Title

Design of a High-Speed Direct Electron Imager for High Vacuum

Author

Gloekler, Ryan Edward

Publication Date

2024

Peer reviewed|Thesis/dissertation

Design of a High-Speed Direct Electron Imager for High Vacuum

By

RYAN E. GLOEKLER

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL AND COMPUTER ENGINEERING

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Charles E. Hunt, Chair

William Putnam

Rajeevan Amirtharajah

Committee in Charge

2024

Acknowledgement

This research endeavor would not have been possible without the continued support of my committee chair and principal investigator Dr. Charles E. Hunt, and Arthur Carpenter of Lawrence Livermore National Laboratory (LLNL). I am especially thankful to my colleagues in the Vacuum Microelectronics Laboratory, especially MA Mort and Jordan Ricci for their contributions and incredible teamwork during our collaboration on this project. Finally, I would like to express gratitude to my family for their motivation and support throughout my academic journey.

Abstract

A data collection system was developed to enable high-speed imaging of electrons in a vacuum, aiding the development of contemporary x-ray imaging technologies. The device was tested in response to visible light at atmosphere and vacuum, and the response to an electron gun was characterized. The system was used to obtain the 53% optical quantum efficiency (QE) of the sensor and compare against theoretical maxima. Amplifier printed circuit boards (PCBs) were designed to convert sensor photocurrent into a voltage, and a microcontroller was used to transmit imager data. Data was received and processed by a custom software system. Imaging hardware is triggered using an 8-ms pulse, coincident with a solenoid-based loop-focusing device trigger. Data is collected over 10 ms at a frequency of 10 kHz. The amplifier PCB was adjusted to be modular and allow for hardware-based sensitivity adjustments. Linear voltage-response to input photocurrent was verified by collecting visible light data of differing intensities. A positively biased phosphor plate was used to verify the electron gun configuration. Despite the electron sensitivity of the photodiode array, the observed noise characteristics of the sensor in response to the electron gun demonstrated that the electron beam had insufficient power to guide electrons to the sensor interface. The design of the imaging system was successful and suggests it may be a suitable replacement for existing x-ray imaging technology.

Table of Contents

Chapter 1

1.1 Background

Developing x-ray imaging systems use a combination of electrical and optical signals to reconstruct and image an incident x-ray signal with high spatial and temporal resolutions. This is accomplished by first converting the incident signal to electrons with the combination of a photocathode and Microchannel Plate (MCP), acting as an electron amplifier. The photogenerated electrons then travel through a vacuum drift chamber, before colliding with a phosphor which subsequently fluoresces, converting the electron signal into a measurable optical output. The converted signal is finally captured from the phosphor plate by a photon imaging device [1] such as a CMOS sensor, which collects the light into individual pixels that can then be read off to generate images of a discrete x-ray event.

Due to the hardware involved in conversion back and forth between photons and electrons, there is a great deal of temporal and spatial resolution loss in the data that is received by the CMOS sensor. MCP's can facilitate high-speed particle and electron detection by delivering well-defined charge clouds but are limited by phosphor screen readout [2]. The fluorescence of the phosphor plate itself is a significant source of spatial resolution loss, especially for sub-nanosecond speed imaging systems [1]. As a result, it is desirable to circumvent the loss associated with electrical and optical conversion steps by imaging electrons directly at the interface. Direct-electron sensitive sensors are in development for such

applications and are capable of ultrafast and high spatial resolution electron detection, but there is a significant need for imaging systems to support them [3].

1.2 Research Description

This research focuses primarily on the development of a high speed, direct-electron imager to verify the viability of direct electron-imaging under high vacuum. The system is designed to image incident electrons generated by a photocathode and MCP pair and may be tested directly using an electron gun under vacuum. The imager triggers a high-current pulse through a solenoid to magnetically focus electrons throughout the length of its drift region, while simultaneously triggering the imager to capture many frames of electron data. The imager uses an electron-sensitive linear photodiode array and produces an output voltage by converting the induced photocurrent with a transimpedance amplifier circuit. The system is driven directly and timed with custom hardware and firmware to capture and store data with a temporal resolution of 100 μs.

The imaging system was tested with visible light at atmosphere and is ready for further direct electron testing under vacuum, which will be performed using an electron source rather than photoelectrons once the electron gun has been properly configured. The transimpedance amplifiers were initially tuned for each test case, based on the estimated and measured maximum photocurrent under each operating condition. Test data samples were stored for analysis, allowing the system to be later tuned by adjusting the driving firmware and transimpedance amplifier circuit parameters individually.

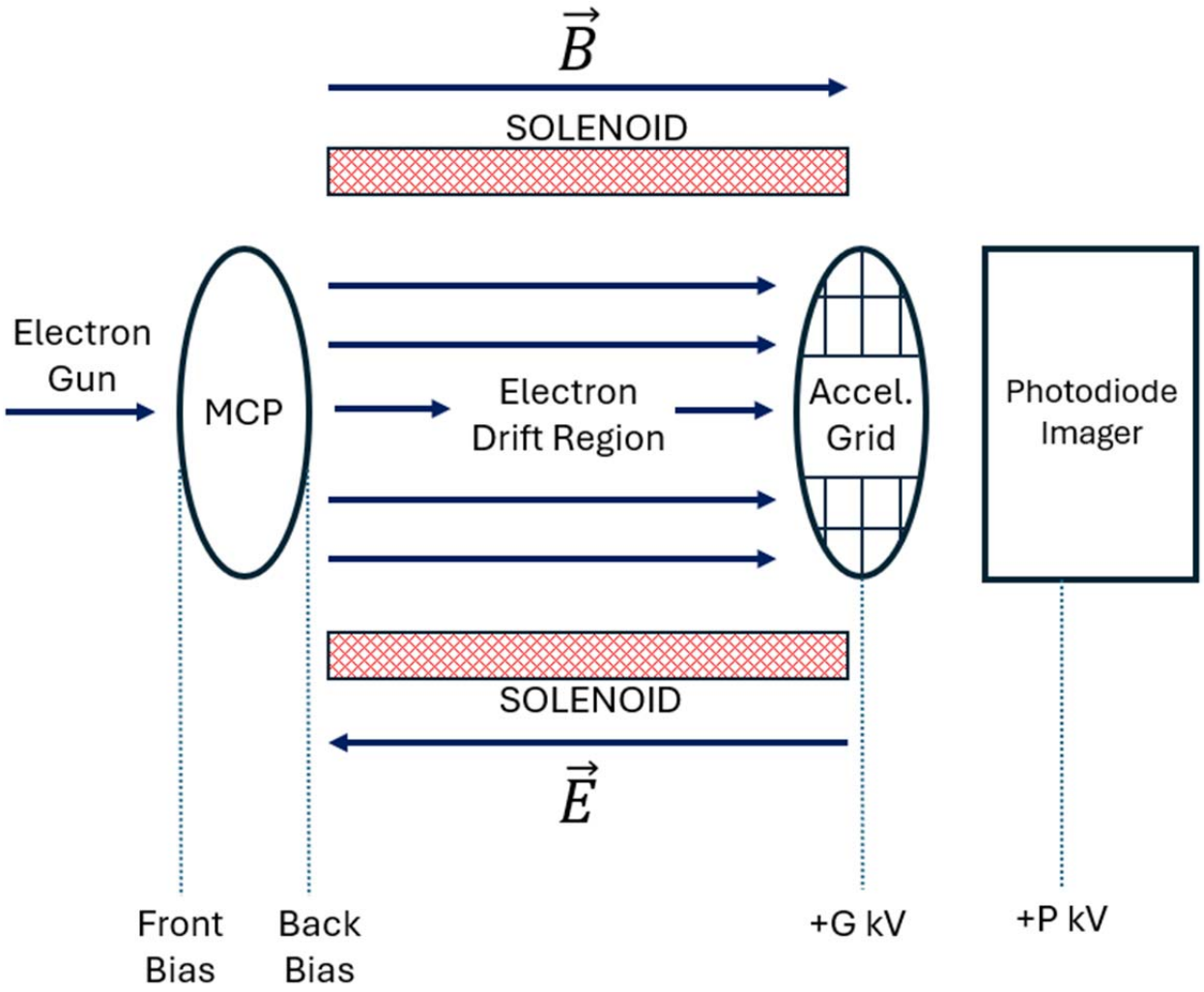

Figure 1: Final photodiode imager setup configured for direct electron testing.

1.3 Solenoid Loop Focusing of Electrons

For the imager to capture frames of electron data in high resolution the electron source needs to be focused, similar to how a camera uses a lens to focus light. While testing with an electron gun, the emitted beam should be focused to decrease spot size and increase spatial resolution at the photodiode interface. In application, such a mechanism is also needed to focus electron charge clouds from an MCP as they travel through the length of the vacuum drift chamber. To accomplish this, a solenoid was constructed to guide electrons from the MCP to the photodiode sensor interface, and was the solenoid pulsed with high current to establish a momentary magnetic field, thereby confining the signal and guiding it to the imager.

The solenoid was constructed using 144 turns of wire over a total length of 8.25 cm and had a measured peak field strength of 340 Gauss with an input current of 17 A. The magnetic field throughout the cylindrical region can be modeled with the following equation, where B is magnetic field strength, μ_0 is the permittivity of free space, z is the axis along which the field points, I is pulsed current, and N and L correspond to number of wire turns and length, respectively [4, pp. 265-267]:

$$\mathrm{B} = \hat{\mathrm{z}}\frac{\mu_0 \mathrm{NI}}{\mathrm{L}} \quad \text{(eq. 1)}$$

1.4 Photodiode Imager Operation

Photodiodes may be used in the context of direct electron imaging due to their operating principle of converting incident photons or electrons into electron-hole pairs at the device interface. The devices are generally constructed using two adjacent semiconductor layers doped to be P-type and N-type, which contain an abundance of holes and electrons, respectively. These two layers create a depletion region, or barrier, which separates electrons and holes due to the built-in electric field. When an incident photon or electron of sufficient energy strikes the depletion region, electron-hole pairs are generated, and electrons and holes move towards opposite ends of the device [5]. The induced current due to the motion of electrons and holes is called the "photocurrent," which is the primary sensing parameter used in a photodiode image sensor and is amplified by the hardware detailed in Chapter 2.2.

Given that each pixel in an array of photodiodes produces its own photocurrent, an array of devices can be used to construct a sensor with spatial resolution determined by the size of each pixel relative to the incident signal. The signal must be focused prior to striking the surface of the imager to maximize the image quality. Pixel read-out can be achieved by converting generated photocurrent into a corresponding voltage, which is sampled to create a digital number representing signal intensity.

1.5 Electron-Sensitive Photodiode Array

The photodiode array was chosen due to its sensitivity to both photons and electrons, allowing the imaging system to be tested with both visible light at atmosphere and direct electrons under vacuum. The array is made up of 16 individual pixels, each having an active area of 2 mm x 5 mm, and a rise time of 500 ns. The device is packaged in a common anode configuration, with eight total anode pins as seen in Figure 2 [6].

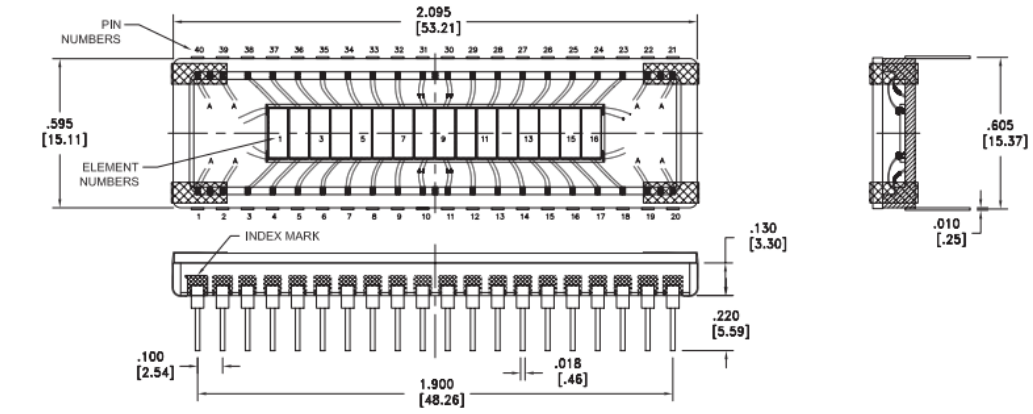

Figure 2: Photodiode array packaging diagram. Schematic from [6].

As shown in Figures 3 and 4, the device exhibits a non-linear typical electron and photon responsivity, so the supporting hardware must be tuned to the expected input signal range with respect to the signal source. This may be done by adjusting transimpedance amplifier circuit parameters as discussed in Section 2.2. The device does not require a bias to operate.

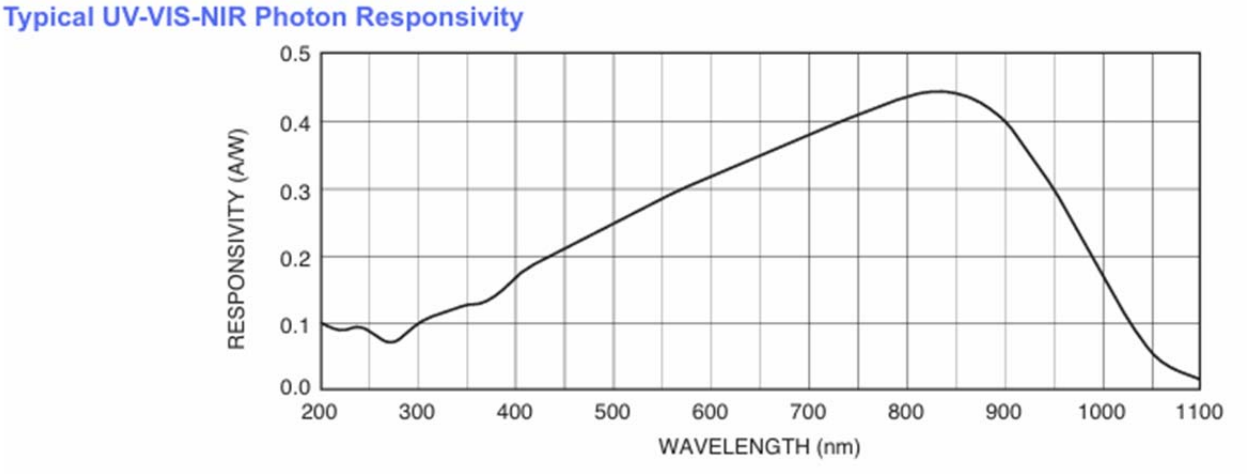

Figure 3: Typical visible light photon response in A/W of the photodiode array. Data from [6].

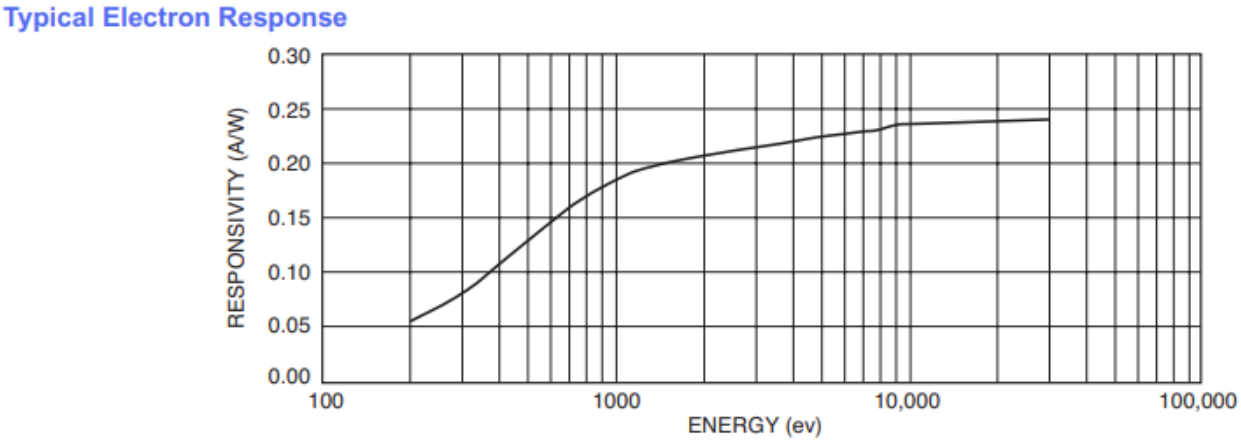

Figure 4: Typical electron responses in A/W of the photodiode array. Data from [6].

