

# Atwater Kent Laboratories Sustainable and Modular Display System

## Team Members

Zach Bergquist, *zbergquist@wpi.edu*

Olivia Hanson, *omhanson@wpi.edu*

Jonathan Lee, *jlee4@wpi.edu*

Quincy Rhodes, *qcrhodes@wpi.edu*

Brandon Terry, *bcterry@wpi.edu*

Tim Vermilyea, *tmvermilyea@wpi.edu*

## Advisor

Alexander Wyglinski, *alexw@wpi.edu*

## Co-Advisor

Shamsnaz V. Bhada, *ssvirani@wpi.edu*



**Worcester Polytechnic Institute**

Worcester, MA

This project proposal is submitted in partial fulfillment of the degree requirements of Worcester Polytechnic Institute.

The views and opinions expressed herein are those of the authors and do not necessarily reflect the positions or opinions of Worcester Polytechnic Institute.

## **Abstract**

The overarching goal of this project was to inspire interest and awareness in Electrical and Computer Engineering (ECE). To fulfill this goal, the project integrates several pre-built and hand-made electrical systems together to create a sustainable, independent, non-intrusive system that interacts with people in an energy efficient and attractive manner. Each part of the system represents something that can be implemented for practical or hobbyist work, showing the wide range of systems that apply to electrical and computer engineering.

# Acknowledgements

Our team would like to acknowledge the following people for their generous contributions to this project:

- **Professor Alexander Wyglinski**, for his constant guidance and support throughout the year
- **Professor Shamsnaz Virani Bhada**, for her guidance with designing the system and team management
- **Professor Jennifer DeWinter**, for her assistance with designing the animations and system positioning
- **Professor Donald R. Brown**, for his assistance and support as ECE Department Head
- **Justin Woodard and National Grid, Inc**, for his suggestions and support, as well as their generous donation of a solar panel, 80/20 mounting materials, and sealed lead-acid battery
- **Bill Appleyard**, for his patience, helpfulness, and support with obtaining hardware for the project and assistance with building the mounts and housings
- **Ronal O'Brien**, for his assistance and advice on how to implement our project safely and effectively
- **Würth Elektronik**, for their very generous donation of two wireless power development kit, as well as engineering support throughout the year
- **Samtec**, for providing samples of IDSD-12-D-02.00 connectors

# Table of Contents

<b>Abstract</b>	II
<b>Table of Contents</b>	IV
<b>List of Figures</b>	VII
<b>List of Tables</b>	XI
<b>List of Acronyms</b>	XII
<b>Executive Summary</b>	XIV
1.1 Motivation	1
1.2 Current State of the Art in Interactive Displays	1
1.3 Proposed Interactive Display	2
1.4 Core Technologies	3
1.4.1 Solar System	4
1.4.2 WPT	5
1.5 Technical Challenges	6
1.6 Contributions	7
1.7 Report Organization	10
<b>Chapter 2: Tutorial Survey of Fundamental Technologies</b>	11
2.1 Solar Systems	11
2.2 Wireless Power Transfer	13
2.3 Modular Display Technology	17
2.4 SPI Data Communication Protocol	19
2.5 Chapter Summary	20
<b>Chapter 3: Proposed Approach</b>	21
3.1 Problem Statement	21
3.2 Design Options Available	21
3.2.1 Solar System	21
3.2.2 Batteries	23
3.2.3 WPT	25
3.2.4 Display	27
3.3 Final Selection of Design	28
3.3.1 Solar System	28

3.3.2 Batteries	29
3.3.3 WPT	31
3.3.4 Display	36
3.4 Project Logistics	39
3.5 Chapter Summary	40
<b>Chapter 4: Methodology and Implementation Display</b>	41
4.1 Main Frame	41
4.2 Brackets	44
4.3 Screws	46
4.4 Custom Printed Circuit Board	47
4.5 Complete Module Design	51
4.6 Chapter Summary	52
<b>Chapter 5: Methodology and Implementation of Software</b>	53
5.1 Operating System	53
5.1.2: Raspbian Installation	54
5.1.3 Raspbian Setup	55
5.2 APA102 library	58
5.2.2 APA102 Functions	60
5.2.3 Module_test.py	61
5.3 OpenWeatherMap API	62
5.3.1 OpenWeather API Outputs	65
5.4 Matrix.py	67
5.4.1 Animations	68
5.4.2 Main()	76
5.5 Chapter Summary	78
<b>Chapter 6: Full System Integration</b>	79
6.1 Solar	81
6.2 WPT	85
6.3 Display	90
6.4 Chapter Summary	90
<b>Chapter 7: Verification</b>	91
7.1 Solar	91
7.2 WPT	92

7.3 Display	94
7.4 Total Integration	97
7.5 Chapter Summary	101
<b>Chapter 8: Conclusion</b>	102
8.1 Future Work	104
<b>Appendix B: APA102 Python Library</b>	106
<b>Appendix C: Led Matrix Driver, Matrix.py</b>	113
<b>Appendix D: Weather-test.py</b>	135
<b>References</b>	138

## List of Figures

Figure 1: Photo of the System Power Path During Demo

Figure 2: Display Outside During Demo

Figure 3: Proposed Display Placement

Figure 4: Proposed LED Display

Figure 5: Proposed Solar Panel Placement

Figure 6: Proposed WPT Placement

Figure 7: Solar Cell, Module, and System Distinction

Figure 8: Sample Solar System

Figure 9: Conductor Impedance vs. Skin Depth

Figure 10: Inductive Coupling Wireless Power System

Figure 11: Magnetic Resonance Wireless Power System

Figure 12: LED Data Frame Operation

Figure 13: APA102C Layout

Figure 14: SK9822 Layout

Figure 15: SPI Clock vs Data

Figure 16: SPI Protocol

Figure 17: Monocrystalline, Polycrystalline, and Thin Film Amorphous Comparison

Figure 18: Lead Acid Battery Comparison

Figure 19: Common Wireless Phone Charging Pad.

Figure 20: Rezenze Power Transmit and Receive Coils

Figure 21: Yingli Solar YL305P-35b Diagram

Figure 22: Interstate Batteries DCM0075

Figure 23: HQST 30A Charge Controller

Figure 24: Custom Inverter/Oscillator Schematic

Figure 25: Custom Inverter/Oscillator Circuit

Figure 26: Oscilloscope Readout of Custom Wireless Power System Results

Figure 27: 200W WPT System Block Diagram

Figure 28: LED Chain of One Module

Figure 29: LED Chain of Multiple Modules

Figure 30: Multiple Viewing Angles of the LED Module

Figure 31: Module Wings

Figure 32: Uneven Wing Heights

Figure 33: Screw Holes

Figure 34: Module Channels

Figure 35: Vertical Edge Connecting Brackets

Figure 36: Horizontal Edge Connecting Brackets

Figure 37: Center Connecting Brackets

Figure 38: Everbilt M3-0.5 x 25 mm Phillips Flat-Head Machine Screws

Figure 39: Left Edge PCB

Figure 40: Right Edge PCB

Figure 41: Left Edge PCB Diagram

Figure 42: Right Edge PCB Diagram

Figure 43: Right Cap PCB

Figure 44: Left Cap PCB

Figure 45: Display Stand Model

Figure 46: SD Card Formatter

Figure 47: Installing Raspbian Image

Figure 48 : Raspi-config

Figure 49: Interfacing Options

Figure 50: SPI Enable

Figure 51: Properly Installed Python 3

Figure 52: APA102 vs Sk9822 Dataframe

Figure 53: Series Data Structure

Figure 54 : OpenWeather Main Page

Figure 55: API selection



Figure 56: Generate Key  
Figure 57: Weather API Response  
Figure 58: Weather() Class Return Structure  
Figure 59: Rain Animation  
Figure 60: Sunshine Animation  
Figure 61: Moon Animation  
Figure 62: Cloud Animation  
Figure 63: Partly Cloudy Animation  
Figure 64: Night Partly Cloudy  
Figure 65: Snow Animation  
Figure 66: Thunderstorm Animation  
Figure 67: Fog Animation  
Figure 68: Static Text Animation  
Figure 69: Scrolling Text Animation  
Figure 70: Full System Block Diagram  
Figure 71: Location of Solar Panel  
Figure 72: Right Leg of Solar Mount  
Figure 73: Middle Leg of Solar Mount  
Figure 74: Left Leg of Solar Mount  
Figure 75: Bottom Connection of Solar Mount  
Figure 76: Back of Solar Mount  
Figure 77: Aideepen DC-DC Buck Converter  
Figure 78: Supplementary Transmitter Circuit Schematic  
Figure 79: Oscilloscope Readout of Supplementary Wireless Power Circuit  
Figure 80: BQ25703a Charge Controller Schematic  
Figure 81: Model of WPT Housing  
Figure 82: WPT System Under Test  
Figure 83: WPT Efficiency vs. Input Power Plot  
Figure 84: Completed and Activated Display  
Figure 85: Display with Acrylic Case  
Figure 86 : Annotated Full System Diagram

Figure 87: “Outside” Electronics

Figure 88: “Inside” Electronics

Figure 89: Circuitry Housing

Figure 90: Battery Housing

Figure 91: Final Running System

## **List of Tables**

Table 1: Display System Cost

Table 2: Current Draw of LEDs With Red Color

Table 3: Solar Panel Test Results

Table 4: WPT Load Testing Data

# List of Acronyms

**A4WP** - Alliance For Wireless Power  
**AC** - Alternating Current  
**AH** - Amp-hour  
**AGM** - Absorbent Glass Mat  
**API** - Application Programming Interface  
**CdTe** - Cadmium Telluride  
**CKI** - Clock Input  
**CIGS** - Copper Indium Gallium Diselenide  
**DC** - Direct Current  
**ECE** - Electrical and Computer Engineering  
**E-Ink** - Electrophoretic Ink  
**EMI** - Electromagnetic Interference  
**EPD** - Electronic Paper Display  
**FET** - Field Effect Transistor  
**I2C** - Inter-Integrated Circuit  
**IoT** - Internet of Things  
**LCD** - Liquid Crystal Display  
**LED** - Light Emitting Diode  
**MOSFET** - Metal Oxide Semiconductor Field Effect Transistor  
**MOSI** - Master Out Slave In  
**MPPT** - Maximum Power Point Tracking  
**NOCT** - Nominal Operating Cell Temperature  
**OLED** - Organic Light Emitting Diode  
**PCB** - Printed Circuit Board  
**PV** - Solar Photovoltaic  
**PWM** - Pulse Width Modulation  
**RF** - Radio Frequency  
**RFID** - Radio Frequency Identification  
**RGB** - Red Green Blue

**SCC** - Supplementary Control Circuit

**SPI** - Serial Peripheral Interface

**SDI** - Serial Data Input

**WPC** - Wireless Power Consortium

**WPI** - Worcester Polytechnic Institute

**WPT** - Wireless Power Transfer

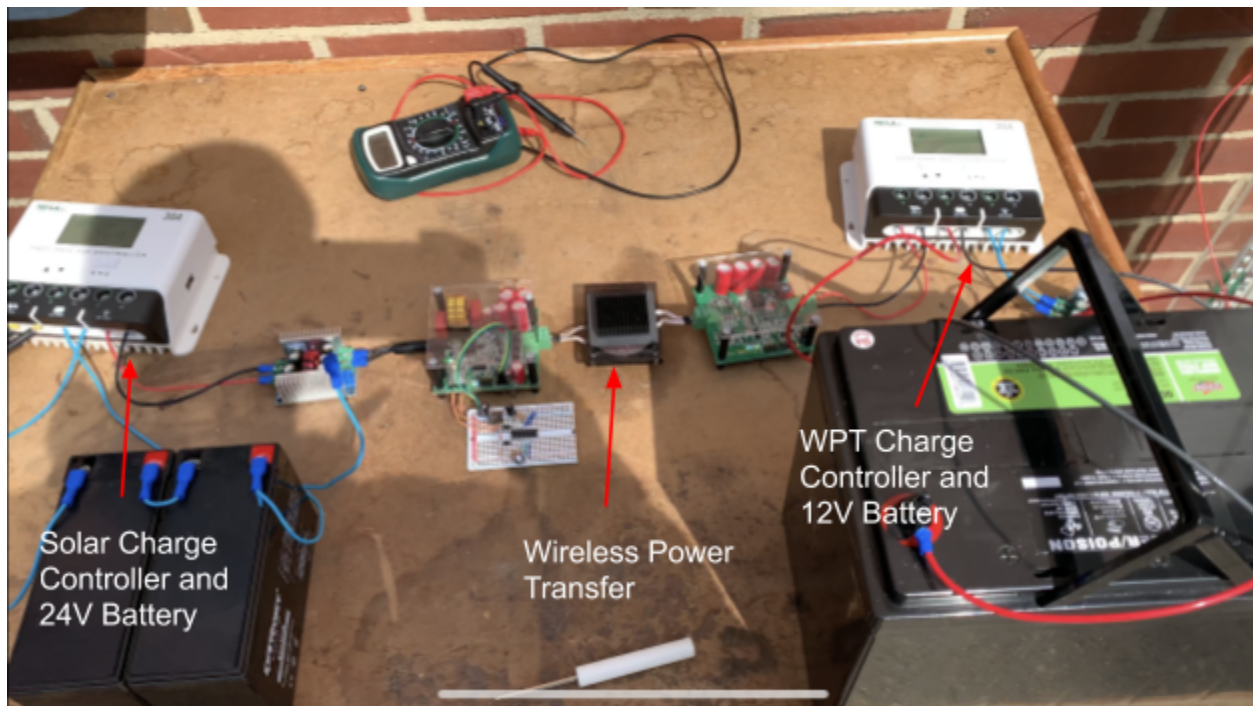
## Executive Summary

ECE is a fascinating field of study and offers a wide range of topics at Worcester Polytechnic Institute (WPI). ECE involves the study of old and new technologies, as there are frequent groundbreaking advances in this subject. However, Atwater Kent Laboratories, WPI's center of ECE, provides little in the way of conveying the many exciting facets of ECE.

This project aims to give Atwater Kent Laboratory visitors an eye-catching display of modern technologies. It sheds light on many intriguing topics available within the ECE major field at WPI. Multiple concepts together to create a complete product and accomplish this goal. The three main aspects of the project are solar power, wireless power transfer (WPT), and a light emitting diode (LED) display. These components exhibit use of modern technology and cover a wide range of ECE topics such as power electronics, radio frequency (RF) circuitry, and embedded processing.

This project consists of a photovoltaic (PV) panel that generates power to be transferred via WPT which powers a custom modular LED display. The system requires no external power sources and is completely self-contained. Since the solar panel can produce enough power necessary for the system, the system does not need any additional power sources. If there is a lack of sunlight to charge the panel, the main battery at full charge can power the display continuously for about 3 days. Considering how the motion sensor is designed to lessen the display's usage, the system can operate for a substantial amount of time. Thus, this system is entirely independent from Atwater Kent Laboratories. To produce and store power for our system, we first use the solar panel to charge two 12 volt, 7 amp-hour batteries in parallel. The batteries charge on a hysteresis curve provided by the first charge controller, up to about 14 volts and no lower than 11 volts. Excess power from the solar panel and the two 12 volt batteries leave the charge controller at 12 volts, which is converted by a boost converter to 24 volts and delivered to the WPT transmitting coil and received by the receiving coil. The transmitter of the WPT unit must be notified on start-up to begin power transfer via a mechanical switch. To ensure no physical action is required, a supplementary control circuit (SCC) was designed to simulate a mechanical switch actuation 5 seconds after start-up power is supplied. The WPT unit passes the power through an exterior window to the remainder of the system. The 24 volt output of the WPT unit goes to the second charge controller, which charges a 12 volt and 75 amp-hour

(AH) battery to serve as a power buffer from the power source to the load. The output of the charge controller from the battery is stepped down via a buck converter to provide the display with 5 volts, used to power the custom display and embedded processor that drives the display. The charge controllers, batteries, and wireless power system during a live demo is shown below in Figure 1.



***Figure 1: Photo of the system power path from solar charge controller to wireless power receiver charge controller. The wireless power transfer units are shown in the center. The 24V 9AH battery is on the left, and the 12V 75AH battery is on the right.***

The display consists of 9 handmade display modules, each with a 10 by 10 grid of addressable RGB LEDs. Using addressable LEDs allows for the selection of individual pixels and pixel colors to provide the widest range of possibilities for images and text. These displays were designed to be built out of 3D printed housings, several custom PCBs, and factory made LED strips and would be assembled by hand using solder and adhesives. The display logic is driven by a Raspberry Pi microprocessor to create a range of eye-catching animations on the display and sense the proximity of bystanders. The display shows a variety of pieces of information spread over the nine modules. The Raspberry Pi grabs time and date data from the

internet and displays this. It also grabs weather information through a weather application programming interface (API) on the internet and displays it. Once a bystander enters a certain range and waits for a period of time, the display will turn on and start displaying the pre-programmed information. Figure 2 below shows the display showing information during the live demo.



*Figure 2: The display is shown outside during the live demo. It is displaying the date, time, temperature, weather, and wind speed. The Raspberry Pi and distance sensor are located at the top of the display.*

The final product was successfully built based upon our initial designs for the project. The final system itself integrated a solar panel, charging circuitry with WPT, and a handmade



modular LED matrix. The final system was hung within a six foot tall acrylic display and the trickle charging circuitry and batteries were to be encased within acrylic boxes as well for protection from the elements. Some small changes needed to be made in the trickle charging circuit due to unexpected integration errors between the charge controller and buck converters and between the WPT's transmitter and supplementary circuit.

The only major change in the system's design was how WPT was implemented. The WPT transmits enough power from transmitter to receiver to power the display. However, the proposed idea to transmit power through the exterior windows failed as the exterior windows are too thick to transmit the required power to the rest of the system. In lieu of this fact, the WPT unit will now be displayed with a representative window pane in between the coils, instead of being mounted on one of Atwater Kent Laboratory's exterior windows.

# **Chapter 1: Introduction**

## **1.1 Motivation**

The primary motivation of this project was to create an attention grabbing system that displays many facets of ECE to inspire interest in ECE. The project accomplishes this by piecing many different concepts together to create a finished product that is self-contained and intriguing. These different concepts stem from multiple different paths of learning in ECE at WPI, and give visitors to Atwater Kent Laboratories insight on what ECE majors are capable of creating. The system involves technology in the fields of power systems, RF circuitry, and embedded processing. The system must also be self-contained, requiring no external power, as a way of promoting sustainability in ECE. Instead, it utilizes solar power as a green carbon neutral energy source.

## **1.2 Current State of the Art in Interactive Displays**

The overall system will be powered using a solar panel that trickle-charges a high capacity battery with WPT. The battery will then be connected to the display, which is located in the Atwater Kent Laboratories stairwell. The display needs to be able to attract the attention of bypassers, students, and potential students to show them how ECE knowledge can be applied. To complete this objective, a display that can create legible characters was needed, show interesting animations, and can be both power and cost efficient.

The display portion of the system is defined by a set of requirements and properties necessary to ensure the display may function properly with the system. Qualities of the desired display include: power efficiency, sufficient resolution, and programmatic accessibility. The display and its computing hardware must run solely on the battery storage for power efficiency. The resolution is deterministic of the content presented on the display. In addition, the computing and programming to display the information must be straightforward to code and well supported for future iterations of the project. After research and discussion, it was decided that designing and building a proprietary display would be the path forward for this system. There were four display types considered during the research process.

In today's day and age, there are a large variety of different display types and more varieties of those specific display types as well. The displays that were considered for this project were handmade and factory built LED matrices, liquid crystal display (LCD) screens, organic light emitting diode (OLED) screens, and electronic ink (E-ink) screens. The handmade LED matrices are self-designed by daisy chaining either LEDs or LED strips together. Factory built LED matrices are the same as handmade matrices, but a producer supplies fully built matrices with a casing or housing to hold the matrix together. LCD screens are one of the most common screen types today, which use crystals to diffuse inorganic light into specific colors. OLED screens are similar to LCD screens, but they use organic materials to produce a wider spectrum of light to the display. E-ink screens use magnetic fields to control black magnetic "ink" within the display to either come to the front of the screen or the back of it within a white liquid. This construction will create a white or black color effect. More details about how these displays work is provided in Chapter 3.

### **1.3 Proposed Interactive Display**

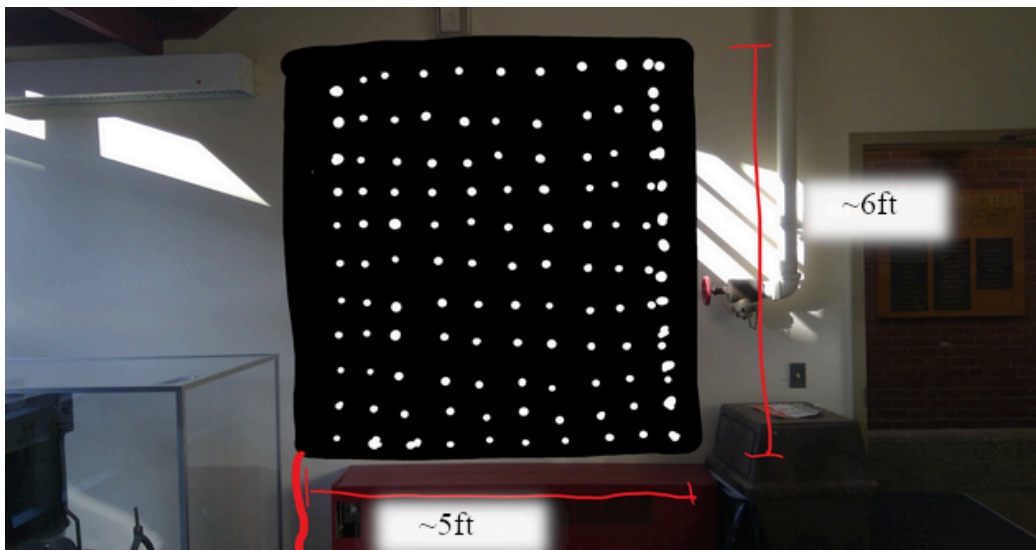
The initial design for the display was a custom built grid of 3000 addressable RGB LEDs. These LEDs can be programmed individually such that the grid would form a pixel screen, able to display information and animations. The proposed size of the display was six feet by five feet, allowing a large enough size to be eye catching while maintaining a reasonable LED density, which is important when trying to convey coherent information.

The display was also meant to be interactive, with external sensors to sense user input. Examples include an ultrasonic sensor to detect the presence of a human, allowing the display to only activate when someone is nearby to save power. A gesture detector could be used for people to give input to the display without having to touch anything. Users could wave their hands in a certain direction to change information displayed or even play games.

Figure 3 and Figure 4 show the proposed location for the display was at the bottom of the front Atwater Kent Laboratories stairwell, an area that receives heavy foot traffic daily. This is also previously the location of old electrical equipment displayed in transparent boxes which have since been removed. This project would serve as a modern replacement to fill this area.



*Figure 3: The displays were proposed to be placed under this staircase since anyone passing by could view the display from both inside and outside. Additionally since the old equipment was being moved we thought it would be nice to replace the older technology with newer modern components.*



*Figure 4: The proposed LED display was six feet tall and 5 feet wide. In the proposed design, the display would have 3000 LEDs spread out within that area and display various animations and texts to attract the eye of both the curious and scholar. Additionally, the display was going to be interactive by having emulators loaded to simulate common games.*

## 1.4 Core Technologies

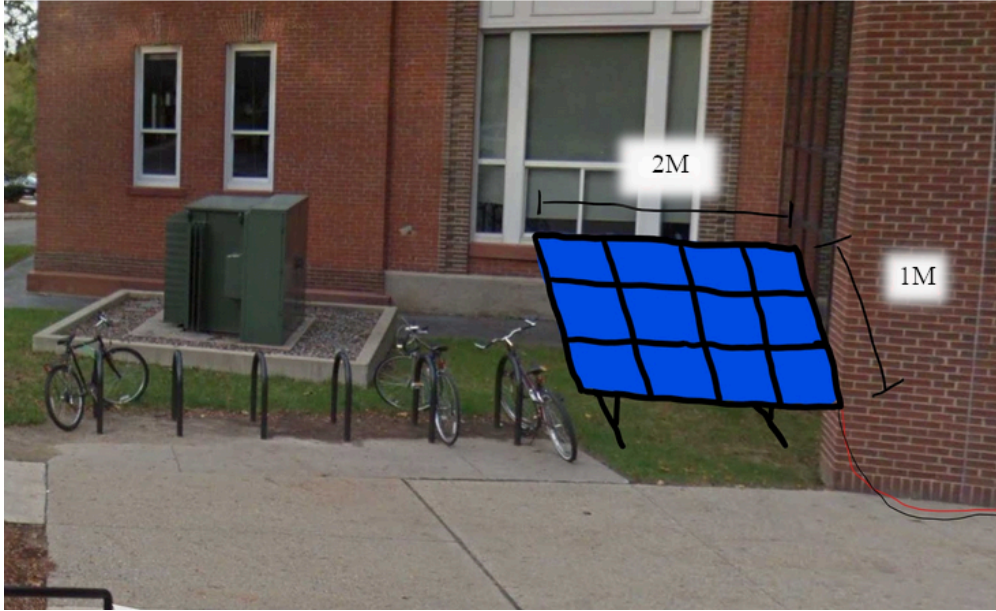
The purpose of this section is to outline the different technologies in this system. These technologies assisted with the creation of different designs, understanding how each system

worked, and provided more options to add flexibility to the design. Before doing any research for this system, it was split into 3 major components. The components were solar, WPT, and display. These were the major components stemming from the initial three objectives of the project: “To create a self-sustainable system that would demonstrate the abilities of electrical and computer engineers and intriguing modern technologies.”

### **1.4.1 Solar System**

Solar energy generation has been increasing in the United States since 2014. As of 2018, solar energy made up about 1.5% of the electricity generation in the United States. A major benefit of solar energy is the minimal effect that the systems have on the environment. Unfortunately, the amount of power generation is variable because the amount of sunlight that reaches Earth’s surface is based on several factors such as season, weather, location, and time of day [1]. Solar energy has a wide variety of applications, ranging from powering buildings to small portable devices and lighting neighborhoods [2]. It can also be used to generate thermal energy, provide power for transportation, and even the distillation of water [3].

This project uses a solar module to generate power for the display inside WPI’s ECE building, Atwater Kent Laboratories. The solar portion of the system consists of three main parts: the solar panel, battery, and charge controller. The solar panel generates power to store in a battery before being converted from 14V to 24V and sent through the WPT system. The different options considered for each part will be discussed in Chapter 3. The originally proposed location for the solar panel was next to the bicycle racks outside Atwater Kent Laboratories, as shown in Figure 5.



*Figure 5: The solar panel would be placed between the bike rack and Atwater Kent Laboratories, keeping out of the way of pedestrians but also in a viewable location. This placement would allow the solar panel to face the sun and attract the attention of visitors.*

### 1.4.2 WPT

Wireless Power Transfer was first discovered in 1898, when Nikola Tesla was able to demonstrate that the high frequency alternating currents produced by his resonant tesla coil transformers were able to transfer power over very short distances without any wires [4]. WPT technology has applications ranging from low-power (microwatts to milliwatts) radio frequency identification (RFID) and satellite communications applications to high power (watts to kilowatts) applications, and have been the focus of most industry research and development [5]. WPT has implemented approaches such as radio waves, optical link, ultrasound, capacitive coupling, inductive coupling or magnetic resonance, and microwaves [5]. Although each technology has their benefits and their drawbacks, inductive coupling has become the most popular technology for WPT at high power [5].

In this project, wireless power is desired for two critical reasons. The first reason is purely practical. The solar panel must be outside to generate electricity, but the display must be inside to avoid exposing it to the elements, and there are walls and windows that separate the two locations. Thus, there must be a third component of this project that can take the power generated

outside the building and route it inside. Wireless power transfer is capable of transferring this magnitude of electricity over a medium like glass windows, so it is a good choice for this solution. The second critical reason goes back to the motivation of this project. One of the goals of this project is to create a system that can showcase modern technologies that spark interest in both ECE and non-ECE minds. Many people might not know about the capabilities of wireless power transfer, and this part of the system is aimed at demonstrating what wireless power can do when coupled with other exciting technologies in the modern world. The location shown in Figure 6 was chosen for WPT based on these constraints.



*Figure 6: The WPT would go onto the front window so that people inside and outside would be able to have a good view of the component and be able to see what it was doing. Additionally, this would showcase a newer piece of technology and garner attention from those interested.*

## 1.5 Technical Challenges

With many separate parts needed to implement this system, the project faced many technical challenges. One of the greatest challenges was making the system independent of grid power. This meant finding a solar panel within the project's budget that could keep the system running daily. This challenge was overcome by allowing the system to consume less power. Various power saving strategies such as low power configurations and brightness modulation for the display kept the project within budget.

WPT was also a challenge. Creating circuitry that could transmit large amounts of power through a window separation of at least 1 centimeter would involve much research, calculation,

and experimentation. Fortunately, a pre-built system was obtained to accomplish this task. The pre-built system removed the complication of manually constructing the system, which accomplished high efficiency and kept the project on schedule.

One of the problems that arose when designing and creating the display was making sure there were enough LEDs for the display to be an adequate size while maintaining a high enough density to make it readable. More LEDs would draw more power and require a higher budget. Furthermore, a larger display would take more time to construct. A modular design would be simpler to construct and allow flexibility with the display's size.

Overall, many of the elements of the system are not specifically designed to work with each other. The solar system is not meant to be directly compatible with the wireless power system, and the display cannot just plug right into the solar panel or wireless power receiver. There are many supporting elements of this system that are *crucial* to its success. One of the most difficult parts in designing this system was integrating each subsystem of the project to ensure that they could work as one fluid system, rather than a few smaller systems with disconnects. There were many moments when roadblocks emerged that blocked the function of one or more parts of the system. It all came down to determining all of the additional components each system needed to function correctly as a whole, which took plenty of time and money. Given a longer timeline with a project like this, it is always important to assess the system as a whole to determine the needs of each subsystem and research every possible solution for integrating systems before making any major decisions.

## **1.6 Contributions**

### **Jonathan Lee**

In A term, member Jonathan Lee contributed to research on the potential displays, prototyping code for LED animations, testing LED power draw, finding and following up with sponsors and donations, and project management. In B term, member Jonathan Lee worked on defining final goals and objectives of the project, feasibility of having a connected WPI database, reaching out to sponsors, and project management. In C term, member Jonathan Lee worked on designing mounts and housings for project components, creating 3D models and figures of potential displays and components, coordinating with associates to print out modules and build



displays, debugging and testing code and display modules, project management, and contributing to the final report.

### **Zachary Bergquist**

In B term, member Zachary Bergquist worked on creating the prototype LED module for the displays, creating a bootable SD card with Raspbian OS, designed the PCBs, and developed and simulated code for LEDs. In C term, member Zachary Bergquist worked on creating a modified library to drive the LEDs, generating code to displaying texts according to a rendering library, generating code for interfacing with an application program interface for weather information, generating code for reading object distance, setting up a secure shell server and virtual network computing server for the Raspberry Pi, maintaining a Github Repository containing code information, setup Raspberry Pi to run code upon startup, testing and debugging displays, and contributing to the final report.

### **Timothy Vermilyea**

In A term, member Timothy Vermilyea worked on initial research for wireless power transfer. He designed, prototyped, and tested the first iteration of the wireless power component of the project, and contacted Würth Elektronik when an alternate solution was needed. He also corresponded with the WPI facilities department to determine initial installation roadblocks. In B Term, Timothy Vermilyea utilized SOLIDWORKS to create the 3D models for the display modules and printed the first prototype module. He also worked on design, layout, and ordering of the custom PCBs and machine screws for the display modules. In C term, Timothy Vermilyea tested and integrated the wireless power transfer element into the full system. He designed, prototyped, and tested the supplementary control circuit for wireless power transfer. He worked on converting spreadsheet animations to Python dictionary entries, and drafted some C++ and Python code for communicating with I<sup>2</sup>C devices. Additionally, he assisted with the building and testing of the solar mount and acrylic display cases, the debugging of the module solder joints, and the wiring of the full system power path. Lastly, he completed integration testing of the charge controllers to and from solar panel and inside battery.

## **Olivia Hanson**

In A term, member Olivia Hanson documented meeting discussions and contributed to research on solar systems, including potential solar panels, charge controllers, batteries, and inverters. She also assisted in finding sponsors and donations, including attending an energy symposium and recruiting Justin Woodard from National Grid to assist with this project. In B term, Olivia Hanson assisted with overall project management including continuing to document all meeting discussions, contact with sponsors, and researching models within the budget for the different parts of the solar system. In C term, Olivia Hanson assisted with the testing of the solar module, documenting meeting discussions, and lead report organization and edits.

## **Brandon Terry**

In A term, member Brandon Terry researched varying display technologies, specifically LED strip libraries and power consumption. The power analysis led to further discussions over display size and overall design of the proprietary display. In B Term, he contributed to the development and design of the modular display. Particularly, he proposed the layout of the LEDs and how to interface with the PCBs. In C term, Brandon Terry researched and integrated the charge controllers for the system. In addition, he debugged intermittent solder joints, shorts, and dead LEDs present on the fabricated LED modules. Throughout the entire project, Brandon Terry gave help wherever needed, such as WPT load testing, solar panel mount construction, Raspberry Pi programmatic solutions, and system integration.

## **Quincy Rhodes**

In A term, member Quincy Rhodes contributed to research on displays and display design, research on DC to DC converters, and LED testing. In B term Quincy Rhodes further contributed research on the display design, creating a model for the display housing, constructing the display for the small scale model, testing DC to DC converters, and creating the alphanumeric library to be used in programming the display. In C term Quincy Rhodes worked on constructing the display modules, debugging the display modules, testing the solar panel and power electronics circuitry, constructing the mount for the solar panel, and contributing to the final report.

## **1.7 Report Organization**

The structure of the report is detailed in this section. Chapter 1 provides background information of the different technologies considered using for the three major components of the project. Chapter 2 covers any information that following users would need to know in order to use, alter, and improve upon the system and it's design. Chapter 3 outlines the decision process for choosing parts along with the initial tests of those components. Chapter 4 discusses how the display system works, explains how each part was made and designed, and summarizes the costs and power estimates of the system. Chapter 5 covers software development, including the decision to use this operating system to show each of the functions the operating system needs to run. Chapter 6 explains the additional elements necessary for ensuring all of the electronics function as one interrupted and cohesive system. Chapter 7 discusses issues encountered within all of the subsystems and how they were resolved. Chapter 8 provides information on the outcome requirements of all of the systems that need to accomplish in order to produce a fully functional system. Lastly, appendices are attached at the end of the report to provide supplementary information that is too large to include in the main text

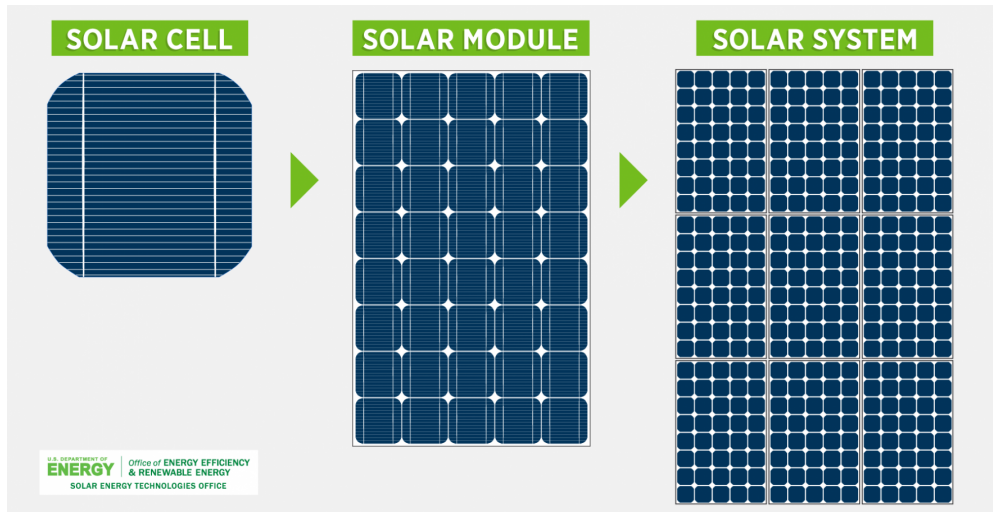
## **Chapter 2: Tutorial Survey of Fundamental Technologies**

The following chapter is an introduction to the fundamental technologies used in this project. These fundamental technologies are solar energy, WPT, and modular display technology. Section 2.1 outlines the various necessary elements of a typical PV system, including the solar panel, charge controller, and battery. Section 2.2 describes the various elements of inductive and magnetic resonant WPT systems such as the coils, oscillator, inverter, and rectifier. It also explains the operation of such a system from an electrical theory perspective. Section 2.3 outlines the theoretical operation of two different types of popular individually addressable LEDs that are often used in strip and matrix type systems. Section 2.4 outlines the basics of the serial peripheral interface protocol and how it will be used in this project.

### **2.1 Solar Systems**

The type of solar energy used in this project was PV technology. PV technology takes direct sunlight and converts it into electrical energy through the use of semiconducting materials, such as silicon. The energy conversion is done through the use of photovoltaic devices such as PV cells. A single cell is small and only produces one or two watts of power, so the cells are connected together in chains to produce larger outputs of power [6]. These are known as panels or modules, and can be built to accommodate almost any power need. When one or more of these panels are connected to an electrical grid with an inverter and battery, it becomes a complete PV system [7]. Distinction between a solar cell, module, and system can be seen in Figure 7.

