5-2015

# Maximum Power Point Tracking and solar power for developing countries around the world

Christopher M. Plate
*University of Arkansas, Fayetteville*

# MAXIMUM POWER POINT TRACKING AND SOLAR POWER FOR DEVELOPING COUNTRIES AROUND THE WORLD

# MAXIMUM POWER POINT TRACKING AND SOLAR POWER FOR DEVELOPING COUNTRIES AROUND THE WORLD

An Undergraduate Honors College Thesis

in the

Department of Electrical Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

by

Chris Plate

This thesis is approved.

Thesis Advisor:

[illegible]

Thesis Committee:

## Abstract

Renewable energy is becoming increasingly important as more and more countries become industrialized. Using solar power as opposed to fossil fuels and coal is becoming cheaper and easier to access throughout the world, yet there are still efficiency hurdles to be overcome. Designing a cheap, easily modifiable and reparable Maximum Power Point Tracker would be a stepping stone to bringing solar power to remote parts of the globe. Once designed and made portable, these devices could be distributed throughout developing countries to enhance renewable energy output.

An Arduino-based tracker was chosen for ease of use and access. This base also allowed for easy modification and repair, and provided a stable supply chain. Simulations showed maximum power point tracking was effective in providing necessary power to load batteries. Once built, voltage and efficiency tests of the tracker showed that it increased solar panel efficiency and output. Finally, the tracker was analyzed through a financial lens, assessing the cost effectiveness for developing areas.

With ever developing microelectronics and solar efficiency, a Maximum Power Point Tracker has the potential to greatly impact electricity consumption in developing countries. As electronics continue to get cheaper and more obtainable, the tracker becomes more plausible and easier to build.

## ACKNOWLEDGEMENTS

I would like to thank Dr. McCann for his support and leadership throughout the thesis process.  I would also like to thank Robert Saunders, Dr. Balda, and Dr. Balda’s graduate students for their help and ideas. Their knowledge really pushed me in the right direction.

Most importantly, I would like to thank my group members, Trevon Riley and Chris Tatum, for providing a system in which to test my design and helping me through the senior project.

Finally, I want to thank all the Electrical Engineering classmates I’ve had, the faculty of the department, my family, and my fiancé for getting me through the last four years and the thesis process. I would not have made it without their help and support.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# 1. INTRODUCTION

## 1.1 Problem: Solar Power Inefficiency

As solar power increases in popularity, the need for this power to become more efficient is evident. Clean, renewable energy sources are becoming more desirable throughout the world, and solar power provides this. Unfortunately, solar energy is not as efficient as traditional energy sources such as coal, but electronics can be used to create more stable and efficient sources to offset the problems associated with using solar panels. The problem that arises is that many of these electronics are quite expensive, and do not necessarily work well outside of a larger system. These systems are often very complex, and not easily repaired or modified.

To fix the problem of price and complexity, a low cost, easy to use electronic system can be created to better provide solar power. Making this system simple to modify, economical, and repairable is a necessity, especially if it is to be deployed in rural or developing areas. By creating a streamlined, hardy device, solar power can be made more readily available and affordable than conventional energy use.

## 1.2 Thesis Statement

It is the goal of this research to develop a simplified, low cost, easily reparable, and durable Maximum Power Point Tracker to be deployed in rural and developing areas of the world. The tracker will cost under $75 to produce and be able to handle power up to 50W.

## 1.3 Approach

The first step in developing the Maximum Power Point Tracker was to decide the type of solar panel and battery it would be connected to. After a 12V battery and solar panel providing 22V at peak power were selected, the topology for the MPPT was selected. A simple buck converter, Arduino controlled tracker was developed on paper and this model went on to simulation.

Once the buck converter was designed, ancillary portions were added to the circuit and rigorously simulated until all necessary pieces were attached. The entire system was simulated meticulously to ensure the design performed as desired. During this simulation process, the tracker was whittled down to its' simplest and most basic components to ensure ease of use, reliability, and cost effectiveness.

The next task was to build a prototype system to see if the system worked in practice. In order to ensure a proper design and easier debugging, a printed circuit board was made for the tracker, and the parts were soldered on. This board was first tested without the solar panel and battery load, and then a battery load was implemented with a solar panel to ensure the system met expectations.

