SMART DC WALL OUTLET DESIGN WITH IMPROVED LOAD VOLTAGE DETECTION

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements of the Degree of Joint Bachelor of Science and
Master of Science in Electrical Engineering

by
Patrick Donovon Granieri
June 2019

COMMITTEE MEMBERSHIP

TITLE:	Smart DC Wall Outlet Design and Load Voltage Detection
AUTHOR:	Patrick Donovon Granieri
DATE SUBMITTED:	June 2019

COMMITTEE CHAIR:	Taufik, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Tina Smilkstein, Ph.D. Associate Professor of Electrical Engineering
COMMITTEE MEMBER:	Majid Poshtan, Ph.D. Associate Professor of Electrical Engineering

ABSTRACT

Smart DC Wall Outlet Design with Improved Load Voltage Detection

Patrick Donovon Granieri

A standard home in the United States has access to the 120V AC power grid for use with home appliances. Many electronics used at home are powered by a DC power supply, which loses energy in the conversion from AC power. The DC House project avoids any conversion between AC and DC by storing energy in batteries as DC power and supplying it directly to DC appliances. While AC systems feature a standardized output voltage, no such standard exists for DC systems. The Smart DC Wall Outlet solves this by automatically adjusting its output voltage to meet any required DC load voltage. A hardware solution was developed using a microcontroller in tandem with a DC to DC Buck converter to monitor trends in the output current and set the output voltage accordingly. The Smart DC Wall Outlet features two 100W output channels that were able to correctly identify the required output voltage of five out of seven test devices. Results indicate that it is possible to generalize the turn on characteristics of DC devices, but that other solutions may find more success.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Taufik for showing me the true utility of power electronics. I would also like to thank my mother and my sister, Barbara and Rebecca Granieri, along with the rest of my family for helping me through university and encouraging me every step of the way. Last, but not least, I would like to thank my friends for being my support when I was in need. Without these people, none of this would have been possible.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

1. INTRODUCTION

Electric power distribution became a necessity in the late 1870's with the advent of arc lamp street lighting systems [1], eventually leading to the rise of electric lighting, climate control and other home appliances available to the modern American home. High voltage alternating current (AC) systems were developed to distribute power to these arc lamp street lights and low voltage direct current (DC) systems were developed to deliver power to indoor incandescent lights. While both forms of power distribution are used in different applications, AC systems were ultimately favored as the primary transmission method due to the existence of the transformer. To reduce the power losses in the transmission lines, power distribution systems try to minimize the amount of current being transmitted through the inherent resistance of the transmission lines. By using transformers, AC systems could step-up and step-down the transmission voltage as needed to minimize the current being transmitted over long distances. DC systems had no alternative to the transformer at the time, leading to a maximum transmission range of less than two miles [2]. As a result, AC power distribution became the standard in the United States and the rest of the world.

While AC systems are used to distribute power across long distances, many devices in the home do not use AC to power their internal circuitry. Whether it be a computer, a television, an LED light or a smartphone, these devices accept the AC voltage from a wall socket and rectify it to a safe DC voltage that the device can use. This rectification comes at a price, as there is always some power lost in the process. Additionally, any device that uses a transformer in the rectification process will

continually draw power from the wall socket even when not in use. Some devices like game consoles and cable boxes waste upwards of 20 watts when not in use [3]. By distributing power with a DC system instead of AC, these inefficiencies could be eliminated.

The DC House Project is a humanitarian effort that focuses on creating a small, independent electrical grid that is entirely powered by DC [4]. All power for the system is generated locally using renewable methods, including DC water turbines, DC wind turbines and solar panels. The power is then stored on-site using batteries, preventing the losses incurred by transmitting the power over long distances. Figure 1 presents a simplified illustration of the DC House Project [5]. This system allows for people who live in rural areas without easy access to a centralized power grid to generate their own power independent of the state or utility companies. By removing the unnecessary conversions from DC to AC, then AC back to DC, the system is made more efficient and can run longer on a single battery charge.

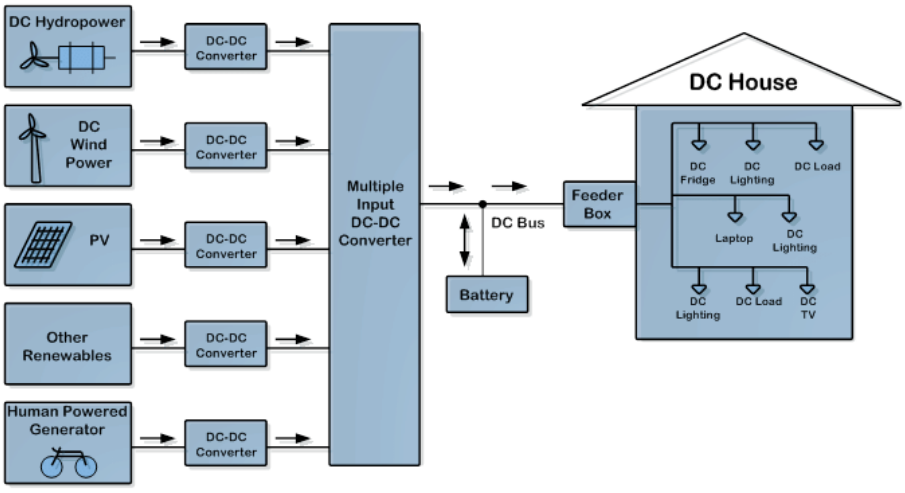

Figure 1: DC House Block Diagram

2. BACKGROUND

2.1. The Need for a Variable DC Wall Outlet

One of the aims of the DC House Project is to provide power to DC devices with the same convenience as their AC alternatives. While AC powered devices operate with a wide range of internal voltage levels, they all accept the same input voltage of $120V_{rms}$ AC in the United States. AC devices benefit from a standardized wall outlet in every country, but there exists no such standard for DC devices. Some DC devices operate using dry cell batteries, accepting input voltages of 9V or in increments of 1.5V (e.g., 1.5V, 3V, 4.5V, etc.). Others accept 12V from car batteries or even 5V from USB chargers like those used by smartphones. As there is no standard for what voltage a DC device should use as its input, the DC House must be able to accommodate all these input voltages. Installing a separate wall socket for each input voltage would be inefficient, as there are so many possible values that the number of sockets needed would be impractical. A DC wall outlet with a variable voltage setting would be a much more appropriate solution.

2.2. DC to DC Voltage Converters

The primary reason that AC power distribution systems were favored over DC systems was due to the existence of the transformer, allowing AC systems to easily change their transmission voltage. DC power systems had no easy method of doing this in the late 1800's, but with the rise of transistors and integrated circuits alternatives have emerged. Known as DC to DC converters, these systems accept a DC voltage as an input

and change it to another voltage with minimal power loss. There are multiple topologies of DC to DC converters that allow for stepping up or stepping down an input voltage. The DC House Project features a 48V power rail inside the house, so the wall socket being designed in this paper will focus on an architecture that steps down the input voltage as most loads require an input voltage less than 48V. The Buck converter is an example of a DC to DC converter that transfers power from the input to the output while decreasing the voltage level. The power stage of a Buck converter is shown in Figure 2.

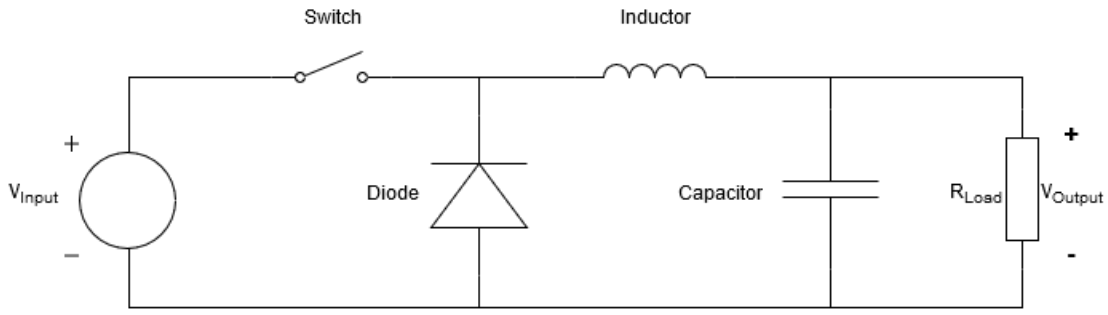

Figure 2: Buck Converter Topology

The Buck converter controls the voltage across the output by rapidly charging and discharging the inductor. The voltage across an inductor is proportional to the change in the current over time, as shown in Equation 2-1 [6].

$$V = L\frac{di}{dt} \tag{2-1}$$

The switch, usually implemented as a MOSFET, is connected to a controller which monitors the voltage at the output. When the switch is closed by the controller a voltage is induced across the inductor, causing it to conduct more and more current. This current must go through the capacitor and the parallel resistive load, causing the output voltage to

go up with the increasing current. When the switch is opened by the controller the input source is disconnected from the circuit, causing the inductor to release the energy that it collected while the switch was closed. The polarity of the voltage across the inductor flips during this period, causing the current through the inductor and the voltage across the output to steadily decrease. The controller uses a feedback network connected to the output to determine when to close or open the switch as necessary to maintain a desired voltage level.

This is a simplistic model of a Buck converter that assumes ideal components are being used. As a more robust model, a capacitor would be attached in parallel to the input source. This would aid in maintaining the input voltage at a constant level, as the Buck converter draws large transients of current from the input and the capacitor helps to filter these out. Furthermore, the diode in use in this model will be a source of power loss when conducting due to its ~0.7V forward voltage drop. This can be replaced with another MOSFET that is synchronously turned on and off by the controller, as it is more power efficient in high current applications.

2.3. Previous Work

The Smart DC Wall Plug project began with the Senior Project of Michael Detmers and Tyler Blauvelt [7]. Their design was based on a different DC to DC converter topology (a Flyback Converter) and featured a single channel with a maximum power output of 80W. The DC to DC converter allowed for four discrete output voltages of 5V, 12V, 19V and 24V for different appliances. In order to select which output voltage

was desired the user needed to manually select a pin. This was undesirable, as selecting the wrong pin on accident could destroy the load device and potentially harm the user.

Edward Sibal continued work on the Smart DC Wall Plug with his proof of concept thesis [8]. This research effort pushed for automating the wall outlet voltage selection, removing the need for the user to select a voltage for the load device. This DC to DC converter used a Buck topology and implemented a microcontroller with a digital potentiometer to vary the output voltage. The voltage of the load device was determined by increasing the output voltage until a spike in the current was detected, upon which the final voltage was set. This method was able to turn on the load device in most cases, but the determined voltage was likely to be significantly below the device's nominal voltage. This would result in higher current draw than normal, causing unnecessary power losses.

Kevin Mendoza focused on improving the project by condensing all the components onto one board and attempted to improve the voltage finding algorithm [9]. While still using the Buck converter, microcontroller and digital potentiometer setup, the new algorithm focused on passing a current threshold at a reference voltage and setting the voltage based on the amount of current being drawn. This method limited the output voltage to only 3V, 6V and 12V, and required a user input to start the process. The algorithm lacked flexibility and was only able to correctly determine the nominal voltage for a small range of devices.

The most recent improvement on the project was implemented by Richard Liu to expand the output voltage range and make the output more accurate [10]. The output range was significantly increased, allowing for output loads from 3V to 36V in discrete steps. In the previous two implementations, the use of a digital potentiometer to adjust the

output voltage resulted in an inverse relationship between the two. This made the output voltage quite sensitive to small steps in the resistance in the higher voltage ranges, as seen in the upper left corner of Figure 3. To help make the output in the higher voltage range more accurate, Liu added a separate current source that could compensate for the lack of resolution. The algorithm for detecting the load's nominal voltage was not changed, and instead assumed that the output voltage had already been detected.

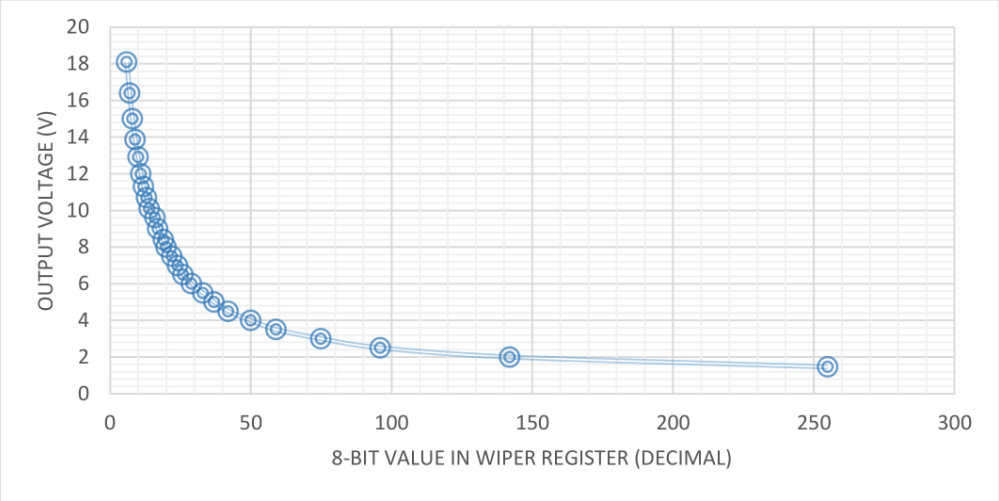

Figure 3: Potentiometer Value vs. Output Voltage [9]

The goal for this thesis is to improve the algorithm for detecting the load's nominal voltage, along with investigating an easier method of changing the output voltage. Potential DC loads will be grouped into different profiles based on their current and voltage characteristics, allowing for an algorithm that can better predict the nominal voltage. The digital potentiometer and current source will be removed from the feedback network and the microcontroller will instead communicate directly with the Buck controller to adjust the output as necessary.

3. DESIGN REQUIREMENTS

3.1. General Design

The Smart DC Wall Outlet functions as a multiple input, multiple output system. It accepts the 48V line voltage from the DC House battery as an input, as well as the voltage and current characteristics of the loads plugged into each channel of the outlet. The system then outputs the calculated voltage level on the appropriate channel. A simplified diagram of the system is shown in Figure 4.

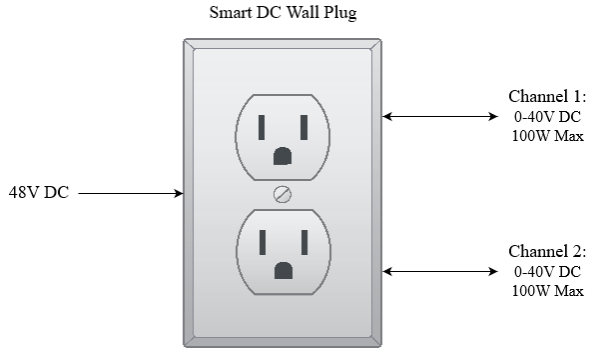

Figure 4: Level 0 Block Diagram

A more detailed view of the system can be seen in Figure 5 with the power path indicated by the bold arrows. Fuses are located at the 48V input to the system along with both the outputs to protect the internal circuitry and any loads from a potential fault condition. The Primary DC to DC Converter will accept the power from the 48V input and generate an appropriate voltage on both output channels. To adjust for any errors in the output, the Primary DC to DC Converter monitors the voltage and current in use by the loads. The Secondary DC to DC Converter is a fixed output step-down converter that takes the 48V

input and produces a 5V source that can power the Microcontroller. The Microcontroller communicates with the Primary DC to DC Converter to gather information about the voltage and current needs of the loads. Using this information, it calculates the load's nominal voltage and commands the Primary DC to DC Converter to adjust each output channel accordingly.

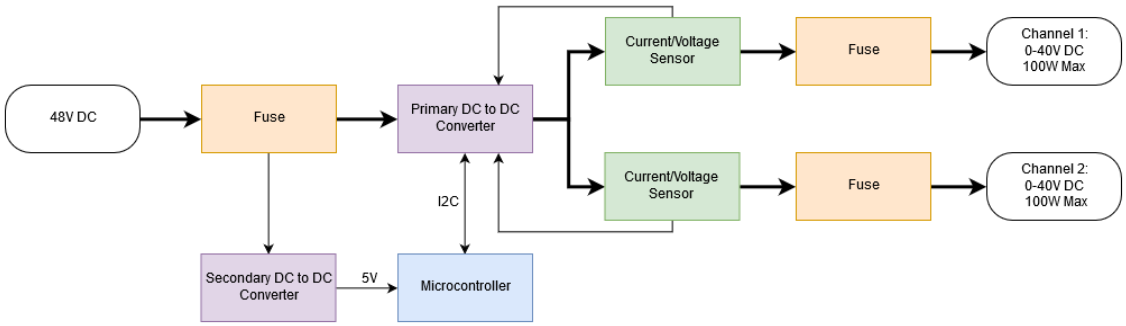

Figure 5: Level 1 Block Diagram

3.2. Electrical Specifications

The electrical requirements of the Smart DC Wall Outlet are dictated by the DC House specifications and the most common needs of available DC powered devices. The DC House supplies a 48V power rail, so that will be used as the input to the smart outlet. The output voltage range must be able to cover a wide range of devices, and studies were done in previous iterations of the project to find a suitable range. Sibal [8] and Liu [10] found that many DC devices were between 3V and 24V; to allow for some extra flexibility, the output range of this project will be from 1V to 40V. The output power on each channel will be set to 100W to accommodate the USB-C maximum of 5A at 20V [11]. The maximum load current will be set to 6A to prevent unnecessarily wide trace

widths on the printed circuit board, and the minimum load current will be set to 0.25A to maintain adequate efficiency and ensure proper operation of the Primary DC to DC Converter. The target efficiency of the system will be 85% at full load, though this will likely suffer under low current applications. The maximum output voltage ripple will be set to 2.5%, and the voltage error will be within 2.5% of the nominal setting. This is to protect load devices from overvoltage conditions. These electrical requirements are shown in Table 1.

Table 1: Electrical Design Requirements

Category	Requirement
Input Voltage	48V
Output Voltage Range Per Channel	1-40V
Output Current Range Per Channel	0.25-6A
Maximum Power Per Channel	100W
Efficiency at Full Load	≥ 85%
Output Voltage Ripple	≤ 2.5%
Steady State Voltage Error	≤ 2.5%

3.3. Physical Specifications

The Smart DC Wall Outlet project is designed to use the size and form factor of the standard wall outlet in the United States. This is defined by the NEMA standard for a duplex wall outlet, as shown in Figure 6 [12]. The maximum size of the final system is set by the size of this faceplate, which has dimensions of 3.12” by 4.87”.

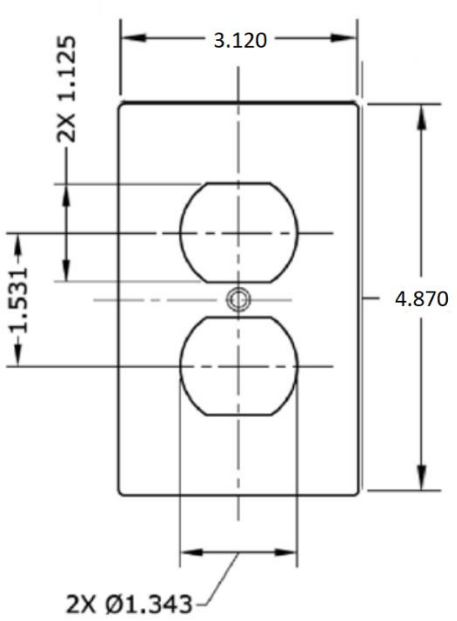

Figure 6: NEMA Wall Outlet Standard (Inches)

Additionally, the system must output power to the same plugs used by this system. In a traditional AC configuration, one prong is used for the hot wire, one is use for the neutral wire, and one is used for ground. DC systems only require a positive supply wire and a ground wire, so the third prong is redundant. In this setup, the DC supply wire will be wired to the hot prong, and the ground wire will be connected to both the neutral and the ground prongs of the receptacle. This configuration is better illustrated in Figure 7 [10]. All physical requirements are listed in Table 2.

