was still being processed and will be flagged and ignored. Once the trigger frequency has been defined, the length of a scan can be calculated by taking the number of records and multiplying this to the busy time, demonstrated in Equation 3.2.

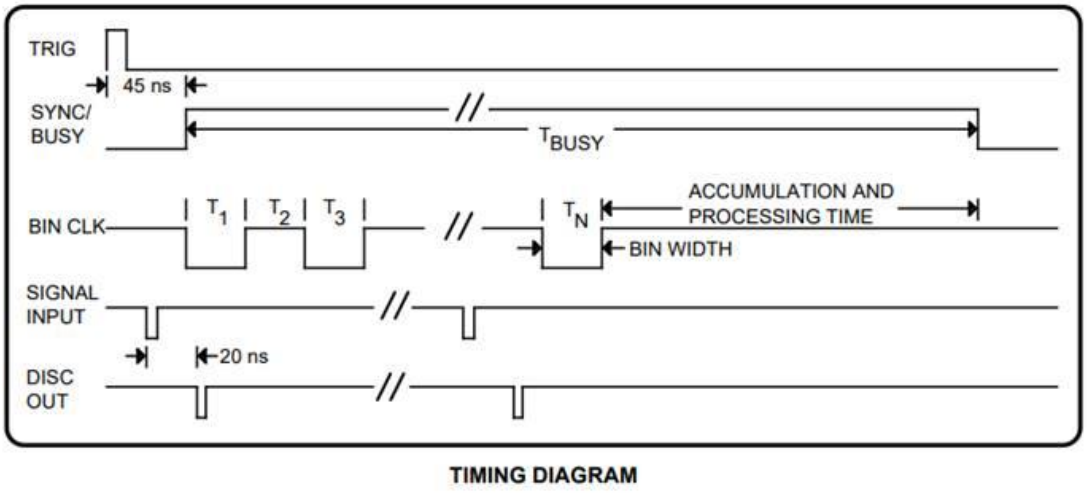


Figure 27 – Timing Diagram of the SR430 for one record

$$T_{busy} = N \cdot T_{bin} + N \cdot 250ns + 150\mu s, \qquad f_{clk} = 1/T_{busy}$$

Equation 3.1 – Solving for T_{busy} and trigger clock input, f_{clk}

$$T_{scan} = \#\ of\ Records \cdot T_{busy}$$

Equation 3.2 – Total scan time of the SR430

To access the SR430's time bins and record lengths, this is found by pressing on the "Mode" key, shown in Figure 28. The mode menu contains all the main photon counting functions of the SR430. The SR430 can operate with either an internal bin clock that uses dedicated bins or an external bin clock where the user can define their own bin size. In this thesis, the bin clock source is set to "internal". To change a value on the SR430, the dark softkeys can be pressed to access menus seen on the screen and the spin knob can be turned to set a numerical value.

(a)

(b)

Figure 28 – Mode menu on the SR430 (a) Accessing menu (b) Mode Menu

A similar process to Figure 28 to access the levels menu of the SR430 by pressing on the "Levels" key. The levels menu will be the main testing menu where parameters such as discriminator level can be changed. To setup the SR430 for photon counting, the parameters shown in

Table 1 are set.

Levels Menu		
Parameter	**Set Value**	**Notes**
Trigger Level	1.25 V	This is the voltage level to cross to trigger a record scan. Can be a value from -2V to 2V in 1mV increments.
Trigger Slope	Positive	This parameter will set the trigger slope. A trigger will occur when the clock pulse passes the trigger level with the set trigger slope. Can also be negative.
Discriminator Level	200mV	This is the voltage level that the incoming signal should pass to be considered a count. Can be a value from -300mV to 300mV in 0.1mV increments.
Discriminator Slope	Positive	This parameter will set the discriminator slope. A count is recorded when the incoming signal passes the disc. level with the set disc. slope. Can also be negative.
Mode Menu		
Bin Clock Source	Internal	Determines if either internal or external time bins will be used. The recommended setting is to set to "internal".
Bin Width	160 ns	The length of each bin. Can be a short value of 5ns, or 40ns, 80ns, doubling up to 10.486ms. Note that longer time bins may cause the bin to saturate easier (max counts per bin: 32,767, regardless of bin size)
Bins/Record	1k (1024)	The number of bins per record. There is no dead time between bins. Can be a value between 1k up to 16k, in increments of 1k.
Records/Scan	10000	The number of records per scan. The current record is added to the next record, where bin positions are added together. Can be a value between 1 to 64k. Note that more records will have a chance of saturating bins.
Trigger Offset	0	The number of time bins to add after a trigger that will not be counted. This is used if the user needed a delay time before testing.

Table 1 – SR430 Main Parameter Setup

3.5 Manual Photon Counting Experimentation

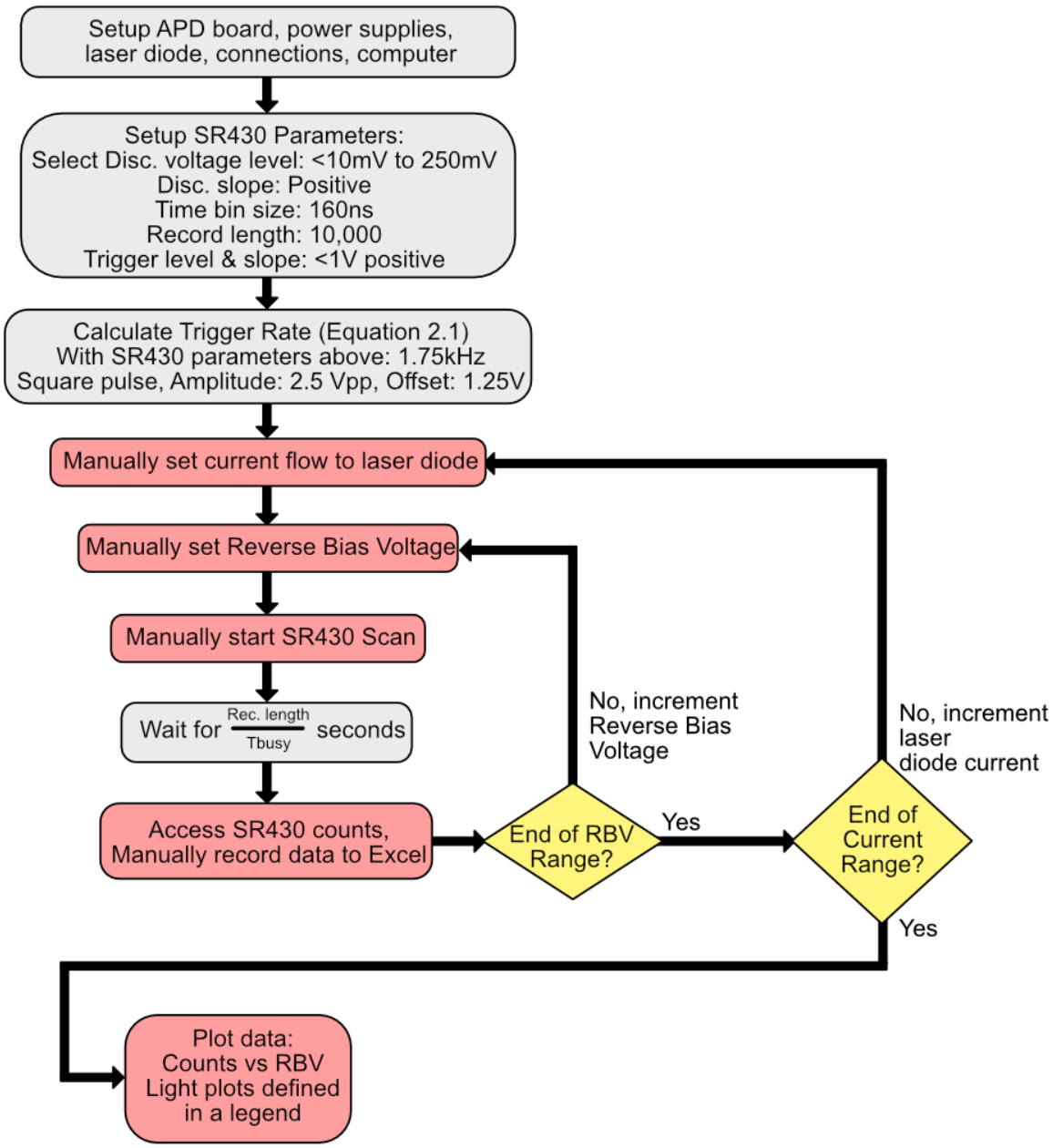


Figure 29 – Manual Photon Counting Flowchart

The process of conducting photon counting experiments is shown in Figure 29. A majority of this chapter has established how to setup the APD board, main connections, and

SR430 setup. Once the user has verified all instrumentation and circuitry, the first step is for the user to manually set the current flow through the laser diode using a current source. A dark counting test is done when the laser diode is off, and a light count consists of setting a desired current to produce counts due to incident photons, or photon counts.

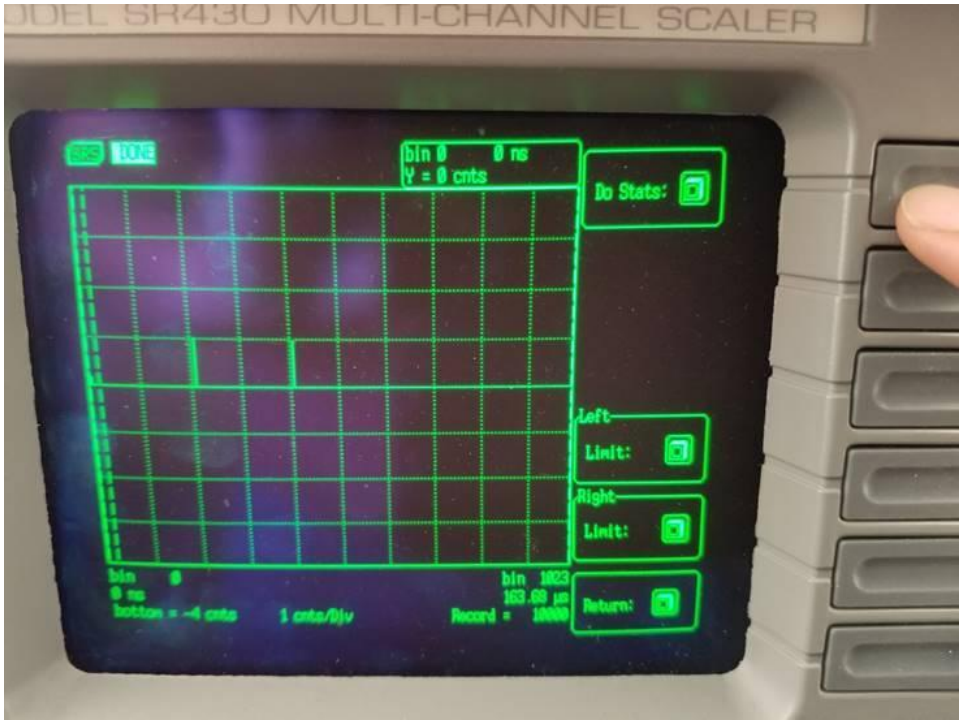
After the laser diode current is set, the reverse bias voltage is manually set. It is preferred that the voltage be set near the breakdown voltage to minimize user workload. To conduct a count, the user will initiate a scan by pressing on the “Stop/Reset” button twice to stop any processes on the SR430 and to reset all count data, seen in Figure 30. The “Start” button is located to the left of the “Stop/Reset” button. If all is done correctly, at the breakdown voltage there should be a few counts produced on screen, shown in Figure 31. If there is nothing on screen, the user may press the “Auto Scale” button to scale the vertical access or can horizontally zoom out by pressing the “Display” button and zooming out.

Figure 30 – Resetting SR430 count data

Figure 31 – Detecting pulses on the SR430

To access the current count data, the “Math” button is pressed to access the math menu of the SR430. Selecting “Stats” followed by “Do Stats”, shown in Figure 32, using the softkeys will provide all the count data that can be recorded manually into Excel. Figure 33 shows all statistical values, where the total counts can be found.

(a)



(b)

Figure 32 – Accessing math menu (a) Math menu (b) Statistics menu

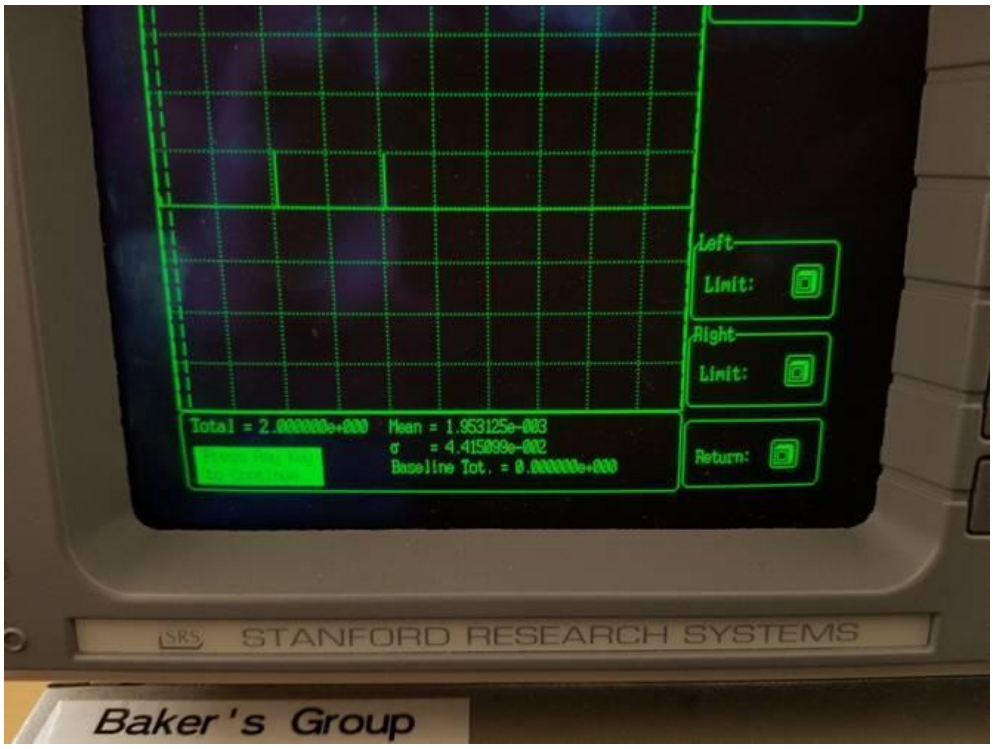


Figure 33 – Obtaining number of counts from the SR430 (counts = 2)

After data is recorded, the user resets all SR430 count data using the “Stop/Reset” button, manually increments the voltage supply, rescans, and repeats until the maximum desired voltage range has been satisfied. This recording process will take a duration of around 30 seconds for each count data, where the time takes into consideration user actions and awareness. The laser diode current can be incremented to emit more incident photons and conduct light counts. The process of scanning repeats until the laser diode current has been satisfied.

3.6 Manual Photon Counting Results

The data results of a sample APD counting experiment are shown in Figure 34. The breakdown voltage of the APD begins at 11.14V, where at this voltage, the SR430 has not detected any pulses while the APD is applied both a dark state and a laser wavelength of 530nm with a current of 20mA. Figure 35 is a graphical representation of the data in Figure 34, where

with manual experimentation, there is a noticeable increase in voltage increments at higher voltages. The difference between the light and dark counts is shown in Figure 36, where the data shows that the difference between the counts is positive and shows that incident light produces more counts.

	A	B	C	D	E
1	DARK (200mV Disc Limit) – Voltage	Total	Photon Counts	LIGHT (530nm; 20mA) (200mV Disc Limit) – Voltage	Total
2	11.14	0	0	11.14	0
3	11.15	1.00E+00	9.84252	11.15	3
4	11.16	1.20E+01	24.6063	11.16	1.70E+01
5	11.17	2.40E+02	457.67718	11.17	3.33E+02
6	11.18	2.27E+03	2608.2678	11.18	2.80E+03
7	11.19	9.62E+03	7701.7719	11.19	1.12E+04
8	11.2	2.55E+04	22701.77238	11.2	3.01E+04
9	11.225	3.93E+04	36599.41062	11.225	4.67E+04
10	11.25	4.57E+04	46904.52906	11.25	5.52E+04
11	11.275	5.20E+04	34296.26094	11.275	5.90E+04
12	11.3	5.55E+04	48612.20628	11.3	6.53E+04
13	11.325	6.36E+04	57854.33256	11.325	7.54E+04
14	11.35	7.14E+04	55698.82068	11.35	8.27E+04
15	11.375	7.88E+04	51446.85204	11.375	8.93E+04
16	11.4	8.40E+04	62942.9154	11.4	9.68E+04
17	11.425	9.16E+04	63572.83668	11.425	1.05E+05
18	11.45	1.01E+05	63105.31698	11.45	1.14E+05
19	11.475	1.09E+05	62903.54532	11.475	1.22E+05
20	11.5	1.16E+05	64370.47452	11.5	1.30E+05
21	11.55	1.32E+05	65930.12022	11.55	1.46E+05
22	11.6	1.50E+05	74606.3016	11.6	1.65E+05
23	11.65	1.68E+05	57923.2302	11.65	1.80E+05
24	11.7	1.88E+05	69990.15972	11.7	2.02E+05
25	11.75	2.07E+05	60949.8051	11.75	2.19E+05
26	11.8	2.30E+05	47716.53696	11.8	2.40E+05
27	11.85	2.47E+05	61569.88386	11.85	2.60E+05
28	11.9	2.66E+05	54301.18284	11.9	2.77E+05
29	11.95	2.87E+05	53651.57652	11.95	2.98E+05
30	12	3.07E+05	44158.46598	12	3.16E+05
31	12.25	4.05E+05	51786.41898	12.25	4.16E+05
32	12.5	4.99E+05	35305.11924	12.5	5.06E+05
33	12.75	5.89E+05	60610.23816	12.75	6.01E+05
34	13	6.80E+05	48011.81256	13	6.90E+05
35	13.25	7.66E+05	53149.608	13.25	7.77E+05
36	13.5	8.51E+05	45280.51326	13.5	8.60E+05
37	13.75	9.38E+05	21437.00856	13.75	9.43E+05
38	14	1.02E+06	67042.32498	14	1.03E+06
39	14.25	1.10E+06	40639.76508	14.25	1.11E+06
40	14.5	1.18E+06	36038.38698	14.5	1.19E+06
41	14.75	1.26E+06	24271.65432	14.75	1.27E+06
42	15	1.34E+06	32672.24514	15	1.35E+06
43	15.25	1.41E+06	45580.71012	15.25	1.42E+06
44	15.5	1.49E+06	63371.06502	15.5	1.51E+06
45	15.75	1.57E+06	45738.19044	15.75	1.58E+06
46	16	1.64E+06	48794.2929	16	1.65E+06

Figure 34 – Example of manual counting result data

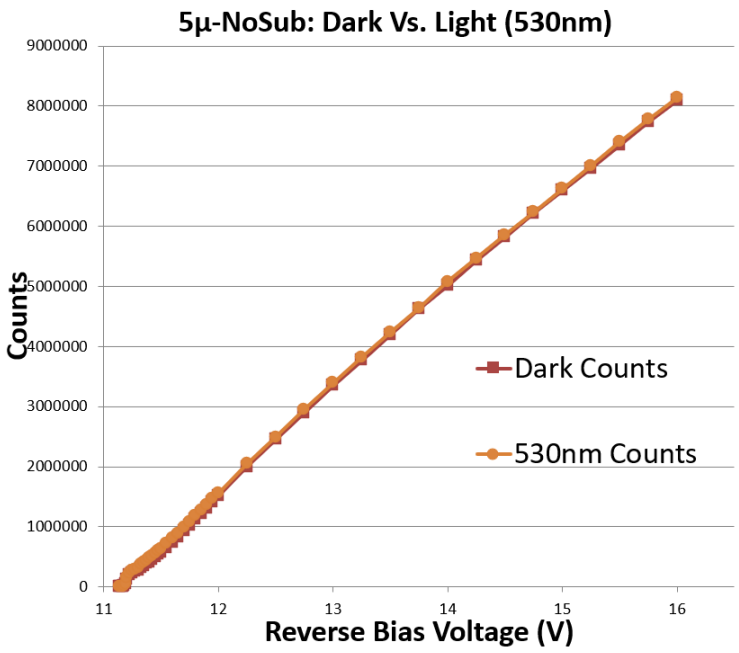


Figure 35 – Plotting results of manual counting test

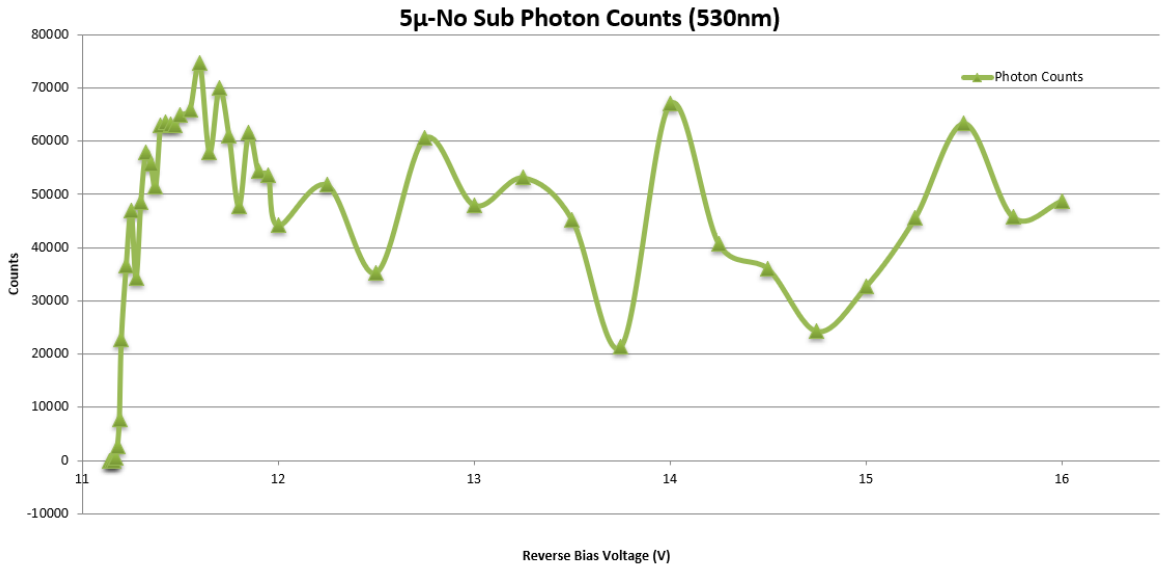
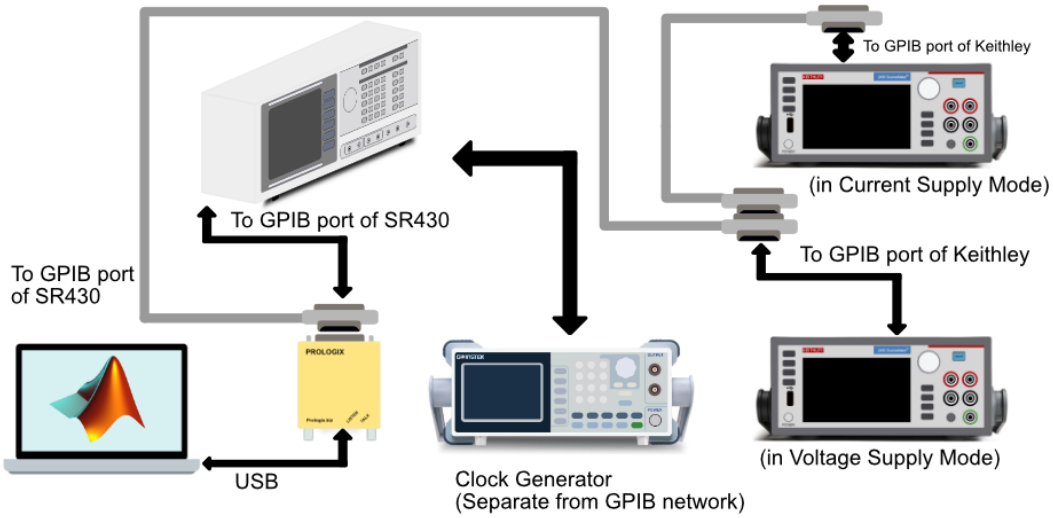
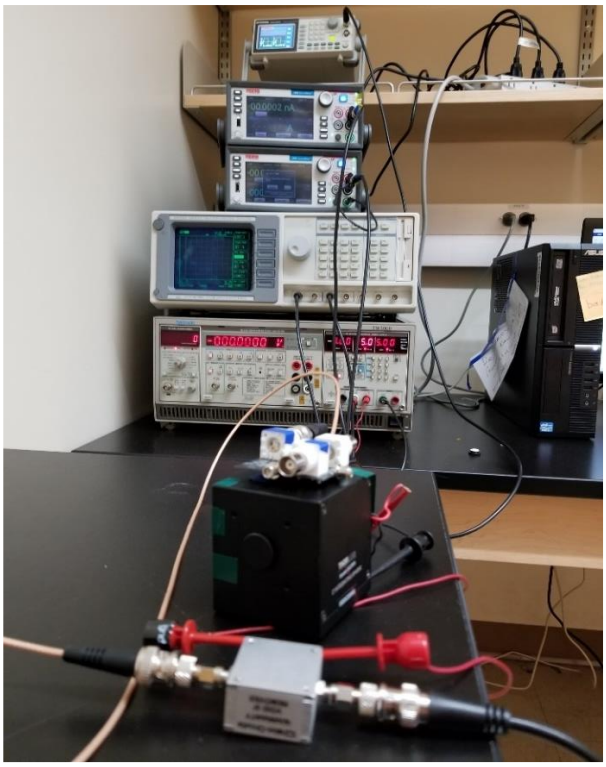


Figure 36 – Light Counts minus Dark counts of manual counting test

Altogether, there are 90 points of data collected, where each point takes roughly 30 seconds to record. The total duration of the experiment is roughly 45 minutes, which for one APD is not so much time, however, will prove to be a tedious task if multiple experiments are to be performed, such as changing the intensity of the laser diode. The conclusion for this experiment is that manual counting is a feasible option, however, it would be ideal to develop a system that would take on the load of performing experiments automatically, better yet, with higher precision and more unique operating points. The following two chapters will go over a method of performing these same experiments using a communication bus system and MATLAB that will speed up this process of experimentation.