Finally, a financial analysis was done on the components and design as a whole to assess the viability of making the tracker in a low cost manner to ensure it could be readily made and distributed to developing areas of the world. The design was checked to ensure it could be mass produced and easily repaired or modified by anyone familiar with Arduinos and electronics. After all these steps were completed the design was deemed ready for use.

## 1.4 Potential Impact

With an affordable, easy to use MPPT developed, areas that were previously unable to utilize solar power due to cost or complexity now have access to a new renewable energy source. This energy can provide opportunities for growth and development formerly unobtainable due to lack of a sustainable and reliable power. The simplicity of the design and programming platform allows multiple devices to be serviced by few technicians, and requires less training and experience to understand, monitor and repair the device. Finally, the project encourages cleaner electrical power by making it easy and cheap to use solar energy compared traditional methods such as coal and oil.

## 1.5 Organization of Thesis

This thesis is organized into six sections, with each section having multiple subdivisions. Section one is an introductory chapter covering the basic problem, the thesis statement, the approach to solving the problem, and the potential impact the solution could have. The second section is focused on background information regarding solar panel workings, the necessity of power point tracking, and basic buck converter information. The third section explains and simulates the design of the MPPT.  Section four describes the physical implementation of the tracker and the results of loaded and unloaded testing. Section five is a financial analysis of the parts used for the tracker and a study on the feasibility of mass producing and distributing the tracker. The last section draws conclusion based on the results of simulation and testing and the practicality of the design and desired usage.

# 2. BACKGROUND

## 2.1 The Solar Panel Dilemma

A solar panel is a photovoltaic device, meaning that it takes solar energy (sunlight) and converts it to DC electricity. The drawback of these panels is that the environment dictates how much power is produced. The output of a solar is a non-linear curve that is characterized by the fill factor, which helps determine the maximum power a solar cell can provide. This factor, abbreviated "FF," is the ratio of the maximum output power of the solar cell to the product of the open circuit voltage and short circuit current of the cell. This can be seen in Equation 2.1:

$$FF = \frac{P_{max}}{V_{OC} * I_{SC}} \qquad (2.1)$$

The open circuit voltage and short circuit current are the maximum values that the solar panel can produce under ideal conditions. In using solar cells for power applications, it is important to operate at the maximum power possible. Picking an operating voltage lower than the open circuit voltage limits the amount of power that can be taken from the solar cell. Most loads need a certain voltage to operate, so fixing the voltage output of a solar panel is imperative. The current can then be changed by varying the resistance of the load to ensure maximum power is being generated. This maximum power is found on the I-V curve of the solar panel, as seen in Figure 1.

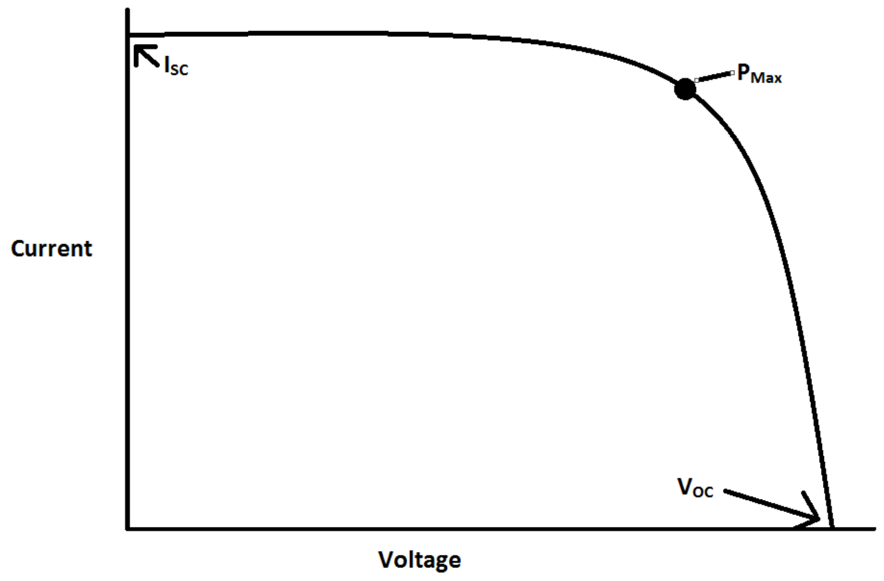

Figure 1: General I-V Characteristic of a Solar Panel

The goal of any charge controller is to achieve the maximum power out of a solar panel given other constraints. This is where the Maximum Power Point Tracker comes in.