Chapter 2: Imager Design and Implementation

2.1 Solenoid Pulsing Circuit

2.1.1 Pulsing Circuit Design

To establish a magnetic field within the solenoid boundaries, it must be pulsed with high current throughout the duration of the data collection period as described by Ampere's law. Since typical laboratory equipment is current limited and cannot support switching at the required speed, it is necessary to design a pulsing circuit to drive current through the coil as shown in Figure 5. The initial targeted design parameters were selected to achieve a 12.5 A pulse through the solenoid for 5 ms. The choice of single pulses rather than a repeated or DC current design was made to avoid damaging the device due to heat buildup on the coil.

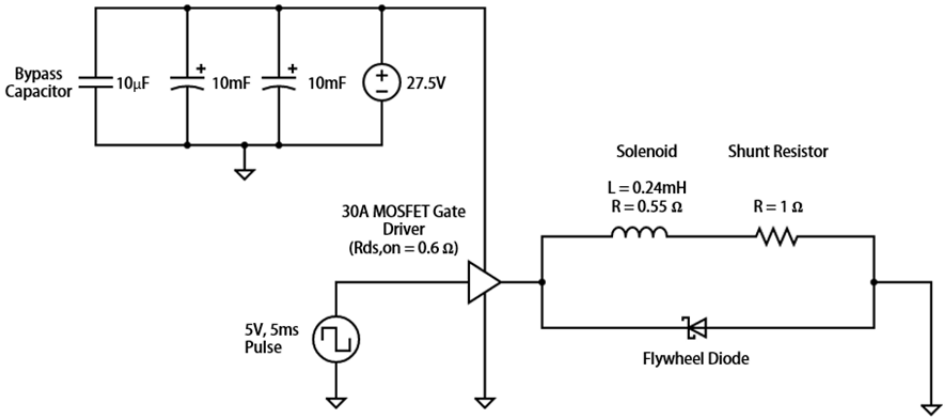

Figure 5: High current pulsing circuit design.

The pulsing circuit is based on a power MOSFET gate driver, which switches to allow the current to flow from a capacitor bank through the solenoid to ground. The chosen gate driver is rated to withstand 30 A and supports a 20 ns switching time [7]. If the power supply maintains a voltage greater than the under-voltage lockout (UVLO) of 12.5 Volts and the input signal is greater than 0.8 Volts, the driver allows current to flow while the trigger signal is high. The design also considers the on-channel resistance, R_{DS}, of the driver while it is in conducting mode, as it will have an impact on the maximum attainable pulse current.

The gate driver is loaded directly with the solenoid, which is in series with a shunt resistor R_s. This resistor was put in place to extend the RC time constant of the circuit, preventing the capacitors from discharging too quickly. The tradeoff of this design choice is that R_s will limit the maximum current that can flow through the load and as a result limit the strength of the induced magnetic field. The solenoid and R_s are connected in parallel with a flyback diode, which prevents sudden voltage spiking across the gate driver's inductive load. This was first designed in LTspice and prototyped before its assembly as a PCB.

2.1.2 Pulsing Circuit Simulation

The circuit was first simulated in LTspice so that the schematic could be adjusted easily as design parameters were tuned. Using a software approach also allowed for a safe design stage, necessary due to the high-current application of the circuit. The circuit model uses two voltage-controlled switches to model the MOSFET gate driver, using the appropriate R_{DS} values from the IXYS datasheet. The circuit was initially simulated by providing a 5-Volt trigger pulse for 5 ms.

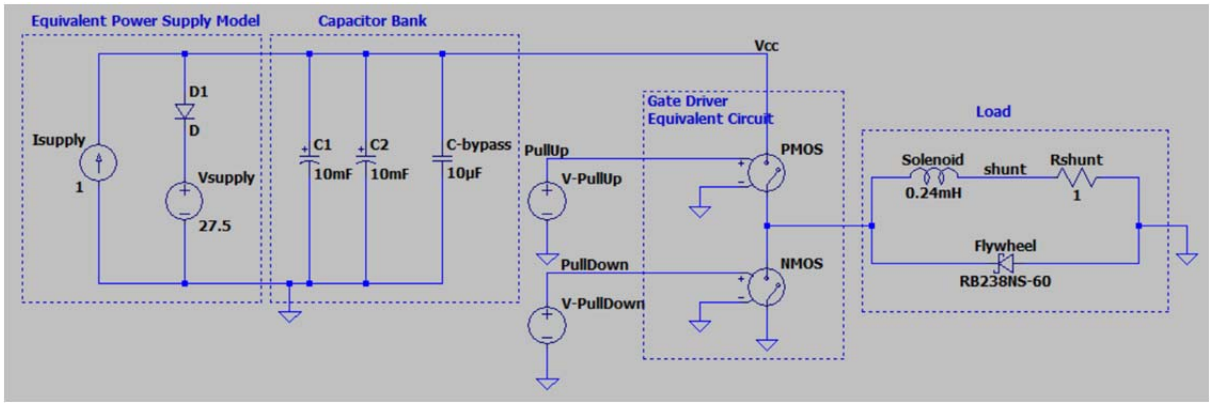

Figure 6: Equivalent LTspice model of the solenoid pulsing circuit.

The simulation was run for 10 ms using the component values shown in Figure 6. The maximum observed current from the solenoid was 13 A, which occurs after roughly 500 µs of rise time. Over the course of the 5 ms switching period the signal drops to about 12 A nearly linearly, making the average pulse current about 12.5 A. The current quickly falls to zero once the input signal is pulled low by the gate driver. The simulation results in Figure 7 confirm that the circuit meets the design requirements, and in testing also showed that driving the circuit with higher supply voltages can yield greater solenoid currents.

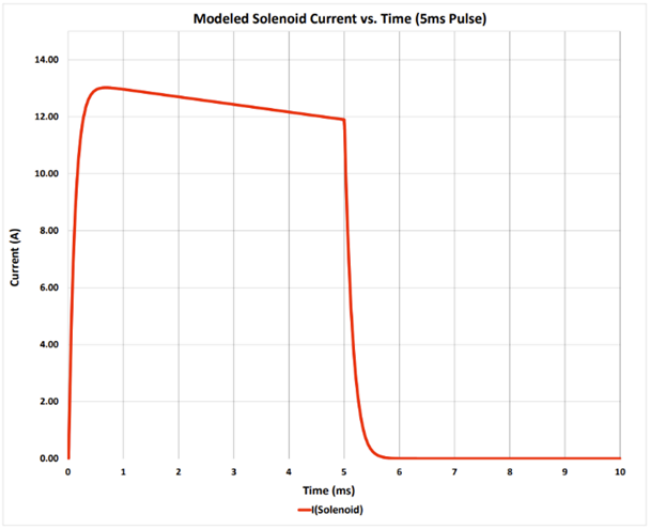

Figure 7: LTspice solenoid current simulation results for 5 ms input trigger.

2.2 Transimpedance Amplifier

The photodiode's hardware interface to the supporting imager hardware was designed as a transimpedance amplifier circuit configuration, which converts a generated photocurrent into a corresponding voltage. The amplifier circuit maintains a linear relationship between current and voltage, which allows for accurate photocurrent detection. This means that incident signal intensity at the interface of the photodiode array module can be directly related to the voltage measured at the Arduino's analog input pins. The circuit works by placing a feedback resistor, R_F, across an operational amplifier, which converts input current to an output voltage by virtue of Ohm's Law. The feedback capacitance, C_F, is included to counteract the effects of unwanted parasitic capacitances that change the amplifier's loop-gain response [8, 9, 10]. By including the op-amp in the design, we leverage its very high input impedance and minimal output impedance to ensure that the output voltage is independent of any load attached to the circuit. This allows for accurate, high-speed photodiode measurements. The device used in this design is a commercial operational amplifier chosen due to its low noise and high precision characteristics, which make it ideal for sensitive instruments [11].

To determine the correct feedback resistor, Ohm's law can be rearranged based on the maximum expected photocurrent. The device photocurrent can be estimated using known source parameters and the responsivity of the photodiode array for a given signal intensity using the following equation, where I_{Ph} is output photocurrent, P is incident signal power, and R is the responsivity of the device:

$$I_{Ph} = P \times R \qquad \text{(eq. 2)}$$

Initial calculations of R_F and C_F as they are shown in Figure 8 are based on expected photocurrent from a 5-mW handheld red laser pointer source with an approximate wavelength λ = 635 nm. Using the above equation and taking R as the visible light responsivity of the

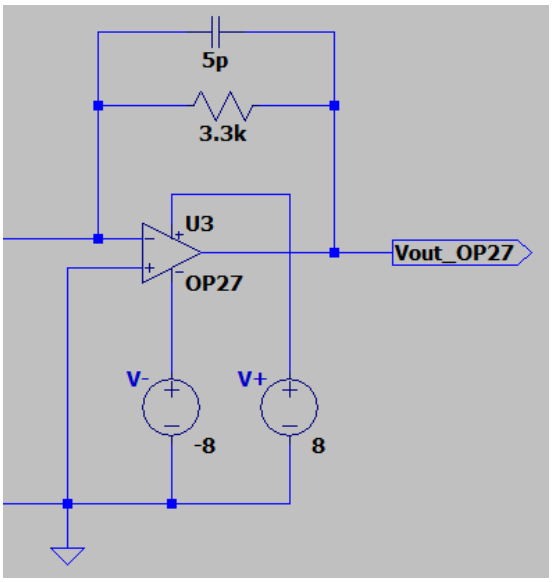

Figure 8: LTspice transimpedance amplifier schematic.

photodiode array at this wavelength given in Figure 2, the maximum photocurrent due to visible light was calculated to be approximately 1.7 mA. This maximum current was used as a placeholder value until measurements of visible light and electron-induced photocurrents could be obtained from the photodiode array. Given that the acceptable voltage range of Arduino's analog pins and ADC module is between 0 and 5 Volts, the required feedback resistance can then be calculated as the following, rounding values to nearby standardized component values [8, 9, 10]:

$$R_F = \frac{V_{oMax} - V_{oMin}}{I_{p-Max}} = \frac{5\ V - 0\ V}{1.7 mA} = 2941.17\ \Omega \rightarrow 3.3\ k\Omega \qquad \text{(eq. 3)}$$

Similarly, the necessary feedback capacitance given the desired circuit amplifier bandwidth can be calculated using the following equation. R_F is the previously computed feedback resistance value, and BW is the desired circuit bandwidth:

$$C_F = \frac{1}{2\pi * R_F * BW} \qquad \text{(eq. 4)}$$

Since the desired sampling bandwidth of the circuit at the time of designing this amplifier was unknown, a placeholder value of 5 pF was used which corresponds to a frequency of 1 MHz. The actual circuit bandwidth was later determined by the chosen Nyquist sampling rate of 10 kHz in the Arduino sampling firmware, so C_F was adjusted to the standard value 10 nF:

$$C_F = \frac{1}{2\pi * 3.3\ \text{k}\Omega * 5\ \text{kHz}} = 9.65\ \text{nF} \qquad \text{(eq. 5)}$$

Output voltage clipping issues are avoided by powering the integrated circuit with ±8 V, where ideally the circuit would only demand ±5 V. The photodiode equivalent circuit shown in Figure 9 was used to accurately model the physical device parameters that may affect photocurrent generation. This model includes the shunt resistant, R_s, and depletion region capacitance, C_D, present at the photodiode P-N junction [12]. By connecting the photodiode model to the inverting input of the op-amp and performing a DC sweep on the photocurrent source, the simulation data in Figure 10 was obtained.

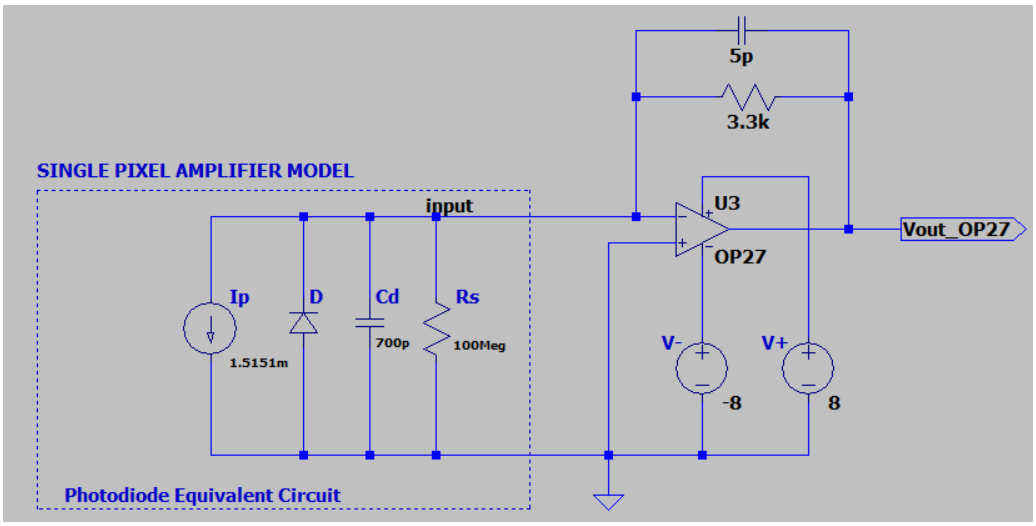

Figure 9: LTspice photodiode model connected to the inverting terminal of the operational amplifier.

Figure 10: Transimpedance amplifier output voltage for DC current sweep of 0-1.5151 mA.

From this simulation, it can be observed that there is a linear relationship between input photocurrent and output voltage over the target current range of 0-1.515 mA, due to the resistor value adjustment to a standardized value. Since the Arduino Mega analog pins operate for an input range of 0-5 V, any excess potential applied to the analog pins will be clipped down to 5 V and will reduce the dynamic range of measurement. The microcontroller uses a 10-bit analog-to-

digital converter, meaning that voltages greater than or equal to 5 V will be digitally represented with a value of 1023. Consequently, it is important to ensure that the circuit's R_F value is tuned accurately to the expected current range and the circuit produces 5 V only when the sensor is fully saturated. Due to the modular nature of the amplifier circuit, the output voltage can be easily adjusted based on the measured current range of a given source. Given that the expected current range of each source will differ, preliminary current measurements must be taken from the photodiode array under saturation conditions prior to data collection.

Breadboard testing of this circuit was accomplished by using a voltage-controlled current source to simulate the photocurrent being generated by the PD array, and results very closely matched what was observed in simulation. However, since the op-amps do not behave ideally, the expected ±5 V had to be increased to ±8 V to ensure that voltages near the ADC input ceiling were not clipped prematurely. The circuit was fully assembled as shown in Figure 11 with all sixteen pixels such that it could be tested on a breadboard in response to incident light from a handheld flashlight and laser pointer.