(a)



(b)

Figure 37 – GPIB System developed in Thesis (a) Diagram (b) Physical Setup

The process of using all the instrumentation mentioned in chapter 3 requires a user to conduct the experiments manually. Any manual process will introduce errors (mostly user-based errors such as timing) and it would be desirable to minimize errors using an automatic process. One such process that will assist in conducting experiments in an efficient way is using an established communication system that would control instruments via a remote controller (such as a computer), shown in Figure 37.

There are two established connection systems that equipment manufacturers may install into test instruments: a General-Purpose Interface Bus (GPIB or IEEE-488) and a Recommended Standard 232 (RS-232) connection system. The objective that each system achieves is to communicate to instruments using Standard Commands for Programmable Instruments (SCPI, often pronounced “skippy”) commands that the manufacturer sets for each instrument and provides materials for the user. This thesis will go in depth on the basics of GPIB, the setup process of how to connect to an instrument, process SCPI commands to achieve communication, and an efficient solution that will create an automated process for retrieving data.

4.1 General Purpose Interface Bus Connection System

GPIB (originally developed as *Hewlett-Packard Interface Bus*) was developed during the 1960s by Hewlett-Packard as a means to create a communication system between instruments and controllers. As many manufacturers began developing more digital testing instruments, the desire to create an interface standard increased. There was an all-out effort in which the United States and Germany during the early 1970s worked on an international digital instrumentation standard. By 1975, the U.S. with the help of members of the Institute of Electrical and Electronics Engineers (IEEE) prepared a document which will be intended as an IEEE standard known as the IEEE Std 488-1975, “Digital Interface for Programable Instrumentation”. The

document aided in establishing a national standard for communication between instrumentation, and later would be published as an international standard [7].

4.2 GPIB Communication

GPIB devices can either be “Talkers”, “Listeners”, and/or “Controllers”. In general, power supplies with GPIB can be both a talker and a Listener, where a power supply may listen for a command (such as changing a supply value) and can talk about the status of the supply (such as calling out its current supply value). A controller is what controls what information is sent on the GPIB (such as addresses) and acts as a switch to send commands to specific instruments.

There are two configurations in which GPIB devices can be configured. A simple and hardware-efficient way of connecting devices is in a linear configuration, where all devices connect to a single line and are daisy-chained together. A more dedicated form of connecting devices is via a star configuration, where each device is connected to a single node (either tying all devices together or directly into a computer). A mix of the two can also be done if the length of the bus between the device and controller is under 20 meters (Wiki). Figure 38 shows the possible types of configurations, where all configurations have no technical advantage over each other and is dependent on personal preference.

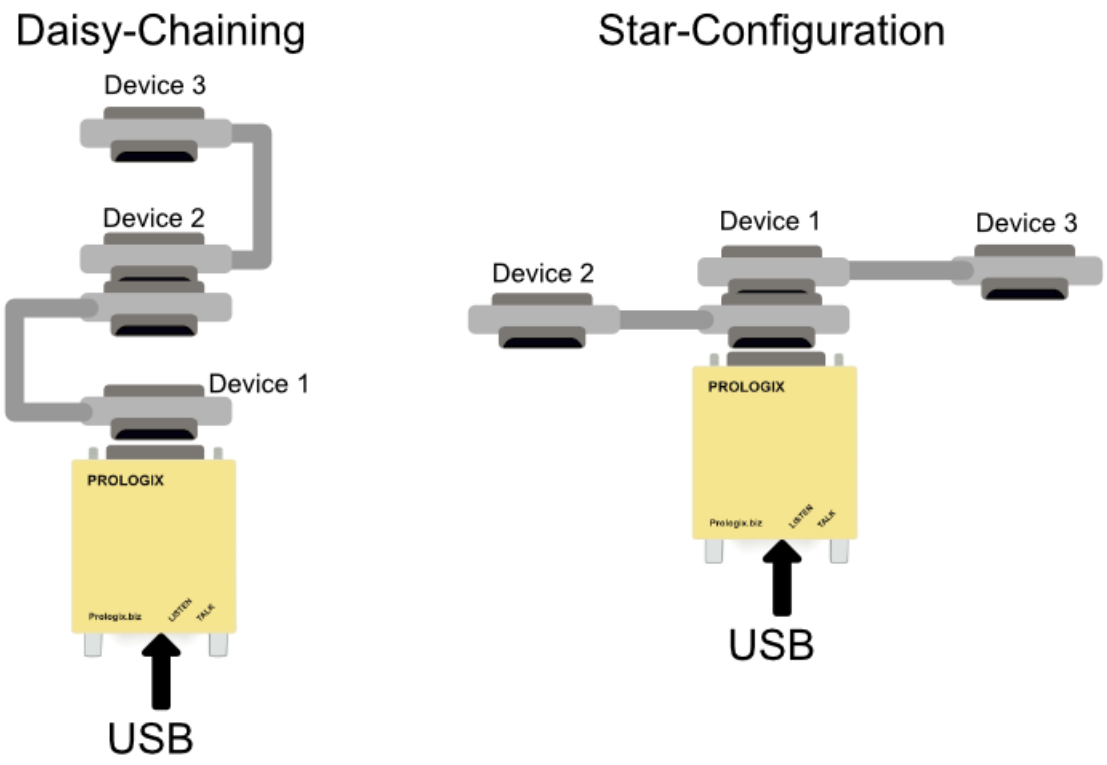


Figure 38 – Daisy Chaining vs. Star-Configuration

There are 31 device addresses that the bus can access, where each device can be assigned a value between 0 and 30. The controller can only access one address at a time, and multiple listeners may have the same address (configured on the instrument), so long as these devices act as listeners. There can also be multiple controllers on the bus, however, only one controller may be considered the “controller-in-charge”.

4.3 Prologix GPIB-USB Controller

(a) (b)

Figure 39 – Prologix GPIB-USB Controller (a) GPIB port (b) USB-B port

A GPIB controller is a special device that connects multiple GPIB-enabled devices to a computer that also has its own set of commands that can override devices on the bus line [12]. For this thesis, a Prologix GPIB-USB (ver. 6) controller is used to communicate with all the devices on the line, shown in Figure 39. The Prologix controller uses a double plus sign command, "++", to denote that the command is directed at the controller. The controller can be used in either a Controller-In-Charge (CIC) mode or in Device mode. While in CIC mode, the controller expects a SCPI command from the computer to the controller which is terminated with either a Carriage Return (CR) or a Line Feed (LF). The controller then will address a GPIB device at the current specified address and sends the SCPI command that was sent from the computer.

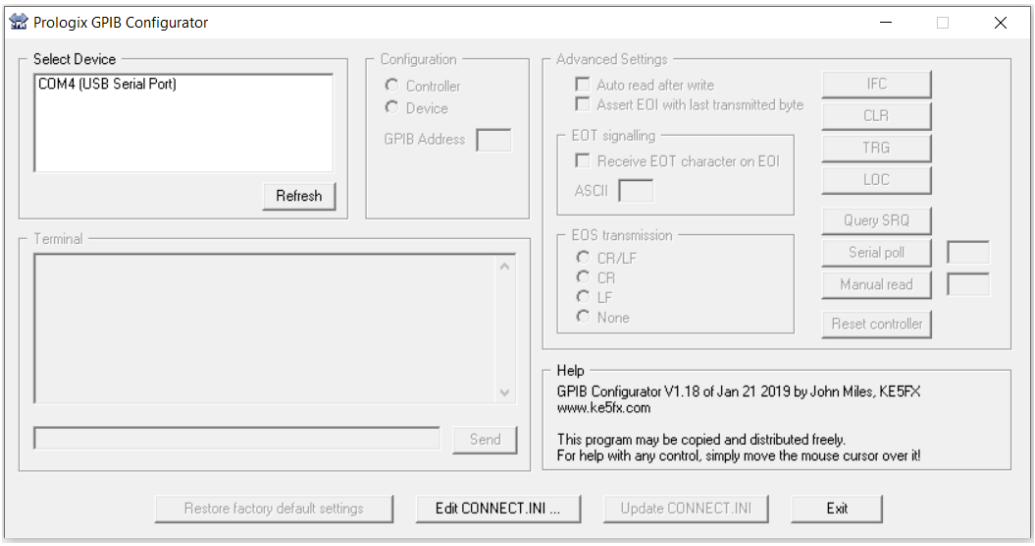
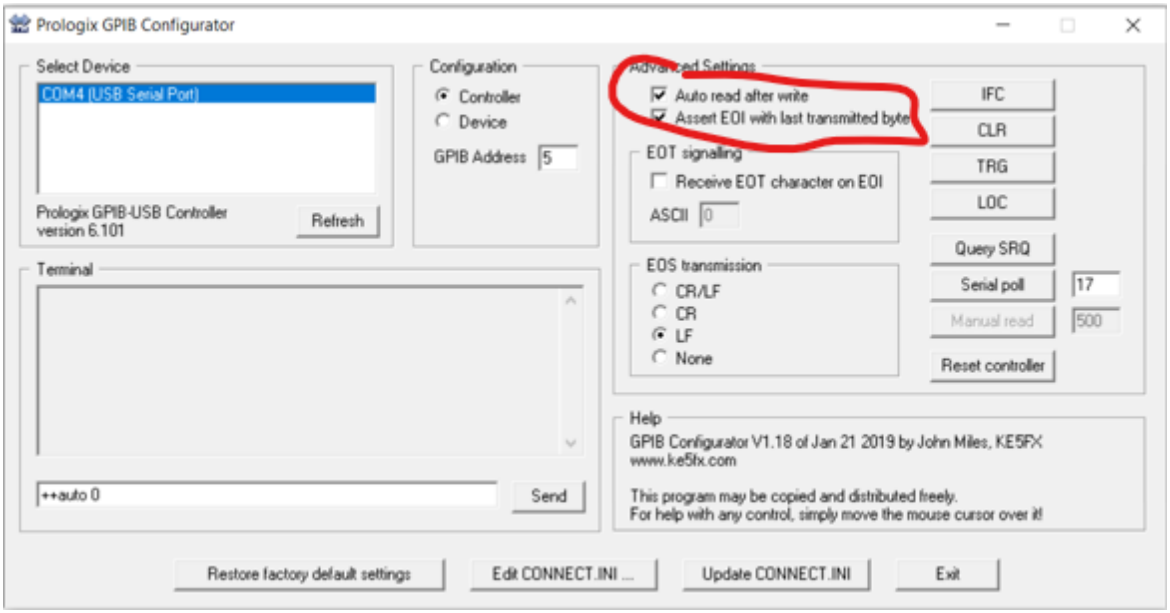


Figure 40 – Prologix GPIB Configuration application

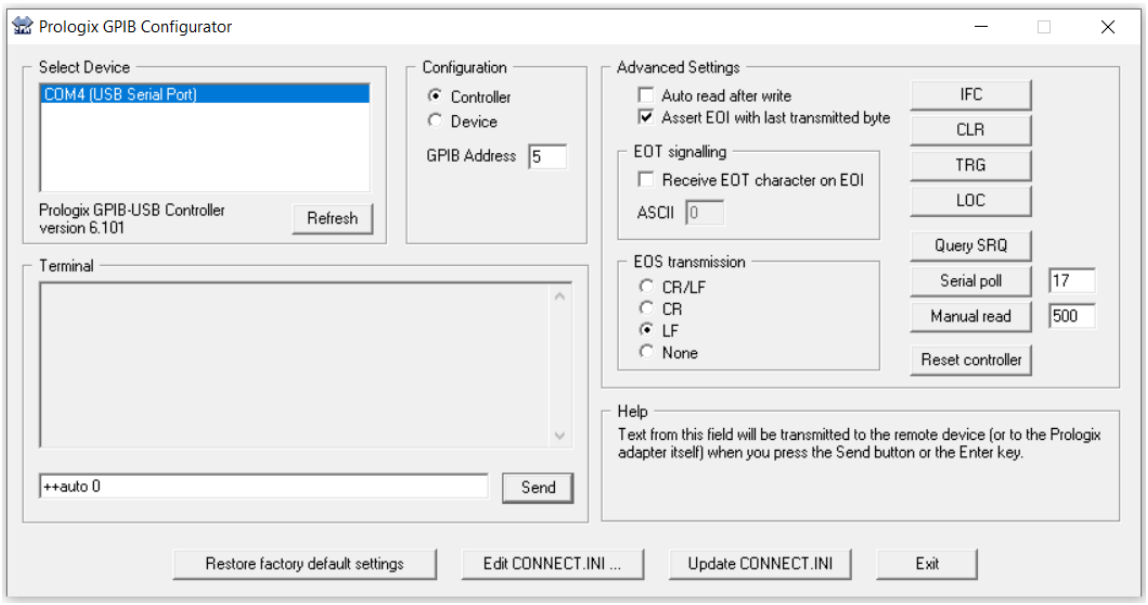
By default, the Prologix controller has the “Read-After-Write” feature enabled, which if desired, can read data from the GPIB instrument that was sent a SCPI command. Note that this can create an additional delay and the feature can be disabled. A specific command can be sent if the user needs to read data at a particular time.

The controller comes with an executable GPIB configurator that can send individual SCPI commands from a computer to the controller, shown in Figure 40. Once the GPIB controller is connected to the computer via USB, the program can be executed and if properly installed, the controller is selectable through a COM port, shown in the “Select Device” section in Figure 40. By selecting a device (in this case, “COM4 (USB Serial Port)” is selected), the controller can now be configured and can send commands. For this thesis, the GPIB controller will exclusively function in “Controller” (Controller-in-Charge) mode.

4.3.1 Initial GPIB Application Setup



(a)



(b)

Figure 41 – Sending a command on the Prologix GPIB terminal, (a) before (b) after

The following settings are recommended to reproduce similar results: The “Auto read after write” feature should be disabled so that the controller can skip having to read from a device (a manual read command can be sent if desired). To disable, either the user may click on the “Auto read after write” box to deselect the feature, or can type the command, “++auto 0”, to disable (and verify) the feature via GPIB commands. To enable the feature again, either the box can be clicked to enable the feature, or the user can type the command, “++auto 1”. Figure 41 demonstrates sending a command to the controller via the terminal.

The next setting to configure is the End-of-Send (EOS) transmission termination characters. When data is sent from the computer to the controller, all non-escaped line feed (LF), carriage return (CR), and escape (ESC) characters are removed, and GPIB terminators can be added depending on the EOS transmission setting. It is important that commands end with a termination, else GPIB devices will misinterpret the command and will not function properly. To enable this feature, either the user may select “CR/LF” (to terminate with both a CR and LF), “CR”, “LF”, and “None” (if desired), from the configurator, or can type the command “++eos [N]”, where N is an integer from 0 to 3 (0 – CR+LF, 1 – CR, 2 – LF, 3 – None). Figure 42 shows an example of sending a command and setting the EOS transmission to “CR/LF”. Once the above settings have been set, the controller is ready to send SCPI commands to GPIB devices via a bus cable.

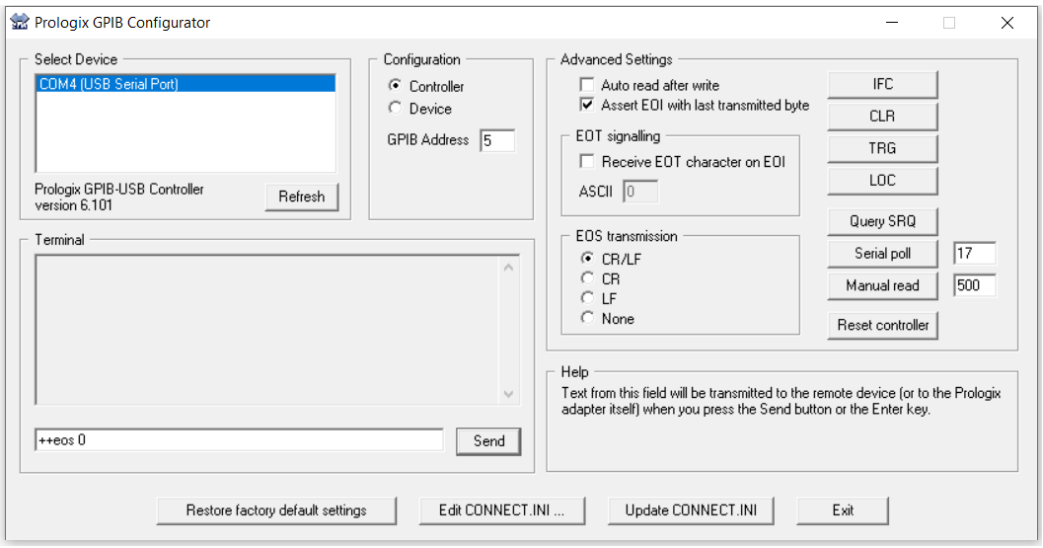


Figure 42 – Sending the command, "++eos 0"

4.3.2 Controller to GPIB Connections

Figure 43 – GPIB Cable

To connect the controller to devices, a standard 24-pin, “double-headed” (connector has male on one side and female on the other) bus cable is used. Figure 43 shows the standard cable, where each end of the cable contains both a male and female connector to make daisy-chaining more of a convenience. The controller male port feeds into the female connector of the bus, and

the remaining male connector can either connect to a GPIB device (an IEEE-488 port) or can be left unconnected, shown in Figure 44.

Figure 44 – Controller and GPIB device connected to bus

4.4 GPIB Device: SR430 Photon Counter

(a)

(b)

Figure 45 – Getting to SR430 Communication menu, (a) Setup key (b) Communication key

The SR430 photon counter is equipped to work with GPIB controllers and devices. Pressing the “SETUP” button will open the setup menu of the SR430 and enable the photon counter to work with GPIB. The “Commun.” prompt can be selected with the softkey that corresponds to the option and will bring up the communication setup for the SR430, shown in Figure 45.

In the “Communication” setup menu, the SR430 is set to output data to the “RS-232” protocol by default. To output data to GPIB, the softkey that corresponds to the “Output To” option can be pressed to cycle to “GPIB”, shown in Figure 46a. After setting the SR430 to function with GPIB, the address must be verified on the SR430 by pressing the softkey that corresponds to the “GPIB” setup menu, seen in Figure 46b. On the GPIB menu, the GPIB address can be accessed and changed. By default, the SR430 is set to address 23, but can be changed if desired (from a value between 0 to 31). For debugging purposes, it is desirable to be able to also control the SR430 manually during idle times. The “Override Remote?” softkey, seen in Figure 46b, can be cycled so that it is set to “Yes” and the user may manually operate the SR430 keys after a SCPI command has been sent, else the SR430 will disable all buttons on the interface and only work remotely via GPIB.

(a)

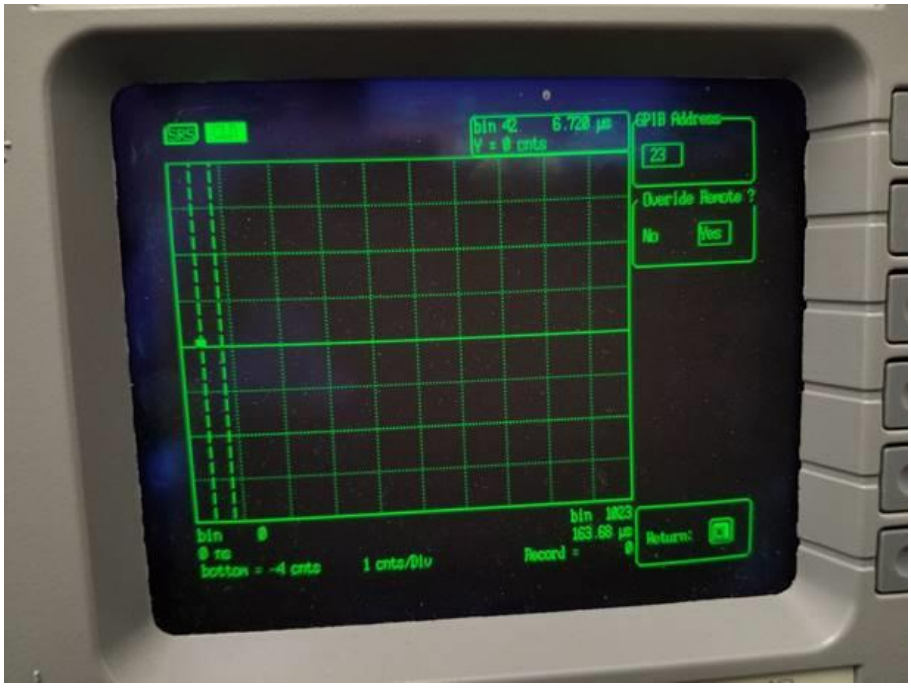


Figure 46 – Configuring GPIB on the SR430 (a) Communication Menu (b) GPIB Menu

4.4.1 Sending SCPI Commands to the SR430

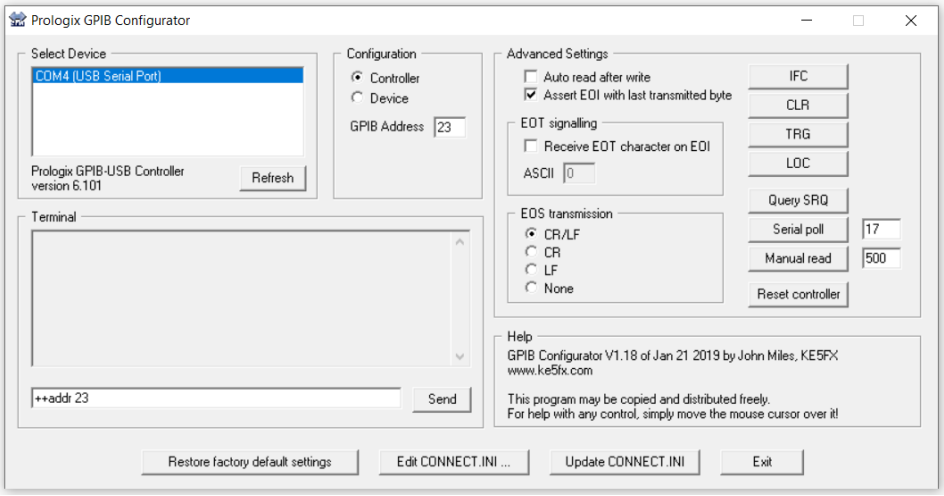


Figure 47 – Configuring GPIB controller to send SCPI commands to SR430

A list of SCPI commands is provided from the manufacturer of the SR430 photon counter and individual actions can be accessed with a quick SCPI command list provided on Page 9 of the SR430 manual (details of each command can be found on Page 81 of the SR430 manual) [13]. Before sending a command to the photon counter, the GPIB controller must be configured so that the address it is calling is the same as the address set on the SR430 (in this case, address 23). By typing into in the Prologix GPIB Configurator application and sending the command, "++addr 23", the GPIB controller is then configured to route all SCPI commands to the SR430 located at address 23, shown in Figure 47. Looking at the commands list, a command that can be sent to the SR430 is to change the trigger voltage level. To change the trigger level, the command, "TRLV {x}" can be sent via the Prologix GPIB configurator terminal where {x} is a value from $-2.000 \leq \{x\} \leq 2.000$ with a resolution of 1mV. For the following example, the trigger level will be set to 1.000 volts by typing into the terminal, "TRLV 1.0", and the trigger level on the SR430 should change to 1.000 volts, as shown in Figure 48. If the trigger level has

not changed, the user may verify that all connections are properly connected tightly, change the EOS transmission to a different mode, or verify that the computer is still able to respond to the controller.