## 2.2 Maximum Power Point Tracking

Maximum power point tracking is the art of producing the maximum power from solar panels at different conditions. In a battery charge setting, an MPPT will look at the battery and PV panel and try to match the voltage needs of the battery, the voltage output of the PV panel, and the maximum current the PV can produce at that time. This will provide the maximum power to the battery possible while still using the full output of the solar panel. A converter is used to accomplish the panel voltage step down and associated step up current. By changing the pulse width modulation of a microcontroller based on panel outcomes, the converter is controlled to provide desired results in the system. This switching also increases the efficiency of the panel by providing the maximum power at every charging state. Power is not wasted as heat or noise, but rather driven into the battery as needed.

## 2.3 Buck Converters

A buck converter is simply a DC-DC converter that steps down voltage and steps up current. They are usually very efficient, and are quite simple when compared to many other converter topologies. A buck converter is worked by having two switches, normally a transistor and diode, control the current in an inductor that is in series with the load. A basic buck converter topology can be seen in Figure 2. The switching controls the current and voltage by altering the duty cycle. When the switch is closed, the inductor allows current to flow to the load at a lower voltage than the source. When the switch is open, the stored energy in the inductor is used to drive the load when it is released as current. By switching fast enough and using corrective circuit elements, a stable output of voltage and current can be seen with minimal ripple, giving a lower voltage and higher current than from the source. Using a variable pulse width modulation can account for varying DC supplies, such as a solar panel.

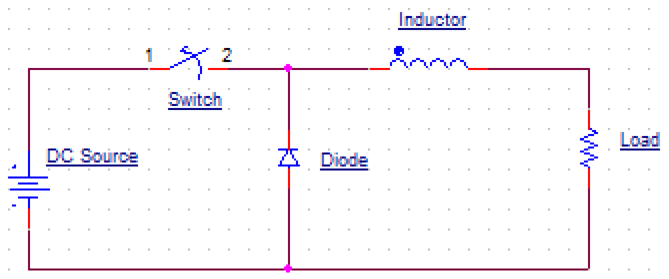

Figure 2: Basic Buck Converter

# 3. MPPT DESIGN AND SIMULATION

## 3.1 Design Parameters

The main goal of this project was to create an affordable, easy to use and manufacture, efficient solar battery charger. With this in mind, a few parameters were set to guide the design process. The first step was to decide what voltage of battery to charge. A very common and easy to find battery is a 12 V, and many power supplies require 12 V to run. These batteries can also store an immense amount of power relative to their size and yet can still be very portable. Overall, a 12 V won out because it is also easier to find and cheaper than a 6 V or a 24 V battery, and the supply of these batteries seems unlimited since there are so many in the world now.

The next parameter decided upon was the microcontroller. An Arduino was chosen for ease of use, supply, and customization ability. Many microcontrollers were studied to provide pulse width modulation, and the Arduino stood out for many reasons. Arduinos can be entirely separate units and just plugged in to a circuit. This is handy for trouble shooting, portability, and functionality. If a microcontroller is needed for more than one task, having two separate circuits with controller plugins as opposed to two circuits with separate microcontrollers is oftentimes cheaper. Arduinos can also come in different shapes and sizes to fit different needs, as opposed to many other microcontrollers which are all one size. Finally, Arduinos are extremely easy to program and use. There is a massive Arduino community with great tutorials on how to code, and Arduinos can work with almost any computer. This ease of use was necessary if the designed controller was going to be distributed.

The final design parameter was simplicity. The controller needed to be as simple as possible in order to facilitate repair, encourage education, and ensure a controller that would be fiscally possible. Making a controller that could be easily understood and repaired enabled almost anyone who wanted to use one able to do so. It also helped ensure multiple controllers could be spread out over a large region. With easy repairs, very few technicians and engineers could monitor large numbers of controllers. The simplicity of the controller would allow these technicians and engineers to also educate those who may want to learn about the controller without requiring an in depth knowledge of circuits and electronics. Finally, creating a simple controller ensured a controller that would at least be feasible with a low amount of money. The cap was set at $75,as this amount was felt to be reasonable for an electronic device that would potentially control much more valuable equipment such as solar panels and batteries. The $75 cap meant that quality did not have to be shirked to get a reasonably performing tracker.