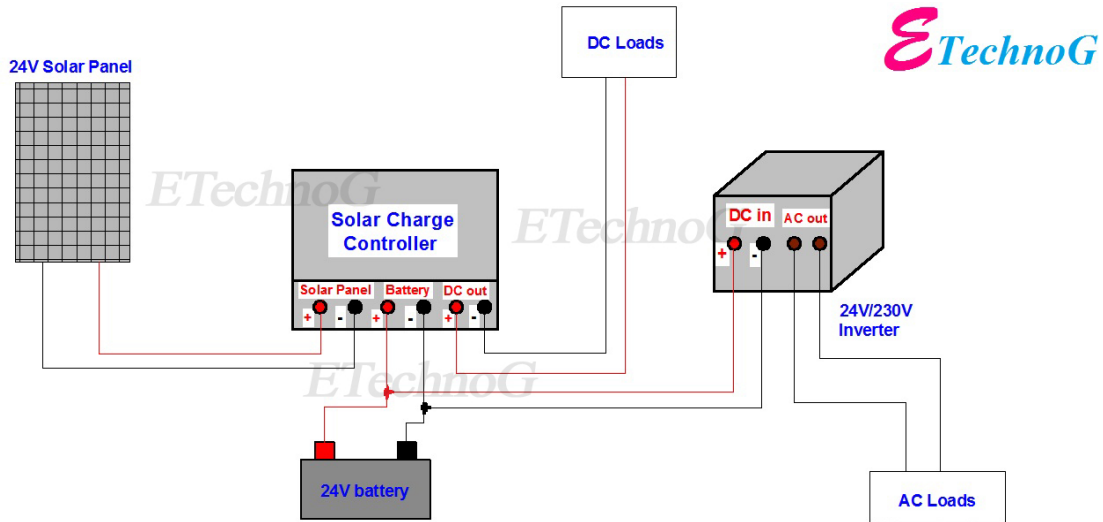
The insulator and metal properties of the semiconducting material allows the conversion light into electricity [8]. There are multiple different types of semiconductor materials used in the solar cells. The most common material is silicon, whose atomic crystal lattice's organized structure provides an efficient energy conversion. Silicon solar cells also have a high efficiency and are sold at a low rate. The cells last up to 25 years without losing more than 20% of their efficiency in that time. Silicon is also easily found and attributes to the high representation in solar modules today.



**Figure 7:** *This figure shows the difference between the solar cell, module, and system. A module consists of many cells while a system consists of many modules.*

Another type of solar cell is called thin-film photovoltaics, where two PV semiconductors used in thin-film cells are copper indium gallium selenide (CIGS) and cadmium telluride (CdTe) [9]. These cells are made by placing one or more layers of the PV material on glass, metal, or plastic. Neither are as resistant to outdoor conditions as silicon, and thus require more protection. The manufacturing processes for CdTe are low cost due to it being the second most common material. However, the efficiency of these cells is not as high as silicon. CIGS cells are more complex because combining four elements makes the process more challenging [9].

Depending on the intended use of the energy generated by the solar cells, an inverter may be necessary, where they are used to convert DC electricity into alternating current (AC) electricity. AC electricity is what is used in homes and for transmission of local energy. Inverters can be implemented into systems by either attaching microinverters to each individual panel or implementing a single inverter into the system to convert the electricity generated by all panels. Attaching the inverters to individual panels is useful if the modules are placed in an area where the system may be partially shaded because it allows each panel to operate individually. Using a single inverter allows the inverter to be easily cooled and serviced when necessary. On average, an inverter will be replaced at least once within a PV module's 25 year lifetime.



**Figure 8:** This figure, taken from the ETechnoG website, displays an example of a solar system. This particular system consists of a solar panel, charge controller, battery and an inverter. [10]

The electricity generated by the system is typically stored in batteries. This allows for the power to be used when there is no direct sunlight shining on the panels, such as during the night. There are several different types of batteries that are commonly used in solar systems, as discussed in Chapter 1. An example of a solar system that incorporates all of these different parts is shown in Figure 8. In this example, the charge controller plays an important role by being connected to the solar panel, battery, inverter, and any loads that may be applied.

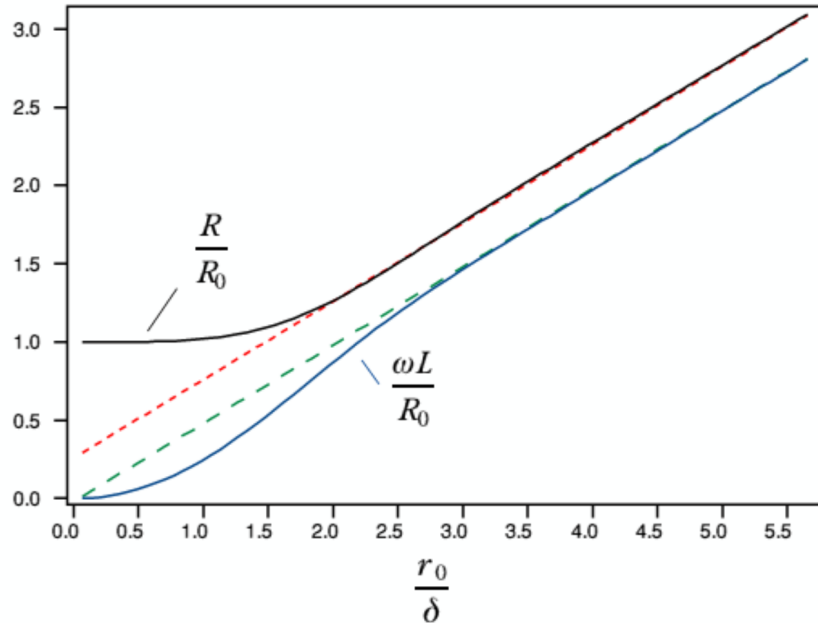
## 2.2 Wireless Power Transfer

WPT via inductive coupling and magnetic resonance can be broken down into three stages: power, transmission-receiving, and rectifying/load [5]. More advanced systems may utilize communication stages to relay information between transmitter and receiver. The power stage typically consists of a rectifier-inverter. The rectifier-inverter takes AC power, converts it into DC power, and reconverts it back to AC power again. This is necessary because wall outlet power typically oscillates at 50-60 Hz, while the frequency required for typical power transfer is much higher. The frequency usually ranges between kilohertz and megahertz depending on the type of power transfer being used. Typical inductive coupling systems operate between 87 and 205 kHz, whereas many resonant systems will operate in the megahertz range [11].

Next, all systems require a transmitter-receiver stage. This stage contains two or more coils, at least one transmitter and one receiver. Both transmitter and receiver coils are typically made of tightly wound Litz wires, coated in enamel to prevent shorting. Litz wire consists of many thin conductors, designed for AC power and to reduce the “Skin Effect” that occurs at higher frequencies [5]. The “Skin Effect” is a phenomenon in the electrical world in which AC in a wire tends to limit the area of travel of electrons to the surface of the wire. This decreases the effective cross-sectional area of the wire and increases resistance [12]. The equation  $\delta = \sqrt{\frac{1}{\pi f \mu \sigma}}$  where  $f$  is the frequency of AC,  $\mu$  is the magnetic permeability of the conductor, and  $\sigma$  is the conductivity of the material allows us to approximate the skin depth of the conductor,  $\delta$ . The skin depth dictates how much of the conductor is used to move electrons [12]. A higher skin depth means that more of the conductor is used and the effective resistance of the conductor is lower [12]. A higher frequency results in a lower skin depth and thus a higher resistance. Figure 9 shows results from tests done at the University of St. Andrews, where wire impedance is plotted as a function of skin depth, normalized by conductor radius [12].

Litz wires decrease the skin effect by using many solid conductors with radii less than the skin depth for that material at particular frequencies, so the skin effect losses become less apparent.

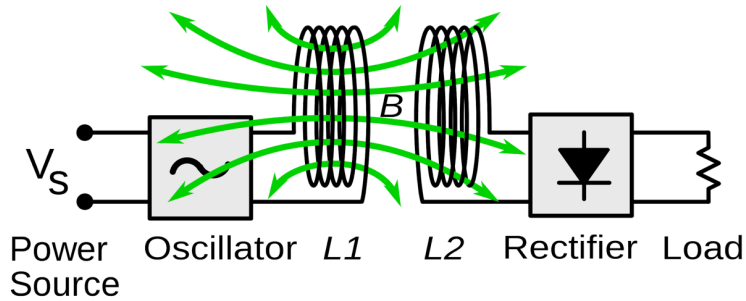
The next component of a WPT system is where inductive coupling and magnetic resonance coupling have the largest difference. The diagram in Figure 10 shows a very basic depiction of an inductive coupling system. The system consists of the previously mentioned power, transmission-receiving, and rectifying/load stages. Figure 11 depicts a magnetic resonance system in the same way.



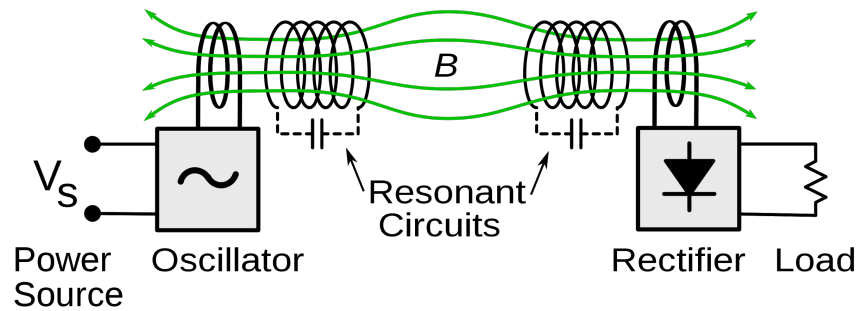
**Figure 9: The impedance of a conductor increases as skin depth increases. Skin depth is normalized by conductor radius,  $R_0$  [12].**

Notice the additional coils and capacitors that make up what are called “resonant circuits” in the transmission-receiving stage of Figure 11. The diagram shows these circuits as separate from the transmitter and receiver coils, but the resonant circuits are just an extension of the already existing coils. Resonance is an electrical phenomenon in which the impedances of elements in a circuit cancel each other out [14]. A resonant circuit in WPT will always have a resistive, inductive, and capacitive element, making a simple RLC circuit [14]. The coil acts as an inductor, and the wire has a DC resistance. Given the equation  $X_C = \frac{1}{j\omega C}$  for capacitive reactance, the reactance moves toward 0 as the frequency increases. Given  $X_L = j\omega L$  for inductive reactance, the reactance increases to  $\infty$  as the frequency increases. At a certain frequency,  $|X_C| = |X_L|$ , but the voltages across the inductor and capacitor in this RLC circuit will be 180 degrees out of phase, making the sum of their reactance zero. As a result, the impedance in the RLC circuit at this particular frequency has only a real element, the DC resistance of the coil. The frequency, in which the circuit has the lowest impedance, is called its “resonant frequency” [14]. Coil manufacturers will often specify the DC resistance of the coil, and the inductance is given. Using the above equations, the ideal capacitance to achieve resonance can be calculated.





**Figure 10: Diagram of a typical inductive coupling wireless power system. Green arrows denote the direction of the magnetic field [3].**



**Figure 11: Diagram of a typical magnetic resonance wireless power system. The magnetic field flows through and is expanded by the resonant circuits [13].**

The third and final stage of a typical WPT system is a rectifying and load stage. The rectifying stage is the process of taking an AC voltage and turning it into a DC voltage. This includes inverting the negative portions of the waveform and smoothing out the ripples with a capacitor. Additionally, most wireless power systems will include either a linear or switching regulator to ensure a constant voltage is delivered to the load. The load in most systems is a battery and battery management system for storage of transmitted power. While the stages outlined are the most basic elements of a WPT system, there are additional elements that can help to enhance the safety and effectiveness of the system. Most consumer devices will have circuit protection elements for overvoltage, overcurrent, and EMI. Some implement communication protocols between transmitter and receiver to maximize efficiency, but all consumer devices require the use of foreign object detection protection as of 2011 [15]. Foreign object detection is required because if a conductive object is placed on a transmitter coil, the

magnetic field created by the coil induces a current in the object. This causes it to rapidly heat up and cause fires, which in turn cause damage and possibly injury.

## 2.3 Modular Display Technology

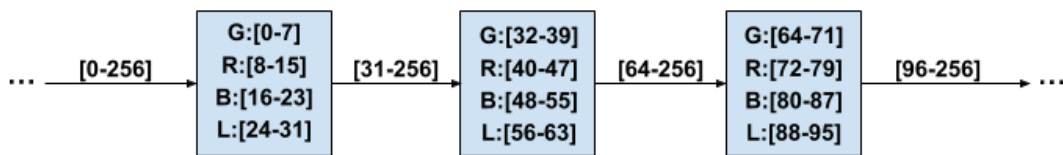
The team took a qualitative approach when researching and defining what form the desired display would take. The purpose of the display is to catch the attention of anyone passing by and to convey information to viewers. Some parameters include power efficiency, availability, modularity, and compatibility. To meet these requirements, the team designed a custom built proprietary system to act as the display. The system involves a matrix layout of RGB LED strips that are controlled by a Raspberry Pi. The LED strips are placed in a 3D printed backing and signals are routed via printed circuit boards (PCB).

The LED strips needed to be addressable and communicate using inter-integrated circuit (I2C) for the Raspberry Pi. In this context, addressable means that each LED in the strip has an address and can be referred to individually so the LED matrix can be any particular color at any specific location. Each LED includes a microchip that reads the first 24-bits provided from the previous LED and passes the remaining data to the next LED in line. There are multiple types of addressable LEDs that exist, two of which were lab tested before implementation. The first LED type is called WS2812b [16]. Even though the WS2812b strip was affordable and power efficient, the communication protocol for the microchip in each LED was undesirable. The WS2812b relied on a single data pin with no clock data and almost no libraries supported on the Raspberry Pi.

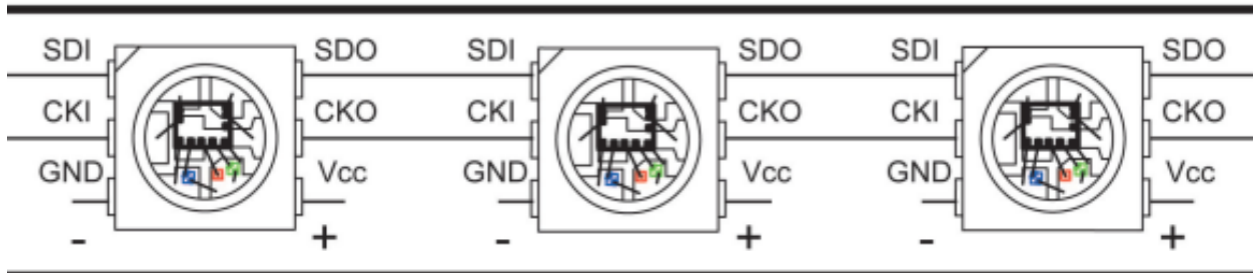
The other LED type is called SK9822 [17], which is similar to the more expensive APA102C [18]. The APA102C's communication protocol is supported on the Raspberry Pi and its power efficiency is adequate, but it is not cost effective. However, the SK9822 is cost effective and communicates in a similar manner as the APA102C. The pin layouts and the daisy chain structure of the two are the same, as shown in Figures 13 and 14. The subtle difference between the two technologies is the time at which the LED color is updated. The SK9822 is updated after the Start Frame, while the APA102C is updated after the corresponding set of LED data. This is described in greater detail in Chapter 5 of this report. The existing libraries for the APA102C are altered to operate correctly with the SK9822 by altering the bit out to operate with the earlier update. Aside from this slight change, the data through the data line and the clock line

is the same. The data line for communication is illustrated in Figure 12. After small scale testing of the SK9822, the LEDs were chosen for the final project [19].

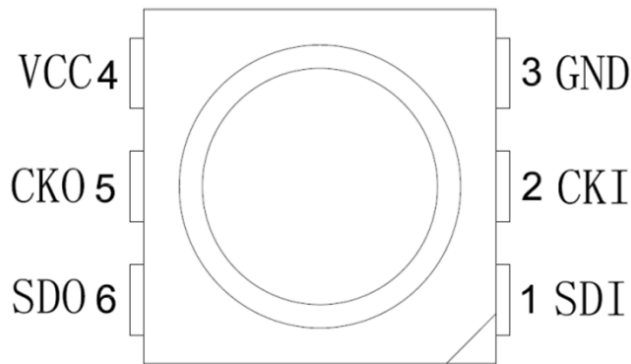
The SK9822 LEDs are controlled by a Raspberry Pi 4. It has a multithreaded, dual core processor with 2 GigaBytes of ram, making it ideal for this project. Raspberry Pi products are well developed and have continued iterations for simple and easy upgradability in the future. The Raspberry Pi 4 has the hardware power and the long term support to ensure success with this project.



**Figure 12:** In the example above, 8-LEDs are cascaded, with 32-bits for each LED. The first 8-bits for green, then red, blue, and light level respectively. All other bits are passed through to the following LED. Note: all three packages, the WS2812b, the APA102C, and the SK9822, operate in this format.



**Figure 13:** The daisy chain structure is repetitive from one LED to the next. This layout is crucial for modularity, as it allows repetition from one module to the next.

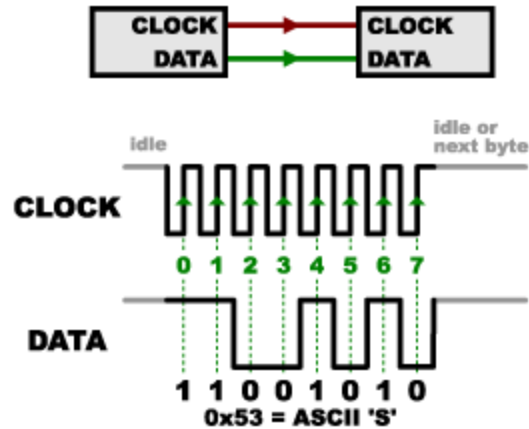


*Figure 14: The pin layout is identical to the pin layout of the APA102C, in Figure 13. The similar layout is crucial for the SK9822 compatibility with the APA102C libraries.*

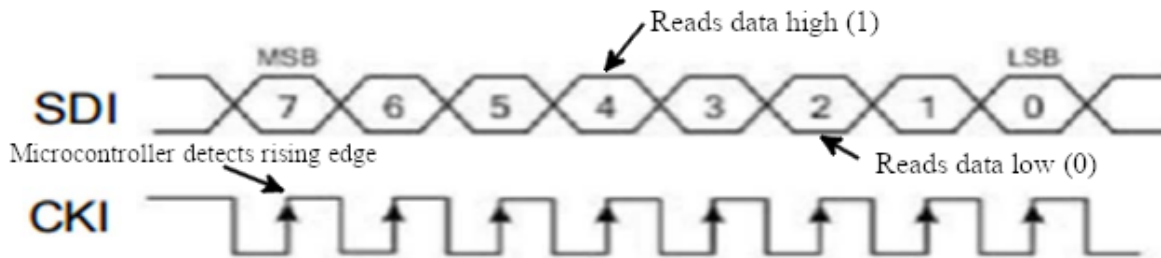
## 2.4 SPI Data Communication Protocol

The serial peripheral interface (SPI) data protocol is utilized when interfacing with the SK9822. The SPI is an interface bus commonly used to send data between microcontrollers and small peripherals, similar to I<sup>2</sup>C and the LED strips used in this application [20]. Unlike most other data buses, SPI is a synchronous data bus. In other words, SPI uses separate lines for data and clock signals to ensure both are in sync. The clock signal oscillates from high to low and tells the microcontroller when to sample data bits. When the microcontroller detects the rising edge of the clock it will read whatever bit of data is present at that time, as shown in Figure 15. Figure 15 shows that 8 bits are read to form the ASCII character ‘S’.

The SPI protocol is used to write bits of data from the SK9822 datasheet [19]. These bits are sent to the slave through the master out slave in (MOSI) pin on the Raspberry Pi (Pin 10). Clock is sent from pin 12 of the Raspberry Pi at a rate of 8MHz. Data is written to each LED one bit at a time and synced with the clock. Figure 16 shows one byte being written to an LED chip. Each clock cycle (CKI) corresponds to a high or low data write (SDI) to the SK9822 (a “1” or “0”). Frames of data were built and clocked into the strip using the SpiDev Python library on the Raspberry Pi.



*Figure 15: Clock is pulsed at a constant rate and data is recorded in sync with the clock for each pulse. In response to the clock, 8 bits are recorded to form the ASCII character 'S'.*



*Figure 16: Taken directly from the SK9822 datasheet, one can see the Serial Data Input (SDI) update as a high or low (1 or 0) bit in sync with the Clock Input (CKI) [19]. Eight bits are shown as one byte of data in the figure.*

## 2.5 Chapter Summary

The operation of a complex system such as the described in this project requires the use of many fundamental technologies. Extensive research was necessary to understand the operation of PV systems, WPT systems, and the available protocols for addressable LEDs. The information outlined in this chapter is a synopsis of the required technologies to build a system of this complexity.

## **Chapter 3: Proposed Approach**

This chapter outlines the process in which technologies and solutions were chosen for the system based on the system requirements. After reviewing the problem statement for this project, all of the possible design options were considered for each system. Various options for each part of the solar system, as well as WPT systems, and display design choices are described. They are evaluated on their feasibility, flexibility, and overall benefit to the system as a whole. Project logistics are also reviewed and explained in this chapter.

### **3.1 Problem Statement**

With this project aiming to bring attention to the many intriguing topics within the ECE major field at WPI, it combined various concepts that showcase modern technology. A PV panel generates power to be transferred via WPT to power a custom modular LED display. The excess power generated by the solar panel charges a battery that will supply power when no sunlight is present. The power generated is converted from 14V, the peak output of the solar panel, to 24V by a boost converter. 24V is delivered to the WPT transmitting coil and received by the receiving coil. The power outputted by the WPT device is used to charge another battery to serve as a power buffer and additional energy storage. The output of the battery is then stepped down to 5V and used to power the custom display and embedded processor that drives the display. The display consists of 9 display modules and is driven by a Raspberry Pi microprocessor to create a range of eye-catching animations on the display.

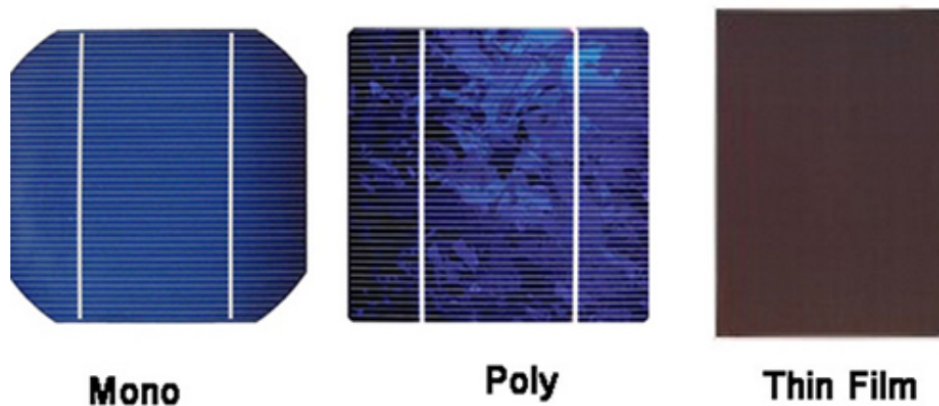
### **3.2 Design Options Available**

#### **3.2.1 Solar System**

The solar panel technologies used on the market today are monocrystalline, polycrystalline, and thin film amorphous [21]. Monocrystalline solar panels are currently the most popular technology. They have the highest efficiency because they are cut from a single source of silicon. As represented in Figure 17, the silicon is formed into bars before being cut into the cells of the panel. These panels convert more than 22.5% of sunlight into electricity.

Monocrystalline solar panels are also space-efficient. They yield the highest power outputs and require the least amount of space compared to any other type. These solar panels also produce more power per square foot of space when used in an array. These solar panels also tend to last longer than their predicted 25 year lifespan [21]. Crystalline solar panels are made from crystalline silicon, a very stable material. These solar panels often last longer than their 25-year warranty [21]. One disadvantage of these panels is that they are more expensive than others with their more complex manufacturing process [21].

The process used to make polycrystalline silicon is simple so the cost of the solar panels is less than monocrystalline. The cells are formed by melting the fragments of silicon together, shown in Figure 17. Polycrystalline solar panels tend to have a slightly lower heat tolerance than monocrystalline solar panels. The efficiency of polycrystalline-based solar panels is typically 14-16% [21]. These solar panels have a low space efficiency, covering a larger surface to output the same electrical power as a solar panel made of monocrystalline silicon.



*Figure 17: This is a comparison of monocrystalline, polycrystalline, and thin film solar cells. Note the silicon fragments that can be seen in the polycrystalline panel compared to the monocrystalline being cut from a single source of silicon. The thin film panel is also a different color because of the different substrates that are added in addition to the silicon.*

While monocrystalline and polycrystalline solar panels have been around for decades, thin film amorphous solar panels are a relatively new technology [22]. They tend to be more expensive than the other two types because they require more mounting equipment, which adds to the overall cost of the system. However, thin film panels do not require as much silicon as a crystalline based solar panel, and the substrates in them can be made out of inexpensive materials

so they have the potential to become less expensive than crystalline based panels. A feature of the thin film module is that they are flexible and allow more creativity with different applications. With the technology being so new and still developing, these panels have lower efficiency rates and tend to degrade faster than crystalline based solar panels [23].

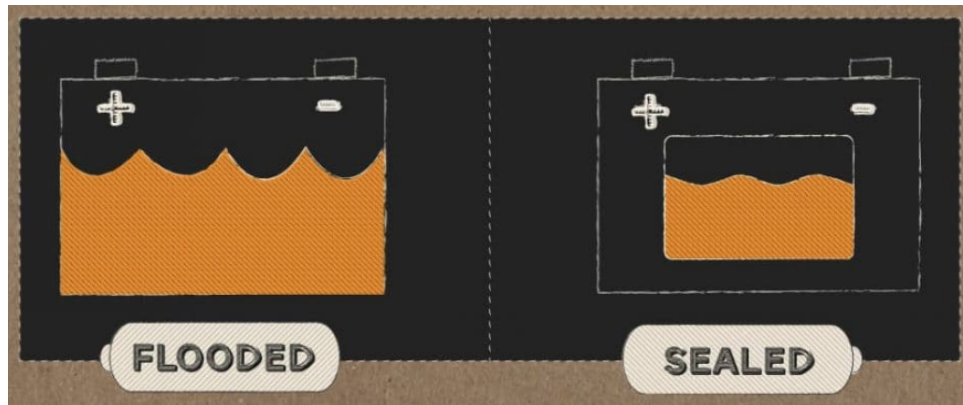
### **3.2.2 Batteries**

The batteries considered for this system are lead acid and Lithium-Ion batteries. There are two types of lead acid batteries, flooded lead acid and sealed lead acid. Flooded lead acid, shown in Figure 18, allow hydrogen fluid and oxygen gas to escape during charging. These batteries last between five and seven years and require more maintenance than sealed lead acid batteries. Flooded lead acid batteries can receive excessive charging periodically to better “equalize” the battery cells so any cells that might not be doing well can be brought to a full state of charge on a regular basis [24]. These have a low upfront cost but require lots of maintenance in the form of checking the distilled water level and refilling it every month to keep it topped off [25]. They also expel built up hydrogen gas and need to be installed in a vented enclosure.

Sealed lead acid batteries are considered maintenance-free because the absorbent glass mat (AGM) or Gel prevents and they do not need water. The AGM and Gel keep the electrolyte mixed with the water so it does not settle to the bottom, which eliminates the need for equalization charging and helps mix the electrolyte with the water in flooded batteries [24]. If the charging or discharging is high enough, the pressure will build up and the valve will allow some gas to escape, eliminating the need for a vented enclosure. These batteries have a higher upfront cost than flooded lead acid batteries and have a lifespan of three to five years.

Lithium-Ion batteries have a longer life-span than sealed lead acid batteries of up to fifteen years. They are lighter and more compact than the lead acid batteries, making them easier to move and store. These batteries are more expensive than the lead acid batteries but they are more stable, safer, and maintenance free [27]. Lithium-Ion batteries also have a higher charge and discharge efficiency and don't lose much capacity when idle, which is good for a system where the energy is not constantly in use.





***Figure 18: The distilled water of the sealed lead acid battery on the right is in a compartment within the battery itself, as opposed to the flooded lead acid battery on the left. The sealed lead acid battery has no access to that compartment, meaning distilled water does not need to be added and the battery is therefore maintenance free [26].***

Both pulse width modulation (PWM) and maximum power point tracking (MPPT) charge controllers were considered for this system. MPPT controllers offer a potential 30% increase in charging efficiency compared to PWM controllers. These controllers also offer the potential ability of having an array with higher input voltage than the battery bank and are offered in sizes up to 80A [28]. MPPT controller warranties are typically longer than PWM units and offer flexibility for system growth. MPPT is the only way to regulate grid connected modules for battery charging. These controllers are more expensive, sometimes costing twice as much as a PWM controller and are larger in physical size.

PWM controllers have been used for years in solar systems, and are well established and inexpensive, usually selling for less than \$350. These charge controllers are available in sizes up to 60 Amps and are durable with passive heat sink style cooling [28]. The solar input nominal voltage of the PWM controllers must match the battery bank nominal voltage. There is no single controller sized over 60 amps direct current (DC) as of yet and many smaller PWM controller units are not UL listed. Many of the smaller PWM controller units come without fittings for conduit and have limited capacity for system growth. They also cannot be used on higher voltage grid connect modules [28].

### 3.2.3 WPT

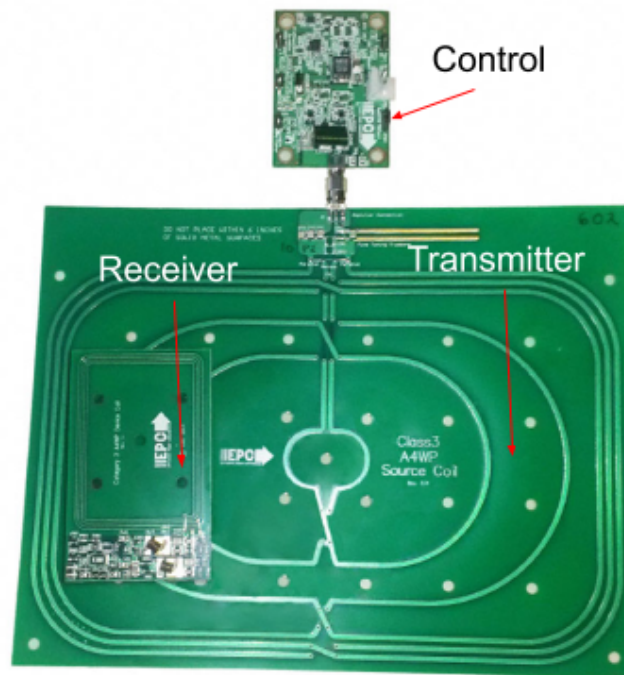
The most widespread use for WPT is in wireless charging for mobile and healthcare devices, which utilizes inductive coupling or magnetic resonance to transfer usually less than 15W of power over distances of only a few millimeters [5]. Figure 19 represents a typical wireless charging pad for a mobile phone.



*Figure 19: The power transmitter on the wireless phone charging pad is located in the pad below the phone, while the power receiver is located within the phone itself.*

These small, low power applications that use induction typically fall under the use of the “Qi” standard. The Qi standard (pronounced CHEE) is a standard created by the Wireless Power Consortium (WPC) in 2008 to govern the use of low wireless power applications [29]. Wireless charging and transfer of power at less than 15W fall under Qi’s Power Class 0. Since WPT is still a relatively new technology and under development, power class 0 encompasses the majority of all wireless power applications [29]. The Qi standard also has power classes 1 and 2, which serve medium (15 - 200W) and high (>200W) power applications, respectively [30]. Power classes 1 and 2 are still under development, and the applications for higher power transfer are currently limited [30]. A similar, but much newer, standard is called Rezence, which is being developed by the Alliance For Wireless Power (A4WP). Rezence utilizes magnetic resonance

technology, which is similar to induction, but the transmitting and receiving coils are tuned with a capacitor to their natural resonant frequencies [31]. This allows magnetic resonance coupled systems to have a greater range and area of effect. Manufacturers using Rezenze often adopt a “1-to-many ” design, where a transmitter coil tuned to the correct resonant frequency can transfer power to multiple receiving coils. A Rezenze magnetic resonance charging mat is shown in Figure 20.



***Figure 20: The transmitter is the large mat in the back, and the receiver is the smaller phone-sized board on top of the mat. There is control circuitry pictured above both pieces [32].***

Users of Rezenze standardized chargers are able to place their devices anywhere on this mat to receive power from the transmitter, unlike magnetic induction where the coils have to be closely coupled to receive power [31]. Qi and Rezenze compatible devices are popular because their operation and designs are standardized. The system for this project has a specific set of requirements that are not satisfied by one standard alone, so the solution adopts principles from both standards to achieve the best solution for this application. This system needs high levels of power transferred, from 100W to 200W, so something under Qi power class 1 might have

worked. However, a forgiving range and tolerance for misalignment was desired, so magnetic resonance technology from Rezence might have also been advantageous. Fortunately, there are other manufacturers out there working on similar solutions as this project and will continue to refine the technology for application in similar systems.

### 3.2.4 Display

LED Matrices are one of the simplest and most flexible of the display choices. This display is an array of LEDs that are connected in series and controlled by a driver microcontroller that will set the color, brightness, select which LEDs to turn on, *etc* [33]. One product focused on was the Flexible LED Matrix created by Sparkfun [34]. This LED Matrix uses individually addressable LED strips that are daisy chained together so instructions can be sent through the entire matrix. It specifically used WS2812B LED, which is an LED strip that has individually addressable LEDs [16]. Individually addressable LED strips allow specific LEDs to be controlled to create pictures, letters, and characters. If they were not individually addressable, the entire strip would only display a single color at a time.

A LCD is a type of display that utilizes the properties of liquid crystals to create visual changes [35]. When the white light is first emitted, it is distributed, diffused, and focused so that it is evenly distributed throughout the screen, preventing “hot spots” from appearing . Next, the light is polarized to only allow light with vertical wavelengths to pass through. This light is then passed through the liquid crystals. The crystals are managed using controllable voltage levels within the thin film transistor and common electrode. Depending on the voltage levels, the liquid crystals will filter primary colors (red, green and blue) within the light. The light is then polarized again so only horizontally polarized light is transmitted [35].

An OLED is a type of display that generally uses the same technology and structure as an LCD screen. However, instead of liquid crystals and white LEDs, an OLED uses organic material that will emit light as electricity is applied to it [36, 37]. That light is then filtered using the same techniques in an LCD screen producing a screen with higher quality colors and images [37].

An electronic paper display (EPD) uses E-Ink to create changes within the screen. E-Ink refers to the positively or negatively charged pigments suspended within microcapsules that are filled with a clear liquid [38]. Using the ink’s property of having a charge, an EPD can move one

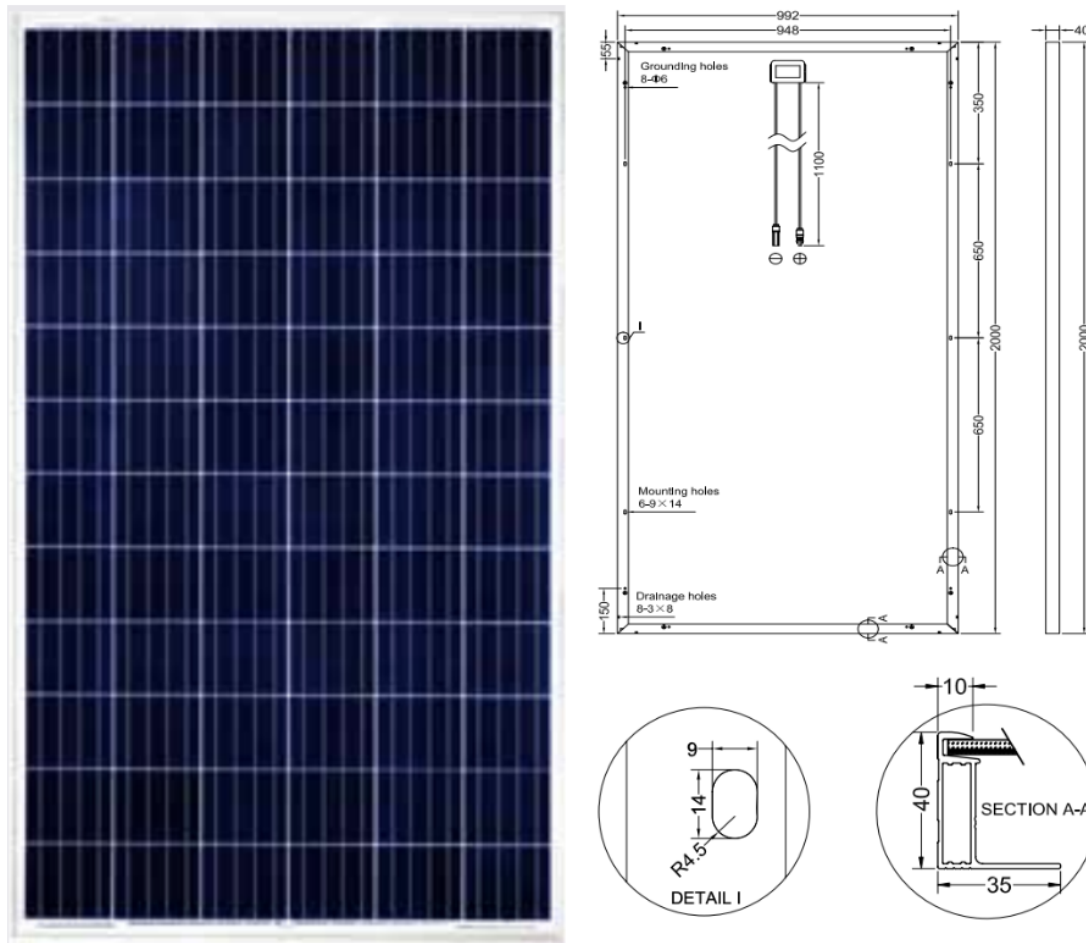
ink color to the top of the capsule, which pushes the other ink to the bottom of the capsule and allows the ink on the top to be seen. Another option is for it to push the ink to the side to create a transparent display [38]. The screen will not be transparent as a result of the second option.