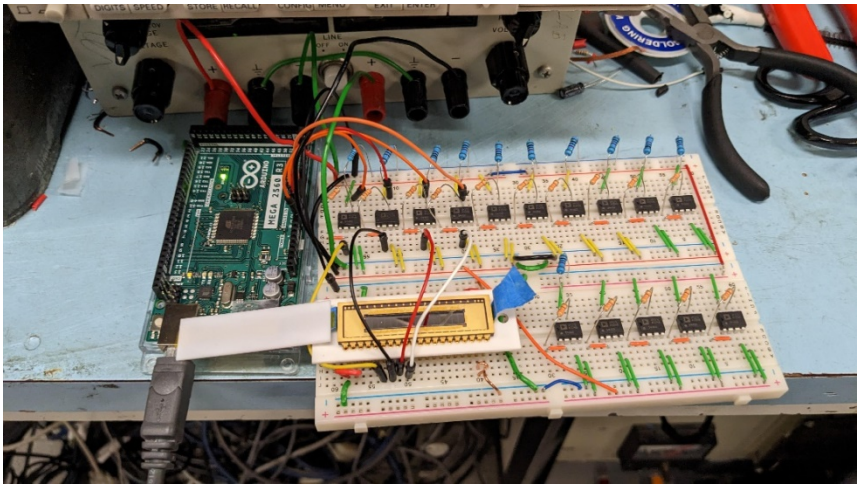

Figure 11: Transimpedance amplifier array connected to the linear photodiode array and Arduino Mega.

2.3 Photodiode Driver PCB

After prototyping the amplifier and verifying its functionality on a breadboard, a PCB was designed such that the photodiode array and microcontroller may be directly connected through the amplifier circuitry. The primary goal of this was to increase reliability and portability for data collection and reduce any errant noise that may decrease ADC accuracy. The PCB design is also much more organized than a breadboard configuration, making the circuit much easier to debug while under test. The PCB was created with Altium Designer due to ease of access and its high prototyping speed.

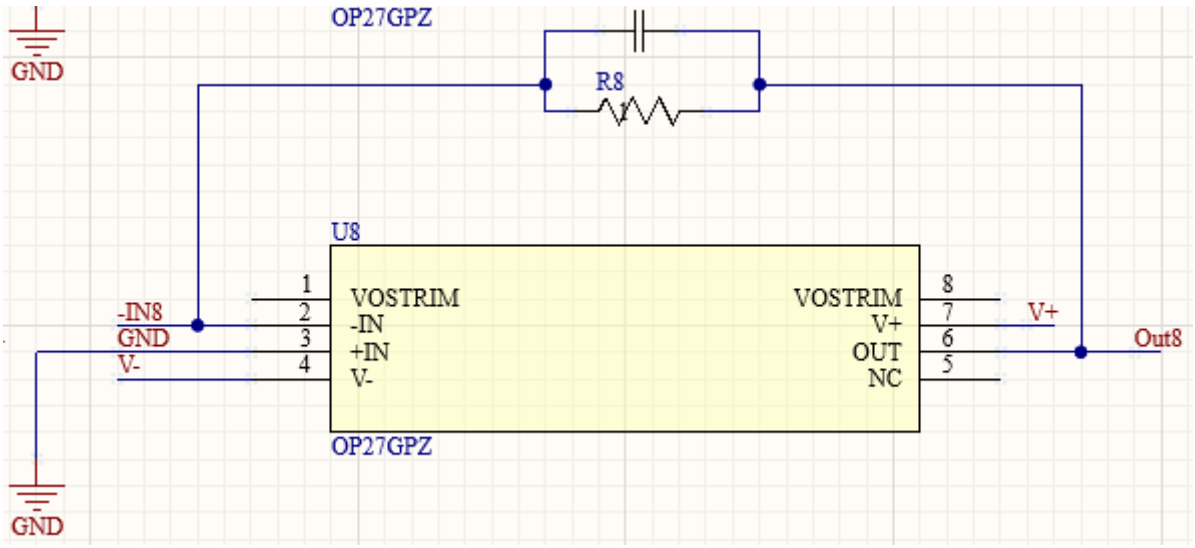

Figure 12: Altium Designer schematic for the transimpedance amplifier.

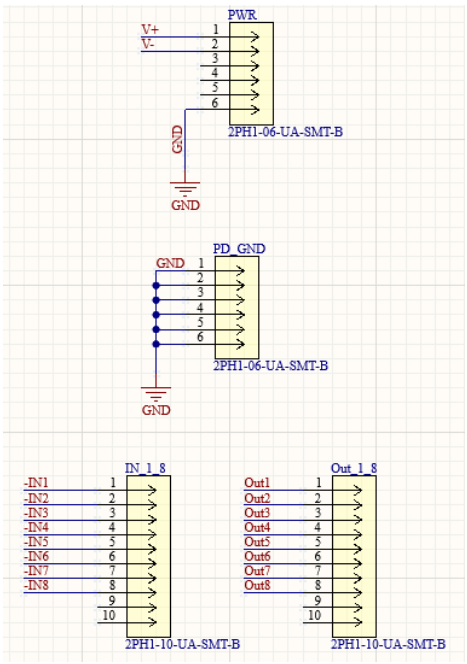

Figure 13: PCB IO header pins, including extra ground pins for photodiode array.

Since the final PCB will be a linear array of transimpedance amplifiers, individual component blocks were used as shown in Figures 12 and 13. This allowed for easy changes to the design as device parameters were changed. As discussed in Section 2.2, the inverting terminal is connected directly to the output photocurrent from the array, and the noninverting terminal is connected to ground. The input voltage terminals In- and In+ are connected to header pins on the board, and the offset voltage trim terminals are left without connection since the typical offset voltage range is in the microvolt range and will not affect sensor readings appreciably [11]. An additional set of header pins were added to the schematic for grounding only, to make it easy for the PD array to share a common on-board ground with the amplifier circuit.

The PCB was designed using a compact footprint of only eight transimpedance amplifiers per board, meaning that two boards should be used to accommodate all sixteen pixels on the PD

array. This decision was made to ensure that the layout of the board was neat and organized, and to allow for a more modular testing approach in which multiple interchangeable boards with different amplifier parameters may be used (see Figure 14). For example, having multiple boards can accommodate the variable input photocurrent ranges generated by visible light and direct electron imaging.

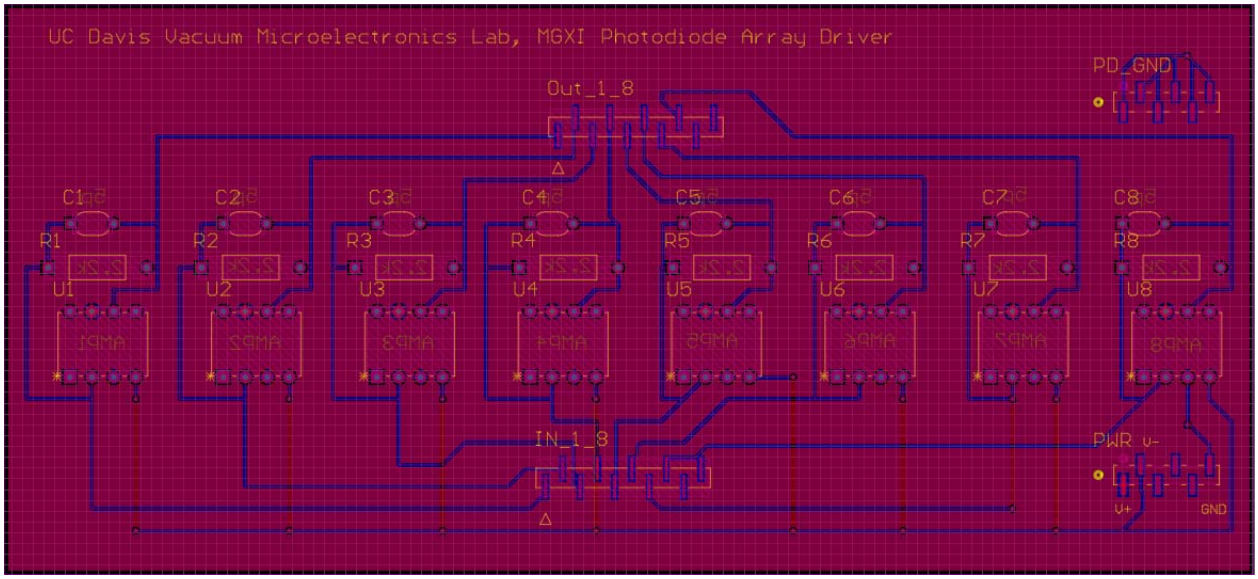

Figure 14: Altium Designer PCB layout for transimpedance amplifier circuit.

A polygon pour was used to create routing for the V+ and ground pins of each op-amp to create a more organized board layout. The headers for power delivery and I/O signals were chosen due to what components were readily available, so there are some input and output pins left with no connection. The final 3D render of the transimpedance amplifier array is shown in Figure 15 with the final assembled board shown in Figure 16. The PCB was fabricated using JLCPCB's prototyping service and assembled using the initial circuit parameters of: R_F = 3.3 kΩ and C = 5 pF.

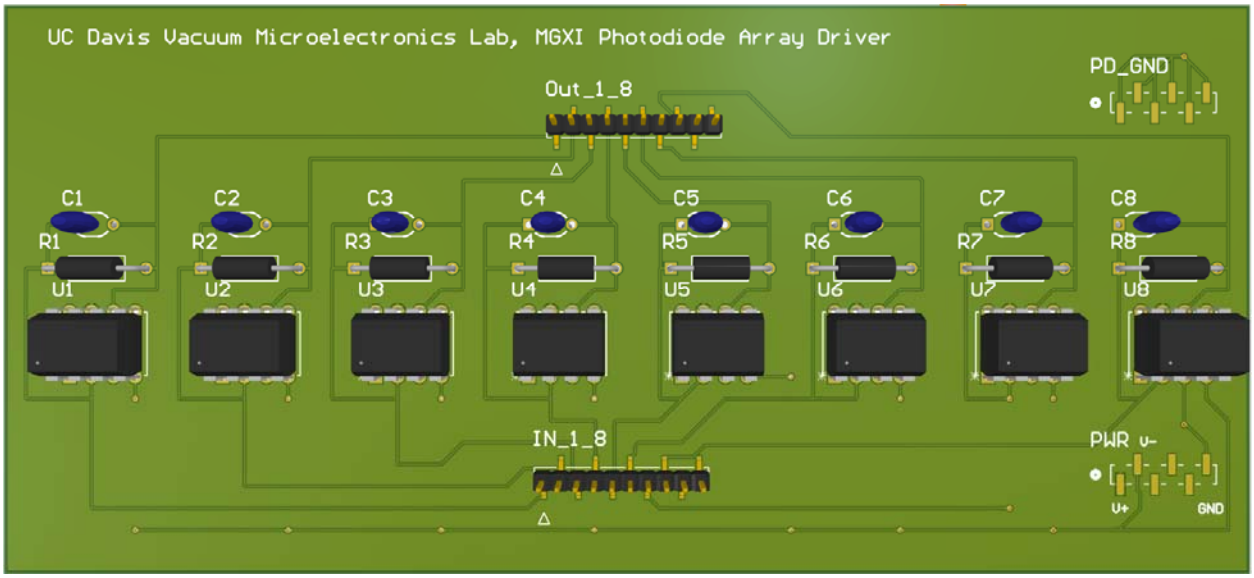

Figure 15: Final Altium Designer 3D render of transimpedance amplifier circuit.

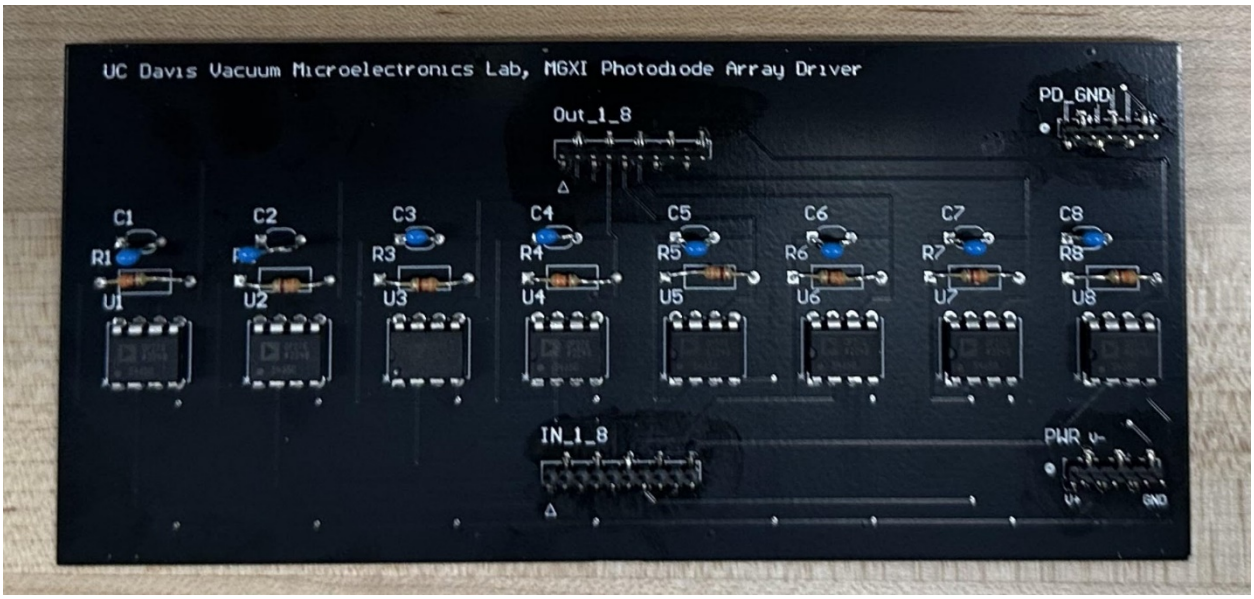

Figure 16: Fabricated and assembled PCB from JLCPCB.

2.4 Microcontroller Interface

2.4.1 Live-plotting firmware

Two different firmware scripts were used for the imager, each for a different software application. The first firmware code is listed as Appendix A-2 and supports the live-updating Python application listed as Appendix A-4. The firmware takes a naive approach to monitoring pixel brightness by reading the Arduino's analog pins all at once and directly communicating them to the serial interface. While the ADC can be sampled effectively at 10 kHz as in Appendix A-1, this frequency is not necessary for live plotting applications. This allows data samples to simply be read from analog pins and written to serial all in one step, eliminating the need for timing code. As a result, the speed of the firmware is largely limited by the serial communication and proved to be adequate for this application and yields a frequency of approximately 5 Hz depending on the computer hardware used. A 175-ms timing delay was added between read/write cycles to increase the stability of the firmware and ensure that corrupted data was not received by the plotting application. After it is received by the Python script, the plot is re-drawn with the newly received data and the process is repeated until interrupted by the user. Given additional development time, this live-updating component could be adjusted to use a dedicated high-speed program and circuit components to effectively eliminate the limitations imposed on the sampling frequency.

2.4.2 Data collection firmware

The primary firmware code used for data collection is listed in Appendix A-1, which supports the data collection Python script listed in Appendix A-3. This firmware takes a sample

from each pixel at 10 kHz over the course of 10 ms, collecting a total of 100 samples per shot. These data points are stored in a 16 x 100 two-dimensional array stored in the Arduino's on-board memory. Each shot is triggered by an interrupt supplied to a digital pin on the microcontroller. This trigger is sent before the solenoid is activated, so that the formation and collapse of the magnetic field can be seen throughout the exposure time (see Figure 17). If necessary, this 10-millisecond period of data collection can be delayed by introducing a software delay according to solenoid pulser rise time, as the inductance contributed by the solenoid may slow down rise time considerably. After collecting 100 data points, each sample value is communicated to the serial interface so that it can be received by the Python data collection software. After sending the data, the firmware resets itself and waits for another trigger. When the data is received by the Python processing script A-3, it is averaged and drawn to the user display and is stored in an Excel spreadsheet for further analysis. The firmware will then continue to facilitate data collection based on the trigger signal until interrupted by the user.