## 3.2 Design of Power Circuitry

After parameters were set, the buck converter was designed. This portion was the backbone of the whole project, and was built first so the rest of the controller could be built around it. The power circuitry can be seen below in Figure 3, highlighted in red boxes.

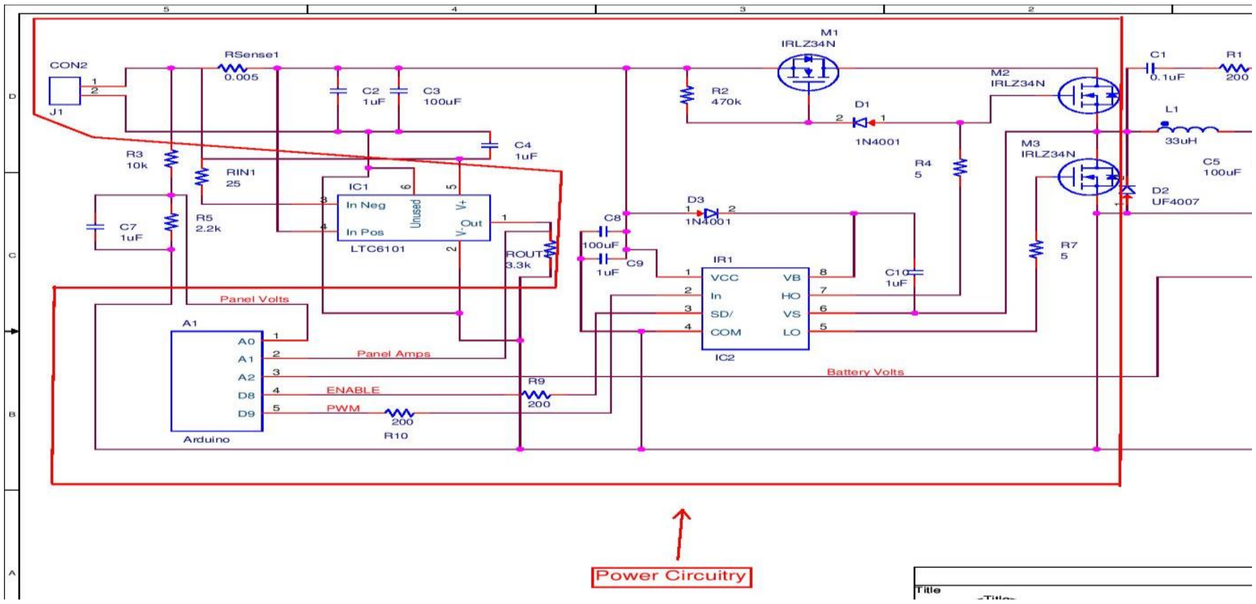

Figure 3: Power Circuitry

The circuitry is fairly strait forward. The power comes in through the jumper and flows through RSense1. The circuit then goes through MOSFET 1, which is there to ensure power flows only the correct way, and not backwards through the system. MOSFET 2 and 3 are then controlled by the IC Driver, which is controlled by the Arduino. This is what charges the inductor of the buck converter. The Arduino controls the Pulse Width Modulation, which is relayed to the MOSFETs through the driver chip. Associated circuitry such as the resistors, capacitors and diodes are there to ensure proper operation of the circuit. The capacitors function to take away transients and correct frequencies. The diodes serve to keep current flowing in the correct direction and protect components. Finally, the resistors help transfer the proper current signal to rightful destinations.

## 3.3 Design of Load Circuitry

The load circuitry was designed with high power in mind. The network can be seen in Figure 4 below.