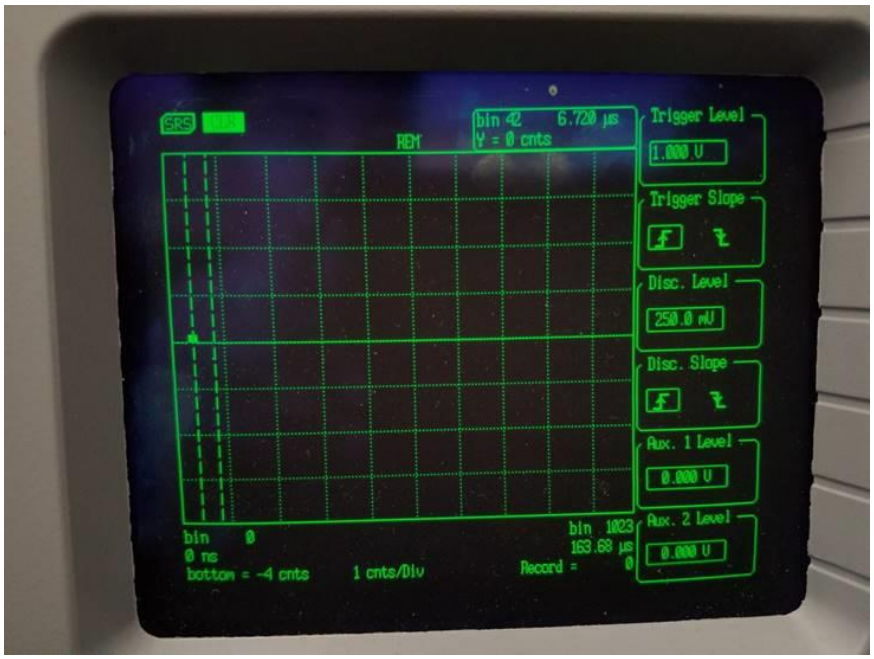


Figure 48 – SR430 Trigger level changed to 1V

Shown in Table 2 are a list of SCPI commands that will be used to assist with creating an automated photon counting test. Note that other commands may be investigated, however, for the scope of this thesis, these are the commands that have been used to create a successful test. This section along with the previous sections in this chapter demonstrate the use of GPIB to control measurement instrumentation. The following section will demonstrate the use of GPIB to control power supplies.

SR430 SCPI Command	Description	Notes
PAUS	The pause command. This pauses any test.	This command can be iterated at any time by the user while the SR430 is performing a test. Used mostly if user wants to rerun a test with new parameters.
CLRS	The clear command. This clears all count data from the SR430.	Should be used after collecting data and performing another test.
SSCN	The start command. This starts a test on the SR430.	Use this command only after setting up clock trigger, else, the SR430 will be stuck at the beginning of a test.
STAT	Functions as the "Do Stats" calculation. The SR430 will perform statistics on the current test data.	The manual process for this function can be done by pressing Math, then Stats, and "Do Stats".
SPAR ? {i}	Returns the result of one of the statistic values. {i} is an integer from 0-3 (0 – mean, 1 – Standard Deviation, 2 – Total Area, 3 – Baseline Area).	The command that is used exclusively is "SPAR ? 2" command. This returns the total counts collected from the test.
DCLV (?) {d}	The Discriminator Level command. This sets the discriminator voltage level. {d} is a decimal value from $-0.3 \leq d \leq 0.3$, with a resolution of 0.2mV. A question mark can be added to read the current discriminator value on the SR430.	For a discriminator level of 15mV, the user may enter "DCLV 0.015".
*STB? {i}	Serial Poll Byte (status bits) of the SR430. These are flags that the SR430 can set if certain conditions are met. {i} is an integer from 0 to 7. All bits are initially set to 1, where a return value of "0" signifies a flag has been raised.	The command "*STB? 0" will return if the SR430 is running a scan. A "0" will correlate to the SR430 currently running a test, where a "1" will correlate to the SR430 not running a test.

Table 2 – SR430 SCPI Commands

4.5 GPIB Device: Keithley 2450 SourceMeter

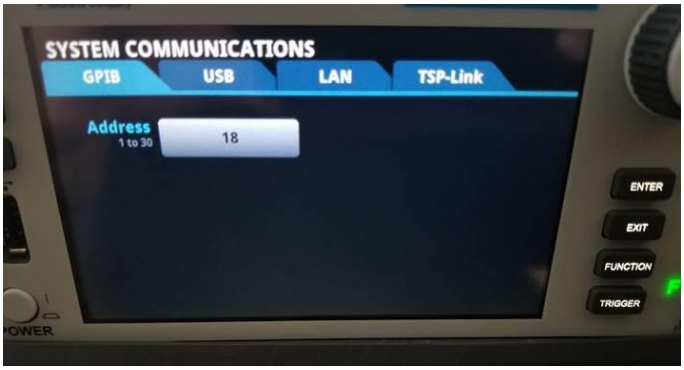
(a)

(b)

Figure 49 – Keithley 2450 SourceMeter, (a) front panel (b) back panel

The full potential of GPIB can be discovered when one uses it to remotely operate power supplies and poll data from measurement instrumentation. In this thesis, devices such as the LED current source and the reverse bias voltage of the APD from chapter 3 can be remotely operated. Setting these values by hand across multiple source-measure units is tedious and time consuming since often 1000 data points or more are needed for an accurate experiment. The Keithley 2450 SourceMeter, shown in Figure 49, is a GPIB-configurable, touchscreen enabled instrument that is capable of supplying voltage and current, as well as being able to meter how much of the supply is being used at any given moment.

(a)



(b)

Figure 50 – GPIB menu of Keithley, (a) system column (b) GPIB menu

To setup the supply with GPIB, the menu key can be pressed to access all the available menus of the Keithley SourceMeter. Looking under the system column, shown in Figure 50a, the GPIB settings is located by pressing the “Communications” button on the touchscreen. There are four available communication systems in place for the supply, where the first tab is the GPIB tab. The default address for the Keithley SourceMeter is 18. Following the flowchart shown in Figure 37, the Keithley SourceMeter can be connected to the GPIB cable by inserting the connector to the back of the supply, shown in Figure 51.

Figure 51 – GPIB connection to Keithley SourceMeter

Before establishing the supply settings used, the main function that is used on the Keithley SourceMeter is the output button, which is an LED button located on the front top right side of the supply. The Keithley SourceMeter’s supply is enabled when the output button emits a blue LED, else, it remains off. This function can be controlled remotely using the terminal.

To communicate to the Keithley SourceMeter, the command, “++addr 18”, is typed into the Prologix GPIB configurator terminal and then sent to the GPIB controller that will direct any

upcoming SCPI commands to the Keithley SourceMeter. The address may be verified by sending an appropriate read command or looking at the GPIB configurator window. To enable the output on the Keithley SourceMeter, the SCPI command, “OUTP:STAT ON”, is typed into the terminal. Respectively, the command, “OUTP:STAT OFF”, can also be typed to disable the output. The supply must have received the messages and toggled the output button remotely.

If the above commands have not been received by the Keithley SourceMeter, the GPIB connectors may be verified, or the user may send the appropriate commands to the Prologix controller to verify the controller is operating. Another setting to check is verifying the command set that the Keithley is using. This can be checked by pressing the menu button, pressing “Settings” on the touchscreen, shown in Figure 50a, and using the “SPCI” command set, shown in Figure 52

Figure 52 – Configuring Keithley with SCPI command set

An additional error that is possible after sending a command is the supply displaying, “Error -420”, shown in Figure 53, which signifies that the supply has been commanded to talk, but has nothing to return to the controller. This error may be removed by verifying that the Prologix controller has been enabled to send a termination character, shown earlier in Figure 42.

Figure 53 – Error -420: Keithley SourceMeter requested to talk but has nothing to say

Shown in Table 3 are the SCPI commands that can be used to interact with the Keithley. There are other commands that can be investigated [14], however, the following commands are what are used to conduct a successful test.

This chapter should serve as a basic reference on operating the system developed throughout this thesis. Chapter 5 will utilize MATLAB to replace the Prologix GPIB configurator application and develop a script that will automate commands sent to the instrumentation used in Chapter 4.

Keithley SourceMeter SCPI Command	Description	Notes
SENS:CURR:RANGE [d]E[I]	The sensed current range output from the Keithley while operating in voltage mode. [d] is a decimal value, “E” sets the value to be an exponential of base 10, and [I] is the exponent.	In GPIB mode, the Auto Range function is disabled, and the user must declare a range at the beginning. Example: “SENS:CURR:RANG 10E-3” sets the current range to 10mA.
SENS:VOLT:RANG [d]	The sensed voltage range output from the Keithley while operating in current mode. [d] is a decimal value, “E” sets the value to be an exponential of base 10, and [I] is the exponent.	In GPIB mode, the Auto Range function is disabled, and the user must declare a range at the beginning. Example: “SENS:VOLT:RANG 20” sets the voltage range to 20V.
SOUR:FUNC <function>	The source function, where <function> is either “VOLT” to source voltage or “CURR” to source current.	Example: “SOUR:FUNC VOLT” will set the Keithley up to source a voltage.
OUTP:STAT <state>	The state of the output, where <state> can either be “ON” or “OFF” to turn on or off the supply from the Keithley.	Example: “OUTP:STAT: OFF” will turn off the output of the Keithley, regardless of the type of supply.
SOUR:VOLT d	Sets the voltage amplitude of the Keithley, where “d” is a decimal from -210V to 210 volts, and a precision of 1μV	Example: “SOUR:VOLT 1.8” will set the voltage output to 1.8V. Note this will not turn on the output of the supply.

Table 3 – Keithley SourceMeter SCPI Commands

Chapter 5 MATLAB Integration

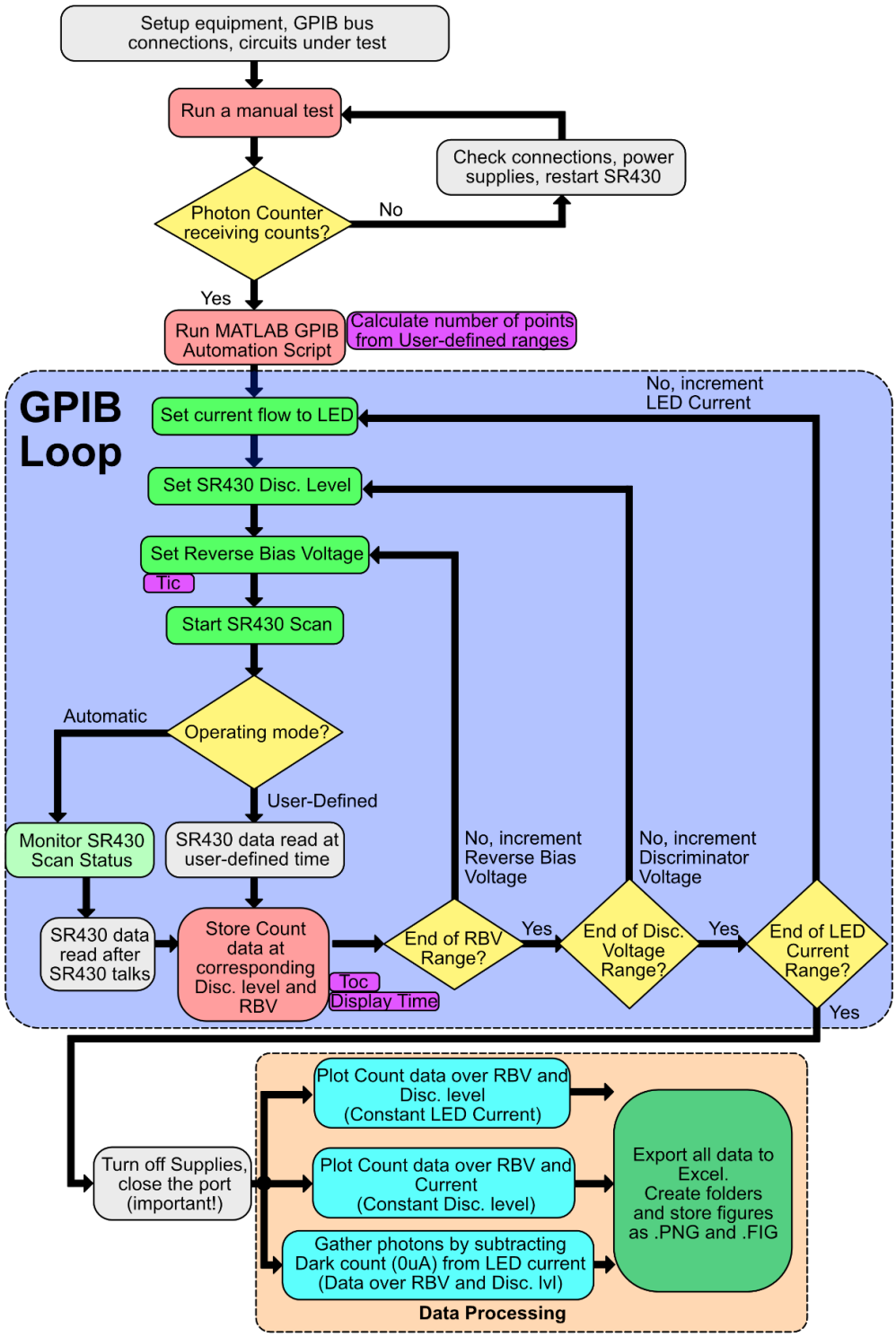


Figure 54 – Automatic Photon Counting using GPIB and MATLAB

Up to this point in this thesis, it has been shown that all the devices discussed are able to operate remotely via GPIB. One may use many terminals to individually control each GPIB device, however, it would be demanding to send multiple SCPI commands from a computer and to process data simultaneously. In this section, MATLAB will be the host environment that will be able to concurrently send SCPI commands and receive incoming data from the GPIB bus cable.

To connect to the GPIB controller, an initial setup can be programmed in MATLAB so that there is a successful link between the controller and computer. In MATLAB, the "serial('port')" function creates a serial port object that can be stored as an object, where 'port' is the associated COM port number that is assigned to the GPIB controller (see Ch. 3 for COM port configuration). In this thesis, "GPIBPort" is the variable that stores the serial object. Parameters of the COM port can be changed by using dot notation. To open the port, the function, "fopen(GPIBPort)", is used to grant MATLAB access to the COM port.

Shown in Code 1 is a main general connection between the GPIB controller and the host computer. By using this script and integrating the COM port configuration from the beginning of Ch. 4, SCPI commands can be chained together, and the system is ready to be automated via MATLAB. An important comment to stress for this thesis: after a COM port has been opened, the user must close this port. If the script must be stopped at any time by the user, the COM should be closed using the "fclose(GPIBPort)" function prior to restarting the script. If the script is executed without closing the port, MATLAB will stop and notify the user that the COM port to be opened has not been closed. At that point, a MATLAB bug where reopening a COM port that has not been closed will be inaccessible and therefore MATLAB must be restarted. The user

may recover the data that has been obtained, however, once the program has restarted, all experimental data will be lost.

```
%% GPIB Communication

GPIBPort = serial('COM5');      % Set up your COM port
GPIBPort.Terminator = 'CR/LF'; % Command Terminator required for controller
                                 % communications

GPIBPort.Timeout = 1;            % Serial Timeout set to 1 second
fopen(GPIBPort);                % Open the Controller port

fprintf(GPIBPort,'++eos 3');     % This is VERY important if we want to send
                                   % commands that use other
                                   % ASCII values. This command will
                                   % append a terminator to the
                                   % controller. We can also use ++eos 2

fprintf(GPIBport,"++auto 0")   % Turns off read-after-write feature of the
                               % Prologix controller. This prevents
                               % the controller from doing unnecessary
                               % reads and suppresses read-errors in
                               % all GPIB devices.

%%%%
% Add SCPI Commands here
% using fprintf(GPIBPort,cmd)
% where "cmd" is a string to be
% sent to GPIBPort
%%%%

response = fgetl(GPIBPort);    % Getting whatever the controller threw out
                                 % from the last command
disp(response)                   % Display result on the MATLAB Command Window

fclose(GPIBPort);              % Close the port **Important**
```

Code 1 – General COM port setup

The operation of this automated system can be visualized in Figure 54. Before running the MATLAB script developed for this thesis, it would be a good measure to verify all connections, instruments, and circuits so that any local issues can be resolved (see top of Ch. 3 for debugging manual processing). A first-time setup is required when setting up the MATLAB script, where

the following global parameters shown in Appendix C are set. Once all first-time setup has been initialized, the user may proceed with changing variables specific to testing. The variables shown in Appendix D are all available to change.

After the test variables have been set, the script can be initialized, and MATLAB will begin the photon counting process. There are two processes that occur when the script is executed: the main process is the GPIB loop that controls all devices on the GPIB bus, and the second process is an added quality-of-life feature in which local MATLAB timers are used to notify the user of the length and duration of the experiment.

The main process is as follows: The LED current will initially be turned off to begin testing the APD for dark counts. The next step is to set the discriminator voltage level of the SR430 to a low setting so that the photon counter can detect pulses. After that, to generate the voltage pulses from the APD, the reverse bias voltage is set at a voltage near the breakdown voltage of the APD. The final step is to start a scan on the SR430 and operate on one of two modes: an automatic mode and a user-defined pause mode. The automatic mode will wait for the SR430 to talk back to MATLAB when scanning is complete, while the user-defined pause mode will read the data from the SR430 after a pause and should be treated as a developer mode. The benefits of the automatic mode are that the user can let the MATLAB script do all the processing, whereas the benefit of the user-defined mode is that the user can force read data and save a few seconds of time which can be valuable if there is an indefinite number of points. Shown in Table 4 are the times that it takes to scan and record one point from the SR430 with the time bins and record lengths set from Ch. 3.

Time Required to Record One Point	
Mode	**Time**
Manual	~30 seconds
Automatic	9.27 seconds
User-Defined (for developing purposes)	8.55 seconds

Table 4 – Time it takes to record one point

For this thesis, a user-defined pause of 6.3 seconds was used in replace of the automatic mode. This time is calculated based on the timing diagram discussed in Ch. 3, recalling that each clock cycle will go through one record plus some processing time. In Layman's terms, the pause is defined as the total amount of records divided by the clock frequency. The actual time it takes to take one point is longer than an SR430 record scan due to MATLAB processing and places in the script where there is a generous amount of pause time to make sure there are no errors during a run.

Looking at Figure 54 again, the green "set" boxes are where the Prologix controller is setting a GPIB address and sending commands to the appropriate device. The inner reverse bias loop has two iterations where there is a change in GPIB address: an address change when sweeping the Keithley voltage source, and an address change to the SR430 to begin a scan at that voltage point. Due to this alternating address change, there are short arbitrary pauses that are added to ensure that the device-specific SCPI commands reach their respective GPIB devices. During experimentation, it was found that a short pause assists with commands fully being sent, whereas if there was no pause, there is a probability where the MATLAB script will run faster than the rate at which a SCPI command is sent, resulting in commands being incomplete. Shown in Code 2 is the innermost loop that changes the reverse bias voltage.

```
% Big Loop to pump up voltage
    counter = 0;
    for voltage = lowVolt:precision:highVolt
        fprintf('Current Voltage: %2.4f Volts \n',voltage)
        tLoop = tic; % For timing purposes
        fprintf(GPIBport,RBVA); % Addr. Change to Keithley voltage source
        KeithleyVoltage(voltage,GPIBport); % Voltage set
```

```
        % Setup the counter address
        fprintf(GPIBport,SR430A);
        SR430startCount(GPIBport); % Script will pause and wait

        counter = counter + 1; %Finished counting
        totalCounter = totalCounter - 1;


        SR430GetCounts(GPIBport); % Retrieve data from SR430

        % Read whatever prev command spits out
        SR430Received = fgetl(GPIBport);
        counts(counter,68iscounter) = str2double(SR430Received);
        disp(SR430Received);

        % Done with counting, increment voltage
        % and do it again
        photonTimeLeft(tLoop,totalCounter) % functions that calculate
        photonTimeElapsed                  % time remaining/elapsed
    end
```

Code 2 – Reverse Bias Voltage Loop

From the code, there are a few variables which the user is not able to access. For example, “RBVA” and “SR430A” correspond to the addresses of the Keithley voltage source and the SR430, respectively. The function, “fprintf”, will only accept character arrays. An example of converting user-defined values into characters to send is shown in Code 3. The function, “int2str” will convert the user-defined integer into a string which can be added onto other character strings and stored into a string that can be sent with the “fprintf” function.

```
%%% Saving address labels to MATLAB variables
RBVA = "++addr " + int2str(KeithleyRBVAddr);
SR430A = "++addr " + int2str(SR430Addr);
CurrA = "++addr " + int2str(KeithleyCurrAddr);
```

Code 3 – Converting integers into strings, combining string arrays

For user-friendly access, user-defined functions are created to describe a set of commands. These user-defined functions contain a set of lines that perform a specific task. Prior to using functions that use a specific GPIB instrument, the GPIB address must be changed. Appendix E demonstrates all the user-defined functions used in this thesis. Note that “sPort” is a parameter that is the serial object, which in this case is “GPIBPort”.