## **3.3 Final Selection of Design**

### **3.3.1 Solar System**

Given the variety of solar technologies that are on the market today, it was necessary to narrow down the options for this project. The ideal module is a monocrystalline solar panel because of their high efficiency and power outputs. However, the team was only able to acquire a polycrystalline solar panel. While the monocrystalline solar panel would have been the first choice for this project, there are advantages to the polycrystalline modules. Over the years, the gap between the efficiencies of monocrystalline and polycrystalline solar panels has decreased through improvements that increased their efficiency. They are also less expensive because the process of creating polycrystalline silicon is not as complex as monocrystalline. They also have a higher temperature coefficient so the output decrease in higher temperatures is lower than monocrystalline modules.

NationalGrid generously donated a 305W Yingli Solar 72 cell polycrystalline solar panel to this project. The front cover of this panel is made of low-iron tempered glass and rated for a peak efficiency of 15.64%. The frame is made of anodized aluminum alloy, which has a high tensile strength and corrosion resistance. The voltages at maximum power and open circuit are 36.1V and 45.4V respectively, while the currents at maximum power and short circuit are 8.45A and 8.93A. This Yingli Solar panel has a nominal operating cell temperature (NOCT) of 46°C. Its physical dimensions are 39 inches by 77.6 inches and weighs about 60 pounds, as shown in Figure 21.



*Figure 21: The panel is 39 inches by 79 inches with a frame width of about 1.5 inches. There are also grounding and mounting holes around the perimeter of the frame*

### 3.3.2 Batteries

The best type of battery for this project is the sealed lead acid AGM battery. These batteries are overall the most cost efficient because although they have a higher upfront cost, they are maintenance free. They do not need to be installed in any specific enclosures and have an adequate charge and discharge rate for the project.

The battery used in this project is the DCM0075 model battery from Interstate Batteries, pictured in Figure 22. Being a sealed lead acid AGM battery, it is maintenance free, spill-proof and has no exposure to hazardous chemicals. This battery is optimized for high-power density, extended cycle life, and flame arresting for safety and long life [39].



*Figure 22: This is a 75-Amp Hour, 12-Volt, AGM battery. Its deep cycle design works with most backup systems and has an efficient glass recombination of up to 99% [39].*

A PWM charge controller was chosen as the ideal charge controller for this project. After looking at the system as a whole, it was decided that the maximum power transferrable is about 150W, or only half of the panel's ability. The losses that would be occurring in that regard render the energy losses with a PWM charge controller inconsequential.

The model chosen for this project is a PWM 30A Charge Controller by HQST [40]. It was designed for off-grid applications and is compatible with both 12V and 24V battery banks, both of which are used in this system. This charge controller also comes with comprehensive self-diagnostics and electronic protection functions to prevent damage from system faults. Safe grounding and compatibility with any negative ground system is also ensured by its negative-ground battery controller. This charge controller is shown below in Figure 23.



*Figure 23: The charge controller features protection against several battery life shortening aspects such as battery short-circuit, over-current, overload, and overheating. Any system issues can be easily diagnosed with assistance from the system error codes and information that display on the LCD screen [40].*

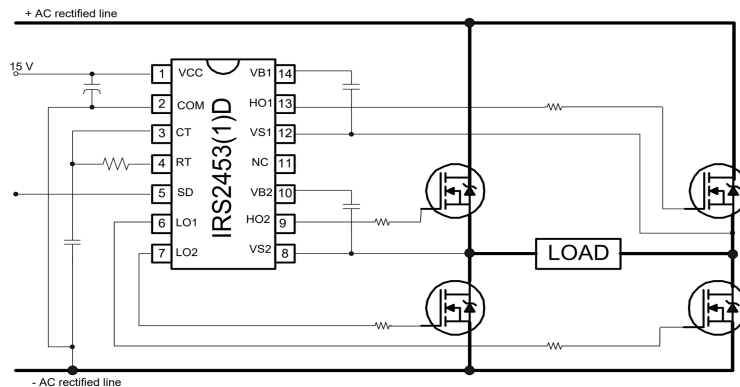
A 300W, 20A DC step down buck converter by Aideepen was chosen for this project because it can be used as a voltage regulator for solar panels and also as a driver for high power LED lights. The operation of this buck converter is further discussed in Chapter 6.

### 3.3.3 WPT

Based on the three part design of a WPT system outlined previously, the creation of a unique WPT design was attempted for this project. The design would feature all elements previously described: an oscillator, inverter, two coils with resonance compensation capacitors, and a rectifier. The first step in the design process was to choose the oscillator and then design the full inverter. The initial design choice was to use inductive coupling technology over magnetic resonance because there was a desire for the highest possible efficiency without coil alignment compensation using resonance. Inductive coupling WPT systems using the Qi standard operate between 87 kHz and 205 kHz, so an oscillator that could create a square wave at a frequency in this range was necessary. A simple option would have been to use the popular 555

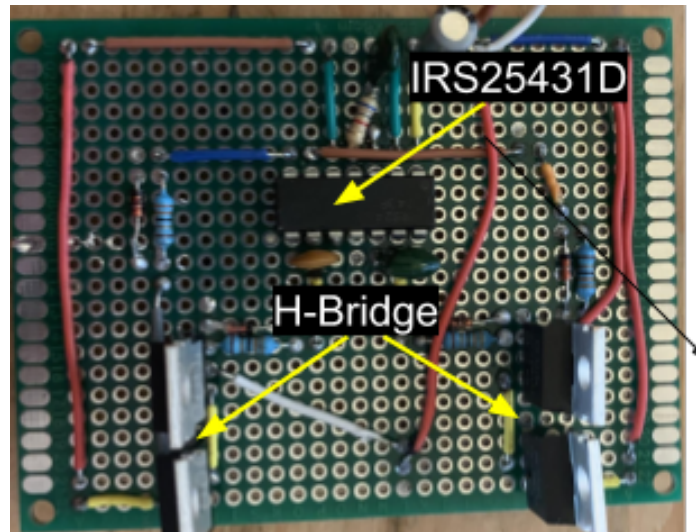


timer or similar. It was also decided that an H-Bridge would be used, which is a common circuit in inverter technology in which 4 field effect transistors (FETs) are driven by a driver circuit and can sink large currents that an oscillator cannot. Digikey was consulted to find a suitable oscillator IC and H-Bridge IC for the application. Fortunately, the IRS24531D, which is an H-bridge driver IC with a built in oscillator, meant that this solution could accomplish with one chip what would normally require two circuits. The typical application circuit for the IRS24531D is shown in Figure 24.



**Figure 24: The IRS24531DS full bridge gate driver with internal oscillator is connected to four N-Channel MOSFETS. All components together create a basic inverter [41].**

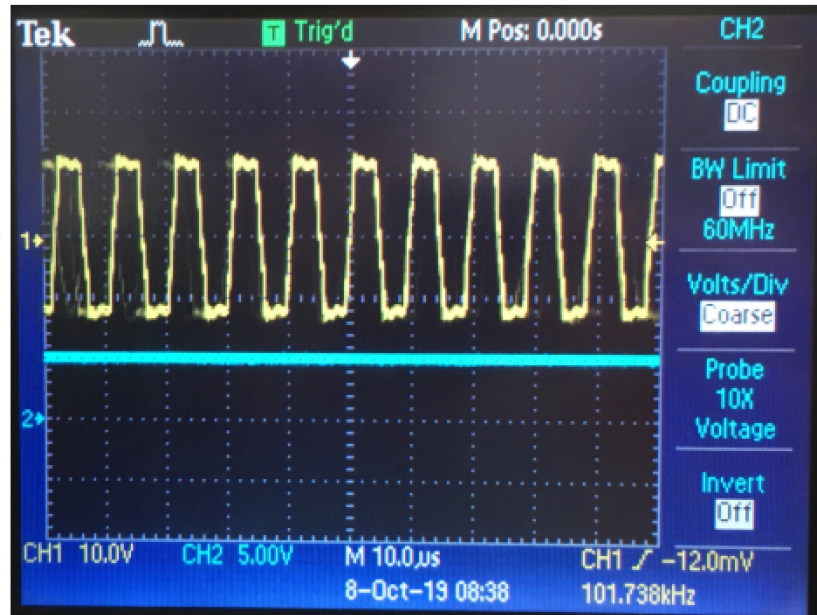
Using the Qi standard and the IRS24531DS datasheet as guides, 100kHz was selected as the operating frequency. A 10nF capacitor was selected for the built-in oscillator and using the equation  $f \approx \frac{1}{1.453 * R_T * C_T}$ , where  $R_T$  is the timing resistor and  $C_T$  is the timing capacitor, the ideal resistor value worked out to be 6.8k $\Omega$ . The series gate resistors were 33 $\Omega$  and the high-side decoupling capacitors were 1 $\mu$ F. The IRF520N N-channel enhancement mode MOSFET was chosen for the FETs that form the H-bridge, particularly for its high current sinking capability and general popularity. The circuit, shown in Figure 25, was assembled and soldered onto a protoboard for testing.



*Figure 25: The IRS25431D is the 14-DIP chip in on the top. The four MOSFETs forming the H-bridge are shown at the bottom. Supporting components were chosen following guidelines set in the IRS25431D datasheet.*

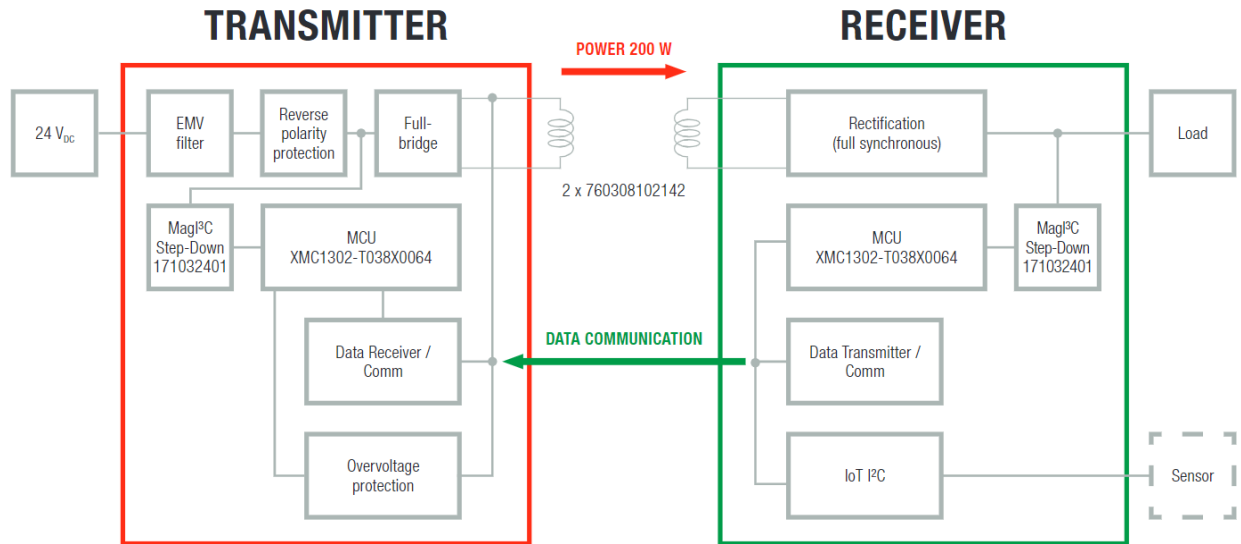
Additionally, a very basic rectifying and regulating circuit was built using 1N4005 diodes [42], a LM7805 5V regulator [43], and an LED to load the receiving end of the transfer system. An oscilloscope photo of the testing is shown in Figure 26, which looked promising at first. Channel 1 shows the output of the inverter and while the rise and fall times of the square wave were slightly slower than desired, the operating frequency was right in the expected range. Channel 2 shows the output of the LM7805 prior to loading the system with the LED, so the output looked smooth and ideal for a first test. However, after loading the system, it could not maintain the efficiency required to meet the dropout voltage to the regulator.

**Output waveform of custom WPT circuit**



*Figure 26: Channel 1 shows the output of the inverter and Channel 2 shows the output of the LM78-5 regulator on the receiving end.*

Initial tests made it clear that this system would be difficult to build, test, and integrate into the rest of the project in the allotted amount of time without considerable support. A 200W wireless power development kit was found during the research process, made by Würth Elektronik in partnership with Infineon [44]. This kit and its available schematics and specifications were inquired after to try and determine if something similar could be replicated in the system. A block diagram of their system is shown in Figure 27.



**Figure 27: This diagram outlines the most important elements of the Würth Elektronik 200W Medium Power Extended Solution Kit [44].**

There are many additional control elements in addition to the traditional elements of a WPT system, making the design complex. An onboard XMC1302 series microcontroller monitors the voltage and current levels during transfer and adjusts the frequency of operation between 110 kHz and 150 kHz to attain maximum power transfer at a given distance between coils [44]. Additionally, circuitry is added to modulate the current flow through the coils to create unidirectional data communication from receiver to transmitter for greater control [44]. The design also includes several protection elements to ensure the design will withstand electromagnetic interference (EMI) and overvoltage/overcurrent situations. This kit does not contain foreign object detection, which should not be needed for the system as it will be fixed and inaccessible. Unfortunately, developing a system of the same quality under the budget and time constraints of this project is not feasible. Fortunately, Würth Elektronik generously donated one of the development kits to the project free of charge. The retail value of this all-in-one development kit is \$350.00. The kit includes an all-in-one transmitter, all-in-one receiver, 24V DC power supply with international adapters, and 9W, 220Ω load resistor. The final system currently utilizes the existing WPT solution to create a power delivery system for the display.

### 3.3.4 Display

There were two major concerns in the display selection process. The power draw from the display could not exceed the power supply, and at the time there were also cost concerns about how the energy from the solar panel would be stored, generate the WPT unit, and set up the solar panel array because each of these processes required funding.

The LCD and LED screens were disregarded in regards to the power concern because they had a high power consumption over time. These displays were defined with high power consumption because they required more power than initially calculated. At the time it was estimated that, at best, 30W would be able to be supplied to the display, but both the LED and LCD screen required a supply of 34W. As a result, both of these options were not selected.

The OLED and E-Ink displays were also disregarded after further budget consideration. Both of these displays were unaffordable given the budget. An OLED display would cost around \$1000 according to the research, but an accurate cost was unable to be obtained because most OLED displays were unavailable in the US. The E-Ink display was also extremely expensive at \$2500, excluding shipping and handling. As a result, these two options were also disregarded.

Ruling out the previous four options flushed out an ideal solution, the handmade LED matrix display. Not only was this option the most affordable at about \$100, it also had a low power consumption of about 11W. Since the device was hand-made, features could also be added to the display such as transparency and modularity. Additionally, a handmade display would add a more technical aspect to this educational project.

The display consists of nine modules, each containing 100 LEDs, three PCBs, and a single 3D printed structure to hold the LED strips. The display is controlled by a single Raspberry Pi microprocessor. The estimated total cost of all components for a single module is shown in Table 1.

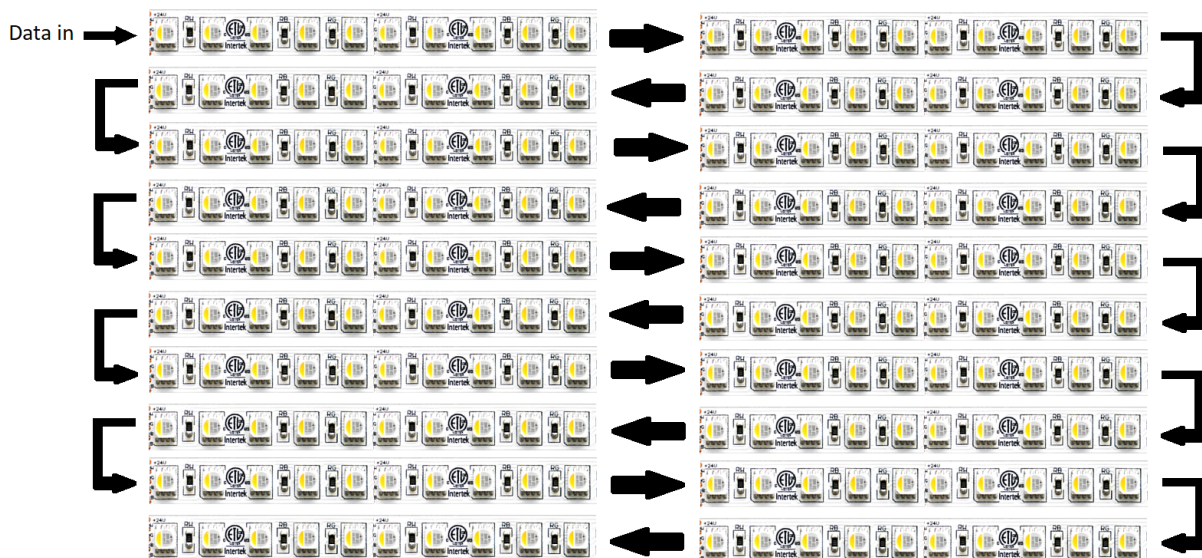
**Table 1: Display System Cost**

<b>Part</b>	<b>Cost</b>
LEDs	\$130.00
Microprocessor	\$35.00
PCBs and wires	\$40.00
Structure	\$10.00
<b>Total</b>	<b>\$215.00</b>

The 100 LEDs in each module are arranged in a 10 by 10 square. This is created by cutting 10 strips of 10 LEDs and attaching them to the structure, spacing them evenly to create a square grid. Two PCBs on each end of the strips connect the LED strips together in a chain to form a continuous 100 LED daisy chain. Connectors on the underside of each of these PCBs allow the modules to be connected together such that two modules can form a chain of 200 LEDs, 3 modules can form 300 LEDs, and so on. This modular design allows many modules to be connected continuously for multiple different display sizes. When multiple displays are connected continuously, one of the terminating displays must be connected to the microprocessor with the Right Cap PCB while the other terminating display's connector must have the Left Cap PCB which, along with the Right Cap PCB, connects all rows of LEDs together to form the chain. A graphical description of this design is shown in Figure 28 and Figure 29.



*Figure 28: The power is connected in parallel while data and clock lines are connected in series as shown by the arrows.*



*Figure 29: When multiple modules are connected, the rows are expanded and data and clock lines are connected to the consecutive row at the terminating modules.*

The proposed display consisted of nine modules for a total of 900 LEDs. The LED chosen was the SK9822, an RGB addressable LED which is relatively affordable and is available in high density strip form, which is ideal for the design [17]. The SK9822 utilizes 5V input and multiple LEDs are connected to 5V and ground in parallel on the strip. The SK9822 has two inputs, data and clock. These signals must be provided externally. The SK9822 also outputs these two signals such that the data and clock are connected to all LEDs in the chain in series. The results of the current draw testing of the SK9822 are detailed in Table 2.

*Table 2: Current Draw of LEDs With Red Color*

LEDs	Brightness		
	10%	50%	100%
100	0.25A	0.46A	0.71A
150	0.28A	0.58A	0.96A
200	0.32A	0.71A	1.20A

Each module has a Left Main and Right Main PCB. These PCBs bundle the left and right ends of the LED strips into connectors that allow strips from different modules to be connected together. The completed display also has a single Left Cap PCB that connects the left ends of all LED rows together and a Right Cap PCB that connects all the right ends together as well as connects the start of the chain to the microprocessor for data and clock input. The underlying structure for each module is a 3D printed plastic frame. This frame is custom designed to have 10 slots for each LED strip as well as slots on each side for the Left Main and Right Main PCBs. Each frame has 4 screw holes that serve the purpose of securing the PCBs as well as the structural connectors that fasten modules together.

### 3.4 Project Logistics

The main goals of the first eight weeks were defining the final system, researching the components, designing the system, and finding sponsors or donations to support the project. The biggest difficulty during this time was the deficit in the budget for building the system. Due to this issue, buying small components for each system to try and build a simple prototype of the system and searching for sponsors or donors was the objective. Near the end of A term, Justin



Woodard agreed to become a sponsor for the project and promised to donate any needed parts for the power component of the system. Thanks to his help, the budget required to build the system was reduced and progress for designing the full scale system could continue.

The second eight weeks were dedicated to finishing the project's design for each system component, switching to a modular display, ordering the parts, and beginning prototyping and debugging for each piece. During this period, separate teams were made to focus on different aspects of the project. Each team focused on either the system's power component, software component, power transfer component, or the display component. The main issue in this period was determining whether WPT was feasible or not. Eventually, Würth Elektronik donated a WPT kit to the project that met the requirements for this system. However, this setback caused constraints on the time needed to begin debugging the prototype system as a whole.

The main goals of the final eight weeks of the project were building and debugging the system, writing the report, and demonstrating the system. During this period the smaller teams were combined to work together on integrating the entire system as a group. One major setback in this period was when it was discovered that there were shorts in all of the initial PCBs and they needed to be redesigned. Additionally, due to an international health crisis, the delivery time for these parts took two weeks increasing the build time and reducing the testing and debugging time for the display by two weeks. Another setback was that the WPT unit's supplementary circuit was not working as intended and was not fixed until the final few days of this period. The foremost setback was that the display modules were having solder joint issues due to the joints sporadically shorting to one another or were not properly connected. In spite of these various setbacks, all goals were achieved albeit a week later than expected.

### **3.5 Chapter Summary**

Devices and components were chosen after careful evaluation of design options for each part of the system. Some compromises had to be made to conserve time and money, such as choosing a polycrystalline panel over a monocrystalline panel based on what the team had access to, or using a development kit for wireless power instead of creating a custom one to save time. Batteries and charge controllers were chosen based on the needs of the other parts of the system. It was paramount to consider the needs of each system segment throughout the process while considering all the design options available.

## **Chapter 4: Methodology and Implementation Display**

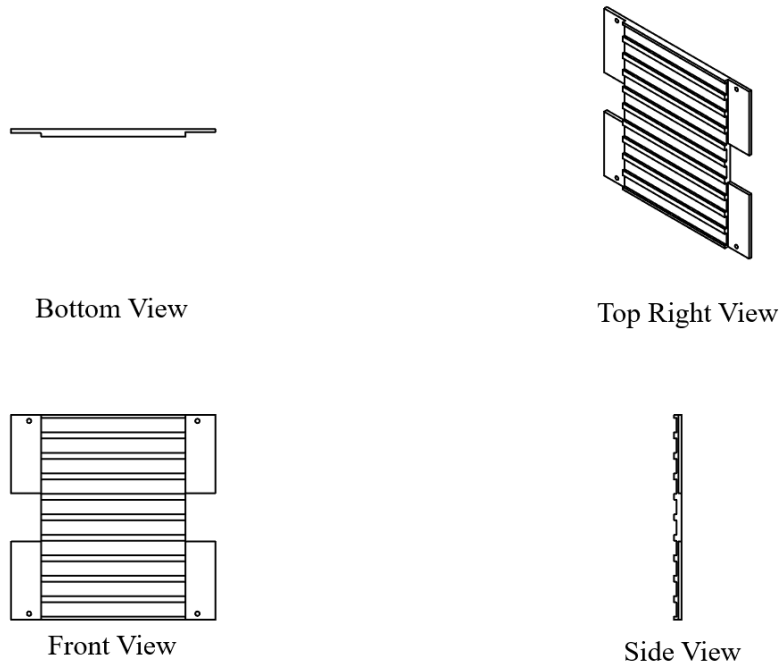
The overall development of the display required specific qualities to ensure it met the design goal. To create a fully modular display, each panel of the display had to be identical and the LEDs needed to be programmable given different height and widths. The design for a modular panel had multiple iterations including one large PCB, milled acrylic with PCBs on the back side, or small PCBs with 3D printing. Each idea was discussed until a final design was determined.

The module was broken up into the following 4 components: main frame, brackets, screws, and PCB's. The main frame is where all of the materials would be connected and hold the major parts of the project together in a neat and organized design while the brackets hold the modules together. There were many iterations and tests of different screws to find ones that met the specific criteria for the design.

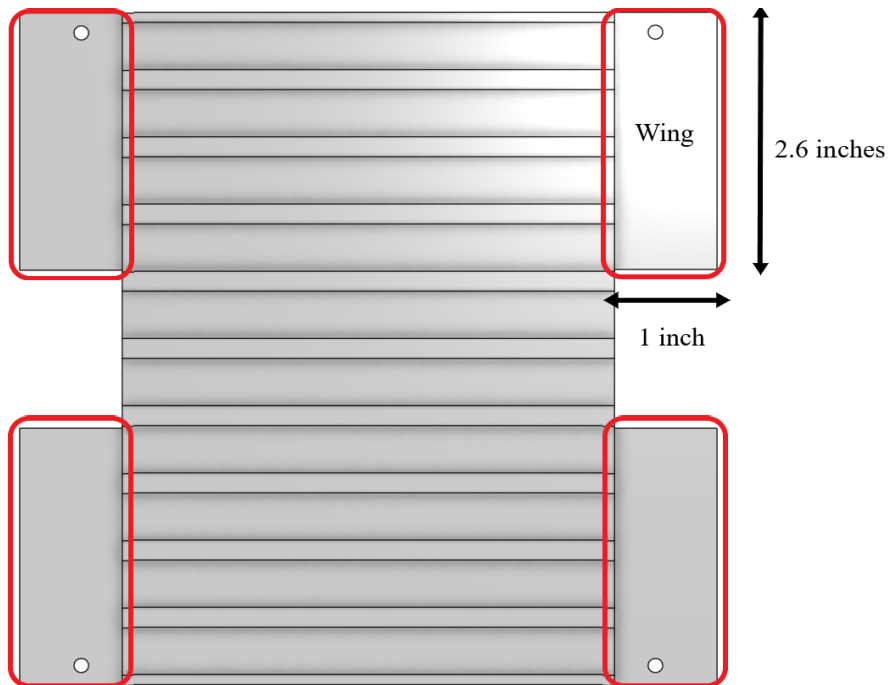
### **4.1 Main Frame**

As previously stated, the main frame was designed to hold major components together so the LED matrix would look neat and organized. The main frame was displayed from various angles in Figure 30. The dimensions of the module were about 6.7 inches tall, 6.7 inches wide, and had a maximum thickness of 0.25 inches. The module had several features for specific reasons, as can be seen in Figure 30. Those features were: wings, channels, uneven wing heights, and screw holes as seen in Figures 31, 32, 33, and 34.

The purpose of the wings in Figure 31 was to hold the PCBs that were going to be mounted onto the module. These PCBs were going to be one inch wide and 6.7 inches long, are the exact dimensions that were prepared on the module. The purpose of the opening in between the top and bottom wings was for connections and other wiring. The purpose of these connections was for creating modularity, which will be explained in Chapter 4.4 .



*Figure 30: Four different viewing angles of the LED module to provide an idea of how the module is shaped.*



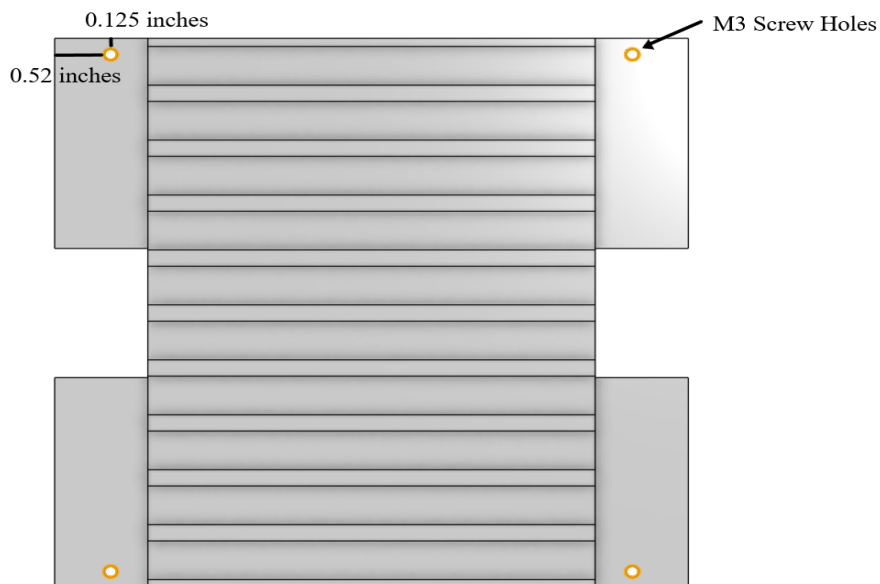
*Figure 31: Figure of the final module design with the wings highlighted and labeled with their dimensions. Each wing is 2.6 inches long and 1 inch wide.*

The uneven heights on the left and right wings that can be seen in Figure 32 is because of a small error made while ordering the PCB's. Some of the PCBs have shorter thicknesses than intended. The designs on the wings were changed so that the PCB's thickness would line up with the bottom of the channels to make up for the misalignment.



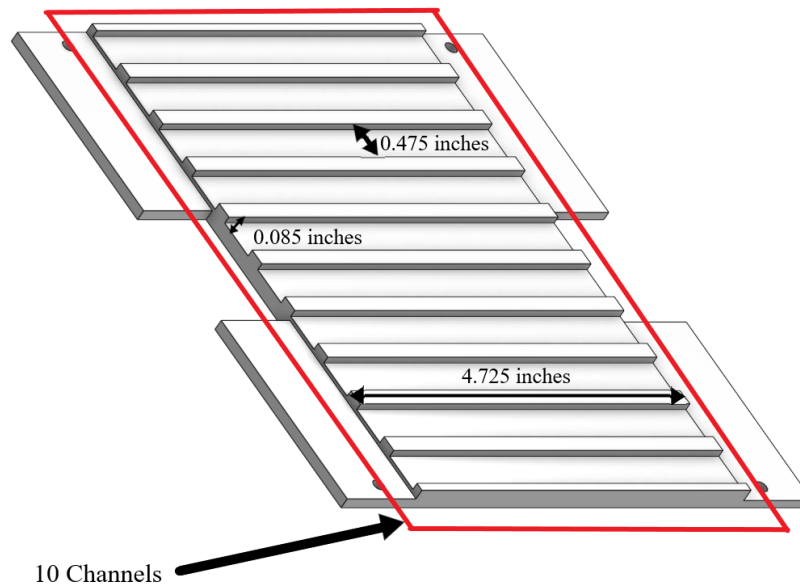
**Figure 32: This figure shows the bottom of the module with measurements of the heights of both wings and a line to compare the difference in the heights of the wings. The right wings have a shorter height of 0.1 inches while the left wings have a height of 0.125 inches.**

The screw holes in Figure 33 are necessary to mount the PCB's to the module and guarantee that they would not separate or detach from it. The screw holes cannot be closer to the center because they need to be kept away from the internal circuitry of the PCB's.



**Figure 33: This figure highlights the screw holes and their measurements as well as the type of screw hole. Each screw hole is made for an M3 size screw and is 0.52 inches from either the side of the module and 0.125 inches from the top or bottom of the module.**

The channels seen in Figure 34 are designed to hold and align the LED strips so the lights look uniform, neat and evenly spread light. The channels are 0.09 inches tall from the bottom of the channel to the top and they are 4.725 inches wide and 0.2 inches wide except at the edges where they are only 0.1 inches wide.

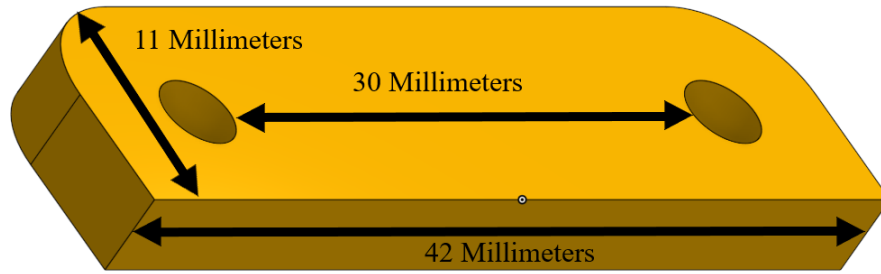


**Figure 34: LED module with all the channels highlighted and dimensions labeled. The channels are 0.475 inches wide, 0.085 inches tall, and 4.725 inches long.**

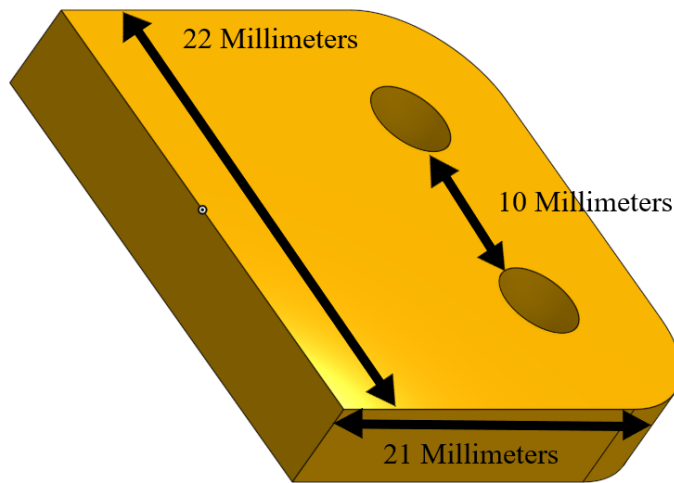
There were difficulties in the design process, but after trying to 3D print out the module it was decided to keep printing them out because it was relatively cheap and easier to create by using a 3D printer. Overall, by using the semi clear PETG, modules were able to be generated at around \$1.67 per module.

## 4.2 Brackets

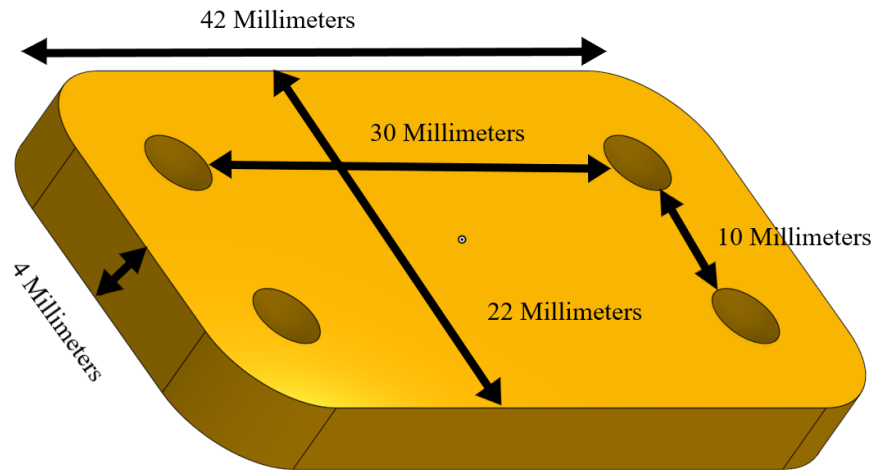
The brackets are designed to hold the module together and prevent misalignment. A total of three different brackets were created for this reason. Figures 35, 36, and 37 show the vertical edge connections, horizontal edge connections, and center connections that were created. The three connections are necessary because the dimensions to connect vertically and horizontally were different.



*Figure 35: All vertical edge connecting bracket dimensions are labeled in this figure. The bracket is 11mm wide, 4mm inches thick, and 42mm long, with the distance between holes being 30mm.*



*Figure 36: The horizontal edge connecting bracket dimensions are 22mm wide, 4mm inches thick, and 21mm long, with the distance between holes being 10mm.*

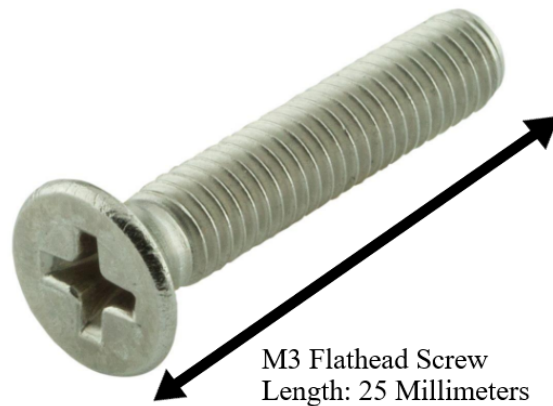


***Figure 37: All center connecting bracket dimensions are labeled in this figure. The bracket is 22mm wide, 4mm inches thick, 42mm long. The distance between holes is 10mm vertically and 30mm horizontally.***

Some difficulties encountered were due to constraints with the PCBs because the original intent was to have three screws per corner to ensure the strength of the connections and to prevent any “wobble room” for the parts, which was infeasible. Additionally, more screws increase the overall cost of the system and as a result, the final design has one screw at each corner of the modules.