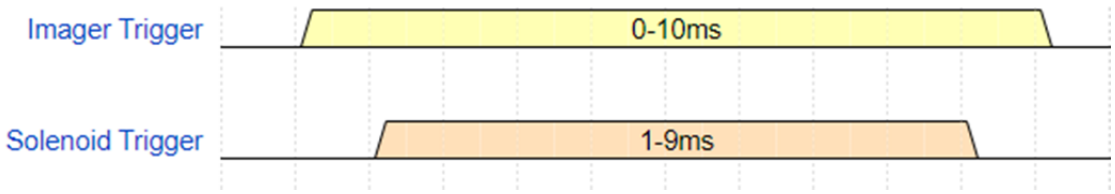

Figure 17: An ideal timing diagram depicting the relative length of each control signal used in data collection.

2.5 Data Collection and Processing Software

There are three primary software components of the imager, each having served a different purpose throughout the design cycle of the imager. The first software component to be designed was the live-plotting software, which was used in part to help verify the functionality of

the photodiode and transimpedance amplifier. It may also be used to help in the setup of the electron source by allowing observation of the target during electron beam configuration. The second software component is the data collection and storage script, which was utilized to read data from serial after being collected at the Arduino. This script is also used to visualize data as it is processed and stored. Finally, the data visualization script was written to display data after it has already been collected.

2.5.1 Photodiode Array Live-Plotting

The live plotting software functions by communicating with the Arduino over the serial interface. This software was designed to visualize the signal incident to the photodiode array at any given time to test the transimpedance amplifier array and configure the electron source. Using Matplotlib [13], a canvas is drawn to model the surface of the photodiode array where each cell is mapped to a corresponding pixel on the linear array as seen in Figure 18. Each sample read from serial is first processed such that it may be plotted and is then used as new data

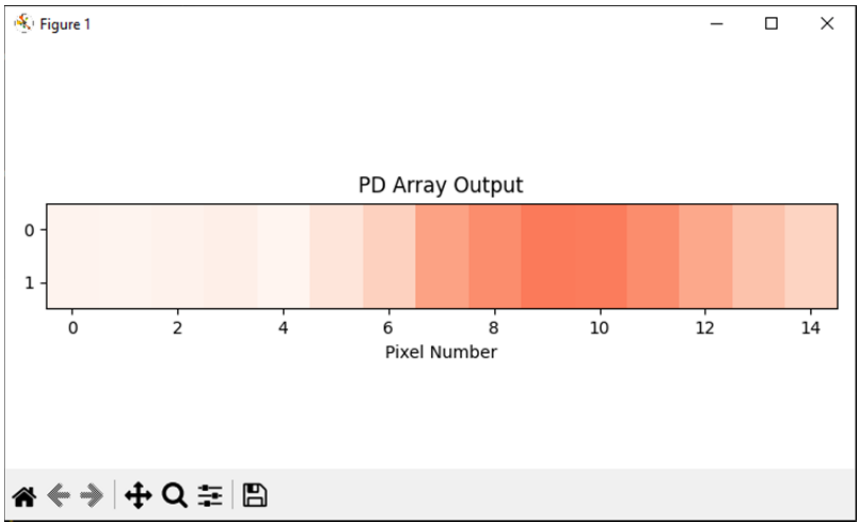

Figure 18: A screenshot of the live-plotting application during operation, with visible light incident to the photodiode array.

to populate the light intensity map of the sensor interface. After the plot is updated, the data sample is written to an Excel spreadsheet for further analysis, and the next set of data is read from serial. This process runs and stores data until interrupted by the user.

2.5.2 Data Collection and Storage

The primary data collection script is listed as Appendix A-3. The function of this script is like the live plotting script, except it runs based on the firmware listed as Appendix A-1. A single-shot data capture method was used in this code to facilitate high frequency measurement, allowing fine-grained temporal resolution over the entire collection period. The data is captured for 10 ms at 10 kHz by the firmware, and then communicated through the serial interface so that it can be processed in detail. Upon receiving the pixel measurements, they are processed in a similar manner to the live-plotting application. However instead of updating a preexisting canvas, the data is averaged over the entire 10ms data collection period and displayed accordingly to reflect the average signal incident to the sensor over the collection period.

After the collected data is visualized, each new data point from serial is stored into an Excel spreadsheet and timestamped. This is done after reading each individual data point from the serial buffer such that the interface is ready to be used again when the user clears the visualization. Ultimately the purpose of signal intensity storage is to keep an archive of different testing configurations and the data they yield, but also to allow the data to be analyzed more thoroughly after the shot has taken place.

2.5.3 Data Visualization

The final software component of the imaging system is the data visualization code, which was designed to display data that has been previously captured and stored as discussed in section 2.5.2. This program takes two primary arguments used to determine which data file should be read from, and the approximate sample time of the desired data (e.g. 0-10 ms). An optional third argument is included, which enables an animation of the data over the course of the entire data collection period. This functionality was included so that it would be easier to visualize and understand imaging events after they have occurred, such as the effect of a collapsing magnetic field on intensity of signal incident to the photodiode array. This can also be used to adjust the timing of the system to better focus on the region of interest. If the animation function is enabled, the image is updated every 100 ms. In this way the animation happens over 10 seconds, relative to the 10 ms data collection period. An example of data visualization is presented in Figure 19.

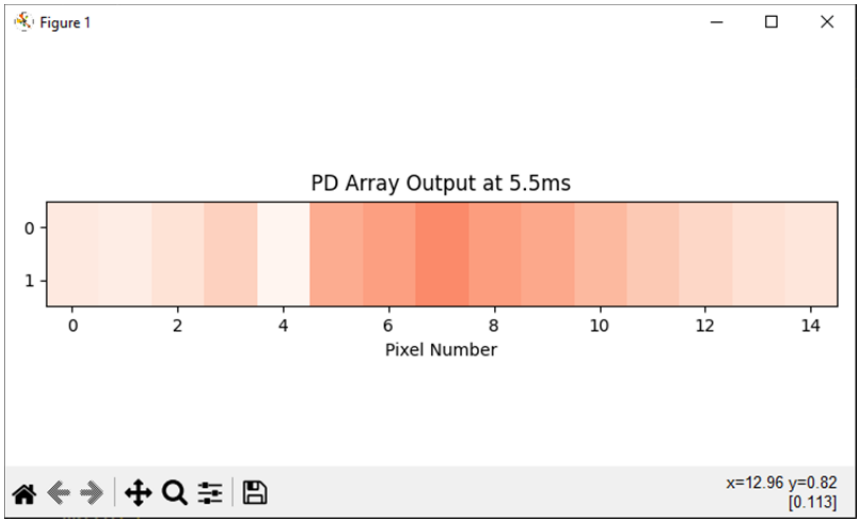

Figure 19: Visualization in response to visible light, displaying data captured at 5.5 ms. Illustrates that pixel 4 is inoperable.

Chapter 3: Experimentation and Results

3.1 Experimental Setup

To experimentally validate the design of the imager, the system was initially constructed with the photodiode array placed outside of vacuum to collect data at atmosphere and characterize its response to visible light. Since two PCBs are required to image all sixteen pixels on the array, the system ties the power and ground signals of the boards together. Because the imaging hardware is very sensitive, the boards' electrical connections were suspended above the workbench using a 3D printed spacer and placed on a ceramic floor tile. The input wires were constructed such that they may either be directly connected to the photodiode for data collection at atmosphere or connected to the vacuum feedthrough for direct electron testing. The microcontroller was connected directly to the output node of each transimpedance amplifier such that it could directly sense amplifier output voltages. The trigger signal was attached to a digital input pin. The full imager configuration is shown in Figures 20 and 22.

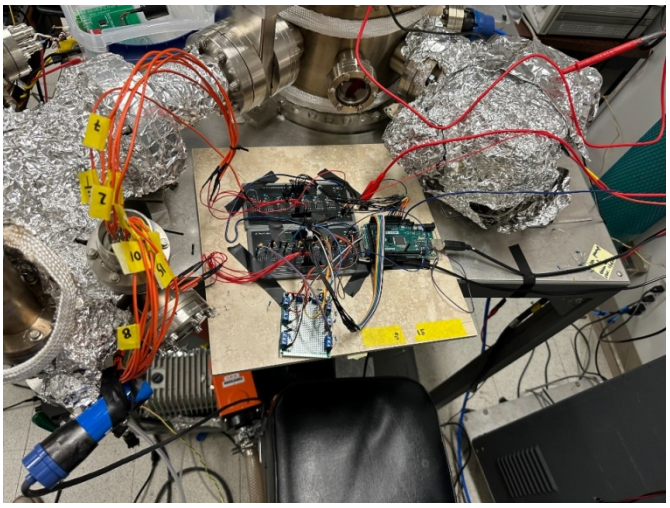

Figure 20: The experimental setup of the photodiode imager.

The trigger was also connected to the input pin of the solenoid pulser board to simulate real imaging conditions while examining the visible light response of the device. The photodiode itself was mounted to a polyether ether ketone (PEEK) ring and placed on the vacuum test stage, so that it can be aligned with the electron drift region. The PEEK ring covers pixels 3-5, so the signal test area is constrained to pixels 6-15 (see Figure 21).

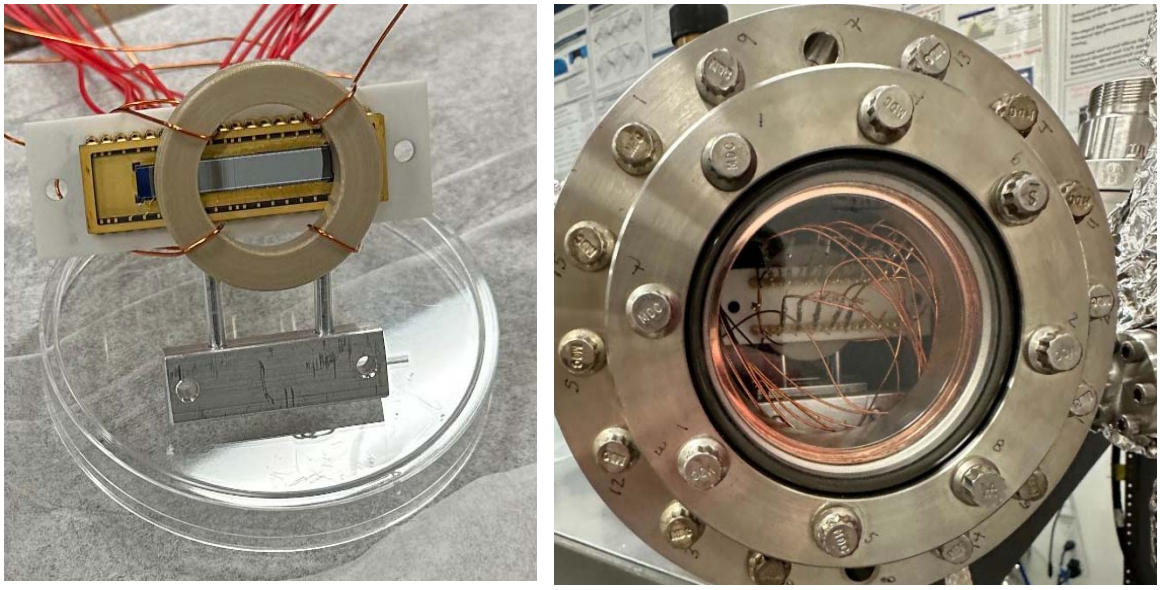

Figure 21: Photodiode imager mounted on a PEEK ring (left) and positioned within the vacuum test chamber (right).

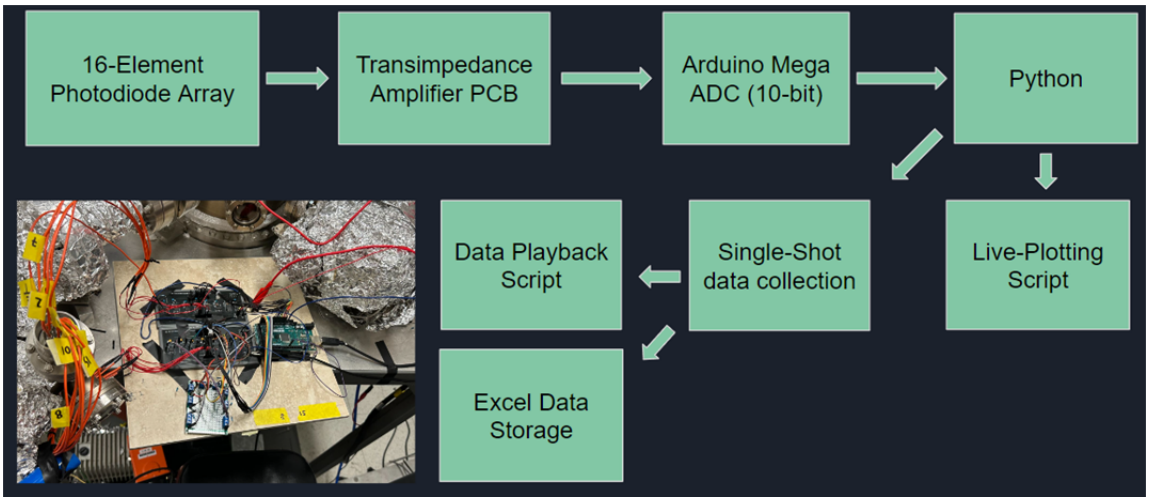

Figure 22: Final experimental hardware setup for visible light and vacuum data collection.

3.2 Experimental Hardware

Several changes were made to the experimental setup in preparation for visible light data collection, which departed from some of the initial parameters. These changes were made in part to streamline the data collection process, but also to ensure that the data collected was as accurate as possible. These changes also facilitate future work, as they allow the device to be more easily modified in preparation for further testing.

3.2.1 Removal of Amplifier Filter Capacitors

One of the more significant changes was the removal of the filter capacitors attached to the output node of the transimpedance amplifiers. Feedback capacitors are typically included in the design of a transimpedance amplifier to increase the stability of the circuit in high-frequency applications where oscillations may be induced while also controlling the frequency response of the amplifier. This helps to ensure that the gain response of the amplifier is consistent and reduces overall noise in the system. However, the choice of feedback capacitor depends directly on the resistor that is used in the circuit as described in Section 2.2. Since the imager must be sensitive to light in the visible spectrum and direct electrons, the resistor values must also be able to be changed. This is primarily because the photodiode array's responsivity to each source differs significantly. As a result, when the resistors are changed the required filter capacitor value must also change to avoid impacting the circuit behavior and prevent it from possibly filtering out the desired signal.

The inclusion of a filter capacitor may also inadvertently change the way the Arduino's ADC works and make it less reliable. This is because the Arduino Mega uses a multiplexed

analog-to-digital converter such as the one shown in Figure 23, where the signal from each analog input is temporarily stored in a capacitor as the single ADC driver is swept over each of the inputs. Since voltage differentials between adjacent inputs (pixels) may contribute to inter-channel crosstalk, the capacitors must be able to discharge as intended [14]. This solution eliminates the need for multiple ADCs, with the tradeoff that large resistance and capacitance values at the input of each analog pin may influence the converter's ability to accurately sense voltages at high operating frequencies. Since the frequency of data collection for the photodiode is 10 kHz, the feedback capacitance was ultimately removed from the TIA circuit to ensure that the performance of the ADC was not compromised and minimize any chance of incident signals being filtered out by the amplifier.