Concurrently to the main voltage loop, a timer and counter are being used to improve the quality-of-life for the user. Shown in Figure 54 in purple boxes are these timers. At the beginning of the script, the number of points is calculated based on the user-defined ranges. Based off the average time it takes to process one count, an estimation time is calculated so that the user may have an estimation on when the photon counting process is complete. As the main voltage loop iterates, the functions, “tic” and “toc”, are used to retrieve MATLAB’s local timer. The user-defined functions, “photonTimeLeft” and “photonTimeElapsed”, were created to return an average estimation of how much time is left on the experiment and the amount of time has passed from the beginning of the script. This is useful for if the user needs to calculate when to check in on the system.

After the reverse-bias voltage range has been completed, the discriminator voltage level is incremented. The reverse-bias voltage loop begins anew and is then repeated. Once all discriminator voltages are tested, the LED current is incremented to begin generating more counts, and consequently, photon counts will be produced. The first and secondary loops are initialized again, and different intensities of LED brightness are tested. At the final max LED brightness, the GPIB code will disable all GPIB devices, turn the outputs of all supplies off, and proceed to data processing. This chapter along with what has been presented from chapters 3 and 4 serve as the basis of this thesis and the results will be presented in chapter 6.

Chapter 6 – Data Analysis

6.1 Data Organization

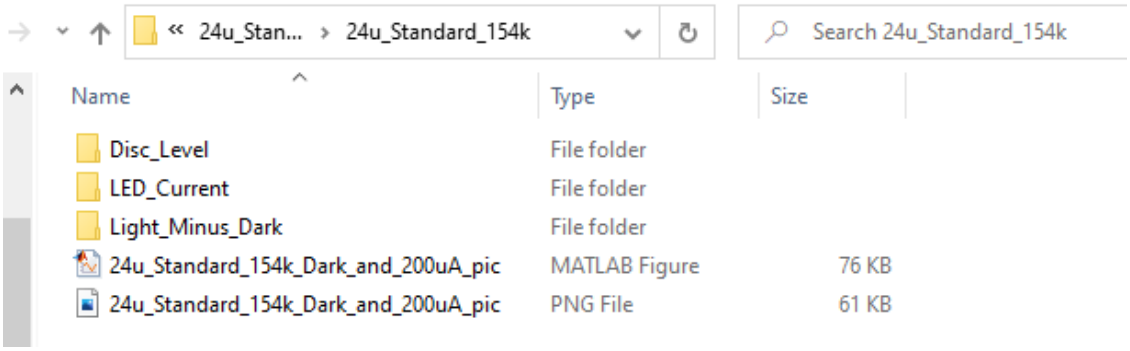


Figure 55 – Data organized in a folder

The main strength of MATLAB is to process, analyze, and present data. The end of the script developed in Chapter 5 will contain count data, user-defined ranges, and workspaces where information can be extracted. The main workspace organizes count data as a function of reverse bias voltage and SR430 discriminator level, where each set of data is at a certain LED current level. MATLAB-generated figures are displayed at the end of the run. Figures are saved both as a PNG and a MATLAB figure for visualization purposes. Concurrently, all data is exported to Excel into a workspace folder corresponding to LED current. For quick data analysis, the script provides a comparison between the highest intensity and the dark counts, seen at the bottom of Figure 55.

Alternative data workspaces that can be created are to organize count data as a function of reverse bias current and LED current, where each set of data is at a certain discriminator level. This data is useful for viewing how count data varies with varying light intensity. Similar to the main workspace, plots and Excel files are generated and can be stored into a workspace folder corresponding to the discriminator levels. Once all the data has been organized into folders, the

folders are stored into a single folder corresponding to the name of the APD tested. This folder can then be accessed to export all data to the user's preferred data processor such as Excel or imported back into MATLAB.

6.2 APD Data Analysis and Comparison

The primary focus of storing all this data is to analyze all the APDs for the counts that are produced due to incident photons, or photon counts of the APDs. To process and analyze this data, a secondary data processing script is created that reads and processes all APD data folders for photon counts. To do this, the script is written where it reads all folders in the current workspace and searches for the main data for each light level, the LED current data. Once all data has been read into MATLAB, the script can reprint individual APD data where the light intensity data can be plotted on the same figure.

6.3 Analysis of Circle APDs

The following figures will discuss the data for the circular APD devices discussed in Chapter 2. Three APD sizes were used, where the size describes the diameter of the circular APD. All APDs are tested with the setup discussed in Chapter 3 and Chapter 4, where the wavelength of light that was used is set to 467nm (dark blue light).

6.3.1 Circle Sizes: Changing Light Intensity

The first experiment involves testing the circle APDs at a constant light intensity. Figure 56, Figure 57, and Figure 58 are the total counts for the 50μm, 24μm, and 5μm circle APDs, respectively. Each plot within each figure is a unique light intensity, where the current running through the LED is either at the maximum intensity of 200μA, 100μA, and an off state. From the figures, as the current increases between each plot, there is a noticeable increase in the amount of counts produced. This is due to the increased number of photons hitting the APD's depletion

region and thus creating an increase in counts. Also, within each plot, as the reverse bias voltage of the APD increases, so does the number of counts. This is due to an increased breakdown probability due to an incident photon with increased bias. Therefore, the photon detection probability (PDE) rises with increased reverse bias for a Geiger-mode APD.

To view the counts produced by incident photons, or photon counts, the off (dark) state of the APD plot is subtracted from each of the light intensity levels, which is shown in Figure 59, Figure 60, and Figure 61 for the 50μm, 24μm, and 5μm circle APDs, respectively. From Figure 59 and Figure 60, there is a noticeable increase in counts due to an increase in light intensity. This positive count shows that by varying the constant amount of light intensity applied to the APD, the APD will proportionally produce extra counts due to the number of photons applied. Note that in Figure 61, there is also an increase in counts, however, as the reverse bias voltage is increased to a certain point, the amount of counts begins to decrease, suggesting that the smaller sized APDs are saturating and therefore increasing the light or bias voltage beyond the peak visible in Figure 61 will lead to an apparent reduction in counts. Some papers refer to this phenomenon as “paralysis” although it has a simple cause. The reason for this count reduction at high bias voltages or high photon flux is because the APD output waveform’s apparent DC level shifts due to pulse pile-up. Once enough pulses pile up, the waveform will look like a positive DC voltage with negative going pulses instead of a ground-level DC voltage with positive going pulses. An even further increase in light will result in a positive DC voltage with no detectable pulses. This is an interesting case where very high photon flux will result in “zero” counts. The APD should never be operated in this region as there is no useful data to be obtained.

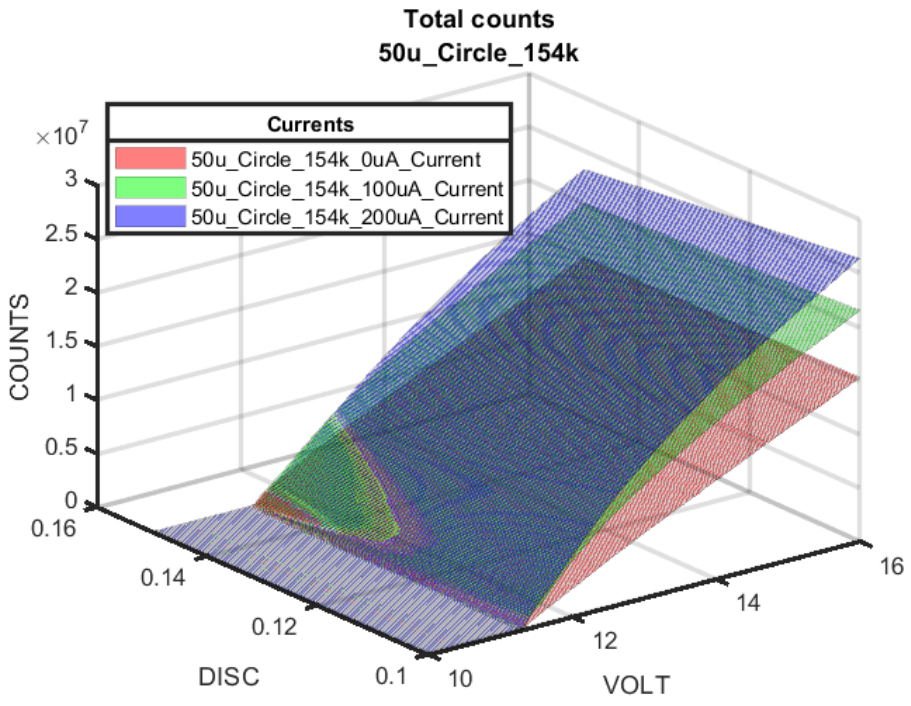


Figure 56 – Total counts as a function of bias voltage and light intensity for a 50µm circle APD.

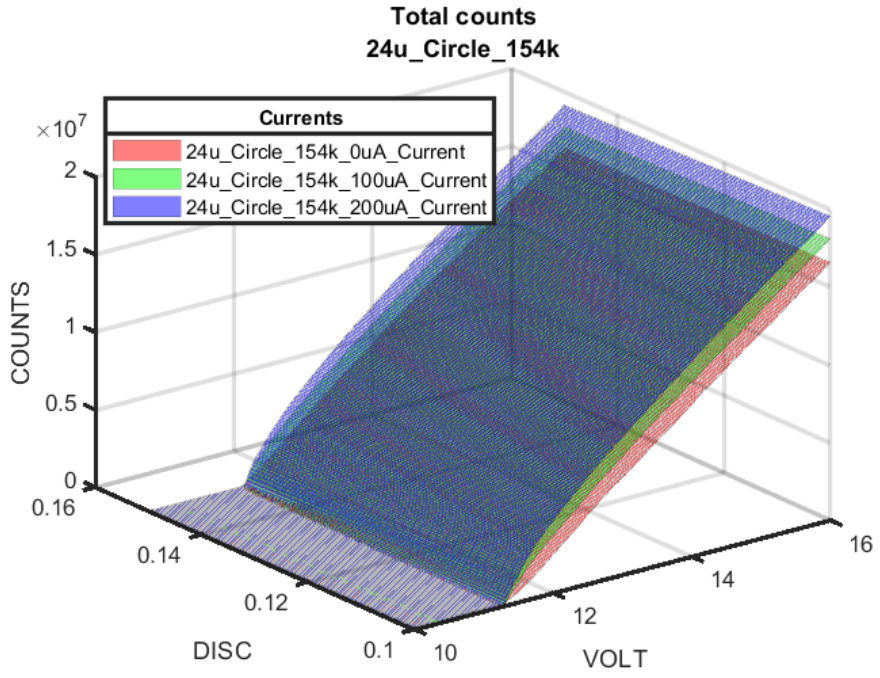


Figure 57 – Total counts as a function of bias voltage and light intensity for a 24µm circle APD.

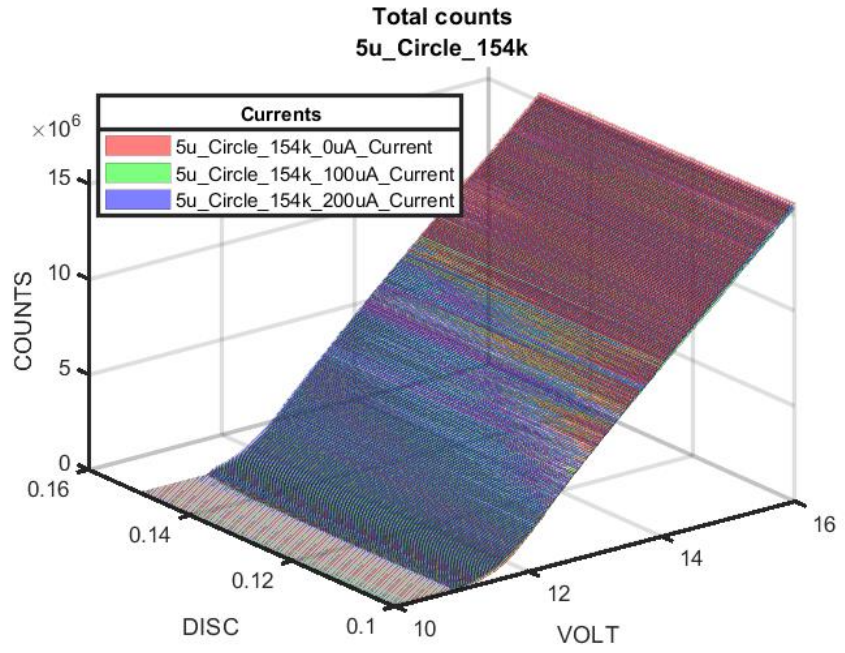


Figure 58 – Total counts as a function of bias voltage and light intensity for a 5μm circle APD.

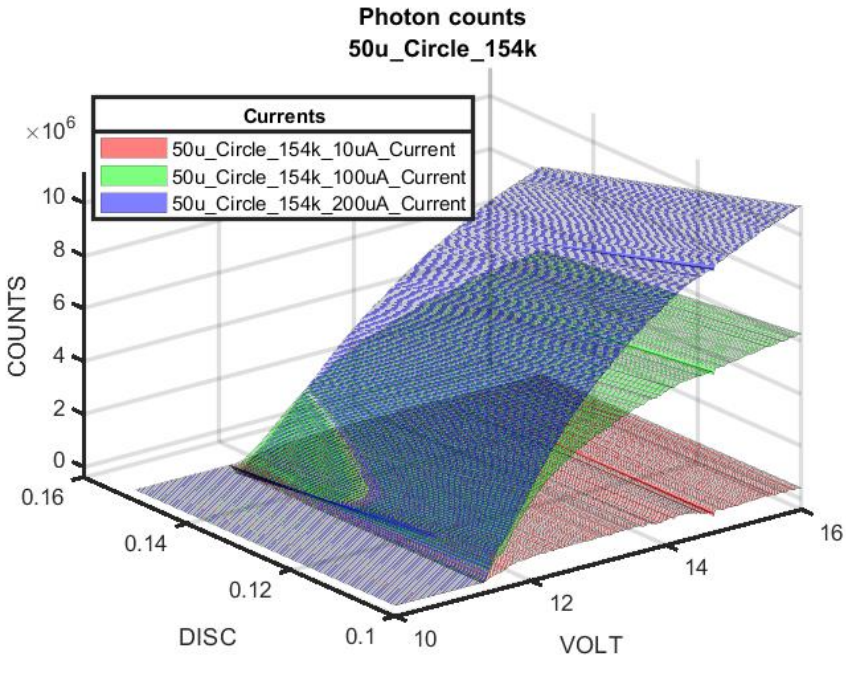


Figure 59 – Photon counts as a function of bias voltage and light intensity for a 50μm circle APD.

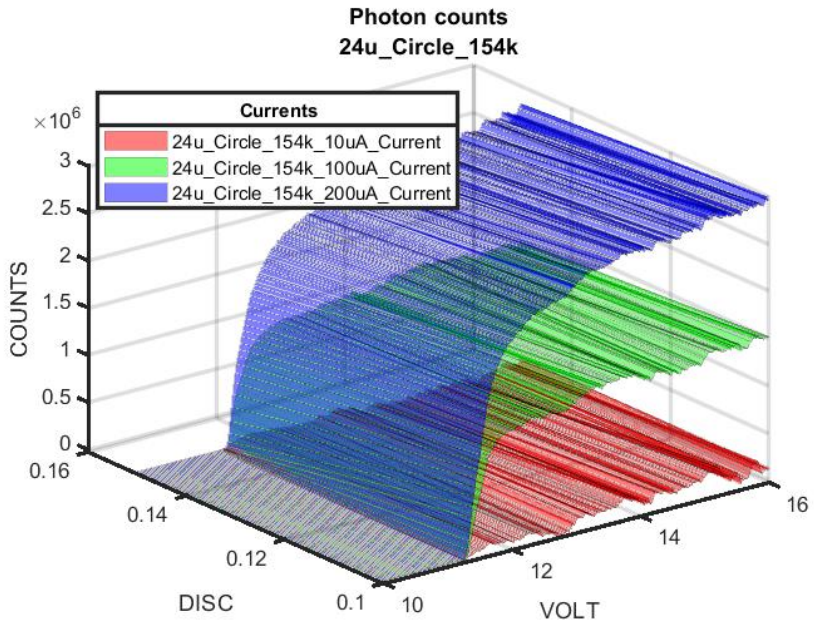


Figure 60 – Photon counts as a function of bias voltage and light intensity for a 24μm circle APD.

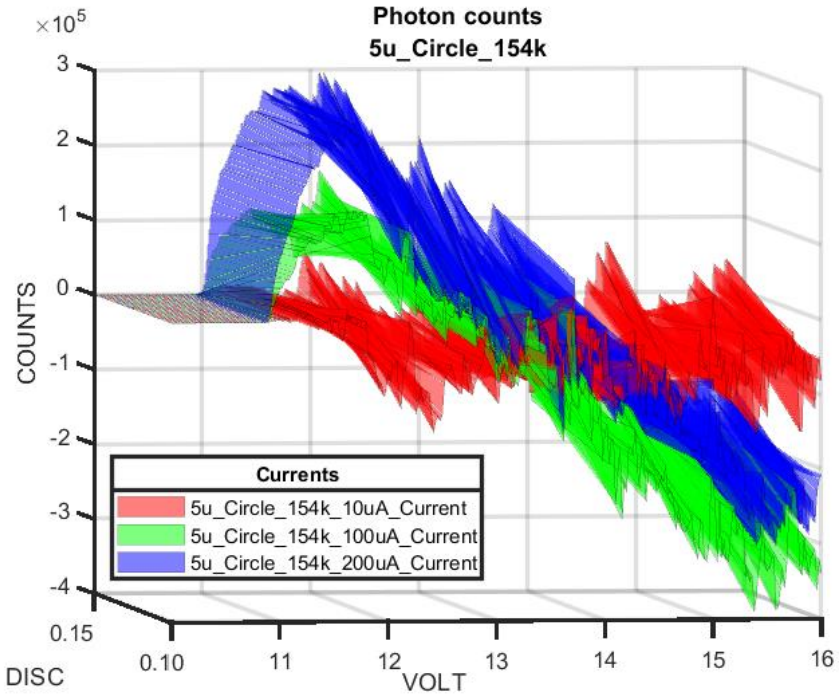


Figure 61 – Photon counts as a function of bias voltage and light intensity for a 5μm circle APD.

6.3.2 Comparing Circle Sizes

Figure 62, Figure 63, and Figure 64 is a reorganization of all the circle APD data from the previous section, where each circular APD size is compared at constant LED drive currents of 200μA, 100μA, and a dark state respectively. From the figures, it is apparent that as the size of the APD increases, the counts increase as well. This is due to there being a physically larger APD area for the photons to hit. From Figure 64, as the APDs are all under a dark state, the total counts produced are roughly equal. Therefore, the size of the APD may not be a main factor when it comes to the number of counts produced in a dark state (dark counts). The dark count of an APD may depend more on factors such as shape, device material stackup, or guard structures used. Figure 65 and Figure 66 represent the photon counts produced at LED currents of 200μA and 100μA, respectively. From the figures, the 50μm and 24μm circle APDs generate a noticeably large amount of photon counts whereas the smaller sized 5μm circle APD produces the least number of counts. Also, as the current is reduced by half, the number of photon counts is alco reduced by half, suggesting that the intensity of incident photons is linear to the amount of photon counts produced by the APDs.

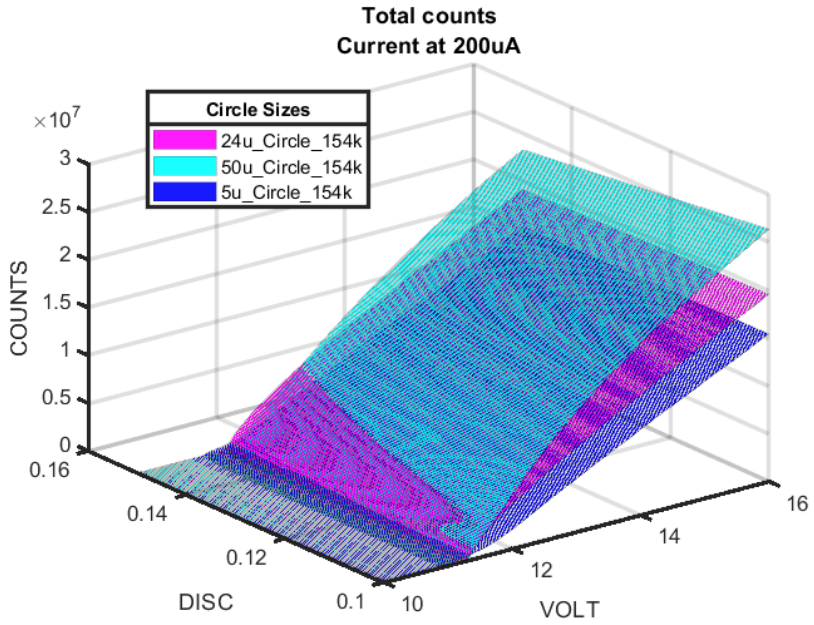


Figure 62 – Total counts of each circular APD at a LED drive current of 200µA.

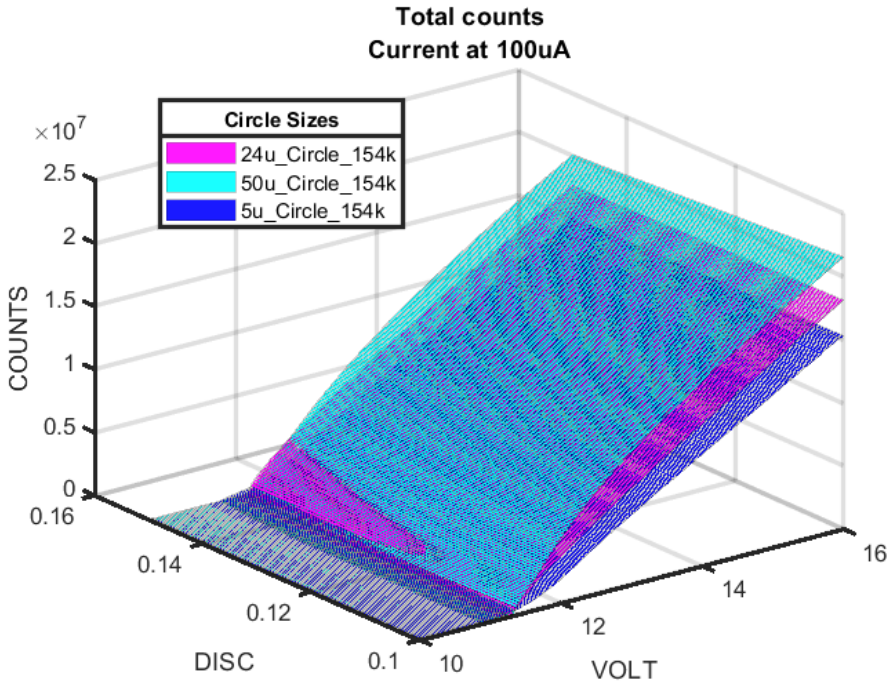


Figure 63 – A Total counts of each circular APD at a LED drive current of 100µA.

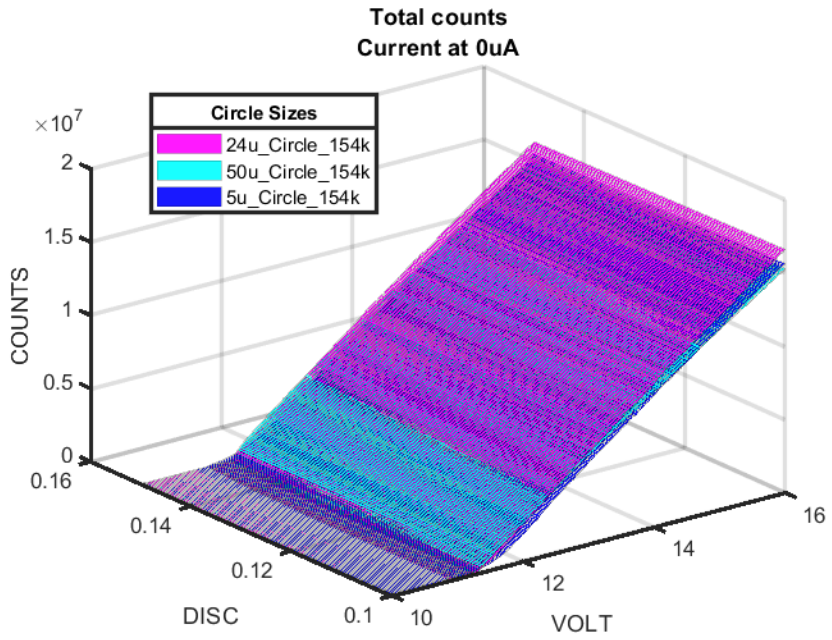


Figure 64 – Total counts of each circular APD under dark conditions.