### **4.3 Screws**

The purpose of the screws is to allow the PCB to be mounted onto the module and provide something to latch to for the bracket. Flat headed M3 screws were used because they have the thinnest heads, allowing the LEDs that are placed over them to remain as flat as possible. The screws are about an inch long so that they can be used to mount behind the module without putting pressure on the wiring behind the module.



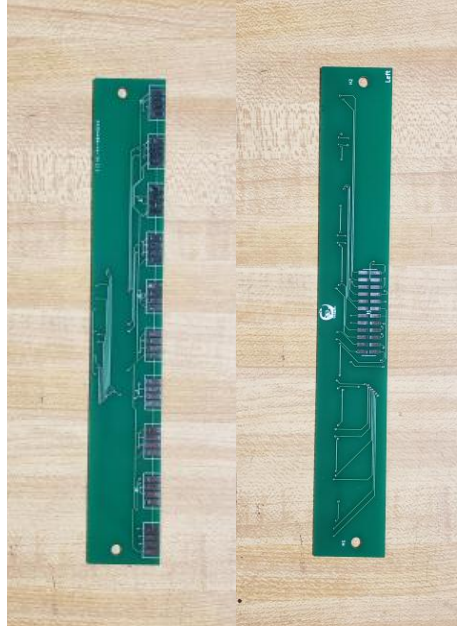
*Figure 38: This figure displays the flat-head machine screws that were selected to mount the PCBs. It is a 25mm long M3 flathead screw [45].*

Finding a type of screw that does not stick up too far from the PCBs was difficult because the LEDs were intended to lay on top of them. The ideal screws are ultra-thin M3 screws but each one costs about a dollar. The less expensive option of flat head screws was chosen that only made the LEDs stick out a couple millimeters.

#### **4.4 Custom Printed Circuit Board**

The final design of the display includes several LED modules that can be connected differently to allow for many topologies and display shapes. Each module consists of 100 LEDs arranged in a square 10 by 10 grid. 10 strips of 10 leds are affixed to a plastic 3D printed surface with channels for each strip so that even spacing is achieved between the strips when assembled. The plastic surface has slots at each end for two PCBs, shown in Figure 39 and Figure 40 [46]. The four solder pads on each end of the LED strips are connected to corresponding pads on the PCB. The purpose of the PCBs are to connect the strips to each other as well as to the connector that is used to connect modules together.





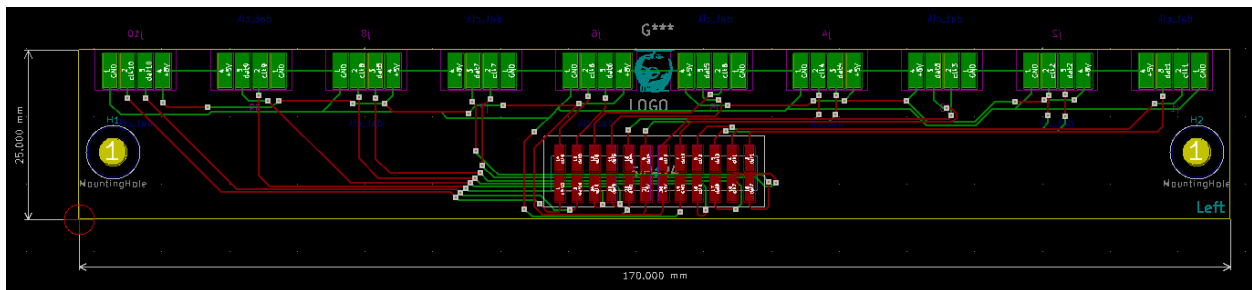
***Figure 39: This figure displays the front and back of the left side PCBs used for the modules. The front is on the left and the right is on the left.***



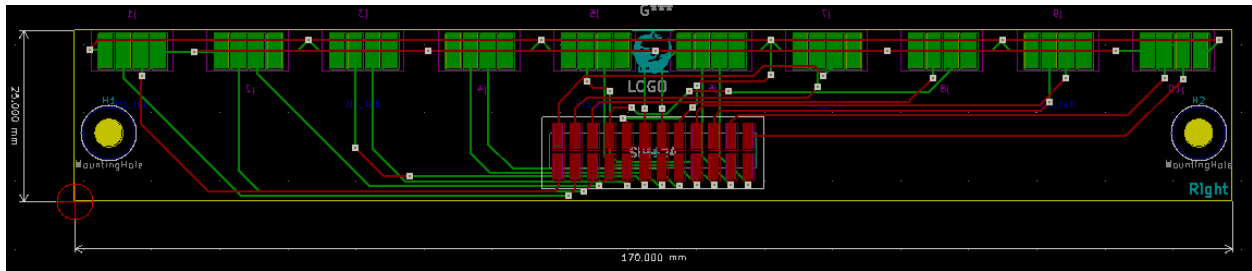
***Figure 40: Right Side PCBs. This figure displays the front and back of the right side PCBs used for the modules. The front is on the left and the right is on the left.***

Four unique PCBs were designed to allow all connections to easily be made and for the system to remain modular. Each module is identical, with two edge PCBs attached to the 3D

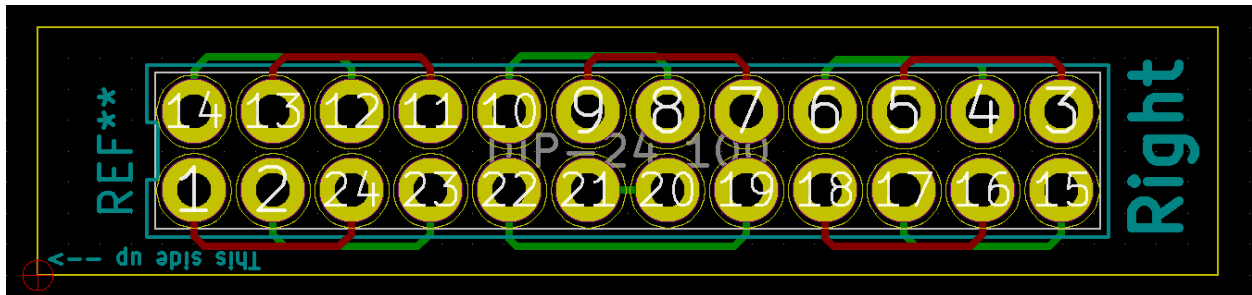
printed backing. The left edge and right edge PCBs interface the LED strips to a 24-pin single surface connector. The signals are passed to one of three connectors, either a cable, a right cap, or a left cap, and each serves a purpose to move the LED signals through the display in a pattern from left to right. The left and right sided PCB diagrams can be seen in Figures 41 and 42, respectively. The small cables, that connect the inner modules together, pass the signals from each LED strip to the LED strip directly across the gap. The right caps are placed on the right side of the rightmost modules to route the LED strip to the next LED below it, allowing the signal to run back across the display from right to left. The design is simple featuring only one component, an input/output 24-pin single surface connector, this PCB is shown in Figure 43. The left caps are placed on the left side of the leftmost modules to route the LED strip to the one below it, allowing the signal to run back across the display from left to right, similar to the right cap. However, in addition to routing the signals from one panel to the other horizontally, the last LED signal is routed downward to the module below it, and/or the first LED signal is passed from the module above it. This PCB is shown in Figure 44. In a display configuration with a vertical component, this allows the signal to be translated to second and subsequent rows of modules.



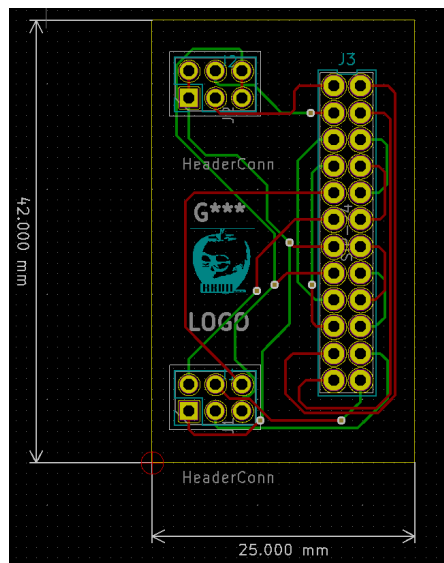
**Figure 41:** The design has the green pads on the leftmost edge of the 3D printed backing on the topside. The red pads are connected to a 24-pin surface mount pin header on the underside.



*Figure 42: The design has the green pads on the rightmost edge of the 3D printed backing on the topside. The red pads are connected to a 24-pin surface mount pin header on the underside.*



*Figure 43: The right cap routes all signals from the right side of the rightmost module back through the module. The single 24-pin sits on the topside of the PCB.*



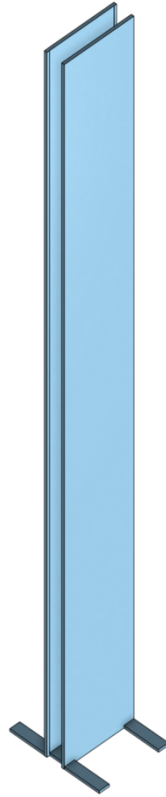
*Figure 44: This PCB sits on the left edge of the leftmost modular. The cap routes signals from the top 6-pin connector through the module via the 24-pin connector, then signals from the module to the bottom 6-pin connector. The top 6-pin connector provides input, and the bottom 6-pin connector is the output. The 6-pin connectors are on the bottom side of the board, and the 24-pin connector is on the topside of the PCB.*

Two issues that appeared in the development and construction of the PCB were having the ordered PCBs from JLC PCB be different thicknesses and having incorrect designs for one of the PCBs. To compensate for the difference in thickness, the 3D printed backing was altered to keep the surface height of the display consistent. Initially, the design for the right edge PCB was incorrect. The PCB was routing the signals from the top row to the bottom, rather than directly across the display. To fix this issue, a new PCB was designed and ordered to properly map the signals from one module to the next.

## **4.5 Complete Module Design**

Each module is constructed by cutting ten strips of ten LEDs and attaching them to the ten channels in the plastic surface using adhesive backing. All PCBs should be assembled before attaching to the module and connecting to the LED strips. Each module has a total of four PCBs that must be assembled. This includes a single surface connector for both left and right PCBs, a through hole connector that must be soldered to the right cap PCB, and three through hole components for the left cap PCB. The PCBs are fixed to each end with screw holes that align in the plastic structure. Each LED strip has four pads on each end that are soldered to corresponding pads on both left and right PCBs. The left and right cap PCBs must be plugged into their corresponding sides on the display module to allow connection to the other modules. Multiple modules can be connected together to create the complete display as a result of how the electrical components were designed. To display these modules correctly, a custom acrylic stand was created that will allow all 9 modules to hang within the casing, shown in Figure 45.

The dimensions for this model were created because all modules, which have a width and length of 6.7 inches and a thickness of 1.25 inches, needed to be considered. Based on these dimensions, the display case needed to encompass at least a volume of 6.7 inches wide, 61 inches long, and 1.25 inches thick. There is a large amount of additional space because at least 2 standoffs at the top and middle of the module were desired.



*Figure 45: This figure depicts the model of the planned LED display case. This stand was made to be 6  $\frac{3}{4}$  inches wide, 72 inches long, and 3 inches thick.*

## **4.6 Chapter Summary**

This chapter outlined the various elements of the implemented display design. Due to the desirable modular characteristic of the display, the implementation had to support modularity in both electronics and mechanical design. In mechanical design, the base of the modules ensured all LEDs would be spaced out evenly vertically and horizontally. The brackets were designed to ensure that the modules would be held flush together and hold the tensile stress of the module weight. On the electrical side, the PCBs were designed so that every clock, data, and power pin on each strip would be translated into a single connector and retranslated into strips again at the other side. To improve the versatility and visual fidelity of the display element, the module stand was designed for the 9 modules to be in a straight vertical configuration.

# Chapter 5: Methodology and Implementation of Software

Chapter 5 describes the steps needed to implement the software of the display. This chapter provides a walkthrough for installing the Raspberry Pi 4 operating system and configuring the necessary settings for use. Additionally, a description of the Python libraries needed to implement the display and an overview of their operation is outlined. In particular, the software libraries needed for the LEDs, the ultrasonic sensor, and the weather API are discussed. Lastly, there are descriptions of the Python scripts used to control the display, with the code provided in appendices.

## 5.1 Operating System

Unlike many microprocessors, the Raspberry Pi 4 is capable of running a myriad of diverse operating systems for a variety of different applications. Images of Raspbian, Ubuntu Mate, Core, and Server; Windows 10 Internet of Things (IoT) Core; PiNet; Mozilla WebThings; and several others are available for use on the Raspberry Pi 4. All Ubuntu operating systems are best for those seeking a less optimized, but more complete version of a Linux operating system, featuring powerful and secure software at the cost of lower performance. Windows 10 IoT Core is best suited for IoT applications, namely, automation and smart home projects controlled via a Windows development environment. Mozilla WebThings is mostly the same but is better suited for a wider variety of devices using different operating systems, whereas PiNet accomplishes the same thing for a classroom environment.

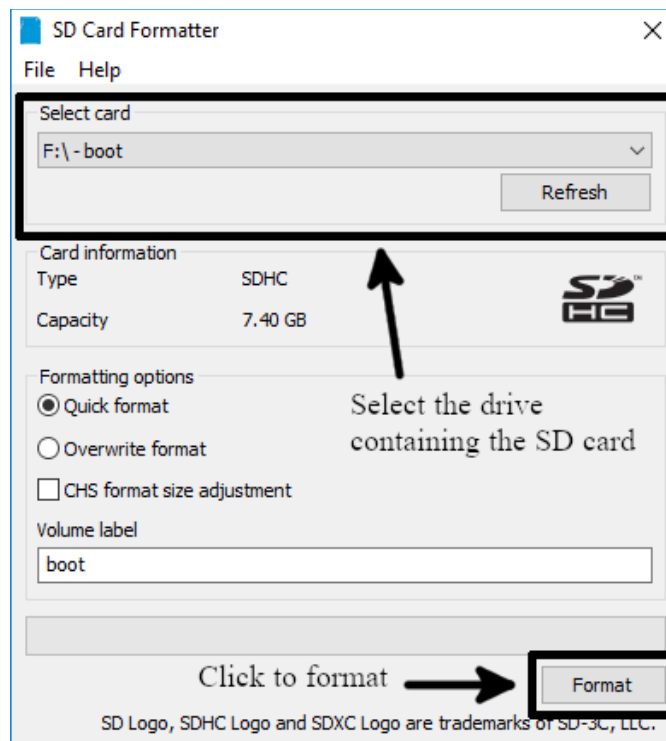
After assessing the multitude of options for potential operating systems to be used in this application, it is decided that Raspbian desktop would be best suited. Raspbian OS is a desktop environment operating system built on Debian Buster, a universal, non-Unix distribution of Linux. The Raspbian distribution of Debian Buster has been modified to make better use of the Raspberry Pi's unique hardware. Raspbian is compatible with tens of thousands of Raspberry Pi specific software packages, ensuring that most libraries used in any given project function correctly. Raspbian comes pre-installed with SPI and I2C support, making interfacing with third party hardware connected to the Raspberry Pi easy and reliable. The Raspbian desktop

environment allows for the use of the Thonny 3 Python Development IDE, assuring Python syntax rules are followed and helps identify bugs in the code. For these reasons, Raspbian Buster with Desktop was chosen as the operating system installed on the Raspberry Pi.

### 5.1.2: Raspbian Installation

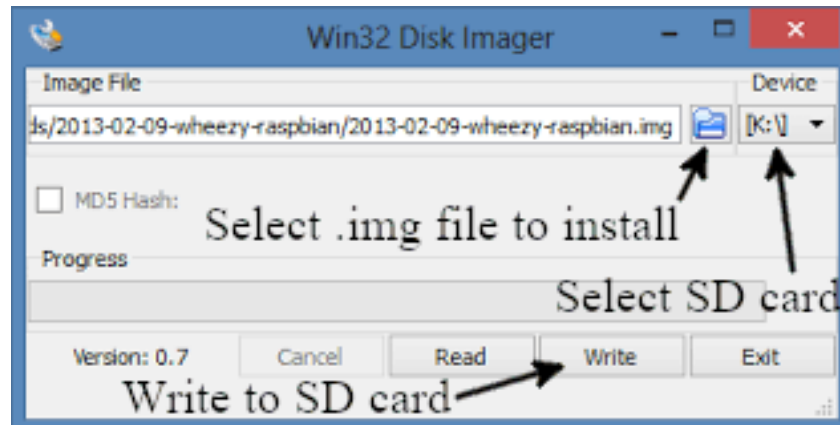
Raspbian Buster with Desktop must be installed onto the Raspberry Pi's internal storage (a micro SD card) before it can be inserted into the Raspberry Pi and booted up for the first time. This is done by inserting the micro SD card into a PC, formatting the SD card using SD Card Formatter, and installing the system image using Win32Disk Imager.

To begin, the Raspbian Buster image must be downloaded from the official Raspberry Pi website [47] as well as the ISO. This will download a .zip file that is extracted upon completion of the download. This extracted file will later be installed as a bootable image on the SD card. However, before the image can be installed the SD card must first be formatted. Using SD Card Formatter, one must select the SD Card drive and wipe it as shown in Figure 46.



*Figure 46: SD card formatter for windows was used for this project. Once installed, one simply needs to select the SD drive and format it with a quick format.*

Once the SD card is formatted, the Raspbian image can be installed on the SD card. Using Win32 Disk Imager, the SD card is selected as a device and the extracted image file is written as shown in Figure 47.



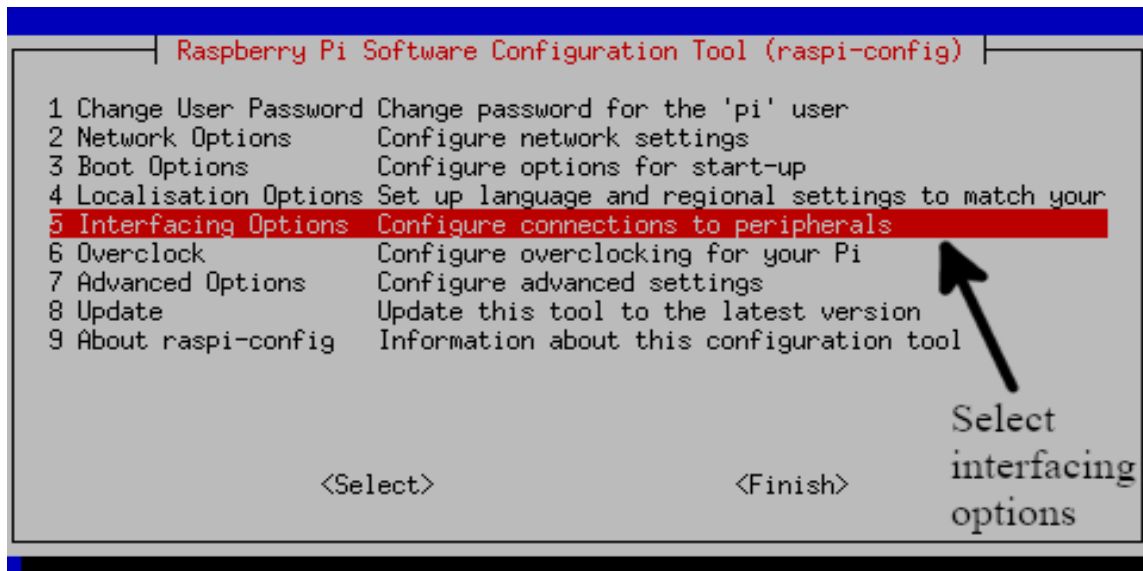
*Figure 47: Using Win32 Disk Imager, one can create a bootable SD card for the Raspberry Pi. Once installed, simply select the unzipped .img file and select the drive containing the SD card. Then, select “write” to create the bootable drive.*

Once this has been completed, the SD card can be removed from the PC and inserted into the Raspberry Pi. With the SD card installed, the Raspberry Pi should boot into the Raspbian Buster Desktop environment.

### 5.1.3 Raspbian Setup

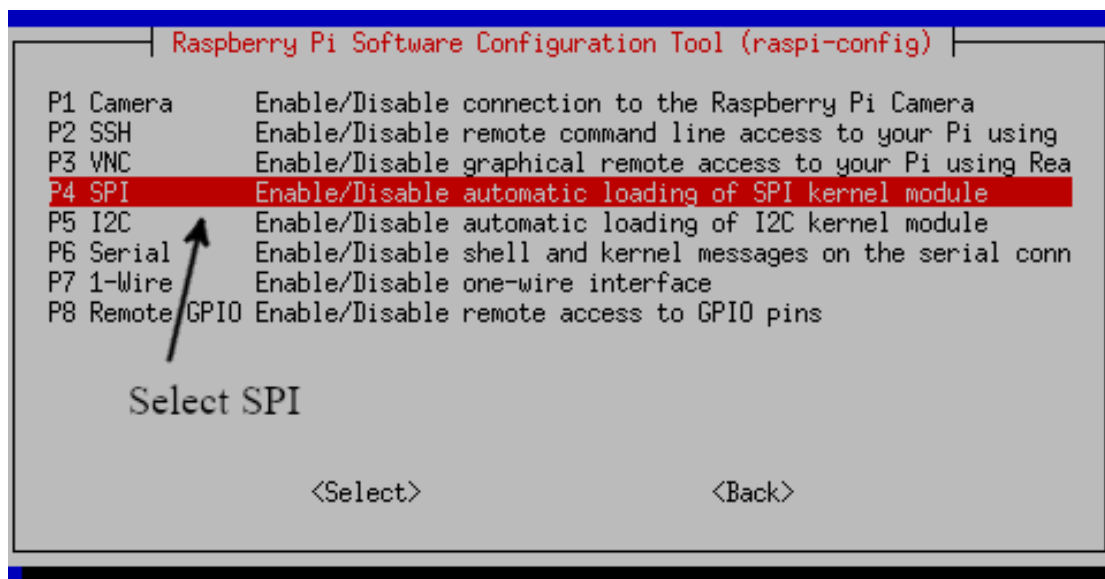
After the initial bootup, there are several libraries and settings that must be set before one can interface with the LEDs. From the Raspbian desktop, launch the terminal and run “sudo raspi-config”, which will open the window shown in Figure 48.





**Figure 48:** After running “sudo raspi-config” in the terminal, this is the resulting screen. From here, the user can change the password, screen resolution, and several other system settings. For this, the user is concerned with changing the interfacing options.

Using the d-pad select “5: Interfacing options” and the screen in Figure 49 will be displayed.



**Figure 49:** Within “Interfacing Options” the user has the option to enable and disable several settings. For this application, the user must enable SPI, VNC, and SSH.



***Figure 50: SPI Enable. After selecting SPI, highlight “<Yes>”. Press enter and reboot the Pi to enable SPI.***

From Figure 49, select “P4 SPI” and press enter. This will be followed by Figure 50, in which one should highlight “<Yes>” and press enter to enable SPI. Following the same procedure, one should enable VNC and SSH to allow remote access to the Raspberry Pi. After that, one should reboot the Raspberry Pi to apply the settings. The SK9822 chip employed in the LED strips in this design interface through the SPI protocol and will not operate properly without enabling this setting. After SPI is enabled, one should install the Python libraries to interface with the LED strips.

Begin by opening a terminal window and running the “python3” command to verify that the proper version of Python is installed. Python 3 will be used for the entirety of this project as Python 2 support has officially ceased as of January 2020. When one is met with a terminal line shown in Figure 51, it is confirmed that the correct version of Python is installed.

```
login as: pi
pi@10.1.1.143's password:
Linux ak-solar-mqp 4.19.97-v7l+ #1294 SMP Thu Jan 30 13:21:14 GMT 2020 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

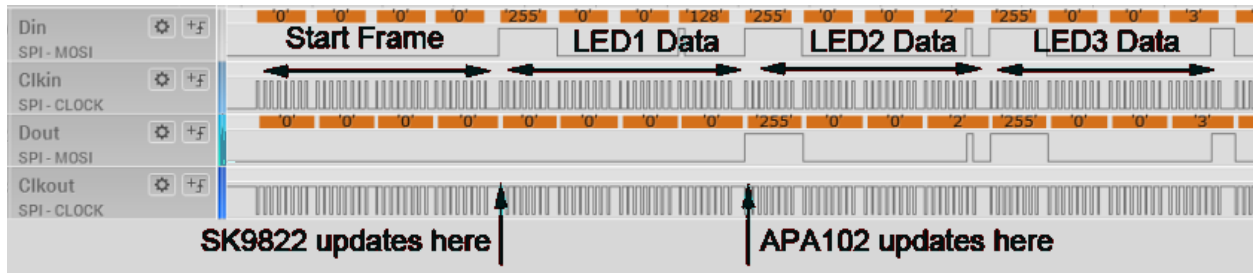
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Mar  8 18:49:46 2020
pi@ak-solar-mqp:~$ sudo raspi-config
pi@ak-solar-mqp:~$ python3
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

**Figure 51: After running the command “python3” the console should display the current Python version. If the Python console is not displayed, Python 3 must be installed.**

Libraries will be installed using the “pip” installer for Python. The pip installs functions by running “sudo pip3 “ followed by the name of the library one is intending to install. For this application, the APA102-pi driver from pypi.org is installed and modified for use with the SK9822 LED strips [48].

## 5.2 APA102 library

The chosen LED strips are driven by the SK9822 integrated circuit chip, which is a Chinese version of the more expensive APA102 driven LED strips. Since the SK9822 is a clone to the APA102, it is not always guaranteed to function correctly with any and all drivers that have been written for the APA102. The way data is cycled through the two ICs is slightly different. Although the data format is more or less the same, the color combination data is written to the SK9822 in a slightly different way. The SK9822 updates the PWM registers in the first cycle after the next start frame, whereas the APA102 updates the PWM register immediately after receiving the data. As a result, the SK9822 has a slightly delayed color update and some drivers may not work correctly [17].

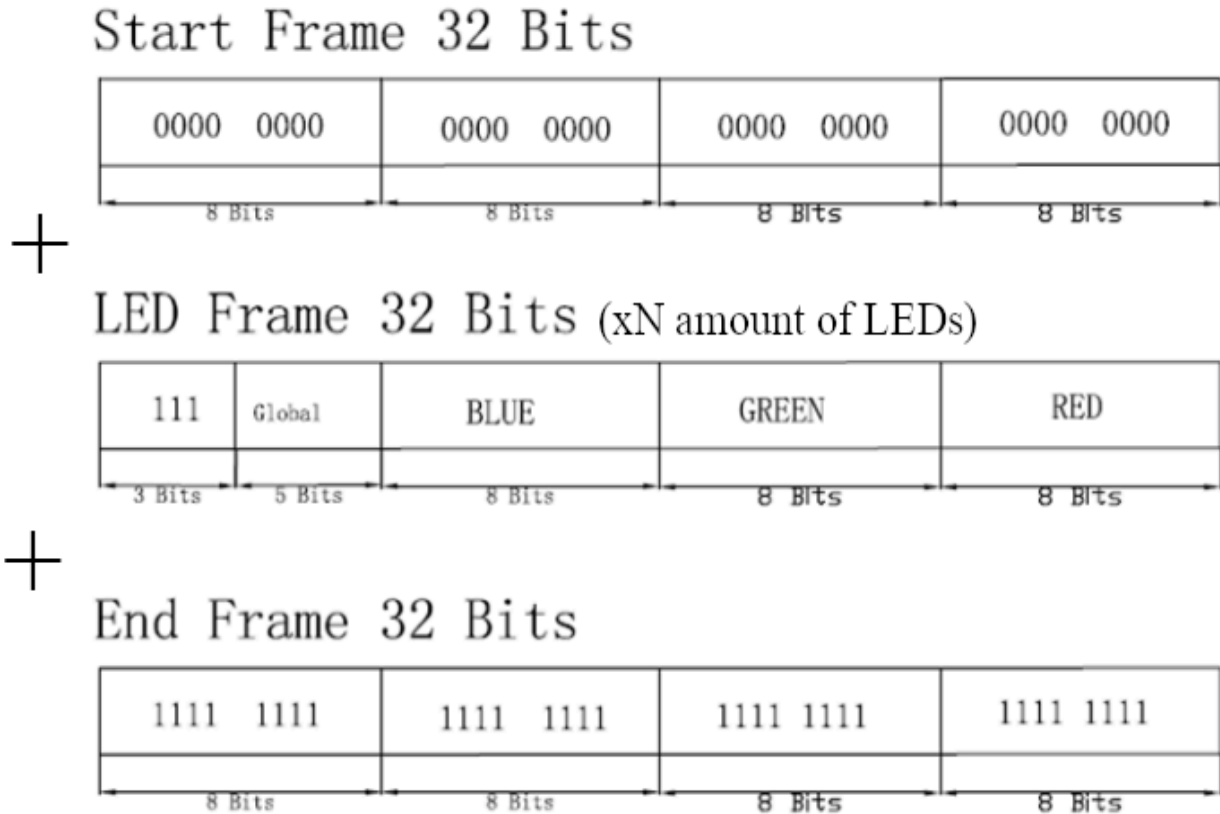


**Figure 52:** This oscilloscope shows a side by side comparison of the APA102 and SK9822 dataframes. The data structure is exactly the same, but where the data is updated differs between the two.

As shown in Figure 52, which was taken from *Tim's Blog* [17], the data frame is exactly the same for both APA102 and SK9822. However, the way the data is updated deviates slightly between the two ICs. Color combinations written to the SK9822 are updated after the first PWM register and do not write the data until the next start frame. However, the APA102 updates the PWM register immediately after receiving the data, which ironically results in a slightly delayed color update.

These differences are due to a deviation in the trigger for data frame updates. In the APA102, data updates are triggered by RGB data transmission as opposed to the update trigger of the SK9822, which is set off by a reset frame of zero bits (0x00000000) at the end of each data frame. Driver libraries written for the APA102 will oftentimes not be compatible with the SK9822 due to this discrepancy. For example, initially the Luma LED Matrix driver was installed due to its robust feature list including matrix resizing, text displaying, and state management. Unfortunately, this driver was written for the APA102 chip, where the data frame is being updated with each RGB data transmission. Unlike other drivers, the Luma LED Matrix driver does not include an end data frame of zeros, which is required to trigger the data update of the SK9822. Unfortunately, this meant that other, less feature heavy, drivers were used.

For a library protocol to work correctly with both chips, they must have the four following features; (i) a start frame of 32 zero bits (<0x00> <0x00> <0x00> <0x00>); (ii) a 32 bit LED frame defining the brightness and the RGB combination to display; (iii) a reset frame of 32 zero bits (<0x00> <0x00> <0x00> <0x00>); (iv) an end frame consisting of at least  $\frac{N}{2}$  bits, with N equaling the number of total LEDs in the entire strip [17].



**Figure 53:** One dataframe of the SK9822 consists of a start frame, an LED frame for each LED, and an end data frame. The start frame consists of 32 zero bits whereas the end frame is 32 one bits. Each LED frame contains 3 one bits to start, 5 bits to set the brightness, and 8 bits each for the amount of red, green, and blue in the color being displayed on that LED [19].

As one can see from Figure 53, the data structure consists of a start frame, the RGB data for each LED, and an end frame of 1's. This entire data frame will commit to the LEDs when a 32 bit frame of 0's is added to the end and trigger a data update. Based on this, the APA102- Pi driver was written to work for both APA102 and SK9822.

### 5.2.2 APA102 Functions

The main functions of the APA102 driver class are `set_pixel`, `set_pixel_rgb`, `show`, `clear_strip`, and `cleanup`. The driver begins by initializing the max brightness, the start LED data, and the bus speed. In the `__init__` method, the number of LEDs in the strip, the RGB map, and

brightness are defined. The data and clock pins for SPI (mosi and sclk respectively) are initialized to bus zero with a previously defined bus speed of 8000000 Hz.

The `clock_start_frame` method builds a 32 bit data frame and writes it to SPI. The `clock_end_frame` method writes another 32 bits of zeros to trigger a data reset. The `clear_strip` method iterates through each LED in the strip in a for loop and sets each pixel color and brightness to zero.

The two most frequently used methods are `set_pixel` and `set_pixel_rgb`. `set_pixel` takes in `led_num`, `red`, `green`, `blue`, and `brightness` percentages as input parameters. This function first sets the brightness by checking for a valid input and then building an 8 bit start frame initializing the LED matrix and assigning global brightness to each LED in the last 5 bits of the 8 bit start frame. After this, the function proceeds to initialize the red, green, and blue lights contained in each LED on the strip. `set_pixel_rgb` takes in `led_num`, `rgb_color`, and `brightness` as input parameters. The function sends whatever combination of red, blue, and green color to create the hex RGB color that is input as the `rgb_color` parameter. It does this by building three 32 bit color frames for red, blue, and green. The function takes the hex value for red and compares it to the input RGB color through an “and” operand and shifts the bits to the right by 16, the same is repeated for green but shifts the bit to the right by 8, and blue without any bit shifting. The result is sending the correct ratio of red, green, and blue to create the desired hex color when passed into `set_pixel`.

### **5.2.3 Module\_test.py**

A simple script is developed to test and debug each individual module of the overall system. The script begins by importing the time library for delays and theAPA102 driver. A LED strip object is initialized using theAPA102 driver setting the number of LEDs in the strip to 100 (for one strip), the global brightness to 1 (the lowest setting for power efficiency), and the order of the RGB color interpretation. The script then iterated through a for loop in the range of 100 indices to light up each LED in the strip one by one with a delay of 0.025 seconds between each light. For testing purposes, each LED is lit up in 0xFF0000, which is the hex color for red. This was chosen for maximum power efficiency. This for loop is nested in an infinite loop so data is continuously being fed through the strip. When debugging each module, it is integral to have

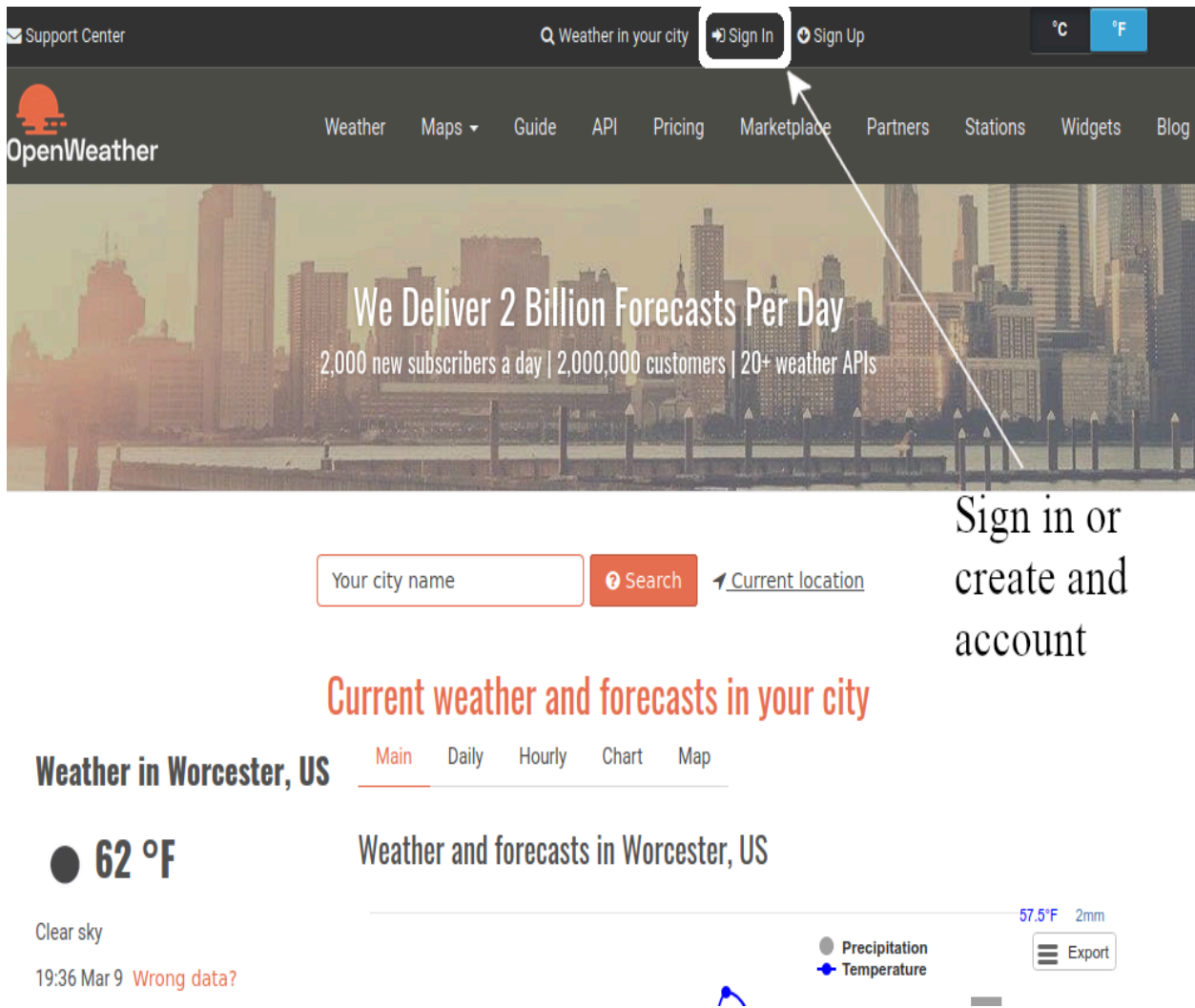
data constantly being fed through the sdata and the clock so that when there is a lack of signal it will be immediately obvious that there is something amiss.

Module `_test.py` calls the `set_pixel_rgb` function from the `APA102` class in the for loop by using the initial parameters set in the initialization of the strip object. After this the `show` function is called from `APA102` and the LED is lit. A 0.025 second delay is programmed in between the setting of the pixel RGB and the displaying to create a cascade animation across the strip so isolated errors can be identified and rectified. When running the test, the user is given the option to either run the test by pressing 1 or clearing the strip by pressing 0. This script was used for the entirety of the testing process and is refined to create animations later on.

### **5.3 OpenWeatherMap API**

After a failed attempt to use html web scraping to collect information on the current weather conditions, it was decided that an API would be a better choice for this application. Attempts at making a request for `accuweather`, `google weather`, and `weather.com` all were refused due to inadequate permissions to access the websites from a headless script. OpenWeather's current weather data API offers a much easier to implement and more reliable method of collecting weather data than doing it manually through html scraping.

To begin using the API, one needs to create a free account and generate an API key on the official OpenWeather website [49]. This can be done by visiting the website, logging in or creating an account, and navigating to the API page. Figures 54, 55, and 56 show exactly how to generate an API key on the OpenWeather website.

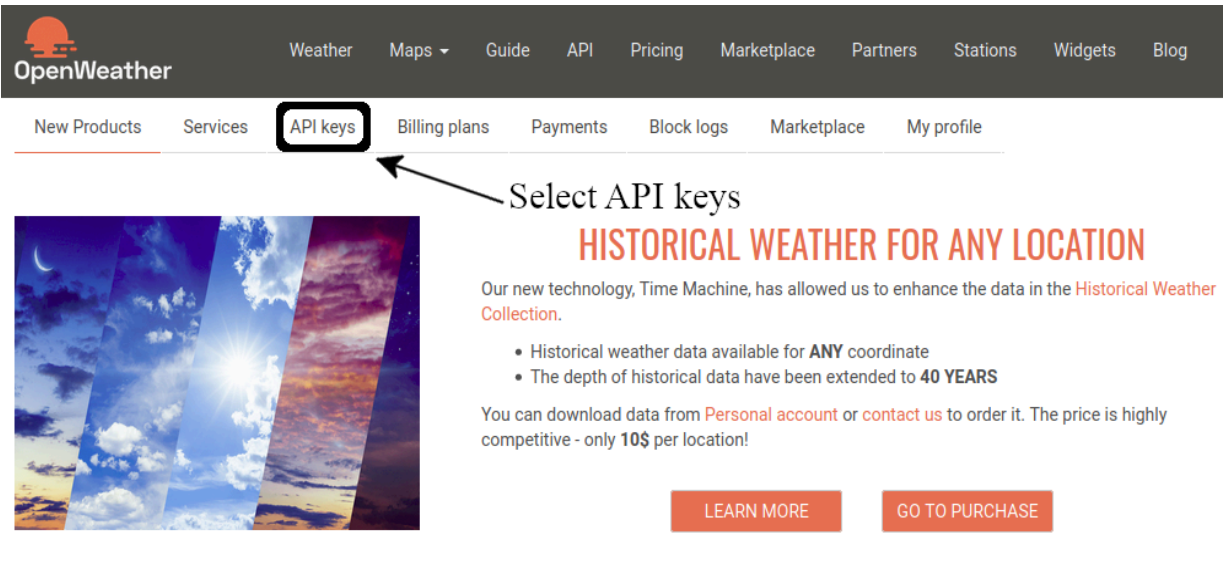


Sign in or  
create and  
account

**Figure 54: The main page to openweathermap.org. Click “Sign In” to sign in or create an account.**

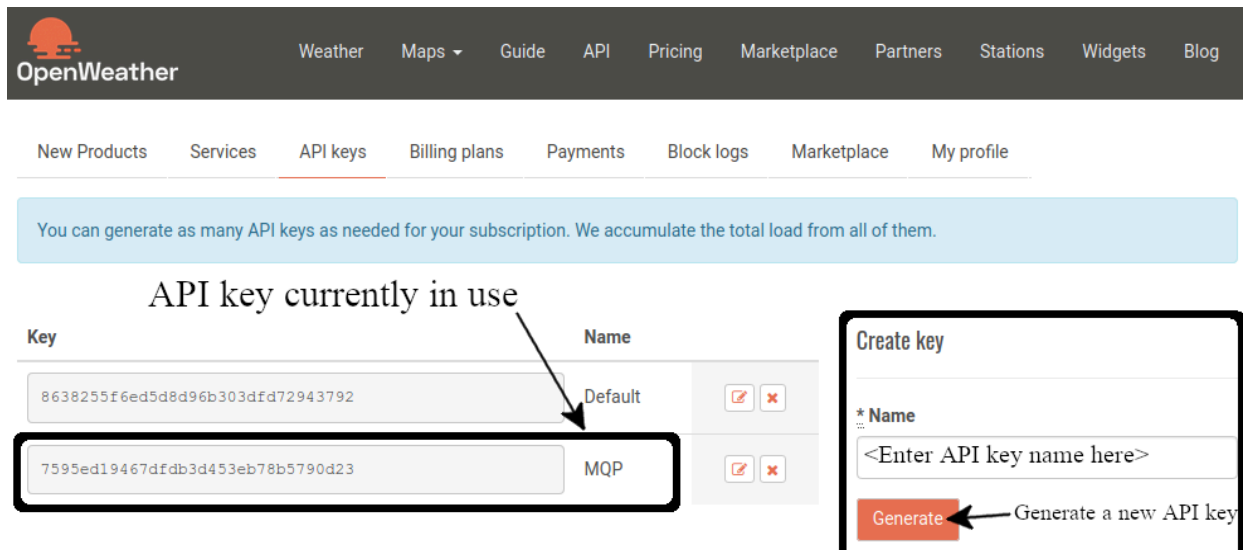
The OpenWeather main page is shown in Figure 54. Here, one shall select the “Sign in” option to create an account or sign in to an already existing one. When creating a new account, it is crucial that only the free account is used, as no premium features are required for this application.





**Figure 55:** The first screen one is met with after signing in. Click on “API keys” to view the accounts API keys.

Once “API keys” is selected, one can generate an API key for use in their project as seen in Figure 56.



**Figure 56:** Enter a name in “Create Key” and click “generate” to create a new API key. The “MQP” API key is what is used for this project.

Using the key generated in Figure 56, one can add the key to their code and begin collecting weather information. Note that only a free account will be used, therefore only current weather conditions and forecasts will be returned by this API.

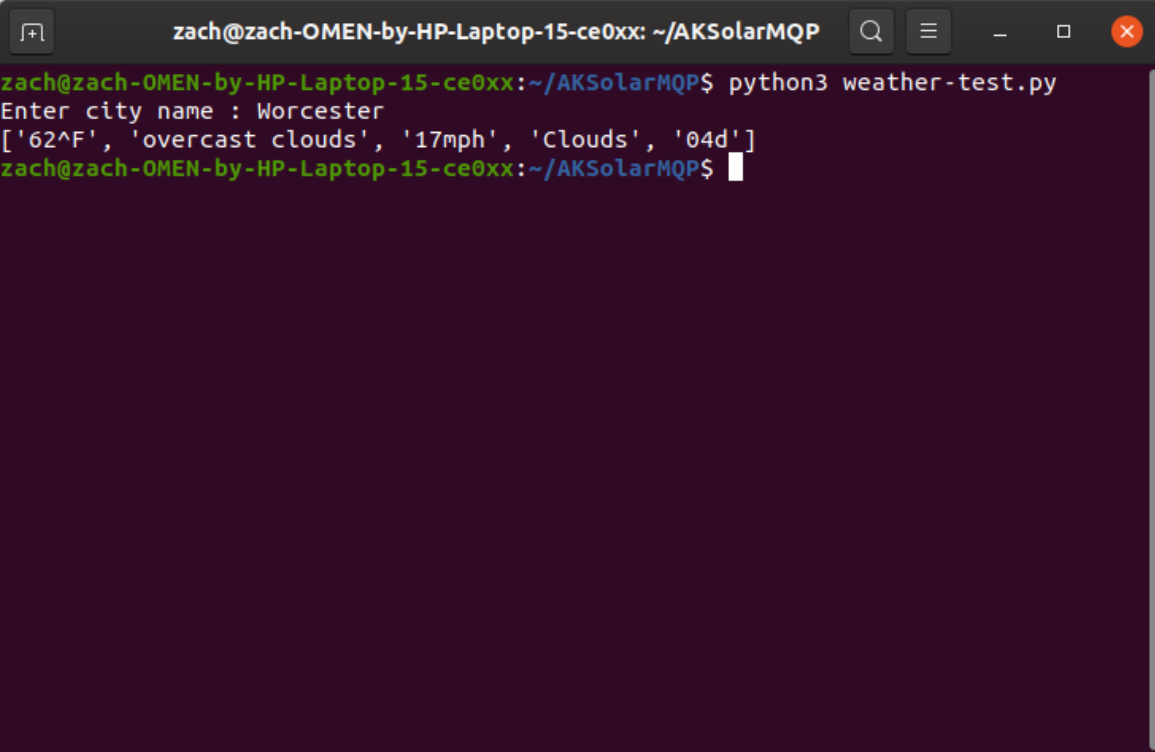
### 5.3.1 OpenWeather API Outputs

Weather-test.py is a simple testing script that was written to test the functionality of the OpenWeather API and investigate the data that is returned. The script first declares the API key, API url, and the city name as an input to create an API call that will return weather for any given city. In this case, Worcester, MA is used for all testing. A url string is concatenated together using these three objects and the result is used as an input parameter to the Python *requests* library. The response of this url request is saved in the “response” object and converted to a JSON data structure for ease of parsing. Figure 57 is an example of a response JSON object created from an API call.

