Creating STEM Contents: Solar Power with a Tracking System

Leo Predanic

George Thuita

Brandon Cenci

**Subsystems Report**

REVISION –

4 December 2022

Subsystems Report

for

Creating STEM Contents: Solar Power with a Tracking System

Prepared by:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Author Date

Approved by:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Leader Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

John Lusher, P.E. Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

T/A Date

**Change Record**

| **Rev.** | **Date** | **Originator** | **Approvals** | **Description** |
| --- | --- | --- | --- | --- |
| **-** | 12/4/2022 | Leo Predanic, George Thuita, Brandon Cenci |  | Draft Release |

**Table of Contents**

[**Table of Contents 48**](#_heading=h.1fob9te)

[**List of Tables**](#_heading=h.3znysh7) **49**

[**List of Figures**](#_heading=h.2et92p0) **50**

[**1.**](#_heading=h.tyjcwt) **Introduction** [**51**](#_heading=h.1fob9te)

[**2.**](#_heading=h.2s8eyo1) **Solar Power Generation Subsystem** [**52**](#_heading=h.1fob9te)

[2.1.](#_heading=h.17dp8vu) Subsystem Introduction [**52**](#_heading=h.1fob9te)

[2.2.](#_heading=h.3rdcrjn) DC Load [**52**](#_heading=h.1fob9te)

[2.3.](#_heading=h.26in1rg) Buck-Boost Converter [**53**](#_heading=h.1fob9te)

[2.4.](#_heading=h.26in1rg) Charge Controller and Battery [**56**](#_heading=h.1fob9te)

[2.5.](#_heading=h.26in1rg) Solar Panel [**58**](#_heading=h.1fob9te)

[2.6.](#_heading=h.26in1rg) Printed Circuit Board Design [**59**](#_heading=h.1fob9te)

[**3.**](#_heading=h.35nkun2) **Electrical I/O Subsystem** [**62**](#_heading=h.1fob9te)

[3.1. Subsystem](#_heading=h.1ksv4uv) Introduction [**62**](#_heading=h.1fob9te)

[3.2.](#_heading=h.2jxsxqh) Analog to Digital Converter [**62**](#_heading=h.1fob9te)

[3.3.](#_heading=h.2jxsxqh) Current Sense Amplifier and Current Sense Resistor [**64**](#_heading=h.1fob9te)

[3.4.](#_heading=h.2jxsxqh) High Side PFET Switches [**67**](#_heading=h.1fob9te)

[3.5.](#_heading=h.2jxsxqh) Final Test Circuit [**69**](#_heading=h.1fob9te)

[3.6.](#_heading=h.2jxsxqh) Stepper Motor and Motor Driver [**71**](#_heading=h.1fob9te)

[3.7.](#_heading=h.2jxsxqh) Subsystem Citations [**73**](#_heading=h.1fob9te)

[**4.**](#_heading=h.2bn6wsx) **GUI & Database Subsystem** [**7**](#_heading=h.1fob9te)**4**

4.1.Subsystem Introduction [**74**](#_heading=h.1fob9te)

4.2.Database [**74**](#_heading=h.1fob9te)

4.3.GUI Outline [**75**](#_heading=h.1fob9te)

4.4.Infographics Tab [**7**](#_heading=h.1fob9te)**6**

4.5.System Animations Tab [**7**](#_heading=h.1fob9te)**8**

4.6.Control Systems Tab [**7**](#_heading=h.1fob9te)**9**

4.7.Integration Considerations **81**

**5**[**.**](#_heading=h.2bn6wsx) **Subsystems Conclusion 84**

**List of Tables**

**Table** [**1.**](#_heading=h.1t3h5sf) **Example Database CSV with Dummy Data 77**

**List of Figures**

[**Figure 1. Collection of Small Devices for First Load**](#_heading=h.1t3h5sf) **52**

[**Figure 2. Handheld Gaming Console for Second Load**](#_heading=h.44sinio) **52**

[**Figure**](#_heading=h.44sinio) **3. Electrical Schematic from Converter to Load 53 Figure 4. Buck-Boost Converter Schematic 54** [**Figure**](#_heading=h.44sinio) **5. Buck-Boost Converter Output for 4.2V Input 55**

[**Figure**](#_heading=h.44sinio) **6. Buck-Boost Converter Output for 2.5V Input 55**

[**Figure**](#_heading=h.44sinio) **7. Physical Buck-Boost Converter Output Across Input 56**

[**Figure**](#_heading=h.44sinio) **8. Solar Cell I-V Curve 56**

[**Figure**](#_heading=h.44sinio) **9. MPPT Algorithm Flow Diagram 57**

[**Figure**](#_heading=h.44sinio) **10. *DFRobot* Solar Power Manager 5V Charge Controller 57**

[**Figure**](#_heading=h.44sinio) **11. Charge Controller Calibration Curve 58**

[**Figure**](#_heading=h.44sinio) **12. Solar Panel Calibration Curve 59**

[**Figure**](#_heading=h.44sinio) **13. System PCB Schematic 60**

[**Figure**](#_heading=h.44sinio) **14. System PCB Layout 61**

[**Figure**](#_heading=h.44sinio) **15. MCP3008 Class 63**

[**Figure**](#_heading=h.44sinio) **16. ADC and Multimeter Measured Voltage 63**

[**Figure**](#_heading=h.44sinio) **17. ADC Voltage Error 64**

[**Figure**](#_heading=h.44sinio) **18. Current Sense Amplifier and Current Sense Resistor 65**

[**Figure**](#_heading=h.44sinio) **19. Current Sense Amplifier and Current Sense Resistor Testbench 65**

[**Figure**](#_heading=h.44sinio) **20. Measured Current and Theoretical Current Values 66**

[**Figure**](#_heading=h.44sinio) **21. Measured Current Error 66**

[**Figure**](#_heading=h.44sinio) **22. PFET Characterization Curve 68**

[**Figure**](#_heading=h.44sinio) **23. Example of a Level Shifter 68**

[**Figure**](#_heading=h.44sinio) **24. Modified Testbench 69**

[**Figure**](#_heading=h.44sinio) **25. Finalized Testbench 69**

[**Figure**](#_heading=h.44sinio) **26. Finalized Testbench Code 70**

[**Figure**](#_heading=h.44sinio) **27. Finalized Testbench Code Output 71**

[**Figure**](#_heading=h.44sinio) **28. ULN2003 Internal and External Circuitry 72**

[**Figure**](#_heading=h.44sinio) **29. ULN2003 Stepper Motor Driver Library Functions 72**

[**Figure**](#_heading=h.44sinio) **30. GUI Skeleton 74**

[**Figure**](#_heading=h.44sinio) **31. GUI Skeleton Code 75**

[**Figure**](#_heading=h.44sinio) **32. GUI Menu 76**

[**Figure**](#_heading=h.44sinio) **33. Infographics Tab Code 76**

[**Figure**](#_heading=h.44sinio) **34. GUI Infographics and Database Display Tab 77**

[**Figure**](#_heading=h.44sinio) **35. Animations Tab Code 78**

[**Figure**](#_heading=h.44sinio) **36. GUI Animations Tab 79**

[**Figure**](#_heading=h.44sinio) **37. Control Systems Tab Code 80**

[**Figure**](#_heading=h.44sinio) **38. Control Systems Tab 81**

[**Figure**](#_heading=h.44sinio) **39. Raspberry Pi Touch Screen 82**

[**Figure**](#_heading=h.44sinio) **40. Additional Display Feature Skeleton 82**

# Introduction

As the demand for engineering solutions increases exponentially, it is imperative to inspire future generations of STEM students by creating captivating demonstration tools to drive their interest and creativity. The goal of this system is to foster this latent interest by being a simple and portable apparatus that is interactable and tracks the system’s performance by displaying the information in a digestible format for K-12 students. The Creating STEM Contents (CSC) system will contain a solar power generation solution to drive a physical load that generates visual, tactile, or aural excitations. The system will have an MCU to take various analog measurements and process it to provide relevant data about the system’s performance. The MCU will store information into a database where a GUI can read in the data and display figures and animations about system operation. The GUI will also feature interactive controls to set operation modes and control the circuitry. This allows through explanation of the concepts of solar power, electronics, and computer science to the target audience. The system is broken down into three subsystems: Solar Power Generation, Electrical I/O, and GUI & Database. Each subsystem has been thoroughly tested and validated. The path for integration is clear for the system specified in the Concept of Operations, Functional System Requirements, and Interface Control Document reports. Further details on how the subsystems’ integration will be done roughly will be discussed.

# Solar Power Generation Subsystem

## Subsystem Introduction

The purpose of this subsystem is to generate the power that will be delivered to the load. Since solar power is not efficiently used directly from the panel, it is usually stored into a battery before being delivered to a load. Since battery voltage output can be very noisy, voltage regulators or converters are used to maintain constant voltage at the output to prevent inconsistent power delivery. To help deliver power to the battery from the solar panel as effectively as possible, a charge controller is used that employs the maximum power point tracking (MPPT) algorithm. The charge controller ensures that the solar panel delivers maximum power by adjusting the current and voltage output relative to the previous time step to ensure the fastest battery charging. These blocks in tandem allow power to be transferred as efficiently as possible. The discussion of the signal path blocks will be performed backwards from the load to the solar panel.

## DC Load

### 2.2.1. Operation

Since the system is designed to be used as a demonstration tool for children, the goal for the load is to be something that has visual and tactile stimulus to better help visualize energy transfer and conversion since electrical energy (on small scales) is not a visible phenomenon. Two types of loads will be employed: a combination of small devices that can directly convert electrical energy into light or motion, and a handheld gaming console (‘Gameboy’) to provide direct interaction with the system. The small devices will be a small DC servo motor, a small fan, and several LEDs. These devices convert the electrical energy into rotational motion and light so it is clear to see the transfer of power from the battery to the load. Since children play video games, using this system to power a gaming console is a powerful way to capture their attention and articulate what goes into making their electronics work. This also provides another way for children to interact with the system and capture their attention.



**Figure 1:** Collection of Small Devices for First Load



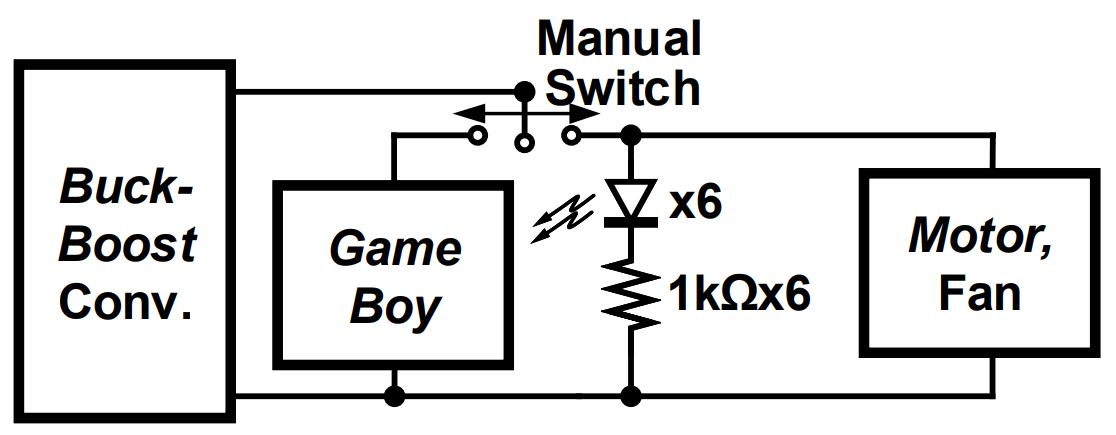
**Figure 2:** Handheld Gaming Console for Second Load

### 2.2.2. Validation

To validate that the loads operate and provide power specifications for the buck-boost converter, the devices were tested using a power supply that measures current output. The LEDs were designed to output around 2mA for a supply of 3.7V. 3.7V is chosen for the supply as well as the output of buck-boost converter due to the power specification of the gaming console. The current draw for the DC servo motor, fan, and 6 LEDs was about 140mA at 3.7V. The gaming console draws about 180mA from a 3.7V supply.

### 2.2.3. Integration Challenges

To prevent excessive power draw, the gaming console will be powered separately from the smaller devices. A manual switch has been chosen to allow the choice of powering the gaming console or the small devices. The figure below illustrates the schematic diagram from the output of the buck-boost converter.



**Figure 3:** Electrical Schematic from Converter to Load

Note that the ‘x6’ refers to 6 parallel branches of a series LED and 1kΩ resistor.