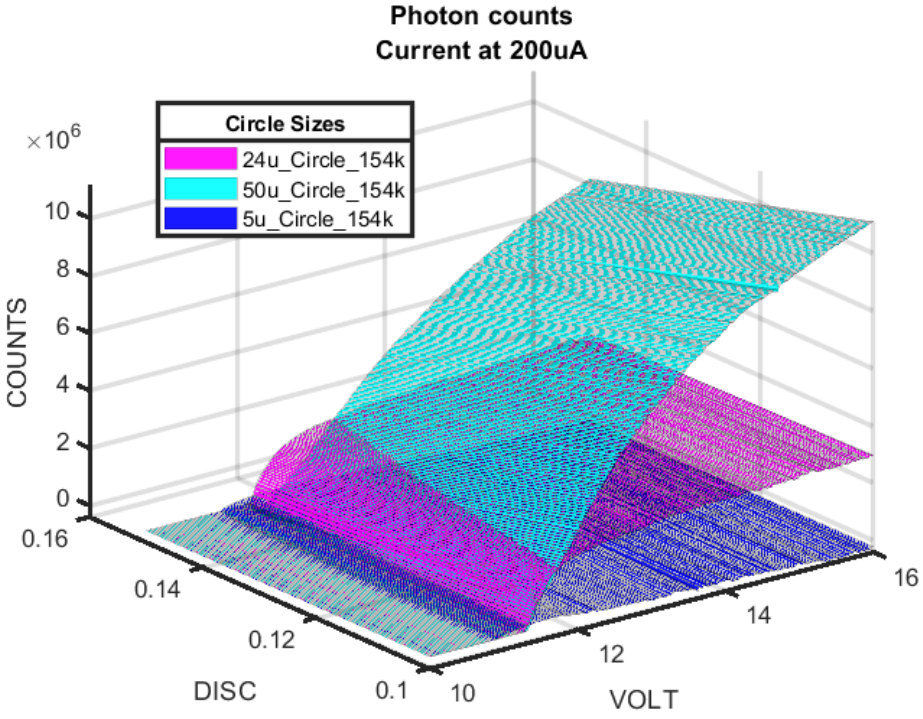


Figure 65 – Photon counts of each circular APD at a LED drive current of 200µA.

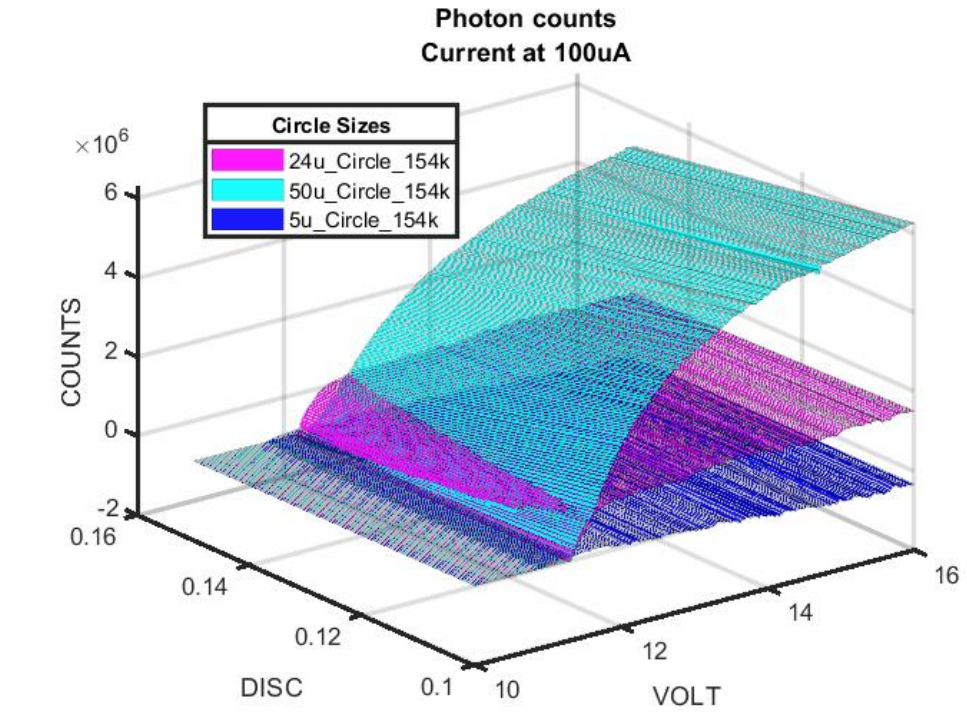


Figure 66 – Photon counts of each circular APD at a LED drive current of 100µA.

6.3.3: Photons/Area of Circle APDs

It is difficult to make conclusions about which APD design is best without having some common unit for reference. In this section, all the photon count data from the previous section is divided by each APD's area resulting in photon counts per square micrometer. By doing so will demonstrate the effectiveness of the APDs to generate photon counts based on size. Figure 67 shows the photon counts per square micrometer for all the circular APDs under illumination from an LED driven at 200μA. In Figure 67, it shows that the smallest APD, the 5μm circle APD, produces the most photon counts at lower reverse bias voltages but drops after a higher reverse bias voltage. While this result may be useful in a future study, this APD will be left out in the following few figures. Figure 68 and Figure 69 show the photons produced per square micrometer area for the 50μm and 24μm circles at LED currents of 200μA and 100μA respectively. From the figures, the 24μm circle APD performs better at generating photon counts than the 50μm circle APD over the entire reverse bias range. This therefore suggests that the 24μm circle device or a similar sized device may the most optimal APD in terms of cost and size. Also, as the LED current is reduced by half, the number of photon counts over area is also reduced by half, meaning that the intensity of light is linearly proportional to the amount of photon counts produced.

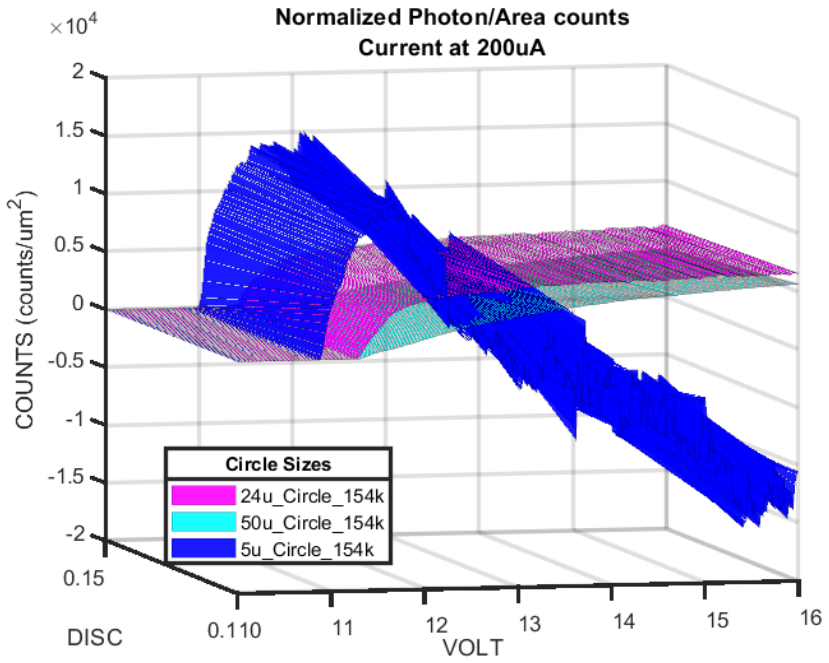


Figure 67 – Photon counts per μm^2 for all three circular APDs with LED current of 200μA.

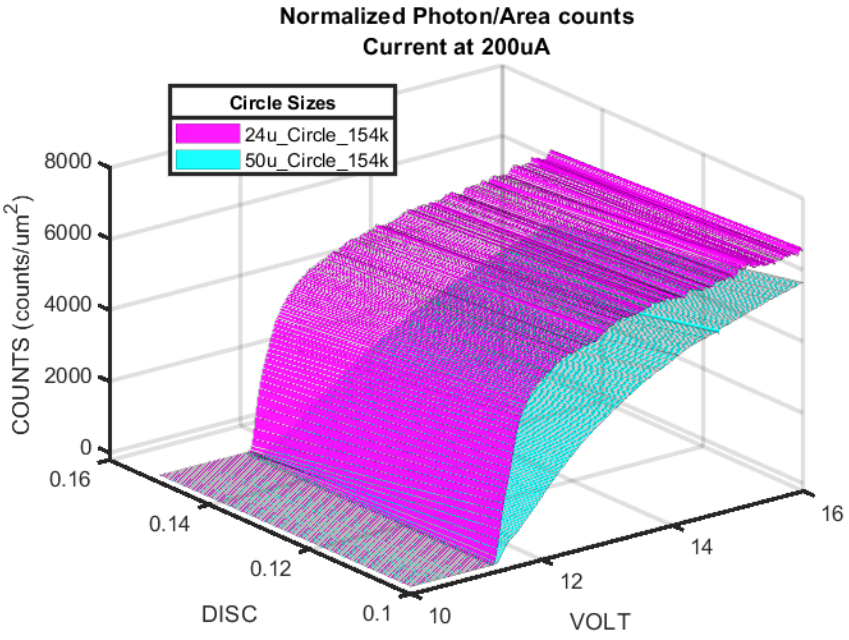


Figure 68 – Photon counts per μm^2 for 24μm and 50μm circular APDs with LED current of 200μA.

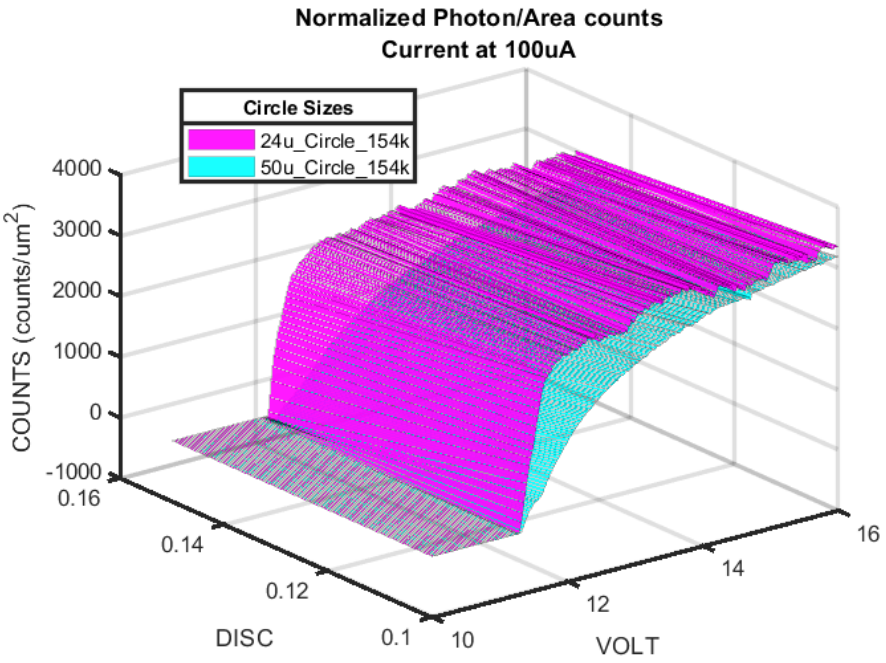


Figure 69 – Photon counts per μm^2 for 24µm and 50µm circular APDs with LED current of 100µA.

6.4 Analysis of Square APDs

The following figures will discuss the data for the square APD devices discussed in Chapter 2. Three APD sizes were used, where the size describes the side of the square APD. All APDs are tested with the setup discussed in Chapter 3 and Chapter 4, where the wavelength of light that was used is set to 467nm (dark blue light). Note that the analysis will follow a similar structure that was used while analyzing the circle APDs.

6.4.1 Square Sizes: Changing Light Intensity

The second experiment conducted involves testing the square APDs at a constant light intensity. Figure 70, Figure 71, and Figure 72 are the total counts for the 50μm, 24μm, and 5μm square APDs, respectively. Each plot within each figure is a unique light intensity, where the current running through the LED is either at the maximum intensity of 200μA, 100μA, and an off state. An observation of these figures can be seen that is similar to the observations of the circle APDs, where as the current increases between each plot, there is a noticeable increase in the amount of counts produced due to the applied light intensity over the APDs. Figure 73, Figure 74, and Figure 75 are the photon counts of the APDs, where the dark state current plot is subtracted out from the different light intensity plots. Similarly to the circle APDs, with higher light intensities, there is a noticeable increase in photon counts. Note that in Figure 75 for the 5um square APD, there is an increase in photon counts up to a certain reverse bias voltage, however, the counts start to decrease after a certain reverse bias voltage, suggesting that the smaller sized APD is saturating, has a higher amount of dark counts with high reverse bias voltage, and therefore the applied light will have a negative effect on the APD.

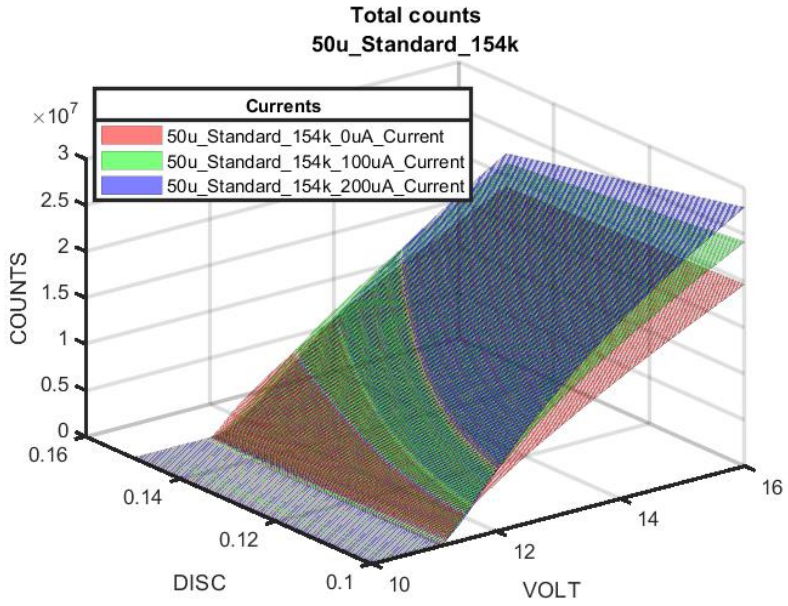


Figure 70 – Total counts as a function of bias voltage and light intensity for a 50µm square APD

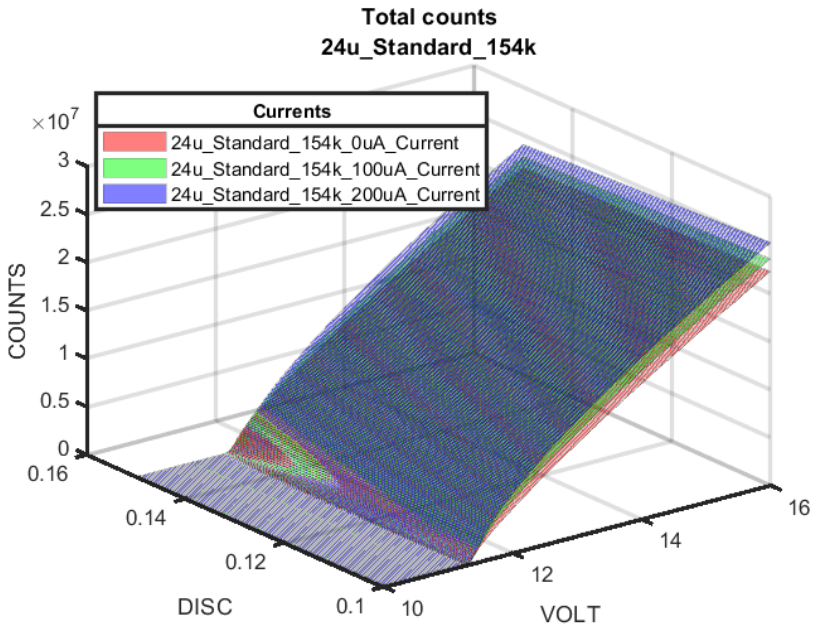


Figure 71 – Total counts as a function of bias voltage and light intensity for a 24µm square APD

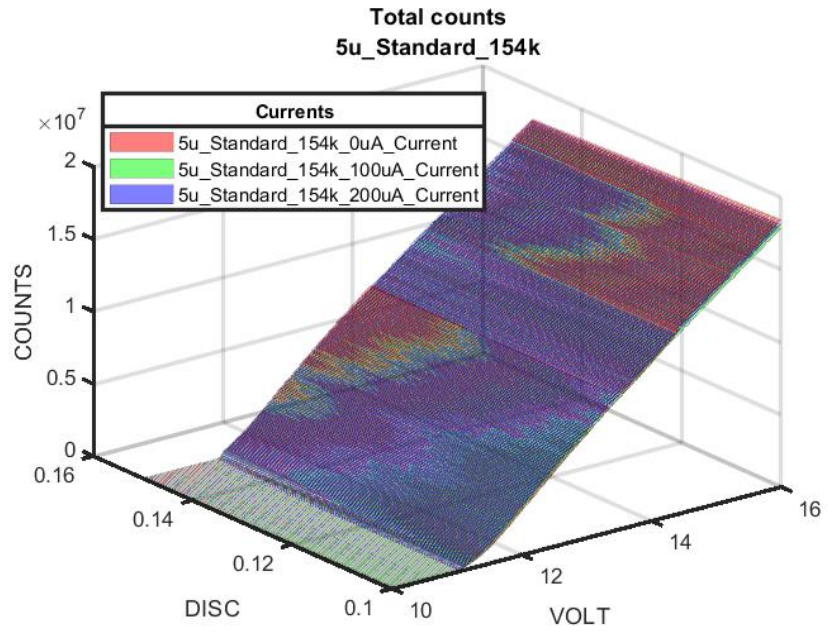


Figure 72 – Total counts as a function of bias voltage and light intensity for a 5µm square APD

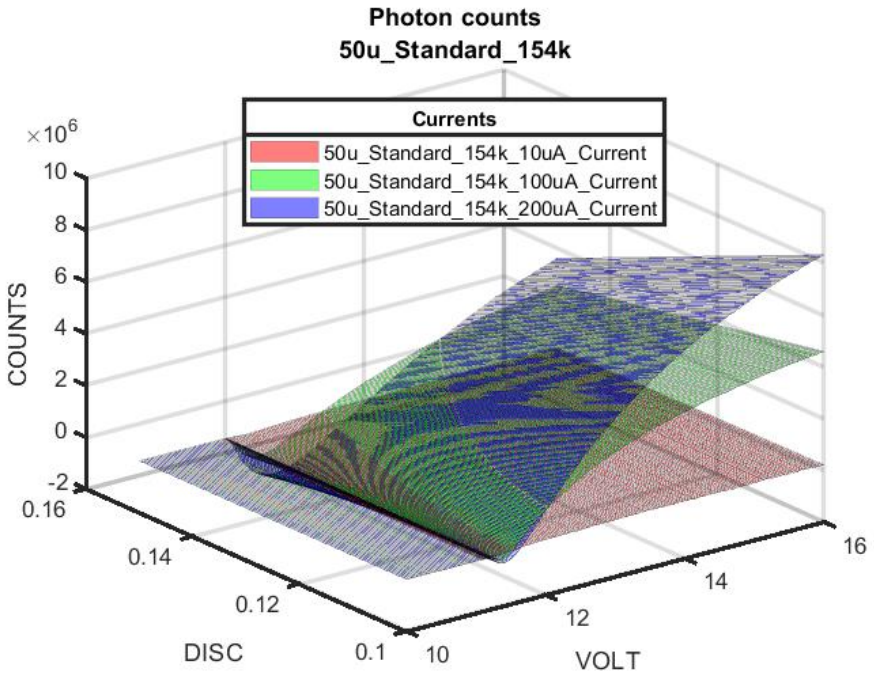


Figure 73 – Photon counts as a function of bias voltage and light intensity for a 50µm square APD

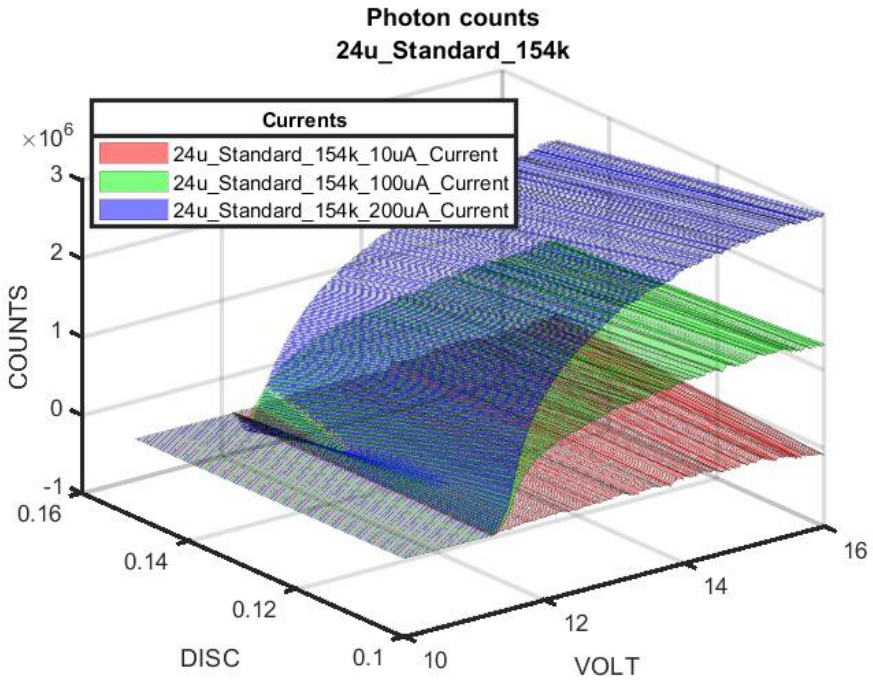


Figure 74 – Photon counts as a function of bias voltage and light intensity for a 24µm square APD

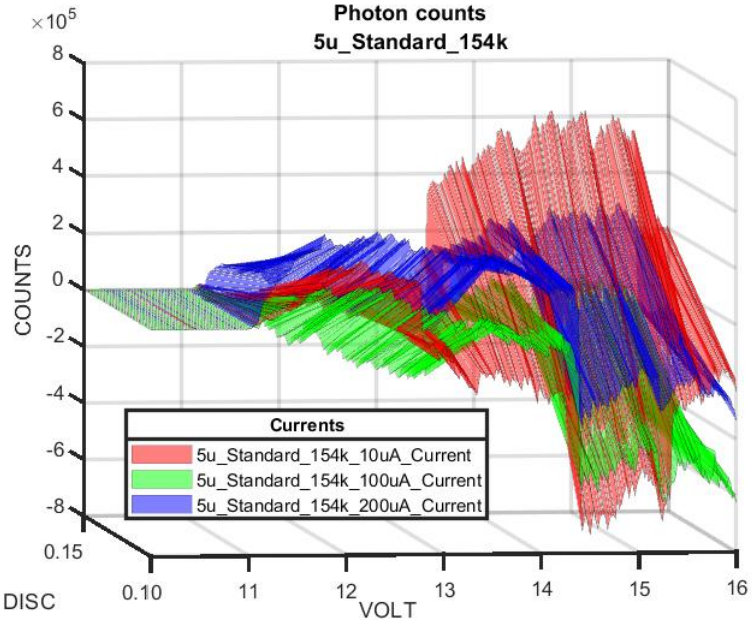


Figure 75 – Photon counts as a function of bias voltage and light intensity for a 5µm square APD

6.4.2: Comparing Square Sizes

Figure 76, Figure 77, and Figure 78 is a reorganization of all the square APD data from the previous section, where each figure corresponds to an LED current of 200µA, 100µA, and a dark state, respectively. From the figures, an observation can be seen that is similar to the observation made for comparing the size of circle APDs, where as the size of the APD increases, the amount of counts increases. From Figure 78, as the square APDs are all under a dark state, the total counts produced are roughly equal. This observation is also like the one made with the square APDs, which suggests that the size of the APD will not be a main contributor to the counts produced under a dark state. Figure 79 and Figure 80 represent the photon counts produced at LED currents of 200µA and 100µA, respectively. Similarly to that of the circle APDs, as the size of the APD increases, there is a noticeable increase in the amount of photon counts produced.