```
{
  "coord": {
    "lon": -122.08,
    "lat": 37.39
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 282.55,
    "feels_like": 281.86,
    "temp_min": 280.37,
    "temp_max": 284.26,
    "pressure": 1023,
    "humidity": 100
  },
  "visibility": 16093,
  "wind": {
    "speed": 1.5,
    "deg": 350
  }
}
```

**Figure 57:** An example of a response to the OpenWeatherMap API. Within this JSON, current weather conditions, current temperature, and wind are of concern to display on the LED matrix.

After the API response, the script checks for the “404” element to make sure the input city is recognized. The script then looks into the “main” element and parses for temperature, pressure, and humidity information. After that, the script looks at the “weather” element and parses for the weather description. The script then compiles a string containing all of this information and prints it in the terminal. A modified version of this script is later added to *matrix.py* as a class that returns the current weather conditions, temperature, and wind as a list.



```
zach@zach-OMEN-by-HP-Laptop-15-ce0xx: ~/AKSolarMQP
zach@zach-OMEN-by-HP-Laptop-15-ce0xx:~/AKSolarMQP$ python3 weather-test.py
Enter city name : Worcester
['62^F', 'overcast clouds', '17mph', 'Clouds', '04d']
zach@zach-OMEN-by-HP-Laptop-15-ce0xx:~/AKSolarMQP$
```

*Figure 58: Weather-test.py is modified for use as a class in matrix.py. The modified class returns a 5 element list, with element 0 reporting the temperature in fahrenheit, element 1 reporting the detailed forecast, element 2 reporting the wind speed, element 3 reporting the main forecast, and element 4 reporting the code indicating night or day.*

## 5.4 Matrix.py

Matrix.py contains fifteen total classes that consist of matrix class, the weather class, a hypersonic sensor driver, and twelve animation classes. Matrix.py appropriately begins with the “matrix” class which computes the index of the LED strip that corresponds to the coordinate position (x,y) in the total matrix. It does this by first computing the number of LEDs per column.

Given the dimensions of the display, the algorithm determines the linear LED address which corresponds to the known coordinates. The algorithm is as follows:

$$\{[x \times L] + |([x\%2] \times L) - y - 1|\}$$

where W is the width, L is the length of the display, and x and y are the X and Y coordinates, respectively. Note, each variable listed is zero indexed. The algorithm above is in two parts. First,  $[x \times L]$ , determines how many rows of LEDs precede the active row. For example, X is 5, then the active row is 6, so the number of LEDs in the rows before is X times the length. The other portion of the equation determines how far down the column the active LED is. The statement  $|([x\%2] \times L) - y - 1|$  decides where to subtract Y from the length or simply add Y, based on whether the signal is running right to left or left to right in the active row. The result is then summed with the total LEDs from preceding rows and this value is the address of the desired LED at the given coordinates.

There are some slight differences between the mathematics above and the code implemented. In the code, an ‘if statement’ is used to fix an error on the odd X coordinates. The factor is off by one, and is fixed in the algorithm above. Also in the code, X and Y are referred to as coord[0] and coord[1], respectively. The code can be found at the beginning of Appendix C.

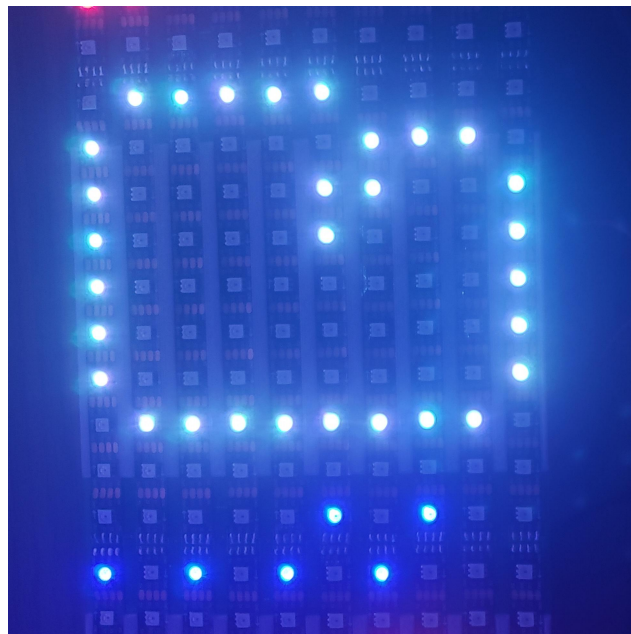
The weather class works exactly as weather-test.py does. The getWeather() function behaves the same at wether-test.py but returns the weather information instead of just printing it. An API URL is built using the API key, a base API URL, and the city name. The city name is hard coded as Worcester, MA and, therefore the API will only collect weather information for Worcester. The list returned by the getWeather function can be seen in Figure 58. The main() function of matrix.py calls weather().getWeather() and saves the current weather conditions to an object that is then inspected to determine what information to show on the display.

The SainSmart HC-SR04 hypersonic sensor, used to determine patron proximity to the display, is driven using the echoSens() class function distance(). The driver works by first initializing which GPIO pins are being used to trigger the sensor and receive its echo. The trigger

pin sends a signal to the trigger of the hypersonic sensor, sending a high frequency sound wave out of the sensor. The distance() function then measures the time it takes to receive a response from the echo pin. When the sound wave is reflected back to the echo pin of the sensor, the echo GPIO pin will pulse to logic high (1). The time it takes to receive an echo from the trigger is directly proportional to the distance an object is from the sensor. With this in mind, the echoSens class converts the measured time between trigger and echo into centimeters and returns it to the main() function.

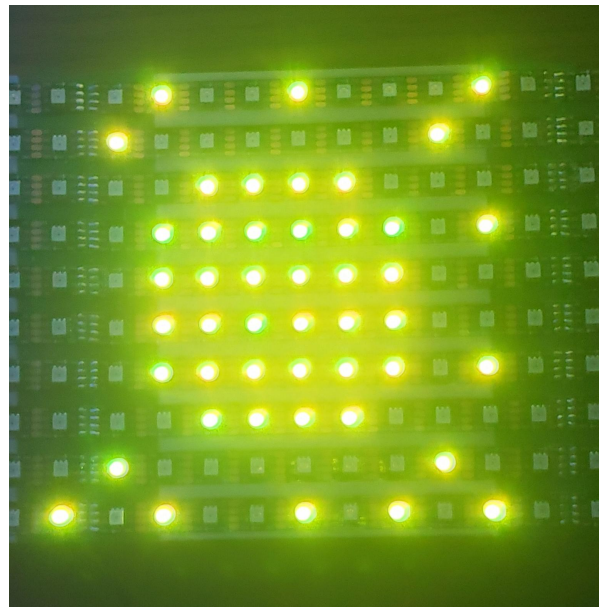
### 5.4.1 Animations

For each animation, arrays are saved as animation objects that are then iterated through to draw a final icon, that is often accompanied by a moving element. Each animation object is an array of tuples containing the coordinates for each pixel in the animation. These tuples are sent to matrix().get\_coord() to convert the x,y coordinates to an index (0-899) in the overall LED strip. Using this index, each pixel is set to a predetermined color most appropriate for the design. Figures 59-69 are every possible animation that can be displayed by matrix.py.



*Figure 59: The rain animation is split into two arrays. One for a static cloud drawing and one for the raining animation. This displays when drizzle or rain is returned by the Weather() class*

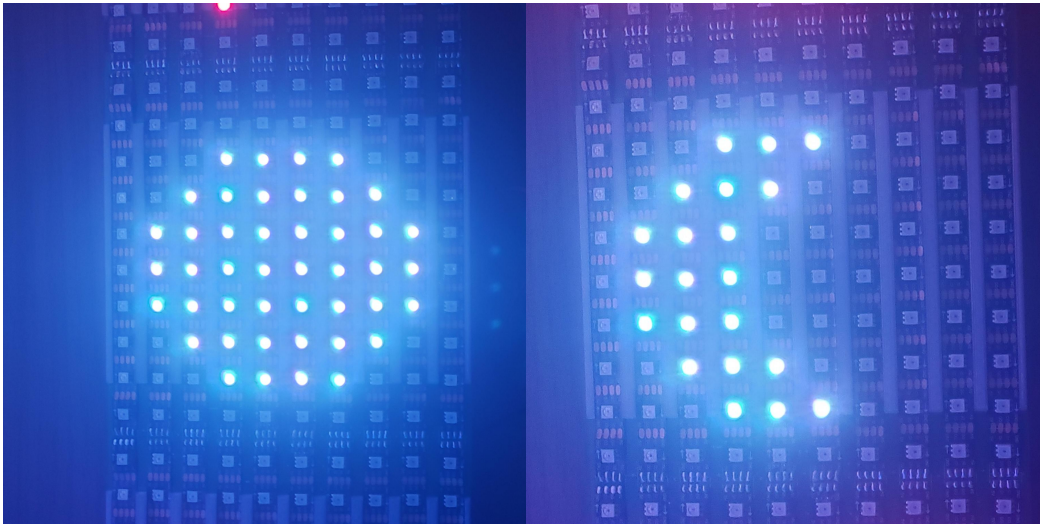
In the raining animation, two arrays are iterated through. The cloud array is first, which contains the coordinates to draw the cloud shown in Figure 59. This is done by iterating through a for loop of the cloud array and assigning a silver color (hex 0xC0C0) to the LED strip index returned by `matrix.get_coord()`. The for loop pauses for 25 milliseconds before drawing the next pixel. Subsequently, the rain array, which contains the “raindrops” shown at the bottom of Figure 59, is iterated through and cleared in an infinite loop. This is done through two for loops contained within an infinite while loop. The first for loop iterates through the array and assigns each LED strip index returned by `matrix.get_coord()` to the color blue (hex 0x0000FF). The second for loop iterates through the same array, but sets each pixel to zero, turning it off. These two loops repeat indefinitely in sequence, creating a cascading rainfall effect. The infinite While loop is broken when the global variable “`stop_animate`” is set to True by the `main()` function.



*Figure 60: There are two arrays in the sunshine animation. One for a static drawing of the sun’s center and another for the rays protruding from it. This displays when clear skies are reported during the day.*

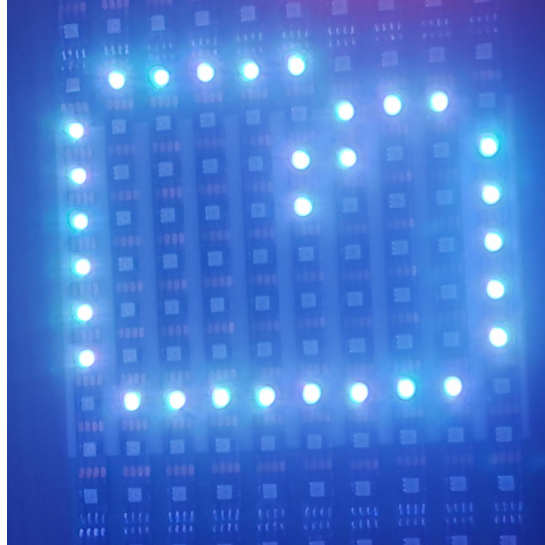
The sun animation functions in exactly the same way. The only difference being the “cloud” array is the middle yellow circle of the sun and the “rain” array is the rays radiating outward. The same static for loop followed by two for loops within an infinite loop structure is

used, only all pixels are set to yellow (hex 0xFFFF00), as seen in Figure 60. Also like the rain animation, the while loop is also broken by the main() function setting “stop\_animation” to True.



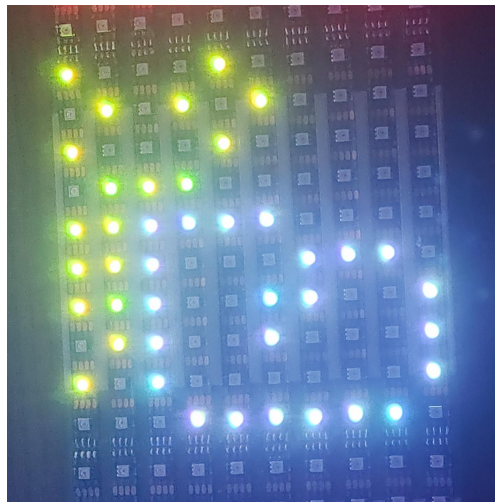
***Figure 61: There are two arrays in the moon animation. One that draws a static picture of the moon and another that removes half the drawing to form a half crescent and then redraws the full moon again. This displays when clear skies are reported during the night.***

Much like the rain and sunshine animations, the moon animation uses two arrays. One for a static image to be drawn initially and a second to animate the filling and erasing of the half crescent. The initial for loop followed by two infinitely looping for loops structure is used again, setting each LED to silver (hex 0xC0C0C0). This loop is again broken by the “stop\_animate” global boolean. Figure 61 shows the beginning and end stages of the animation.



***Figure 62: There is only one array for the cloud animation that draws a static image of a cloud. This is displayed when broken clouds or overcast clouds are reported.***

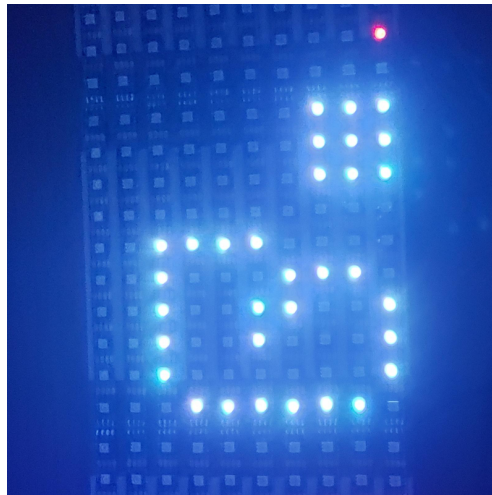
The cloud animation only has one array. The cloud array is iterated through a for loop, like the other animations, and individually sets each pixel to silver (hex 0xC0C0C0). The difference here being, the animation ends here. The cloud animation just draws the icon and keeps it displayed until the entire display clears.



***Figure 63: There are two arrays in this animation. One draws a static image of a cloud and another animates a sun shining behind the cloud. This is displayed when broken clouds or overcast clouds are reported during the day.***

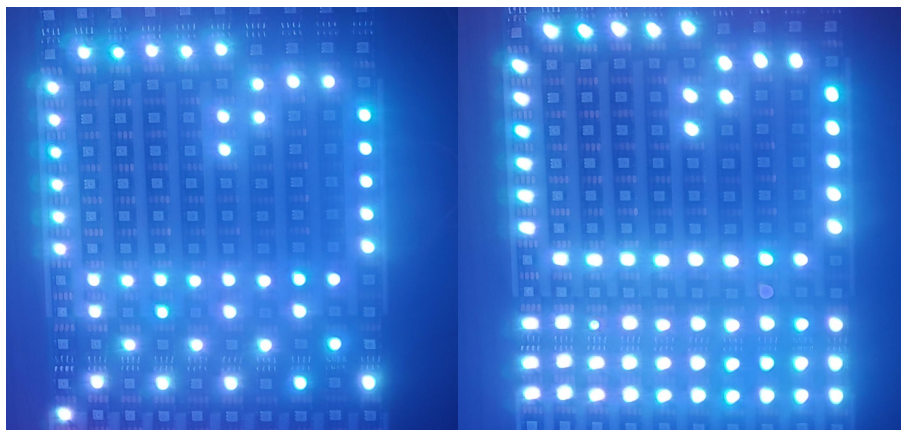


The partly cloudy animation functions exactly as the rain, sun, and moon animations. There is a cloud array and a sun array. The cloud array is drawn in an initial for loop and then sun animates a radiating movement through an infinite while loop containing a draw and a clear for loop iterating through the sun array. This animation is stopped by the “stop\_animate” global boolean. Figure 63 represents the final stage of the sun animation.



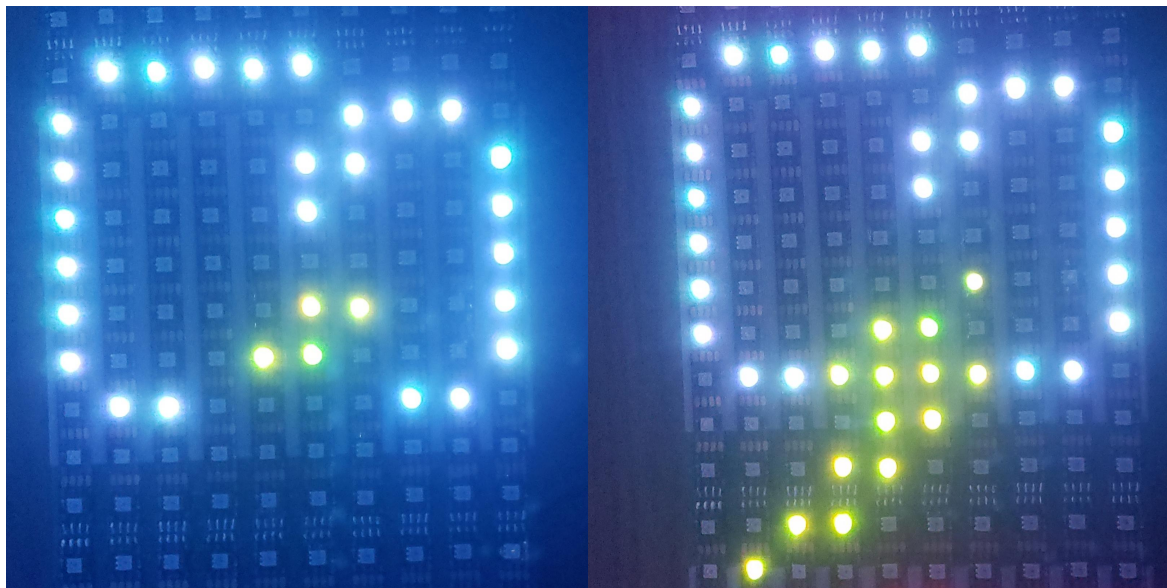
***Figure 64: There are two arrays in this animation. One that draws a static cloud and another that draws a small moon in the distance. This is displayed when clear skies are reported at night.***

The same applies to the nighttime partly cloudy animation, only the sun array is replaced by a moon array found in the top right of Figure 64, which shows the moon fully drawn.



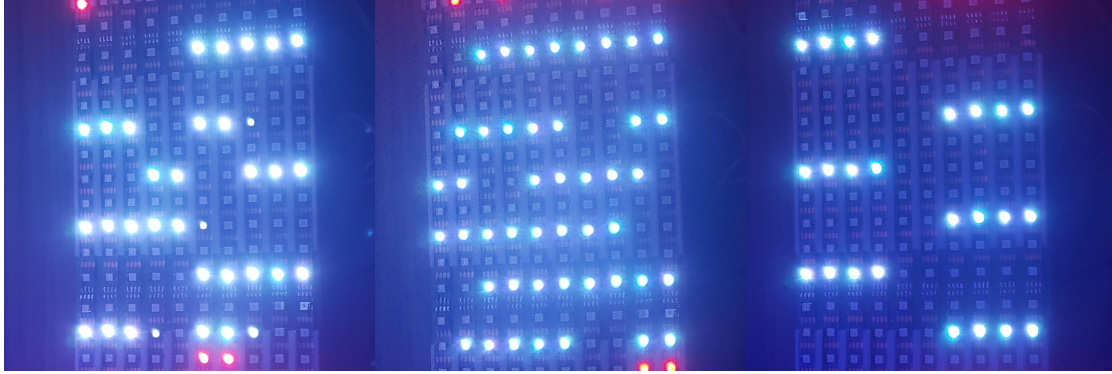
***Figure 65: There are two arrays in this animation. One draws a static cloud and another animates snow falling from it and accumulating at the bottom of the module. This is displayed when snow is reported.***

The snowing animation is essentially a copy of the rain animation with a few important distinctions. The rain array is replaced by the snow array, which staggers instead of falling in one direction. Additional pixels are also added to the snow array to fill in the bottom of the module, as seen in the right frame of Figure 65. The exact functional structure is used with the initial for loop and two infinite for loops drawing and clearing the snow array. The “stop\_animate” global variable breaks this infinite loop.



***Figure 66: There are two arrays in this animation. One that draws a static cloud and another that draws a lightning bolt striking through it. This is displayed when thunderstorms are reported.***

Again, using the same structure as with snow, rain, and sun animations, the thunderstorm animation first draws a cloud using the cloud array. A lightning bolt is then drawn by setting each pixel in the bolt array to yellow (hex 0xFFFF00) and erased using the same while loop structure. The pause between drawing pixels is slightly reduced to emphasize the speed of the lightning bolt. Figure 66 above shows this animation.



***Figure 67: This animation is split into two arrays. Every other row starting from the top scrolls to the left and every other row starting from the second highest line scrolls to the right in a loop. This is displayed when fog is reported.***

For the fog animation there are two arrays. One array scrolls to the right the other scrolls to the left. This is done by first concatenating the arrays and iterating through the result in a for loop to display the starting image. The left and right arrays are then shifted 9 pixels to the right and left respectively. All under an infinite while loop, the left array is iterated through a for loop which shifts each element of left to the left and appends each shifted pixel to a temporary “clear” array. After that, the right array goes through the same type of loop, only pixels are shifted to the right. In both loops, the pixel RGB is set to silver (hex 0xC0C0C0). After the right for loop, the shifted versions of the left and right arrays are built and ready to display. For each iteration of the infinite loop containing both the left and right for loops, the strip is updated. This allows for a frame by frame animation of complete shapes, rather than a pixel by pixel animation type used in the other animations. Each frame is displayed for  $\frac{1}{4}$  of a second before being cleared. Each frame is cleared by iterating through the temporary “clear” array and setting each pixel to zero. The strip is not updated until the completion of the clear array loop, so the entire frame is cleared instead of pixel by pixel. Figure 67 shows various frames of the fog animation. The “stop\_animate” global boolean is used to break the infinite loop.



*Figure 68: This animation converts a string of ASCII characters into pixel indexes on the matrix. With this, static letters are drawn and remain in place.*

The `staticText()` class takes a string as an input and then converts it into an array of pixels corresponding to each letter. This is done by inheriting the `characters.py` library; a separate file containing a dictionary of the pixel coordinates required to draw any ASCII character on this display. The `staticText()` class iterates through each character of the input string and compares it to the dictionary of ASCII characters in `characters.py`. When it finds the matching key element in the dictionary, `staticText()` adds the corresponding array of tuples that are required to draw that character to a new array. Once the final string array is built, the y coordinate is shifted according to the “y\_shift” input parameter and the resulting array is iterated through. Each tuple element contains the x and y coordinates that are then converted to an LED strip index through `matrix().get_coord()` and set to an input color. In each iteration of the loop, a pixel is drawn creating a cascading effect almost like drawing with a pencil or pen. The result is shown above in Figure 68.



**Figure 69:** *This animation converts a string of ASCII characters into pixel indexes on the matrix. These pixels are shifted to the right and scroll from the right edge to the left in an infinite loop.*

The `scrollText()` class uses the same algorithm as `staticText()` to convert input strings into tuple arrays of pixel coordinates required to draw the string. Much like the fog animation, `scrollText` shifts the array to the right and then frame by frame creates a new array shifted one pixel to the left. This creates a scrolling effect from right to left. Shifting the array to the right initially makes the text appear as if it has been strung around the back of the display and back around to the other side when the end of the word is reached. Figure 69 shows various frames of the string “61°F” as it moves from right to left in a loop. The “`stop_animate`” global boolean is again used to end the animation.

## 5.4.2 Main()

The `main()` function of `matrix.py` initializes the matrix leds, measures patron’s distance from the display, determines the correct weather animation to display, and builds all the additional information to show on the display. After initializing the LED matrix, the `main()` function calls the `echoSens().distance()` function every half second to determine how far away any objects are from the display. This function call is within a while loop that will infinitely loop until a distance of less than 60 centimeters is recorded. When the `main()` function detects an object less than 60 centimeters away, it begins building the data that is to be shown on the display.

The `main()` function first uses the `datetime` Python library to determine the time and date, saving the date as a single string object and the hours and minutes of the current time as two

separate string objects. Weather information is collected using the `weather().getWeather()` function and threads are created for the temperature, date, hour, minute, am/pm, wind label, and wind speed.

The `main()` function uses a variety of if statements to determine which weather animation is appropriate to show on the display given the current weather conditions returned by `weather().getWeather()`. If the main weather forecast is foggy the animation is set to the `fog()` class. If the main weather forecast is thunderstorms the animation is set to the `thunder()` class. If the main weather forecast is drizzle or rain the animation is set to the `rain()` class. If the main weather forecast is snow the animation is set to the `snow()` class. If the main weather forecast is clear and the `weather().getWeather` function returns a nighttime icon code animation is set to the `moon()` class, otherwise it is set to the `sunshine()` class. If the main weather forecast is cloudy, the `main()` function must determine how severe before assigning an animation. If the main weather forecast is cloudy and the forecast description is “few clouds” or “scattered clouds” the animation is set to the `partcloud()` class. If this is all true and the icon code is for the night time, then the animation is set to the `nightpartcloud()` class. Finally, if the main forecast is cloudy and the forecast description is “broken clouds” or “overcast clouds” the animation is set to the `cloud()` class. In the event of a forecast that is not common to the area, such as tornadoes or dust, the display will simply show a scrolling text of the main forecast.

Once the weather animation is determined, the `main()` function can start each thread and run every animation from the date to the wind speed in parallel, effectively displaying all of the information it collected. The display remains on for 60 seconds, then the strip is cleared and the “`stop_animate`” global boolean is set to false to clear any secondary animations. The GPIO pins are then reset and the whole process starts over again by measuring object distance until an object is detected 60 cm or closer to the display again. This process loops forever as long as power is connected to the Raspberry Pi. The `matrix.py` script is added to the end of the “`/etc/profiles`” configuration file built into the OS of the Raspberry Pi, making the script run automatically after boot. This ensures that whenever the Raspberry Pi is turned off and back on again, the `matrix.py` script will automatically run without any input from the user. As long as the Raspberry Pi and display has power, the script is running and will display weather information when approached.

## 5.5 Chapter Summary

This chapter guides a user through the installation and development of all necessary software for implementing the display part of this system. Raspberry Pi setup is an important part of the software methodology. The correct versions of Raspian must be installed and the correct config settings set for the application code to work properly. Software libraries were a critical component of software for this project. Using the existing APA102 Python library means that low-level driver software does not need to be written to implement an application. Other important areas to note are the testing software, like `module_test.py`, which helps to debug certain areas of the display and fix any errors that occur. The application code, made up by the `weather-test.py`, `matrix.py`, `echo_sens.py` and `charcters.py` scripts, is the most important part of the software methodology. These scripts provide the necessary logic for full functionality of the display.

## Chapter 6: Full System Integration

Chapter 6 details the process of implementing the system based on the project design described in previous chapters. This chapter includes changes and additions to the proposed design they were found to be needed during the implementation. The details of the working system, as well as the overall success and shortcomings of the initial version of the project are detailed here. The overall block diagram of the system is shown in Figure 70, noting the key in the upper right corner for guidance. The elements of the system flow from top to bottom to indicate the source of power and the end device that uses said power. The box denoted “SCC” stands for supplementary control circuit, and is the circuit that tells the wireless power transfer system to begin transfer after power-up. Furthermore, the dotted box describes the area at which the window to transfer power over will be located.



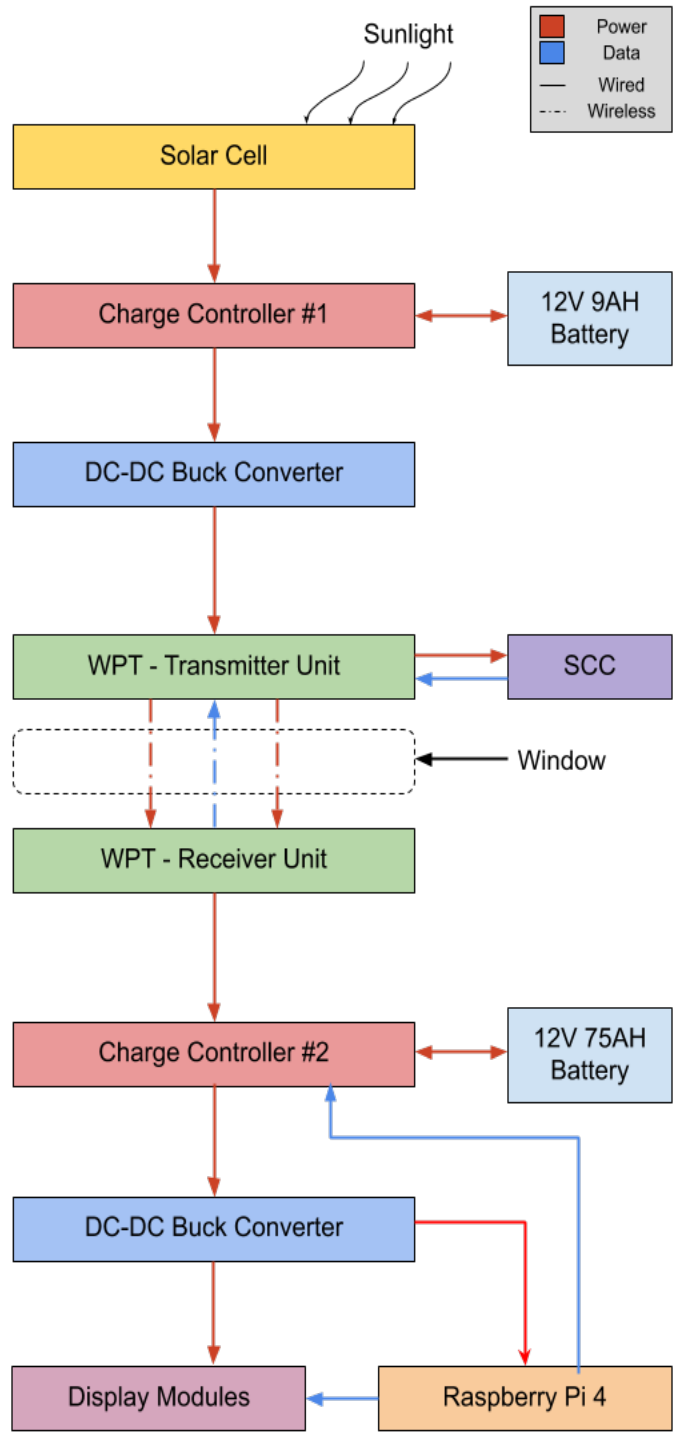


Figure 70: The diagram flows from top to bottom, indicating the direction of power flow. The window is located between the WPT transmitter and receiver.

## 6.1 Solar

The purpose of the solar portion of the system is to generate power for the system as a whole. One of the necessary considerations was the location of the solar panel for maximum power generation. If any part of the solar panel is blocked from direct sunlight, including things like clouds and passing shadows, the power output of the panel decreases. The solar panel will be placed next to the bicycle racks outside the Atwater Kent Laboratories at WPI, as shown in Figure 71.

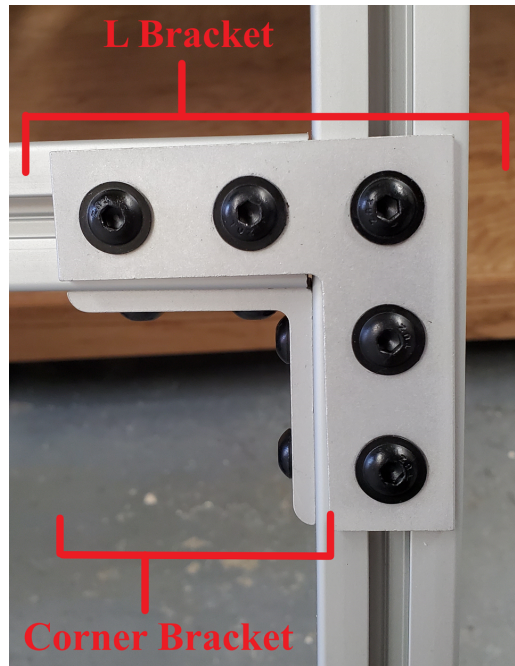


***Figure 71: The solar panel is located to the left of the entrance to Atwater Kent Laboratories, next to the bicycle racks. This location was chosen for its proximity to the stairwell and the amount of sun available in that area.***

The solar panel mount constructed with assistance from WPI Facility's Bill Appleyard used three centimeter thick 80/20 Aluminum bars and various connectors that were donated by Justin Woodard from National Grid. The mount for the solar module uses two major frame parts: a mount frame and L frame. The mount frame was made to directly connect to the solar panel and provide sections for the L frame to support the solar mount. These frame pieces were also designed to be shiftable so the mount can be angled at a varying range of angles from the ground.

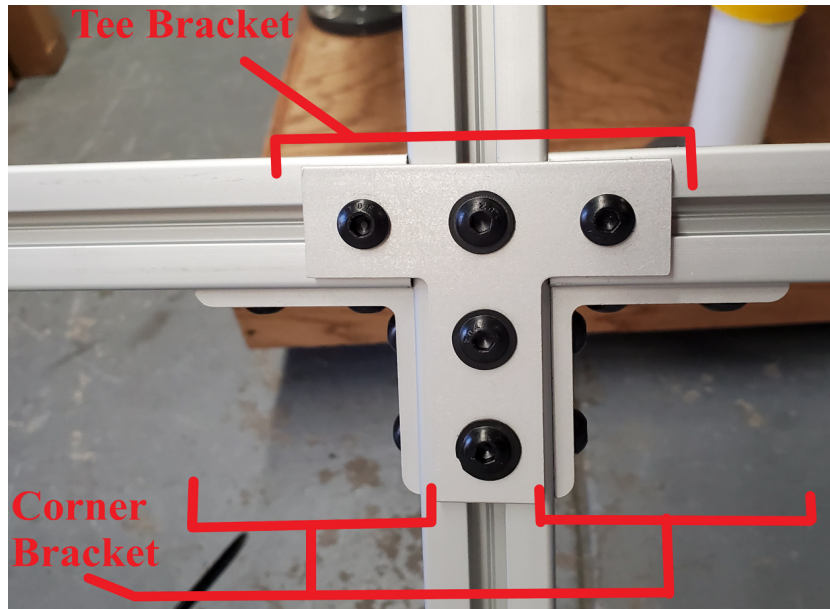
The mounting frame is attached directly to the back of the solar panel, securely connecting to the L frame. The mounting frame needed to line up with screw holes on the solar panel. As a result, the frame's dimensions are 194 centimeters long and 91 centimeters wide

instead of 2 meters long and 1 meter wide. Additionally, the middle connector on the frame needed to be slightly off-center in order to connect to the L frame properly, because the stake from the L frame is set directly in the center as seen in Figure 76.

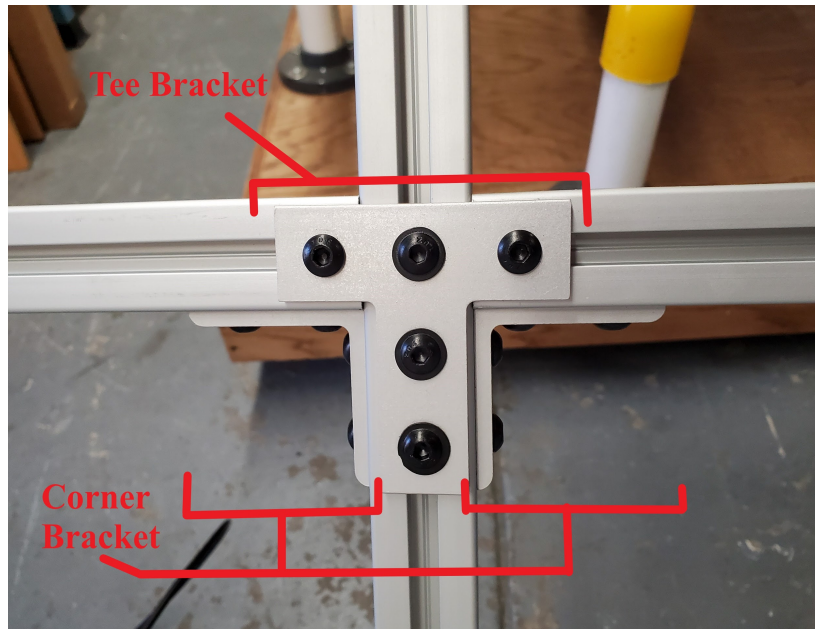


***Figure 72: This picture depicts the right leg of the solar mount connected with an L bracket and corner bracket.***

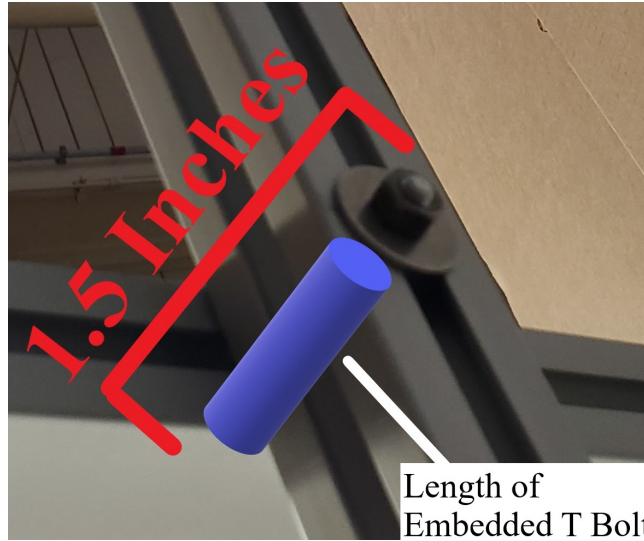
The L frame is the main support for the solar mount. Since the L frame has two connectors at the bottom of the mounting frame that can change angles, the solar mount can also be set up to hold the solar panel up at different angles off the ground. The range of angles that the solar panel can achieve is about 5 to 85 degrees. This design allows the panel to be set up on terrain that isn't completely flat. The pieces used to build this frame are 4 corner brackets, 1 tee bracket, and 2 L brackets, as shown in Figures 73, 74, 75, and 76. In order to adjust the frame for the different lengths, the brackets on the frame itself can be loosened and shifted to the lengths needed for the angle required as seen in Figure 72.



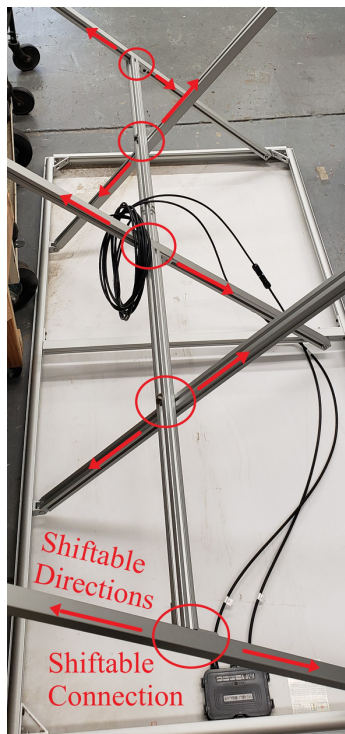
*Figure 73: Middle Leg of Solar Mount. This picture depicts the middle leg of the solar mount connected with a tee bracket and two corner brackets.*



*Figure 74: This picture depicts the left leg of the solar mount connected with an L bracket and corner bracket.*



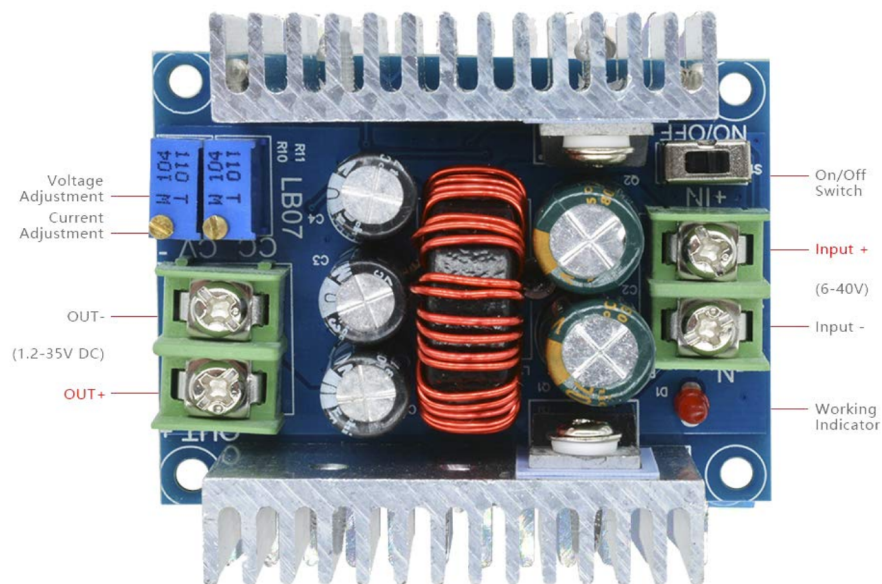
*Figure 75: This picture depicts the screw connection between the bottom frame and the connecting frame between the legs.*



*Figure 76: This picture depicts the whole solar mount connected to the flipped solar panel. All connections are highlighted and the movable parts and the directions are marked.*

## 6.2 WPT

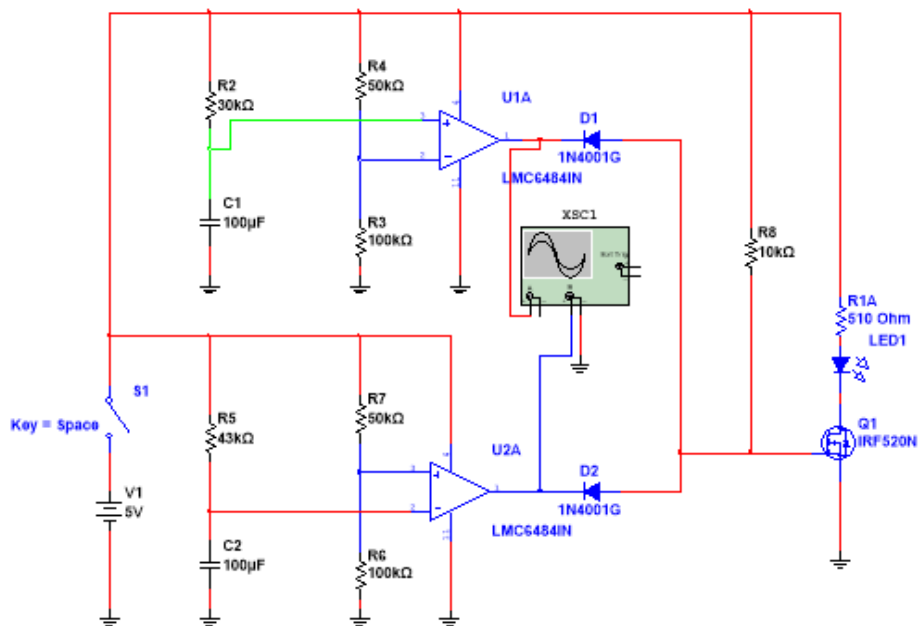
Upon linking the solar charge controller to the WPT system, it was discovered that the charge controller did not regulate its load-side voltage to 24V, but rather just connected the load to the battery output through a relay. Due to the fact that this part of the system uses two sealed lead acid batteries in series, this means that the battery voltage would fluctuate from over 25V to under 24V. A separate component is needed to ensure that the input voltage to the wireless power system is a constant 24V. The ideal solution is to use a buck-boost DC-DC converter to convert voltages both above and below 24V to 24V. However, a requirement for anything in the power path of the system is that it must be capable of sustaining a current flow of 6A. Of the available buck-boost converters available on Amazon, none hold a wattage rating that was sufficient for the intended output power for the device. However, there are plenty of buck converters that have >150W ratings and ideal voltage input and output ranges for the system, such as the Aideepen 20A 300W Buck Converter [50], shown in Figure 77.



**Figure 77: This module features a wide input and output range, large heatsinks, and both output voltage and current adjustment knobs.**

This part allows for the input voltage to be regulated to *no more than 24V*. At below a battery voltage of 24V, the buck converter output will be the battery voltage. Any voltage between 19V and 24V is an acceptable input for the wireless power transmitter [5].

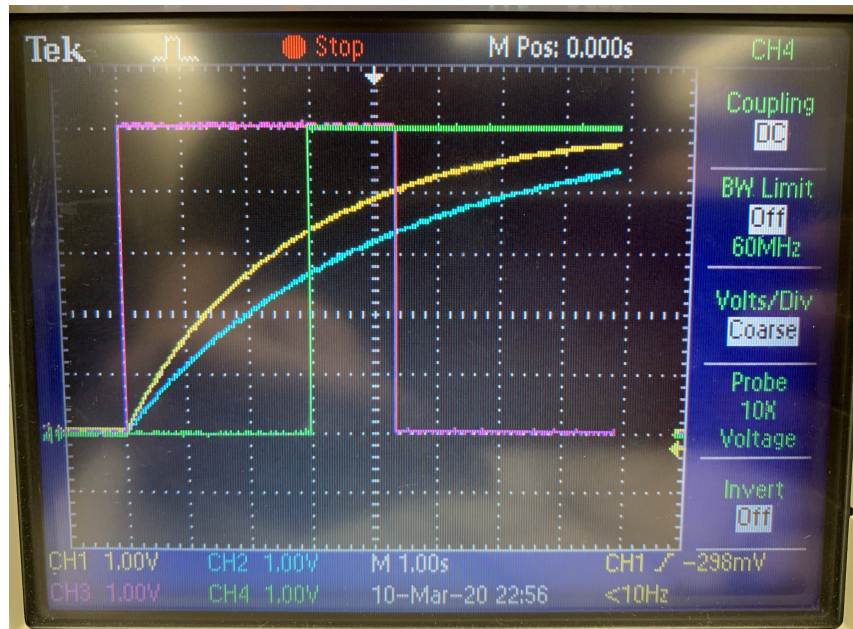