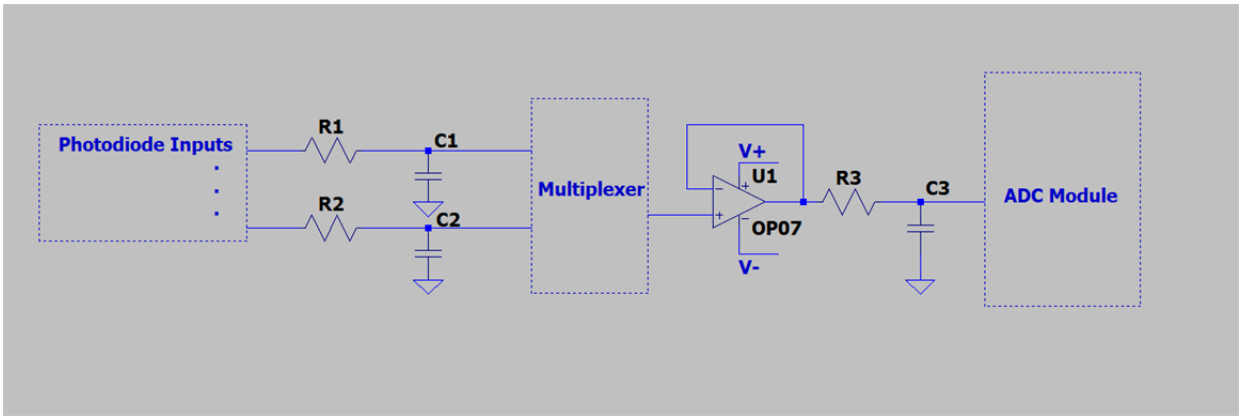

Figure 23: Circuit diagram for a multiplexed analog-to-digital converter.

3.2.2 Resistor Hot-Swap Board

In testing it became clear that to properly tune the boards for different intensities of incident signal it would be necessary to have tunable resistances at each of the transimpedance amplifiers. This ensures that the same set of boards may be used for different testing configurations and different sources. Using potentiometers is an attractive choice, as they allow for easily tunable resistance and consequently, tunable sensitivity. When the tunable boards were assembled as shown in Figure 24, it was found that while they did allow for direct sensitivity adjustments, they also introduced a large amount of variable thermal noise and the imager became unstable.

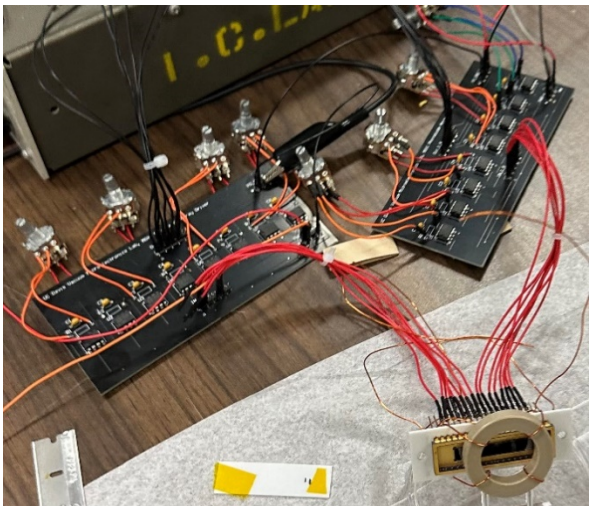

Figure 24: The photodiode imager connected to the amplifier boards, configured with potentiometers in place of resistors.

To increase the signal-to-noise ratio, the potentiometers were replaced by a hot-swap resistor board shown in Figure 25, which was soldered to the parent PCB for pixels 8-15. This significantly improved the signal quality and overall performance of the system.

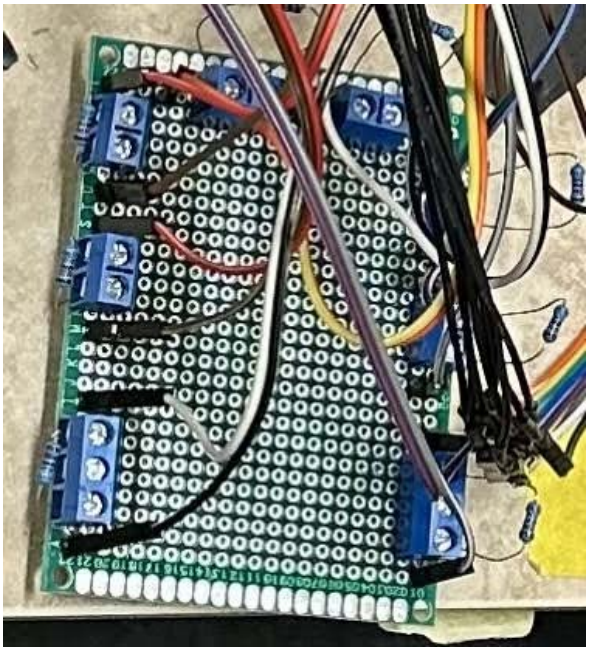

Figure 25: The resistor hot-swap daughter board, which is connected to pixels 8-15 of the transimpedance amplifier array.

3.2.3 Tuning the Analog-to-Digital Converter Reference Voltage

In some cases, the amplifier subsystem may require more precise tuning than can be obtained by changing resistor values alone. For example, the responsivity of the photodiode array to direct electrons is significantly lower than to visible light, and the system must use very large resistors to be sufficiently sensitive. This issue may be resolved by utilizing a sensitivity parameter built into the microcontroller's ADC system. The ADC uses an internal reference potential by default to approximate the amplifier output voltage but can be programmed to instead use an external potential on the range of 1-5 Volts for comparison. By setting the reference voltage to a smaller value with an external power supply, the ADC's signal sensitivity can be increased while maintaining the granularity of its digital number output.

3.3 Sensor Quantum Efficiency Approximation

The quantum efficiency (QE) of a sensor is the ratio of the number of charge carriers generated by a photodetector to the amount of incident particles and is used to quantitatively evaluate a detector's effectiveness in converting incident radiation into electrical signal. To accurately estimate the quantum efficiency of a device, a well-known source must be used. The amount of photocurrent generated experimentally by this source can then be measured and compared to the theoretical maximum.

To experimentally measure the quantum efficiency of the photodiode array used in the imager, a 5-mW, 532-nm wavelength laser was used. The sensor was first experimentally verified by comparing the theoretical maximum photocurrent to the measured value. The laser was mounted on a test station and positioned such that it was centered on a single pixel as shown in Figure 26. Given that the diameter of the laser aperture is 4.8 mm, the spot size of the signal is significantly larger than the surface area of a single pixel (see Figure 27). To adjust for this, the area of all three pixels with incident signal on them and laser spot size were considered in the

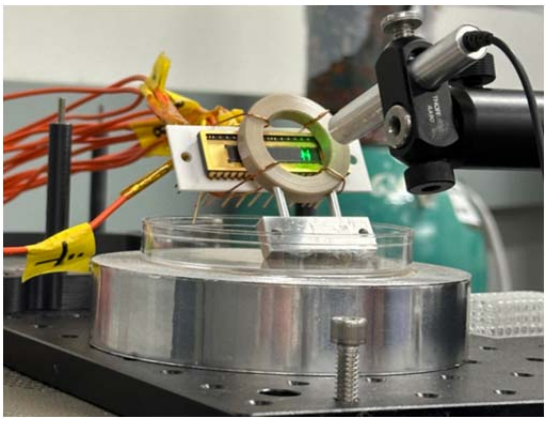

Figure 26: The experimental QE measurement setup.

preliminary calculations.

The total beam area was then used as an input parameter to the photocurrent approximation script, which can be found in code listing A-6. This script generates the photocurrent response to the laser, given its power, the cross-sectional area of irradiation, and the responsivity of the detector. Utilizing the entire spot size of the laser eliminates the need to account for the power distribution of the laser throughout its cross-sectional area. Using this method, the theoretical maximum photocurrent induced by this source is 1.3475 mA, which is divided across all three of the irradiated pixels.

One hundred samples were then collected from each of the relevant pixels and averaged to give the mean signal intensity at each pixel over 10 ms. The average voltage at each amplifier was then converted into photocurrent via Ohm's Law and summed to reconstruct the total induced photocurrent at the sensor. These values were then confirmed by placing a sensitive handheld multimeter between the pixels and ground. The total induced photocurrent across all three pixels was 1.145 mA. By calculating the ratio between this and the expected theoretical

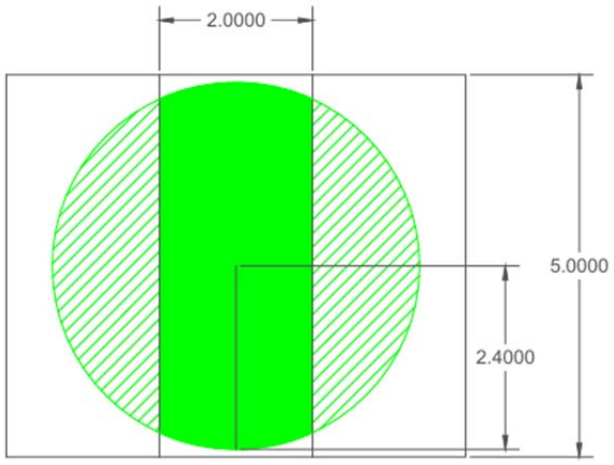

Figure 27: A CAD drawing used to calculate the area (mm^2) of overlap between laser spot size and photodiode pixel.

maximum, the produced photocurrent is approximately 85% of the expected value. This value is appropriate due to the inherent measurement error of the laboratory setting due to factors such as laser alignment and laser power fluctuations.

To confirm this the ratio was calculated once more, instead using the photocurrent generated by only a single pixel. First, the percentage of the spot size constrained within the center pixel was calculated as shown in Figure 27. The cross-sectional beam area was then substituted with this partial spot size in the photocurrent approximation code. To account for the non-uniform distribution of power across pixels, the averaged data was used to calculate the percentage the photocurrent concentrated in the center pixel relative to the total. The power incident to the center pixel was then calculated as:

$$P_{pixel} = \frac{(P_{total} * \alpha)}{A_{Overlap}} \qquad \text{(eq. 6)},$$

where α represents the current density concentrated within the pixel of interest. Using this new power value, the theoretical maximum photocurrent is 1.127 mA. The average measured current generated by one pixel over 100 samples was 0.957 mA, so the approximate ratio of produced current to expected value is again 0.849 or 85% at this optical wavelength.

The quantum efficiency was calculated after verifying that the ratio of the experimentally measured photocurrent to the theoretically maximum photocurrent fell within an acceptable margin of error, ensuring the reliability of the results. The quantum efficiency was then calculated using the same data to maintain consistency. Given that the QE is defined as the ratio

of generated electrons to the number of incident photons, equation 7 may be used to complete this analysis where $N_{electrons}$ and $N_{photons}$ are the number of electrons and photons, respectively.

$$QE = \frac{N_{electrons}}{N_{photons}} \qquad \text{(eq. 7)}$$

The number electrons were computed simply as the measured photocurrent divided by the charge of a single electron as shown in equation 8 where I_{ph} is the measured photocurrent and q is the charge of an electron. Similarly, the number of incident photons were calculated using the total power of the laser and the energy of a single photon at the laser's nominal wavelength, which is shown in equation 9 where P_{laser} is the power of the laser, lambda is the wavelength, h is Planck's constant, and C is the speed of light.

$$N_{electrons} = \frac{I_{ph}}{q} \qquad \text{(eq. 8)}$$

$$N_{photons} = \frac{P_{laser}}{E_{ph}} = \frac{(P_{laser} * \lambda)}{h * c} \qquad \text{(eq. 9)}$$

Using the experimentally obtained photocurrent, the quantum efficiency was calculated to be approximately 0.534 or 53.4% when using the current collected across all three pixels. While some silicon photodiodes may have quantum efficiencies up to 90%, it is typical for such devices to exhibit between 40% and 80% efficiency [15]. Given that the QE measurement falls within the typical range of values under laboratory measurement constraints, this estimation of quantum efficiency is considered acceptable for this type of device.

3.4 Observed Visible Light Response

3.4.1 Flashlight Intensity Data

A similar testing setup was used to test the imager's response to polychromatic visible light, using a mobile phone flashlight instead of the green laser source (see Figure 28). The

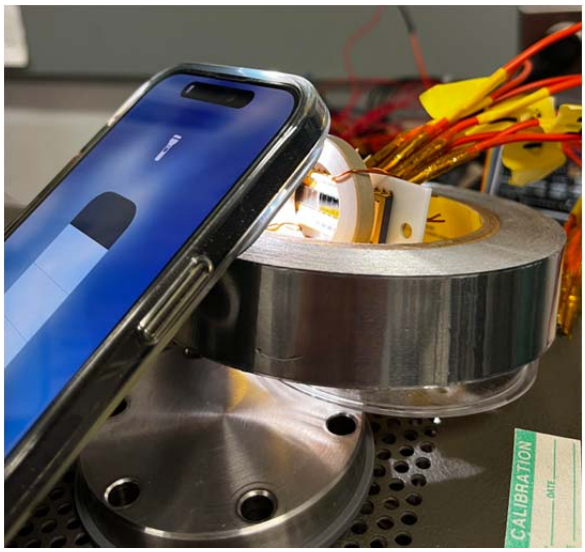

Figure 28: The polychromatic light testing setup, using different light intensities from a phone flashlight.

phone flashlight has four intensity values, and the device was spaced at a fixed distance away from the imager to ensure consistency. Only pixels 8-15 were connected to the imager hardware for these tests, as to simulate the operating conditions of the system under vacuum. Two shots per light intensity level were collected with the single-shot script, and values for each pixel were averaged over all 100 samples before being averaged among different shots.

The resulting data was processed and plotted with Excel, which showed that each shot exhibited an exponential-like intensity distribution across pixels as shown in Figure 29. The

signal intensity in each plot is normalized to a maximum of 1, with a peak indicating amplifier saturation or an out-of-range signal.

Stacking these plots for each incident light intensity showed that this response is consistent across each light intensity setting. From Figure 30 it is evident that the light intensity steps of the mobile phone flashlight increase in a mostly linear fashion. As expected, the exponential rate of decay is also a function of light intensity, with the peak brightness resulting in the most extreme decay.

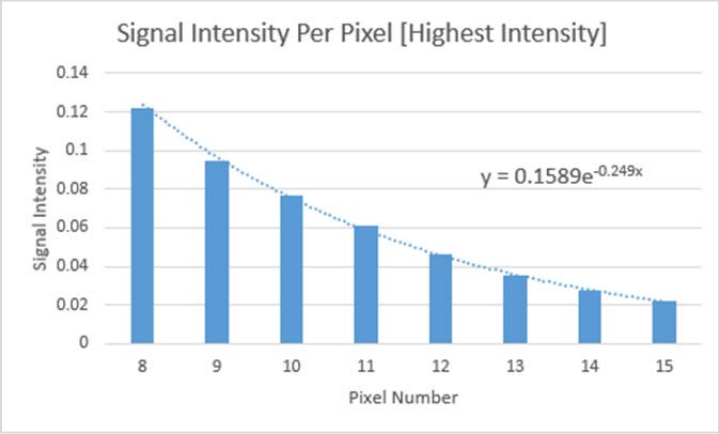

Figure 29: Signal intensity across pixels, showing exponential decay.

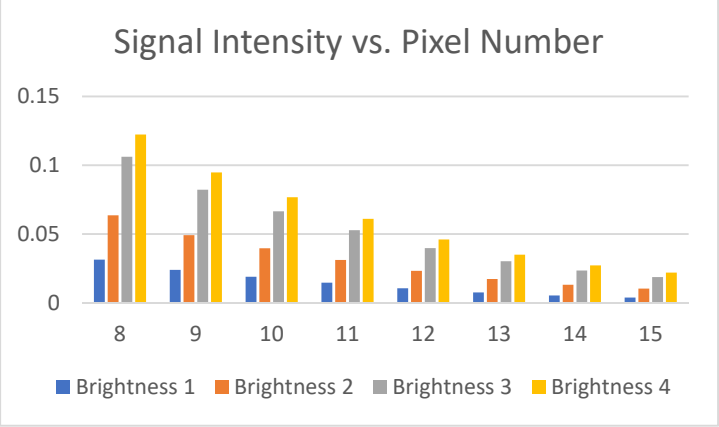

Figure 30: Signal intensity data for each experimental brightness level.