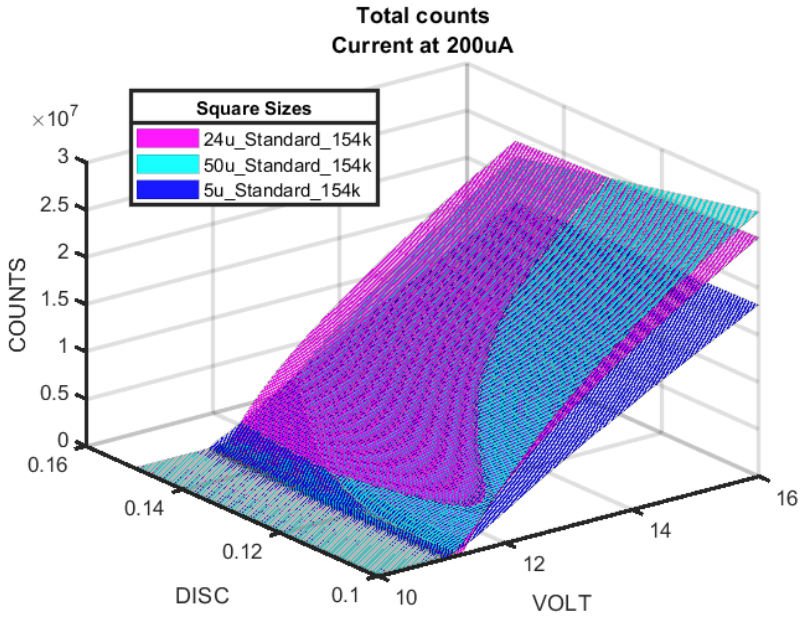


Figure 76 – Total counts of each square APD at a LED drive current of 200µA

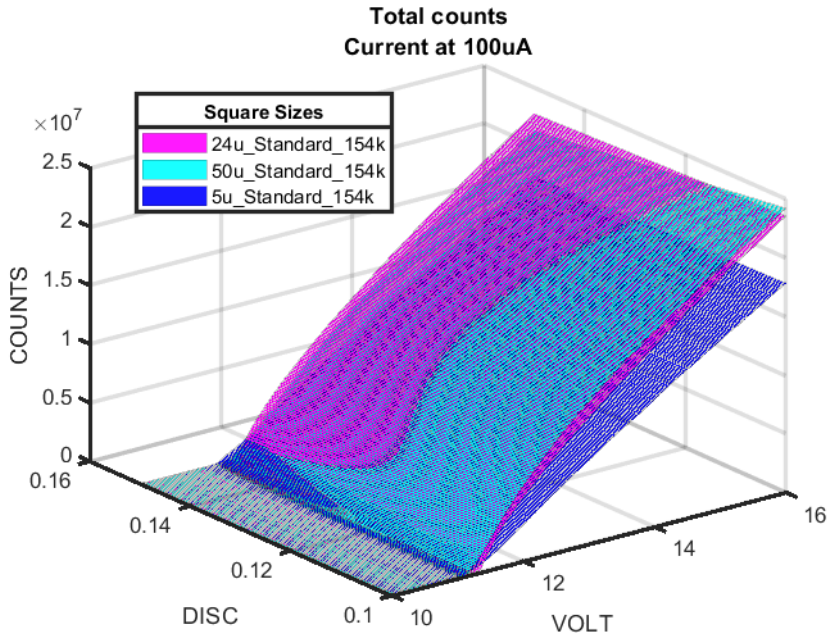


Figure 77 – Total counts of each square APD at a LED drive current of 100µA

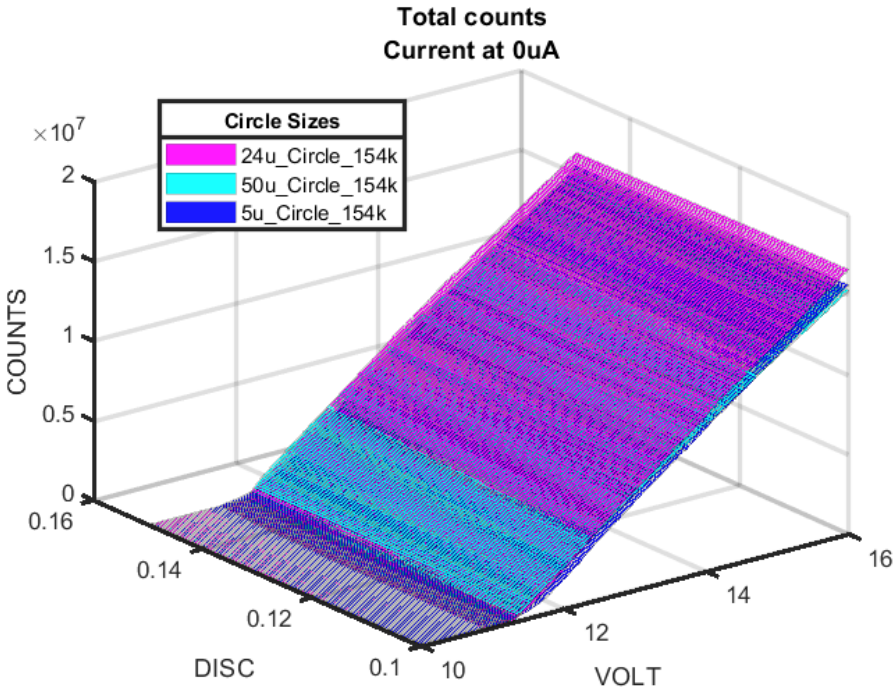


Figure 78 – Total counts of each circular APD under dark conditions

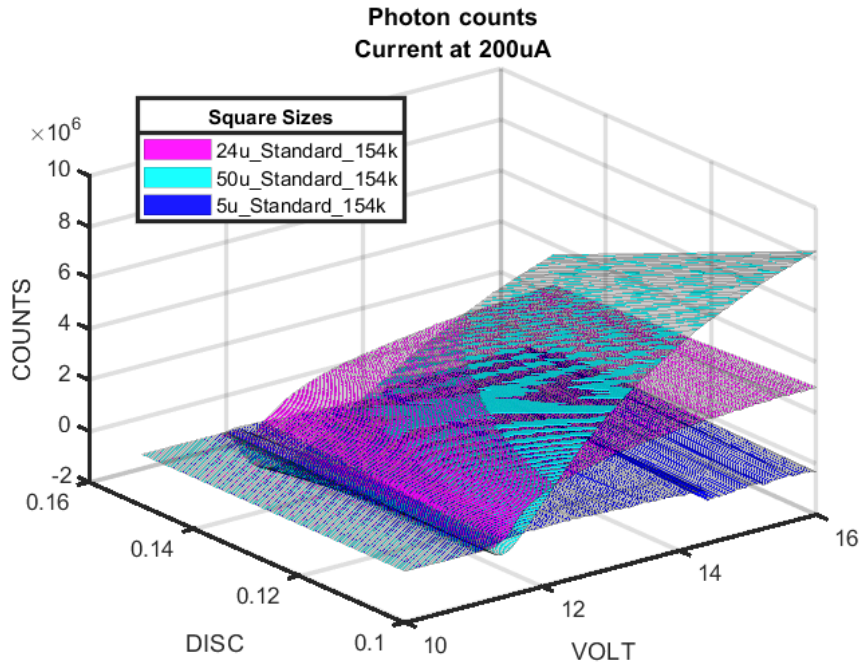


Figure 79 – Photon counts of each square APD at a LED drive current of 200µA

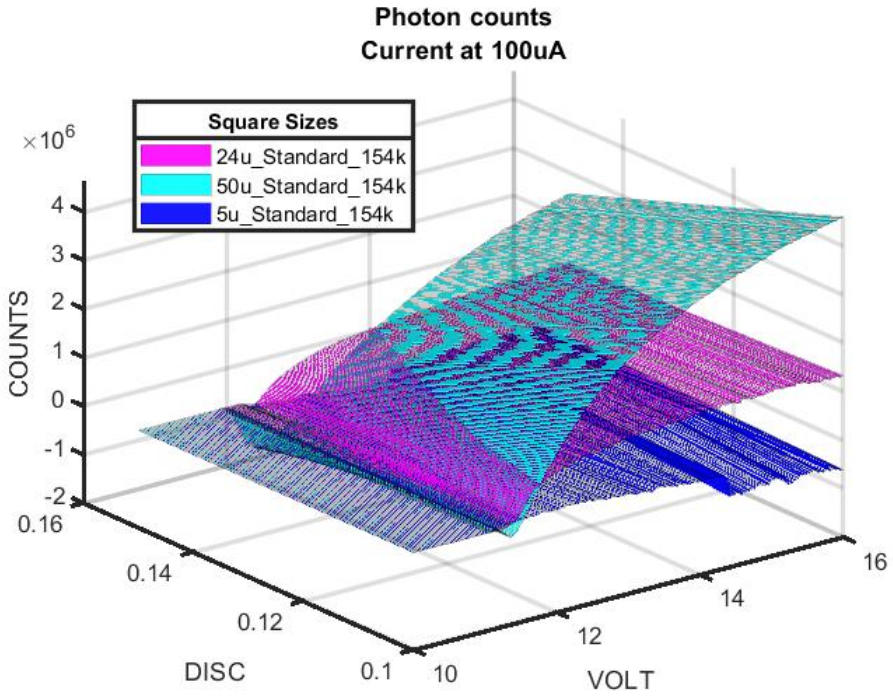


Figure 80 – Photon counts of each square APD at a LED drive current of 200µA

6.4.3: Normalizing photons/area of square:

This section will take the photon count data from all the square APDs at 200μA (shown in Figure 79) and divides each individual plot by its respective square area, where the size of the square APD is the length of one side of a square. Doing so will demonstrate the effectiveness of the APDs to generate photon counts based on size. From Figure 81, similarly to what has been observed with the circle APDs, the smallest APD, the 5μm square APD, produces the most photon counts at lower reverse bias voltages but drops after a certain reverse bias voltage. While still being useful data, this APD will be left out in the following figures. Figure 82 and Figure 83 show the photons produced over a certain area for the 50μm and 24μm square APDs at LED currents of 200μA and 100μA, respectively. Similar to what was observed from the circle APDs, the 24μm square APD performs better at generating photon counts than the 50μm circle APD over the entire reverse bias range. Note that as the LED current is reduced by half, the number of counts also decreases by half, suggesting that the amount of light applied on the APD is linear to the amount of photon counts produced.

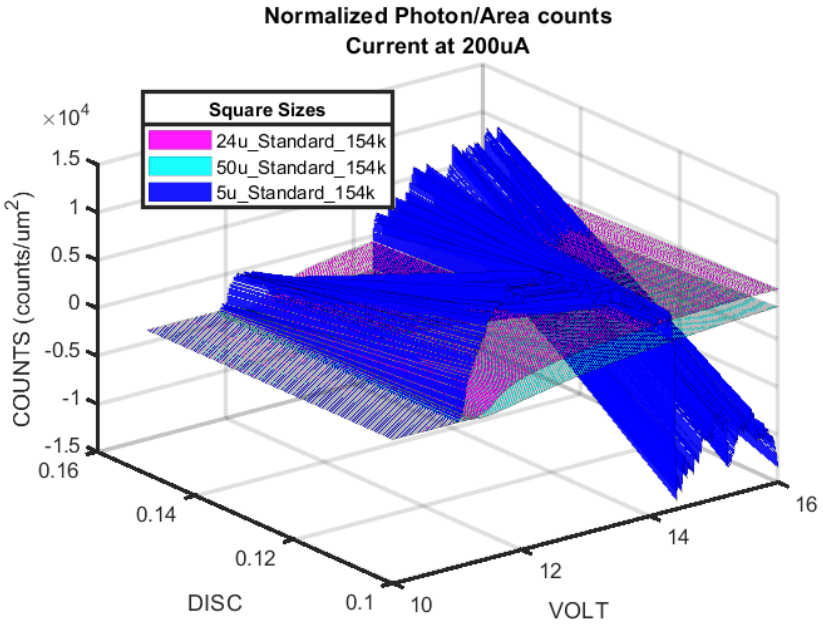


Figure 81 – Photon counts per μm^2 for all three square APDs with LED current of 200μA

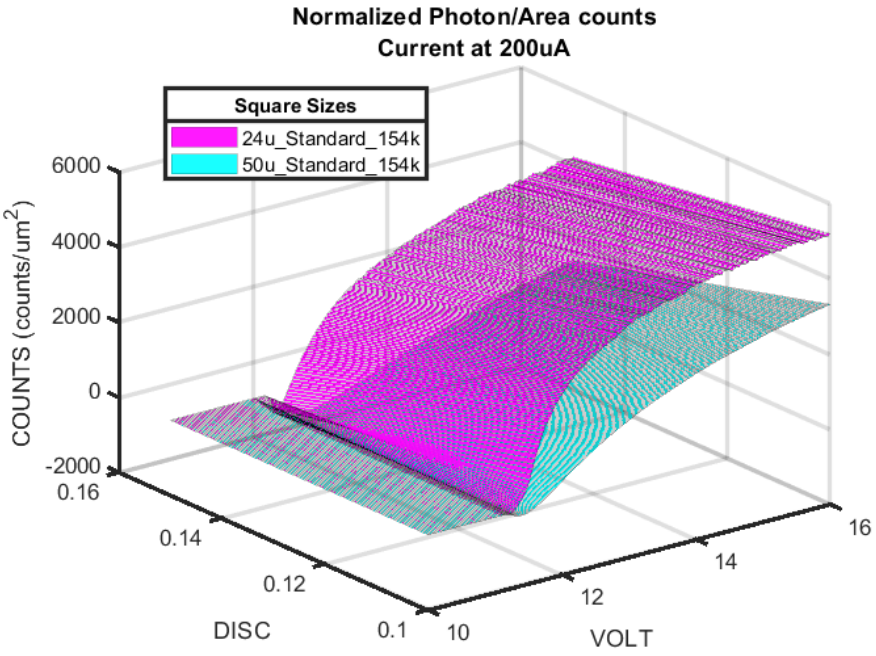


Figure 82 – Photon counts per μm^2 for 24μm and 50μm square APDs with LED current of 200μA

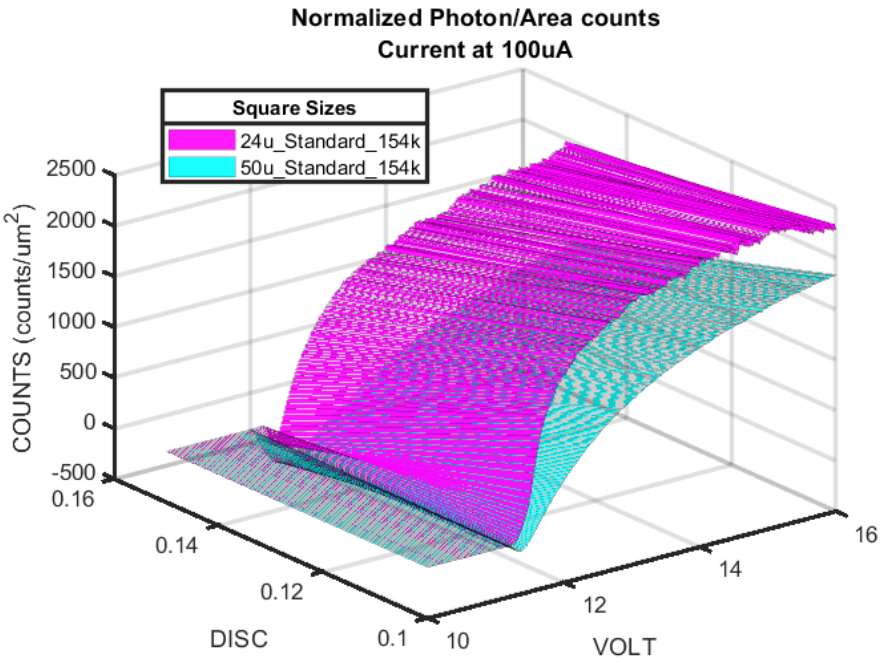


Figure 83 – Photon counts per μm^2 for 24µm and 50µm circular APDs with LED current of 100µA

6.5 Analysis of All APDs

From this point, all experiments have fully been conducted on all APDs. This section will present all the APD data presented in this Thesis from the circular and square APDs and be analyzing all the APD data for count data and areas of interest.

6.5.1 APDs: Changing Light Intensity

Each plot shown in Figure 84, Figure 85, and Figure 86 represents a unique APD, where each figure is at an LED current of 200μA, 100μA, and a dark state, respectively. From Figure 84, looking at a discriminator voltage of 100mV, the topmost plot is the 50μm square APD, followed by the 50μm circular APD. The next plot down is the 24μm square APD, followed by the 24μm circular APD. The final two plots are the 5μm square APD, with the bottommost plot being the 5μm circular APD. Within shapes, the larger sizes produce the most counts whereas the smaller sized APDs produce the least amount of counts. Within sizes, the square APDs produce more counts than the circular APDs. Figure 86 are all of the dark counts of the APDs, where the top three plots are the square APDs and the bottom 3 plots are the circle APDs. From this figure, the square APDs produce more dark counts than the circle APDs, which is due to the square APDs having corners that can likely produce a higher electric field, therefore a higher chance for more electrons to flow in a dark state, whereas the circle APDs are more uniform in shape around the edges and therefore produce fewer dark counts.

Subtracting the light counts from the dark counts, Figure 87 and Figure 88 are the photon counts of all APDs at 200μA and 100μA LED driver current, respectively. From Figure 87, the topmost plot is the 50μm circle APD, followed by the 50μm square APD. The middle plots are the 24μm square and circle APD, which from if viewing Figure 88, the 24μm circle APD produces more photon counts than the 24μm square APD. At the bottom of the figures are the

5µm circle and square APDs, which from a practical point of view, both produce the least amount of photon counts.