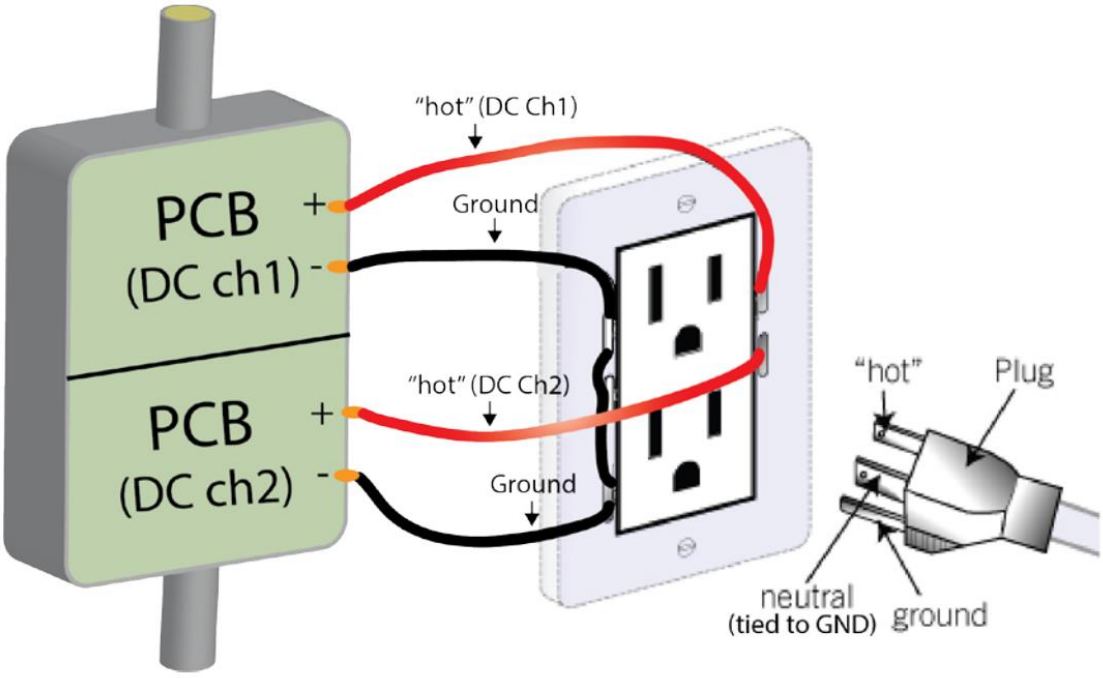

Figure 7: Smart DC Wall Outlet Wiring Diagram

Table 2: Physical Design Requirements

Category	Requirement
Maximum Width	3.12”
Maximum Height	4.87”
Form Factor	NEMA Duplex Wall Outlet

4. DESIGN AND SIMULATION RESULTS

4.1. Solution Statement

The two primary functions of the Smart Wall Plug project are to detect the optimal supply voltage for each load and to deliver that voltage efficiently. The output voltage for both channels is provided by the hardware of the system, featuring two individual Buck converter topologies as outlined in Figure 2 and a Buck controller integrated circuit (IC) to manage them. The algorithm for detecting the nominal voltage of the loads is implemented in software on a microcontroller. This microcontroller can communicate with the Buck controller IC, allowing it to set the output voltages and request the output current measurements for both channels.

4.2. Primary and Secondary Buck Controller Selection

The primary Buck controller is one of the most essential pieces of hardware in this build, and its selection should be taken with great care. The input and output range of the controller must at minimum accommodate the specifications listed in Table 1. In addition, a digital communication interface is desired to reduce the total number of components required on the board. The LTC3889 is a 60V Dual Output Step-Down Controller that fits these requirements [13]. It features an input voltage range from 5V to 60V, and two independent output channels ranging from 0V to 40V with a 10mV resolution. The controller uses a version of the Inter-Integrated Circuit (I2C) serial communication protocol known as the Power Management Bus (PMBus), which allows a microcontroller to digitally set target output voltages and monitor telemetry [14]. The

controller's digital architecture aligns with one of this project's main goals of reducing the complexity involved in changing the output. An example application circuit can be seen in Figure 8. This IC can use a switching frequency ranging from 85kHz to 500kHz, where higher switching frequencies allow for smaller inductors and capacitors at the cost of power efficiency. The smart DC wall outlet uses a switching frequency of f_{osc}=300kHz to reduce the physical space needed by components while still maintaining a respectable efficiency.

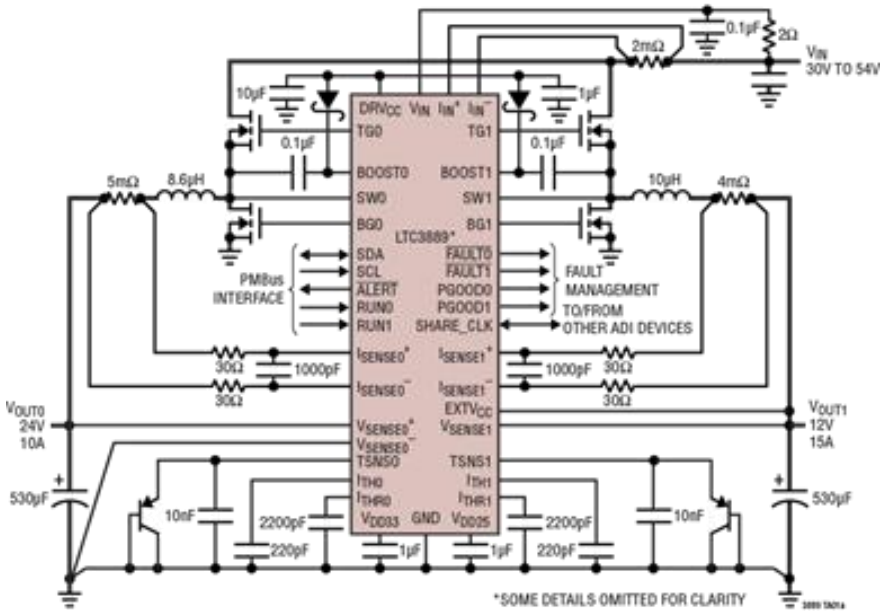

Figure 8: Example LTC3889 Circuit Schematic [13]

While the primary Buck controller manages the output voltage to the wall plug, some components require their own low voltage power supply on the board. Microcontrollers and status LEDs typically operate off a 5V supply, and the LTC3889 features an external power input that can use a 5V supply to increase the efficiency of its internal LDO regulators. A prospective Buck regulator would need to accept a 48V input

voltage, output 5V, and be able to supply at least 250mA conservatively assuming 100mA for the microcontroller, 100mA for the LTC3889 and 50mA for three status LEDs. The LT8619-5 is a 60V, 1.2A Synchronous Monolithic Buck Regulator with a fixed output of 5V that meets these specifications [15]. Since this chip features internal switching components fewer parts are needed to operate the IC, as can be seen in Figure 9. The LT8619-5 can use a switching frequency anywhere between 300kHz and 2.2MHz. This chip was configured to use a switching frequency of 1.4MHz to minimize the size needed by the inductors and capacitors, as the efficiency of this regulator is not as important as the primary Buck converter.

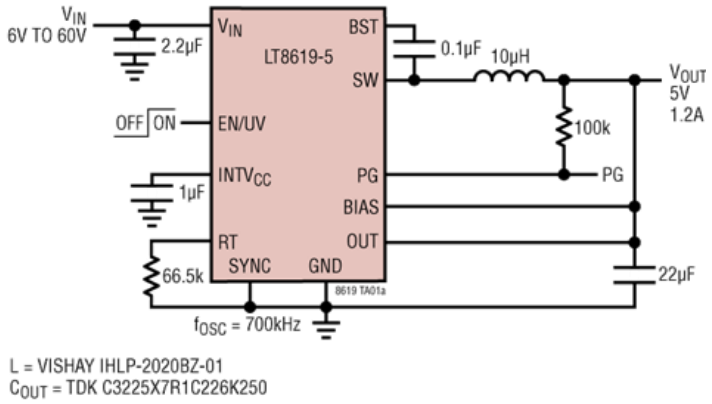

Figure 9: Example LT8619-5 Circuit Schematic [15]

4.3. Microcontroller Selection

The microcontroller used in this project must be able to read and write to the numerous digital ports on both the LTC3889 and the LT8619-5, along with controlling three status LEDs. The LTC3889 requires two ports for use with the I2C-based PMBus system, along with seven general purpose input/output (GPIO) pins. The LT8619-5

requires one GPIO pin to be monitored, and the three status LEDs will require one GPIO pin each to turn them on and off individually. In total, the microcontroller will need 11 GPIO pins and two pins for use with I2C. The microcontroller must also feature a relatively quick internal clock to process calculations on floating point numbers, enough memory to store the load voltage detection algorithm, and fit within the physical constraints listed in Table 2.

The ATmega328P on the Arduino Uno board was chosen to be the microcontroller for use in the smart DC wall outlet. The Uno features 20 GPIO pins, two pins with dedicated hardware for processing I2C connections, 2KB of SRAM and 32KB of Flash Memory [16]. The microcontroller uses a modest internal clock speed of 16MHz for calculations, and its dimensions measure 2.7in by 2.1in. Beyond meeting these requirements, the ease of use and financial cost were carefully considered. As this microcontroller was purchased for a separate project, it was essentially free to use compared to similar microcontrollers that cost more than $30. In addition, the Arduino IDE provides a quick setup and rapid prototyping which can be invaluable when working under a deadline.

4.4. Inductor Sizing

The primary energy storing device used in the Buck DC to DC converter is the inductor. The smart DC wall outlet features two individual buck converters generating the output voltages of the system, as well as a third Buck converter providing 5V as an internal supply. A practical method for determining the critical inductance for each inductor is by specifying a desired ripple current, Δi_L [3]. From Figure 2, Δi_L can be

calculated when the high side switch is conducting using Equation 2-1 for the voltage across an inductor.

$$\frac{di_L}{dt} = \frac{V_{in} - V_{out}}{L} \tag{4-1}$$

By using discrete time instead of continuous time for this calculation, the ripple current can be related to the inductance, frequency of oscillation and duty cycle of the system.

$$\frac{di_L}{dt} \approx \frac{\Delta i_L}{\Delta t_{on}} = \frac{\Delta i_L}{T*D} = \frac{\Delta i_L * f_{osc}}{D} \rightarrow \Delta i_L = \frac{V_{in} - V_{out}}{L * f_{osc}} * D \tag{4-2}$$

In Equation 4-2, T represents the period of oscillation for the switch, f_{osc} represents the frequency of oscillation for the switch, and D represents the duty cycle percentage of the switch during that period. As shown in [3], the steady-state duty cycle of a buck converter can be related to the input and output voltage.

$$D = \frac{V_{out}}{V_{in}} \tag{4-3}$$

This substitution gives us the final equation for calculating the ripple current in the inductor of a Buck converter, which can be rearranged to calculate the critical inductance:

$$\Delta i_L = \frac{V_{out} * (V_{in} - V_{out})}{V_{in} * L * f_{osc}} \rightarrow L = \frac{V_{out} * (V_{in} - V_{out})}{V_{in} * \Delta i_L * f_{osc}} \tag{4-4}$$

Using Equation 4-4, the worst-case condition for the primary Buck converter inductors is calculated when the output voltage is V_{out}=24V, or half the input voltage of V_{in}=48V. The maximum average current at this output voltage is 4.2A, as limited by the maximum power specification of 100W per channel. For this calculation, the ripple current value is taken to be 30% of the maximum output current, giving Δi_L=1.26A. Assuming a switching frequency of f_{osc}=300kHz, the critical inductance can be calculated as:

$$L = \frac{24V*(48V-24V)}{48V*1.26A*300kHz} \approx 32\mu H \quad (4\text{-}5)$$

The datasheet for the LT8619-5 Buck controller suggests a different method for calculating the inductor's critical value [15], shown in Equation 4-6.

$$L = 2 * \frac{V_{out}+V_{SW(Bot)}}{f_{osc}} \quad (4\text{-}6)$$

The critical inductance for the secondary Buck converter is found in Equation 4-7 using the bottom switch voltage drop value of $V_{SW(Bot)}$=0.264V, a frequency of oscillation of f_{osc}=1.4MHz, and an output voltage of V_{out}=5V.

$$L = 2 * \frac{5V+0.264V}{1.4MHz} \approx 7.5\mu H \quad (4\text{-}7)$$

When selecting inductors for this configuration, the inductance must be larger than the critical values calculated in Equations 4-5 and 4-7, as well as being rated to handle the maximum output current plus half of the ripple current. Two inductors with a saturation current of 11A and an inductance of 33µH were selected for the primary Buck converter, while an inductor with a saturation current of 1A and an inductance of 10µH was selected for the secondary Buck converter.

4.5. Capacitor Sizing

The Buck converter suffers from AC signal components on the input and output of the system, so capacitors are typically used to compensate and provide a more ideal DC signal. The input supply faces current pulses with very fast rise and fall times, and the input capacitor must be able to handle these large current spikes. The calculation for determining the input capacitor RMS current rating comes from [13], as shown in Equation 4-8.

$$I_{RMS} = \frac{I_{max}}{V_{in}} \sqrt{V_{out} * (V_{in} - V_{out})} \quad (4\text{-}8)$$

The RMS current is maximized when the output is half of the input, occurring when V_{out}=24V on the primary Buck converter. Using the maximum average current of I_{max}=4.2A at this operating point, the RMS current rating of the input capacitor for the primary Buck converter is I_{RMS}=2.1A. For the secondary Buck converter with a maximum average output current of I_{max}=250mA, the RMS current rating needed is I_{RMS}=75mA. Typical input capacitance values range from 5µF to 1005µF, with multiple capacitors added in parallel to meet the RMS current rating. For the primary Buck converter, two parallel 22µF capacitors with RMS current ratings of 3A were selected. For the secondary Buck converter, a 5µF capacitor with an RMS current rating of 250mA was selected.

The output capacitor of a Buck converter helps smooth the noise at the output of the inductor and stores some energy to satisfy transient loads. The voltage ripple at the output is primarily controlled by the equivalent series resistance (ESR) of this capacitor, as shown in Equation 4-9 from [13].

$$\Delta V_{out} \approx \Delta i_L * \left(ESR + \frac{1}{8 * f_{osc} * C} \right) \quad (4\text{-}9)$$

An output ripple voltage of less than 2.5% for the primary Buck converter is desired with the worst case occurring at an output value of 3V, creating an output voltage ripple of ΔV_{out} ≤75mV. The maximum ripple current occurs when the output voltage is half of the input, with Δi_L=1.26A as previously calculated. The process of finding a capacitor that satisfies this condition is iterative, but a set of three capacitors in parallel were chosen with two electrolytic capacitors featuring an ESR of 28mΩ and capacitance of 220µF,

and one ceramic capacitor rated at 10µF with a negligible ESR. Similar calculations were done for the secondary Buck converter, and a capacitor with an ESR of 30mΩ and capacitance of 22µF was selected.

4.6. Current Sense Resistor Sizing

The current being drawn by the load of each output channel of the primary Buck converter is calculated by measuring the voltage drop across a small resistor. The LTC3889 has a 75mV maximum range on the differential input, so the peak current going through the desired resistor should not generate a voltage larger than this. The desired resistance value can be calculated using Equation 4-10.

$$R_{sense} = \frac{V_{sense(max)}}{I_{max}+\frac{\Delta i_L}{2}} \tag{4-10}$$

Using $V_{sense(max)}$=75mV, a maximum output current of I_{max}=6A, and a ripple current of Δi_L=1.5A, the current sense resistors should be sized to R_{sense}=10mΩ. Two current sense resistors with a power rating of 1W and a resistance of 10mΩ were selected for use with the primary Buck converter. The LT8619-5 monitors the output current internally, so no current sense resistor is necessary for this subsystem.

4.7. Power Converter Simulation

Unfortunately, there is no SPICE model available for the LTC3889 IC due to its digital interface, so no simulations could be performed using this IC. The component values chosen can still be verified using simulation, so a simulation was constructed using a different Buck controller. Specifically, the LTC3892 [17] was chosen as it was used in Liu's iteration of the smart DC wall outlet and met similar operating

requirements. The simulation was configured as similar as possible to the LTC3889, using pulse skipping for increased efficiency and a switching frequency of 350kHz, slightly higher than the specified 300kHz. The schematic used in the simulations is shown in Figure 10.

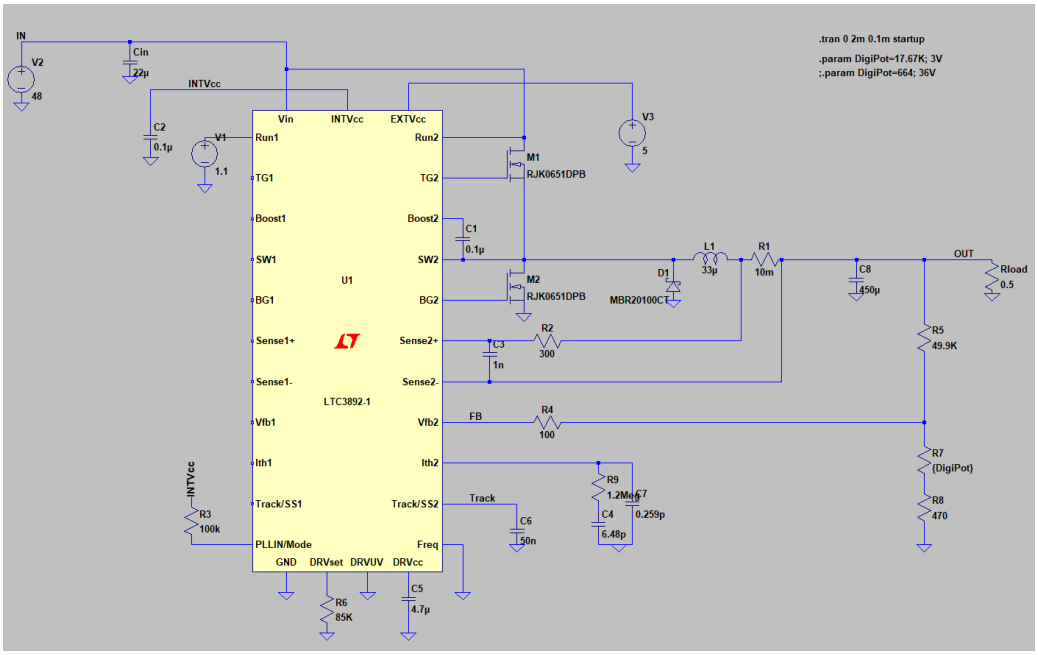

Figure 10: LTC3892 Simulation Schematic

The converter was simulated at the minimum and maximum output voltages of 3V and 36V, each at the minimum and maximum load conditions. Figure 11 and Figure 12 show the transient waveform from start-up to steady state of the 3V output at 0.25mA and 6A respectively. Figure 13 and Figure 14 show similar plots with a 36V output at 0.25mA and 2.78A respectively. All four plots show that the ripple voltage and steady state error are within 2.5% of the nominal value, confirming that these component sizes are valid. Since the output capacitance is so large it can take more than 4ms to reach high output voltages, but a fast response time is not crucial so this can be tolerated.