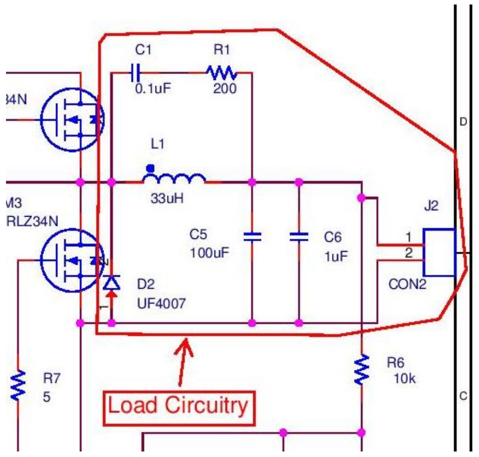

Figure 4: Load Circuitry

The inductor was designed first, using Equation 3.1 below.

$$\boldsymbol{L} = \frac{(\boldsymbol{V_{in,max}} - \boldsymbol{V_{out}}) * \boldsymbol{V_{out}}}{\boldsymbol{I_{ripple}} * \boldsymbol{f_{switching}} * \boldsymbol{V_{in}}} \tag{3.1}$$

Using this led to a calculated inductor value of 28.27uH, so a 33uH inductor was chosen to provide for transients and possible over voltages. The diode was placed to create the rest of the buck converter components. The capacitor in parallel to the inductor was calculated using Equation 3.2.

$$C_{out} = \frac{I_{L,Ripple} * I_{out,max}}{8 * f_{switching} * V_{C,Ripple} * V_{out}} \tag{3.2}$$

Calculations of these components led to an 0.085uF rating for the output capacitor. To save on parts and account for the overcompensation of the inductor, a 0.1uF capacitor was put in along with a 200Ω resistor to finish the load circuitry. The final capacitors were added to ensure a stable output. This load circuitry allowed for the desired high power output while still keeping the cost low and the design simple.

## 3.4 Design of Sensing Networks

The final part of the design process was the design of the sensing networks throughout the system. The sensing networks can be seen in the schematic below, Figure 5.

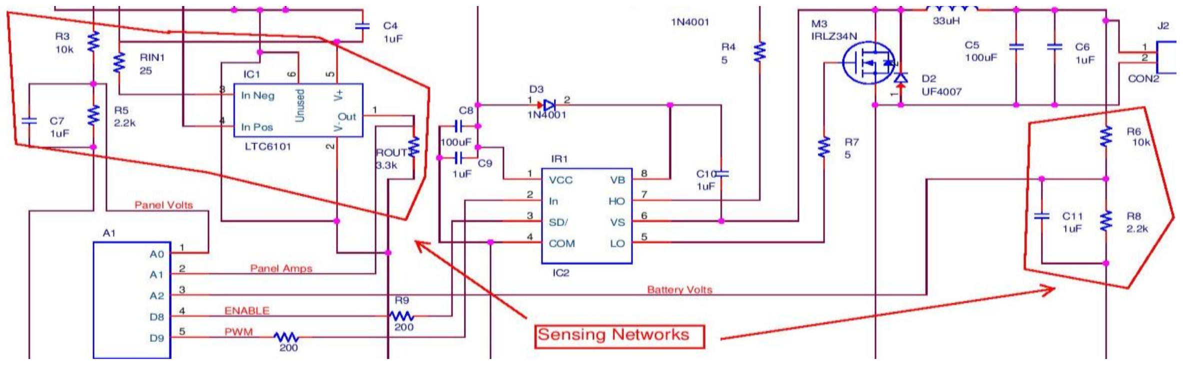

Figure 5: Sensing Networks

Most of the sensing networks were based off of a voltage divider. In order to drop the voltage down to a readable level, a voltage divider was put at the input of the solar panel and output of the battery panel. The voltage needed to be brought down to about 20% of the expected values so that the Arduino could read the values. Using the voltage divider equation (Equation 3.3), the resistance values were found.

$$V_{out} = \frac{R_2 * V_{in}}{R_1 + R_2} \tag{3.3}$$

The resistors had to be sufficiently large enough to make sure there was not a lot of current draw, so kilo-ohm resistors were chosen. The dividers had to have filters as well, so capacitors were added in parallel with the second resistor. The current sensor was chosen to provide an extremely reliable current reading to the Arduino. This was necessary to ensure the most effective and efficient tracking. With a more accurate reading, the tracker can make more informed decisions about the activity of the solar panel, and adjust the output accordingly. With the sensor networks in place, the physical design was completed, and the coding could begin.