3.4.2 Rectangular Laser Pointer Data

A similar methodology was used to test the imager's response to visible light in the 600-650 nm range, using a generic handheld red laser pointer. The spot size of the laser pointer was measured to be approximately 3 mm x 5 mm, so it was tested both centered vertically on one pixel and horizontally across a group of three pixels. The horizontal beam is shown in Figure 31.

Figure 31: A dark image of the red laser pointer spot size.

Given that the spot size of the laser is about 33% larger than the area of a single pixel, it is expected that the laser has overlap into the neighboring pixels. This was demonstrated in the data, where some signal was seen in the adjacent pixels due to the overlapping area. However, most of the generated photocurrent was still concentrated within the target pixel due to the collimated nature of laser light. This was also observed in the data, which showed that 98.6% of the signal collected was concentrated within the target pixel.

The experimental data confirms that when the laser source is rotated 90 degrees, the signal bleeds symmetrically into the neighboring pixels with the target pixel only containing 59.5% of the incident signal. As expected, the plot in Figure 33 is dimmer than pixel data in Figure 32 due to the distribution of signal across pixels 12, 13 and 14. Taking the brightest frame from the horizontal shot data shows that when the laser pointer is centered on pixel 13, the photocurrent generated in neighboring pixels is relatively uniform.

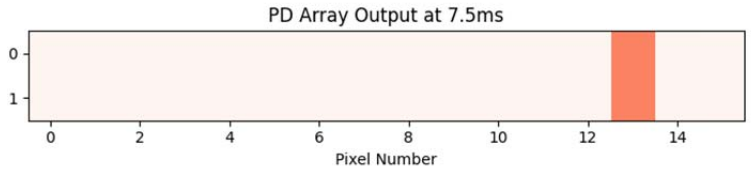

Figure 32: The brightest frame of red laser pointer data centered vertically on pixel 13.

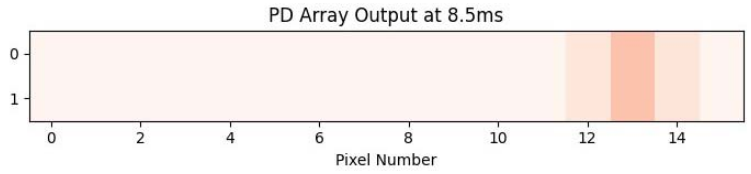

Figure 33: The brightest frame of red laser pointer data in the horizontal orientation.

3.5 Preliminary Direct Electron Testing and Noise Data

Prior to imager experimentation, a phosphor plate was positioned in place of the photodiode array in the chamber to confirm that the electron source was operating as intended and a beam could be formed.

Since the photocathode requires a significant bias for thermionic emission, it is biased negatively, allowing a gradually increasing potential to be applied to the MCP and phosphor plate to guide electrons through the drift region. An additional wire mesh grid was placed between the MCP and the phosphor, which was biased slightly lower than the phosphor to provide another source of electrical attraction to guide the electrons through the drift region. By pulsing the solenoid as discussed in Section 2.1.1, the beam is confined into a focused spot as it fluoresces on the surface of the phosphor plate. As the potential applied to the phosphor plate increased, the fluorescence brightness increased due to more electrons reaching the plate's surface. A particularly bright example is shown in Figure 34, and the focusing hardware is shown in Figures 35 and 36.

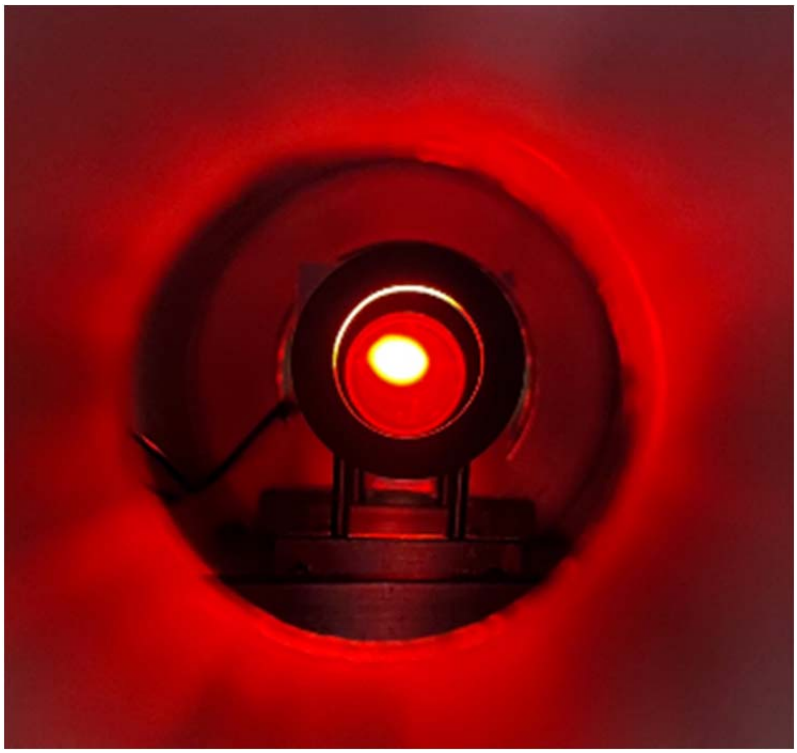

Figure 34: Solenoid-focused electron beam incident to the positively biased phosphor plate.

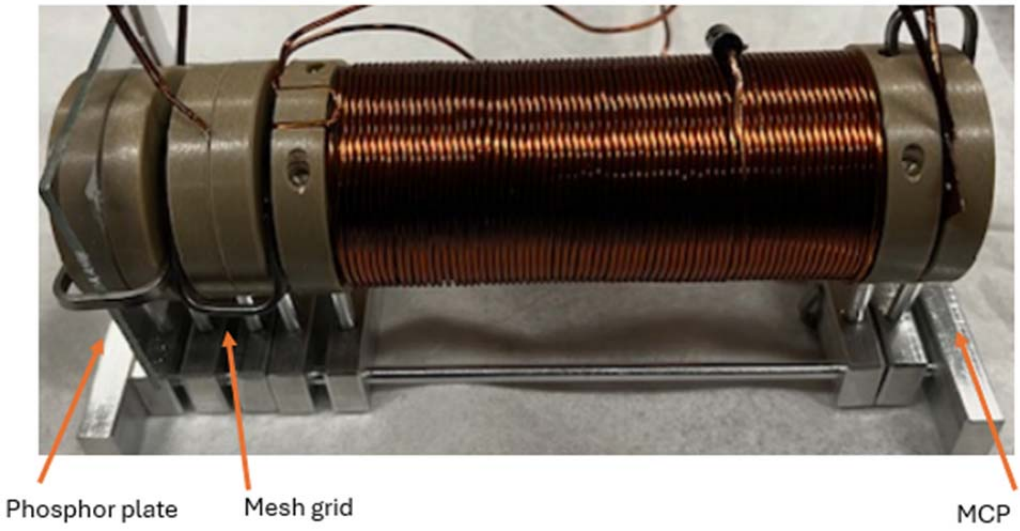

Figure 35: The full solenoid testing setup placed under vacuum.

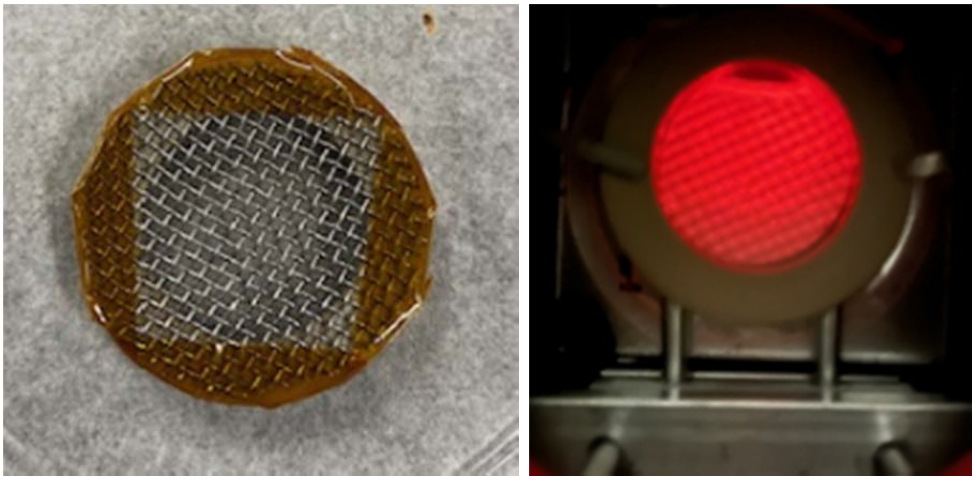

Figure 36: The mesh grid added to the system to pull electrons to the phosphor (left) and the grid pattern visible on the phosphor fluorescence (right).

After validating the electron beam, the photodiode was placed under vacuum to replace the phosphor plate and was electrically connected to the amplifier array via a feedthrough as shown in Figures 21 and 22. The functionality of the imager was then confirmed by flooding the chamber with visible light using a flashlight and confirming that the feedthrough's effect on the

generated photocurrent was negligible. The beam was then re-formed in with the same procedure as to maintain consistent results. However, for the imager to function properly, the photodiode must remain grounded relative to the transimpedance amplifiers' power supply. To simulate this, additional biases were tested on the phosphor plate between ground and the grid voltage. Under these operating conditions, fluorescence only became visible once the potential applied to the phosphor surpassed that of the mesh grid. Consequently, it was hypothesized that electrons would be unable to reach the photodiode array when it was grounded, using the existing electron beam configuration. Single-shot data was then taken from the imager under various sensitivity configurations to discern whether electrons could be seen at the interface of the sensor.

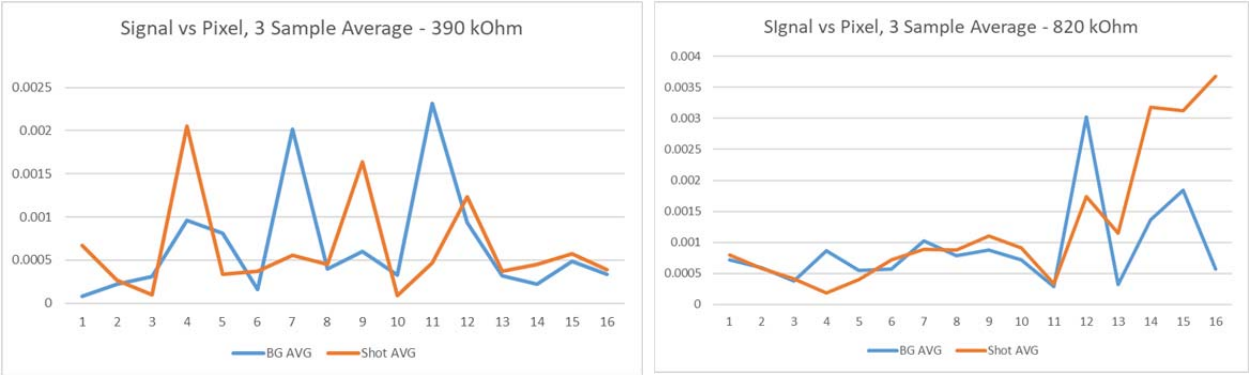

Figure 37: Direct electron noise characteristics collected with various resistor sizes.

Figure 37 shows the results obtained by pulsing the solenoid as the shot was taken, compared to dark images taken without the electron beam or solenoid pulse. In this test configuration, pixels 0-7 use the pre-soldered 3.3 kΩ resistors, while pixels 8-15 are connected to the resistor hot-swap board. This was done to discern the impact of thermal noise on the system, while leaving the original 3.3 kΩ resistors on the left half of the sensor to provide a baseline of noise, given that they are not positioned directly in the path of the beam. It can be

observed given the scale of these plots that the variation we see is due to the intrinsic noise of the system, and the noise level increases with the impedance of the resistors used on the hot-swap board. Interestingly, the thermal noise due to the connected impedance was not significantly higher on the 390 kΩ test. In the 820 kΩ test however, the noise level on the right half of the sensor is much greater than on pixels 0-7.

The plots were generated using the average of three background shots and three real-data shots with the electron beam enabled. Since the real-data shots closely match the trend of the background data, it can be concluded that the sensor is not currently seeing electrons at its interface. This is largely due to limitations with the testing setup because the photodiode array is unable to be biased higher than the MCP and mesh grid in the same manner as the phosphor plate. Given that the MCP and mesh grid are positively biased and in the path of the electrons as they make their way toward the imager, it is unlikely any will overcome the attractive force of the nearby hardware and be imaged. Furthermore, the surrounding vacuum chamber also being grounded also makes it is unlikely that the electrons are able make it to the imager without being sunk elsewhere. Since this result was obtained both with the photodiode array and with the decreased phosphor plate bias, it follows that the testing setup must be changed such that the MCP bias is much closer to ground to allow electrons to pass through and collide with the imager.

Chapter 4

4.1 Conclusion

In this research, a high-speed direct electron and visible light imager was developed to provide proof of concept that direct-electron imagers are a feasible replacement for existing x-ray imaging technologies and may provide meaningful improvements to temporal and spatial resolution. Significant work was done to develop the imager's supporting hardware and software subsystems to facilitate high sensitivity and high frequency measurements. A custom PCB was designed to amplify imager photocurrents and convert them to measurable voltages, and several custom software components were designed to allow various distinct imager operating modes. The imager was tested extensively at atmosphere with visible light to ensure consistent and repeatable operation.

The imaging system was used to experimentally determine that the quantum efficiency of the photodiode array in response to 532-nm incident light was approximately 85%. This value was calculated using two different methods, with the error between them being less than 0.002%. Polychromatic light from a mobile phone flashlight was analyzed to characterize the system's ability to image visible light of differing intensities and confirm the device's linear relationship of incident signal with photocurrent and output voltage. For each intensity value, the sensor displayed an exponential decay away from the target pixel, in which the rate of decay was a function of the source brightness. Furthermore, signal confinement testing was performed using a handheld red laser pointer in two different orientations to confirm the spatial resolution of the imager.

The photodiode array was also placed under vacuum in place of the pre-existing phosphor plate testing stage to characterize its response to direct electron signal, and connected to the resistor hot-swap board so that its hardware sensitivity could be adjusted. While the imager operated as expected in response to photons while under vacuum, further changes to the system such as signal shielding will be necessary to obtain quantitative data in response to the direct electron source. Based on the successful visible light testing performed and the specification of the photodiode array, we are very confident that the imager will accurately sense direct electrons under vacuum with high spatial and temporal resolution following the completion of the electron source configuration.

4.2 Future Work

4.2.1 Finalize Electron Source

The next validation step for the imager is to characterize its response to direct electrons. This was not possible using the existing setup due to the photodiode needing to be grounded to operate. Since the configuration used for the phosphor plate testing used increasing potential on the MCP, mesh grid, and phosphor plate, electrons are being attracted to these components and the photodiode array is not able to collect any signal. At the time of writing, a tube of wire mesh shielding has been added to the vacuum chamber and will be biased very near the potential of the electron source, which will allow a greater percentage of the signal to be guided through the free space before being amplified by the MCP. This will also allow an adjustment of the potentials used to bias the electron source, MCP, and grid to be more negative, which will allow the grounded photodiode array to collect the signal.

4.2.2 Implementation of hCMOS Sensor

The work presented in this research provides proof of concept that high-speed direct electron imaging is feasible and should be iterated upon further. After the electron source is fully configured and the response of the photodiode array is characterized, the sensor may then be replaced with hybrid-CMOS sensors in development by the Lawrence Livermore National Laboratory [2]. This would greatly benefit the temporal and spatial resolution of the system, allowing electrons to be imaged at nanosecond time scale across 25-μm pixels. The system will be integrated with the existing hardware and will operate using the existing photodiode trigger as its coarse trigger. The result of this work will be a very high-speed direct electron imaging system, capable of capturing data in critical diagnostic applications where traditional imagers are insufficient.