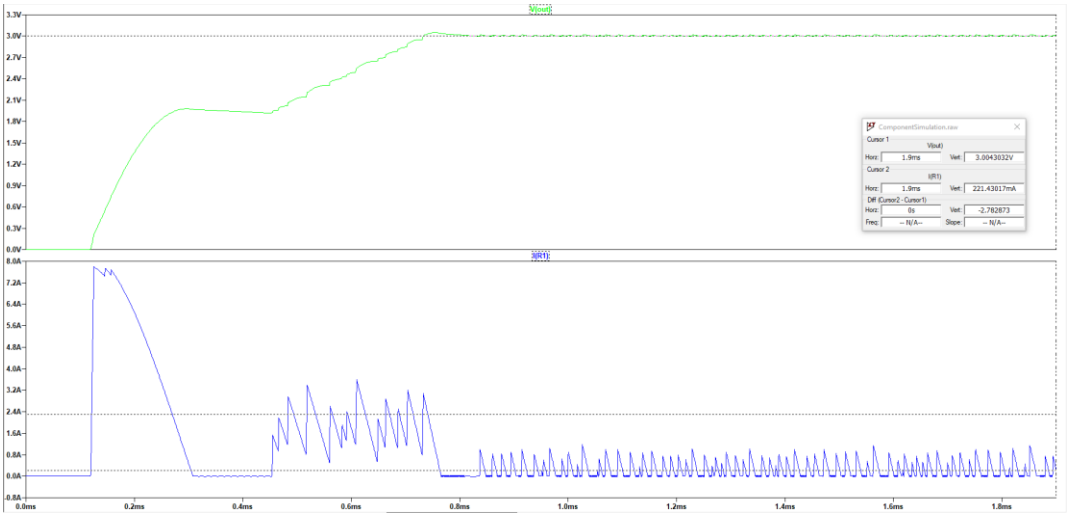

Figure 11: Simulated Output Voltage of 3V, Load Current of 250mA

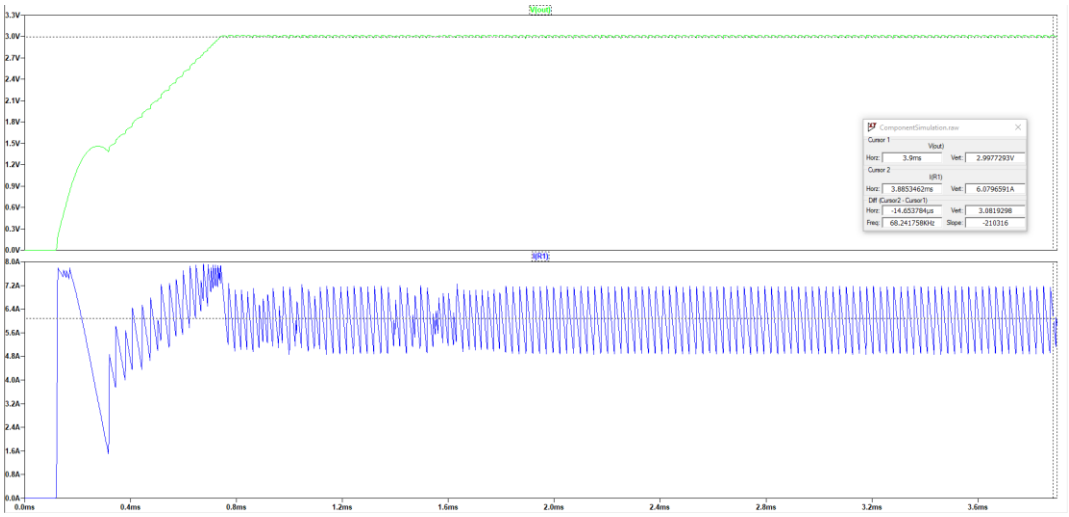

Figure 12: Simulated Output Voltage of 3V, Load Current of 6A

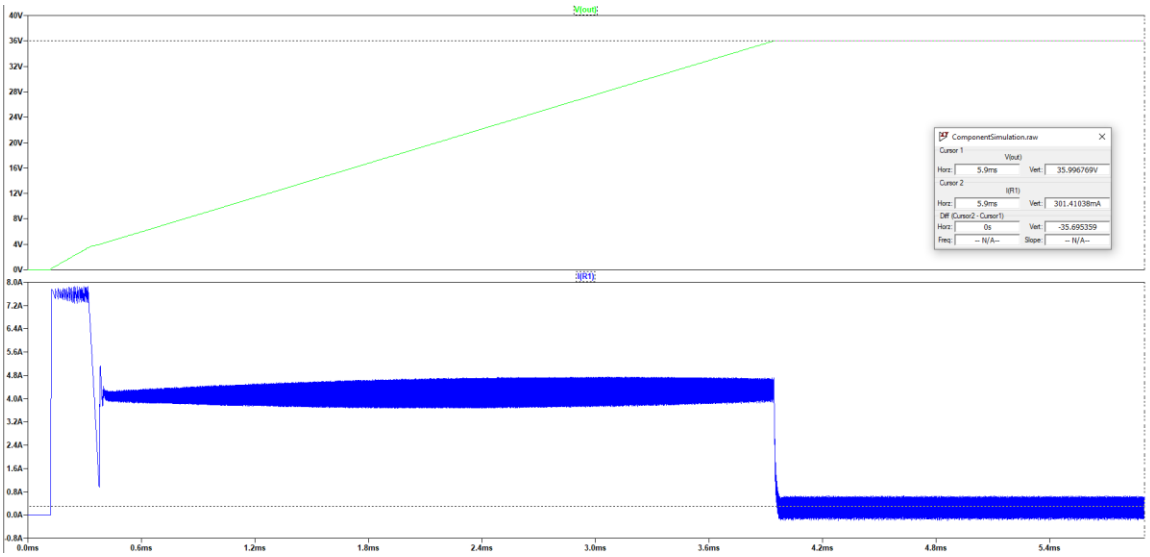

Figure 13: Simulated Output Voltage of 36V, Load Current of 250mA

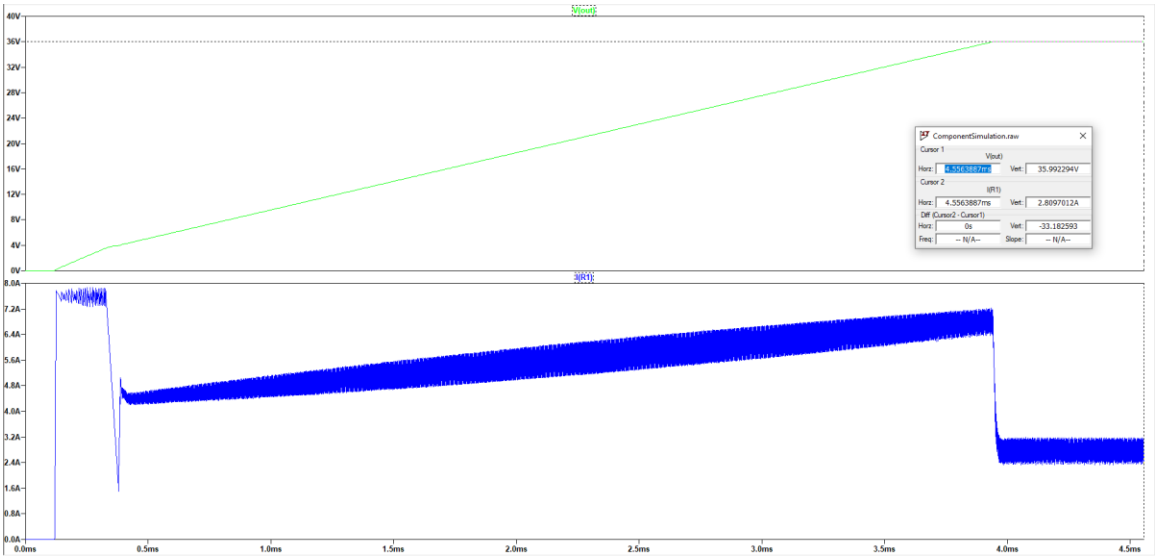

Figure 14: Simulated Output Voltage of 36V, Load Current of 2.78A

4.8. Schematic Design

The schematic for the Smart DC Wall Outlet is shown in Figure 15. The two output channels of the primary Buck converter are shown symmetrically configured on the left and right of the LTC3889 IC. The secondary Buck converter is shown supplying auxiliary power to the ICs and status LEDs. The header connections to and from the board are shown with fuses, limiting the total current into the system to 5A and the total current out of each channel to 8A. The Arduino Uno uses an internal logic level of 5V while the LTC3889 uses an internal logic level of 3.3V, so the I2C level shifters were included to allow the two devices to communicate directly. The status LEDs have been configured to turn on when the Arduino Uno applies 5V at the gate of the MOSFETs.

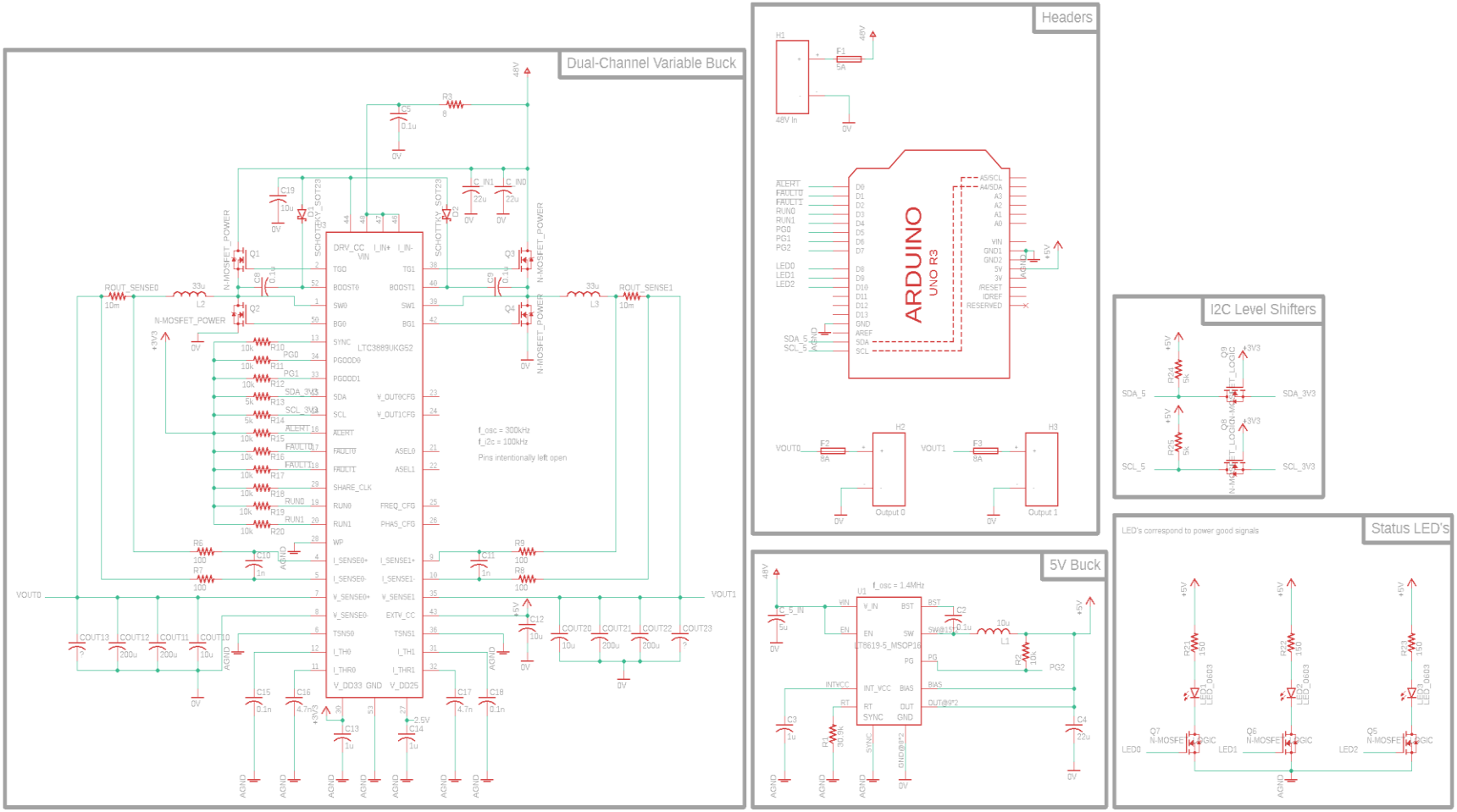

Figure 15: Smart DC Wall Outlet Schematic

4.9. Test Device Selection

In order to develop a robust algorithm for detecting a load's nominal voltage, a wide range of DC devices must be studied. Devices that run on lithium ion batteries behave differently than those that run on alkaline batteries or are powered by USB. By gathering data from devices with different power sources, input voltages and load currents, trends that are common between all of them will become apparent. The major limiting factor for selecting devices to test was the financial cost, so most of the devices selected were inexpensive or free to use. Additionally, these devices must be able to be turned on just by applying an input voltage without any user interaction, so no devices with power toggle buttons were accepted. The devices selected for testing are listed in Table 3.

Table 3: Test Device Selection

Device	Input Voltage	Load Current	Power Source
Portable TV	4.5V	450mA	Alkaline Batteries
Radio	5V	250mA	Lithium Ion Battery
Raspberry Pi PC	5V	300mA	USB Cable
Android Phone	5V	450mA	Lithium Ion Battery
Windows Phone	5V	475mA	Lithium Ion Battery
Computer Fans	12V	125mA	Power Adapter
Speakers	15V	70mA	Power Adapter

4.10. Data Analysis

The turn-on characteristics of these devices were observed by sweeping the input voltage and recording the current draw at each step. The voltage sweep begins at 1V, then increases in steps of 0.1V until at least 1V after the nominal voltage. These raw voltage sweeps can be found in Appendix A. In general, these devices tend to use very little current at 1V, then feature a large spike in current around the voltage where the device first turns on. Since these plots tend to be noisy, a moving average with a window size of 5 samples can be used on the data to provide a clearer indication of trends. Additionally, since these plots feature large jumps in the rate at which the current is increasing per voltage step near the turn on voltage, the derivative of these plots could provide some additional information. The moving average and the derivative with respect to the current over the voltage were calculated for each test device, shown in Figure 16 through Figure 22.

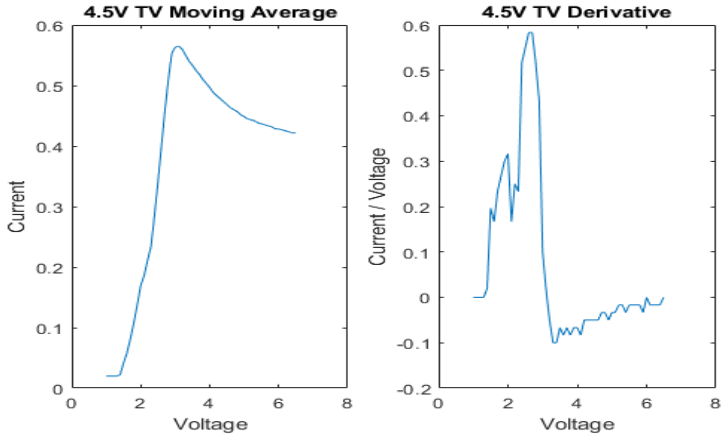

Figure 16: 4.5 TV Moving Average and Derivative

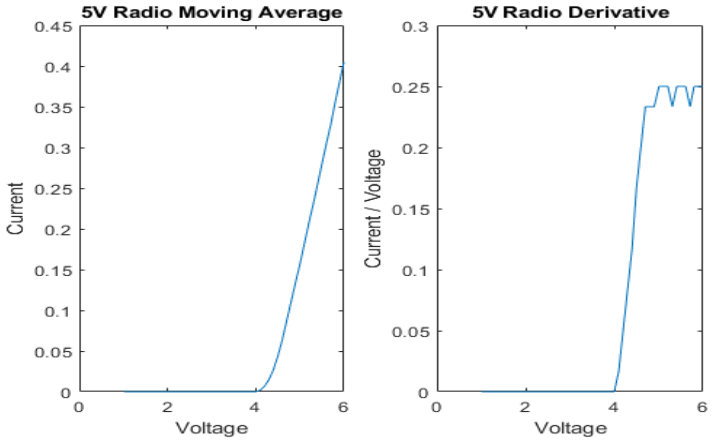

Figure 17: 5V Radio Moving Average and Derivative

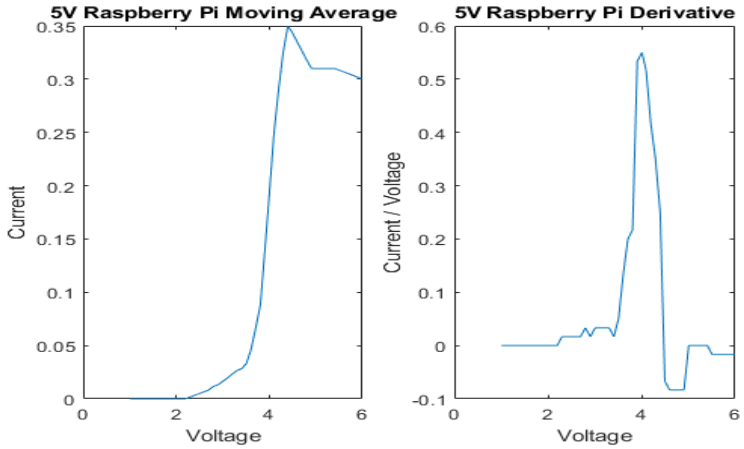

Figure 18: 5V Raspberry Pi Moving Average and Derivative

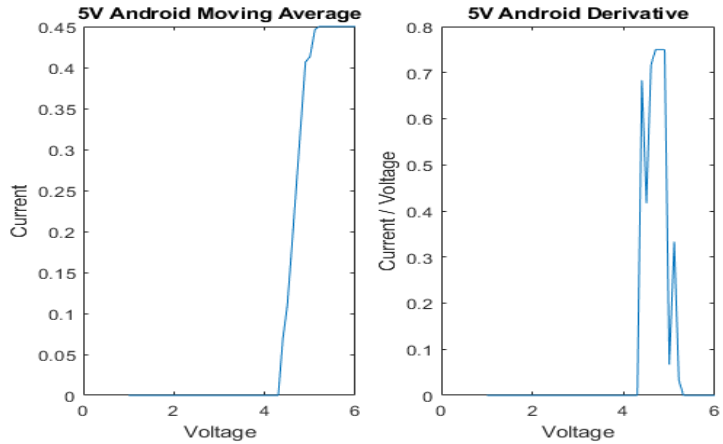

Figure 19: 5V Android Moving Average and Derivative

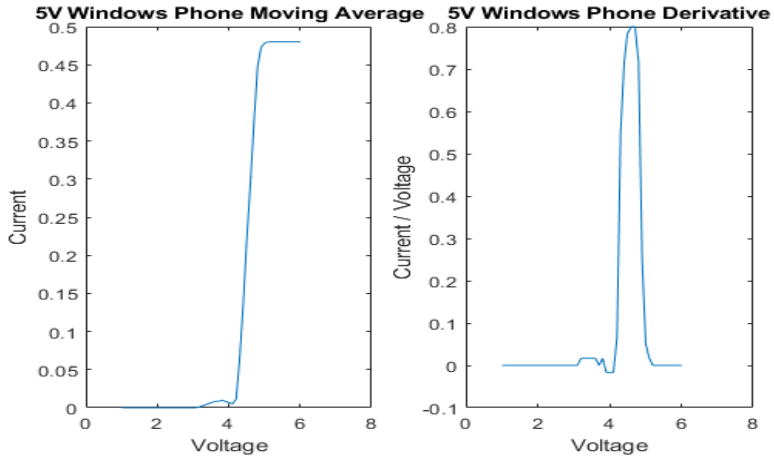

Figure 20: 5V Windows Phone Moving Average and Derivative

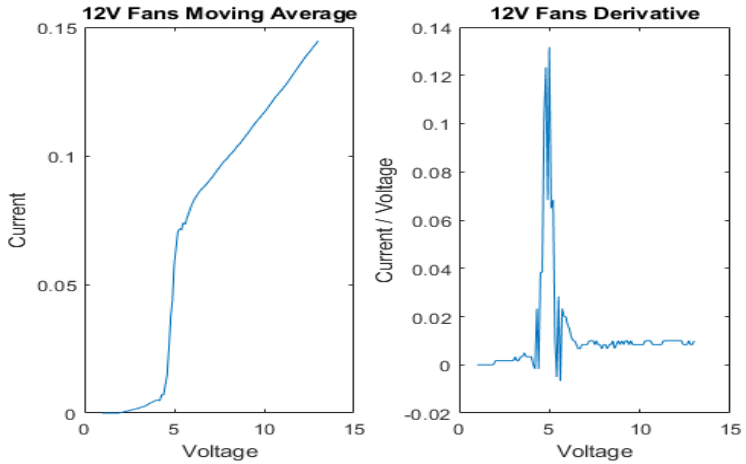

Figure 21: 12V Fans Moving Average and Derivative

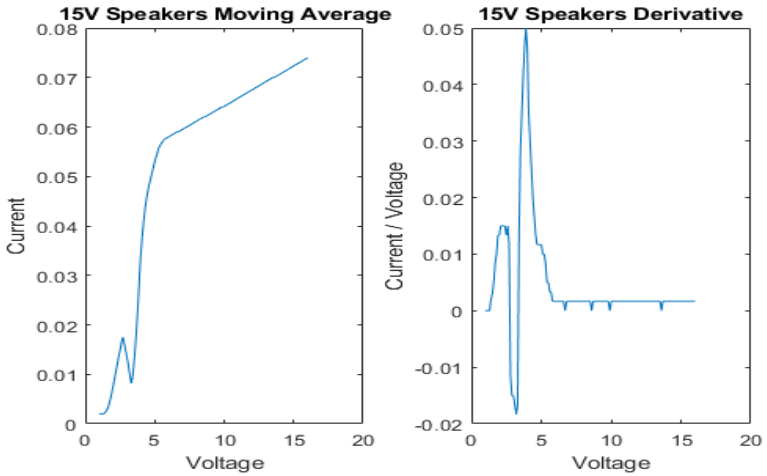

Figure 22: 15V Speakers Moving Average and Derivative

Analysis of these figures grants a few insights about the devices' nominal voltage. Primarily, the turn on voltage of every test device is directly indicated by the largest positive increase in current per voltage step. This is most clearly illustrated by Figure 20. The derivative of the current over voltage reaches its peak value at 4.7V, which is close to the nominal voltage of 5V. For most of the devices tested, this maximum derivative value is found just before the nominal voltage. The three devices that do not conform exactly to this pattern are the 5V Radio, the 12V Fans and the 15V Speakers.