The one contained issue within the pre-built wireless power kit is that the transmitter is designed to power-up, but not immediately initiate power transfer. The user is intended to initiate transfer themselves by pressing in the rotary encoder on the board. This poses a problem for a permanent installation of the system, considering there cannot be any user interaction to initiate transfer once the system is set up. If the system loses power, such as at night, it needs to automatically resume transfer once it receives power again. The easiest fix to this problem is to reprogram the microcontroller on the transmitter to perform the transfer start after a certain delay, rather than a button press. However, reprogramming the XMC1300 family of microcontrollers requires the XMC™ Link debug probe, which comes at an additional \$90 cost [51]. The alternative solution is to create a circuit that would simulate the pressing of the rotary encoder automatically after a short delay. The circuit that is used is denoted as a supplementary control circuit (SCC) in the block diagram and is shown in Figure 78.



**Figure 78:** This design consists of two operational amplifiers, two diodes, and a FET. When power is applied to the system, the circuit creates a ~1 second pulse after a delay of ~3 seconds.

The heart of this control circuit is the LMC6484IN, a simple rail-to-rail operational amplifier [43]. The LMC6484IN is configured as two single supply op-amp comparators, with the rails connected to 5V and Ground. The entire operation is controlled by the interaction of the

timing resistors and capacitors with the non-inverting input of one amplifier, and the inverting input with the other. There is a voltage divider network on the inverting input of one amplifier and the non-inverting input of the other to bias the inputs at  $\sim 3.3\text{V}$ . Switch S1 simulates the circuit receiving power. Two reverse bias diodes are on the outputs of the amplifiers. An oscilloscope readout of the module under operation is shown in Figure 79.

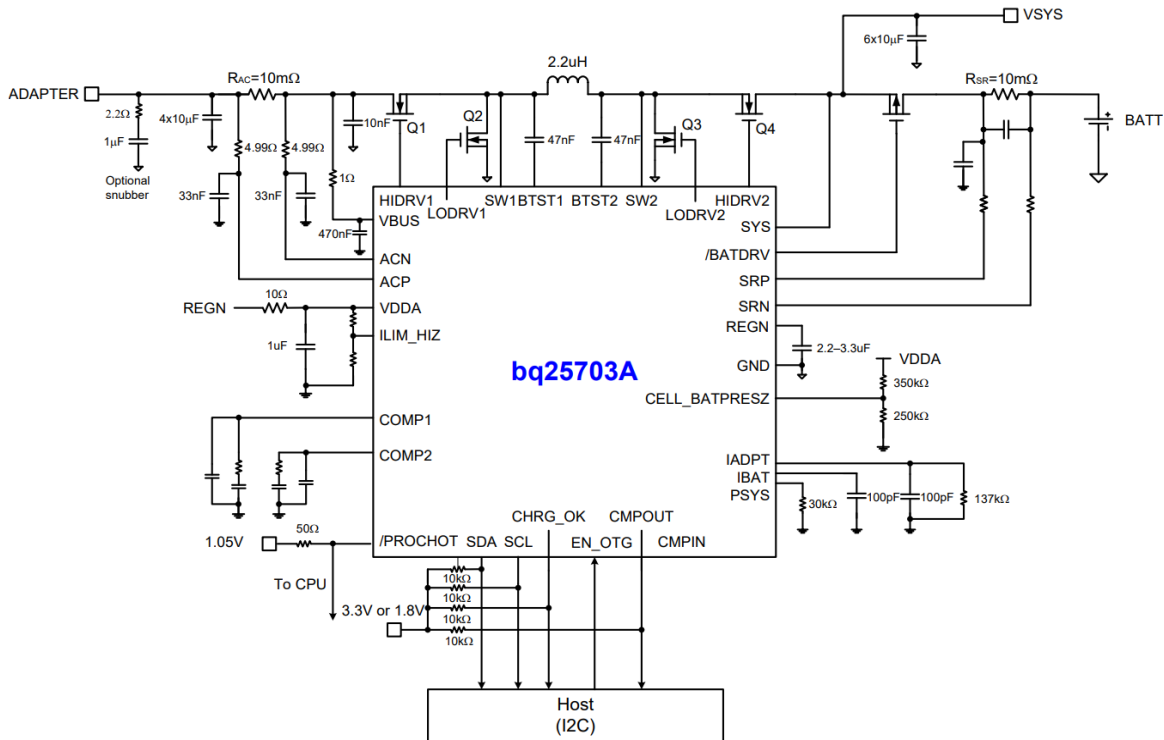


**Figure 79:** Channels 1 and 2 show the voltages on C1 and C2, respectively. Channels 3 and 4 show the LMC6484IN outputs, where there is approximately 1.3 seconds of overlap between the signals.

Once power is applied to the circuit, the voltage on the non-inverting input of U1A begins to increase to  $V_{CC}$  as  $C_1$  is charged through  $R_2$ . Given the equation  $V = V_0 * (1 - e^{-\frac{t}{C_1 R_2}})$ , and solving for  $t$  using the values for  $R_2$  and  $C_1$  in the schematic, the charge time to hit the bias threshold is approximately 3s. The voltage on the inverting input of U2A also begins to increase to  $V_{CC}$  as  $C_2$  is charged through  $R_5$ . Using the same equation, the charge time to hit the bias threshold is approximately 4.3s. While U1A outputs HIGH after 3 seconds, U2A remains high for 4.3 seconds from power-up, then goes LOW. This allows for 1.3 seconds between both output changes where both outputs will be high. During this period of time, there is no current through the pull-up resistor, because there is no path to ground. This causes the voltage on the IRF520N MOSFET gate to jump to 5V and complete the circuit to start power transfer.



The other charge controller operates after the wireless power receiver. This unit is responsible for converting the unregulated DC output of the receiver to a constant voltage output supply and a ~12V charge to and from the battery. The circuit is controlled by the BQ25703a charge management integrated circuit, which is programmed via I<sup>2</sup>C from the Raspberry Pi. The power from the input to the output is regulated by four MOSFETs, which intersect the line from ADAPTER and V<sub>SYS</sub>/BATT shown below in Figure 80 [52]. The BQ25703a is programmed to draw 6A from the wireless power receiver, and provide a constant current out. The constant current draw is important, as it keeps the wireless power receiver operating in a stable state.



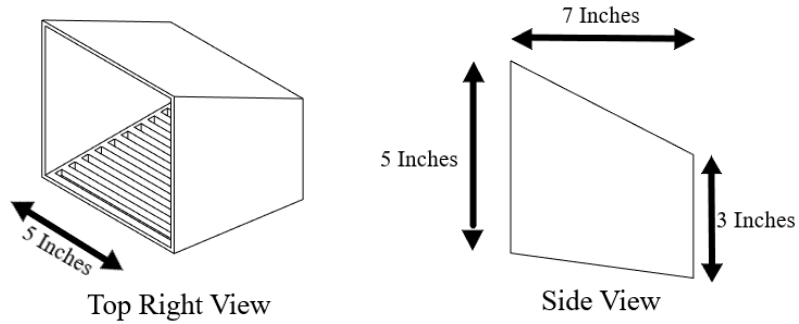
**Figure 80: The main elements of this application schematic are the inductor and the FETs Q1-Q4, which create a buck-boost configuration that can be used to charge a wide range of batteries.**

Unfortunately, after testing and debugging the BQ25703a circuit for a while, the charger would not provide an output to the battery. The chip was receiving successful I<sup>2</sup>C commands, but there was no output to the battery, no matter what was tried. Given the fact that time was running out, it was decided that a second charge controller, identical to the one on the solar side of the system, would be used to replace the BQ25703a. The 20A 300W buck converter comes in handy here, as it allows the user to set an output current limit, a feature that the system using the

BQ25703a would not need, but that this new system element would require to keep charge current below 6A.

Like the solar charge controller, the charger connects the battery straight to the load without regulation. The load for the charger is the display and Raspberry Pi, which both run off of regulated 5V sources. In order to create a 5V source for the load, an additional power stage is required. For this, an additional DC-DC buck converter is used. The Aideepen 20A 300W Adjustable Buck Converter is used once again for this part of the system. The 20A MAX output current, 15A suggested output current is plenty to ensure the display and Raspberry Pi have a regulated 5V input with plenty of current to run everything.

Figure 81 shows a model of the housing for mounting the WPT units on the windows. The model is simply a representation of the actual housing. The housing itself will be thinner than shown in Figure 81 because only a small amount of space is needed to house the WPT units. To mount the housing onto the windows, four suction cups will be used that are made to hold up window mounted bird feeders which are also made out of acrylic frames. Placed between the suction cups will be the receiver or transmitting coils and the attached heat sink that is connected to it. Behind these pieces will be the WPT's circuit board and affiliated circuitry. In the model, the top and bottom of the housing was slanted so that water would not gather anywhere and will drip off of the housing at the outer edge. On the bottom of the housing, vents were added to allow the WPT's circuits to passively cool faster and the originally clear acrylic was changed to mirrored acrylic to prevent heat from sun exposure from gathering. Additionally, to protect the charge controller circuits, batteries, and other electrical components that are housed outside and inside, simple acrylic boxes will be made to house them within. The outside circuitry housing is 12 inches long by 12 inches wide by 24 inches tall. The inside circuitry housing dimensions are the same as the outside circuitry housing, but it is 12 inches tall instead of 24 inches.



**Figure 81:** *The dimensions of this window mount for the WPT housing are 5 inches by 5 inches by 7 inches.*

### 6.3 Display

The power output from the solar panel and WPT, which will be stored in a battery inside the Atwater Kent Laboratories, will be used to light up the LED display. The display will show animations such as weather, temperature, time, and date, and will have features such as motion detection for power conservation. The display will be turned off until an object comes within 23.6 inches, which will help minimize the power draw. The nine modules will be displayed vertically, connected top to bottom in a single line, which will be suspended within an acrylic casing and displayed in the entrance of the Atwater Kent Laboratories.

### 6.4 Chapter Summary

The initial implementation of the system came with many of the initially expected challenges. Multiple parts required supplemental circuitry to be integrated properly, but the overall design of the system did not need to change drastically based on the findings. After making necessary changes the project as a whole works as intended and adheres to the proposed implementation.

## Chapter 7: Verification

Many measures were taken to ensure that the system works properly based on the overall design. This chapter includes the details of troubleshooting the initial implementation as well as the results of any testing done of specific parts of the system to determine its capability. This chapter includes load testing for the solar and WPT and debugging of the display hardware.

### 7.1 Solar

Unlike other aspects of this project, the solar system is entirely off the shelf. Testing entailed making sure the solar panel was operational and can be implemented into the built system at the completion of the project. The solar module was tested at several different loads to prove its operability. The results of this are shown in Table \_\_.

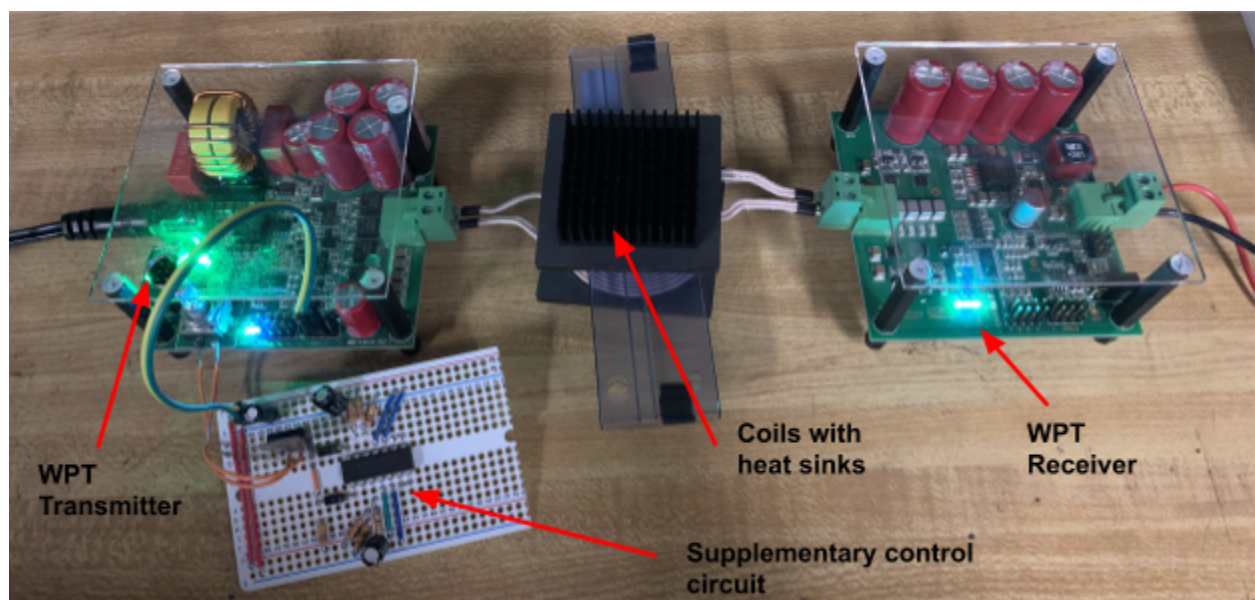
*Table 3: Solar Panel Test Results*

<b>Load</b>	<b>Light cloud at 12PM</b>
None	14.2V
1k $\Omega$	12.8V
10k $\Omega$	13.0V
100k $\Omega$	13.9V

As can be seen from the load testing results, the panel can output a relatively stable voltage even under a wide range of loads. Under a minimal load of 100k $\Omega$ , output voltage drops only slightly of 300mV from no load. Under a heavy load of 1k $\Omega$ , the output voltage drops 1.4V from no load, but still remains well above the operating voltage of the system, as the boost converter requires at least 10.5V to supply power to the WPT. This heavy load is much greater than the expected average load of the system. Even under cloudy conditions, the panel will be able to supply enough power for the system to run.

## 7.2 WPT

Given the fact that the WPT solution for the project is a complete solution, its implementation is fairly easy electronically. The first step of integrating WPT into the system is to verify the operation of the devices. This means performing tests at various input currents and varying the load conditions to determine the total power transferred and the efficiency of the device under load. To do this, a current limited DC power supply and a DC electronic load in constant current (CC) mode is used. In this test, the coils of the transmitter and receiver are aligned and separated by approximately 10mm to simulate real installation conditions. The transmitter is connected to the lab power supply and set at 24V. The receiver is connected to the electronic load and the current is set at 500mA. The load input is turned on, followed by the supply output. The wireless power transfer system under test is shown in Figure 82.



**Figure 82:** *The transmitter has the SCC connected, and the coils have heat sinks to prevent thermal damage. The green and blue lights indicate normal operating conditions. The component tube between the coils is simply used as a non-metallic spacer.*

After transfer starts, values for both input voltage and current, as well as output voltage and current are recorded. Using the input parameters, the input power,  $P_{IN}$ , to the transmitter is calculated using the equation  $P = IV$ . The same is done for output power,  $P_{OUT}$ , using the output parameters. Using the input and output power, the efficiency of the system is computed using the

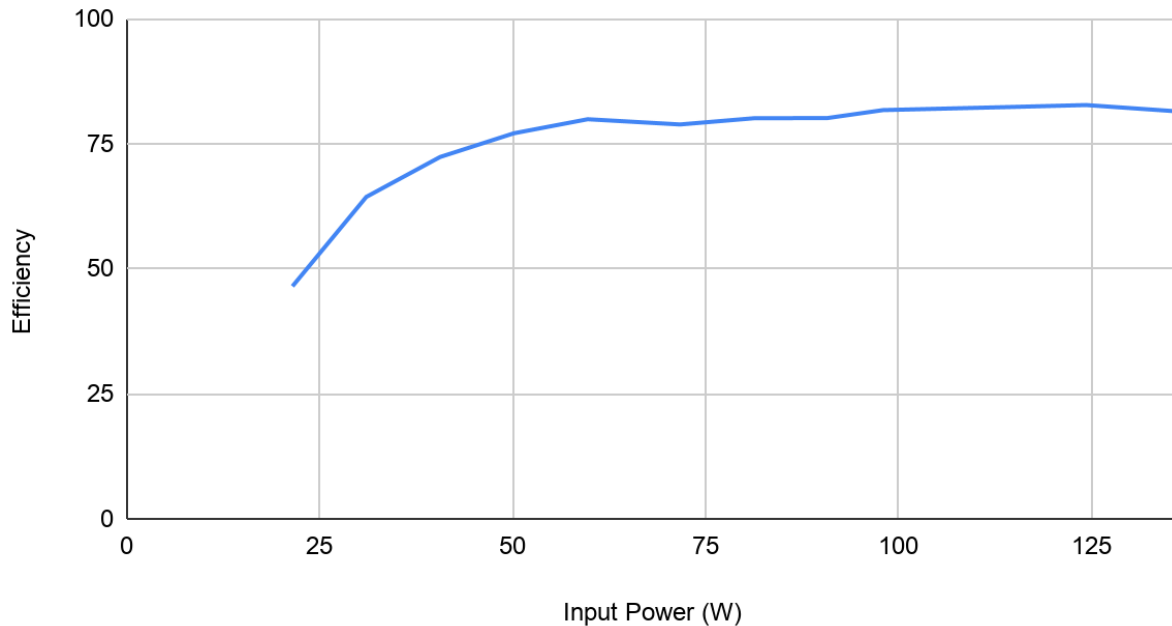
equation  $\eta = \frac{P_{OUT}}{P_{IN}}$ . The results of the test are shown in Table 4, which was carried out in a controlled lab environment with a constant current DC electronic load and current limited power supply.

**Table 4: WPT Load Testing Data**

<b>V<sub>IN</sub> (V)</b>	<b>V<sub>OUT</sub> (V)</b>	<b>I<sub>IN</sub> (A)</b>	<b>I<sub>OUT</sub> (A)</b>	<b>P<sub>IN</sub> (W)</b>	<b>P<sub>OUT</sub> (W)</b>	<b>Efficiency (%)</b>
23.9	18.5	5.7	6	136.23	111	81.47985025
23.9	18.7	5.2	5.5	124.28	102.85	82.75667847
23.9	18.9	4.8	5	114.72	94.5	82.37447699
23.9	17.8	4.1	4.5	97.99	80.1	81.743035
23.9	18.2	3.8	4	90.82	72.8	80.15855538
23.9	18.6	3.4	3.5	81.26	65.1	80.11321683
23.9	18.85	3	3	71.7	56.55	78.87029289
23.9	19.1	2.5	2.5	59.75	47.75	79.91631799
23.9	19.35	2.1	2	50.19	38.7	77.10699342
23.9	19.6	1.7	1.5	40.63	29.4	72.36032488
23.9	20	1.3	1	31.07	20	64.37077567
23.9	20	0.9	0.5	21.51	10	46.49000465

The results for system efficiency are plotted against input power to form the plot shown in Figure 83. Looking at the plot, it can be seen that the system can achieve efficiencies upwards of 80%, which is pretty impressive considering the coils have a centimeter of separation. However, the efficiency rapidly drops off at lower input powers. At lower than 40W, efficiency drops to less than 75%, and at lower than 25W, efficiency starts to drop below 50%. This could be caused by software frequency compensation or ideal hardware ranges located at higher currents, with the general assumption that the 200W kit will be operated at higher power loads most of the time [44].

## Efficiency vs. Input Power



*Figure 83: Efficiency tends to increase with input power. Max efficiency is reached at slightly over 100W.*

Upon testing the module, particularly at higher currents like 6A, parts of the system start to get quite hot. Würth Elektronik recommends that for applications transferring over 150W, air cooling should be used to prevent damage to the components [44]. However, this is not a problem, considering the input power to the system is regulated to 150W. Additionally, 40mm x 40mm aluminum heat sinks are placed on the coils to assist with heat dissipation.

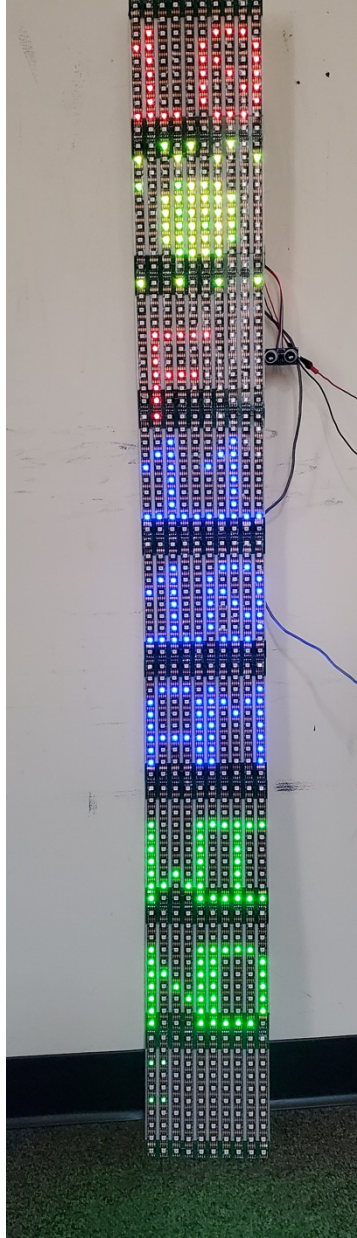
### 7.3 Display

The first error encountered when constructing the display was an error in the PCB design. The Right Main PCB was incorrectly designed so that the signals were not routed correctly between individual strips from left to right. This error led to a significant delay in the progress of constructing the display, as the PCB had to be redesigned and ordered before any module could be completed. The PCBs were successfully redesigned, and weeks later the modules were completed after receiving the new Right Main PCBs.

Troubleshooting the display modules began by testing each of the nine modules individually by using a test program that would turn all the LEDs in the module on sequentially with a single color . In testing the modules for the first time, only one of the nine modules worked completely. Eight modules had a similar problem where only a consecutive few of the ten LED strips would turn on. Many of the modules were detected to have shorts where the ends of the LED strips were soldered to the left and right PCBs. These shorts were unexpected because they were difficult to identify visually, as the shorts occurred where solder had flowed underneath the strip and contacted the pads on the bottom of the strip. Later it was determined that the high frequency of these issues was due to the adhesive on the underside of the strip allowing the solder to easily flow between pads. These errors were resolved by analyzing the signals at each solder joint where the following strip was not activated, and resoldered the joints more carefully so no short would occur.

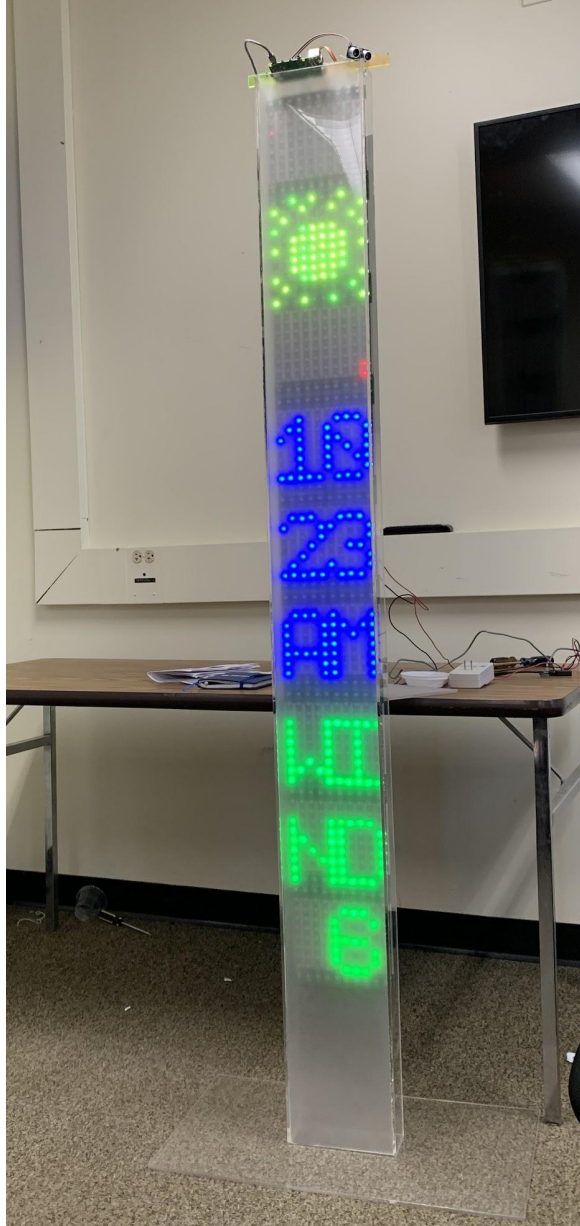
Another, however less frequent, cause of fault in the modules was that a couple LEDs were found to be faulty. This was determined by analyzing the signals output by a single LED and finding that either the data, clock, or both signals had been corrupted by the faulty LED, thus not allowing any subsequent LEDs to function. Of the total 900 LEDs, two were found to be faulty. These LEDs were replaced by replacing the entire strip of which the faulty LED was located. After all hardware errors were remedied, all nine display modules were able to work together to form a continuous display, as seen in Figure 84 below.





*Figure 84: From top to bottom the display shows the following information: date, weather animation, outside temperature, time, wind speed.*

As seen in Figure 85, the display case was changed to a four sided box instead of having open sides. This change was implemented to prevent anyone from touching the circuitry within the case. The new design also provided support for the acrylic holding up the modules.

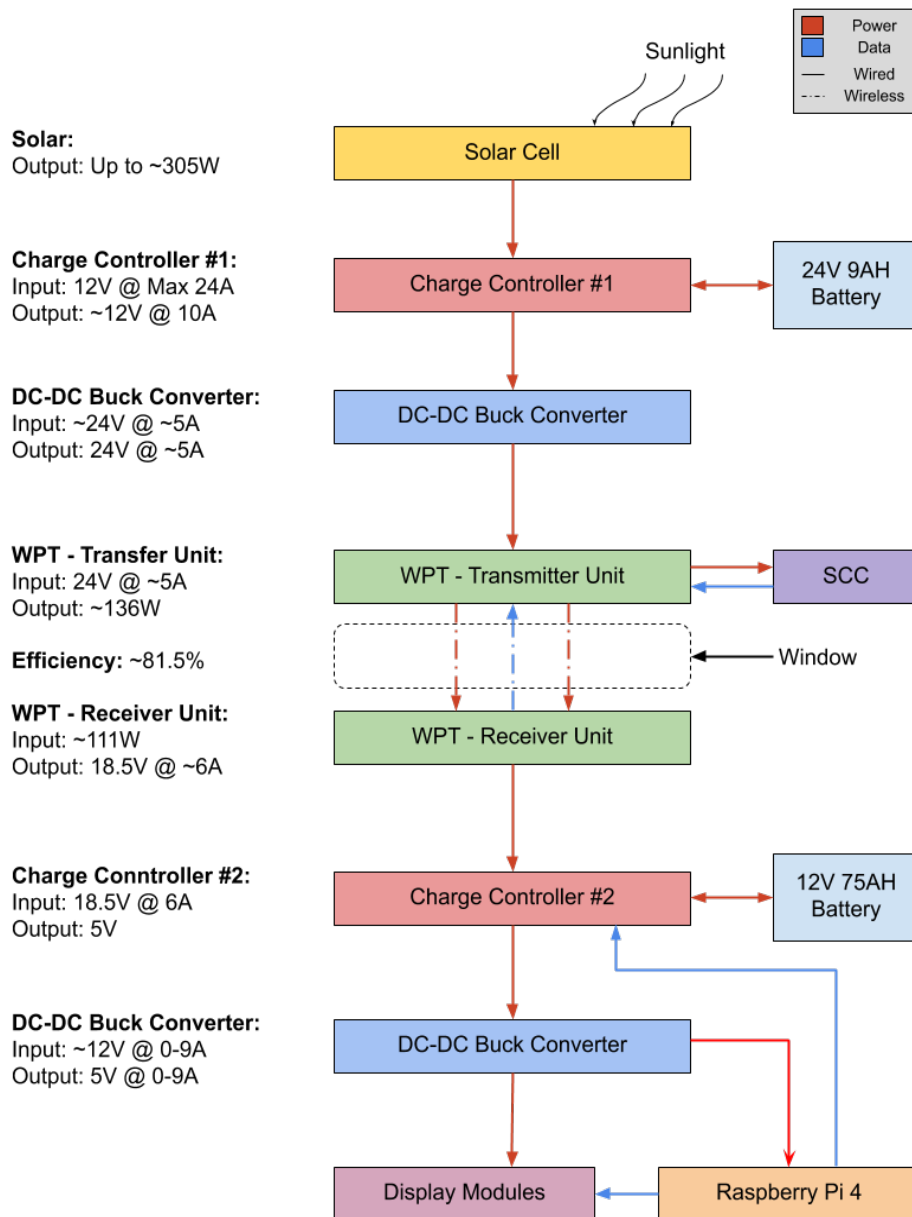


*Figure 85: Picture of the final display case with the LED matrices running the program within it.*

## 7.4 Total Integration

Each part of the system is tested individually to ensure that they work as standalone elements. Then, they are integrated into the full system. The full system block diagram that was used in Chapter 6, but with additional annotations, is shown in Figure 86. Voltage and current inputs and outputs are denoted at each stage of the system. All power flowing through the system

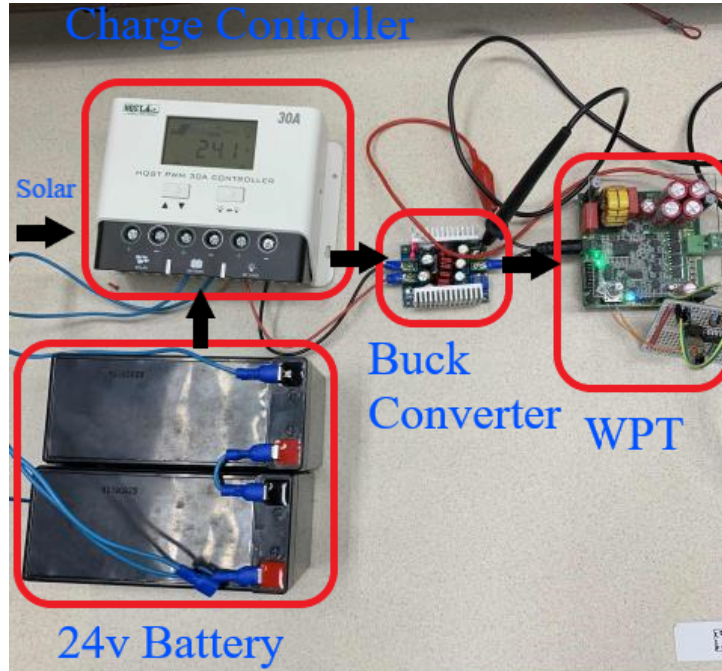
is DC, with the exception of the wireless power transfer system, which has a contained high frequency AC element to drive the coils.



**Figure 86 :** The text on the left indicates the various voltage and current readings at all points of the system. Data transfer is denoted with a blue arrow, while power is denoted with a red arrow.

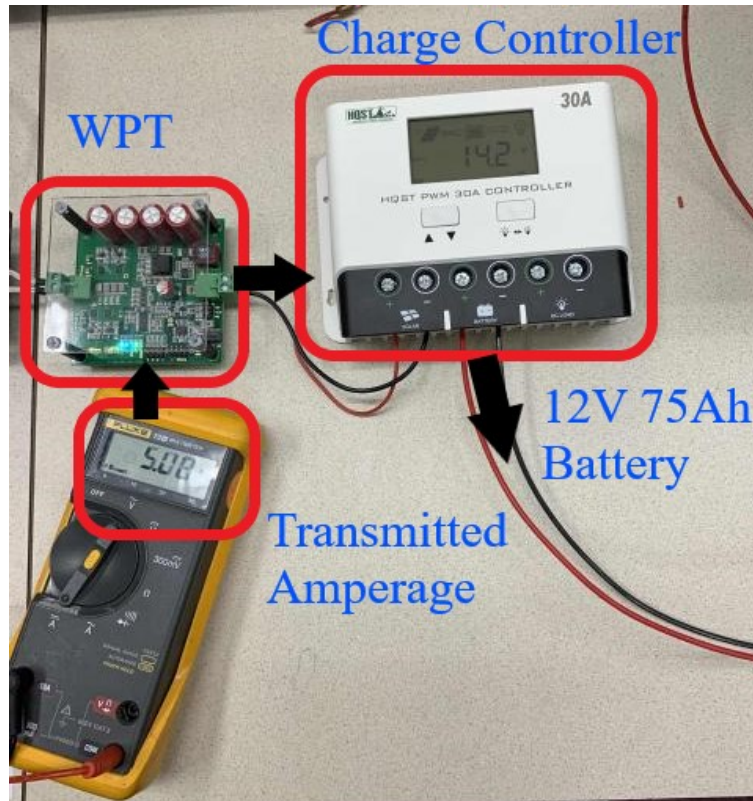
Before integrating every single element of the project into one system, elements are tested in groups of items that grow with each iteration to remove confusion and complexity, and to isolate any errors. For example, the components that would reside on the outside of the

building were tested first. This includes the photovoltaic panel, the charge controller, the 24V 9AH battery, the buck converter, and the wireless power transmitter with supplementary control circuit. Figure 87 illustrates the testing of the “outside” electronics.



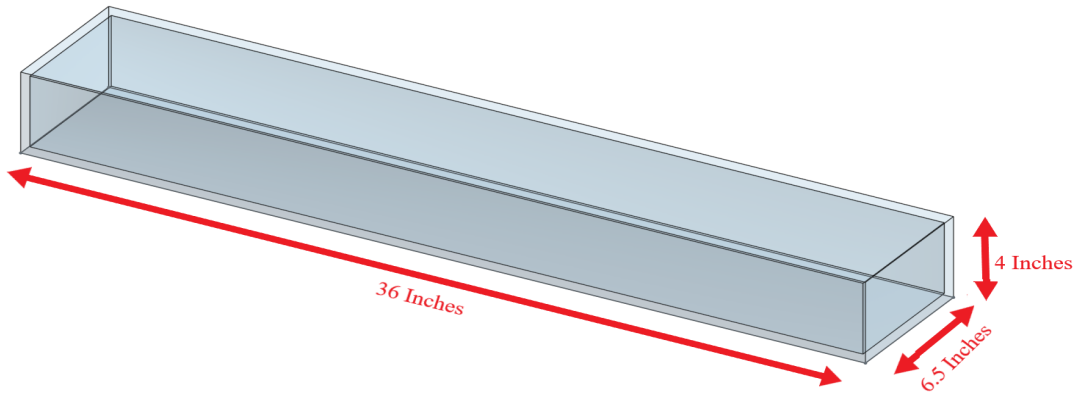
**Figure 87: These components are housed in the outside box and provide the necessary power path to the wireless power transmitter.**

On the inside of the building, the electronics include the wireless power receiver, the second charge controller, the 12V 75AH battery, and the second buck converter. These components are pictured in Figure 88. The second buck converter is not shown, because it is connected in close proximity to the Raspberry Pi and display, in the event that long wires cause large, unwanted voltage drops. Additionally, an ammeter is connected in series with the wireless power input for testing. As shown, the ammeter reads about 5A. This number varies from 4.5A to 6A based on coil alignment and distance, but never exceeds 6A, which would indicate successful transfer testing for this system.

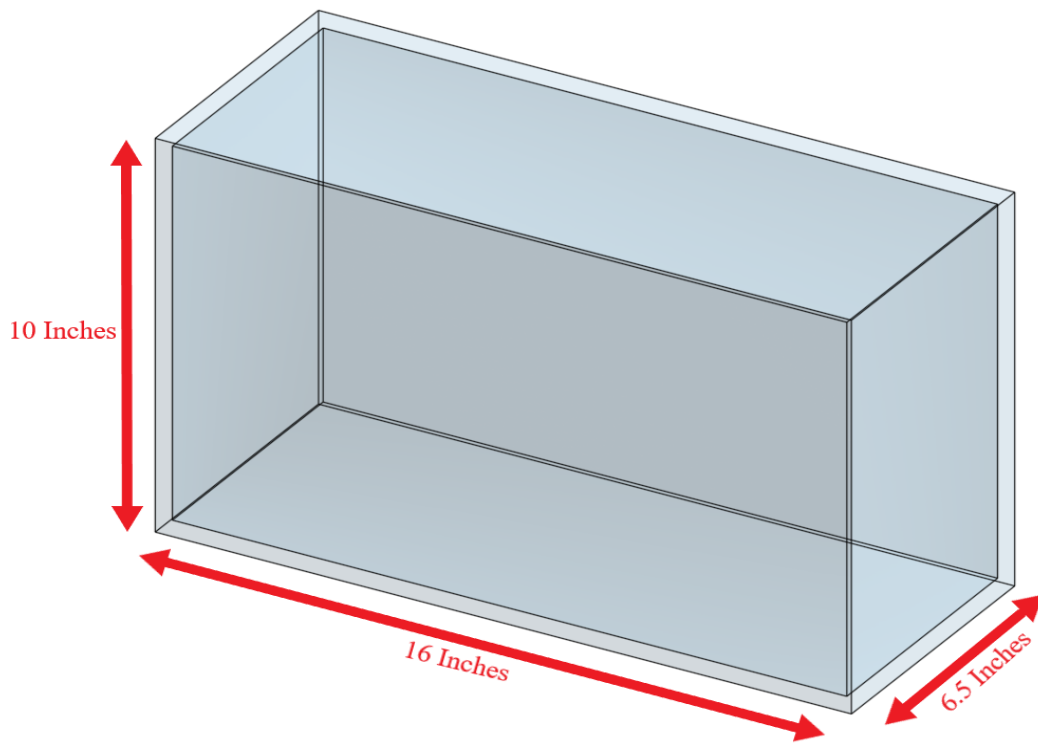


**Figure 88:** *These components are housed in the inside box and provide the necessary power path from the wireless power receiver to the battery and even further, the display.*

Power transfer through the window was inefficient and sometimes unfeasible because the windows of the Atwater Kent Laboratories are thicker than initially anticipated. As a result, facilities would either need to replace the window with a thinner piece of glass or simply wire a power line through the window. It was decided on routing a wire through the window and into a new display casing to the WPT because there are uncertainties with mounting the display case on the window. The new display casing dimensions can be seen in Figure 89. Additionally, a new housing to hold the batteries was created, as seen in Figure 90.



*Figure 89: The circuitry housing is 36 inches long by 6.5 inches wide by 4 inches tall.*



*Figure 90: The battery housing is 16 inches long by 6.5 inches wide by 10 inches tall.*

## 7.5 Chapter Summary

Even with rigorous testing of every component, initial implementation of the system proved to have many problems that needed to be worked out. Several changes were made to ensure everything was working properly. Despite these issues, the overall design of the systems did not change much, as it still proved effective after implementation was complete and the entire system worked as intended.

## Chapter 8: Conclusion

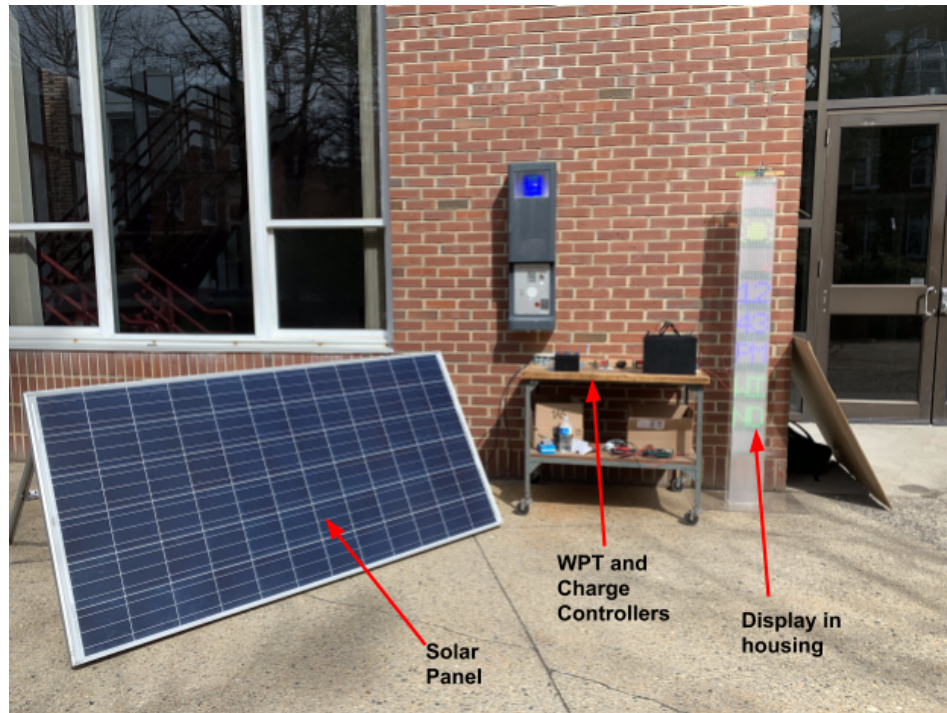
The solar panel works as intended, providing necessary power to the system during daylight hours. Figure 69 shows the panel outside Atwater Kent Laboratories in its final location. It has a maximum output of 305W and an average of approximately 200W, which is more than enough to power the system. The excess power is used to charge a battery which will provide power to the system when the panel is not operational. The panel, which is in new condition has ample longevity, and the only maintenance required is cleaning the surface of the panel about every six months.

As a prebuilt system, the operating characteristics of the WPT component of this project are very defined. The system has an unregulated input, so the power source has to be a 24V regulated power supply. While the system advertises 200W of input power, the actual maximum input power for the stock modules is closer to 150W. To increase to 200W, active cooling is required. At 24V, this works out to 6.25A of input current. The final system is designed to accept a *regulated* 24V 6A input source. The input to the system is a 24V output boost converter, which will act as a regulated constant voltage power supply for the transmitter module. Efficiency of the system, as shown in section 7, increases with input current. At an input of 5.2A and a load current of 5.5A, the system is expected to operate at 83% efficiency. Based on the fact that coil current has an effect on output parameters, the output voltage will likely vary from 18 to 22V. The output of the system feeds into the charger, which has an input voltage range of 0 to 24V, and an input current range of 0 to 6.35A, ideal ranges for these output parameters.

When constructed as intended and provided necessary power, and signal from an external microprocessor, seven of the nine constructed modules work properly to form a single, continuous display. Although not all of the modules that were initially anticipated were acquired, the display still succeeds as a self-contained and working device for showing a wide range of information.

Coupled with the completed software this module display will have a variety of features. Using the openweathermap API, the display should be able to identify different weather conditions. Using this information, the APA102 Python driver will import custom created animation arrays and loop through them to display what the weather is like outside. In addition to this, an ultrasonic sensor will measure pulse data width signals using I<sup>2</sup>C to determine people's

distance from the Raspberry Pi and will display an unlocking animation that corresponds to the distance one is from the display. Once unlocked, the display will show the weather animation, the time, and the date.



***Figure 91: Picture of the final system with the solar panel generating power, the wireless power transfer system transferring power, and the display running the program, completely off-grid.***

There are 4 mounts or housings are the solar mount, the WPT and circuitry housings, battery housing, and the display case. The solar mount, if built as intended, should be able to hold up the solar module at a 45 degree angle and should be staked into the ground after determining a place to construct it. The mount should be into the ground because it will prevent the mount from shifting and it is more aesthetic than putting a plate with sandbags under the solar module.

The WPT and circuitry housing will be inside of Atwater Kent Laboratories. This housing, that also needs to be built, will protect the WPT and other circuitry from getting touched or damaged. To connect the solar panel to the components within this housing, a drilled hole through the building will need to be made to wire the components together in the safest way possible. The battery housing is a simple acrylic box that also needs to be built. They will encompass the batteries to protect them from being tampered with.



The display case will stand on its own, as seen in Figure 91 above along with the rest of the system. To set up the LED matrix within the case, simply slide the matrix down the opening at the top and store the Raspberry Pi in the box attached at the top. The display case will also protect and support the LED matrix and Raspberry Pi from all sides and the front of the display case is sanded down to translucence. The translucence will act as a light diffuser and blur the electronics behind the acrylic. The circuitry won't be viewable unless the viewer looks in from the side.

Overall, all of these displays need to be installed and set up for permanent residence. To judge the cost of setting all of these displays and wiring them all, we consulted Ronald O'Brien. From his estimates, the cost of implementing this system into the building would be around 500 to 800 dollars. This cost will include, drilling through the wall for the wires, setting up the solar mount, and buying and connecting all the wires.

## **8.1 Future Work**

The best way to expand upon this project in the future would be to research and implement an alternate method of WPT. The WPT was the weakest part of the system, only able to transmit a portion of the power generated from the solar panel. To bolster the entire system's reliability and efficiency, the WPT would be the first place to make changes such that it can support a greater power transfer and at a higher efficiency. This means replacing the current WPT system or finding an optimal window to utilize WPT.

The display is optimized for expansion because it is a modular system. Additional display modules could be created to enlarge the display and change its shape. This would also mean the hardware and software for the display could be updated as well. A larger display would make room for more animations and information. The display could also be made interactive, with sensors for detecting human input.

If this project were to be implemented in the future, an optimal location should be found based on where the solar panel will be most efficient, ideally a rooftop. Additionally, the WPT will not work very well through thick exterior windows. If the hardware for WPT technology were to be improved, this would potentially not be an issue. For future work, a thin medium through which to transfer power should be prepared to properly demonstrate the system's uses instead of just showcasing the technology.

## Appendix A: Communications Python Code