## 3.5 Arduino Coding

The first step in coding the Arduino was deciding which sort of algorithm to use to adjust the pulse width modulation. A Perturb and Observe (P&O) algorithm was chosen for its simplicity and compatibility with the buck converter previously designed. The algorithm is very common in trackers, and tends to work very well.

The premise of a P&O program is very simple. The voltage of the solar panel is changed incrementally either up or down, and the resulting effect on power is measured. If the power is decreased, the algorithm changes the incrimination in the opposite direction. Once the tracker reaches the maximum power peak, it will naturally oscillate around that value.

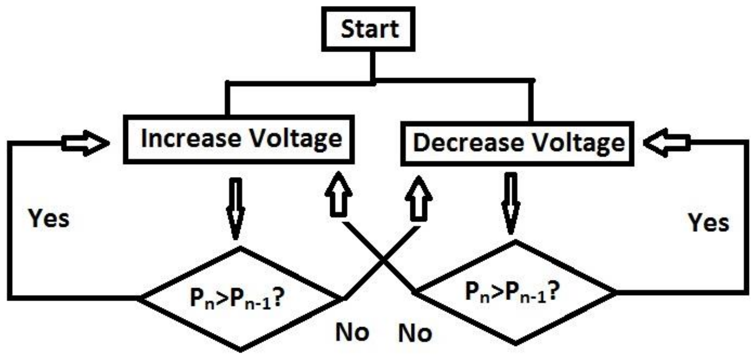

Figure 6: P&O Algorithm

Above is the block diagram for the algorithm used in this design. By sampling once a second and adjusting the PWM, it can be implemented in the rest of the circuit. Using the Arduino programming language, the algorithm was relatively easy to implement and upload. As can be seen in the next section, it simulated correctly, meaning that the P&O code worked as desired.

Another component of the coding was to implement “smart” tools to ensure maximum battery charging capabilities. Four stages of charging were implemented to accommodate different battery states. The on state was created to get the maximum value of power out of the panels when the power produced by the panels is almost low enough to not run the charger. The PWM is set to 100% to ensure the maximum power is transferred to the battery. The bulk state was created to run the MPPT algorithm. The charger runs the maximum amount of current possible out of the solar panels to transfer the maximum power possible at battery voltage by adjusting the PWM. The float state was created for when the battery reaches its maximum charge state. The tracker keeps the battery at this state by feeding a very small amount of current into the battery by adjusting the PWM. Finally, the off state was created for when the power of the solar panel is not enough to charge the battery. The charger turns off so there is no power backflow from the battery that could damage the charger.

With the Perturb and Observe algorithm and the states of the charger, the coding was completed. These two components provided for maximum charging and safety, and completed the Maximum Power Point Tracking.

## 3.6 System Simulation

The circuit was built and simulated in OrCAD PSpice. After the current sensor and Gate Driver were created, the system was run at different voltage to check the output. The desired output voltage was 13.2 V, and the input voltage ranged from 35-14 V. The circuit schematic can be seen in Figure 7, and the input waveform can be seen in Figure 8. Finally, the output waveforms can be seen in Figures 9 and 10. The circuit simulated as expected, with the output voltage staying the same while the input current and output voltage changed in a linear fashion.