The 5V Radio does feature a large step in the derivative value of the current over voltage, but this value is held constant unlike the other devices. This is because the 5V Radio does not draw a large jump in current at the turn on voltage, but instead begins ramping up at a constant slope as seen in Figure 17.This is likely due to the simplistic lithium ion battery charging circuit within the 5V Radio. The voltage at which the device begins ramping up is close to the listed nominal voltage of the device, so this starting point can be used to trivially determine the nominal voltage.

The 12V Fans and the 15V Speakers both present a maximum derivative between 4V and 5V, but nominally require a voltage three times larger. The plots relating to the 12V Fans and 15V Speakers are shown in Figure 21 and Figure 22. Both devices function if they are provided a 5V input voltage but may not perform optimally. The fans do not spin as fast as they normally would, and the speakers produce a much quieter sound. One characteristic differentiates the 12V Fans and the 15V Speakers from the other devices: the current per voltage step steadily increases after the tun on voltage. With the other devices being tested (except for the previously handled 5V Radio), the current draw after

the turn on voltage either stagnates or begins to decrease. Unfortunately, neither of these devices exhibit any indication of what their nominal voltage might be. It is possible to infer that the nominal voltage is much higher than the turn on voltage for these devices, but just how much higher is left to speculation.

4.11. Algorithm Development

Using the turn on voltage characteristics previously identified, an algorithm can be developed to estimate the nominal voltage of a load device. A flowchart outlining the algorithm developed in MATLAB is shown in Figure 23. The process begins with sweeping the input voltage from 1V to 3V, as nearly all DC devices can be assumed to handle this voltage range. The algorithm attempts to find a large spike in the current draw in this range to find the turn on voltage. If no spike is detected, the input voltage is increased in 1V increments until one is detected. Once a peak in the current draw is detected, the peak needs to be verified. With some devices, like the 15V speakers seen in Figure 22, there is a small initial jump in current usage followed by a much larger increase at the true turn on voltage. For all the devices tested, these false turn on voltage indicators appear within 1V of the actual turn on voltage. By increasing the input voltage 1V beyond the detected peak value, the current draw peak can either be confirmed as the true turn on voltage, or a new larger peak can be detected indicating that turn on voltage is likely higher.

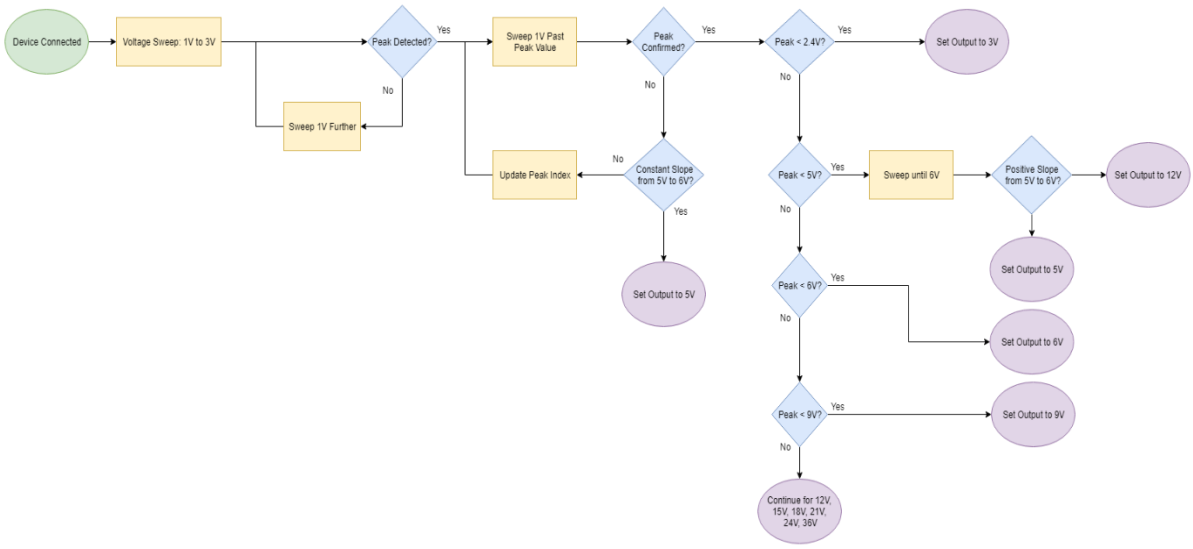

Figure 23: Nominal Voltage Detection Algorithm Flowchart

While attempting to confirm the peak in the current draw rate, the algorithm continues monitoring the maximum voltage being sent to the load device. Once the output voltage surpasses 6V, the algorithm looks to see if the device could be a battery charging device like the 5V radio. If the current draw rate remains constant from 5V to 6V and the peak has yet to be confirmed, a battery charging device has been detected and the output is set to 5V. Otherwise, if the current draw rate is variable over this range, the device is likely not a battery charging device and the algorithm continues trying to confirm the peak current draw rate.

Once the peak is confirmed, the algorithm attempts to set the output voltage. If the peak value detected is below 2.4V, the output is set to 3V. Most 3V devices are powered by alkaline batteries, which decrease their output voltage as they lose charge. These battery voltages can drop to less than 2V total, which explains why a 3V device would have a lower turn on voltage. Continuing, if the peak value detected is below 5V, the

algorithm looks to see if the device might need more power. As found in the previous section, larger nominal voltage devices draw an increasing amount of current as the input voltage goes up. This is detected by finding the median current draw rate of the device from 5V to 6V. If the median value is positive, it indicates that overall the device is drawing more current after its turn on voltage, implying a need for a larger input voltage. If the median value is close to 0 or negative, the device does not require a larger input voltage and the nominal voltage should be set close to the turn on voltage. The current draw rate was specifically investigated from 5V to 6V as most of the devices tested required 5V or less. This was primarily done for safety, as all the devices tested would be able to handle being powered at 6V. Ideally, the current draw rate would be investigated 2 volts after the peak turn on voltage, not just from the 5V to 6V range. More testing would need to be done to see if most devices could handle an input voltage two volts higher than nominal.

If the device under test was identified as needing a higher input voltage, the voltage was set to 12V. As mentioned previously, there is currently no indicator for exactly what voltage a device requires, so the output is estimated at being 12V. In this case the two higher voltage devices being tested were able to be safely be powered by 12V, but further studies of DC devices would need to be conducted to identify a more general metric for finding the nominal voltage of a high input voltage device. If the device did not continue to draw an increasing amount of current after the turn on voltage, the device was set to the closest output voltage of 5V. For all devices with a larger detected turn on voltage, the output voltage was set to the next closest output voltage. These output voltages include 6V, 9V, 12V, 15V, 18V, 21V, 24V and 36V. More output

voltages can be added in the future, but these appeared to be the most common levels used commercially.

This algorithm was implemented in MATLAB and was successfully able to identify nominal voltages for the test devices. This algorithm will next be implemented on the microcontroller to be tested on the devices in real time.

5. HARDWARE CONSTRUCTION AND TESTING RESULTS

5.1. Bill of Materials

The first step towards construction of the smart DC wall plug involved selecting physical parts for each of the components shown in the circuit schematic from Figure 15. All the components used in the smart DC wall outlet were purchased directly from Digi-Key, except for the two Buck controllers which were obtained for free from Analog Devices. The main limiting factors for selecting components were size and cost. The total size of the board was limited by the physical dimensions listed in Table 2, requiring components to be small enough to fit on the board. Since the construction of the board was done by hand, the minimum package size for the resistors and capacitors was set to 16mm by 8mm to allow for easier manipulation. Other components such as the power MOSFETs and inductors needed to be larger to handle the large amounts of current passing through the system. With regards to cost, the cheapest possible components were chosen provided they satisfied the maximum current and voltage ratings and were reasonably sized. The finalized list of components chosen is shown in Figure 24.

Designators	Qty Needed	Safety Qty	Value	Description	Digikey PN	Footprint	Link	Price/Unit	Total Price
C_5_IN	1	1	5u	5V Buck Input Cap, I_RMS_MAX = 0.1A	493-13293-1-ND	Radial Through Hole 1		$0.68	$0.68
C10, C11	2	2	1n	Current Sense Low Pass Caps. Ceramic	1276-1195-1-ND	0603 (1608 Metric)		$0.10	$0.20
C12, C19, COUT10, COUT20	4	4	10u	Output Caps, DRVcc Cap, EXTVcc Cap. Vmax = 20V. Ceramic	490-12457-1-ND	1206 (3216 Metric)		$0.55	$2.20
C15, C18	2	2	0.1n	I_TH Caps, used for control loop compensation. Vmax = 3.5, Ceramic	399-8980-1-ND	0603 (1608 Metric)		$0.18	$0.36
C16, C17	2	2	4.7n	I_THR Caps, Vmax = 3.5, Ceramic	490-9669-1-ND	0603 (1608 Metric)		$0.18	$0.36
C2, C5, C8, C9	4	4	0.1u	Decoupling Caps. Vmax = 48V, Ceramic	311-1523-1-ND	0603 (1608 Metric)		$0.22	$0.88
C3, C13, C14	3	3	1u	Decoupling Caps. Vmax = 3.3V. Ceramic	1276-1036-1-ND	0603 (1608 Metric)		$0.10	$0.30
C4, C_IN1, C_IN2	3	3	22u	Input Cap and 5V Output Cap. Vmax = 48V. Electrolytic. Need 2 for input	P122450-ND	Radial Through Hole 1		$2.82	$8.46
COUT11, COUT12, COUT21, COUT22	4	4	200u	Output Caps. Vmax = 40V, Electrolytic	1572-1396-ND	Radial Through Hole 1		$2.88	$11.52
COUT13, COUT23	2	2	?	Extra space if another cap is needed	-	-	-	$0.00	$0.00
D1, D2	2	2	-	Schottky diode from DRVcc to Boost	497-2529-1-ND	SOT-23-3		$0.45	$0.90
F1	1	2	5A	Fuse, 5A	507-1194-1-ND	1206 (3216 Metric)		$0.35	$0.70
F2, F3	2	4	8A	Fuse, 8A	507-1056-1-ND	2-SMD (Custom?)		$0.41	$1.64
H1, H2, H3	6	6	-	Input, Output Headers. Black and red banana plugs	BKCT3149-0-ND, BKCT	Axial Through Hole		$1.20	$7.20
L1	1	1	10u	5V Buck Inductor. Max Current of 1A? Maybe less.	732-1025-1-ND	1919 (4848 Metric)		$1.45	$1.45
L2, L3	2	2	33u	Buck Inductors. Max Current of 10A?	732-2524-1-ND	Nonstandard		$7.06	$14.12
LED1, LED2, LED3	3	3	-	3 Green LED's	160-1446-1-ND	0603 (1608 Metric)		$0.29	$0.87
Q1, Q2, Q3, Q4	4	4	-	Power Mosfets to handle main switching	SQJA68EP-T1_GE3CT-I	PowerPAK® SO-8		$0.90	$3.60
Q5, Q6, Q7, Q8, Q9	5	5	-	Used for LED's and I2C Level Shifting. Has body diode I=0.2A	SI1050X-T1-GE3DKR-N	SC-89-6		$0.59	$2.95
R1	1	1	30.9k	Used to set 5V Buck freq. (1%)	541-30.9KHDKR-ND	0603 (1608 Metric)		$0.10	$0.10
R13, R14, R24, R25	4	4	5k	Pull up resistors for the I2C lines.	YAG5090CT-ND	1206 (3216 Metric)		$0.56	$2.24
R2, R10, R11, R12, R15, R16, R17, R18, R19,	10	10	10k	Pull Up resistors for buck.	RHM10KADCT-ND	0603 (1608 Metric)		$0.13	$1.26
R21, R22, R23	3	10	150	Current Limiters for LED's	A129678CT-ND	0603 (1608 Metric)		$0.03	$0.33
R3	1	1	8	Used to filter Vin pin (1%)	P8.2AJCT-ND	0603 (1608 Metric)		$0.19	$0.19
R6, R7, R8, R9	4	10	100	Isense Feedback RC network.	RMCF0603FT100RCT-I	0603 (1608 Metric)		$0.02	$0.18
ROUT_SENSE0, ROUT_SENSE1	2	2	10m	Output Sesning Resistors, 10mR	311-.010TCT-ND	2512 (6432 Metric)		$0.66	$1.32
U1	1	1	LT8619-5	Secondary buck controller	LT8619EMSE-5#PBF-N	16-MSOP-EP		$6.00	$6.00
U2	1	2	ARDUINOR3	Arduino Uno Rev. 3, Breakaway Headers	S1012EC-40-ND	0.1" Pitch		$0.51	$1.02
U3	1	1	LTC3889	Primary buck controller	LTC3889IUKG#PBF-ND	52-QFN (7x8)		$16.69	$16.69
								Total Cost:	$87.72

Figure 24: Bill of Materials

5.2. PCB Layout and Fabrication

Once the components were selected, the footprints for each part had to be generated. Most of the resistors and capacitors used the standardized 0603 package size, which was readily available in the EAGLE footprint library. Most of the other components required custom footprints which needed to be made from scratch. Dimensions and tolerances for each of these parts were obtained from their respective datasheets. After generating footprints for every component, the parts were laid out and connected as shown in Figure 25 and Figure 26. The primary concerns for the board layout were to minimize the length of the power path and to isolate the small signal components from the power path. The power paths on the board are clearly illustrated in in Figure 25 as the wider traces on the left half. These traces were made to be 2.54mm, allowing a maximum temperature increase of 20°C as calculated from [18]. The power path connecting the input header to the output headers includes the switching MOSFETs, inductors, input and output capacitors, and the current sense resistors.

The remainder of the components were routed away from the noisy, high power components. The Arduino Uno header was placed on the right half of the board, oriented to allow the microcontroller to plug in from the bottom. Using the natural isolation that the Arduino Uno provided, the LTC3889 and its supporting components were placed in between the header pins. Not shown in Figure 25 and Figure 26 are two internal layers, consisting of ground planes that spanned the width of the board and a signal layer used to connect components to the 5V Buck regulator seen in the top right. Separate signal and power ground planes were used to improve isolation between sections of the board.