```
from smbus2 import SMBus
import time

class SMBusI2C(object):

    def read16(addr, cmd):

        with SMBus(1) as dev:
            dat = dev.read_i2c_block_data(addr, cmd, 2)

        return dat

    def write16(addr, cmd, data):

        with SMBus(1) as dev:
            lsb = data & 0x00FF
            msb = (data & 0xFF00) >> 8

            dev.write_byte_data(addr, cmd, data)
            dev.write_byte_data(addr, cmd + 1, data)

if __name__ == '__main__':

    addr = 0x6B

    time.sleep(1)

    write16(addr, 0x00, 0x020E) #Default ChargeOption Values
    write16(addr, 0x04, 0x3400) #Set MaxChargeVoltage to 12288mV
    write16(addr, 0x02, 0x1000) #Set ChargeCurrent to 512mA
    time.sleep(1)
    current = read16(addr, 0x02)
    voltage = read16(addr, 0x04)

    time.sleep(1)
```

## Appendix B: APA102 Python Library

```
"""This is the main driver module for APA102 LEDs"""
import Adafruit_GPIO as GPIO
import Adafruit_GPIO.SPI as SPI
from math import ceil

RGB_MAP = {'rgb': [3, 2, 1], 'rbg': [3, 1, 2], 'grb': [2, 3, 1],
           'gbr': [2, 1, 3], 'brg': [1, 3, 2], 'bgr': [1, 2, 3]}
```

```
class APA102:
    """
    Driver for APA102 LEDS (aka "DotStar").
```

(c) Martin Erzberger 2016-2018

Public methods are:

- set\_pixel
- set\_pixel\_rgb
- show
- clear\_strip
- cleanup

Helper methods for color manipulation are:

- combine\_color
- wheel

The rest of the methods are used internally and should not be used by the user of the library. This file is the main driver, and is usually used "as is".

Very brief overview of APA102: An APA102 LED is addressed with SPI. The bits are shifted in one by one, starting with the least significant bit.

An LED usually just forwards everything that is sent to its data-in to data-out. While doing this, it remembers its own color and keeps glowing with that color as long as there is power.

An LED can be switched to not forward the data, but instead use the data to change its own color. This is done by sending (at least) 32 bits of zeroes to data-in. The LED then accepts the next correct 32 bit LED frame (with color information) as its new color setting.

After having received the 32 bit color frame, the LED changes color, and then resumes to just copying data-in to data-out.

The really clever bit is this: While receiving the 32 bit LED frame, the LED sends zeroes on its data-out line. Because a color frame is 32 bits, the LED sends 32 bits of zeroes to the next LED.

As seen above, this means that the next LED is now ready to accept a color frame and update its color.

So that's really the entire protocol:

- Start by sending 32 bits of zeroes. This prepares LED 1 to update its color.
- Send color information one by one, starting with the color for LED 1, then LED 2 etc.
- Finish off by cycling the clock line a few times to get all data to the very last LED on the strip

The last step is necessary, because each LED delays forwarding the data a bit. Imagine ten people in a row. When you yell the last color information, i.e. the one for person ten, to the first person in the line, then you are not finished yet. Person one has to turn around and yell it to person 2, and so on. So it takes ten additional "dummy" cycles until person ten knows the color. When you look closer, you will see that not even person 9 knows its own color yet. This information is still with person 2. Essentially the driver sends additional zeroes to LED 1 as long as it takes for the last color frame to make it down the line to the last LED.

""""

# Constants

MAX\_BRIGHTNESS = 31 # Safeguard: Max. brightness that can be selected.

LED\_START = 0b11100000 # Three "1" bits, followed by 5 brightness bits

```
BUS_SPEED_HZ = 8000000 # SPI bus speed; If the strip flickers, lower this value
```

```
def __init__(self, num_led, global_brightness=MAX_BRIGHTNESS,
             order='rgb', mosi=10, sclk=11, bus_speed_hz=BUS_SPEED_HZ,
             ce=None):
    """Initializes the library.

    """
    self.num_led = num_led # The number of LEDs in the Strip
    order = order.lower()
    self.rgb = RGB_MAP.get(order, RGB_MAP['rgb'])
    # Limit the brightness to the maximum if it's set higher
    if global_brightness > self.MAX_BRIGHTNESS:
        self.global_brightness = self.MAX_BRIGHTNESS
    else:
        self.global_brightness = global_brightness

    self.leds = [self.LED_START, 0, 0, 0] * self.num_led # Pixel buffer

    # MOSI 10 and SCLK 11 is hardware SPI, which needs to be set-up differently
    if mosi == 10 and sclk == 11:
        self.spi = SPI.SpiDev(0, 0 if ce is None else ce, bus_speed_hz) # Bus 0
    else:
        self.spi = SPI.BitBang(GPIO.get_platform_gpio(), sclk, mosi, ss=ce)

    def clock_start_frame(self):
        """Sends a start frame to the LED strip.

        This method clocks out a start frame, telling the receiving LED
        that it must update its own color now.
        """
        self.spi.write([0] * 4) # Start frame, 32 zero bits

    def clock_end_frame(self):
        """Sends an end frame to the LED strip.
```

As explained above, dummy data must be sent after the last real colour

information so that all of the data can reach its destination down the line.

The delay is not as bad as with the human example above.

It is only 1/2 bit per LED. This is because the SPI clock line needs to be inverted.

Say a bit is ready on the SPI data line. The sender communicates this by toggling the clock line. The bit is read by the LED and immediately forwarded to the output data line. When the clock goes down again on the input side, the LED will toggle the clock up on the output to tell the next LED that the bit is ready.

After one LED the clock is inverted, and after two LEDs it is in sync again, but one cycle behind. Therefore, for every two LEDs, one bit of delay gets accumulated. For 300 LEDs, 150 additional bits must be fed to the input of LED one so that the data can reach the last LED.

Ultimately, we need to send additional  $\text{numLEDs}/2$  arbitrary data bits, in order to trigger  $\text{numLEDs}/2$  additional clock changes. This driver sends zeroes, which has the benefit of getting LED one partially or fully ready for the next update to the strip. An optimized version of the driver could omit the "clockStartFrame" method if enough zeroes have been sent as part of "clockEndFrame".

```
"""
```

```
# Send reset frame necessary for SK9822 type LEDs
```

```
self.spi.write([0] * 4)
```

```
# Round up num_led/2 bits (or num_led/16 bytes)
```

```
for _ in range((self.num_led + 15) // 16):
```

```
self.spi.write([0x00])
```

```
def clear_strip(self):
```

```
    """ Turns off the strip and shows the result right away. """
```

```
    for led in range(self.num_led):
```

```
        self.set_pixel(led, 0, 0, 0)
```

```
    self.show()
```

```
def set_pixel(self, led_num, red, green, blue, bright_percent=100):
```

```
"""Sets the color of one pixel in the LED stripe.
```

The changed pixel is not shown yet on the Stripe, it is only written to the pixel buffer. Colors are passed individually.

If brightness is not set the global brightness setting is used.

```
"""
```

```
if led_num < 0:
```

```
    return # Pixel is invisible, so ignore
```

```
if led_num >= self.num_led:
```

```
    return # again, invisible
```

```
# Calculate pixel brightness as a percentage of the
```

```
# defined global_brightness. Round up to nearest integer
```

```
# as we expect some brightness unless set to 0
```

```
brightness = ceil(bright_percent * self.global_brightness / 100.0)
```

```
brightness = int(brightness)
```

```
# LED startframe is three "1" bits, followed by 5 brightness bits
```

```
ledstart = (brightness & 0b00011111) | self.LED_START
```

```
start_index = 4 * led_num
```

```
self.leds[start_index] = ledstart
```

```
self.leds[start_index + self.rgb[0]] = red
```

```
self.leds[start_index + self.rgb[1]] = green
```

```
self.leds[start_index + self.rgb[2]] = blue
```

```
def set_pixel_rgb(self, led_num, rgb_color, bright_percent=100):
```

```
    """Sets the color of one pixel in the LED stripe.
```

The changed pixel is not shown yet on the Stripe, it is only written to the pixel buffer.

Colors are passed combined (3 bytes concatenated)

If brightness is not set the global brightness setting is used.

```
"""
```

```
self.set_pixel(led_num, (rgb_color & 0xFF0000) >> 16,
```

```
                (rgb_color & 0x00FF00) >> 8, rgb_color & 0x0000FF,
```

```
                bright_percent)
```

```

def rotate(self, positions=1):
    """ Rotate the LEDs by the specified number of positions.

    Treating the internal LED array as a circular buffer, rotate it by
    the specified number of positions. The number could be negative,
    which means rotating in the opposite direction.
    """
    cutoff = 4 * (positions % self.num_led)
    self.leds = self.leds[cutoff:] + self.leds[:cutoff]

def show(self):
    """Sends the content of the pixel buffer to the strip.