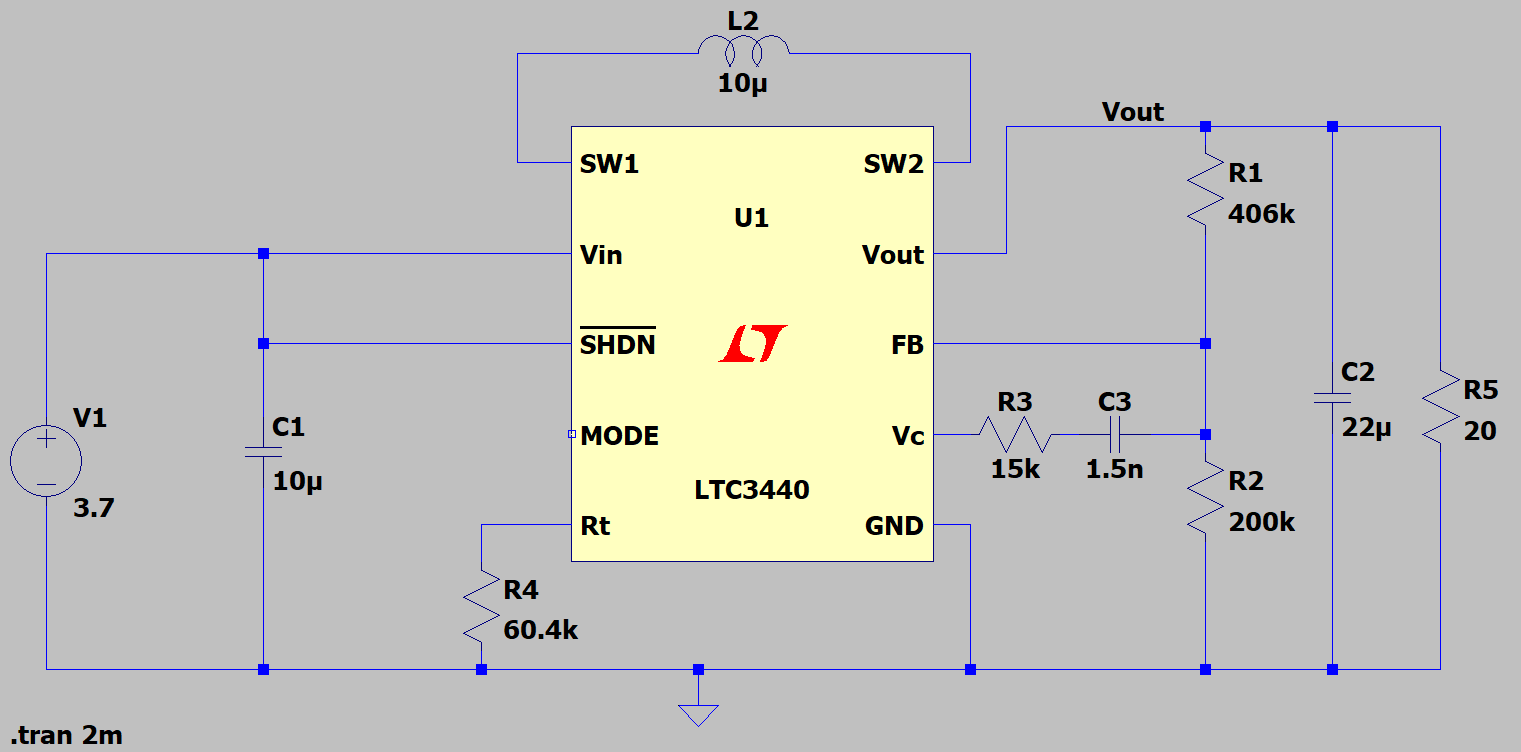
## Buck-Boost Converter

### 2.3.1. Operation

To design an efficient converter, the output power requirement must be known. Since the gaming console takes the most power requiring 180mA from the converter, an additional margin is chosen to avoid dropout if the current output differs from the expected draw due to unexpected variations. For this reason, a maximum current output of 200mA is chosen with a required 3.7V voltage output to power the gaming console. The converter needs to be able to regulate to 3.7V at the output with some margin greater than 3.7V input due to overcharging of the battery. It also needs to boost the voltage at the output when the battery voltage is lower than 3.7V as the battery discharges. The LTC3440 solution was chosen for its high efficiency of 95% and low footprint. Since the LTC3440 is an *Analog Devices* solution, their proprietary simulation model is available on *LTSpice* as their proprietary simulator. The main topology was chosen from the datasheet to obtain a 200mA maximum output current and the inductors and capacitors were chosen to meet the efficiency specification. For 95% efficiency, the module must be tuned to an oscillator frequency of 1MHz. The timing resistor value is given by the equation below:

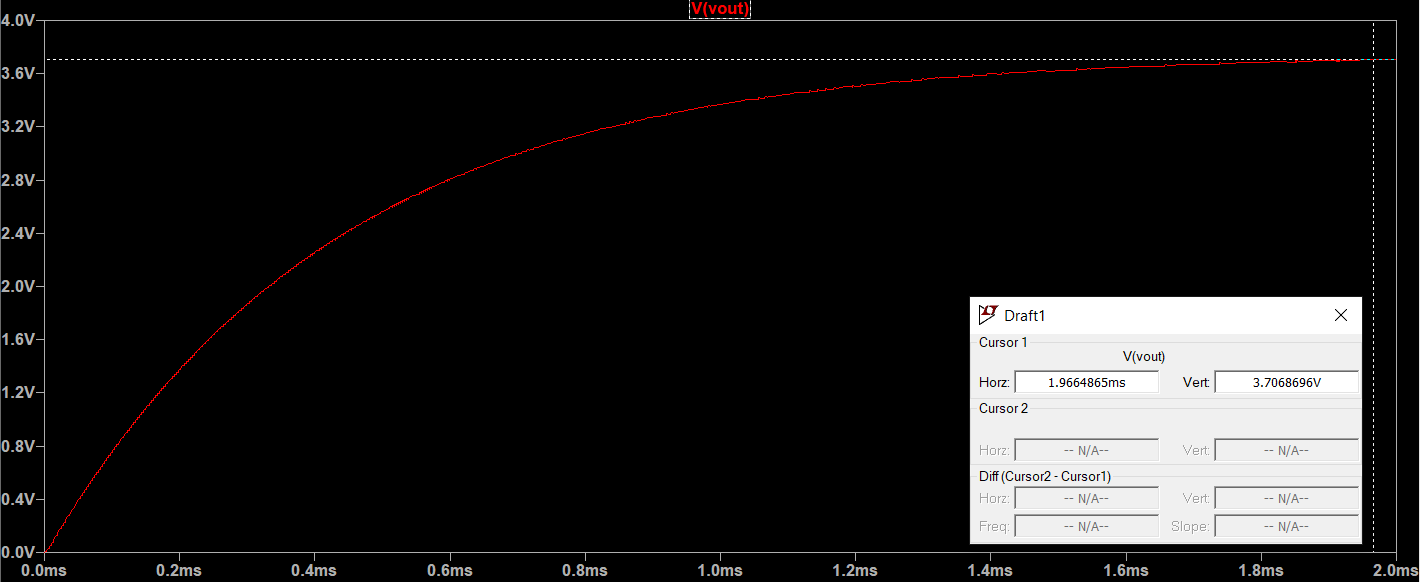
This value was slightly adjusted to 60.4kΩ for optimal performance in simulation. To set the output voltage, the values for the voltage division network needed to be calculated. The module requires a feedback voltage of 1.22V. Therefore the output voltage is specified by the equation below:

R2 is suggested to be 200kΩ by the manufacturer so by the setting the output voltage to 3.7V, R1 can be solved for and yields a result of 406kΩ. To find the input capacitor, it was recommended to place at least 4.7µF so a 10µF capacitor was used. To minimize the ripple at the output, it was recommended to use a 4.7µF capacitor but to minimize the ripple and output noise a 22µF capacitor was placed. For high current output such as this design, an inductor of at least 4.7µH is recommended so a 10µH inductor was used. To compensate the feedback loop for system stability, a sufficient phase margin must be provided. Type II compensation of a series capacitor and resistor was chosen since the size of the output capacitor is not stringently small. According to the datasheet, for this type of compensation a 1.5nF capacitor and 15kΩ resistor is sufficient to stabilize the loop. A schematic of the buck-boost converter is shown in the figure below.

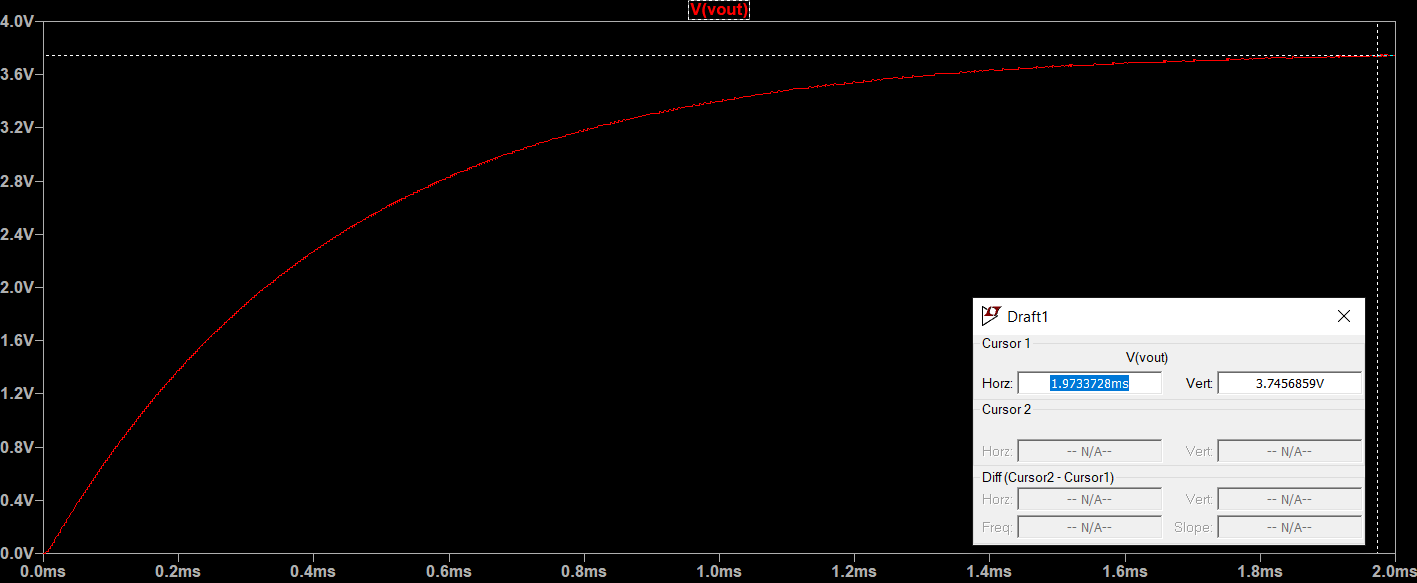


**Figure 4:** Buck-Boost Converter Schematic

The maximum and minimum input voltage were found such that the output voltage remained around 3.7V and found to be 4.2V and 2.5V respectively.



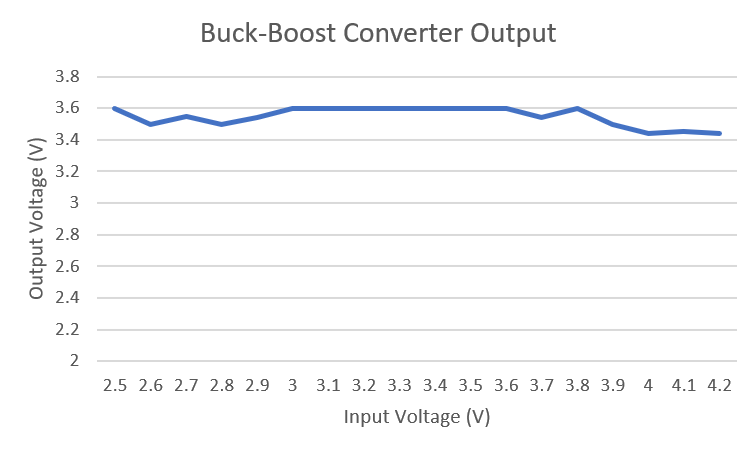
**Figure 5:** Buck-Boost Converter Output for 4.2V Input



**Figure 6:** Buck-Boost Converter Output for 2.5V Input

### 2.3.2. Validation

The buck-boost converter was constructed on a breadboard and its performance was validated by measuring the output voltage across the simulated maximum and minimum input voltages. The results are shown in the figure below. The output of the converter stabilized around 3.6V instead of the expected 3.7V. This is a result of the breadboard parasitics with the high oscillator frequency. The parasitic capacitors are large enough at 1MHz that they can pass some non-negligible amount of ripple current, dropping the voltage slightly below the 3.7V mark. Since this effect will not be present when the subsystem is implemented on PCB, there is no redesign necessary. Additionally, both loads can be powered with a 3.6V output. The small devices’ current draw at 3.7V input is 285mA and at 2.5V input is 350mA. The gaming console’s current draw at 3.7V input is 350mA and at 2.5V input is 600mA. The plot is shown in the figure below.