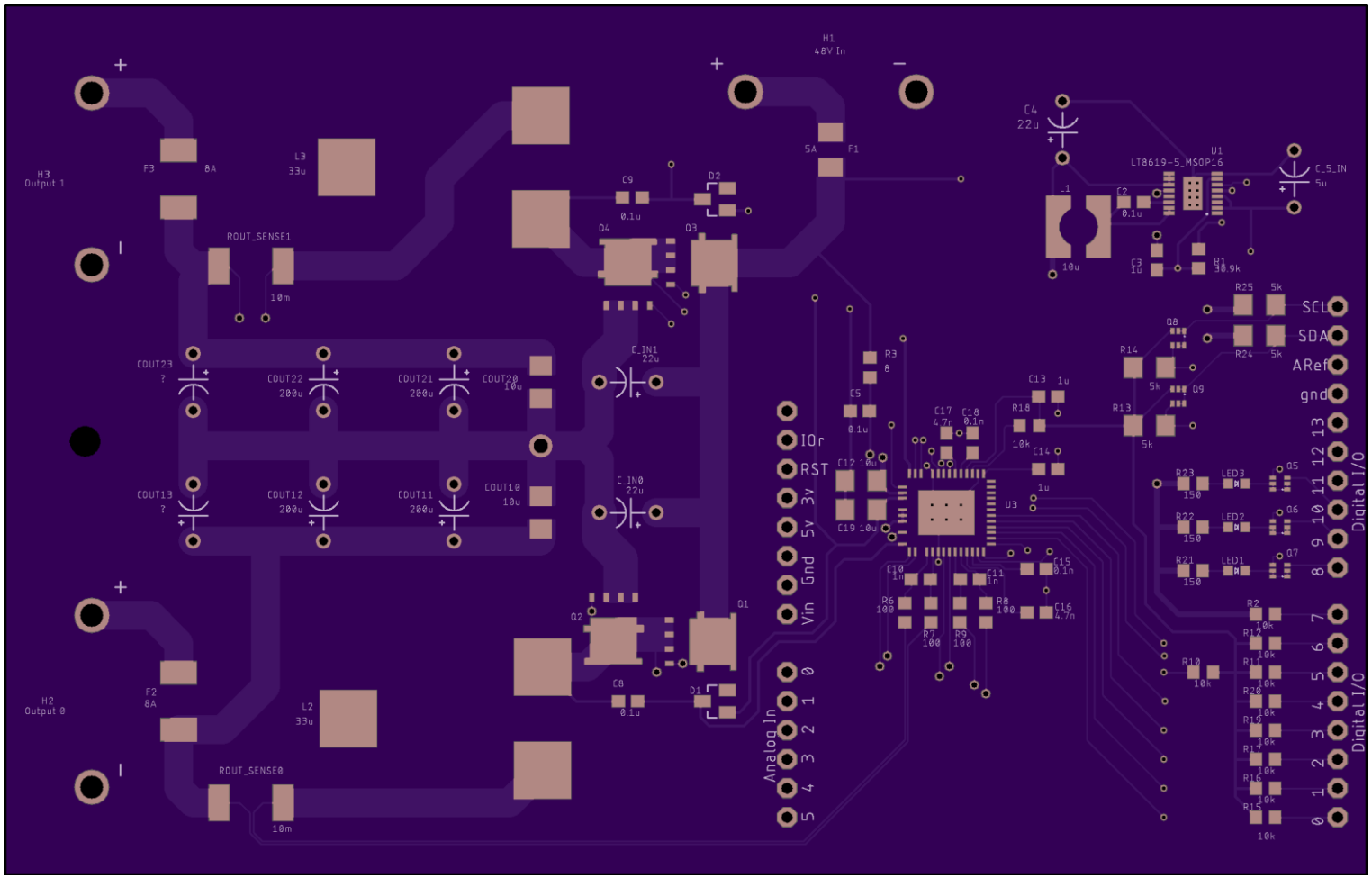

Figure 25: PCB Layout - Top

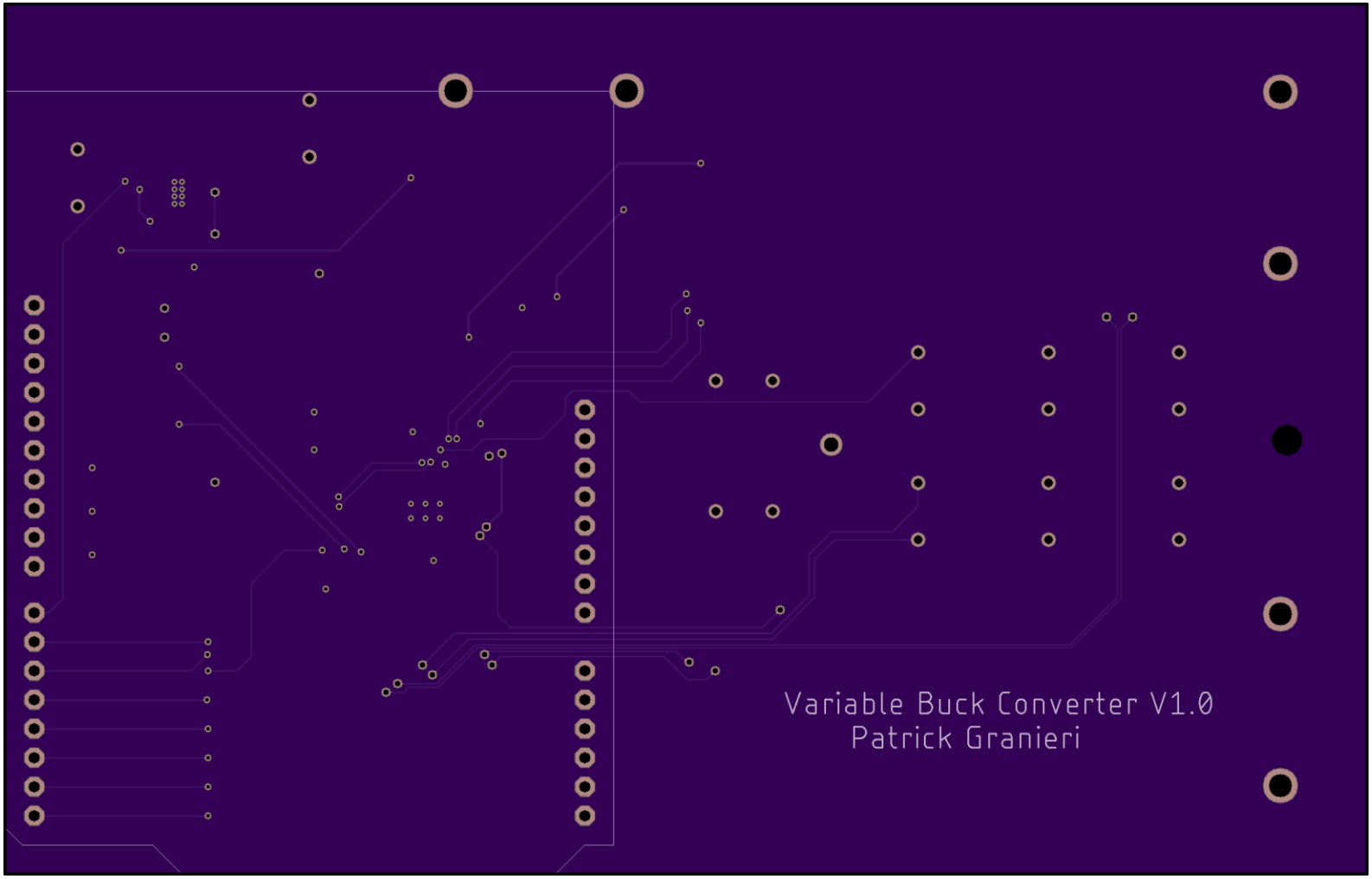

Figure 26: PCB Layout - Bottom

After the PCB layout process was completed, the board was ready for fabrication. Oshpark was selected to print the board, as they offered a reasonable combination of cost and time-to-ship for a four-layer board.

5.3. Board Assembly

Upon receipt of the board, it was inspected for any possible defects. Before adding any components to the board, the surface mount component pads were lightly tinned with solder to prepare for use with the heat gun. Construction began by soldering the LTC3889 IC to the board, since this part controlled the rest of the board and needed to work before anything else could. Followed by soldering on the microcontroller pin headers, the I2C level shifters and a few configuration resistors, the Buck controller was connected to the Arduino Uno. The Uno was able to talk to and receive information from the LTC3889 using an I2C connection, so construction continued with the power path.

Next the input and output connectors were soldered on, followed by the current limiting fuses. Only one channel of the power path was soldered to start with, consisting of the power MOSFETs, the inductor, input and output capacitors, and the current sensing resistor. Additionally, the capacitors were added to the boost and gate drive pins of the LTC3889, along with the Schottky diode to assist in biasing the top N-channel MOSFET. After the assembly of one channel was complete, the Arduino Uno commanded the primary Buck controller to output a series of voltages on the output. Once these were successfully observed, the second power channel was soldered on in a similar fashion.

Finally, the auxiliary systems were attached, consisting of the 5V buck converter, the microcontroller pull-up resistors, and the status LEDs. The Arduino Uno was able to be powered by the board alone after these additions. After testing that the Uno could read information from the input pins and signal the status LEDs, the board was officially complete. The full assembly can be seen in Figure 27.

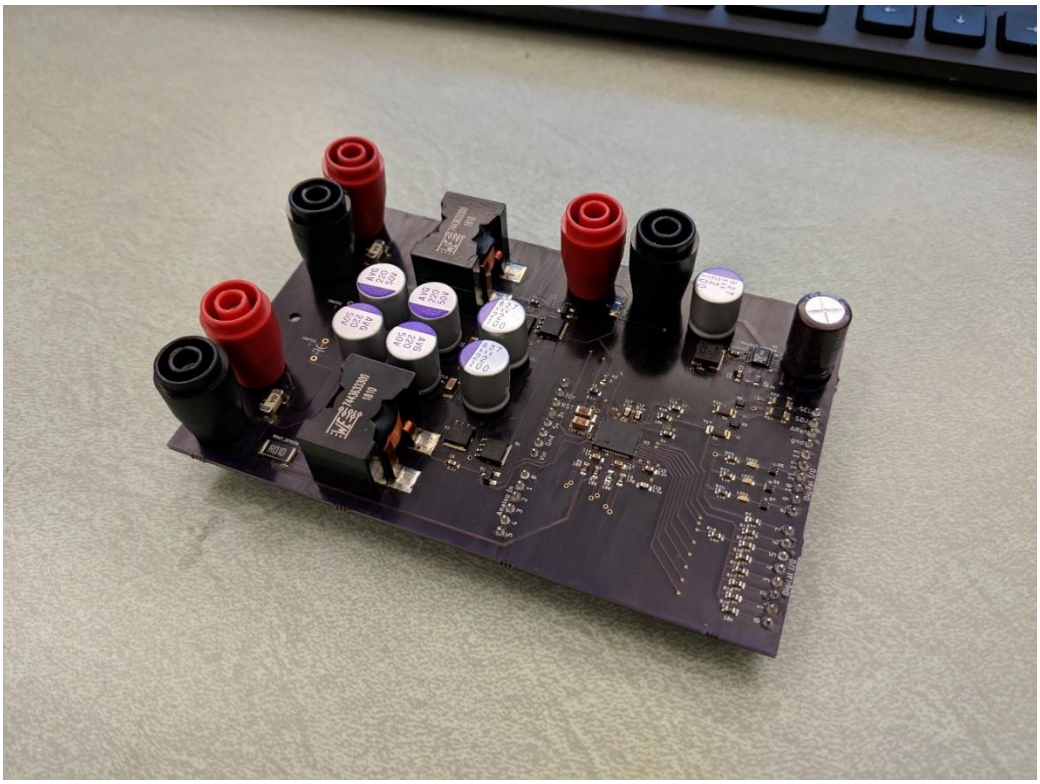

Figure 27: Fully Assembled Smart DC Wall Outlet

5.4. Microcontroller Programming

The algorithm developed in Section 4.11 was ported directly from MATLAB to C for use with the Arduino Uno. Though few changes were made to the algorithm, the Arduino Uno needed many new functions to interface with the LTC3889 properly. The

entirety of the code used on the Arduino Uno can be found in Appendix C. The most basic additions include functions such as `PMBUS_WRITE_WORD()` or `PMBUS_READ_BYTE()`, which use the PMBUS interface to send data to and from the LTC3889 IC. Higher level functions include `change_vout()` and `set_current_limit()`, which use the PMBUS functions to send the many commands needed to safely set a new output voltage or current limit. Other functions that were present in MATLAB but not on the Arduino platform had to be reproduced, such as `moving_avg()` and the ability to find the median value of an array.

One significant change came in the form of a finite state machine in `main()`. The three states used enumerate the different statuses of the load. State '0' handles the disconnected case, and the program waits here for 10 seconds before moving to state '1'. The program searches for the nominal voltage of the load device in state '1', as outlined previously. If no device is detected, the program is sent back to state '0'. Otherwise, if a device is found and the nominal voltage is determined, the output voltage is updated accordingly and sent to state '2'. State '2' maintains the current output voltage and monitors the current being used by the load. Once the device is disconnected and the output current goes to zero, the program is sent back to the disconnected state '0'. This finite state machine is more clearly displayed in Figure 28.

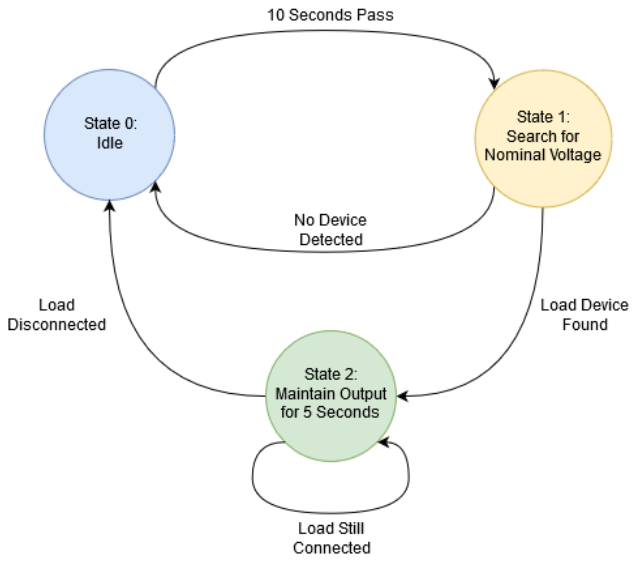

Figure 28: Arduino Uno Finite State Machine

While implementing the algorithm on the Arduino Uno, it was discovered that there was not enough RAM available on the device. The algorithm uses large arrays of floating-point numbers to gather data about the device, which use more than 2KB of space in RAM alone. The Arduino Uno was replaced with an Arduino Mega, as it contains 8KB of ram.

5.5. Board Testing Results

Before implementing the nominal voltage detection algorithm, the system was tested supplying power to the test devices. Unfortunately, while testing the smart DC wall outlet with the standard 48V input used by the DC House, a small pop was heard and the board stopped working. After thorough analysis of the board, it was determined that one of the diodes used in boosting the switching node voltage had failed. Likely due to an overcurrent condition, a diode with a higher current rating was obtained and installed on

the board. Once the other failed components on the board were replaced, the system was tested again but starting at a 24V DC supply instead of 48V.

Using this lower supply voltage, the system was recorded successfully powering devices and detecting nominal voltages. Following the collection of the necessary figures and data points, testing continued with the 48V supply. It showed promising results at first, but after a few load devices had been connected a small pop was heard and the board failed once more. It is unclear why the board failed but it is likely related to the switching MOSFET gate drive subsystem where a short was located. This issue would have been further investigated if more time and funding were available, but thankfully the system was recorded operating from the lower supply voltage. All the following metrics and figures were collected using this 24V supply.

First the output characteristics of the smart DC wall plug were verified against the electrical design specifications from Table 1. As previously mentioned, the input supply voltage used was 24V instead of 48V, but this was still high enough to supply power to all the test devices. The efficiency, output voltage ripple and error were found for each test device as shown in Appendix B. Figure 29 is copied below from Appendix B for convenience. The output of the Buck converter suffers from 400mV peak to peak high frequency noise at the switching frequency of 300kHz, likely due to resonance between the inductors and capacitors in the power path. To reduce this effect in the future, a capacitor with low ESR at the noise frequency could be added in parallel to the other output capacitors [19]. Ignoring this high frequency noise, the largest output voltage ripple stays below 25mV peak to peak, meeting the specification for every device tested. The output voltage ripple evaluations are shown in Table 6.

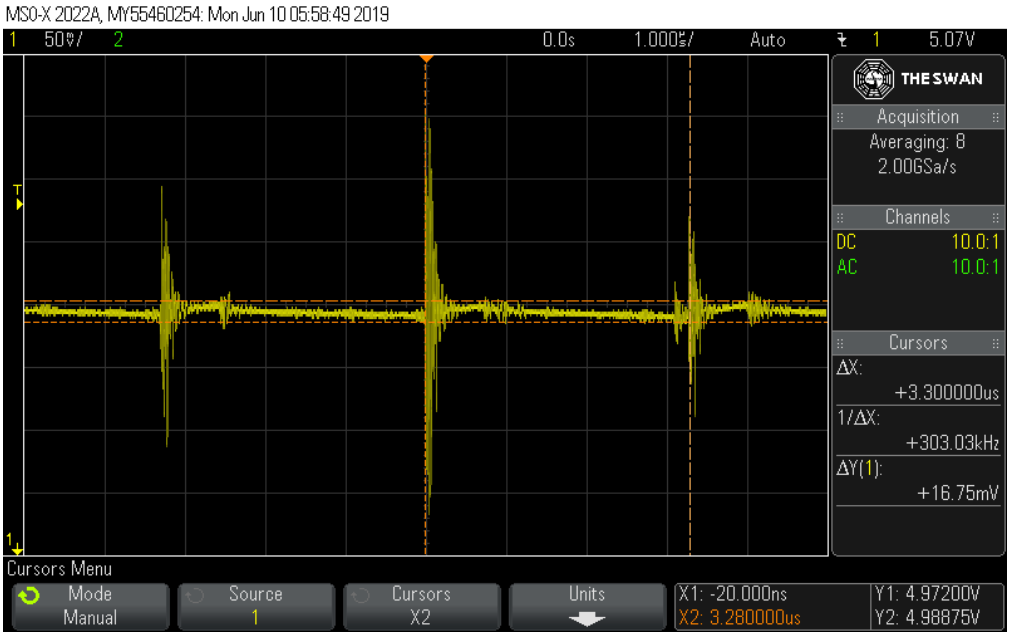

Figure 29: 5V TV Output Regulation

The efficiency and steady state voltage error were measured using the input power supply, the oscilloscope and the current sensing resistor of primary buck converter in tandem. The results of these measurements are listed in Table 4 and Table 5. Overall the steady state voltage error is within limits, but the system efficiency is below the desired 85%. Without any load connected, the smart DC wall outlet consumes 20mA from the 24V supply, using almost half a watt just in standby. For load devices with low power requirements, this extra half watt of power usage is significant in the efficiency calculations. Figure 30 from the LTC3889 datasheet demonstrates how the efficiency of the system significantly increases as the load current increases. Due to the wide output current range, it is hard to find a buck converter that is efficient at low current conditions while also being able to handle large currents.

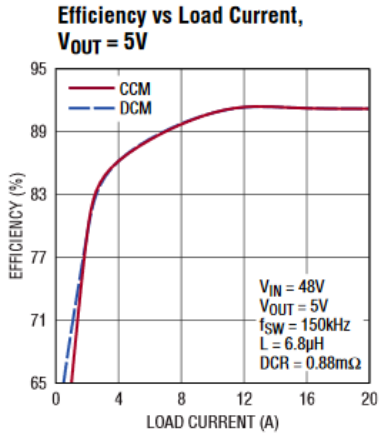

Figure 30: LTC3889 Efficiency Plot [13]

Table 4: Efficiency Evaluation

Device	Input Voltage	Input Current	Output Voltage	Output Current	Efficiency	Target Efficiency
4.5V TV	24V	0.12A	4.98V	0.44A	76.1%	≥85%
5V Android Phone	24V	0.12A	4.98V	0.45A	77.8%	≥85%
5V Windows Phone	24V	0.13A	4.98V	0.48A	76.6%	≥85%
5V Raspberry Pi	24V	0.09A	4.98V	0.35A	80.7%	≥85%
5V Radio	24V	0.07A	4.98V	0.23A	68.2%	≥85%
12V Fans	24V	0.10A	12.01V	0.15A	75.1%	≥85%
15V Speakers	24V	0.07A	15.00V	0.07A	62.5%	≥85%

Table 5: Output Voltage Percent Error Evaluation

Device	Output Voltage	Target Voltage	Percent Error	Target Error
4.5V TV	4.98V	5V	0.4%	≤2.5%
5V Android Phone	4.98V	5V	0.4%	≤2.5%
5V Windows Phone	4.98V	5V	0.4%	≤2.5%
5V Raspberry Pi	4.98V	5V	0.4%	≤2.5%
5V Radio	4.98V	5V	0.4%	≤2.5%
12V Fans	12.01V	12V	0.1%	≤2.5%
15V Speakers	15.00V	15V	0%	≤2.5%

Table 6: Output Voltage Ripple Evaluation

Device	Output Voltage	Output Voltage Ripple	Output Voltage Ripple Percent	Target Ripple Percent
4.5V TV	4.98V	16.75mV	0.3%	≤2.5%
5V Android Phone	4.98V	10.50mV	0.2%	≤2.5%
5V Windows Phone	4.98V	10.00mV	0.2%	≤2.5%
5V Raspberry Pi	4.98V	14.50mV	0.3%	≤2.5%
5V Radio	4.98V	9.25mV	0.2%	≤2.5%
12V Fans	12.01V	21.75mV	0.2%	≤2.5%
15V Speakers	15.00V	15.00mV	0.1%	≤2.5%

5.6. Algorithm Testing Results

The nominal voltage detection algorithm was first tested with the 5V devices. The algorithm successfully determined the nominal output voltage of the 4.5V TV, the 5V Android Phone, the 5V Windows Phone and the 5V Raspberry Pi. A plot depicting the algorithm successfully finding the nominal voltage during this process is shown in Figure 31.