    Todo: More than 1024 LEDs requires more than one xfer operation.
    """
    self.clock_start_frame()
    # xfer2 kills the list, unfortunately. So it must be copied first
    # SPI takes up to 4096 Integers. So we are fine for up to 1024 LEDs.
    self.spi.write(list(self.leds))
    self.clock_end_frame()

def cleanup(self):
    """Release the SPI device; Call this method at the end"""

    self.spi.close() # Close SPI port

    @staticmethod
    def combine_color(red, green, blue):
        """Make one 3*8 byte color value."""

        return (red << 16) + (green << 8) + blue

def wheel(self, wheel_pos):
    """Get a color from a color wheel; Green -> Red -> Blue -> Green"""

    if wheel_pos > 255:

```



```
wheel_pos = 255 # Safeguard
if wheel_pos < 85: # Green -> Red
    return self.combine_color(wheel_pos * 3, 255 - wheel_pos * 3, 0)
if wheel_pos < 170: # Red -> Blue
    wheel_pos -= 85
    return self.combine_color(255 - wheel_pos * 3, 0, wheel_pos * 3)
# Blue -> Green
wheel_pos -= 170
return self.combine_color(0, wheel_pos * 3, 255 - wheel_pos * 3)

def dump_array(self):
    """For debug purposes: Dump the LED array onto the console."""

    print(self.leds)
```

## Appendix C: Led Matrix Driver, Matrix.py

```
from time import sleep
from apa102_pi.driver import apa102
from threading import Thread
from characters import abcLib
import requests, JSON
from datetime import datetime
import RPi.GPIO as GPIO
import time

class matrix:
    def get_coord(coord,led_num):
        col_const = led_num//10
        led_index = (9-coord[0])*col_const+abs((((9-coord[0])%2)*col_const)-coord[1])-1
        if coord[0]%2 ==1:
            led_index = led_index+1
        return led_index
    """
    coord = get_coord(8,39,led_num)
    strip.set_pixel_rgb(coord, 0xFF0000)
    strip.show()
    """
class rain(Thread):
    def __init__(self,led_num,strip):
        self.led_num = led_num
        self.strip = strip
        super(rain, self).__init__()
    def run(self):
        cloud = [(1,10),(2,10),(3,10),(4,10),(5,10),(0,11),(6,11),(7,11),
                (8,11),(0,12),(5,12),(6,12),(9,12),(0,13),(5,13),(9,13),
                (0,14),(9,14),(0,15),(9,15),(0,16),(9,16),(1,17),(2,17),
                (3,17),(4,17),(5,17),(6,17),(7,17),(8,17),]
```

```

rain = [(2,18),(4,18),(6,18),(8,18),(1,19),(3,19),(5,19),(7,19),
        (0,20),(2,20),(4,20),(6,20)]
shift = 0
for pos in cloud:
    off = pos[1]
    offset = off+shift
    coord = (pos[0],offset)
    pixel = matrix.get_coord(coord,self.led_num)
    self.strip.set_pixel_rgb(pixel, 0xC0C0C0)
    sleep(0.025)
    self.strip.show()
while True:
    global stop_animate
    for pos in rain:
        off = pos[1]
        offset = off+shift
        coord = (pos[0],offset)
        pixel = matrix.get_coord(coord,self.led_num)
        self.strip.set_pixel_rgb(pixel, 0x0000FF)
        sleep(0.025)
        self.strip.show()
    for pos in rain:
        off = pos[1]
        offset = off+shift
        coord = (pos[0],offset)
        pixel = matrix.get_coord(coord,self.led_num)
        self.strip.set_pixel(pixel,0,0,0)
        self.strip.show()
    if stop_animate:
        break

```

```

class sunshine(Thread):
    def __init__(self, led_num, strip):
        self.led_num = led_num

```

```

self.strip = strip
super(sunshine, self).__init__()
def run(self):
sun = [(3,13),(4,13),(5,13),(6,13),(2,14),(3,14),(4,14),
        (5,14),(6,14),(7,14),(2,15),(3,15),(4,15),(5,15),(6,15),
        (7,15),(2,16),(3,16),(4,16),(5,16),(6,16),(7,16),(2,17),
        (3,17),(4,17),(5,17),(6,17),(7,17),(3,18),(4,18),(5,18),
        (6,18)]
shine = [(0,15),(9,18),(9,15),(0,18),(9,13),(1,19),(1,12),(8,12),(8,19),
(0,11),(3,11),(6,11),(9,11),(9,20),(6,20),(3,20),(0,20),
(2,10),(4,10),(7,10),(0,13),(0,20),(3,20),(6,20),(9,20)]
shift = 0
for pos in sun:
off = pos[1]
offset = off+shift
coord = (pos[0],offset)
pixel = matrix.get_coord(coord,self.led_num)
self.strip.set_pixel_rgb(pixel, 0xFFFF00)
sleep(0.025)
self.strip.show()
while True:
global stop_animate
for pos in shine:
off = pos[1]
offset = off+shift
coord = (pos[0],offset)
pixel = matrix.get_coord(coord,self.led_num)
self.strip.set_pixel_rgb(pixel, 0xFFFF00)
sleep(0.025)
self.strip.show()
for pos in shine:
off = pos[1]
offset = off+shift
coord = (pos[0],offset)

```

```

    pixel = matrix.get_coord(coord,self.led_num)
    self.strip.set_pixel(pixel,0,0,0)
    self.strip.show()
    if stop_animate:
        break
class moon(Thread):
    def __init__(self, led_num, strip):
        self.led_num = led_num
        self.strip = strip
        super(moon, self).__init__()
    def run(self):
        moon = [(3,12),(4,12),(5,12),(6,12),(2,13),(3,13),(4,13),
(5,13),(6,13),(7,13),(1,14),(2,14),(3,14),(4,14),(5,14),(6,14),
(7,14),(8,14),(1,15),(2,15),(3,15),(4,15),(5,15),(6,15),
(7,15),(8,15),(1,16),(2,16),(3,16),(4,16),(5,16),(6,16),
(7,16),(8,16),(1,16),(3,17),(2,17),(3,17),(4,17),(5,17),(6,17),
(7,17),(3,18),(4,18),(5,18),(6,18)]
        stars = [(6,12),(5,13),(6,13),(7,13),(4,14),(5,14),(6,14),(7,14),(8,14),
(4,15),(5,15),(6,15),(7,15),(8,15),(4,16),(5,16),(6,16),
(7,16),(8,16),(5,17),(6,17),(7,17),(6,18)]
        shift = 0
        for pos in moon:
            off = pos[1]
            offset = off+shift
            coord = (pos[0],offset)
            pixel = matrix.get_coord(coord,self.led_num)
            self.strip.set_pixel_rgb(pixel, 0xA9A9A9)
            sleep(0.025)
            self.strip.show()

        while True:
            global stop_animate
            for pos in stars:
                off = pos[1]

```

```

offset = off+shift
coord = (pos[0],offset)
pixel = matrix.get_coord(coord,self.led_num)
self.strip.set_pixel_rgb(pixel, 0xA9A9A9)
sleep(0.025)
self.strip.show()
sleep(1)
for pos in stars:
    off = pos[1]
    offset = off+shift
    coord = (pos[0],offset)
    pixel = matrix.get_coord(coord,self.led_num)
    self.strip.set_pixel(pixel,0,0,0)
    self.strip.show()
    sleep(1)
if stop_animate:
    break

```

```

class cloud(Thread):
    def __init__(self, led_num, strip):
        self.led_num = led_num
        self.strip = strip
        super(cloud, self).__init__()
    def run(self):
        cloud= [(1,10),(2,10),(3,10),(4,10),(5,10),(0,11),(6,11),(7,11),(8,11),
                (0,12),(5,12),(6,12),(9,12),(0,13),(5,13),(9,13),(0,14),(9,14),
                (0,15),(9,15),(0,16),(9,16),(0,16),(9,16),(1,18),(2,18),(3,18),
                (4,18),(5,18),(6,18),(7,18),(8,18)]
        shift = 0
        for pos in cloud:
            off = pos[1]
            offset = off+shift
            coord = (pos[0],offset)

```

```

pixel = matrix.get_coord(coord,self.led_num)
self.strip.set_pixel_rgb(pixel, 0xC0C0C0)
sleep(0.025)
self.strip.show()

```

```

class partcloud(Thread):

```

```

    def __init__(self, led_num, strip):

```

```

        self.led_num = led_num

```

```

        self.strip = strip

```

```

        super(partcloud, self).__init__()

```

```

    def run(self):

```

```

        cloud= [(2,14),(3,14),(4,14),(5,14),(2,15),(6,15),(7,15),
(8,15),(2,16),(5,16),(6,16),(9,16),(2,17),(5,17),
(9,17),(2,18),(9,18),(3,19),(4,19),(5,19),(6,19),
(7,19),(8,19)]

```

```

        sun = [(0,15),(1,15),(0,14),(1,14),(0,16),(1,16),(1,17),(0,18),
(0,10),(4,10),(1,11),(3,11),(5,11),(0,12),(4,12),(1,13),
(2,13),(3,13)]

```

```

        shift = 0

```

```

        for pos in cloud:

```

```

            off = pos[1]

```

```

            offset = off+shift

```

```

            coord = (pos[0],offset)

```

```

            pixel = matrix.get_coord(coord,self.led_num)

```

```

            self.strip.set_pixel_rgb(pixel, 0xC0C0C0)

```

```

            sleep(0.025)

```

```

            self.strip.show()

```

```

        while True:

```

```

            global stop_animate

```

```

            for pos in sun:

```

```

                off = pos[1]

```

```

offset = off+shift
coord = (pos[0],offset)
pixel = matrix.get_coord(coord,self.led_num)
self.strip.set_pixel_rgb(pixel, 0xFFFF00)
sleep(0.025)
self.strip.show()
for pos in sun:
    off = pos[1]
    offset = off+shift
    coord = (pos[0],offset)
    pixel = matrix.get_coord(coord,self.led_num)
    self.strip.set_pixel(pixel,0,0,0)
    self.strip.show()
if stop_animate:
    break
class nightpartcloud(Thread):
    def __init__(self, led_num, strip):
        self.led_num = led_num
        self.strip = strip
        super(nightpartcloud, self).__init__()
    def run(self):
        cloud= [(2,14),(3,14),(4,14),(5,14),(2,15),(6,15),(7,15),
(8,15),(2,16),(5,16),(6,16),(9,16),(2,17),(5,17),
(9,17),(2,18),(9,18),(3,19),(4,19),(5,19),(6,19),
(7,19),(8,19),(7,10),(8,10),(9,10),(7,11),(8,11),
(9,11),(7,12),(8,12),(9,12)]

        shift = 0
        for pos in cloud:
            off = pos[1]
            offset = off+shift
            coord = (pos[0],offset)
            pixel = matrix.get_coord(coord,self.led_num)
            self.strip.set_pixel_rgb(pixel, 0xC0C0C0)

```



```
sleep(0.025)
self.strip.show()
```

```
class snow(Thread):
```

```
    def __init__(self, led_num, strip):
        self.led_num = led_num
        self.strip = strip
        super(snow, self).__init__()
    def run(self):
        cloud = [(1,10),(2,10),(3,10),(4,10),(5,10),(0,11),(6,11),(7,11),
                (8,11),(0,12),(5,12),(6,12),(9,12),(0,13),(5,13),(9,13),
                (0,14),(9,14),(0,15),(9,15),(0,16),(9,16),(1,17),(2,17),
                (3,17),(4,17),(5,17),(6,17),(7,17),(8,17),]
        snow = [(1,18),(3,18),(5,18),(7,18),(2,19),(4,19),(6,19),(8,19),
                (1,20),(3,20),(5,20),(7,20),(9,20),(0,21),(2,21),(4,21),
                (6,21),(8,21),(1,21),(3,21),(5,21),(7,21),(9,21),(0,20),
                (2,20),(4,20),(6,20),(8,20),(0,19),(1,19),(3,19),(5,19),
                (7,19),(9,19)]
        shift = 0
        for pos in cloud:
            off = pos[1]
            offset = off+shift
            coord = (pos[0],offset)
            pixel = matrix.get_coord(coord,self.led_num)
            self.strip.set_pixel_rgb(pixel, 0xD6ECEF)
            sleep(0.025)
            self.strip.show()
        while True:
            global stop_animate
            for pos in snow:
                off = pos[1]
                offset = off+shift
                coord = (pos[0],offset)
                pixel = matrix.get_coord(coord,self.led_num)
```

```

self.strip.set_pixel_rgb(pixel, 0xC0C0C0)
sleep(0.1)
self.strip.show()
for pos in snow:
    off = pos[1]
    offset = off+shift
    coord = (pos[0],offset)
    pixel = matrix.get_coord(coord,self.led_num)
    self.strip.set_pixel(pixel,0,0,0)
    self.strip.show()
    if stop_animate:
        break
class staticText(Thread):
    def __init__(self,strip,test_str,y_shift,led_num,color):
        self.strip = strip
        self.test_str = test_str
        self.shift = 5
        self.y_shift = y_shift
        self.led_num = led_num
        self.color = color
        super(staticText, self).__init__()

    def run(self):
        l = abcLib().abc
        pxl_str = []
        pxl_str.extend(l[self.test_str[0].upper()])
        index = 1
        # shift is used to separate different letters into a full string
        #Build an array of pixels to display full string
        while len(self.test_str)>index:
            pixels = l[self.test_str[index].upper()]
            # Account for shift when degree sign is used
            pixels = [(pixel[0]+self.shift*index ,pixel[1]) for pixel in pixels]
            pxl_str.extend(pixels)

```

```

index+=1
down = self.y_shift
pxl_str = [(pixel[0],pixel[1]+down) for pixel in pxl_str]
for led in pxl_str:
    led_index = matrix.get_coord(led,led_num)
    self.strip.set_pixel_rgb(led_index,self.color)
    sleep(0.025)
    self.strip.show()
class thunder(Thread):

    def __init__(self, led_num, strip):
        self.led_num = led_num
        self.strip = strip
        super(thunder, self).__init__()

    def run(self):
        cloud= [(1,10),(2,10),(3,10),(4,10),(5,10),(0,11),(6,11),(7,11),(8,11),
(0,12),(5,12),(6,12),(9,12),(0,13),(5,13),(9,13),(0,14),(9,14),
(0,15),(9,15),(0,16),(9,16),(0,16),(9,16),(1,17),(2,17),(7,17),
(8,17)]
        bolt = [(5,15),(6,15),(4,16),(5,16),(3,17),(4,17),(5,17),(6,17),(4,18),
(5,18),(3,19),(4,19),(2,20),(3,20),(1,21)]
        shift = 0
        for pos in cloud:
            off = pos[1]
            offset = off+shift
            coord = (pos[0],offset)
            pixel = matrix.get_coord(coord,self.led_num)
            self.strip.set_pixel_rgb(pixel, 0xD6ECEF)
            sleep(0.025)
            self.strip.show()
        while True:
            global stop_animate
            for pos in bolt:

```

```

    off = pos[1]
    offset = off+shift
    coord = (pos[0],offset)
    pixel = matrix.get_coord(coord,self.led_num)
    self.strip.set_pixel_rgb(pixel, 0xFFFF00)
    self.strip.show()
    for pos in bolt:
        off = pos[1]
        offset = off+shift
        coord = (pos[0],offset)
        pixel = matrix.get_coord(coord,self.led_num)
        self.strip.set_pixel(pixel,0,0,0)
        self.strip.show()
    if stop_animate:
        break
class fog(Thread):

    def __init__(self, led_num, strip):
        self.led_num = led_num
        self.strip = strip
        super(fog, self).__init__()

    def run(self):
        right = [(0,13),(1,13),(2,13),(3,13),(4,13),(5,13),
        (8,13),(9,13),(0,17),(1,17),(2,17),(3,17),
        (4,17),(5,17),(6,17),(7,17),(0,21),(1,21),
        (2,21),(3,21),(4,21),(5,21),(8,21),(9,21)]
        left = [(2,10),(3,10),(4,10),(5,10),(6,10),(7,10),
        (8,10),(9,10),(0,15),(1,15),(4,15),(5,15),
        (6,15),(7,15),(8,15),(9,15),(2,19),(3,19),
        (4,19),(5,19),(6,19),(7,19),(8,19),(9,19)]
        together = []
        together.extend(right)
        together.extend(left)

```

```

for pos in together:
    pixel = matrix.get_coord(pos,self.led_num)
    self.strip.set_pixel_rgb(pixel,0xC0C0C0)
    self.strip.show()
    s = 0
    left = [(pixel[0]+9,pixel[1]) for pixel in left]
    right = [(pixel[0]-9,pixel[1]) for pixel in right]
    while True:
        global stop_animate
        clear = []
        for led in left:
            move_l = led[0]-s
            led_index = matrix.get_coord((move_l,led[1]),led_num)
            clear.append((move_l,led[1]))
            self.strip.set_pixel_rgb(led_index,0xC0C0C0)
        for led in right:
            move_r = led[0]+s
            led_index = matrix.get_coord((move_r,led[1]),led_num)
            clear.append((move_r,led[1]))
            self.strip.set_pixel_rgb(led_index,0xC0C0C0)
        s+=1

    if move_r == 19:
        s=0
        self.strip.show()
        sleep(.25)
        for c in clear:
            self.strip.set_pixel(matrix.get_coord(c,led_num),0,0,0)
        self.strip.show()
        if stop_animate:
            self.strip.clear_strip()
        break

```

```

class scrollText(Thread):

    def __init__(self,strip,test_str, y_shift, led_num,color):
        self.strip = strip
        self.test_str = test_str
        self.shift = 6
        self.y_shift = y_shift
        self.led_num = led_num
        self.color = color
        super(scrollText, self).__init__()

    def run(self):

        l = abcLib().abc
        pxl_str = []
        pxl_str.extend(l[self.test_str[0].upper()])
        index = 1
        # shift is used to separate different letters into a full string
        #Build an array of pixels to display full string
        while len(self.test_str)>index:
            pixels = l[self.test_str[index].upper()]
            # Account for shift when degree sign is used
            if self.test_str[index]=='^':
                self.shift = 6
            elif self.test_str[index-1]=='^':
                self.shift = 5
            else:
                self.shift = self.shift
            pixels = [(pixel[0]+self.shift*index ,pixel[1]) for pixel in pixels]
            pxl_str.extend(pixels)

```

```

index+=1
s = 0
temp_s = -1
down = self.y_shift
pxl_str = [(pixel[0]+9,pixel[1]+down) for pixel in pxl_str]
while True:
clear = []
for led in pxl_str:
move = led[0]-s
led_index = matrix.get_coord((move,led[1]),led_num)
clear.append((move,led[1]))
self.strip.set_pixel_rgb(led_index,self.color)
s+=1
temp_s +=1
if move == -1:
s = 0
self.strip.show()
sleep(.15)
for c in clear:
self.strip.set_pixel(matrix.get_coord(c,led_num),0,0,0)
self.strip.show()
"""
for led in pxl_str:
coord = matrix.get_coord(led, self.led_num)
self.strip.set_pixel(coord,0, 0, 0)
self.strip.show()
"""
if stop_animate:
break
class Weather():
def __init__(self, city):
self.city = city
def getWeather(self):
# Enter your API key here

```

```

api_key = "7595ed19467dfdb3d453eb78b5790d23"

# base_url variable to store url
base_url = "http://api.openweathermap.org/data/2.5/weather?"
city_name = self.city

# complete_url variable to store
# complete url address
complete_url = base_url + "appid=" + api_key + "&q=" + city_name

# get method of requests module
# return response object
response = requests.get(complete_url)

# JSON method of response object
# convert JSON format data into
# python format data
x = response.json()
#print(x)

# Now x contains list of nested dictionaries
# Check the value of "cod" key is equal to
# "404", means city is found otherwise,
# city is not found
if x["cod"] != "404":

# store the value of "main"
# key in variable y
y = x["main"]

# store the value corresponding
# to the "temp" key of y
current_temperature = y["temp"]

```



```

# store the value corresponding
# to the "pressure" key of y
current_pressure = y["pressure"]

# store the value corresponding
# to the "humidity" key of y
current_humidity = y["humidity"]

wind = x["wind"]["speed"]
# convert from meters/sec to mph
wind = wind*2.237
wind = str(int(wind))+ "mph"

# store the value of "weather"
# key in variable z
z = x["weather"]

# store the value corresponding
# to the "description" key at
# the 0th index of z

#convert from kelvin to Farenheit
current_temperature = (current_temperature-273.15)*(9/5)+32
current_temperature = int(current_temperature)
temperature_str = str(current_temperature)+'^F'
weather_description = z[0]["description"]

# Build list of info to return back to the matrix
weather_info = [temperature_str, weather_description, wind,
                x["weather"][0]["main"],x["weather"][0]["icon"]]

return weather_info

else:

```

```

    print(" City Not Found ")
class echoSens():

    def __init__(self):
        #GPIO Mode (BOARD / BCM)
        GPIO.setmode(GPIO.BCM)

        #set GPIO Pins
        self.GPIO_TRIGGER = 15
        self.GPIO_ECHO = 18

        #set GPIO direction (IN / OUT)
        GPIO.setup(self.GPIO_TRIGGER, GPIO.OUT)
        GPIO.setup(self.GPIO_ECHO, GPIO.IN)
        #self.distance = 0

    def distance(self):
        # set Trigger to HIGH
        GPIO.output(self.GPIO_TRIGGER, True)

        # set Trigger after 0.01ms to LOW
        sleep(0.00001)
        GPIO.output(self.GPIO_TRIGGER, False)

        StartTime = time.time()
        StopTime = time.time()

        # save StartTime
        while GPIO.input(self.GPIO_ECHO) == 0:
            StartTime = time.time()

        # save time of arrival
        while GPIO.input(self.GPIO_ECHO) == 1:
            StopTime = time.time()

```

```

# time difference between start and arrival
TimeElapsed = StopTime - StartTime
# multiply with the sonic speed (34300 cm/s)
# and divide by 2, because there and back
distance = (TimeElapsed * 34300) / 2

return distance

```

```

class fill():
    def __init__(self, strip, led_num, dist, color):
        self.color = color
        self.dist = dist
        self.strip = strip
        self.led_num = led_num
        # build mass of leds to fill the matrix
        flow = 90-dist
        #print(flow)
        self.fill = [(i,j) for i in range(10) for j in range(flow,90)]

    def light(self):
        for led in self.fill:
            led_index = matrix.get_coord(led, self.led_num)
            self.strip.set_pixel_rgb(led_index, self.color)
            #sleep(0.01)
            strip.show()
            sleep(2)
            strip.clear_strip()

```

```

if __name__ == '__main__':

```

```

led_num = 900
strip = apa102.APA102(num_led=led_num, global_brightness=1, order='rgb')
display = False
#fill(strip,led_num,20,0xFF0000).light()

#### Testing for filling module with distance??
"""
while True:
    dist = echoSens().distance()
    print ("Measured Distance = %.1f cm" % dist)
    print(int(dist))
    if int(dist)<70:
        #test_num = (int(dist)//10)*10
        test_num = ((round(60/int(dist))*10))
        print("this: ", test_num)
        fill(strip,led_num,test_num,0xFF000).light()

    time.sleep(1)
"""
while True:
    while True:
        dist = echoSens().distance()
        print ("Measured Distance = %.1f cm" % dist)
        print(int(dist))
        if int(dist)<60:
            display = True
            break
        sleep(0.5)

    if display is True:
        now = datetime.now()
        date = now.strftime("%b-%d")
        current_time = now.strftime("%H:%M")
        noon = "am"

```

```

hour = current_time.split(":")[0]
if int(hour)>12:
    noon = "pm"
    hour = str(int(hour)-12)
if (int(hour)-12)<10:
    hour = "0" + hour
elif int(hour)==0:
    hour = "12"
    noon = "am"
else:
    pass
minute = current_time.split(":")[1]
stop_animate = False

city = 'worcester'
weather = Weather(city).getWeather()
temperature = weather[0]
temp_scroll = scrollText(strip,temperature,22,led_num,0xFF0000)
date_scroll = scrollText(strip,date,0,led_num,0xFF0000)
dis_hour = staticText(strip,hour,30,led_num,0x0000FF)
dis_min= staticText(strip,minute,40,led_num,0x0000FF)
dis_noon = staticText(strip,noon,50,led_num,0x0000FF)
wind_top = staticText(strip,"wi",61,led_num,0x00FF00)
wind_bottom = staticText(strip,"nd",71,led_num,0x00FF00)
wind = scrollText(strip,weather[2],80,led_num,0x00FF00)

# Adjust values to test different conditions
"""
weather[1] = "scattered clouds: 25-50%"
weather[3] = 'Dust'
weather[4] = '01d'
"""
if weather[3]=='Fog':
    animation = fog(led_num,strip)

```

```

elif weather[3]=='Thunderstorm':
    animation = thunder(led_num,strip)

elif weather[3]=='Drizzle' or weather[3] == 'Rain':
    animation = rain(led_num,strip)

elif weather[3] == 'Snow':
    animation = snow(led_num,strip)

elif weather[3] == 'Clear':
    if 'n' in weather[4]:
        animation = moon(led_num,strip)
    else:
        animation = sunshine(led_num,strip)

elif weather[3] == 'Clouds':
    if weather[1] == "few clouds: 11-25%" or weather[1] == "scattered clouds: 25-50%":
        if 'n' in weather[4]:
            animation = nightpartcloud(led_num,strip)
        else:
            animation = partcloud(led_num,strip)
    elif weather[1] == "broken clouds: 51-84%" or weather[1] == "overcast clouds: 85-100%":
        animation = cloud(led_num,strip)
    else:
        animation = scrollText(strip, weather[3],10,led_num,0xC0C0C0)
    else:
        animation = scrollText(strip, weather[3],10,led_num,0xFF0000)

animation.start()
temp_scroll.start()
date_scroll.start()
dis_hour.start()
dis_min.start()

```

```
dis_noon.start()
wind_top.start()
wind_bottom.start()
wind.start()
sleep(60)
strip.clear_strip()
stop_animate = True
GPIO.cleanup()
sleep(1)
```

## Appendix D: Weather-test.py

```
# Python program to find current
# weather details of any city
# using openweathermap api

# import required modules
import requests, JSON

# Enter your API key here
api_key = "7595ed19467dfdb3d453eb78b5790d23"

# base_url variable to store url
base_url = "http://api.openweathermap.org/data/2.5/weather?"

# Give city name
city_name = input("Enter city name : ")

# complete_url variable to store
# complete url address
complete_url = base_url + "appid=" + api_key + "&q=" + city_name

# get method of requests module
# return response object
response = requests.get(complete_url)

# JSON method of response object
# convert JSON format data into
# python format data
x = response.json()
print(x)

# Now x contains list of nested dictionaries
# Check the value of "cod" key is equal to
# "404", means city is found otherwise,
```



```

# city is not found
if x["cod"] != "404":

    # store the value of "main"
    # key in variable y
    y = x["main"]

    # store the value corresponding
    # to the "temp" key of y
    current_temperature = y["temp"]

    # store the value corresponding
    # to the "pressure" key of y
    current_pressure = y["pressure"]

    # store the value corresponding
    # to the "humidity" key of y
    current_humidity = y["humidity"]

    # store the value of "weather"
    # key in variable z
    z = x["weather"]

    # store the value corresponding
    # to the "description" key at
    # the 0th index of z
    weather_description = z[0]["main"]

    # print following values
    print(" Temperature (in kelvin unit) = " +
          str(current_temperature) +
          "\n atmospheric pressure (in hPa unit) = " +
          str(current_pressure) +
          "\n humidity (in percentage) = " +

```

```
        str(current_humidity) +  
        "\n description = " +  
        str(weather_description))  
  
else:  
    print(" City Not Found ")
```

# References

- [1] “U.S. Energy Information Administration - EIA - Independent Statistics and Analysis,” Solar explained - U.S. Energy Information Administration (EIA), 04-Dec-2019. [Online]. Available: <https://www.eia.gov/energyexplained/solar/>.
- [2] P. Mondal, “Solar Energy: 10 Major Application of Solar Energy – Explained!,” Your Article Library, 27-Feb-2014. [Online]. Available: <http://www.yourarticlelibrary.com/energy/solar-energy-10-major-application-of-solar-energy-explained/28197>.
- [3] L. Richardson, “The 5 Most Common Uses of Solar Energy in 2020: EnergySage,” Solar News, 18-Feb-2020. [Online]. Available: <https://news.energysage.com/most-common-solar-energy-uses/>.
- [4] “Nikola Tesla and his work in wireless energy and power transfer,” Contemporary Sci Innovation, 19-Feb-2016. [Online]. Available: <https://sites.suffolk.edu/xenia/2016/02/17/nikola-tesla-and-his-work-in-wireless-energy-and-power-transfer/>. [Accessed: 10-Mar-2020].
- [5] M. Etemadrezaei, “Wireless Power Transfer,” Power Electronics Handbook (Fourth Edition), 15-Sep-2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128114070000246>. [Accessed: 10-Jan-2020].
- [6] “Solar PV,” Solar PV | Student Energy. [Online]. Available: <https://www.studentenergy.org/topics/solar-pv>.
- [7] “Solar Photovoltaic Technology Basics,” Energy.gov. [Online]. Available: <https://www.energy.gov/eere/solar/articles/solar-photovoltaic-technology-basics>.
- [8] B. Usher, “Renewable Solar Energy,” in Renewable Energy: A Primer for the Twenty-First Century, Chichester, NY: Columbia University Press, 2019, pp. 43–53.
- [9] “Solar Photovoltaic Cell Basics,” Energy.gov. [Online]. Available: <https://www.energy.gov/eere/solar/articles/solar-photovoltaic-cell-basics>.
- [10] “Wiring Diagram of Solar Panel with Battery, Inverter, Charge controller and Loads,” ETechnoG. [Online]. Available: <https://www.etechnog.com/2019/01/wiring-diagram-of-solar-panel.html>.
- [11] “Wireless Charging Pad,” iF WORLD DESIGN GUIDE. [Online]. Available: <https://ifworlddesignguide.com/entry/233059-wireless-charging-pad>. [Accessed: 08-Mar-2020].
- [12] Lesurf, J. (n.d.). Skin Effect Internal Impedance & Types of Wire. [online] St-andrews.ac.uk. Available at: [https://www.st-andrews.ac.uk/~www\\_pa/Scots\\_Guide/audio/skineffect/page1.html](https://www.st-andrews.ac.uk/~www_pa/Scots_Guide/audio/skineffect/page1.html) [Accessed 25 Jan. 2020].

- [13] “Resonant inductive coupling,” Wikipedia, 02-Feb-2020. [Online]. Available: [https://en.wikipedia.org/wiki/Resonant\\_inductive\\_coupling](https://en.wikipedia.org/wiki/Resonant_inductive_coupling). [Accessed: 08-Mar-2020].
- [14] “Series Resonance in a Series RLC Resonant Circuit,” Electronics-Tutorials.com. [Online]. Available: <https://www.electronics-tutorials.ws/accircuits/series-resonance.html>. [Accessed: 02-Feb-2020].
- [15] V. Muratov, “Methods for Foreign Object Detection in Inductive Wireless Charging,” in Qi Developer Forum, 16-Nov-2017.
- [16] WorldSemi, “WS2812B Datasheet,” DigiKey. [Online]. Available: <https://www.digikey.com/en/datasheets/parallax-inc-28085-ws2812b-rgb-led-datasheet>. [Accessed: 28-Feb-2020].
- [17] “SK9822 – a clone of the APA102?,” Tim's Blog, 14-Dec-2016. [Online]. Available: <https://cpldcpu.wordpress.com/2016/12/13/sk9822-a-clone-of-the-apa102/>. [Accessed: 17-Feb-2020].
- [18] Shiji Lighting, “APA102C RGB Full Color LED control IC,” Adafruit Industries, 1-Aug-2014. [Online]. Available: <https://cdn-shop.adafruit.com/datasheets/APA102.pdf>. [Accessed: 28-Feb-2020].
- [19] Dongguang Opsco Optoelectronics, “SK9822 Integrated Light Source Intelligent Control,” SK9822 datasheet, March. 2016.
- [20] M. Grusin, “Serial Peripheral (SPI),” Serial Peripheral Interface (SPI). [Online]. Available: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all>. [Accessed: 25-Feb-2020].
- [21] A. Sendy, “What are the pros and cons of Monocrystalline, Polycrystalline and Thin Film solar panels?,” Solar Reviews, 06-Feb-2020. [Online]. Available: <https://www.solarreviews.com/blog/pros-and-cons-of-monocrystalline-vs-polycrystalline-solar-panels>.
- [22] “Thin film vs. mono/polycrystalline panels,” Energy Matters. [Online]. Available: <https://www.energymatters.com.au/panels-modules/thin-film-monocrystalline/>.
- [23] “Amorphous Silicon Solar Panels,” Energy Informative. [Online]. Available: <https://energyinformative.org/amorphous-silicon-solar-panels/>.
- [24] “Lead Acid Batteries,” Morningstar Corporation. [Online]. Available: <https://www.morningstarcorp.com/lead-acid-batteries/>.
- [25] “Flooded vs. Sealed Rechargeable Batteries,” Sure Power, Inc, 02-Jan-2014. [Online]. Available: <https://www.sure-power.com/2014/01/flooded-vs-sealed-rechargeable-batteries/>.

- [26] W. Burlin, "AGM Battery vs. Lead Acid: Pros and Cons of Solar Batteries," Wholesale Solar, 27-Jan-2020. [Online]. Available: <https://www.wholesalesolar.com/blog/lead-acid-battery-comparison/>.
- [27] "Lithium-based Batteries Information," Lithium-based Batteries Information – Battery University, 01-Jun-2018. [Online]. Available: [https://batteryuniversity.com/learn/article/lithium\\_based\\_batteries](https://batteryuniversity.com/learn/article/lithium_based_batteries).
- [28] "MPPT Vs PWM Solar Controllers," IINNO Ltd, 29-Jan-2018. [Online]. Available: <https://iinno-lighting.com/mppt-vs-pwm-solar-controllers/>.
- [29] "Qi Specifications," Wireless Power Consortium, 2017.
- [30] "Qi innovation beyond Power class 0," Micropross.com, 15-Jan-2018. [Online]. Available: <https://www.micropross.com/news-Qi-innovation-beyond-Power-class-0-81-n>. [Accessed: 04-Feb-2020].
- [31] "Resonant wireless power transfer," Infineon.com, May-2018. [Online]. Available: [https://www.infineon.com/dgdl/Infineon-Whitepaper\\_WirelessCharging\\_Wireless\\_Resonant\\_Power\\_Transfer-WP-v01\\_00-EN.pdf?fileId=5546d462636cc8fb0163aaf282900eef](https://www.infineon.com/dgdl/Infineon-Whitepaper_WirelessCharging_Wireless_Resonant_Power_Transfer-WP-v01_00-EN.pdf?fileId=5546d462636cc8fb0163aaf282900eef). [Accessed: 02-Feb-2020].
- [32] A. Frumusanu, "The State of Wireless Charging Standards in Mobile," AnandTech, 02-Apr-2015. [Online]. Available: <https://www.anandtech.com/show/9130/wireless-charging-standards-in-mobile/3>. [Accessed: 08-Mar-2020].
- [33] N. Poole, "Everything You Didn't Want to Know About RGB Matrix Panels," News - SparkFun Electronics, 19-Mar-2018. [Online]. Available: <https://www.sparkfun.com/sparkx/blog/2650>. [Accessed: 28-Feb-2020].
- [34] "8\*32 Pixel SK9822 LED Flexible Matrix," iPixel LED Light Co., LTD. [Online]. Available: <http://www.ipixelleds.com/index.php?id=166>. [Accessed: 01-Mar-2020].
- [35] "Understanding Today's LCD Screen Technology," Samsung Display PID, 01-Aug-2017. [Online]. Available: <https://pid.samsungdisplay.com/en/learning-center/blog/lcd-structure>. [Accessed: 28-Feb-2020].
- [36] "OLED vs. LCD," LG USA. [Online]. Available: <https://www.lg.com/us/experience-tvs/oled-tv/oled-vs-lcd>. [Accessed: 28-Feb-2020].
- [37] "OLED technology: introduction and basics," OLED, 23-Dec-2018. [Online]. Available: <https://www.oled-info.com/oled-technology>. [Accessed: 28-Feb-2020].
- [38] "Electronic Ink," E Ink | Electronic Ink. [Online]. Available: <https://www.eink.com/electronic-ink.html>. [Accessed: 28-Feb-2020].

- [39] “Interstate Batteries DCM0075 12V Maintenance-Free Deep Cycle 75AH AGM Battery,” Sump Pumps Direct, 19-Mar-2017. [Online]. Available: <https://www.sumpumpsdirect.com/Interstate-Batteries-DCM0075/p50878.html>.
- [40] “HQST 30 Amp PWM Solar Charge Controller,” HQST. [Online]. Available: <https://hqsolarpower.com/hqst-30-amp-pwm-solar-charge-controller/>.
- [41] International Rectifier, “IRS2453(1)D(S),” IRS24531D datasheet, April. 2016.
- [42] ON Semiconductor, “Axial Lead Standard Recovery Rectifiers,” 1N4005 datasheet, 2005
- [43] Texas Instruments, “LM340, LM340A and LM7805 Family Wide VIN 1.5-A Fixed Voltage Regulators,” LM7805 datasheet, Feb. 2000 [Revised Sep. 2016].
- [44] “200W Development Kit Manual,” Würth Elektronik Online. [Online]. Available: [https://www.w-e-online.com/web/en/electronic\\_components/produkte\\_pb/demoboards/wireless\\_power/design\\_kit\\_200\\_w/wireless\\_power\\_200wkit\\_page.php](https://www.w-e-online.com/web/en/electronic_components/produkte_pb/demoboards/wireless_power/design_kit_200_w/wireless_power_200wkit_page.php). [Accessed: 10-Jan-2020].
- [45] “Everbilt M3-0.5 x 25 mm. Phillips Flat-Head Machine Screws-39458,” The Home Depot. [Online]. Available: <https://www.homedepot.com/p/Everbilt-M3-0-5-x-25-mm-Phillips-Flat-Head-Machine-Screws-39458/203091630>. [Accessed: 09-Mar-2020].
- [46] “PCB Prototype & PCB Fabrication Manufacturer,” JLCPCB. [Online]. Available: <https://jlcpcb.com/>. [Accessed: 28-Feb-2020].
- [47] “Raspberry Pi Desktop,” Raspberry Pi, Feb-2020. [Online]. Available: <https://www.raspberrypi.org/downloads/raspberry-pi-desktop/>. [Accessed: 09-Mar-2020].
- [48] M. Tinue, “apa102-pi,” GitHub. [Online]. Available: <https://github.com/tinue/apa102-pi>. [Accessed: 11-Mar-2020].
- [49] OpenWeatherMap.org, “Current weather and forecast,” openweather. [Online]. Available: <https://openweathermap.org/>. [Accessed: 09-Mar-2020].
- [50] “Aideepen 20A 300W DC Step Down Buck Converter,” Amazon. [Online]. Available: [https://www.amazon.com/Aideepen-Constant-Current-Adjustable-Converter/dp/B0747QDRW9/ref=sr\\_1\\_3?keywords=300w buck converter&qid=1583852145&sr=8-3](https://www.amazon.com/Aideepen-Constant-Current-Adjustable-Converter/dp/B0747QDRW9/ref=sr_1_3?keywords=300w+buck+converter&qid=1583852145&sr=8-3).
- [51] Infineon, “XMCTM Link,” XMCTM Link user guide, November. 2015.
- [52] “BQ25703A I2C Battery Buck-Boost Charge Controller with System Power Monitor & Processor,” BQ25703A | TI.com. [Online]. Available: <http://www.ti.com/product/BQ25703A>. [Accessed: 28-Feb-2020].