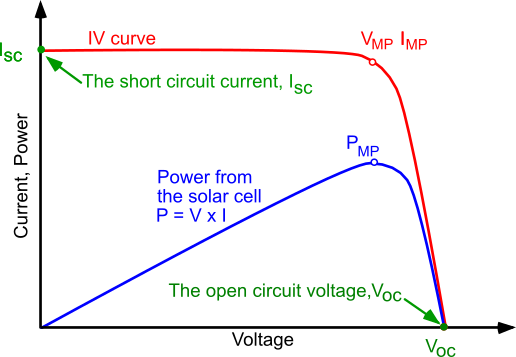


**Figure 7:** Physical Buck-Boost Converter Output Across Input

## Charge Controller and Battery

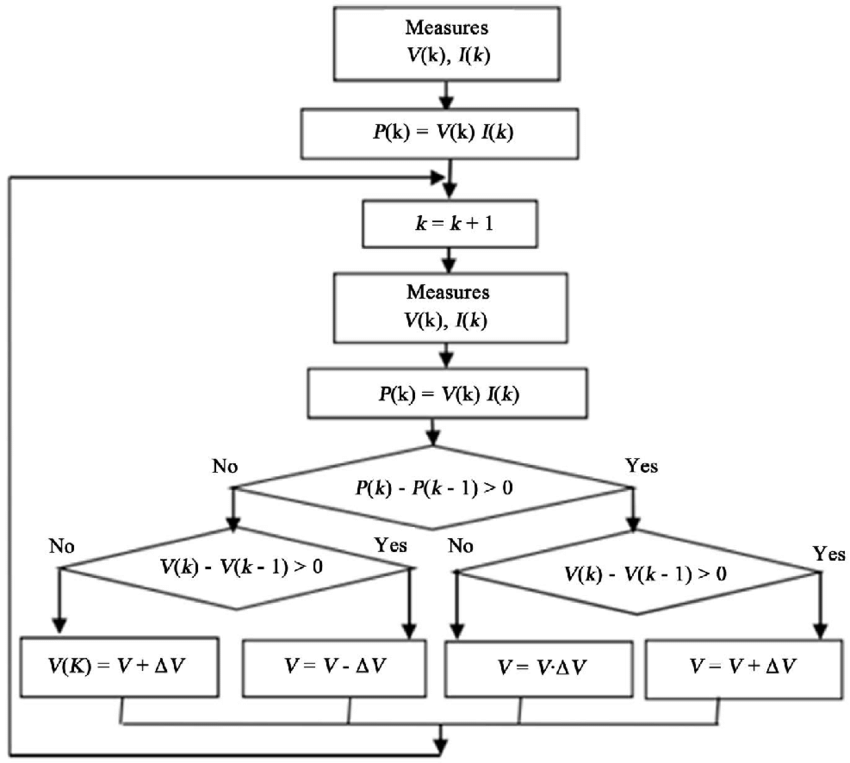
### 2.4.1. Operation

The charge controller relies on solar panel input power which isn’t very stable and is subject to rapid changes in output due to quick changes in incoming light which vary in intensity and angle of incidence. As a result, the output can pulsate and experience extreme fluctuations which is unacceptable for battery charging since a battery needs a constant source to pump current into the battery. Similar requirements exist for loads as they can not operate correctly with large swings in power which can shut the devices on and off randomly. To combat this, charge controllers are used to regulate the output of solar cells in such a way to provide constant output for a battery to charge or to another load. This system requires the charging of a 3.7V battery so a charge controller fitting this requirement is needed as well as the implementation of the MPPT algorithm. The MPPT algorithm works to keep the solar panel in a state of maximum output by controlling the current output. A solar panel I-V curve is shown in Figure 8.



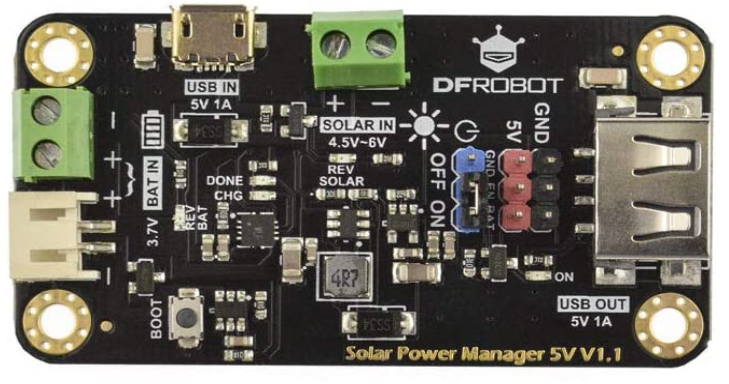
**Figure 8:** Solar Cell I-V Curve

It can be seen that at the knee of the I-V curve, the product of voltage and current produce their maximum output current and the MPPT algorithm attempts to keep the power point at this value. The flow diagram for the MPPT algorithm is shown in Figure 9.



**Figure 9:** MPPT Algorithm Flow Diagram

The charge controller chosen for 3.7V batteries using the MPPT algorithm with a small footprint is the *DFRobot* Solar Power Manager 5V. The device is shown in Figure 10.

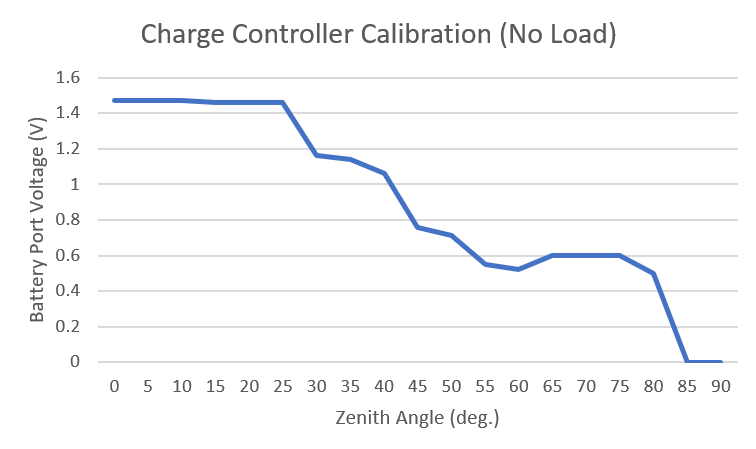


**Figure 10:** *DFRobot* Solar Power Manager 5V Charge Controller

The device works in constant current mode and can drive up to 900mA into the battery for fast charging. It has a 95% rated efficiency. For the maximum power point of the solar panel at 5.5V 100mA, the charge controller is expected to drive a 3.7V battery with a minimum of 140mA for a charge time of 6.4 hours for a 900mAh battery capacity. The discharge time for the battery is approximately 3.1 hours for the small devices load. This meets the charge and discharge time specifications of 12 hours and 2 hours respectively.

### 2.4.2. Validation

The charge controller displays an LED that lights up when power to the battery is being supplied named ‘CHG’. At sufficient lighting conditions, when the solar panel is exposed the battery will charge and the LED will be excited. Since the battery can overcharge to 4.2V, the battery voltage will be monitored by the ADC and charging will be stopped at 3.7V. To better measure the effect of light variations on the charge controller output, the battery load was removed and the battery output port of the charge controller was measured with a multimeter as the angle of incidence of the solar panel was swept across the normal vector. The resulting calibration curve is shown in Figure 11.



**Figure 11:** Charge Controller Calibration Curve

It can be seen that the charge controller regulates the output voltage well for a large angle of variation until the angle is swept too far. This indicates that the MPPT algorithm can no longer regulate the output voltage since the solar panel is not driving sufficient power to power the charge controller. This will be remedied in single-axis operation mode as the solar panel will be swept to maximize a conceptual flux of light through the panel surface. The physical battery takes 2.3 hours to power the small devices load and 7.9 hours to charge using the solar panel and charge controller with a full charge at around 910mAh.

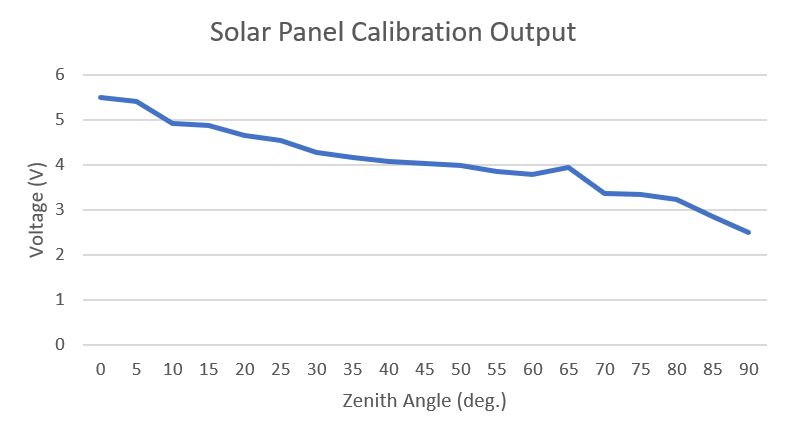
## Solar Panel

### 2.5.1. Operation

The final block and beginning of the signal chain is the solar panel. The solar panel converts radiant energy into electrical energy. As discussed in Section 2.4.1, the solar panel maximum power point is 5.5V 100mA. This was chosen sufficiently large to provide enough voltage headroom for battery charging. This is also sufficient size for the small form factor of the system.

### 2.5.2. Validation

To validate the device, the panel was exposed to sufficient lighting conditions and the angle of incident was swept from maximum to minimum flux. The plot is shown in the figure below. The voltage was measured using a multimeter. It can be seen that the voltage is very sensitive to angle of incident and despite the additional headroom provided by the charge controller, it will not be as efficient in fixed-axis operation mode as single-axis operation mode which is expected. Due to non-negligible reflections off of walls and other objects, the voltage does not drop to zero at the maximum angle however it is not sufficient to power the charge controller and hence charge the battery.

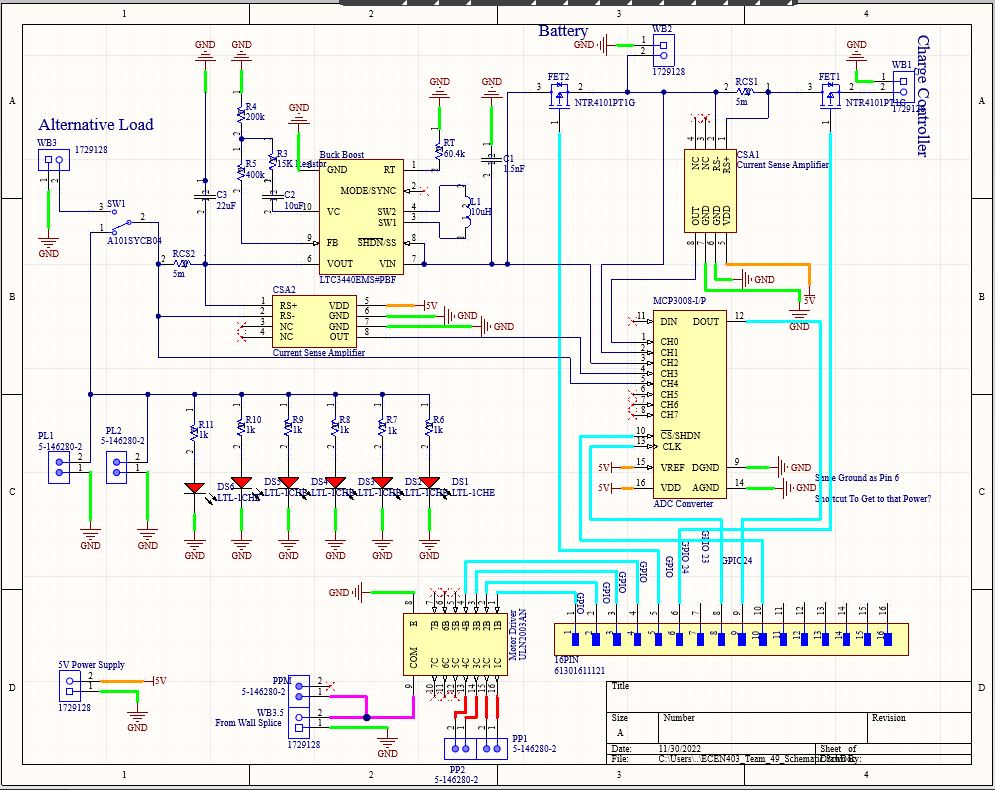


**Figure 12:** Solar Panel Calibration Curve

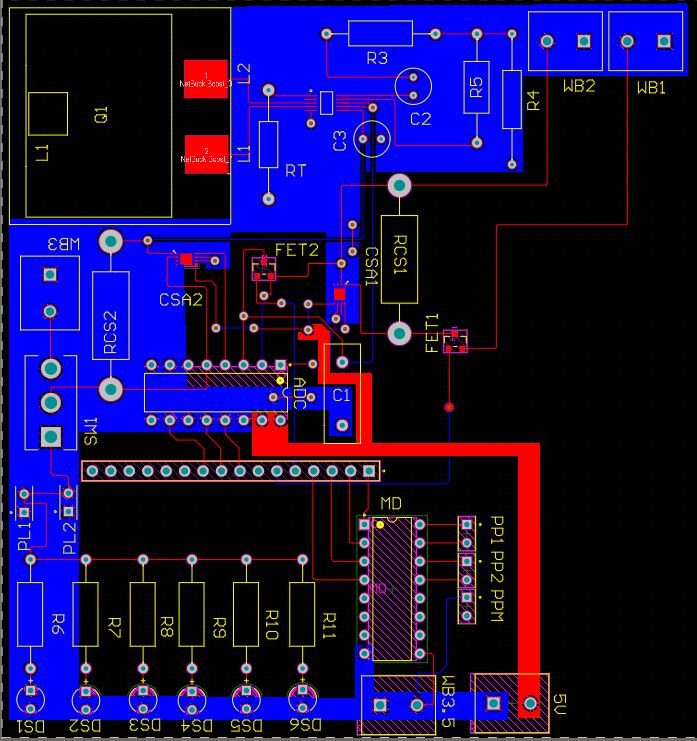
## Printed Circuit Board (PCB) Design

### 2.6.1. Operation

The printed circuit board is finalized and fit to deliver power and measure power to the board components meaning that both the Solar Power Generation and portions of the Electrical I/O subsystems will be present on this board. A terminal block will receive power from the charge controller into the FET switch to the Current Sense Amplifier to another terminal block that will be charged and provide power to the load. In order to deliver 5V to the ADC and the 2 Current Sense Amplifiers, a polygon pour copper sheet was used from the physical power support, through a terminal block, to the through hole nets of the ADC and the surface mounts of the current sense amplifiers. This 5V will be supplied by the Raspberry Pi. A separate polygon pour copper sheet was also used in the bottom layer to provide the common ground for all passive components and ICs. We decided to use a 16-pin header to provide pin connections from the Raspberry Pi’s GPIO and ground pins to the PCB board. The motor driver IC will be powered by an independent terminal block that will be sourced from a spliced wire connection from the wall. It will also deliver 5V to the motor driver. The 5V polygon pour sheet could not be used to power the current because the draw was more than the raspberry pi could supply through its 5V output GPIO. This terminal block will also supply the stepper motor. 2 pin dip pins will be used to supply the passive motor and the fan from the board that are placed in parallel with the LED’s. Another separate terminal block will supply power to the game console once the switch placed between it and the passive loads is switched on.