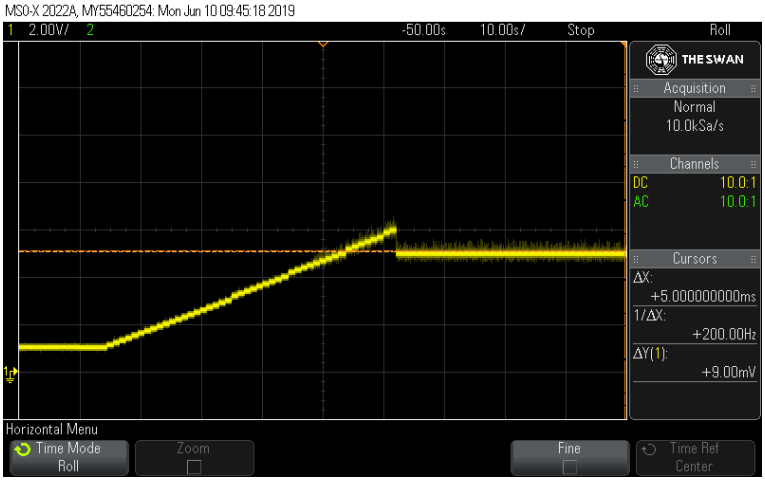

Figure 31: 5V Android Algorithm Success

The algorithm faced some issues when trying to discern the nominal voltage of the 5V Radio. Figure 32 shows the algorithm sweeping the output voltage to just above 6V, then setting the output voltage to 6V rather than the desired 5V. This indicates that a current peak was confirmed after 5V, rather than identifying that the current draw rate remained constant after the turn on voltage. The experimental load current draw was recorded from the smart DC wall outlet in Figure 33. This plot clearly shows that the current draw did not remain constant as previously assumed and instead featured dips around the 5V and 6V mark. These dips likely caused the algorithm to mistakenly confirm the peak current draw rates.

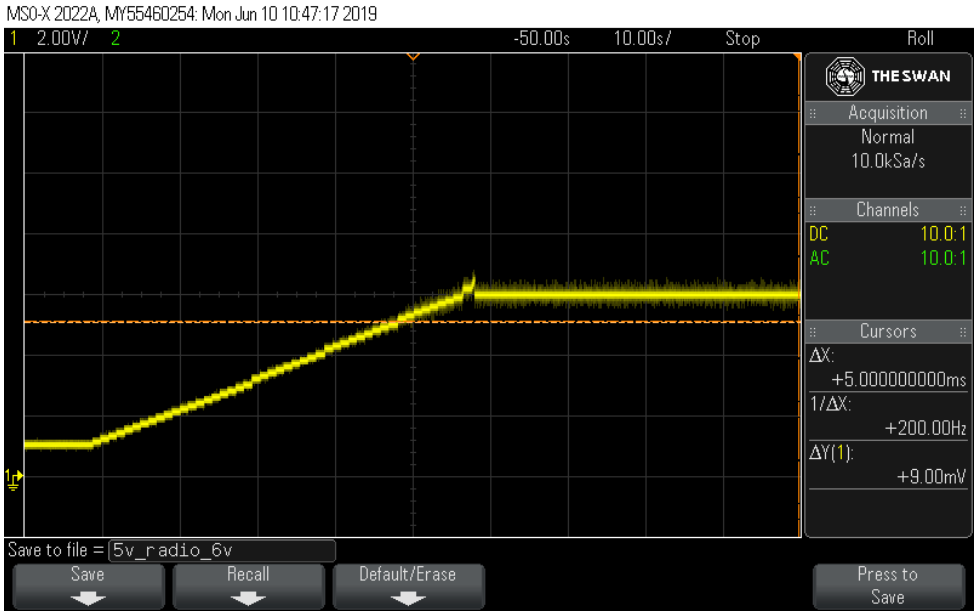

Figure 32: 5V Radio Algorithm Failure

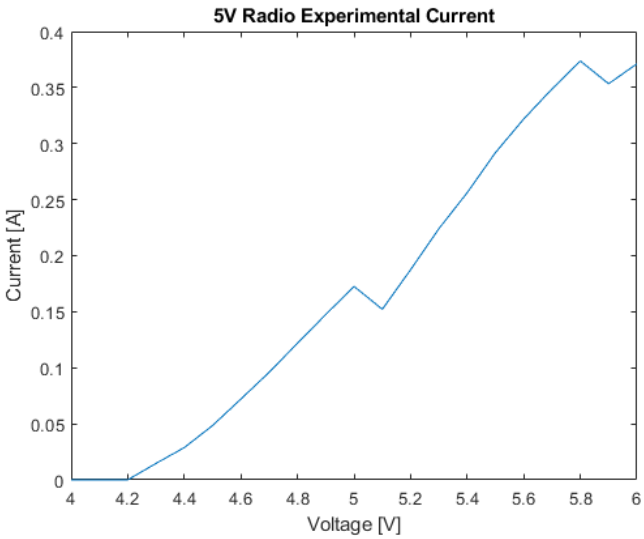

Figure 33: 5V Radio Experimental Current Draw

Testing continued with the 12V Fans and the 15V Speakers. The algorithm is shown correctly determining the output voltage of the 12V Fans in Figure 34. The 15V speakers were mistakenly classified as a 5V device, as seen in Figure 35. After analyzing the system's current readings from the 15V Speaker, it was discovered that the algorithm correctly identified the turn on voltage of the device as being below 5V but failed to identify the slope after the turn on voltage as being positive. Since the 15V Speakers only draw 70mA, the current draw rate calculated for the critical region was positive but failed to pass the minimum threshold. The algorithm could be further tuned to use a smaller threshold for this calculation, but this would make the system more susceptible to noise.

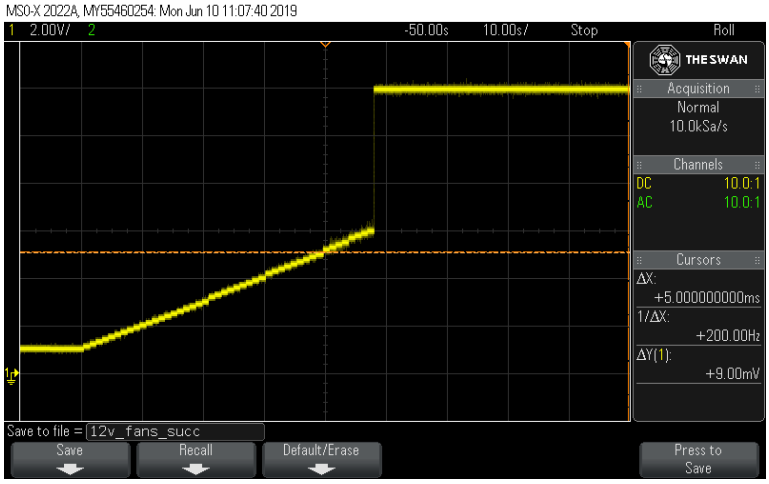

Figure 34: 12V Fans Algorithm Success

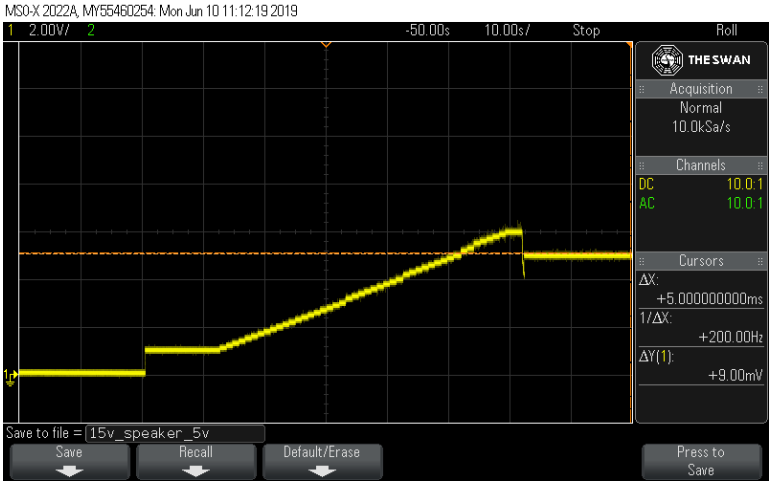

Figure 35: 15V Speakers Algorithm Failure

In total, the nominal voltage detection algorithm was successfully able to classify five out of the seven test devices. With further tuning this could likely be improved, but it is hard to know if the algorithm would be able to generalize to other devices that have not yet been tested. The turn on characteristics of DC devices vary significantly between themselves, and it is difficult to design a process that handles every case.

6. CONCLUSION

The Smart DC Wall Outlet presented in thesis extracts power from the DC House power supply and automatically converts it to an appropriate output DC voltage. This project eliminates energy losses created by converting between AC and DC power systems by supplying DC power directly to DC devices. The Smart DC Wall Outlet features two output channels capable of providing 100W of power each, with a theoretical output range from 1V to 40V. The power conversion efficiency of the system features a minimum of 70% for low load currents around 250mA and increases proportionally as the output current rises. Both the output voltage ripple and percent error are less than 0.4% of the target value, surpassing the 2.5% design specification.

The two primary improvements for this iteration of the Smart DC Wall Outlet are the digital interface and the improved voltage finding algorithm. Previous designs for this project utilized potentiometers and current sources to set the output voltage of the DC to DC converter, leading to issues with accuracy and general system complexity. This system uses an improved Buck controller, the LTC3889, to digitally monitor and manage the output voltage of the system. Using this chip, a microcontroller can communicate with the Buck controller directly to change the output as needed instead of manipulating components in the feedback path.

Earlier version of the nominal voltage detection algorithm set the output voltage as either 3V, 6V or 12V depending only on the amount of current being drawn by the load. The algorithm implemented in this iteration finds the turn on voltage of the device by finding the voltage that creates the largest increase in current draw. Following this, it observes the rate of current draw after the turn on voltage, and classifies the device as

being a high power or low power device. Using the turn on voltage and the power classification, the output voltage is set appropriately to one of the following values: 3V, 5V, 6V, 9V, 12V, 15V, 18V, 21V, 24V and 36V. This was found to work with five out of the seven devices tested, with the possibility to improve the results with further testing.

The Smart DC Wall Outlet contains a few areas that could use further improvement, with the worst area being the nominal voltage detection process. The algorithm used in this project only works on devices that can be turned on by applying power to the input and cannot be used on devices that implement a toggle button. Additionally, the turn on characteristics of DC devices vary significantly, and it is near impossible to design a ruleset that could handle every case. The ideal method for detecting the output voltage would involve digitally communicating with the load device, like how USB-C negotiates what voltage to supply. Alternatively, a reinforcement learning network could be trained on a wide set of DC devices and used to predict what output voltage would suit a load device best. Other minor improvements that could be made involve shrinking the physical size of the system. The PCB used for this project was sized to house a full Arduino Uno header, and this could be drastically reduced if only the microcontroller IC was placed on the board. Finally, the efficiency of the system could be further improved for small current loads, as this system was only able to achieve 70% efficiency when the target was 80%.

BIBLIOGRAPHY

[1] Edison Tech Center, "Arc Lamps," 2016. [Online]. Available: http://edisontechcenter.org/ArcLamps.html. [Accessed 4 April 2018].

[2] C. L. Sulzberger, "Triumph of AC - from Pearl Street to Niagra," *IEEE Power and Energy Magazine*, vol. 99, no. 3, pp. 64-67, 2003.

[3] Taufik, Introduction to Power Electronics, 2018, pp. 35-36.

[4] Taufik, "The DC House Project," 13 September 2010. [Online]. Available: https://web.calpoly.edu/~taufik/dchouse/. [Accessed 4 April 2018].

[5] Taufik and M. Muscarella, "Development of DC house prototypes as demonstration sites for an alternate solution to rural electrification," in *2016 6th International Annual Engineering Seminar*, 2016.

[6] C. K. Alexander and M. N. O. Sadiku, "6.4 Inductors," in *Fundamentals of Electric Circuits*, New York, McGraw-Hill, 2013, p. 226.

[7] M. Detmers and T. Blauvelt, "Variable DC Voltage Wall Outlet for The DC House Project," San Luis Obispo, 2011.

[8] E. Sibal, "Smart Wall Plug Design For The DC House Project," San Luis Obispo, 2012.

[9] K. Mendoza, "Smart Wall Outlet Design and Implementation for the DC House Project," San Luis Obispo, 2014.

[10] R. Liu, "Smart DC/DC Wall Plug Design for the DC House Project," San Luis Obispo, 2017.

[11] D. Ngo, "USB Type-C and Thunderbolt 3: One port to connect them all," CNET, 26 October 2016. [Online]. Available: https://www.cnet.com/how-to/usb-type-c-thunderbolt-3-one-cable-to-connect-them-all/. [Accessed 12 April 2019].

[12] NEMA, "Wiring Devices - Dimensional Specifications," 14 April 2016. [Online]. Available: https://www.nema.org/standards/pages/wiring-devices-dimensional-specifications.aspx?key=67ri900e6rt5af. [Accessed 12 April 2019].

[13] Analog Devices, Inc., "LTC3889 Datasheet and Product Info," 2019. [Online]. Available: https://www.analog.com/en/products/ltc3889.html. [Accessed 2 June 2019].

[14] SMIF, Inc., "PMBus Power Management Defined," 28 March 2005. [Online]. Available: http://pmbus.org/Home. [Accessed 2 June 2019].

[15] Analog Devices, Inc., "LT8619-5 Datasheet and Product Overview," 2019. [Online]. Available: https://www.analog.com/en/products/lt8619.html. [Accessed 2 June 2019].

[16] Arduino, "Arduino - Board," 2019. [Online]. Available: https://www.arduino.cc/en/reference/board. [Accessed 2 June 2019].

[17] Analog Devices, Inc., "LTC3892 Datasheet and Product Info," 2015. [Online]. Available: https://www.analog.com/en/products/ltc3892.html. [Accessed 4 June 2019].

[18] B. Suppanz, "Trace Width Calculator," 2018. [Online]. Available: https://www.4pcb.com/trace-width-calculator.html. [Accessed 9 June 2019].

[19] Crane Aerospace & Electronics, "Measurement and Filtering of Output Noise of DC-DC Converters," 2016. [Online]. Available: http://www.interpoint.com/product_documents/DC_DC_Converters_Output_Noise.pdf. [Accessed 10 June 2019].