Appendix A: Device Firmware and Software

A-1: Arduino 10-kHz Data Collection Firmware

```
// REFERENCES:
// adapted from https://stackoverflow.com/questions/28887617/arduino-fill-array-with-values-from-
//analogread [16]
// https://forum.arduino.cc/t/use-external-trigger-to-start-the-timer/567070/3
// https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

// Ryan Gloekler, UC Davis. Hunt Vacuum Microelectronics Lab
// regloekler@ucdavis.edu
// Takes an input trigger from DG535, which causes an interrupt. 100 data samples are taken at 10KHz
// (10 ms total sampling time). Data is then sent to serial, where it can be retrieved by python for
processing.
// Last edit: 6/12/2024

// The code was updated to use polling rather than an interrupt service routine as initially written.
// This was done to prevent noise from the DG535 from triggering, and included a line of code to check
// the input pin again after a set time, to confirm that the trigger was received.

// storing 100 values @ 10KHz = 10ms of data. Solenoid should be active ~5ms.
const unsigned int readings  = 100;
const unsigned int numPixels = 16;
unsigned int analogVals[readings][numPixels]; // create an array to store data readings in
unsigned int i = 0;

const int POLLING_PIN  = 5;
volatile int triggeredFLAG = 0; // initially set interrupt flag to low..

void setup()
{
  // set the polling pin as input, and set it low initially.
  pinMode(POLLING_PIN, INPUT);
  digitalWrite(POLLING_PIN, LOW); // ensure internal pull-ups are off

  Serial.begin(115200);
  analogReference(EXTERNAL); // use an external analog reference for ADC. Allows for manual tuning of
adc sensitivity. 0 --> 1023 == 0 --> x Volts, x is the external reference voltage
}

void loop()
{
```

```
 static uint32_t tStart = micros(); // grab start time
 const uint32_t period = 100; // setting measurement frequency to 10KHz
 uint32_t tNow = micros();

 if (digitalRead(POLLING_PIN) == HIGH){
  delay(0.1); // if this value is unstable, change back to 0.250 (250us). This was stable and worked with
the electron gun
  if (digitalRead(POLLING_PIN) == HIGH){
   triggeredFLAG = 1; // if the pin is still high after 500us, its a shot instead of noise
  }
  //triggeredFLAG = 1; // set the trigger flag high to take data
 }

 if (tNow - tStart >= period && triggeredFLAG) // check to see if collection is active and interrupt is
received
 {
  tStart += period; // update start time to ensure consistent and near-exact period

  // TAKE DATA FROM PIXELS
  analogVals[i][0] = analogRead(A0);
  analogVals[i][1] = analogRead(A1);
  analogVals[i][2] = analogRead(A2);
  analogVals[i][3] = analogRead(A3);
  analogVals[i][4] = analogRead(A4);
  analogVals[i][5] = analogRead(A5);
  analogVals[i][6] = analogRead(A6);
  analogVals[i][7] = analogRead(A7);
  analogVals[i][8] = analogRead(A8);
  analogVals[i][9] = analogRead(A9);
  analogVals[i][10] = analogRead(A10);
  analogVals[i][11] = analogRead(A11);
  analogVals[i][12] = analogRead(A12);
  analogVals[i][13] = analogRead(A13);
  analogVals[i][14] = analogRead(A14);
  analogVals[i][15] = analogRead(A15);

  i++; // increment data counter value for analogVals

  if (i>=readings) {
   int counter = 0;
   Serial.println("Captured Data, exiting");
   for (int x = 0; x < 100; x++) {
    for (int y = 0; y <= 15; y++) {
     Serial.print(analogVals[x][y]);
     Serial.print(", ");
    }
```

```
      Serial.println("Sample: " + String(counter));
      counter++;
    }
    delay(1000); // wait one second for stabilization, then become available for interrupts again
    triggeredFLAG = 0;
    i = 0; //reset to beginning of array, so other readings overwrite existing data
   }
  }
}
```

A-2: Arduino PD Live-Plot Firmware

```
// Ryan Gloekler, UC Davis. Hunt Vacuum Microelectronics Lab
// regloekler@ucdavis.edu
// Simply collects data from the Arduino Analog-Digital Converter pins A0-A15
// Sends collected data to serial bus, such that Python can read and process sensor data
// Last edit: 2/29/2024

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
  // read the input on analog pins:
  int sensorValue = analogRead(A0);
  int sensorValue1 = analogRead(A1);
  int sensorValue2 = analogRead(A2);
  int sensorValue3 = analogRead(A3);
  int sensorValue4 = analogRead(A4);
  int sensorValue5 = analogRead(A5);
  int sensorValue6 = analogRead(A6);
  int sensorValue7 = analogRead(A7);
  int sensorValue8 = analogRead(A8);
  int sensorValue9 = analogRead(A9);
  int sensorValue10 = analogRead(A10);
  int sensorValue11 = analogRead(A11);
  int sensorValue12 = analogRead(A12);
  int sensorValue13 = analogRead(A13);
  int sensorValue14 = analogRead(A14);
  int sensorValue15 = analogRead(A15);


  // send the ADC values to serial for Python collection
  Serial.print(String(sensorValue) + ',');
```

```
  Serial.print(String(sensorValue1) + ',');
  Serial.print(String(sensorValue2) + ',');
  Serial.print(String(sensorValue3) + ',');
  Serial.print(String(sensorValue4) + ',');
  Serial.print(String(sensorValue5) + ',');
  Serial.print(String(sensorValue6) + ',');
  Serial.print(String(sensorValue7) + ',');
  Serial.print(String(sensorValue8) + ',');
  Serial.print(String(sensorValue9) + ',');
  Serial.print(String(sensorValue10) + ',');
  Serial.print(String(sensorValue11) + ',');
  Serial.print(String(sensorValue12) + ',');
  Serial.print(String(sensorValue13) + ',');
  Serial.print(String(sensorValue14) + ',');
  Serial.println(String(sensorValue15));
  delay(175); // 175 ms between sensor readings (preserve serial data integrity for live plotting)

}
```

A-3 Python Data Processing and Storage Script

```
# @author Ryan Gloekler, UC Davis. Hunt Vacuum Microelectronics Lab
# regloekler@ucdavis.edu
# Reads and processes Serial data received from Arduino ADC module, sampling at 10 KHz for 10 ms.
# Plots the average data across the sensor over this time period.
# Stores data samples for each shot in labeled Excel files for later analysis
# Last updated: 6/12/2024

from datetime import *
from time import *
from matplotlib import pyplot as plt
from matplotlib.pyplot import draw
import serial, re, sys, csv, os
import numpy as np

MULTIPLIER = 10 # added an in-code multiplier to help with data visualization as shots take place
#----------------------- DEFINE HELPER FUNCTIONS -----------------------------

# grab current data from the serial bus
def grab_serial(ser):
   line = ser.readline()
   #line = sys.stdout.write(str(ser.readline()))
   sys.stdout.flush()
   return str(line)
```

```
# used for scaling a matrix of values - not used for averaged pixel value cases
def scale_values(data_array):
   # normalize
   for val in range(len(data_array)):
      try: data_array[val] /= 1023 # arduino uses a 10-bit ADC, 1024 values
      except: print("Couldn't convert... Trying again")
   return data_array

# create a new array that can be plotted (expected data format)
def make_plottable(input_data):
   plottable_data = []
   for val in range(0,2): plottable_data.append(input_data)
   return plottable_data

# creates a new csv file that background data can be written to
def create_csv(filenm):
   # create a csv file to write background noise data to, and a header array
   file = open('./' + filenm, 'a', encoding = 'UTF8', newline='')
   header = ['Pixel 1', 'Pixel 2', 'Pixel 3', 'Pixel 4', 'Pixel 5', 'Pixel 6', 'Pixel 7',
   'Pixel 8', 'Pixel 9', 'Pixel 10', 'Pixel 11', 'Pixel 12', 'Pixel 13', 'Pixel 14', 'Pixel 15', 'Pixel 16', 'Sample No.',
'Serial Time']

   writer = csv.writer(file)
   writer.writerow(header)
   return writer

#----------------------- IMPLEMENT MAIN FUNCTIONALITY --------------------------
# since the heavy lifting is done in firmware, we just need to read all the
# data from serial here...
def main():
   # search for csv, create one if it is not found
   filenm = 'shot_data/data_collection_dump.csv'
   if filenm not in os.listdir('./'):
      file_writer = create_csv(filenm)
   else:
      file_writer = csv.writer(open(filenm, 'a', encoding = 'UTF8', newline=''))

   # set up serial communication
   ser = serial.Serial(port='COM3', baudrate=115200, timeout = 1)
   # setup regex for data parsing
   non_decimal = re.compile(r'[^\d,]+')

   # set sample counter to 0
   counter = 0
```

```
  # main loop of operation - constantly scan serial,
  # until we receive data from ADC (firmware has been triggered)
  averaged = []
  averaged_np = []
  new_shot = True
  while True:
    line = grab_serial(ser).split(', ')
    if len(line) > 2: # get only proper pixel data from serial
      received_time = perf_counter()
      # if a shot has occurred, create a new CSV file for it.
      if new_shot:
        cur_time = strftime("%Y-%m-%d_%H-%M-%S", gmtime())
        print(cur_time)
        new_shot = False # set the new shot flag, so we don't re-create files for each line

        newfile = 'shot_data/shot_' + cur_time +'.csv'
        if newfile not in os.listdir('./'):
          file_writer = create_csv(newfile)
        else:
          file_writer = csv.writer(open(newfile, 'a', encoding = 'UTF8', newline=''))


      # create processing array and start counting data points
      processed = []
      counter += 1

      # convert all values to integers, omit all other characters
      for val in range(len(line)):
        converted = non_decimal.sub('', line[val])
        processed.append(converted)

      # trim array to include only floating point values
      processed = [int(val) for val in processed]

      # scale values to range of 0-1, omitting the sample number row
      scaled = scale_values(processed[0:-1])

      # optionally include background subtraction (with noise data from Excel)
      #bg_vals =
[0.002600196,0.076774194,0.077556207,0.028670577,0.002424242,0,0.000371457,0.051104594,0.085
278592,0.110215054,0.011612903,0.005679374,0.09914956,0.0028348,0.105493646,0.082091887]
      bg_vals = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
      scaled = [val - bg_vals[scaled.index(val)] for val in scaled]

      # convert data to floating point and get a plottable version for later
      scaled_converted = [float(val) for val in scaled]
```

```
averaged.append(scaled_converted)

plottable = make_plottable(scaled_converted)

# add metadata to the data, for final Excel readability
scaled.append(processed[-1])
scaled.append(str(datetime.now()))
# print(scaled)

# write the current set of pixel data to csv, and close file
file_writer.writerow(scaled)

# print out a scaled version of the data to terminal, much like is done in live-update script
printable = [val * MULTIPLIER for val in scaled[0:-2]]
print(printable)

# if received a complete set of data, plot it
if counter % 100 == 0:
   final_time = perf_counter()
   print('Final data transfer, processing, and csv write time is: ' + str(final_time - received_time))
   # if we are ready to finish handling this shot, average data
   # for plotting

   # print(averaged)

   # create a numpy array object to handle averaging
   # uncomment this block to enable averaging over the total 10ms
   # period
   print('\nPLOTTING DATA AVERAGED OVER 10ms')
   np_array = np.array(averaged)
   averaged_np = np.mean(np_array, axis=0)
   plottable = make_plottable(averaged_np)
   plottable = [[val*MULTIPLIER for val in row] for row in plottable]

   # Set the figure size, and scale
   plt.rcParams["figure.figsize"] = [7, 3.50]
   plt.rcParams["figure.autolayout"] = True

   print('\nPlotting light-intensity map...')

   # set up the figure for plotting pixels
   fig, ax = plt.subplots(1,1)
   image = plottable
   im = ax.imshow(image, cmap='Reds', vmin = 0, vmax= 1)
   ax.set_title('PD Array Output')
   ax.set_xlabel('Pixel Number')
```

```
            plt.show() # plot the last sample received
            averaged = [] # reset the averaging list for the next shot

            new_shot = True # reset shot flag for next shot data collection

if __name__ == '__main__':
   main()
```

A-4: Python Sensor-Data Live Plotting Script

```
# Ryan Gloekler, Hunt Vacuum Microelectronics Lab, MGXI Project
# @author: regloekler@ucdavis.edu
# last updated 6/12/2024

# Operation: Run draw_pixels_updated.py from the terminal. This will create a data
# file containing background signal, for the first 150 samples. The program will then
# terminate. Subsequent runs will not terminate, unless by the user, and will instead
# plot the live data, so that changes at the photodiode interface can be seen in
# real time. The values displayed have the average of the background data subtracted from
# them, so as to ensure that as much noise can be cut from the sensor readings as possible.

from datetime import *
from matplotlib import pyplot as plt
from matplotlib.pyplot import draw
import numpy as np
import serial, re, sys, csv, os, time

MULTIPLIER = 10 # a value that multiplies all of the observed pixel values by 10, so that they more easily
show up on live updating script

bg_data = 'background_pixelData.csv' # used to create csv files...
data_csv = 'bg_subtracted_data.csv'  # used to create new data files (bg_sub)

# grab current data from the serial bus
def grab_serial(ser):
   line = ser.readline()
   #line = sys.stdout.write(str(ser.readline()))
   sys.stdout.flush()
   return str(line)

# used for scaling a matrix of values - not used for averaged pixel value cases
def scale_values(data_array):
   # normalize
   for val in range(len(data_array)):
```

```
        try: data_array[val] /= 1023
        except: print("Couldn't convert... Trying again")
    return data_array

# create a new array that can be plotted (expected data format)
def make_plottable(input_data):
    plottable_data = []
    for val in range(0,2): plottable_data.append(input_data)
    return plottable_data

# creates a new csv file that background data can be written to
def create_csv(filenm):
    # create a csv file to write background noise data to, and a header array
    file = open('./' + filenm, 'w', encoding = 'UTF8', newline='')
    header = ['Pixel 1', 'Pixel 2', 'Pixel 3', 'Pixel 4', 'Pixel 5', 'Pixel 6', 'Pixel 7',
    'Pixel 8', 'Pixel 9', 'Pixel 10', 'Pixel 11', 'Pixel 12', 'Pixel 13', 'Pixel 14', 'Pixel 15', 'Pixel 16', 'TIME']

    writer = csv.writer(file)
    writer.writerow(header)
    return writer

# get the averaged data from the existing data file
def collect_csv_data():
    # read from csv file, and get the average of each column for
    # background subtraction

    data_array = []
    with open(bg_data, "r") as f:
        reader = csv.reader(f, delimiter="\t")
        for i, line in enumerate(reader):
            data_array.append(line)
        data_array.pop(0)

    averaged = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    for row in range(len(data_array)):
        data_row = data_array[row][0].split(',')
        for val in range(len(data_row)):
            averaged[val] += float(data_row[val])

    averaged = [element / len(data_array) for element in averaged]
    return averaged

# main loop: creates figure, sets up serial communication, gets data from serial,
# performs background subtraction, and live plots the PD array data
def main():
    averaged_time = []
```

```
timer_counter = 0
# establish comms with the Arduino
ser = serial.Serial(port='COM3', baudrate=9600, timeout = 1)

# setup regex, and grab some serial data
non_decimal = re.compile(r'[^\d,]+')

placeholder = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.0, 0.8, 0.5, 0.2]

# Set the figure size, and scale
plt.rcParams["figure.figsize"] = [7, 3.50]
plt.rcParams["figure.autolayout"] = True

print('\nPlotting light-intensity map')

# set up the figure for plotting pixels
fig, ax = plt.subplots(1,1)
image = make_plottable(placeholder)
im = ax.imshow(image, cmap='Reds', vmin = 0, vmax= 1)
ax.set_title('PD Array Output')
ax.set_xlabel('Pixel Number')

# initiate the main loop of the program... Grab and plot data
# log the first several hundred samples to a csv file for background subtraction

# artifact of older code that used background subtraction from Excel file
background_counter = -1
writer = None
averaged = collect_csv_data()
data_writer = create_csv(data_csv)

# create a variable to store the maximum observed pixel value, over many cycles
CURRENT_MAX = 0
while True:

   initial_time = time.perf_counter()
   # grab a new set of data from each of the pixels
   new_data = grab_serial(ser).split(',')

   # convert all values to integers, so they may be scaled
   for val in range(len(new_data)):
      converted = non_decimal.sub('', new_data[val])
      if converted: new_data[val] = int(converted)

   # scale the values for plotting, and get an array from them that can be
   # plotted
```

```
    scaled = scale_values(new_data)
    #print(type(scaled[0]))


    # optionally include background subtraction (with noise data from Excel)
    # bg_vals =
[0.002531769,0.074379277,0.074613881,0.026774194,0.002326491,0,0.000645161,0.048807429,0.084
623656,0.109032258,0.010420332,0.00516129,0.099921799,0.00285435,0.105913978,0.08113392]
    bg_vals = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    scaled = [val - bg_vals[scaled.index(val)] for val in scaled]


    plottable = [[]]

    """ SKIPPING BG SUBTRACTION; negligible dark current, not needed for live updating purposes.
    if bg_data in os.listdir('./'): # don't bg sub if there is no bg
      for val in range(len(scaled)):
        try:
          scaled[val] = float(scaled[val]) - float(averaged[val]) # perform the background subtraction,
and plot
        except:
          print("Couldn't update.. Trying again")
    """
    #print(scaled, end='')
    for val in scaled: print('%.7f' % (float(val) * MULTIPLIER), end = ', ')
    print(',  MAX VALUE: ' + str('%.7f' % max(scaled)) + ' Pixel: ' + str(scaled.index(max(scaled))), end = ' ')

    if max(scaled) > CURRENT_MAX:
      CURRENT_MAX = max(scaled) * MULTIPLIER
    print('Overall MAX: ' + str('%.7f' % CURRENT_MAX))



    plottable = make_plottable(scaled)
    plottable = [[val*MULTIPLIER for val in row] for row in plottable]


    #print(plottable)
    try: # update the canvas...
      image = plottable
      im.set_data(image)
      fig.canvas.draw_idle()
      plt.pause(0.00001)
    except:
      print('Couldnt update canvas... Data error.')
```

```
        # write the scaled data to the appropriate spreadsheet
        if writer: writer.writerow(scaled)
        else:
            scaled.append(str(datetime.now())) # add the time to the data...
            data_writer.writerow(scaled)

        # check conditionals for background data
        if background_counter > -1: background_counter += 1
        if background_counter >= 150: exit('Finished Collecting Background data.')
        final_time = time.perf_counter()

        timer_counter += 1
        averaged_time.append(final_time - initial_time)
        if timer_counter % 200 == 0:
            print(np.mean(averaged_time))
    file.close()

if __name__ == '__main__':
    main()
```