**Figure 13:** System PCB Schematic



**Figure 14:** System PCB Layout

### 2.6.2. Validation

In order to generate the necessary NC Drill and Gerber files for manufacturing , the PCB design must pass the Design Rule Check (DRC) implemented in Altium. The stock rules implemented by the program require solder mask, silk, hole clearance, and width constraints to be greater or equal to 1 mil or 0.254mm. These tests were run and the errors produced from these tests were fixed.

# Electrical I/O Subsystem

## Subsystem Introduction

The purpose of this subsystem is to act as the brain of the integrated system. Since the system is to be able to track its efficiency and performance in all operation modes, the subsystem uses an MCU (Raspberry Pi Model 4 B) to process and control various aspects of the system. There are two measurement nodes that the system uses to monitor power for battery charging and power transfer at the load. An ADC is used to sample the voltage at various points along the signal chain to monitor the power transfer and verify that various components are operating as expected. Two current sense amplifiers are placed in the signal path across a very small current sense resistor to convert the differential resistor voltage into a readable voltage for the ADC. The ADC can then compute the current from this voltage and use the node voltage to find the power transfer at the node of interest. To help isolate the system from losing power during battery charging or discharging, high side PFET switches are used to separate the load from the battery during charging and separate the charge controller from the battery during discharge. Finally, a motor driver is used since the MCU can not generate sufficient power to drive the inductive load of the stepper motor.

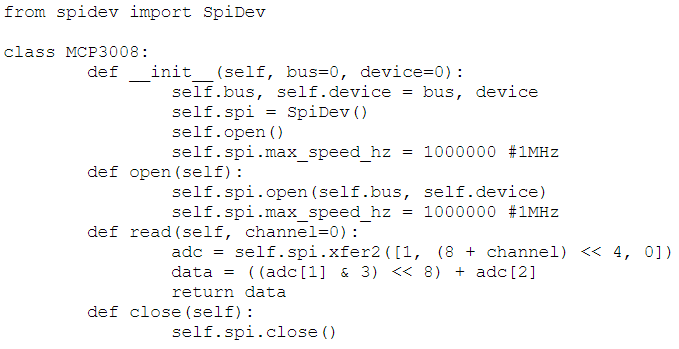
## Analog to Digital Converter (ADC)

### Operation

The ADC operates on the principle of converting a continuous (analog) voltage signal measured in volts to a binary (digital) signal transmitted serially to be read by an MCU. For this ADC, the MCP3008 was used due to its ability to interface well with the Raspberry Pi Model 4 B. The Serial Peripheral Interface (SPI) is used since it can synchronize data on the rising or falling edge of a clock allowing the MCU to access serially transmitted information and convert it to the corresponding parallel data to be available as a parallel bit binary number. There are 8 channels available on the MCP3008. Since there are two measurement nodes of interest in the system to measure power transfer, a total of 4 channels must be used: two for the node voltages and two for each current sense amplifier voltage output. With 4 channels to spare, several of the channels are tapped into various nodes in the system to monitor the voltage levels to detect if the system is operating as expected and can be used to debug incorrect system performance.

### ADC Library Code

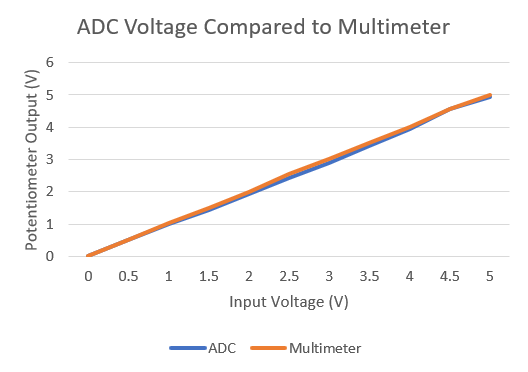
To receive the binary voltage data correctly from the ADC, the ADC must be excited in a way to correctly transmit and receive the data otherwise the voltage will not be able to be read. The ADC uses the SpiDev library to write a simple class named after the MCP3008 that performs several operations to be able to read data. The class code is provided below.



**Figure 15:** MCP3008 Class

### Validation

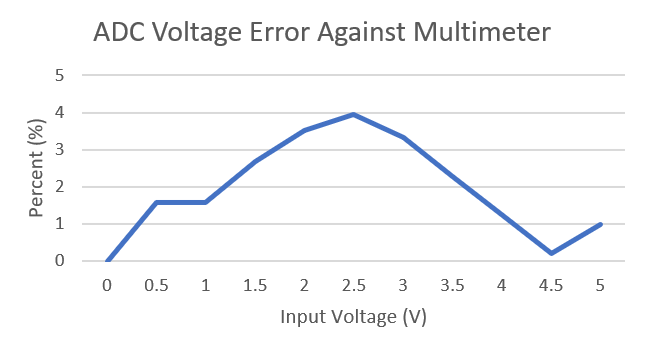
To validate the ADC, the channel tapped into the common node of a potentiometer with a supply of 5V. The knob was rotated to generate a common node voltage between 0V and 5V to measure voltage between ground and the ADC voltage reference. The power for the device is supplied by the Raspberry Pi. To verify the accuracy of the measurement, the common node was also sampled by a precise multimeter and the performance was compared.



**Figure 16:** ADC and Multimeter Measured Voltage

It can be seen that the voltage of the ADC follows the multimeter measurements quite accurately. To quantify the error performance since a minimal accuracy specification of 90% is set, we use to following equation below:

Here, the multimeter measurement is defined as the ‘accepted’ value and the ADC measurement is defined as the ‘measured’ value to compute the error. The error was computed according to the voltage difference between the ADC and multimeter. The error is shown in Figure 17.



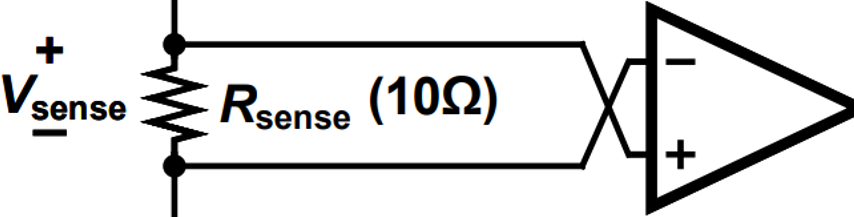
**Figure 17:** ADC Voltage Error

We see that maximum error in voltage reading is about 4%. Since the accuracy specification is 90%, we can tolerate this error. To improve the error, the Raspberry Pi voltage reference was measured over several operation cycles. The average voltage reference provided was found to be 5.21V. Since the MCP3008 is a 10-bit ADC, the number of binary states is 2 to the 10th power yielding 1024 binary states. To convert the digital value to an analog value for display, the received digital value is divided by 1023.0 which is the maximum binary number in decimal and is then normalized to the voltage reference (5.21V) to find the analog voltage value at the specified node. This yields a resolution of 5mV for the ADC which is acceptable. If large error is a problem for ADC measurement, a low-dropout regulator (LDO) can be used as a voltage reference due to their extremely stable voltage regulation with large supply noise or variation.

## 3.3. Current Sense Amplifier (CSA) and Current Sense Resistor (CSR)

### 3.3.1. Operation

To measure current, a current sense amplifier is used. It operates on the premise of sampling a differential voltage across a small current sense resistor which is on order of several milliOhm to tens of Ohms depending on its relative loading and voltage drop. Ideally, a current sense resistor would be as small as possible to generate a sizable enough differential voltage to be amplified. An example of a current sense amplifier and resistor is shown in Figure 18.



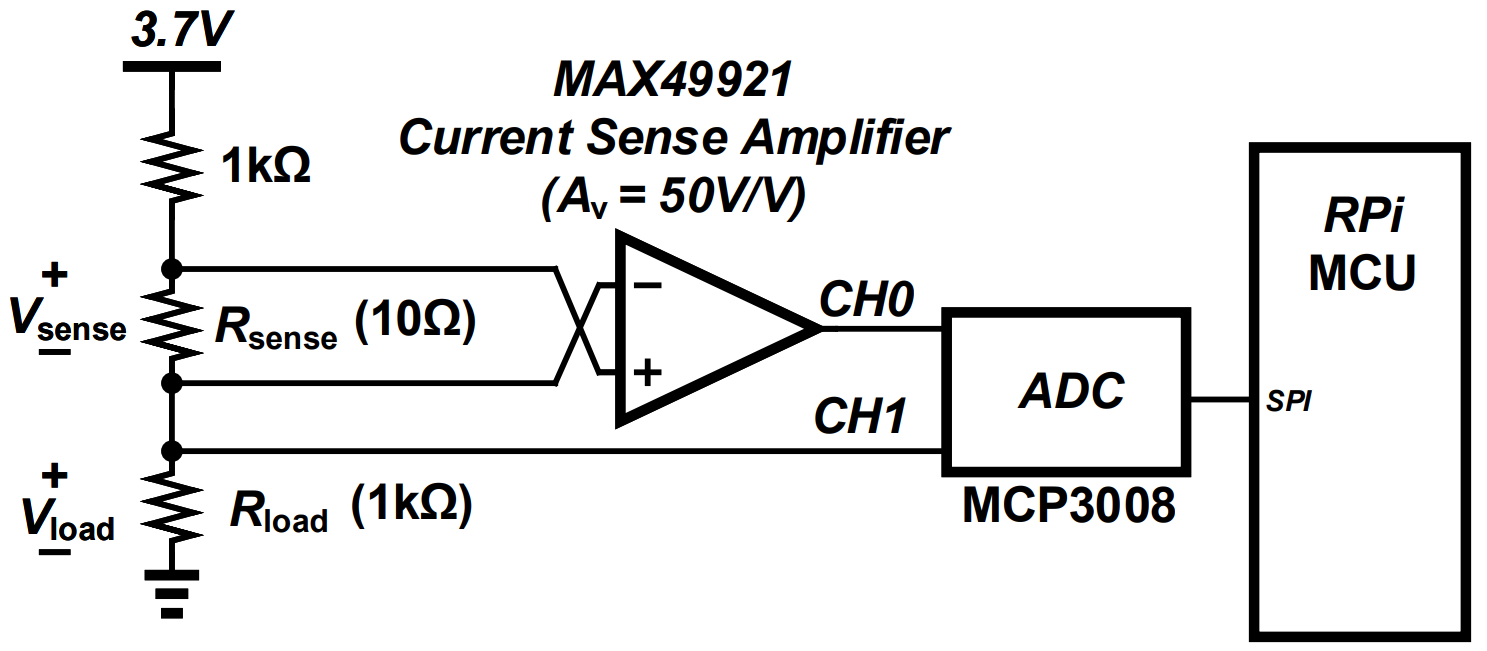
**Figure 18:** Current Sense Amplifier and Current Sense Resistor

The voltage at the output is the differential voltage ‘Vsense’ multiplied by the gain of the amplifier ‘AV’. The differential voltage across the resistor is generated by the product of the current ‘Isense' and value of the resistor ‘Rsense'. To find the current through Rsense , the process can be reversed by sampling the output of the CSA using an ADC and reversing the equation. The equation used to compute the current is shown below:

The MAX49921 CSA is used in this subsystem which has a differential-to-single ended gain of 50V/V. With the figure above, an example current can be computed using the Isense equation. Assuming the output voltage is 1V, the current through Rsense is [1/(50\*10)] = 0.002A or 2mA. This procedure is used to find the current at the measurement node. Since the node current and voltage are now known, power transfer at that node can be calculated.