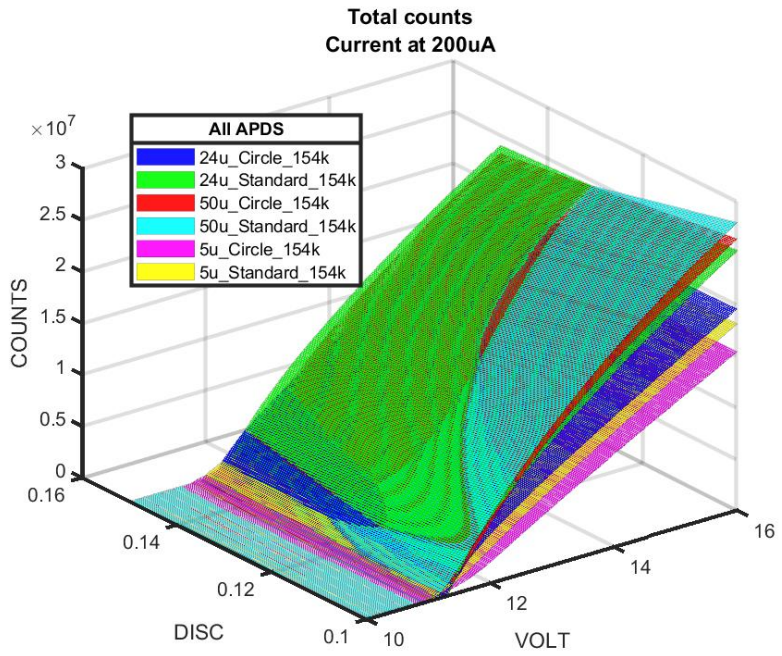


Figure 84 – Total counts of all APDs at an LED drive current of 200µA

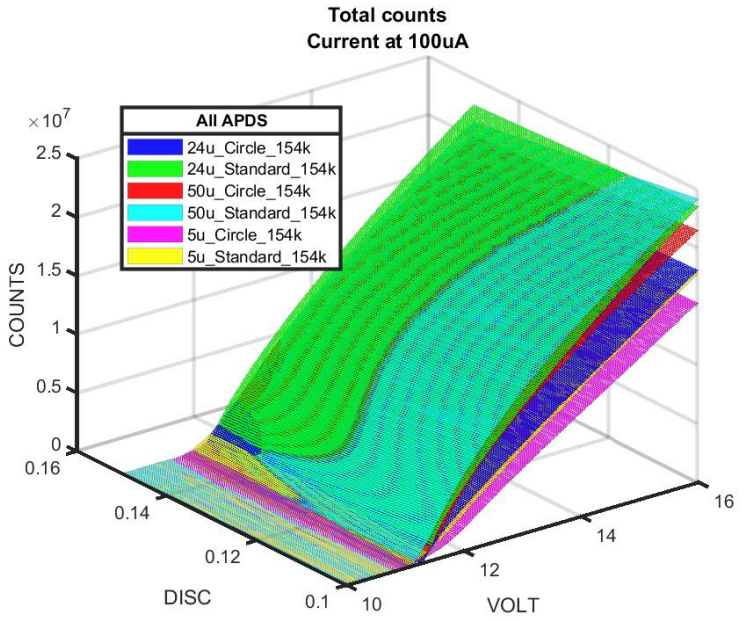


Figure 85 – Total counts of all APDs at an LED drive current of 100µA

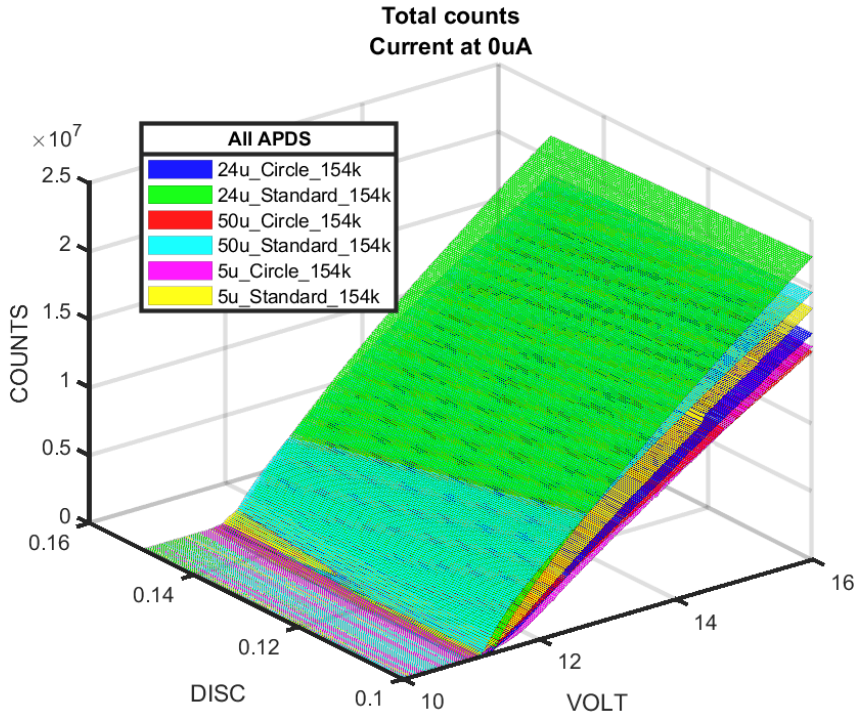


Figure 86 – Total counts of all APDs under a dark state

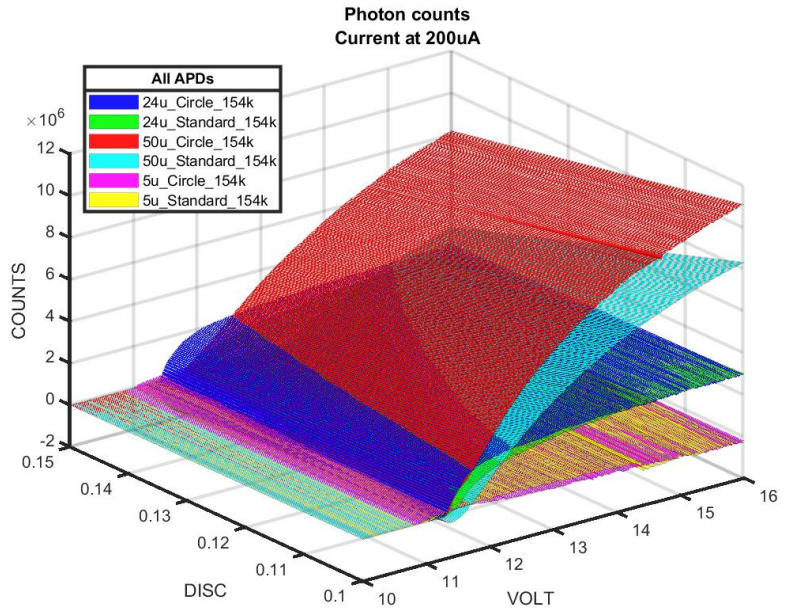


Figure 87 – Photon counts of all APDs at a LED drive current of 200µA

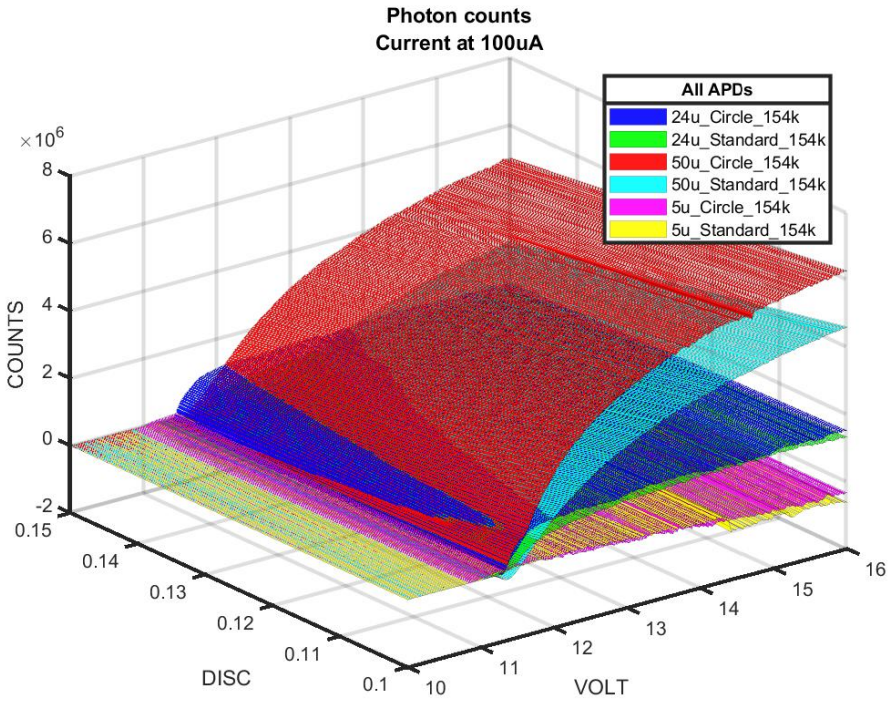


Figure 88 – Photon counts of all APDs at a LED drive current of 100µA

6.5.2 Comparison of all shapes and sizes:

It would be of interest to take all the photon count data and normalize each APD with respect to its shape area. Figure 89 is the normalized photon count data for all APDs. One may observe that the 5μm circle and 5μm square APDs produce the greatest number of counts for a particular voltage range. Although the two small APDs produce interesting results, these two APDs will be removed in Figure 90 and Figure 91 for clarity. From Figure 90 the topmost plot shown is the 24μm circle APD, followed by the 24μm square and 50μm circle APDs and the bottommost plot is the 50μm square APD. The 24μm circle APD performs consistently higher than all the other APDs, where with the larger APDs having a larger size and chance of having a larger depletion region capacitance, and the square APDs having corners which may produce higher electric fields that can produce more dark current pulses. Note that as the intensity of the LED current is cut in half, shown in Figure 91, the counts of all APDs also are cut in half, suggesting that the photon counts retrieved are linearly proportional to the amount of light applied to the APDs.

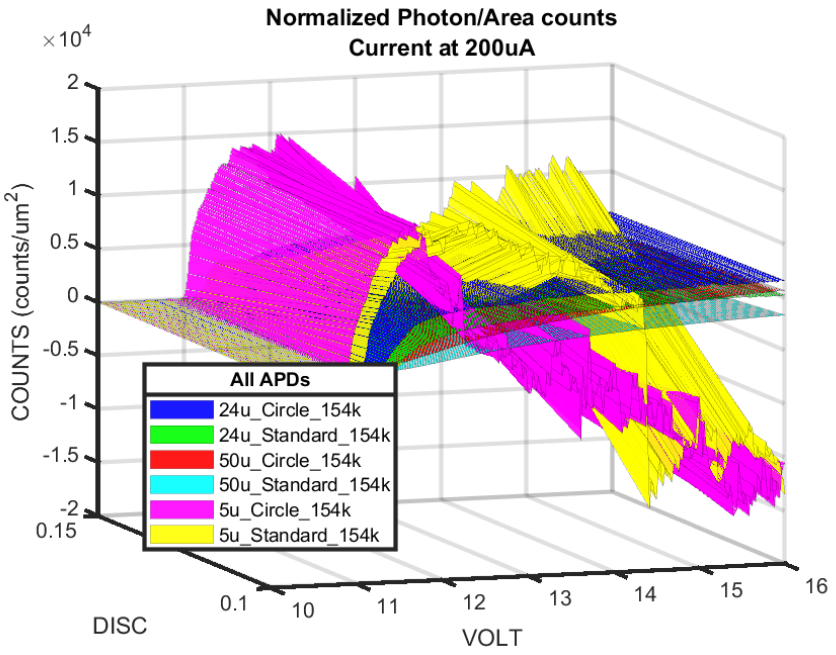


Figure 89 – Photon counts per μm^2 for all APDs with LED current of 200μA

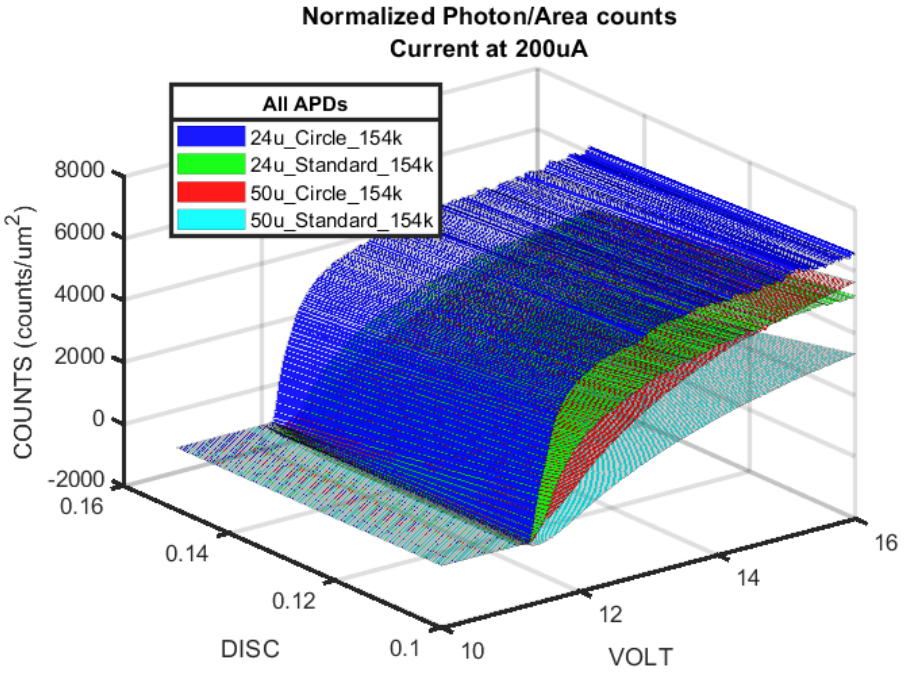


Figure 90 – Photon counts per μm^2 for the 24μm and 50μm APDs with LED current of 200μA

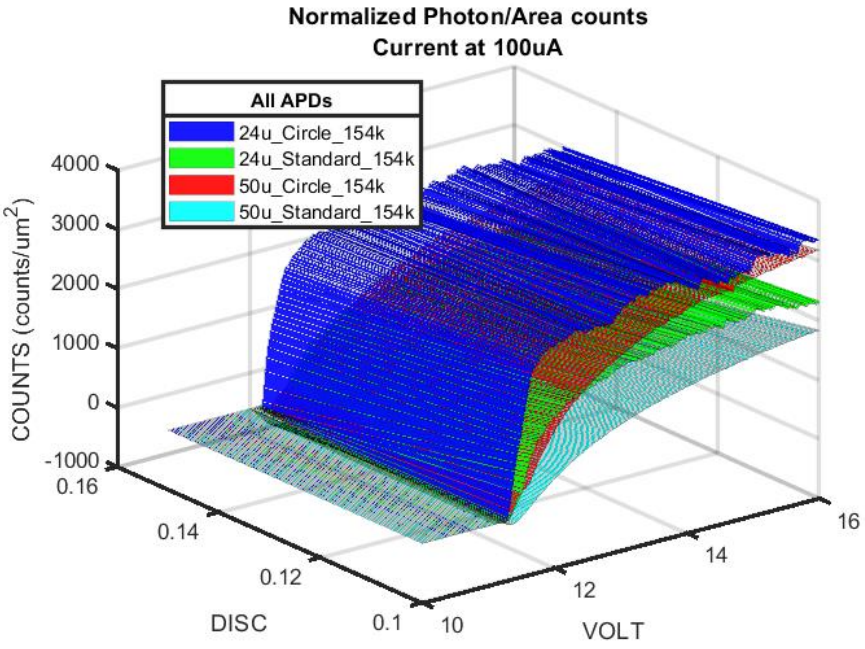


Figure 91 Photon counts per μm^2 for the 24µm and 50µm APDs with LED current of 100µA

Chapter 7 Conclusions

The process of designing an automated photon counting system utilizing GPIB has created a new path for how future APD tests and experiments are conducted. The system in place is currently able to vary reverse bias voltages, illumination LED current level, and different SR430 discriminator thresholds in a much more efficient manner than previous manual testing. An additional test parameter that can be explored in the future is using a temperature chamber to create a four-dimensional plot of all the tests conducted with temperature. Another expansion of this thesis can be in the investigation of how to integrate the current system to communicate with instruments built with RS-232 communication instead of GPIB IEEE-488. From the results of the APDs tested in this thesis, it can be concluded that there is an effective size and shape of SiGe APDs that can be used for future testing. Additional parameters such as testing for quantum efficiency, responsivity, power uniformity, and other parameters are needed to fully characterize the APDs presented through this thesis. The automated photon counting system presented in this thesis has proven itself to extract accurate characterization data from SiGe APDs and will certainly serve as a foundation upon which additional test parameters can be added in the future.

Appendix A: Automated GPIB MATLAB script

```
%% SR430 Photon Counter and Keithley GPIB Communication
% By David Santiago
clear variables
format short
%%%%%%%% Title Editor %%%%%%%%%%%%%
% NOTE: If you press pause or stop the code for any reason,
% send the command:
% fclose(GPIBport);
% else, you'll have to restart MATLAB lol. This closes the COM port.


TestName = "filename_test"; % The name of the test to be conducted

%%% Parameter Creators
% Note, all you have to do is CHANGE the Disc Level's highs/lows/precision,
% also, you can change the LED Currents highs/lows/precision.
% No need to create any names, the code does it on its own lol

%%% Make sure to check the voltage ranges!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%   Test Editor     %%%%%%%%%%%%%%%%%%%%%%%%%
% In Volts
lowVolt = 10;    % Low Voltage range
highVolt = 20;   % Upper volt range
precision = 1e0;  % Voltage Precision

lowDisc = 50e-3;     % Low Discriminator range
precisionDisc = 50e-3; % Disc. Range
highDisc = 150e-3;    % Upper Disc. Range

discUnit = 1e-3;     % The Units of the Disc Voltage (for mV, type 1e-3)


% In amps
lowCurr = 0;                % Low Current Range
precisionCurr = 100e-6;     % Precision Current
highCurr = 100e-6;            % High Current Range

currUnit = 1e-6; % The Units of the LED Current (for uA, type 1e-6)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% LEAVE EVERYTHING ALONE BELOW

%%% Filename generator for LED current
tempCount = 1;
for CurrNamePos = lowCurr:precisionCurr:highCurr

    if currUnit == 1e-6
        LEDCVNPrefix = "uA";
    elseif currUnit == 1e-3
        LEDCVNPrefix = "mA";
```

```
    elseif currUnit == 1e-0
        LEDCVNPrefix = "A";
    else
        LEDCVNPrefix = "units";
    end

    LEDCurrentValueName(1,tempCount) = num2str(CurrNamePos./currUnit) +
LEDCVNPrefix;
    tempCount = tempCount + 1;
end

LEDCurrentSuffix = "Current";

%%% Filename generator for discriminator voltage levels
tempCount = 1;
for DiscNamePos = lowDisc:precisionDisc:highDisc

    if discUnit == 1e-6
        DVNPrefix = "uV";
    elseif discUnit == 1e-3
        DVNPrefix = "mV";
    elseif discUnit == 1e-0
        DVNPrefix = "V";
    else
        DVNPrefix = "units";
    end


    DiscValueName(1,tempCount) = num2str(DiscNamePos./discUnit) + DVNPrefix;
    tempCount = tempCount + 1;
end
DiscSuffix = "DiscLvl";

LightMinusDarkSuffix = "LightMinusDark"; % Light Minus Dark Title at end of
file


DCount = length(DiscValueName);

%%%% Storing filenames into arrays
for Fcount2 = 1:DCount
    filenameDisc(1,Fcount2) = TestName + "_" + DiscValueName(Fcount2) + "_" +
DiscSuffix;

end

LCount = length(LEDCurrentValueName);
for Fcount = 1:LCount
    filenameLight(1,Fcount) = TestName + "_" + LEDCurrentValueName(Fcount)
+"_" + LEDCurrentSuffix; % just give it a name. Make sure it has quotes
around it

end


for Fcount3 = 2:LCount
```

```
filenamePhotonDisc(1,Fcount3-1) = TestName + "_" +
LEDCurrentValueName(Fcount3) + "_" + LightMinusDarkSuffix;

end



savePic = 1;  % 1 to save pic as a fig and a png.  0 to disable
saveData = 1; % 1 to save count data. 0 to disable.



%%% Hardware Configurations %%%
Keithley_SenseCurrent_Range = 10e-3; % this in Amps, either 10mA, 100mA, etc.
                              % Enter a decimal number, this sets the MAX %
                              current range
Keithley_SourceVolt_Range = 30; % In volts. Upper range for Source voltage.
Keithley_CurrLimit = 10e-3; % in amps. Current Limiter when using the voltage
                              % power supply


LED_SenseVolt_Range = 20;      % in Volts, this sets the MAX voltage range
LED_SourceCurr_Range = 1e-3;  % in amps, the upper range for source current
LED_VoltLimit =10;             % in volts. Voltage limiter when using the
                               % current power supply

COMAddr = 3;     % This is the Prologix Controller COM Port number
                 % Pleases enter an integer


% GPIB address setup, values as integers
KeithleyRBVAddr = 18;
SR430Addr = 23;
KeithleyCurrAddr = 19;

%%% Suppressing read error (not needed, but good to have)
suppWarn = 'MATLAB:serial:fgetl:unsuccessfulRead';
warning('off',suppWarn)

%% Setting up COM Port
COM = "COM" + int2str(COMAddr);
GPIBport = serial(COM); % the COM port WILL change, depending on your PC
% CR (Carriage Return) used to terminate internal query responses
% LF (Line Feed) is optional but helps
GPIBport.Terminator = 'CR/LF';

% Timeout is 1 seconds, default is 10 seconds if left alone
GPIBport.Timeout = .5; % half a second chose to speed up timeout

%% Using GPIB
% Opening the port from USB
fopen(GPIBport);

fprintf(GPIBport,"++auto 0") % Turns off read-after-write feature of the
                              % Prologix controller to prevent unneeded reads

%%% Saving address labels to MATLAB variables
```

```
RBVA = "++addr " + int2str(KeithleyRBVAddr);
SR430A = "++addr " + int2str(SR430Addr);
CurrA = "++addr " + int2str(KeithleyCurrAddr);

%%%% Part one: Prepare Keithleys, GPIB Addr: 18 %%%%
fprintf(GPIBport,RBVA); % Accessing the Keithley GPIB address
SensCurr = "SENS:CURR:RANG " + num2str(Keithley_SenseCurrent_Range);
SourceV = "SOUR:VOLT:RANG " + num2str(Keithley_SourceVolt_Range);
CurrLim = "SOUR:VOLT:ILIM " + num2str(Keithley_CurrLimit);

%%% SETTING UP KEITHLEY VOLTAGE SOURCE, SENDING GPIB COMMANDS
fprintf(GPIBport,SensCurr); % This range setting is required
fprintf(GPIBport,SourceV);
fprintf(GPIBport,CurrLim);
pause(1/10); % a delay of 0.1 seconds in MATLAB

KeithleyOn(GPIBport)    % Turn on output of First Keithley Voltage Source

pause(1/2)
%%% Setup 2nd Keithley, GPIB Addr: 19 %%%
fprintf(GPIBport,CurrA);        % Addr. of 2nd Keithley
SensVoltLED = "SENS:VOLT:RANG " + num2str(LED_SenseVolt_Range);
SourceC = "SOUR:CURR:RANG " + num2str(LED_SourceCurr_Range);
VoltLim = "SOUR:CURR:VLIM " + num2str(LED_VoltLimit);

%%% SETTING UP KEITHLEY CURRENT SOURCE, SENDING GPIB COMMANDS
fprintf(GPIBport,"SOUR:FUNC CURR");
fprintf(GPIBport,"SENS:FUNC ""VOLT""");
fprintf(GPIBport,"SOUR:CURR 0");
fprintf(GPIBport,SensVoltLED);
fprintf(GPIBport,SourceC);
fprintf(GPIBport,VoltLim);

KeithleyOn(GPIBport)    % Turn on output of 2nd Keithley


%%% Setting up ranges and a MATLAB timer
% Here, storing ranges into vectors.
voltRange = (lowVolt:precision:highVolt).';
discLvlRange = (lowDisc:precisionDisc:highDisc).';
currLvlRange = (lowCurr:precisionCurr:highCurr).';

totalCounter = length(voltRange)*length(discLvlRange)*length(currLvlRange);

% Timer setup
Time = 8.51*totalCounter; % An estimate for how long it will take to perform
                          % a test
if(Time >= 60)
    Time = Time/60;
    minTime = fix(Time);
    secTime = (Time - minTime)*60;
    fprintf('Estimated Time left: %6.0f minutes and %2.0f seconds
\n',minTime,secTime)
else
    fprintf('Estimated Time left: %6.2f seconds \n',Time)
end
tic;
```

```
% The BIGGEST loop
currCounter = 0;
for curr = lowCurr:precisionCurr:highCurr
fprintf(GPIBport,CurrA);
fprintf('\nStarting at Current Level %d \n',curr);

KeithleyCurrent(curr,GPIBport);

currCounter = currCounter + 1;

% Even Bigger loop to do discriminator levels
discCounter = 0;
for disc = lowDisc:precisionDisc:highDisc
    fprintf(GPIBport,SR430A);   % SR430 address
    strDiscLvlCMD = "DCLV";
    strDiscLvlVal = num2str(disc);
    DiscLvlString = strDiscLvlCMD + strDiscLvlVal;
    fprintf('\nStarting at Disc Level %s \n',DiscLvlString);
    fprintf(GPIBport,DiscLvlString);
    discCounter = discCounter + 1;

    % BIIIG Loop to pump up voltage
    counter = 0;
    for voltage = lowVolt:precision:highVolt
        fprintf('Current Voltage: %2.4f Volts \n',voltage)
        tLoop = tic;
        fprintf(GPIBport,RBVA);
        KeithleyVoltage(voltage,GPIBport); %Voltage set

        % Setup the counter address
        fprintf(GPIBport,SR430A);
        SR430startCount(GPIBport); % Script will pause and wait

        counter = counter + 1; %Finished counting
        totalCounter = totalCounter - 1;


        SR430GetCounts(GPIBport);

        % Read whatever prev command spits out
        SR430Received = fgetl(GPIBport);
        counts(counter,discCounter) = str2double(SR430Received);
        disp(SR430Received);

        % Done with counting, increment voltage
        % and do it again
        photonTimeLeft(tLoop,totalCounter)
        photonTimeElapsed
    end
end

% Store Complete counts to a bigger array
KompleteCounts(:,:,currCounter) = counts;
```

```
end



fprintf(GPIBport,RBVA); % Turning off the Keithley
KeithleyOff(GPIBport);

pause(1/10)
fprintf(GPIBport,CurrA); % Turning off the Keithley
KeithleyOff(GPIBport);


% Close the port, send data to Excel and a graph
fclose(GPIBport);
%% Workspace
% Graphing 3D plot, X -> disc, Y-> Volts, Z-> Counts, each graph is a
% different light state
mkdir(TestName, 'LED_Current')
currentFolder = pwd;

for k = 1:currCounter

[row,col] = size(KompleteCounts(:,:,k));

for i = 1:row
    for j = 0:col-1
        discAxis(i,j+1) = j*precisionDisc+lowDisc;
        voltAxis(i,j+1) = lowVolt +(i-1)*precision;
    end
end

figure
if col == 1
    plot(voltAxis,KompleteCounts(:,col,k))
    ylabel('Counts')
    xlabel('Reverse Bias Voltage')

else
    surf(discAxis,voltAxis,KompleteCounts(:,:,k))
    xlabel('Discriminator Level')
    ylabel('Reverse Bias Voltage')
    zlabel('Counts')
end
title(filenameLight(k),'Interpreter','none')
prettyGraph
picFN = filenameLight(k) + "_pic";

LEDFigPath = fullfile(currentFolder,TestName,'LED_Current',picFN);
if(savePic)

    print(LEDFigPath,'-dpng')
    savefig(LEDFigPath)
end

countsFile = KompleteCounts(:,:,k);
```

```
voltFile = voltRange;

data1 = [voltFile,zeros(length(voltFile),1),countsFile];
data1Disc = [NaN(1,2),discLvlRange.'];
data1T = vertcat(data1Disc,data1);

LEDFN = filenameLight(k)+".xlsx";

LEDDataPath = fullfile(currentFolder,TestName,'LED_Current',LEDFN);

if(saveData)
    xlswrite(LEDDataPath,data1T)
end

end



%% Workspace
% 3D plots of X -> curr, Y-> Volts, Z-> Counts, each graph is a different
% discriminator level
mkdir(TestName,'Disc_Level')
for m = 1:discCounter

countsDisc = squeeze(KompleteCounts(:,m,:));

[row,col] = size(countsDisc);


[currAx,voltAx] = meshgrid(currLvlRange,voltRange);

figure
if col == 1
    plot(voltAxis,squeeze(KompleteCounts(:,m,col)))
    ylabel('Counts')
    xlabel('Current Level')

else
    surf(currAx,voltAx,countsDisc)
    xlabel('Current Level')
    ylabel('Reverse Bias Voltage')
    zlabel('Counts')
end
title(filenameDisc(m),'Interpreter','none')
prettyGraph
picFN = filenameDisc(m) + "_pic";

DiscFigPath = fullfile(currentFolder,TestName,'Disc_Level',picFN);


if(savePic)
    print(DiscFigPath,'-dpng')
    savefig(DiscFigPath)
end

countsFile = countsDisc;
voltFile = voltRange;
```

```
data2 = [voltFile,zeros(length(voltFile),1),countsFile];
data2Curr = [NaN(1,2),currLvlRange.'];
data2T = vertcat(data2Curr,data2);

DiscFN = filenameDisc(m)+".xlsx";
DiscDataPath = fullfile(currentFolder,TestName,'Disc_Level',DiscFN);


if(saveData)
    xlswrite(DiscDataPath,data2T)
end

end


%% Workspace
% Light Minus Dark in 2D, Subtracting all current levels with 0uA current
mkdir(TestName,'Light_Minus_Dark')

darkCounts = KompleteCounts(:,:,1); % 0uA matrix
for k = 2:currCounter


[row,col] = size(KompleteCounts(:,:,k));

[discAx,voltAx] = meshgrid(discLvlRange,voltRange);

photonCounts = KompleteCounts(:,:,k) - darkCounts;


figure
if col == 1
    plot(voltAxis,KompleteCounts(:,col,k))
    ylabel('Counts')
    xlabel('Reverse Bias Voltage')

else
    surf(discAx,voltAx,photonCounts)
    xlabel('Discriminator Level')
    ylabel('Reverse Bias Voltage')
    zlabel('Counts')
end



title(filenamePhotonDisc(k-1),'Interpreter','none')
prettyGraph
picFN = filenamePhotonDisc(k-1) + "_pic";

LMDFigPath = fullfile(currentFolder,TestName,'Light_Minus_Dark',picFN);

if(savePic)
    print(LMDFigPath,'-dpng')
    savefig(LMDFigPath)
end
```

```
countsFile = photonCounts;
voltFile = voltRange;

data3 = [voltFile,zeros(length(voltFile),1),countsFile];
data3Disc = [NaN(1,2),discLvlRange.'];
data3T = vertcat(data3Disc,data3);

LMDFN = filenamePhotonDisc(k-1)+".xlsx";

LMDDataPath = fullfile(currentFolder,TestName,'Light_Minus_Dark',LMDFN);


if(saveData)
    xlswrite(LMDDataPath,data3T)
end

end


%% Adding plots - Light Vs Dark

if currCounter == 1 % If just one curve
    disp('Nothing to add :)')
else
    [row,col] = size(KompleteCounts(:,:,1));

        for i = 1:row
            for j = 0:col-1
                discAxis(i,j+1) = j*precisionDisc+lowDisc;
                voltAxis(i,j+1) = lowVolt +(i-1)*precision;
            end
        end

    figure

    if col == 1
        plot(voltAxis,KompleteCounts(:,col,1),'-k','linewidth',2)
        ylabel('Counts')
        xlabel('Reverse Bias Voltage')

        hold on
        plot(voltAxis,KompleteCounts(:,col,end),'-b','linewidth',2)
        hold off

    else

surf(discAxis,voltAxis,KompleteCounts(:,:,1),'FaceAlpha',0.5,'FaceColor','k')
        xlabel('Discriminator Level')
        ylabel('Reverse Bias Voltage')
        zlabel('Counts')

        hold on

surf(discAxis,voltAxis,KompleteCounts(:,:,end),'FaceAlpha',0.5,'FaceColor','b
')
        hold off
    end
```

```
end

filenameAddLightDark = TestName + "_" + "Dark_and_" +
LEDCurrentValueName(1,end);

title(filenameAddLightDark,'Interpreter','none')
prettyGraph

lgd = legend('Dark',LEDCurrentValueName(1,end));
title(lgd,'LED Current')


picFN = filenameAddLightDark + "_pic";

LMDFigPath2 = fullfile(currentFolder,TestName,picFN);


if(savePic)
    print(LMDFigPath2,'-dpng')
    savefig(LMDFigPath2)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% Functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function KeithleyOn(sPort)
% This Function will make the Keithley
% turn ON whatever is the Source mode
    fprintf(sPort,'OUTP:STAT ON');   % Command to turn Keithley ON
end
function KeithleyVoltage(inputVolt,sport)
% This function will set the voltage of the Keithley.
% The first parameter is a value of type double
% The second parameter is your port
% EX.
% KeithleyVoltage(1.0,yourPort);
    voltToStr   = num2str(inputVolt);
    cmdVoltStr  = 'SOUR:VOLT';
    sourceStr = cmdVoltStr + " " + voltToStr;
    fprintf(sport,sourceStr);
    pause(1/10)
end

function KeithleyCurrent(inputCurr,sport)
% This function will set the Current of the Keithley.
% The first parameter is a value of type double
% The second parameter is your port
% EX.
% KeithleyCurrent(1e-3,yourPort);
    currToStr   = num2str(inputCurr);
    cmdCurrStr  = 'SOUR:CURR';
    sourceStr = cmdCurrStr + " " + currToStr;
    fprintf(sport,sourceStr);
    pause(1/10)
end

function KeithleyOff(serialp)
```

```
% This Function will make the Keithley
% turn OFF the Output.
% Make sure you have your port as a parameter
% EX.
% KeithleyOFF(yourPort);
        fprintf(serialp,'OUTP:STAT OFF');    %Turn off
end
function SR430startCount(sp)
% This Function starts a new photon count record
% Make sure you make the first parameter your port
% EX: SR430startCount(yourPort);
    fprintf(sp,'PAUS');
    fprintf(sp,'CLRS');
    fprintf(sp,'SSCN');

    % Automatic Code
%     fprintf(sp,'*STB? 0');
%     fprintf(sp,'++read');
%     r = fgetl(sp);
%     bool = str2double(r);
%     while bool ~= 1
%         fprintf(sp,'*STB? 0');
%         fprintf(sp,'++read');
%
%         r = fgetl(sp);
%         bool = str2double(r);
%     end
%     disp('End')
    % End of Automatic

    % Manual Code
    pause(6.3);
    % End of Manual
end
function SR430GetCounts(port)
% This Function will sum up all visible bins (locally
% in the SR430) and send the total sum
    fprintf(port,'STAT');
    pause(1)
    fprintf(port,'SPAR ? 2'); % Get all counts
    fprintf(port,'++read');
end
function exportData(stringF,v,d)
% This Function will export data to a new Excel File
% Make First parameter have a string of your choice (NO "-" or ".")
% Second parameter is the volt range as 1 big column
% Third parameter is the count data, already as a big column
% EX.   exportData('myFile',voltRange,counts);
    filename = strcat(stringF,'.xlsx'); % new filename
    xlswrite(filename,[v,d]) % Creates new Excel File
end
function photonTimeLeft(t,totalCounter)
% This will tell you how much time is left using the total counts left
% t is of type unsigned integer 64bit. Just use t = tic before you use
% this function
    T = toc(t);
    estTime = T*totalCounter;
```

```
    if(estTime >= 60)

        estTime = estTime/60;
        minTime = fix(estTime);
        secTime = (estTime - minTime)*60;

        fprintf('Estimated Time left: %4.0f minutes and %2.0f seconds
\n',minTime,secTime)
    else
        fprintf('Estimated Time left: %4.2f seconds \n',estTime)
    end
    fprintf('Points left: %u \n',totalCounter)
end
function photonTimeElapsed
% This Function will tell you how much time has passed
    Q = toc;
    if(Q >= 60)

         Q = Q/60;
         minTime = fix(Q);
         secTime = (Q - minTime)*60;

         fprintf('Time Elapsed: %4.0f minutes and %2.0f seconds
\n',minTime,secTime)
    else
         fprintf('Time Elapsed: %4.2f seconds \n',Q)
    end

end
function prettyGraph % This function is just to
grid on              % Have nice looking graphs
ax = gca;            %
ax.FontSize = 11;
ax.LineWidth = 1;
set(gcf,'color','w');
end
```