A. Test Device Voltage Sweeps

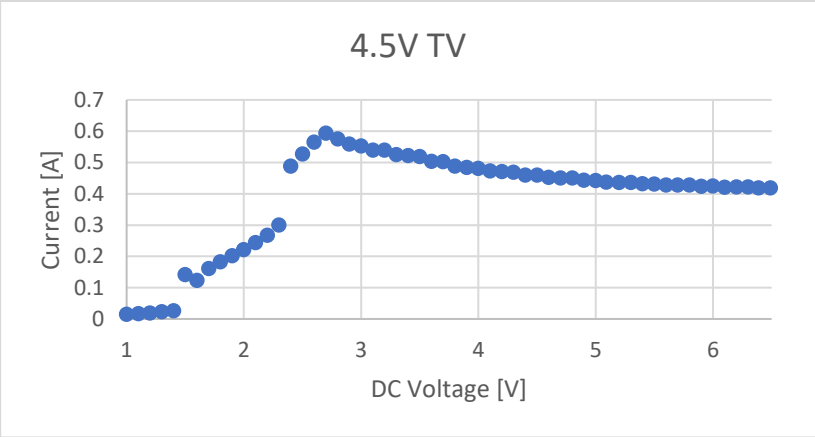

Figure 36: 4.5V TV Voltage Sweep

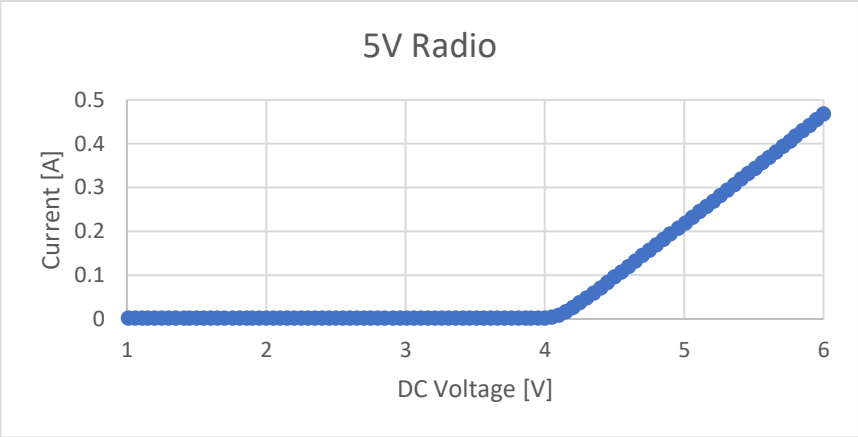

Figure 37: 5V Radio Voltage Sweep

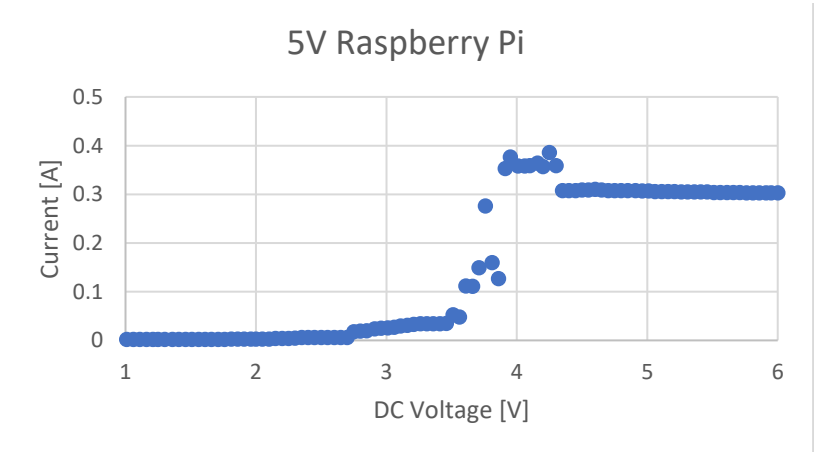

Figure 38: 5V Raspberry Pi Voltage Sweep

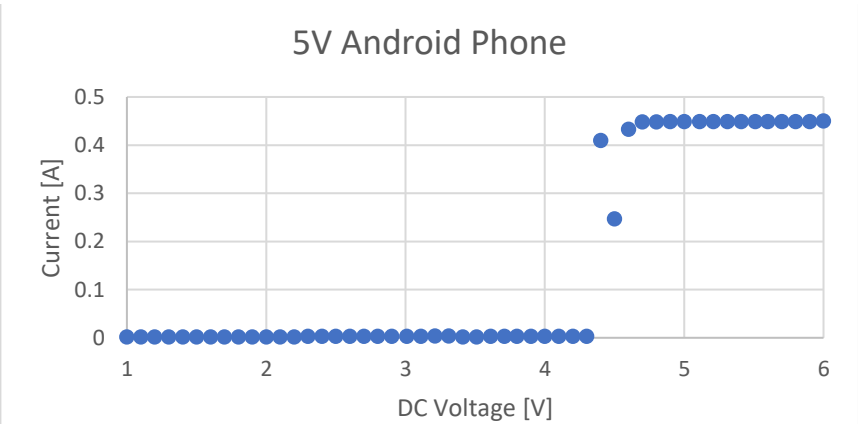

Figure 39: 5V Android Phone Voltage Sweep

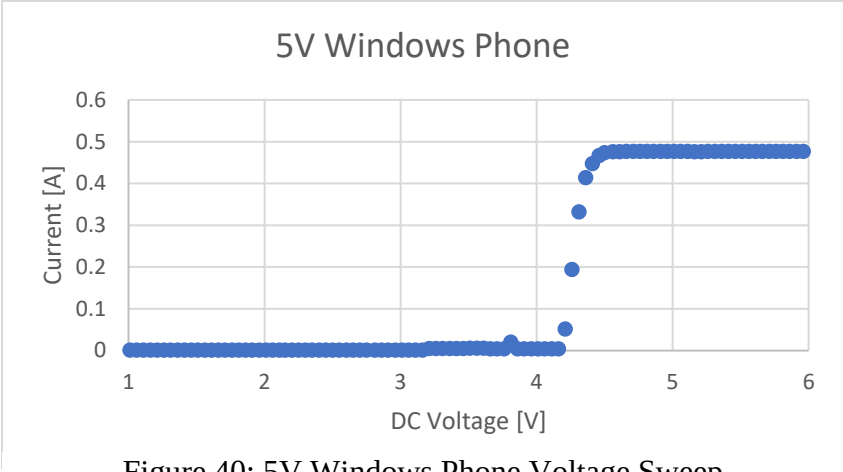

Figure 40: 5V Windows Phone Voltage Sweep

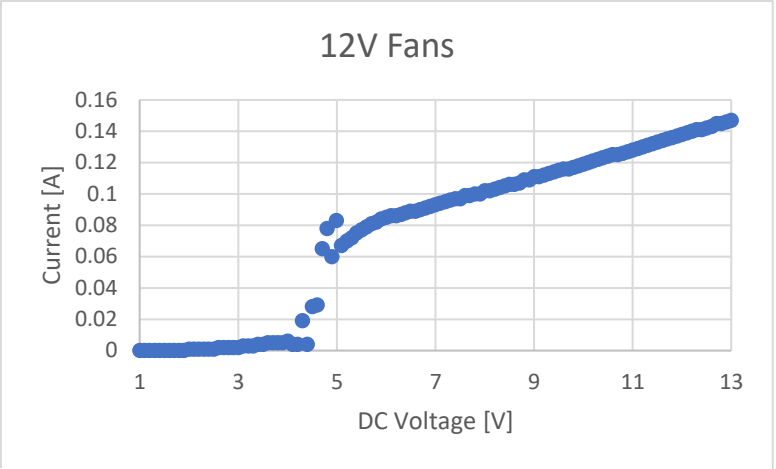

Figure 41: 12V Fans Voltage Sweep

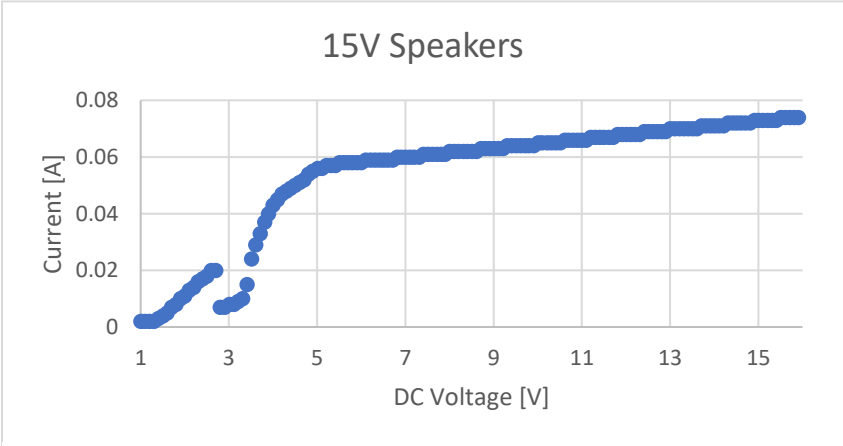

Figure 42: 15V Speakers Voltage Sweep

B. Output Regulation

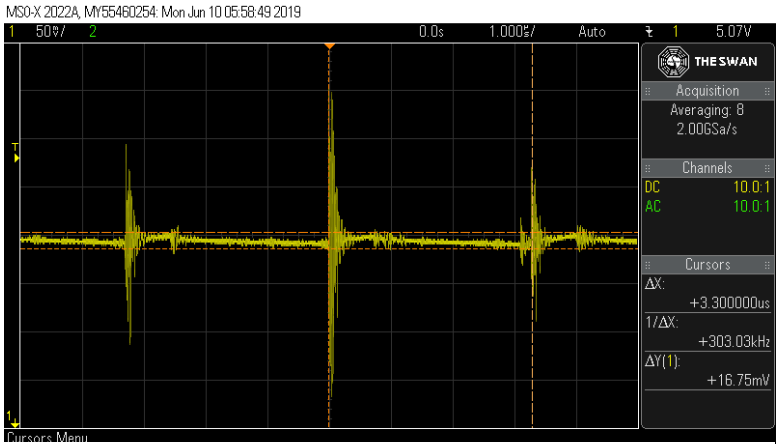

Figure 43: 4.5V TV Output Regulation

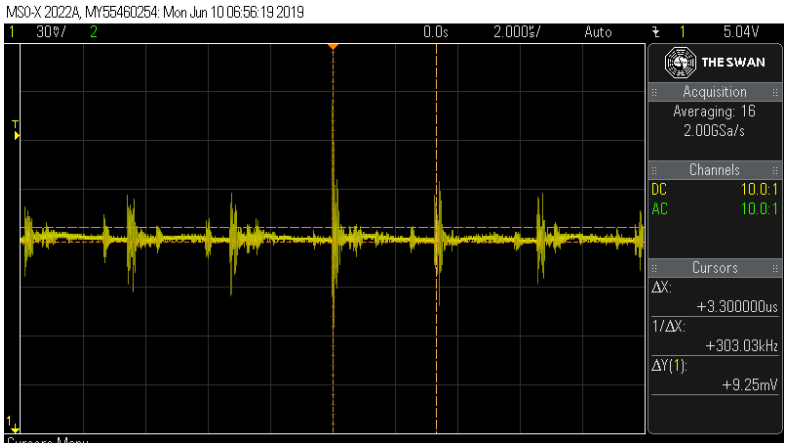

Figure 44: 5V Radio Output Regulation

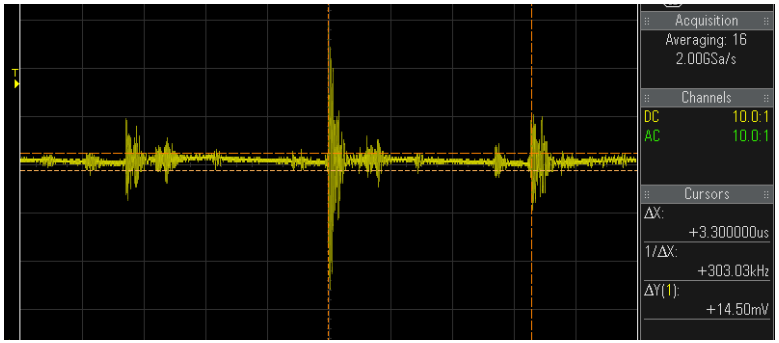

Figure 45: 5V Raspberry Pi Output Regulation

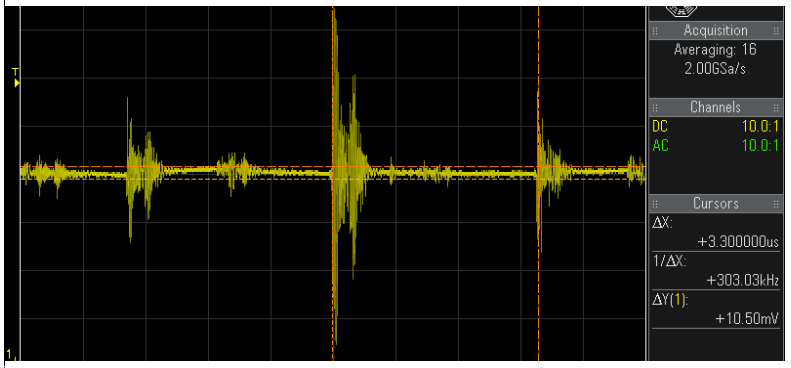

Figure 46: 5V Android Phone Output Regulation

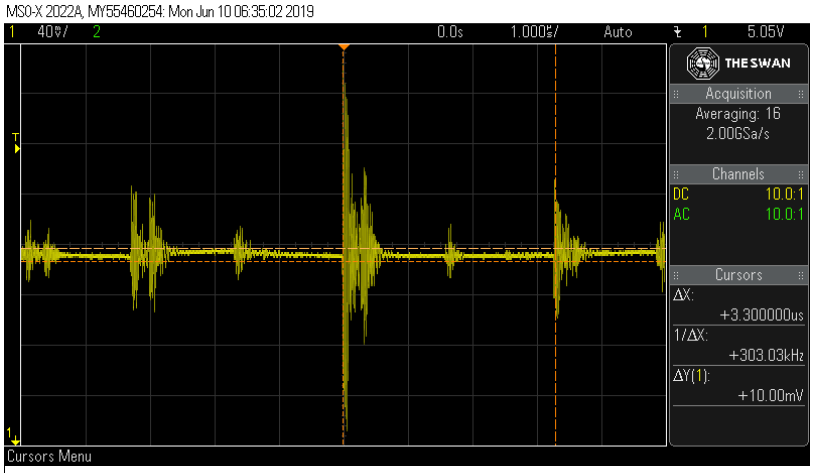

Figure 47: 5V Windows Phone Output Regulation

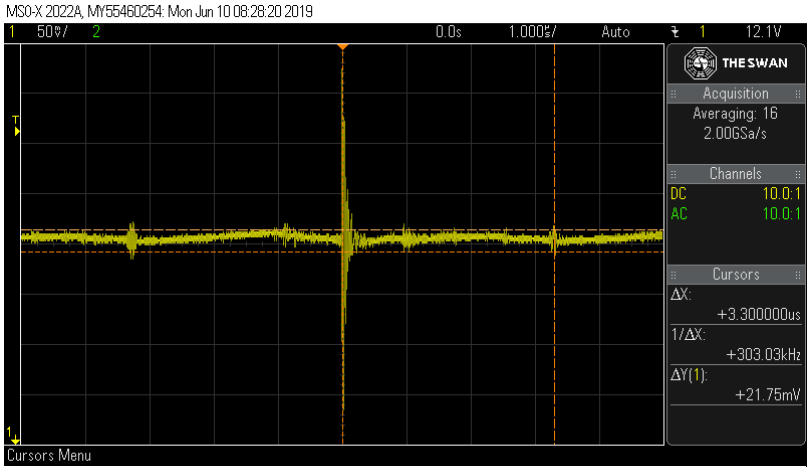

Figure 48: 12V Fans Output Regulation

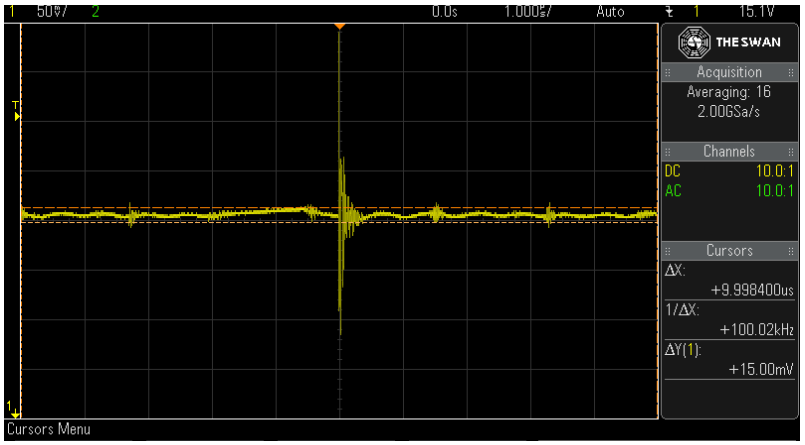

Figure 49: 15V Speakers Output Regulation

C. Microcontroller Code

```
#include <Wire.h>

const uint8_t SLAVE_ADDRESS = 0x5B;
const uint8_t MFR_COMMON = 0xEF;
const uint8_t MFR_BUSY_MASK = 0x70;

const uint8_t window_size = 5;

void setup() {
  Serial.begin(9600);

  Wire.begin();
  Wire.setClock(100000); //100kHz
}

void loop() {
  bool on = false;
  int state = 0;
  float voltage = 1.0;
  float new_voltage = 1.0;

  //regulate_output(5.0, voltage);

  while(1) {
    switch(state) {
      case 0:
        //No device connected. For now just wait 5 seconds before sweep attempts
        Serial.println(F("State 0: No device detected."));

        delay(10000); //5 seconds
        state = 1;
        break;

      case 1:
        //Device Connected. Find nominal voltage
        Serial.println(F("State 1: Finding Nominal Voltage"));

        voltage = find_nominal();

        if (voltage < 3.0) {
          //No device detected
          state = 0;
        } else {
          //Device detected
          state = 2;
          on = true;
        }
        break;

      case 2:
        //Nominal voltage set. Maintain until device disconnected
        Serial.println(F("State 2: Maintaining Output Voltage"));

        delay(5000); //5 seconds

        on = is_connected();
```

```
        if (!on) {
          //Revert to 1V,
          voltage = change_vout(1.0, voltage);
          state = 0;
        }

        break;
    }
  }
}

void regulate_output(float target, float previous) {
  //This function sets the output then continuously prints the current usage.
  float current_val = 0.0;

  change_vout(target, previous);

  while(true) {
    current_val = read_current();
    Serial.print(F("Current: "));
    Serial.print(current_val, 3);
    Serial.println(F("A"));
    delay(1000);
  }
}

float find_nominal() {
  //This is the voltage finding algorithm
  //Arr size set to 150 to reduce ram usage. Should be increased for high volt devices
  const int arr_size = 150;
  const float upper_limit = 12.1;

  int start = 0;
  int finish = 20;

  float voltage = 1.0;
  float mean = 0.0;
  float dev = 0.0;
  int peak_index = 0;
  bool peak_found = false;
  bool peak_confirmed = false;
  bool passed_6v = false;
  bool battery_case = false;

  //These arrays hold all of the mean current values and derivatives
  float mean_current[arr_size];
  float mean_current_diff[arr_size];
  float slope_check[10];

  Serial.println(F("Grabbing 1 to 3V"));

  //Get 1V-3V, calc mean and derivative
  voltage = get_current_range(voltage, start, finish, mean_current);
  get_diff(start, finish, mean_current, mean_current_diff);
  peak_index = get_max_index(mean_current_diff, finish);
  mean = get_mean(mean_current_diff, finish);

  while(!peak_found) {
    Serial.println(F("Checking if Peak Found"));
```

```
    //Peak found when value is > 180% of mean
    if ((mean_current_diff[peak_index] > 1.8 * mean) && (mean_current[peak_index] >=
0.02)) {

      Serial.print(F("Peak found: "));
      Serial.println(index_to_volt(peak_index), 2);

      peak_found = true;
    } else {
      Serial.println(F("Peak not found yet"));

      //Grab next voltage range while less than upper limit
      if (index_to_volt(finish + 10) <= upper_limit) {
        Serial.println(F("Finding New Range"));
        start = finish + 1;
        finish = finish + 10;

        voltage = get_current_range(voltage, start, finish, mean_current);
        Serial.println(F("Got range, calc diff"));
        get_diff(start, finish, mean_current, mean_current_diff);
        Serial.println(F("Got diff, getting peak"));
        peak_index = get_max_index(mean_current_diff, finish);
        Serial.println(F("Got peak, getting mean"));
        mean = get_mean(mean_current_diff, finish);
        Serial.println(F("Colleted new range"));
      } else {
        //Failure, hit upper limit
        Serial.println(F("Failure to find peak"));

        Serial.println(F("Mean Current: "));
        for (int k = 0; k <= finish; k++) {
          Serial.print(index_to_volt(k), 2);
          Serial.print(F("V: "));
          Serial.println(mean_current[k], 4);
        }

        change_vout(1.0, voltage);
        return 1.0;
      }
    }
  }

  Serial.println(F("Attempting to confirm peak"));
  while(!peak_confirmed && index_to_volt(finish) < upper_limit) {
    peak_confirmed = true;

    Serial.println(F("Checking if need to increase range"));

    if (peak_index + 10 > finish) {
      Serial.println(F("Increasing range"));
      //Does this pass 6V?
      if((peak_index + 10 > 50) && !passed_6v) {
        //Need to check for battery case
        start = finish + 1;
        finish = 50;
        passed_6v = true;
      } else {
        start = finish + 1;
        finish = peak_index + 10;
      }
```

```
      voltage = get_current_range(voltage, start, finish, mean_current);
      get_diff(start, finish, mean_current, mean_current_diff);
    }

    if ((passed_6v) && finish <= 50) {
      Serial.println(F("Checking if battery"));
      battery_case = true;

      for (int i = 41; i <= 50; i++) {
        dev = (abs(mean_current_diff[i] - mean_current_diff[peak_index]) /
mean_current_diff[peak_index]) * 100;

        if (dev > 15) {
          //Slope was not constant, std > 15%
          Serial.println(F("Not battery"));
          battery_case = false;
          break;
        }
      }

      if (battery_case) {
        Serial.println(F("Battery Confirmed!"));
        change_vout(5.0, voltage);
        return 5.0;
      }
    }

    if(finish - peak_index < 10) {
      //If battery checked on this iteration, wait til next iteration
      peak_confirmed = false;
    } else {
      Serial.println(F("Checking volt after peak"));
      //Try to confirm volt after peak
      for (int i = peak_index + 1; i <= finish; i++) {
        if (mean_current_diff[i] - mean_current_diff[peak_index] > -0.001) {
          peak_confirmed = false;
          peak_index = i;
          Serial.print(F("New peak found: "));
          Serial.println(index_to_volt(peak_index));
          break;
        }
      }
    }
  }

  if(!peak_confirmed) {
    Serial.println(F("Could not confirm peak in limits"));
    //Print debug info
    Serial.println(F("Mean Current: "));
    for (int k = 0; k <= finish; k++) {
      Serial.print(index_to_volt(k), 2);
      Serial.print(F("V: "));
      Serial.println(mean_current[k], 4);
    }

    Serial.println(F("Diff: "));
    for (int k = 0; k <= finish; k++) {
      Serial.print(index_to_volt(k), 2);
      Serial.print(F("V: "));
      Serial.println(mean_current_diff[k], 4);
```

```
    }
    change_vout(1.0, voltage);
    return 1.0;
  }

  if (index_to_volt(peak_index) < 2.4) {
    Serial.println(F("3V Device"));
    change_vout(3.0, voltage);
    return 3.0;
  } else if (index_to_volt(peak_index) <= 5.0) {
    Serial.println(F("Turn on less than 5V"));
    if (finish < 50) {
      //collect data up to 6V
      start = finish + 1;
      finish = 50;

      Serial.println(F("Extending current to 6V"));
      voltage = get_current_range(voltage, start, finish, mean_current);
      Serial.println(F("Calculating Diff"));
      get_diff(start, finish, mean_current, mean_current_diff);
    }

    Serial.println(F("Filling Slope Check"));
    //Check the slope from 5V to 6V
    for (int i = 41; i <= 50; i++) {
      slope_check[i%10] = mean_current_diff[i];
    }

    Serial.println(F("qsort"));
    qsort(slope_check, sizeof(slope_check) / sizeof(slope_check[0]),
sizeof(slope_check[0]), sort_float);

    Serial.print("Median: ");
    Serial.println(slope_check[5], 2);
    //median
    if(slope_check[5] >= 0.02) {
      Serial.println(F("12V Device"));
      if(upper_limit >= 12.0) {
        change_vout(12.0, voltage);
        return 12.0;
      } else {
      Serial.println(F("Failure, went above limit"));

        Serial.println(F("Mean Current: "));
        for (int k = 0; k <= finish; k++) {
          Serial.print(index_to_volt(k), 2);
          Serial.print(F("V: "));
          Serial.println(mean_current[k], 4);
        }

        Serial.println(F("Diff: "));
        for (int k = 0; k <= finish; k++) {
          Serial.print(index_to_volt(k), 2);
          Serial.print(F("V: "));
          Serial.println(mean_current_diff[k], 4);
        }

        change_vout(1.0, voltage);
        return 1.0;
      }
```

```
    } else {
      Serial.println(F("5V Device"));

      Serial.println(F("Mean Current: "));
      for (int k = 0; k <= finish; k++) {
        Serial.print(index_to_volt(k), 2);
        Serial.print(F("V: "));
        Serial.println(mean_current[k], 4);
      }

      Serial.println(F("Diff: "));
      for (int k = 0; k <= finish; k++) {
        Serial.print(index_to_volt(k), 2);
        Serial.print(F("V: "));
        Serial.println(mean_current_diff[k], 4);
      }

      change_vout(5.0, voltage);
      return 5.0;
    }
  } else if ((index_to_volt(peak_index) < 6) && 6 <= upper_limit) {
      Serial.println(F("6V Device"));
      change_vout(6.0, voltage);
      return 6.0;
  } else if ((index_to_volt(peak_index) < 9) && 9 <= upper_limit) {
      Serial.println(F("9V Device"));
      change_vout(9.0, voltage);
      return 9.0;
  } else if ((index_to_volt(peak_index) < 12) && 12 <= upper_limit) {
    Serial.println(F("12V Device"));
    change_vout(12.0, voltage);
    return 12.0;
  } else if ((index_to_volt(peak_index) < 15) && 15 <= upper_limit) {
    Serial.println(F("15V Device"));
    change_vout(15.0, voltage);
    return 15.0;
  } else {
    change_vout(1.0, voltage);
    return 1.0;
  }
}

int sort_float(const void *cmp1, const void *cmp2) {
  //Sorting function for qsort
  float a = *((float *)cmp1);
  float b = *((float *)cmp2);

  return a > b ? 1 : (a < b ? -1 : 0);
}

float get_mean(float arr[], int finish) {
  float sum = 0.0;

  for(int i = 0; i <= finish; i++) {
    sum += arr[i];
  }

  return sum / (finish + 1);
}
```

```
int get_max_index(float arr[], int finish) {
  float max_val = 0.0;
  int index = 0;

  for (int i = 0; i <= finish; i++) {
    if (arr[i] > max_val) {
      max_val = arr[i];
      index = i;
    }
  }

  return index;
}

void get_diff(int start, int finish, float current[], float current_diff[]){
  //Calculate discrete derivative
  if (start == 0) {
    current_diff[0] = 0;
    start = 1;
  }

  for (int i = start; i <= finish; i++) {
    current_diff[i] = (current[i] - current[i-1]) / 0.1;
  }
}

float get_current_range(float voltage, int start, int finish, float current[]) {
  //collect data over specified current range
  float new_voltage = index_to_volt(start);

  //If starting index is different that current voltage
  if (abs(new_voltage - voltage) > 0.01) {
    voltage = change_vout(new_voltage, voltage);
  }

  current[start] = read_current();

  //Grab the current from this range of voltages
  for (int i = start + 1; i <= finish; i++) {
    voltage = change_vout(index_to_volt(i), voltage);
    delay(1000); //500ms per voltage change
    current[i] = read_current();
  }
  //Get moving average
  moving_avg(start, finish, current);

  return voltage;
}

void moving_avg(int start, int finish, float current[]) {
  //Assumes that the window size is less than 10, the first call starts at 0 and all
further calls start > window_size;

  float current_buffer[window_size] = { };

  if(start == 0) {
    for(int i = 0; i < window_size; i++) {
      current_buffer[i] = current[i];
      current[i] = 0.0;