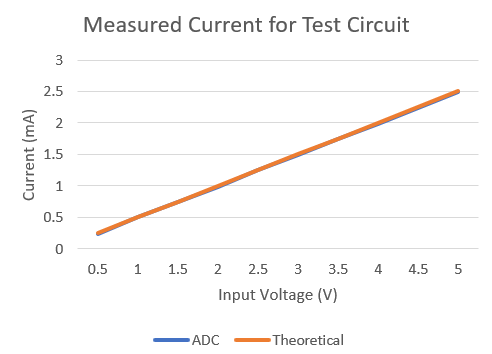
### 3.3.2. Validation

To validate the current sense amplifier, a simple testbench was used and is shown in the figure below.



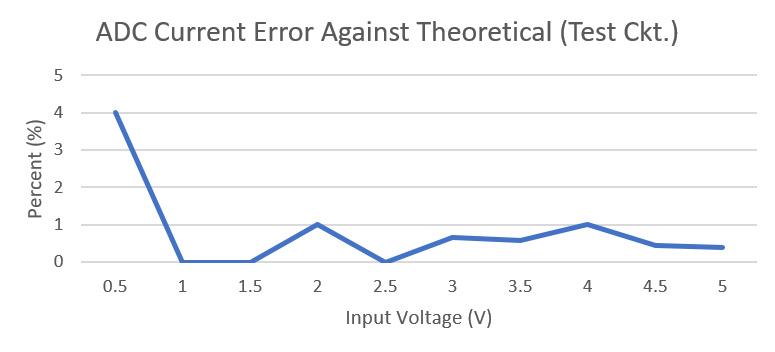
**Figure 19:** Current Sense Amplifier and Current Sense Resistor Testbench

To measure different currents, the supply voltage (shown as 3.7V) was adjusted between 0V and 5V and the current was measured by the procedure specified in Section 3.3.1. To find the accuracy of the current measurement, it is compared to the theoretical value of current in the testbench without the presence of the CSR to verify its lack of loading on the overall testbench which has been discussed to be desirable. The performance between the measured current and the theoretical value of current is shown in Figure 20.



**Figure 20:** Measured Current and Theoretical Current Values

It can be seen that the current measurement is very accurate and due to the choice of Rsense, there is little effect on overall testbench current. The error for the CSA and CSR pair is computed using the same error equation specified in Section 3.2.3 and is shown in Figure 21.



**Figure 21:** Measured Current Error

The maximum error is similar to the case of the ADC at about 4%. Since the accuracy specification is at 90%, this error can be tolerated. The main source of error is due to the ADC since it takes the voltage measurement at the output of the ADC. Therefore, accuracy improvements are the same as discussed in Section 3.2.3.

Now, the power at a node can be computed. For the testbench, the measured current is 1.85mA and the measured voltage is 1.84V. The corresponding power delivered to Rload is 3.41mW. Theoretical calculations yield a power of 3.42mW delivered to Rload, this yields an error of 0.37%, an extremely accurate measurement.

### 3.3.3. Integration Challenges

During integration, the CSA is expected to receive current on the order of several hundreds of milliAmps. A 10 Ohm CSR will saturate the output of the CSA due to the use of the same voltage reference as the ADC. A 100mA current will create a 1V drop across the CSR, which is unacceptable. A much smaller CSR has to be chosen. Assuming an average of 500mA, a 5mΩ CSR will generate a drop of 2.5mV. With supply voltages around 3.7V, this voltage drop is negligible and is sufficient to drive a readable voltage output of 125mV at the ADC which can be accurately read.

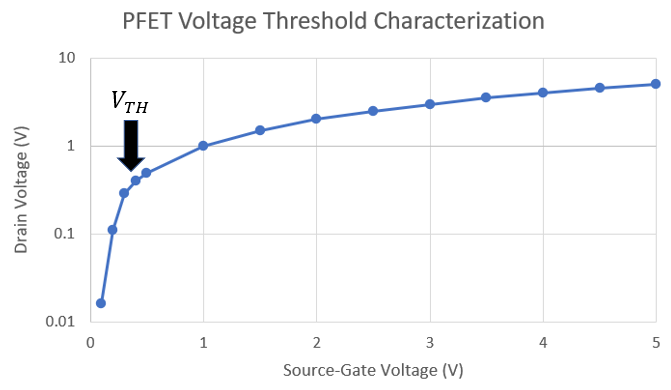
## 3.4. High Side PFET Switches

### 3.4.1. Operation

In order to prevent excessive power losses, the load needs to be isolated from the battery during charging and the charge controller needs to be isolated from the load during discharging. An electronic switch is desired to allow the system to be self-sufficient to reduce the amount of mechanical interaction with the system. The switches will therefore be driven by the MCU via General-Purpose-Input-Output (GPIO) pins. A transistor based switch will be used. To decide between the use of an NMOS or PMOS transistor, their performances are compared. An NMOS transistor is used as a pull down switch meaning that it can pass negative supply voltages well but struggles to pass positive supply voltage. The drop from drain to source is usually Vsupply - VTH where VTH is the threshold voltage of the transistor. Since threshold voltages for discrete power MOSFETs are usually around 1.5V, the drop is significant. PMOS transistors can pass positive supply voltages without this large of a drop. If a PFET is chosen with sufficiently small ‘on drain-source’ (Rds,on) resistance, the voltage drop is negligible and the PFET can be used as a high quality switch. Since the supply voltage for the system will be around 3.7V, the NTP4101P is chosen due to its threshold voltage range from 0.4V to 1.2V and low Rds,on of 70-80mΩ at the desired voltage range.

### 3.4.2. Validation

To test the operation of the transistor, the source is set to the desired voltage and swept from 5V to 0V. The gate is grounded to generate a negative VGS to saturate the device and the drain voltage is measured. The drain voltage is plotted on a logarithmic scale to exacerbate the characterization curve. When the transistor is in saturation mode, the plot of output against input is fairly linear. Once the threshold is reached, the output starts to sag quickly as the device enters triode mode where no current is passed. The input against output voltage figure is shown in Figure 22.

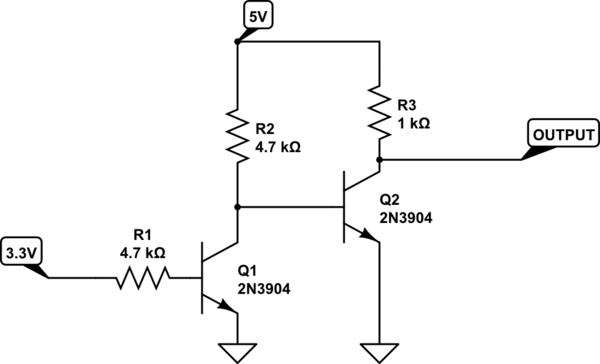


**Figure 22:** PFET Characterization Curve

For source-gate voltages of 500mV or greater, the trend holds fairly linear so it can be seen that the transistor is saturated. Around 500mV, the output begins to sag and for lower voltages, the output drops rapidly. This means that the transistor has crossed in triode mode and will no longer pass current. This point is the threshold voltage of the device. For this system, it is desirable that the threshold voltage is around 1V but 500mV is tolerable. With multiple transistors, the majority of them will be closer to 1V so they can be exchanged if threshold voltage becomes an issue.

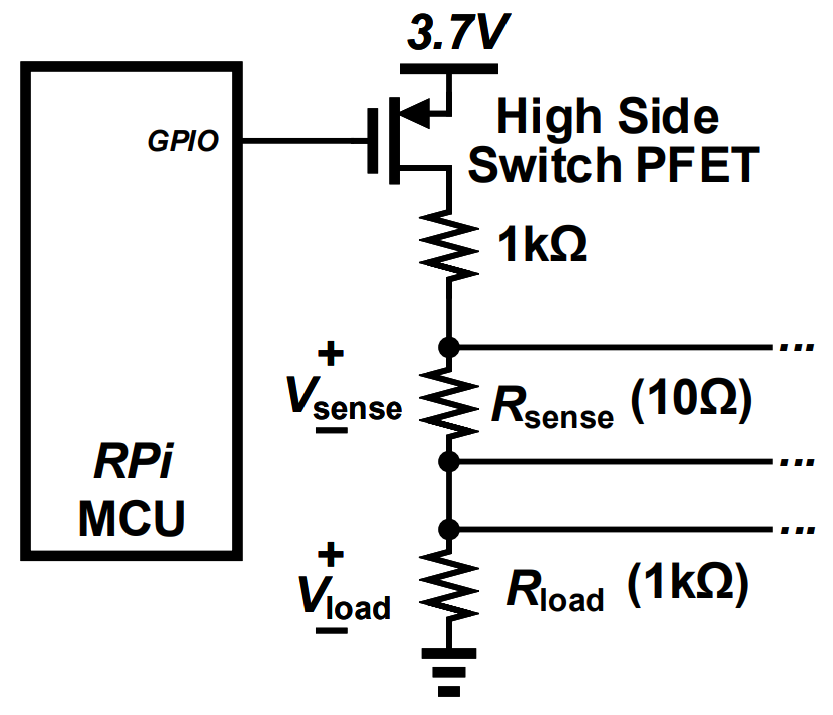
**3.4.3. Integration Challenges**

A potential concern is that a threshold voltage of 500mV may not be sufficient to effectively push the device into triode for higher current as discussed in Section 3.3.3. Since the MCU GPIO output is 3.3V, the source-gate voltage will be 400mV, since the threshold voltage is close to this value, it is possible that the device will operate in the subthreshold region for high current draw. This will have to be measured during integration to verify that the transistor can indeed block high current draw. There are two solutions if the transistor doesn’t operate as an effective switch. The first option is to use a transistor with a threshold voltage that is closer to 1V more consistently across process-voltage-temperature (PVT) variations than the NTP4101P. Another solution is to implement a level shifter. This will use several BJT transistors to adjust the DC operating point of the circuit to output a chosen voltage. In this case, 3.7V or higher can be chosen to ensure that the transistor is in deep triode to pinch-off any current. An example level shifter is shown in Figure 23.



**Figure 23:** Example of a Level Shifter

The PFET has also been added into the testbench. The PFET source is connected to the supply, the gate is connected to a GPIO pin, and the drain is connected to the series combination of resistors. The circuit is shown in Figure 24.

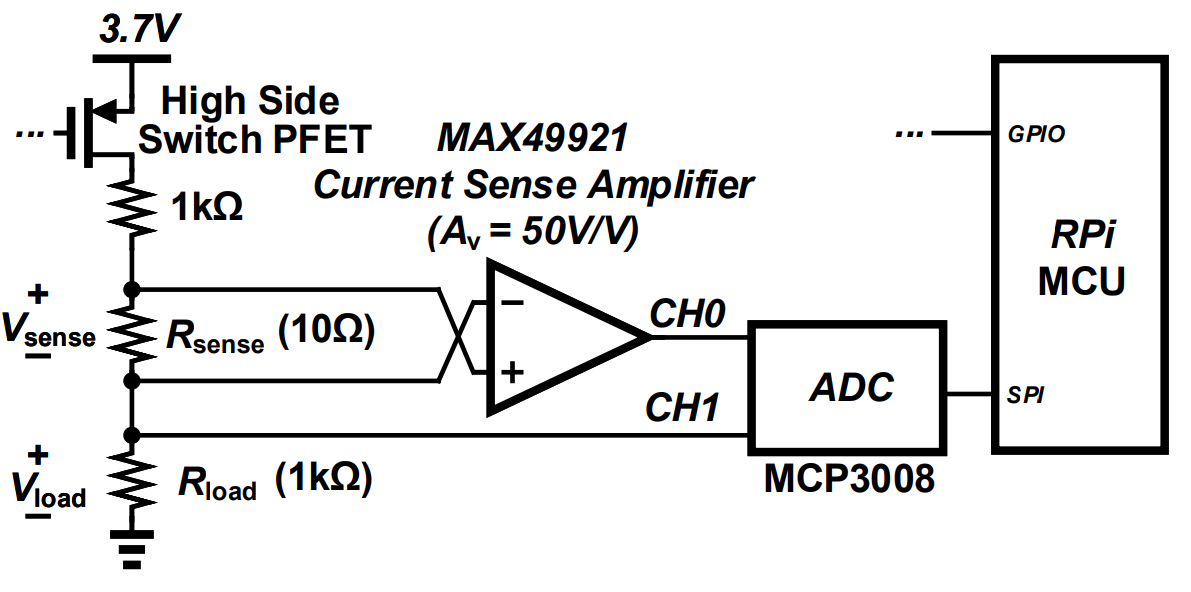


**Figure 24:** Modified Testbench

## 3.5. Final Test Circuit

### 3.5.1. Operation

With the addition of the PFET switch, the completed test circuit is shown in Figure 25.