Appendix B: MATLAB Script to Generate Plots

```
%% Data Processing
% By David Santiago
%
% This script takes the data folders obtained
% from the Automated GPIB MATLAB script and
% combines all data into a single figure.
% Create a new folder, where data folders of
% interest are saved into that folder. Save this
% script in same folder and run.
clc
clear variables
close all
LED_curr_low = 0;
LED_curr_high = 200e-6;
precision = 10e-6;
curr_levels = length(LED_curr_low:precision:LED_curr_high);

%% Find the Data Folders
folderName = findPhotonFolder;
numPhotonFolders = length(folderName);
%% EXTRACT DATA
indexCurr = [1,3,14:21,2,4:13]; % For indexing purposes


for i = 1:numPhotonFolders
    folder = string(folderName(i));
    tempDirLED = "./" + folder + "/LED_Current";

    d=dir(fullfile(tempDirLED,'*.xlsx*'));  % return the .xls files in the
given folder
    for j=1:length(d)
        tempF = d(indexCurr(j)).name;     % take filename and convert
        tempF = tempF(1:end-5);                 % to string
        tempS = convertCharsToStrings(tempF);

        KompleteFileNames(j,i) = tempS; % store ALL filenames
        % Set up the Import Options and import the data
        opts = spreadsheetImportOptions("NumVariables", 2);

        % Specify sheet and range, x: disc voltage, y: count data
        opts.Sheet = "Sheet1";
        opts.DataRange = "C2:D302";

        % Specify column names and types, discriminator voltage
        opts.VariableNames = [tempF+"_100mV", tempF+"_150mV"];
        opts.VariableTypes = ["double", "double"];

        % Import the data
        tbl = readtable(tempDirLED+"/"+tempF, opts, "UseExcel", false);

        % Convert to output type
        temp100mV = table2array(tbl(:,1));
        temp150mV = table2array(tbl(:,2));
        countTemp = [temp100mV,temp150mV];
```

```
        %% Clear temporary variables
        clear opts tbl
        KompleteCounts(:,:,j,i) = countTemp; % store all counts into matrix
    end
end
warn = warning('query','last'); % turn off warning
warnID = warn.identifier;
warning('off',warnID)

%% DATA REPRINTING
% This reprints all data into individual figures
% This will produce the most figure windows, so be prepared
% for many windows to pop up
voltage = 10:20e-3:16; % voltage range
V = [voltage;voltage].';
disc = 100e-3:50e-3:150e-3; % disc range
D = disc.*ones(301,2);

for k = 1:i % Sweeping different test result folders

    for m = 1:j % Sweeping different current tests
        figure
        surf(V,D,KompleteCounts(:,:,m,k))
        title(KompleteFileNames(m,k),'interpreter','none')
        ylabel('DISC');
        xlabel('VOLT');
        zlabel('COUNTS');
    end
end


%% DATA PROCESSING: All APDS
% This recombines all APD data into a figure.
% Each figure represents an LED current drive
color = ['b','g','r','c','m','y'];
current = 0:10:200;
for m = 1:j

        figure
    for k = 1:i

surf(V,D,KompleteCounts(:,:,m,k),'FaceAlpha',0.9,'EdgeAlpha',0.2,'FaceColor',
color(k))
        hold on
    end
    title({"Total counts", "Current at "+
num2str(current(m))+"uA"},'interpreter','none')
    ylabel('DISC');
    xlabel('VOLT');
    zlabel('COUNTS');
    hold off
    legend(folderName,'interpreter','none','location','best')
    title(legend,'All APDS')
    prettyGraph

end
```

```
%% Photon Processing: All APDS
% This will take the dark (low reference) counts
% and subtract this value from all the light counts.
% This will result in counts due to incident photons
color = ['b','g','r','c','m','y'];
current = 0:10:200;
for m = 2:j

        figure
    for k = 1:i
        surf(V,D,KompleteCounts(:,:,m,k)-
KompleteCounts(:,:,1,k),'FaceAlpha',0.9,'EdgeAlpha',0.2,'FaceColor',color(k))
        hold on
    end
    title({"Photon counts", "Current at "+
num2str(current(m))+"uA"},'interpreter','none')
    ylabel('DISC');
    xlabel('VOLT');
    zlabel('COUNTS');
    hold off
    legend(folderName,'interpreter','none','location','best')
    title(legend,'All APDs')
    prettyGraph

end
%% Data Processing: Normalized Photon/Area Counts
% This will take the photon data and divide it by the area of
% the APD. Note you may have to change the area
diameter = [24e-6,50e-6,5e-6];
radius = diameter./2;
AreaCircle = pi.*radius.^2;
AreaSquare = diameter.^2;
Area =
[AreaCircle(1),AreaSquare(1),AreaCircle(2),AreaSquare(2),AreaCircle(3),AreaSq
uare(3)];

color = ['b','g','r','c','m','y'];
current = 0:10:200;
for m = 2:j

        figure
    for k = 1:i
        surf(V,D,1e-12*(KompleteCounts(:,:,m,k)-
KompleteCounts(:,:,1,k))./Area(k),'FaceAlpha',0.9,'EdgeAlpha',0.2,'FaceColor'
,color(k))
        hold on
    end
    title({"Normalized Photon/Area counts", "Current at "+
num2str(current(m))+"uA"},'interpreter','none')
    ylabel('DISC');
    xlabel('VOLT');
    zlabel('COUNTS (counts/um^2)');
    hold off
    legend(folderName,'interpreter','none','location','best')
    title(legend,'All APDs')
    prettyGraph
```

```
end
%% Functions
function photonFolderNames = findPhotonFolder(string)
% This function will look at current working directory for photon data
% folders.

if (nargin==1)
    cd = string;
else
    cd = pwd;
end
files = dir(cd); % pwd is the command to look at current directory
flag = [files.isdir]; % marks which file is a directory
photonFolder = files(flag); % returns only directories
photonFolderNames = {photonFolder(3:end).name}; % Start at 3 to
                                                % skip . and .. directories
End

function prettyGraph % This function is just to
grid on              % Have nice looking graphs
ax = gca;            %
ax.FontSize = 10;
ax.LineWidth = 2;
set(gcf,'color','w');
end
```

Appendix C: First-Time MATLAB Global Variable Setup

First-Time MATLAB Setup		
Variable	**Data**	**Notes**
COMAddr	3	The COM port address assigned to the GPIB Prologix controller
Keithley Voltage Source (to reverse bias the diode)		
Keithley_SenseCurrent_Range	10e-3 (10mA)	Sets the sense current setting to 10mA
Keithley_SourceVolt_Range	30	Sets the voltage range to 30V
Keithley_CurrLimit	10e-3 (10mA)	Sets max current to output at 10mA
Keithley Current Source (for LED)		
LED_SenseVolt_Range	20V	Sets the sense voltage setting to 20V
LED_SourceCurr_Range	1e-3 (1mA)	Sets the current range to 1mA
LED_VoltLimit	10V	Sets max voltage to output at 10V
GPIB Settings		
SR430Addr	23	The GPIB address of the SR430
KeithleyRBVAddr	18	The GPIB address of the Keithley Voltage source
KeithleyCurrAddr	19	The GPIB address of the Keithley Current source

Appendix D: MATLAB User-Defined Test Variables

MATLAB User-Defined Test Variables		
Variable	**Data**	**Notes**
TestName	"your_testName"	The name of the test. Must start with a letter, and can contain alphabetical chracters, digits, or underscores
Keithley Voltage Source (to reverse bias the diode)		
lowVolt	10	The lowest voltage to be tested
highVolt	20	The highest voltage to be tested
precision	1e-3 (1mV)	Upward increments from lowest voltage to highest voltage
SR430 Photon Counter		
lowDisc	50e-3 (50mV)	Lowest discriminator voltage
highDisc	150e-3 (150mV)	Highest discriminator voltage
precisionDisc	50e-3 (50mV)	Upward increments from lowest discriminator voltage to highest.
Keithley Current Source (for LED)		
lowCurr	0 (or reference)	Lowest current to output. For dark counts, set this to 0.
highCurr	100e-6 (100μA)	Highest current to output. This may be set as the maximum current the LED can be safely tested.
precisionCurr	10e-6 (10μA)	Upward increments from lowest current to highest.
Miscellaneous		
discUnit	1e-3 (1mV)	For labeling purposes. Attaches appropriate engineering metric unit
currUnit	1e-6 (1μA)	For labeling purposes. Attaches appropriate engineering metric unit

Appendix E: MATLAB User-Defined Functions

User-Defined Functions	
Keithley SourceMeter (General)	
Function	**Notes**
KeithleyOn(sPort)	This function will turn on the source output of a Keithley power supply. The supply to be turned on is determined by the current GPIB address set on the Prologix controller.
KeithleyOff(sPort)	This function will turn off the source output of a Keithley power supply. The supply to be turned off is determined by the current GPIB address set on the Prologix controller.
Keithley Voltage Supply	
KeithleyVoltage(inputVolt,sPort)	This function will set the voltage on the Keithley voltage supply. This function must be sent after setting the correct GPIB address on the Prologix controller. A small arbitrary pause of 1/10 of a second occurs at the end of the function. "inputVolt" is a value of type double. Ex: KeithleyVoltage(1.0,sPort) sets the voltage on the Keithley to 1 volt.
Keithley Current Supply	
KeithleyCurrent(inputCurr,sPort)	This function will set the current on the Keithley current supply. This function must be sent after setting the correct GPIB address on the Prologix controller. A small arbitrary pause of 1/10 of a second occurs at the end of the function. "inputCurr" is a value of type double. Ex: KeithleyCurrent(1e-3,sPort) sets the current on the Keithley to 1 milliamp.
SR430	
SR430startCount(sPort,mode)	This function starts a scan on the SR430. Prior to running this function, the GPIB address on the Prologix controller must be changed to the SR430. By default, the automatic mode is set. "mode" is an integer, where a value of 1 will turn on the user-defined mode. The time inside this function must be calculated from Ch. 2 of this thesis.
SR430GetCounts(sPort)	This function will sum all visible time bins (locally on the SR430 screen, see Ch. 2) and send the total count to MATLAB.

Miscellaneous	
exportData(string,v,d)	This function will export data to a new Excel file. “stringF” is a character string, and will serve as the filename of the Excel file. “v” is a vector of type double, which is defined by the reverse bias voltage range. “d” is a vector of type integer, and is the count data.
photonTimeLeft(t,totalCounter)	This function will output the estimated time left using the total counts left. “t” is of type unsigned integer, which is the function “tic” which when used with “toc” will return the difference in time between the two functions, in minutes and seconds.
photonTimeElapsed	This function will output the total time elapsed at the time of execution, in minutes and seconds.
prettyGraph	This function will format the current MATLAB figure with a grid, font size, width of the gridlines, and color the background white. This function is for presentation purposes.

References

[1] Roychoudhuri, C., Kracklauer, A. F., & Creath, K. (Eds.). (2019). *Nature of light: What is a photon?* CRC Press, Chapter 3.

[2] Mohan, Nishant, et al. “Photon-Counting Optical Coherence-Domain Reflectometry Using Superconducting SINGLEPHOTON Detectors.” *Optics Express*, Optica Publishing Group, 22 Oct. 2008, https://opg.optica.org/oe/fulltext.cfm?uri=oe-16-22-18118&id=172970.

[3] Fletcher, Jenna. “Central Serous Retinopathy: What It Is, Symptoms, and Treatment.” *Medical News Today*, MediLexicon International, 13 Jan. 2018, https://www.medicalnewstoday.com/articles/320606.

[4] Donati, Silvano. “Semiconductor Photodetectors.” Photodetectors: Devices, Circuits and Applications, Wiley-IEEE Press, Upper Saddle River, NJ, 2000, pp. 109–110.

[5] Pearsall, T. P. “Quantum Efficiency.” *Photonics Essentials*, McGraw-Hill, New York, 2010, p. 52.

[6] Stanford Research Systems. “SR430 Manual.” May 2013. Operator’s manual.

[7] Fisher, Eugene R., and C. W. Jensen. “Layman's Introduction to the GPIB.” *PET and the IEEE 488 Bus (GPIB)*, OSBORNE/McGraw-Hill, 1982, p. 6.

[8] Baker, R. J, *CMOS Circuit Design, Layout, and Simulation, Fourth Edition*, John Wiley and Sons, Inc. , 2010. pp. 43.

[9] “Photodiode Basics.” *Wavelength Electronics*, Wavelength Electronics, 11 Feb. 2020, https://www.teamwavelength.com/photodiode-basics/.

[10] Hamamatsu Photonics, *Si Photodiode*, pp. 3.

[11] Gentry, D. (2018) *Design, Layout, and Testing of SiGe APDs Fabricated in a BiCMOS Process* [master's thesis]. University of Nevada, Las Vegas.

[12] *Prologix. "GPIB-USB Controller." May 2013. http://prologix.biz/downloads/PrologixGpibUsbManual-6.0.pdf*

[13] Stanford Research Systems. "SR430 Manual." May 2013. Operator's manual.

[14] Keithley. "Model 2450 Interctive SourceMeter Instrument." May 2015. Reference manual.

Curriculum Vitae

DAVID SANTIAGO
david.aristeo.santiago@gmail.com
http://www.linkedin.com/in/david-santiago-6922b0195
CMOSedu.com/jbaker/students/david/david.htm

EDUCATION
University of Nevada, Las Vegas – B.S. in Electrical Engineering, May 2020
GPA: 3.64
University of Nevada, Las Vegas – M.S. in Electrical Engineering, Est. May 2023
GPA: 4.00

EXPERIENCE
JT4,LLC Engineer I ***September 2022 – Present***

Freedom Photonics Layout Design Engineering Intern ***December 2021 – June 2022***
- Completed layout designs in TowerJazz's SBC18HA SiGe BiCMOS 180nm process with radiation hardening techniques
- Successfully completed a $5mm^2$ tape-out mission

UNLV Teaching Assistant ***August 2020 – Present***
- Prepared remote lesson plans for electrical and computer engineering students
- Provided quality feedback to enhance student learning
- Courses: Signal Processing (Analog/Digital), Circuits, Electromagnetics

UNLV Research Assistant ***January 2019 - Present***
- Conducted research for Dr. R. Jacob Baker at UNLV, with a focus on data analysis, circuit design, instrumentation, and board debugging.
- IC design/ layout using Cadence Virtuoso in ON Semiconductor's C5 and TowerJazz's SBC18HA 180nm processes

Tau Beta Pi, The Nevada Beta Chapter Leadership: Fall 2019 VP, President 2020 Fall 2019 – Fall 2020
- Managed the only engineering honor society at the UNLV College of Engineering
- Maintained good communication with the college, members, future initiates, as well as with the national executives

ADDITIONAL SKILLS
Analog/Digital Circuit Design, Assembly Language, Automated Test Equipment, C, C++, CAD, Cadence Virtuoso, Digital Filter Design, Diva DRC, Eagle, Embedded Systems Design, GPIB Interfacing, IC Design/Layout, Linux Command Line, LTspice, MATLAB, ModelSim, PCB Design, Quartus, Signal Processing, Soldering, Verilog