```

```
      for(int j = 0; j <= i; j++) {
        current[i] += current_buffer[j];
      }
      current[i] = current[i] / ((float) i + 1);
    }

    for(int i = window_size; i <= finish; i++) {
      current_buffer[i % window_size] = current[i];
      current[i] = 0.0;

      for(int j = 0; j < window_size; j++) {
        current[i] += current_buffer[j];
      }
      current[i] = current[i] / window_size;
    }
  } else {
    //fill current buffer first

    for(int i = 0; i < window_size - 1; i++) {
      current_buffer[i] = current[start - window_size + 1 + i];
    }

    for(int i = start; i <= finish; i++) {
      current_buffer[(i - start + window_size - 1) % window_size] = current[i];
      current[i] = 0.0;

      for(int j = 0; j < window_size; j++) {
        current[i] += current_buffer[j];
      }

      current[i] = current[i] / window_size;
    }
  }
}

float index_to_volt(int index) {
  return (index + 10) / 10.0;
}

bool is_connected() {
  float current = 0.0;

  current = read_current();

  return current >= 0.01;
}

void set_led(uint8_t input) {
  const int led0 = 8;
  const int led1 = 9;
  const int led2 = 10;

  const int LED0_MASK = 0x1;
  const int LED1_MASK = 0x2;
  const int LED2_MASK = 0x4;

  digitalWrite(led0, (input & LED0_MASK) > 0);
  digitalWrite(led1, (input & LED1_MASK) > 0);
  digitalWrite(led2, (input & LED2_MASK) > 0);
}
```

```
float read_current() {
  const uint8_t READ_IOUT = 0x8C;

  uint16_t current_L11 = 0;

  current_L11 = safe_read_word(READ_IOUT);

  return lin5_11ToFloat(current_L11);
}

float change_vout(float desired, float previous) {
  //Changes the output voltage, and the upper/lower limits
  //See LTC3889 datasheet for more information
  const uint8_t VOUT_OV_FAULT_LIMIT = 0x40;
  const uint8_t VOUT_OV_WARN_LIMIT = 0x42;
  const uint8_t VOUT_MARGIN_HIGH = 0x25;
  const uint8_t VOUT_COMMAND = 0x21;
  const uint8_t VOUT_MARGIN_LOW = 0x26;
  const uint8_t VOUT_UV_WARN_LIMIT = 0x43;
  const uint8_t VOUT_UV_FAULT_LIMIT = 0x44;

  uint16_t new_ov_fault = getLin16(desired + 0.5);
  uint16_t new_ov_warn = getLin16(desired + 0.375);
  uint16_t new_margin_high = getLin16(desired + 0.25);
  uint16_t new_vout = getLin16(desired);
  uint16_t new_margin_low = getLin16(desired - 0.25);
  uint16_t new_uv_warn = getLin16(desired - 0.375);
  uint16_t new_uv_fault = getLin16(desired - 0.5);

  set_current_limit(desired);

  Serial.print(F("Previous: "));
  Serial.print(previous);
  Serial.print(F(", Desired: "));
  Serial.println(desired);

  if(desired > previous) {
    //Change upper limits first, then lower
    safe_write_word(VOUT_OV_FAULT_LIMIT, new_ov_fault);
    safe_write_word(VOUT_OV_WARN_LIMIT, new_ov_warn);
    safe_write_word(VOUT_MARGIN_HIGH, new_margin_high);
    safe_write_word(VOUT_COMMAND, new_vout);
    safe_write_word(VOUT_MARGIN_LOW, new_margin_low);
    safe_write_word(VOUT_UV_WARN_LIMIT, new_uv_warn);
    safe_write_word(VOUT_UV_FAULT_LIMIT, new_uv_fault);
  } else {
    //Change lower limits first, then upper
    safe_write_word(VOUT_UV_FAULT_LIMIT, new_uv_fault);
    safe_write_word(VOUT_UV_WARN_LIMIT, new_uv_warn);
    safe_write_word(VOUT_MARGIN_LOW, new_margin_low);
    safe_write_word(VOUT_COMMAND, new_vout);
    safe_write_word(VOUT_MARGIN_HIGH, new_margin_high);
    safe_write_word(VOUT_OV_WARN_LIMIT, new_ov_warn);
    safe_write_word(VOUT_OV_FAULT_LIMIT, new_ov_fault);
  }

  return desired;
}
```

```
void set_current_limit(float voltage) {
  //Sets the current limit to be <6A and <100W
  const uint8_t IOUT_OC_FAULT_LIMIT = 0x46;
  const uint8_t IOUT_OC_WARN_LIMIT = 0x4A;

  uint16_t new_oc_warn = 0;
  uint16_t new_oc_fault = 0;

  if(voltage < 16.7) {
    new_oc_fault = getLin5_11(int8_t(-3), int16_t(600)); //set to 75mV (7.5A)
    new_oc_warn = getLin5_11(int8_t(-3), int16_t(480)); //set to 60mV (6.0A)
  } else {
    float fault_limit = (100.0 / voltage) + 1;
    float warn_limit = fault_limit * 0.8;

    int16_t fault_mantissa = round((fault_limit * 10) / pow(2, -3));
    int16_t warn_mantissa = round((warn_limit*10) / pow(2, -3));

    new_oc_fault = getLin5_11(int8_t(-3), fault_mantissa); //set to 75mV (7.5A)
    new_oc_warn = getLin5_11(int8_t(-3), warn_mantissa); //set to 60mV (6.0A)
  }

  safe_write_word(IOUT_OC_FAULT_LIMIT, new_oc_fault);
  safe_write_word(IOUT_OC_WARN_LIMIT, new_oc_warn);
}


void write_ee_prom() {
  //Run Once to Configure the behavior on startup
  const uint8_t PAGE = 0x00;
  const uint8_t MFR_PWM_MODE_LTC3889 = 0xD4;
  const uint8_t FREQUENCY_SWITCH = 0x33;
  const uint8_t MFR_RVIN = 0xF7;
  const uint8_t VIN_OV_FAULT_LIMIT = 0x55; //L11
  const uint8_t IOUT_CAL_GAIN = 0x38;
  const uint8_t OT_FAULT_RESPONSE = 0x50;
  const uint8_t UT_FAULT_RESPONSE = 0x54;
  const uint8_t CLEAR_FAULTS = 0x03;
  const uint8_t OPERATION = 0x01;
  const uint8_t STORE_USER_ALL = 0x15; //Send Byte
  const uint8_t MFR_RESET = 0xFD;

  uint8_t pwm_mode_setting = B11001000; //High Current Range, Servo Mode, 9V drive
voltage, Discontinuous operation
  uint16_t switch_freq = getLin5_11(int8_t(0), int16_t(300)); //300kHz
  uint16_t in_res = getLin5_11(int8_t(0), int16_t(2)); //0mR
  uint16_t out_res = getLin5_11(int8_t(0), int16_t(10)); //10mR
  uint16_t vin_fault_limit = getLin5_11(int8_t(0), int16_t(54)); //54V

  float voltage = 5.0;

  safe_write_byte(PAGE, uint8_t(0x00)); //Set channel to 0
  safe_write_byte(MFR_PWM_MODE_LTC3889, pwm_mode_setting); //Set operating mode
  safe_write_word(FREQUENCY_SWITCH, switch_freq); //Set switching frequency
  safe_write_word(MFR_RVIN, in_res); //Set input filter resistance
  safe_write_word(VIN_OV_FAULT_LIMIT, vin_fault_limit); //Set VIN faul limit
  safe_write_word(IOUT_CAL_GAIN, out_res); //Set current sense resistance
  safe_write_byte(OT_FAULT_RESPONSE, uint8_t(0)); //Set no response for OT event
  safe_write_byte(UT_FAULT_RESPONSE, uint8_t(0)); //Set no response for UT event
```

```
  voltage = change_vout(1.0, voltage);

  safe_send_byte(CLEAR_FAULTS); //Clear any faults
  delay(10);
  safe_write_byte(OPERATION, uint8_t(0x80)); //Set the device to begin output

  safe_send_byte(STORE_USER_ALL); //Save settings to EEPROM

  delay(1000);

  safe_send_byte(MFR_RESET);
}

float test_output(float new_voltage, float old_voltage) {
  const uint8_t PAGE = 0x00;
  const uint8_t MFR_PWM_MODE_LTC3889 = 0xD4;
  const uint8_t FREQUENCY_SWITCH = 0x33;
  const uint8_t MFR_RVIN = 0xF7;
  const uint8_t VIN_OV_FAULT_LIMIT = 0x55; //L11
  const uint8_t IOUT_CAL_GAIN = 0x38;
  const uint8_t OT_FAULT_RESPONSE = 0x50;
  const uint8_t UT_FAULT_RESPONSE = 0x54;
  const uint8_t CLEAR_FAULTS = 0x03;
  const uint8_t OPERATION = 0x01;
//  const uint8_t MFR_RESET = 0xFD;

  //High Current Range, Servo Mode, 7.4V drive voltage, Discontinuous operation
  uint8_t pwm_mode_setting = B11000100;
  uint16_t switch_freq = getLin5_11(int8_t(0), int16_t(300)); //300kHz
  uint16_t in_res = getLin5_11(int8_t(0), int16_t(2)); //0mR
  uint16_t out_res = getLin5_11(int8_t(0), int16_t(10)); //10mR
  uint16_t vin_fault_limit = getLin5_11(int8_t(0), int16_t(54)); //54V

  safe_write_byte(PAGE, uint8_t(0x00)); //Set channel to 0
  safe_write_byte(MFR_PWM_MODE_LTC3889, pwm_mode_setting); //Set operating mode
  safe_write_word(FREQUENCY_SWITCH, switch_freq); //Set switching frequency
  safe_write_word(MFR_RVIN, in_res); //Set input filter resistance
  safe_write_word(VIN_OV_FAULT_LIMIT, vin_fault_limit); //Set VIN faul limit
  safe_write_word(IOUT_CAL_GAIN, out_res); //Set current sense resistance
  safe_write_byte(OT_FAULT_RESPONSE, uint8_t(0)); //Set no response for OT event
  safe_write_byte(UT_FAULT_RESPONSE, uint8_t(0)); //Set no response for UT event

  change_vout(new_voltage, old_voltage);

  safe_send_byte(CLEAR_FAULTS); //Clear any faults
  delay(1);
  safe_write_byte(OPERATION, uint8_t(0x80)); //Set the device to begin output

  return new_voltage;
}

uint8_t safe_read_byte(uint8_t command_byte) {
  uint8_t mfr_common_val = 0;
  uint8_t part_ready = 0;
  uint8_t retval = 0;

  do {
    mfr_common_val = PMBUS_READ_BYTE(MFR_COMMON);
    part_ready = (mfr_common_val & MFR_BUSY_MASK) == MFR_BUSY_MASK;
  } while (!part_ready);
```

```
  retval = PMBUS_READ_BYTE(command_byte);

  return retval;
}

uint16_t safe_read_word(uint8_t command_byte) {
  uint8_t mfr_common_val = 0;
  uint8_t part_ready = 0;
  uint16_t retval = 0;

  do {
    mfr_common_val = PMBUS_READ_BYTE(MFR_COMMON);
    part_ready = (mfr_common_val & MFR_BUSY_MASK) == MFR_BUSY_MASK;
  } while (!part_ready);

  retval = PMBUS_READ_WORD(command_byte);

  return retval;
}

uint8_t safe_write_byte(uint8_t command_byte, uint8_t data_byte) {
  uint8_t mfr_common_val = 0;
  uint8_t part_ready = 0;
  uint8_t retval = 0;

  do {
    mfr_common_val = PMBUS_READ_BYTE(MFR_COMMON);
    part_ready = (mfr_common_val & MFR_BUSY_MASK) == MFR_BUSY_MASK;
  } while (!part_ready);

  retval = PMBUS_WRITE_BYTE(command_byte, data_byte);

//  if(retval == 0) {
//    Serial.print(F("Successful Write Byte\n"));
//  } else {
//    Serial.print(F("Failed Write Byte\n"));
//  }

  return retval;
}

uint8_t safe_write_word(uint8_t command_byte, uint16_t data_word) {
  uint8_t mfr_common_val = 0;
  uint8_t part_ready = 0;
  uint8_t retval = 0;

  do {
    mfr_common_val = PMBUS_READ_BYTE(MFR_COMMON);
    part_ready = (mfr_common_val & MFR_BUSY_MASK) == MFR_BUSY_MASK;
  } while (!part_ready);

  retval = PMBUS_WRITE_WORD(command_byte, data_word);

//  if(retval == 0) {
//    Serial.print(F("Successful Write Word\n"));
//  } else {
//    Serial.print(F("Failed Write Word\n"));
//  }
```

```
  return retval;
}

uint8_t safe_send_byte(uint8_t command_byte) {
  uint8_t mfr_common_val = 0;
  uint8_t part_ready = 0;
  uint8_t retval = 0;

  do {
    mfr_common_val = PMBUS_READ_BYTE(MFR_COMMON);
    part_ready = (mfr_common_val & MFR_BUSY_MASK) == MFR_BUSY_MASK;
  } while (!part_ready);

  retval = PMBUS_SEND_BYTE(command_byte);

//  if(retval == 0) {
//    Serial.print(F("Successful Send Byte\n"));
//  } else {
//    Serial.print(F("Failed Send Byte\n"));
//  }

  return retval;
}

uint8_t PMBUS_READ_BYTE(uint8_t command_byte) {
  uint8_t retval = 0;
  uint8_t errors = 0;

  Wire.beginTransmission(SLAVE_ADDRESS);
  Wire.write(command_byte);
  errors = Wire.endTransmission(false);

  //Request the byte
  Wire.requestFrom(SLAVE_ADDRESS, uint8_t(1));
  retval = Wire.read();

  return retval;
}

uint16_t PMBUS_READ_WORD(uint8_t command_byte) {
  uint8_t low = 0, high = 0, errors = 0;
  uint16_t retval = 0;

  Wire.beginTransmission(SLAVE_ADDRESS);
  Wire.write(command_byte);
  errors = Wire.endTransmission(false);

  Wire.requestFrom(SLAVE_ADDRESS, uint8_t(2));
  low = Wire.read();
  high = Wire.read();

  retval = (high << 8) | low; //combine the two

  return retval;
}

uint8_t PMBUS_WRITE_BYTE(uint8_t command_byte, uint8_t data_byte) {
  uint8_t stat = 0;

  Wire.beginTransmission(SLAVE_ADDRESS);
```

```
  Wire.write(command_byte);
  Wire.write(data_byte);
  stat = Wire.endTransmission();

  return stat;
}

uint8_t PMBUS_WRITE_WORD(uint8_t command_byte, uint16_t data_word) {
  uint8_t stat = 0;
  uint8_t high = 0;
  uint8_t low = 0;

  high = (data_word & uint16_t(0xFF00)) >> 8;
  low = (data_word & uint16_t(0x00FF));;

  //low byte, then high byte
  Wire.beginTransmission(SLAVE_ADDRESS);
  Wire.write(command_byte);
  Wire.write(low);
  Wire.write(high);
  stat = Wire.endTransmission();

  return stat;
}

uint8_t PMBUS_SEND_BYTE(uint8_t command_byte) {
  uint8_t stat = 0;

  Wire.beginTransmission(SLAVE_ADDRESS);
  Wire.write(command_byte);
  stat = Wire.endTransmission();

  return stat;
}

uint16_t getLin5_11(int8_t exponent, int16_t mantissa) {
  uint16_t retval = 0;

  retval |= (int16_t(exponent) << 11); //place signed exponent in top 5 bits
  retval |= (mantissa & 0x7FF); //place signed mantissa in bottom 11 bits

  return retval;
}

float lin5_11ToFloat(uint16_t data_word) {
  //https://embeddedartistry.com/blog/2018/7/5/linear11-the-easy-way
  typedef struct {
    int16_t mantissa : 11;
    int16_t exponent : 5;
  } linear11_t;

  typedef union {
    linear11_t linear;
    uint16_t raw;
  } linear11_val_t;

  float retval = 0.0;

  linear11_val_t val;
```

```
  val.raw = data_word;

  retval = val.linear.mantissa * (float) pow(2,val.linear.exponent);

  return retval;
}

uint16_t getLin16(float num) {
  uint16_t retval = 0;

  retval = round(num / pow(2, -10)); //Divide by 2^-10 to get the mantissa

  return retval;
}

float lin16toFloat(uint16_t mantissa) {
  float retval = 0.0;

  retval = mantissa * pow(2, -10); // multiply by 2^-10 to get the float

  return retval;
}
```