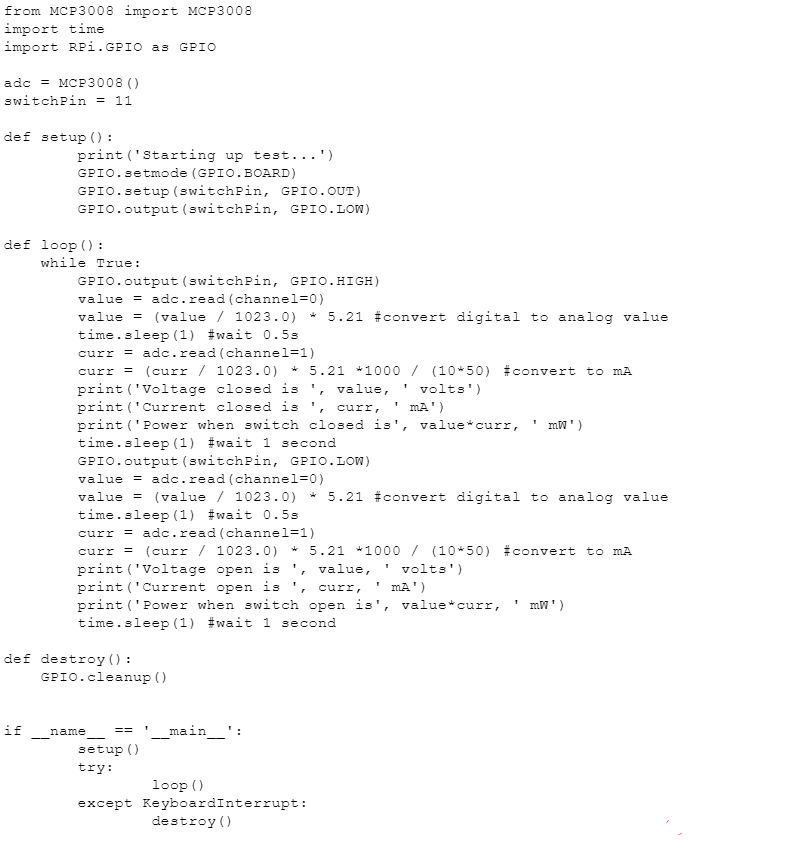


**Figure 25:** Finalized Testbench

Note that the ellipsis denotes that the PFET and the GPIO pin on the MCU are connected.

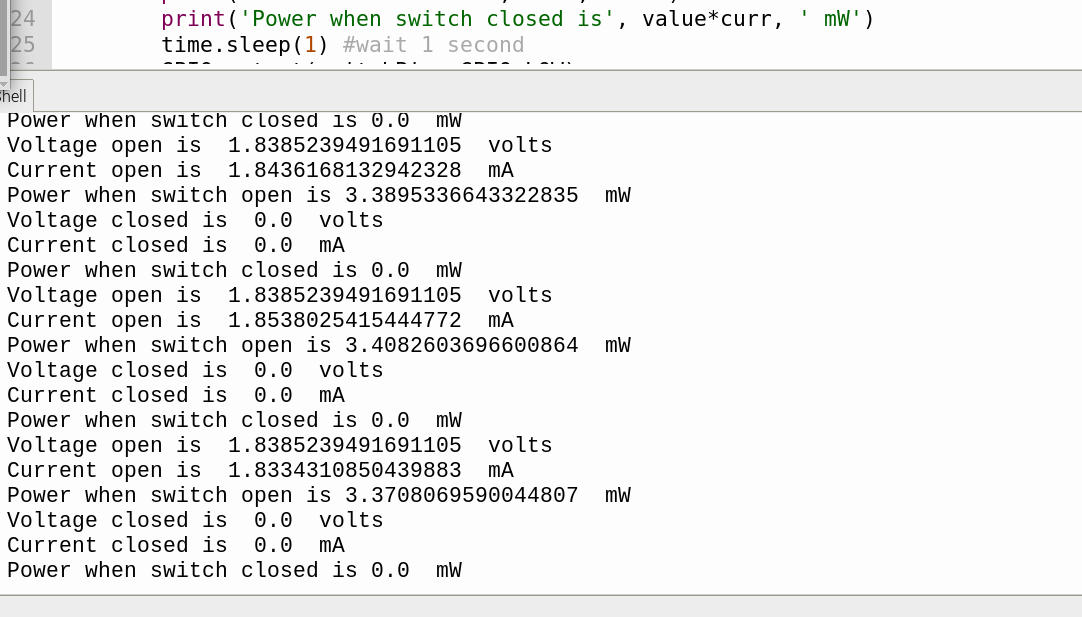
This circuit is to create an example scenario of how the measurement node would work in the integrated system. The PFET is to isolate the supply from the load and connect it when power to the load is desired. One channel of the ADC directly taps into the node voltage across the load to measure the voltage across it. The CSR is in series with the load and the CSA differentially taps the CSR to obtain the differential voltage and output the single-ended voltage to another ADC channel. The MCU records these values and converts the CSA output into a current value based on the known CSR and CSA gain. The power is then computed and all the relevant data can be displayed. We expect to see that when the GPIO for the PFET goes low that PFET is saturated and current is passed to the load. When the GPIO goes high, the PFET is pushed into triode and the current is pinched off resulting in no power transfer. If there is a measurable current or voltage when the PFET is off, the PFET is leaking and this problem needs to be solved according to the options given in Section 3.4.3.

### 3.5.2. Validation



**Figure 26:** Finalized Testbench Code

The code shown in the figure above is used to test the final test circuit. The code flow is described in Section 3.5.1. The output of the code is shown in Figure 27. It will repeatedly switch the PFET on and off until the user kills the program.



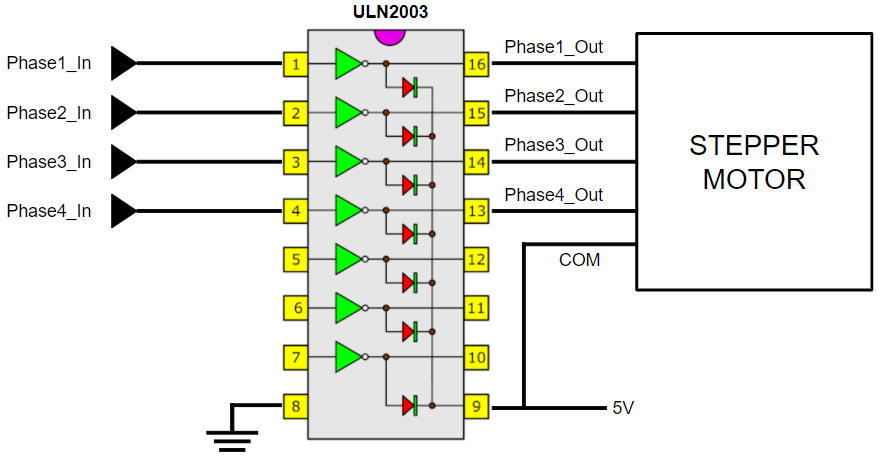
**Figure 27:** Finalized Testbench Code Output

It can be seen that when the GPIO pin 11 is high, the switch is off and no current is delivered to the load. When GPIO pin 11 is low, the switch is on and current is delivered to the load. Since the measured voltage and current is zero, we can see that there is no leakage from the supply when the switch is off. This might not be the case with high current drives. As discussed in Section 3.3.2, the power delivered to the load is around 3.4mW. The theoretical power transfer was calculated to be 3.42mW. The error for this testbench is 0.6% which is well within the accuracy specification. If testing different power transfers is desired, the supply voltage can be adjusted to avoid changing the passive components to further verify accuracy.

## 3.6. Stepper Motor and Motor Driver

### 3.6.1. Operation

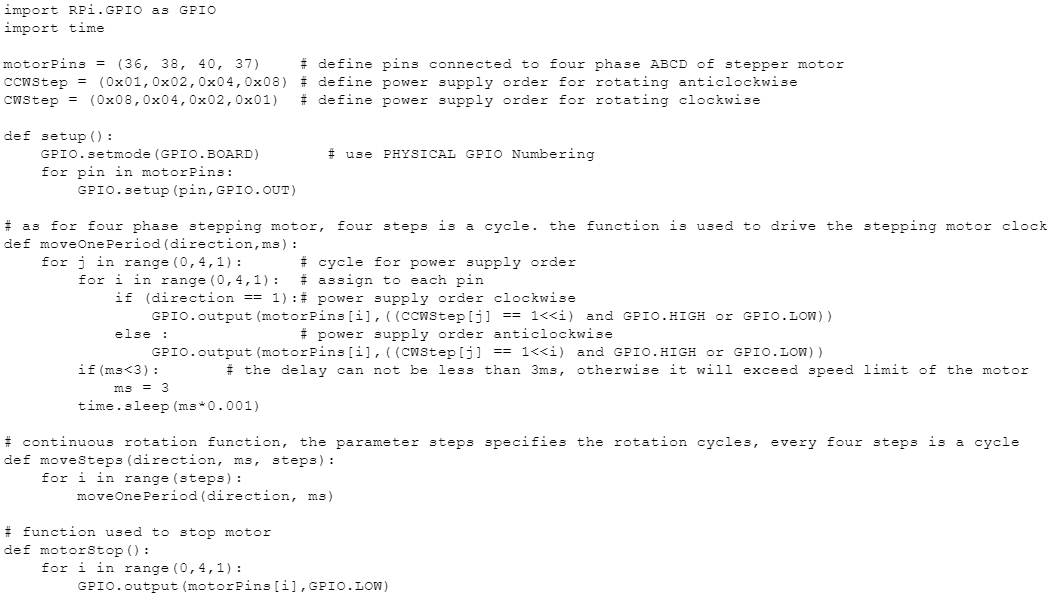
Since the system will track its efficiency as it charges the battery using the solar panel, the panel will need to rotate to be able to maximize its power output when operating in single-axis mode. To finely control its rotation, a stepper motor is used. Since the MCU can not directly drive the stepper motor due to its low power output and the motor’s large inductive loads, an external module must be used with a sufficiently large power supply isolated from the MCU to drive the motor. Because of the system’s miniature size and light solar panel, a smaller low torque stepper motor running off a 5V can be used. Since the stepper motor will be low torque, a module using Darlington BJT pairs such as the ULN2003 can be used to drive the motor. The module connections are shown in Figure 28.



**Figure 28:** ULN2003 Internal and External Circuitry

The inputs phase one through four can be driven from the MCU directly and can be coded to control the rotation of the motor.

### 3.6.2. Stepper Motor Library Code



**Figure 29:** ULN2003 Stepper Motor Driver Library Functions

The functions to control the stepper motor operation are shown in the figure above. To verify the motor operates, the motor driver circuit is connected to the MCU and a small 5V 2A wall adapter and a small testbench is used to verify that the motor can rotate 360 degrees in both directions. With 2048 steps per 360 degree rotation, that yields a 0.2 degree resolution of angular displacement which is more than sufficient for this system.

### 3.6.3. Integration Challenges

Since the solar panel will not receive sunlight at optimal angles during charging, further investigation must be done into the angle increments of rotation to best align the panel with incoming maximum sunlight. This can be thought of as flux through the surface of the panel. The optimal angle increment per unit time of time must be found and used to test battery charging during full system integration.

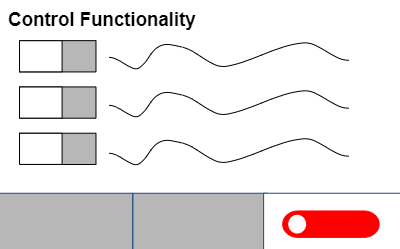
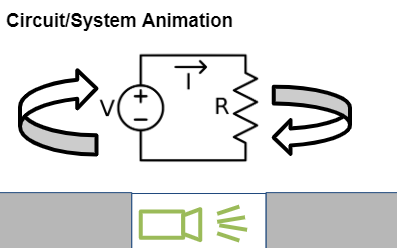
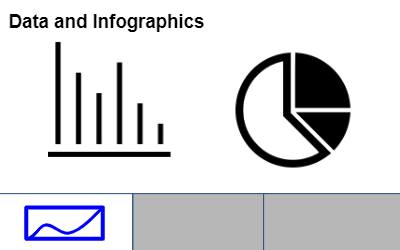
***3.7. Subsystem Citations***

Code for the library of the ADC and stepper motor were obtained via help from *Tutorials for Raspberry Pi*’s article on using the MCP3008 and *Freenove*’s starter kit section on stepper motors.

# 4. GUI & Database Subsystem

## 4.1. Subsystem Introduction

The purpose of this subsystem is to provide a means of storing data collected by the system and then relaying said data to users in an engaging and informative manner. Given the long charging period of using solar panels for energy generation, it is necessary to take the proper measurements to ensure characterization of the system over time. We record these voltage and current, and power measurements into a database. In this case, our database is in the form of a CSV file internally kept on the Raspberry Pi Model 4 B. The stored data is then extracted and processed into various graphs that give insight into the system’s operation over time. In order to display these graphs, alongside providing system animations and providing user input into the system, we designed a GUI using Python that is to be run on our Pi and displayed to a connected touchscreen. This GUI is segmented into three major tabs: The infographics tab designed to display the graphs created from the database as well as real-time measurements, the system-animations tab designed to provide a visual representation of the system in its operations modes, and the control systems tab designed to provide input operation to the system, giving users a more intimate way of interacting with our project.