A-5: Python Excel Data Visualization Script

```
# @author Ryan Gloekler, UC Davis. Hunt Vacuum Microelectronics Lab
# regloekler@ucdavis.edu
# Last updated: 6/12/2024

# NOTE: this program expects that the data being processed (csv file) contains
# only one shot, separate shots of interest into separate csv files.

# Operation: Run the script as: py read_excelData.py 'datafile.csv' x [animate]
# where x is the time values you would like to visualize, and animate is an optional
# parameter, allowing for a playback animation of all 100 data points over 10 seconds
# (10ms data collection period)

import os, sys, csv
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.pyplot import draw

# create a new array that can be plotted (expected data format)
def make_plottable(input_data):
    plottable_data = []
    for val in range(0,2): plottable_data.append(input_data)
    return plottable_data
```

```
# grab the appropriate sample number (divide time by .1 ms)
def time_to_row(sample_time):
   # each row represents a sample, taken 100 us apart
   sample_length = 0.1 # ms
   sample_num = sample_time / sample_length
   sample_rounded = round(sample_num)
   return sample_num, sample_rounded

def plot_sample(data, time_val):
   # Set the figure size, and scale
   plt.rcParams["figure.figsize"] = [7, 3.50]
   plt.rcParams["figure.autolayout"] = True

   print('\nPlotting data closest to ' + str(time_val) + 'ms.')

   # set up the figure for plotting pixels
   fig, ax = plt.subplots(1,1)
   ax.set_title('PD Array Output at ' + str(time_val) + 'ms')
   ax.set_xlabel('Pixel Number')

   image = data
   im = ax.imshow(image, cmap='Reds', vmin = 0, vmax= 1)
   plt.show() # plot the last sample received
   return

def main():
   global ANIMATE
   ANIMATE = False
   if len(sys.argv) < 2:
      exit("Provide Excel file name and desired time")
   else:
      filenm, time = sys.argv[1], float(sys.argv[2])
      excel_data = []

   if len(sys.argv) > 3:
      ANIMATE = True
      print('Animating data over 10ms.')

   print("Displaying data from the data point closest to " + str(time) + "ms")
   sample_value = time_to_row(time)[1]

   # catch out of bounds time values
   if sample_value > 100: sample_value = 100
   if sample_value < 1:   sample_value = 1
```

```
    with open(filenm, mode = 'r') as datafile:
        linereader = csv.reader(datafile)
        for line in linereader: excel_data.append(line)

    # remove header from excel data
    excel_data.pop(0)
    if ANIMATE:
        # set up the figure for plotting pixels
        # Set the figure size, and scale
        plt.rcParams["figure.figsize"] = [7, 3.50]
        plt.rcParams["figure.autolayout"] = True

        fig, ax = plt.subplots(1,1)
        image = make_plottable([1] * 15)
        im = ax.imshow(image, cmap='Reds', vmin = 0, vmax= 1)
        ax.set_title('PD Array Output')
        ax.set_xlabel('Pixel Number')

        for i in range(len(excel_data) - 1):
            datapoint = excel_data[i][:-2]
            scaled_converted = [float(val) for val in datapoint]
            plottable = make_plottable(scaled_converted)

            image = plottable
            im.set_data(image)
            fig.canvas.draw_idle()
            plt.pause(0.1) # pause for 100ms before drawing next frame
            # this means that 100 samples * .1 s = 10s
            # therefore, we are animating 10ms of data over 10s

    else:
        datapoint = excel_data[sample_value][:-2]
        scaled_converted = [float(val) for val in datapoint]
        plottable = make_plottable(scaled_converted)

        plot_sample(plottable, time)
    return

if __name__ == "__main__":
    main()
```

A-6: Photocurrent Approximation Script

```
# -*- coding: utf-8 -*-
```

```
"""
Created on Tue Apr 30 14:01:39 2024

@author: mamort@ucdavis.edu
"""

import numpy as np
import math

source_type = "laser"

"""   532 LASER CASE   """
a_overlap = 9.3145          # [mm^2] area overlap of circular laser over single rectangle pixel - calculated by Jordan using autocad
A_PD_pix = 2 * 5    # [mm^2] area of single PD pixel
A_532 = a_overlap   # area of pixels lit up by laser

p_3r = 4.9e-3         # [mW] power of class 3a laser
a_3r = np.pi * 2.4**2       # [mm^2] laser beam area ; d of clear aperture is 4.8mm
g_overlap = 0.8364  # percent of signal Gaussian beam power on single pixel - from real data
P_532 = (p_3r * g_overlap) / a_overlap

R_532 = 0.275     # [A/W] based on wavelength of laser & PD responsivity

"""   3 pix 532 LASER CASE   """
a3pix_overlap = 9.3145          # [mm^2] area overlap of circular laser over single rectangle pixel - calculated by Jordan using AutoCad
A_PD_3pix = 2 * 5    # [mm^2] area of single PD pixel
#A_3pix532 = A_PD_3pix * 3  # area of pixels lit up by laser

p_3r_3pix = 4.9e-3         # [mW] power of class 3a laser
a_3r_3pix = np.pi * 2.4**2       # [mm^2] laser beam area ; d of clear aperture is 4.8mm
g3pix_overlap = 0.8364  # percent of Gaussian beam power on single pixel - from real data
P_3pix532 = (p_3r_3pix/a_3r_3pix)
A_3pix532 = a_3r_3pix   # beam fully covered by 3 pixels
R_3pix532 = 0.275     # [A/W] based on wavelength (532nm) of laser & PD responsivity

"""   LASER CASE   """
p_3a = 4.3e-3         # [mW] power of class 3a laser
a_3a = 3 * 5       # [mm^2] laser beam area

A_PD_pix = 2 * 5    # [mm^2] area of single PD pixel
A_laser = A_PD_pix * math.floor(a_3a//A_PD_pix)     # area of pixels lit up by laser

P_laser = p_3a/A_PD_pix
R_laser = 0.35     # [A/W] based on wavelength (~650nm) of laser & PD responsivity
```

```
"""   3 pix LASER CASE   """
p_3a_3pix = 4.9e-3         # [mW] power of class 3a laser
a_3a_3pix = 2.5 * 5        # [mm^2] laser beam area
g_laser_overlap = 0.59494
P_3pixlaser = ((p_3a_3pix * g_laser_overlap) / a_3a_3pix)
A_3pixlaser = a_3a_3pix
R_3pixlaser = 0.35      # [A/W] based on wavelength (~650nm) of laser & PD responsivity

"""   E- GUN CASE   """
#i_gun = 1.1e-6         # [A] max current measured on phosphor screen
i_gun = 0.4e-6
V_grid = 3000          # [V] grid in front of PD
V_HK = -6600           # [V] 'ground' of e-gun
V_gun = V_grid - V_HK   # [V] potentials between HK & grid
a_gun = 50 * 50         # [mm^2] area of phosphor screen that collected 1.1uA
P_gun = i_gun * V_gun / a_gun

A_gun = A_PD_pix * 1   # [mm^2] full detector area of all 16 pixels

R_gun = 0.225          # [A/W] based on eV of grids & PD responsivity

unity = 1          # assume if PD worked ideally
QE_est = 0.95      # assumed if PD is Si based
QE_PD = 0.9        # random number - will update when have better idea what this is

if source_type == "laser":
   P = P_laser        # [mW/mm^2] power density of irradiation on detector
   A = A_laser        # [mm^2] detector cross-sectional area/ area of beam irradiation
   R = R_laser        # [A/W] responsivity of the material
   eta = unity       # quantum efficiency of detector
elif source_type == "3pixlaser":
   P = P_3pixlaser        # [mW/mm^2] power density of irradiation on detector
   A = A_3pixlaser        # [mm^2] detector cross-sectional area/ area of beam irradiation
   R = R_3pixlaser        # [A/W] responsivity of the material
   eta = unity       # quantum efficiency of detector
elif source_type == "532laser":
   P = P_532        # [mW/mm^2] power density of irradiation on detector
   A = A_532        # [mm^2] detector cross-sectional area/ area of beam irradiation
   R = R_532        # [A/W] responsivity of the material
   eta = unity       # quantum efficiency of detector
elif source_type == "3pix532":
   P = P_3pix532        # [mW/mm^2] power density of irradiation on detector
   A = A_3pix532        # [mm^2] detector cross-sectional area/ area of beam irradiation
   R = R_3pix532        # [A/W] responsivity of the material
   eta = unity       # quantum efficiency of detector
```

```
elif source_type == "egun":
    P = P_gun        # [W/mm^2] power density of irradiation on detector
    A = A_gun        # [mm^2] detector cross-sectional area/ area of beam irradiation
    R = R_gun        # [A/W] responsivity of the material
    eta = unity      # quantum efficiency of detector
else:
    print ("need source type")

I = P * A * R * eta
print("estimated photocurrent generated by PD [mA]: \n", I*1e3)
```

References

[1] S. Tremsin, J. V. Vallerga, and O. H. W. Siegmund, "Overview of spatial and timing resolution of event counting detectors with microchannel plates," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 949, p. 162768, Jan. 2020. doi:10.1016/j.nima.2019.162768

[2] O. Jagutzki, "A broad-application microchannel-plate detector system for advanced particle or photon detection tasks: Large area imaging, precise multi-hit timing information and high detection rate," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 477, no. 1–3, pp. 244–249, Jan. 2002. doi:10.1016/s0168-9002(01)01839-3

[3] M. S. Dayton, "The electron detection performance of the 'Icarus' hCMOS imaging sensor," *Hard X-Ray, Gamma-Ray, and Neutron Detector Physics XXIII*, September 2021. doi:10.1117/12.2594346

[4] F. T. Ulaby and U. Ravaioli, *Fundamentals of Applied Electromagnetics*. Harlow, Essex, England: Pearson Education Limited, 2015, pp. 265-267

[5] "Photodiode basics," Wavelength Electronics, https://www.teamwavelength.com/photodiode-basics/#:~:text=A%20photodiode%20is%20a%20semiconductor,abundance%20of%20electrons%20(negative) (accessed March 5, 2024).

[6] Opto Diode Corporation, "Photodiode 16 Element", AXUV16ELG datasheet, November 2019.

[7] IXYS Corporation, "30-Ampere Low-Side Ultrafast MOSFET Drivers", DS-IXD_630-R04 datasheet, April 2017.

[8] "How to design Transimpedance amplifier circuits: Video: Ti.com," Videos, https://training.ti.com/how-design-transimpedance-amplifier-circuits (accessed February 29, 2024).

[9] Transimpedance Amplifier Circuit, https://www.ti.com/lit/an/sboa268a/sboa268a.pdf?ts=1708964666468&ref_url=https%253A%252F%252Fwww.google.com%252F (accessed February 29, 2024).

[10] "What you need to know about Transimpedance amplifiers – part 1," SSZTBC4 Technical article TI.com, https://www.ti.com/document-viewer/lit/html/SSZTBC4#:~:text=Transimpedance%20amplifiers%20(TIAs)%20act%20as,output%20current%20to%20a%20voltage (accessed Feb. 29, 2024).

[11] Low Noise, Precision Operational Amplifier, https://www.analog.com/media/en/technical-documentation/data-sheets/OP27.pdf (accessed March 4, 2024).

[12] Swe, T. & Yeo, Kiat Seng. (2001). An Accurate Photodiode Model for DC and High Frequency SPICE Circuit Simulation. Nanotech. 1.

[13] "Visualization with python," Matplotlib, https://matplotlib.org/ (accessed Aug. 18, 2024).

[14] Maithil, "Demystifying high-performance multiplexed data-acquisition systems," Demystifying High-Performance Multiplexed Data-Acquisition Systems | Analog Devices, https://www.analog.com/en/resources/analog-dialogue/articles/demystifying-data-acquisition-systems.html (accessed June 12, 2024).

[15] Dr. R. Paschotta, "Quantum efficiency," – quantum yield, laser, gain medium, fluorescence, photodiode, https://www.rpphotonics.com/quantum_efficiency.html#:~:text=In%20the%20visible%20and%20near,depending%20on%20the%20wavelength%20region. (accessed Sep. 2, 2024).

[16] K. Rudalevicius, "Arduino: Fill array with values from analogread()," Stack Overflow, https://stackoverflow.com/questions/28887617/arduino-fill-array-with-values-from-analogread (accessed Jun. 23, 2024).

[17] R. Gloekler, "RGloekler/PD_ARRAY_CODE: Working Directory for Photodiode Array Interface Firmware and software," GitHub, https://github.com/RGloekler/PD_ARRAY_CODE (accessed Jun. 23, 2024).