**Figure 30:** GUI Skeleton

## 4.2. Database

### 4.2.1. Operation

The database is what allows us to monitor the various measurement points of our system over an extended period of time. Given that our system is powered by solar energy, the charging period to get our battery from fully discharged to fully charged is in the order of a few hours. Given our expected loads, it also takes multiple hours to fully discharge the battery. We therefore need to be able to monitor our system over this long period of time. The database system, as mentioned before, is simply a CSV file (or files if necessary) kept internally in the memory of the MCU (Raspberry Pi). The database takes input and is written to by a process detailed in the Electrical I/O subsystem, which monitors the voltage and current measurements at key points in our system. The database is then processed by a Python program run internally on the PI. This program uses the “Pandas” standard Python library to create database objects from the read CSV file. The database objects are then used as input into Matplot graphs, to properly display the requisite values as functions of time. These graphs are then properly outputted to the GUI, where they are displayed and then updated as neccesary.

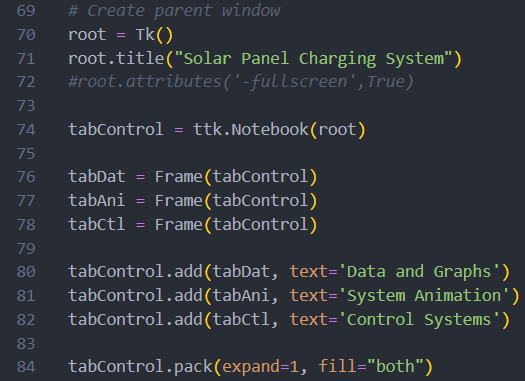
### 4.2.2. Validation

Given that we have not yet integrated our I/O and GUI subsystems together, we instead use a CSV file stored with dummy data in order to ensure our database and can properly read and process data from the CSV in the first place. In our case, we were able to successfully pull data from the database and partition it depending on what we wish to look at. Using a dedicated column in our CSV to mark the current operating mode of our system, we are able to show which data is collected when our system is either discharging or charging the battery, as well as show when our system is in fixed-axis or free-axis mode. This means we have validated graphs that properly display all the necessary information to the user.

## 4.3. GUI Outline

### 4.3.1. Operation

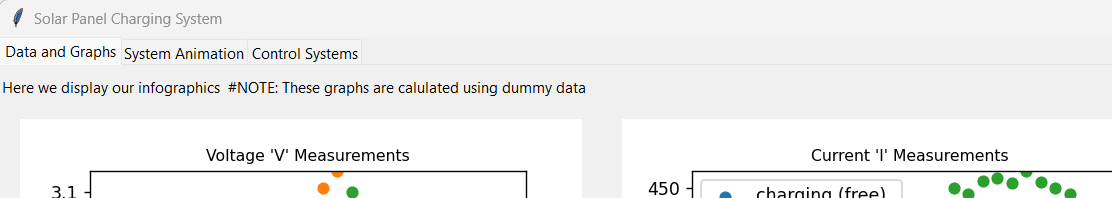
The GUI was designed by utilizing the ‘Tkinter’ standard Python library. Tkinter allows us to create a GUI with a tree-like structure, which is helpful for partitioning out the major elements of our GUI into discrete tabs. These tabs can be interacted with, along with the GUI itself, via a 7” touchscreen attached directly to the Raspberry Pi. This choice to use a touchscreen is due to the fact that it will draw less power overall, it will decrease the overall size requirement for the system, as well as be more accessible and familiar to young students, who are our designated audience.



**Figure 31:** GUI Skeleton Code

In the above code, we detail how we create the overall GUI outline. We create a root window using Tkinter, which we assign a title. We then create a ttk.notebook object, allowing the implementation of three tabs. It is also here where we would finalize our formatting for how the window should look. All of our testing has been done on personal computers, which creates an adjustable window that can vary in size. Once we properly test our connected touchscreen, the GUI will take up the entire screen and be formatted and spaced accordingly.

### 4.3.2. Validation



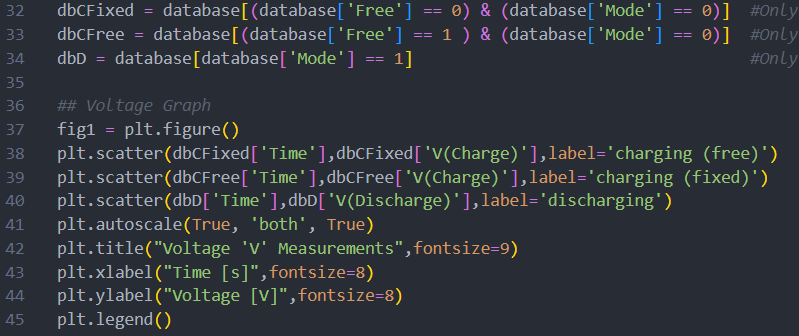
**Figure 32:** GUI Menu

Using the above code, we successfully create a Tkinter window that has discrete, labeled tabs. We are able to therefore separate all of the content we want to display on our GUI, saving valuable screen space. Once we format our GUI for the touchscreen, we expect to have the GUI functionality be better scaled and more easily navigated with respect to the smaller screen. We do this to ensure that our target audience of K-12 students are able to properly access all of the content without issue.

## 4.4. Infographics Tab

### 4.4.1. Operation

The infographics tab is used to display all of the interesting data about our system. It is in this section that we display the graphs that we had created, as mentioned earlier in the database section. We wish to plot three graphs to display here: voltage measurements, current measurements, and power calculations from the prior measurements. Using ‘Matplotlib’ to actually plot our data, we then utilize a special type of TK frame to properly display the graphs alongside individual keys and axes labels. Thanks to the ‘Pandas’ library, we are able to separate out our data on the basis of both column and row. For example, this means that we can isolate the voltage measurements for both the panel-side and load-side sections of the battery. We can then further isolate these columns by making a dataframe from only the tabs that have the ‘charging’ and ‘free-axis’ indicators. In practice, this means we can plot the charging and discharging of the battery while also highlighting when our system is free as well as when it is fixed.

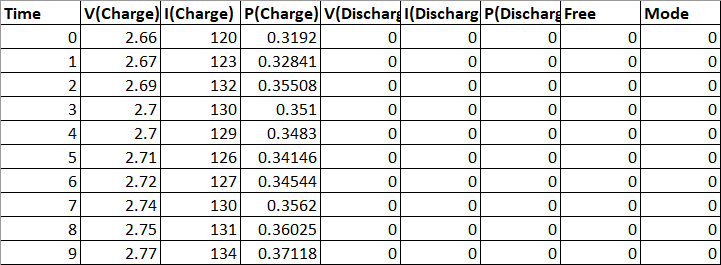


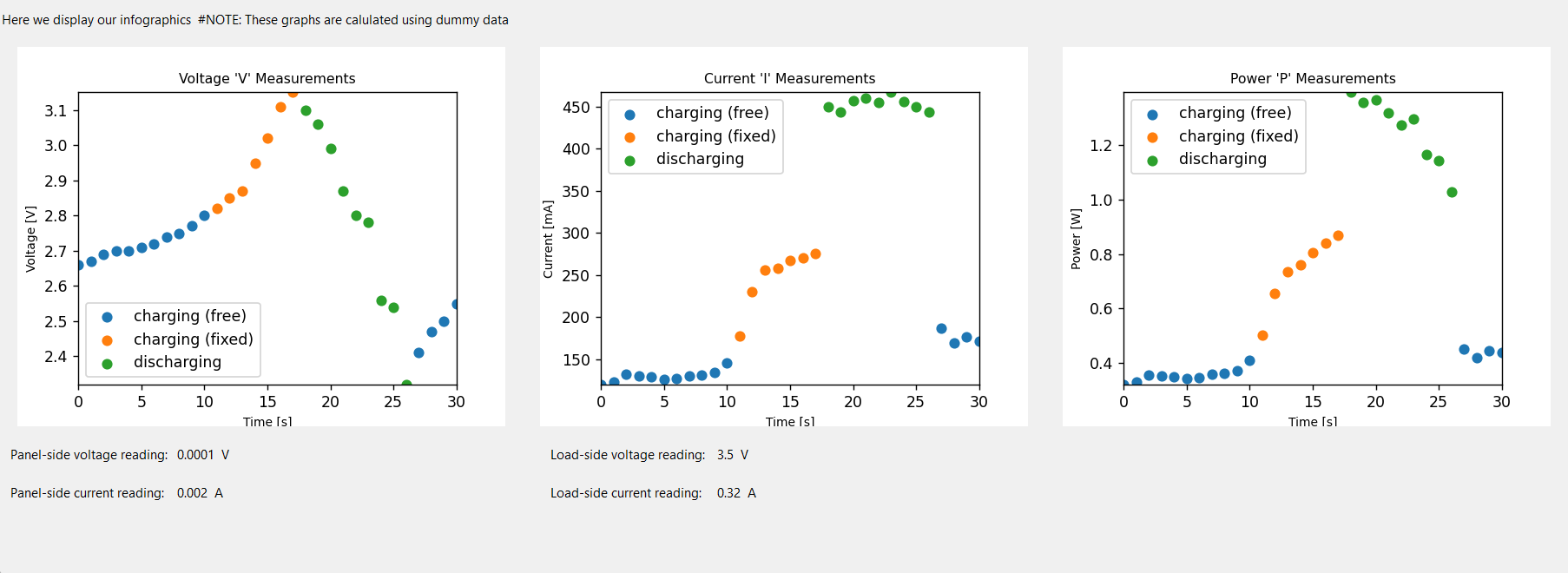
**Figure 33:** Infographics Tab Code

In the above code, we first recognize the CSV database file and then open it using ‘Pandas’. In our case, we are using dummy data collected into a CSV file, as we have not implemented the actual reading of the system via the I/O subsystem. We then create our plot by using different columns of the database. In this case, we create two different lines for our graph corresponding to the data in the charging voltage and discharging voltage columns of our CSV. Note how we need only specify the title of the column to extract that data. In the case of our finalized graphs, we would create a sub-database for each voltage, based on the individual rows and if the system is in fixed or free axis operation. This would allow us to create a voltage graph with 3 unique lines: voltage while charging in free-axis, voltage while charging in fixed-axis, and voltage while discharging. This process is repeated for the current and power columns, creating two more respective graphs to display on our GUI. We also include live readings for voltage and current on both our panel-side and load-side system.

### 4.4.2. Validation

**Table 1:** Excerpt from Example Database CSV with Dummy Data





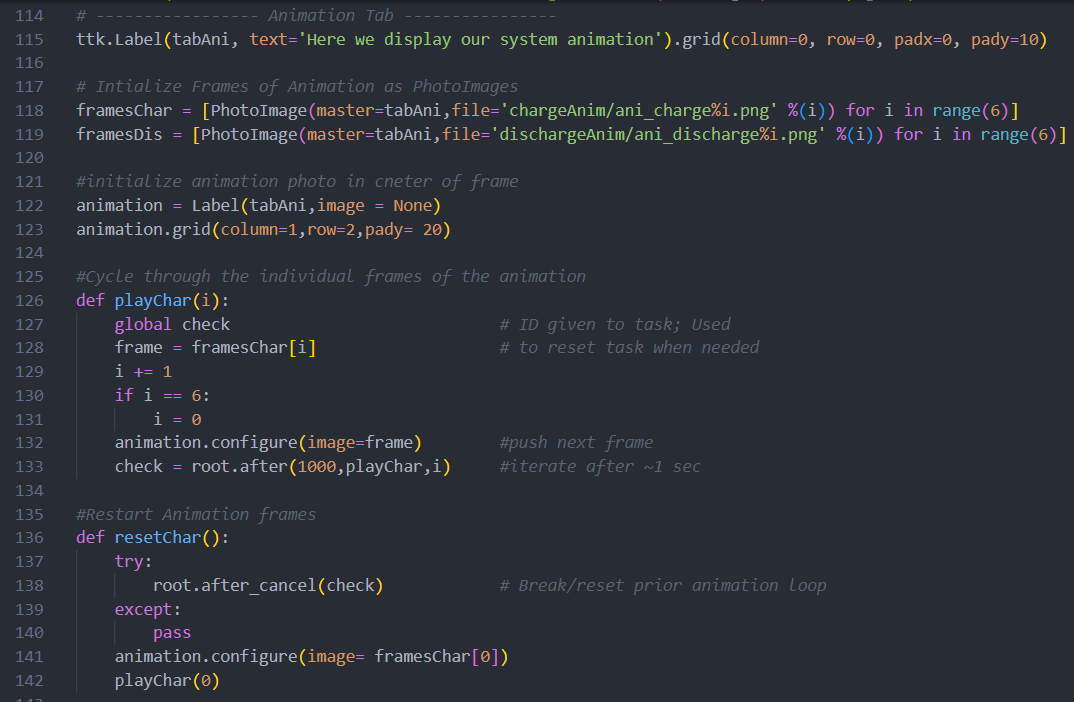
**Figure 34:** GUI Infographics and Database Display Tab

Using the dummy data previously given, we were able to successfully create and display the requisite graphs we intended to show. It is important to note how these graphs will be far more filled, given how our database plans on storing data over many hours rather than a few seconds as displayed here. We also were able to output the real-time current and voltage measurements at the bottom of this tab. In our case, since we do not have the proper I/O integration, these values come from hard-coded variables in our program, whereas in real operation, the values come from the real-time inputs specified in the electrical I/O subsystem.

## 4.5. System Animations Tab

### 4.5.1. Operation

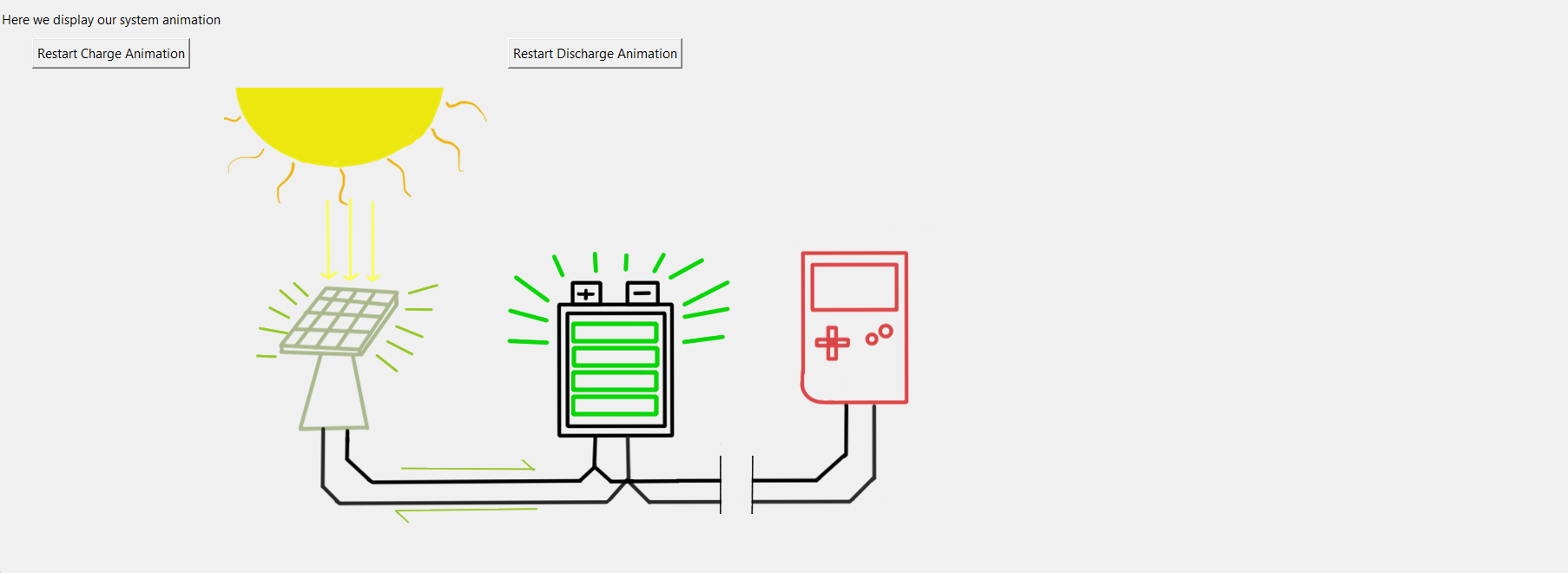
This tab is designed to house the system animations created by our team for the sake of giving a simple, visual aid as to the operation of the system in its respective modes. In actuality, this means that this tab consists of nothing more than the animation itself, as well as a button to add functionality to restart either of the animations when necessary. When pressed, the button will reset the animation or start the other animation, depending on the button pressed.



**Figure 35:** Animations Tab Code

In the code above, we begin by initializing all of the frames of our animations as separate *tkinter* ‘PhotoImage’ objects. This allows us to iterate through each picture as though there was an animation. We then initialize the center image of the frame to place our actual animation. Following this are the functions used to actually iterate through the pictures, indexing through the array with a ~1 second delay. The process is repeated for the discharge animation as well. Finally, we create two buttons designed to initialize and reset the animations when pressed.

### 4.5.2. Validation



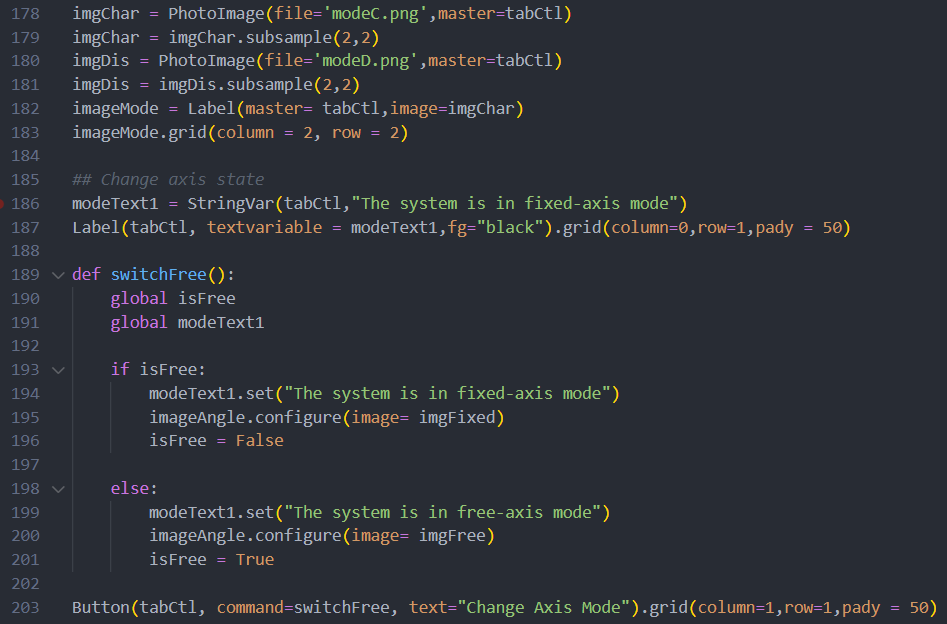
**Figure 36:** GUI Animations Tab

Using the above code, we are able to successfully create the animation and iterate through the frames at a steady pace. There is the slight issue that occasionally, the frames between the two animations will overlap/clip into one another. This is due to the ‘after’ command having a slight delay before loading the next animation frame. To fix this, one would only need to have a check from the user input. If the button is pressed, a flag would go off that would tell the loop that the next iteration begins back at frame 0.

## 4.6. Control Systems Tab

### 4.6.1. Operation

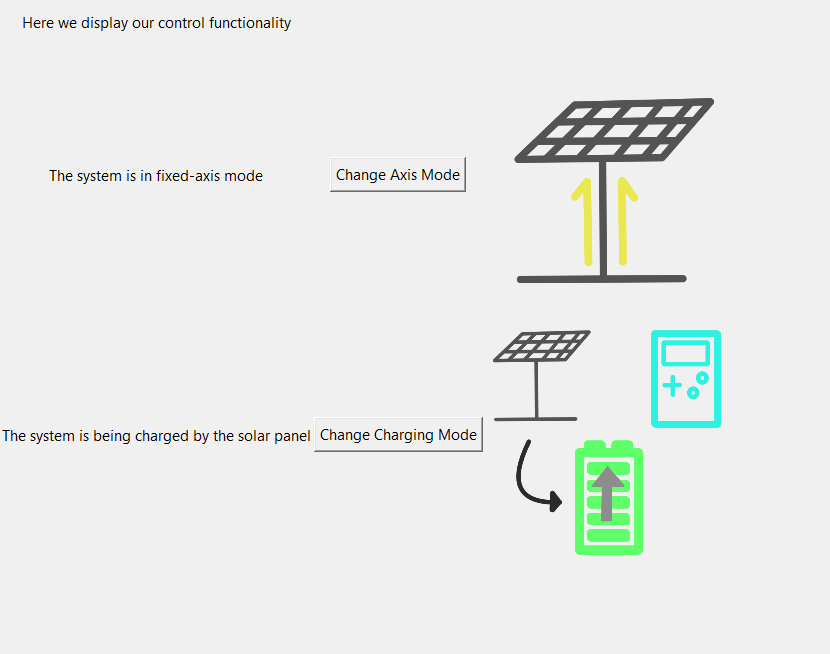
The control systems tab contains the various ways in which our student can interact with the system. Currently, the only things that we determine to be viable user input is the setting of two operation modes. The first option is the operating mode of the solar panel itself. The panel can either be operated in fixed-axis mode, or in free axis mode. When the user switches the button to free-axis mode, the solar panel would then be able to move in response to the system signals, most likely moving as it receives a periodic signal to move X degrees at a time. When the system switches to fixed-axis mode, it could either remain still from the point of activation, or move to a set angle, for instance, 90 degrees straight vertical. The other operating mode that can be changed is whether or not our system is in charging or discharging mode. The user can switch between the two stages freely, either having the panel and battery be connected or having the battery and load be connected. It is important to note that this system greatly intertwines with the workings of the I/O subsystem. As such, the GUI serves to only change interval variables and raise internal flags, which are then processed and fed as input into the system.



**Figure 37:** Control Systems Tab Code

In the above code, we first start by initializing some icons we will use to represent the different modes that our system can take. This is done by creating ‘PhotoImage’ objects which will then be displayed. For our label, the text is stored in a special string variable, allowing us to change the text displayed as we interact with the button. We then define functions that actually change the state of the system when pressed. In this case, the above code covers the first of two buttons. Notice how the function not only updates the text and image in the GUI upon interaction, but it does so by changing the state of a global variable. This allows us to use said global variable to change the system state elsewhere, namely in the code of the I/O subsystem. If need be in the future, we can also create a more complex, in-depth function tied to the button, allowing for more complex interaction on the user’s part.

### 4.6.2. Validation



**Figure 38:** Control Systems Tab

As shown above, the GUI successfully creates the desired label, button, and image all in tandem. Upon pressing the button, we successfully change the text along with changing the icon to another image to represent the new state. We also see a change in the internal variables of the system, allowing us to verify that the system itself is changing outside of the GUI. For future iterations, we plan to not only better format the buttons and text and images to better fit the constraints of our touchscreen interface, but we also leave plenty of space in this tab to add any additional functionality as needed.

## 4.7. Integration Considerations

### 4.7.1. Raspberry Pi Touch Screen

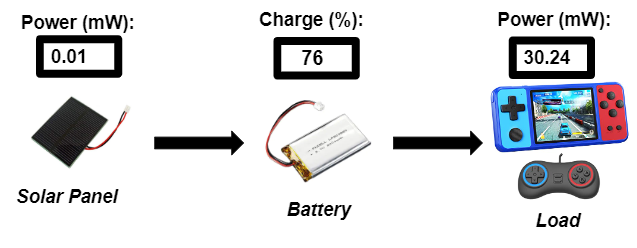
When integrating the final system, the final decision had been made to switch to a Raspberry Pi touch screen that will be used as the main form of communication between the system and the user. As such, a lot of the formatting and sizing and overall visual design of the GUI was limited by running the Python code on normal-sized monitors. The main GUI elements will be readjusted to be easily accessible on the touch screen but the basic functionality and operation will remain the same.



**Figure 39:** Raspberry Pi Touch Screen

### 4.7.2. Infographics

An additional suggestion has been given by the project sponsor to add an additional infographic for the GUI which should be more visually appealing to our target audience of K-12 students. While the current graphs and measurements are definitely informative, there is the chance they might not be seen as engaging. As such, the project sponsor advised the creation of a better visual aid to use in tandem with our real-time measurements. In theory, this visual aid would consist of a simplified circuit diagram of our system. On this diagram, we could include text that details the real time power draw both from the battery into the load as well as from the solar panel into the battery. This would be displayed alongside a measurement of the current battery voltage. This visual aide would be beneficial to those not as familiar with electronics, as the concept of voltage and current may be misleading. By diagraming our system in terms of power consumption and generation, there is a more realistic comparison between our system and real world electronics, allowing for a more comfortable understanding of the system.



**Figure 40:** Additional Display Feature Skeleton

### 4.7.3. Visual Quality

After we have validated and finalized all functionality for the GUI subsystem, we will be able to work on improving the overall design quality of our GUI. This should not be an entirely impossible task, as Tkinter allows for the usage of preset ‘themes’ to automatically change the look and feel of all the included modules in the GUI. This means that if we decide to design our own theme, we can then easily apply said theme to our system and be left with a uniquely beautiful GUI that is both streamlined and engaging.

# 5. Subsystems Conclusion

In summary, all the subsystems have been fully tested and completely validated. All the necessary elements are functional and are ready for integration. The next steps involve code development for the combination of all of the subsystems, test of circuit elements on board, and the quality of life improvements for the GUI to be implemented on the touch screen for the minimum system footprint.