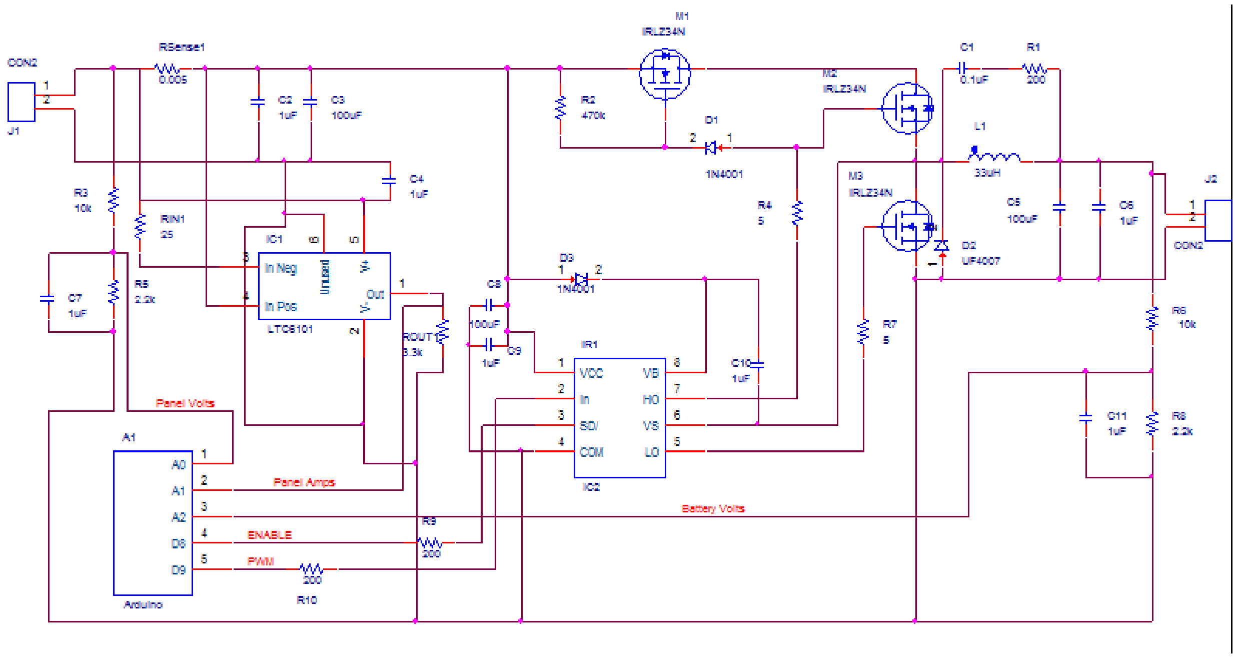

Figure 7: Full MPPT Schematic

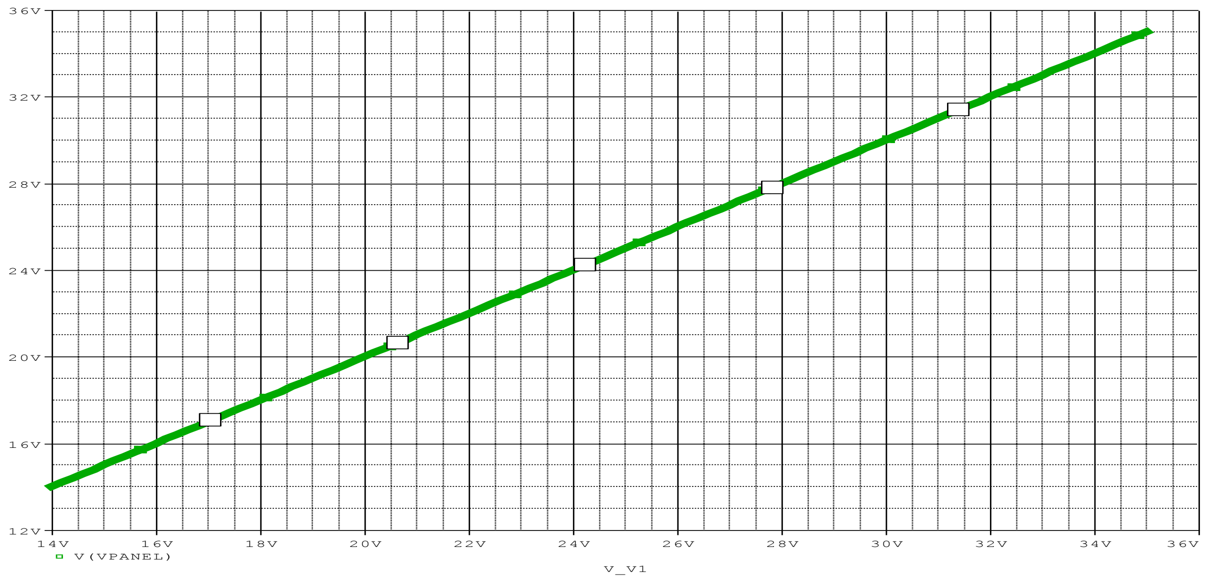

Figure 8: Panel Input Voltage

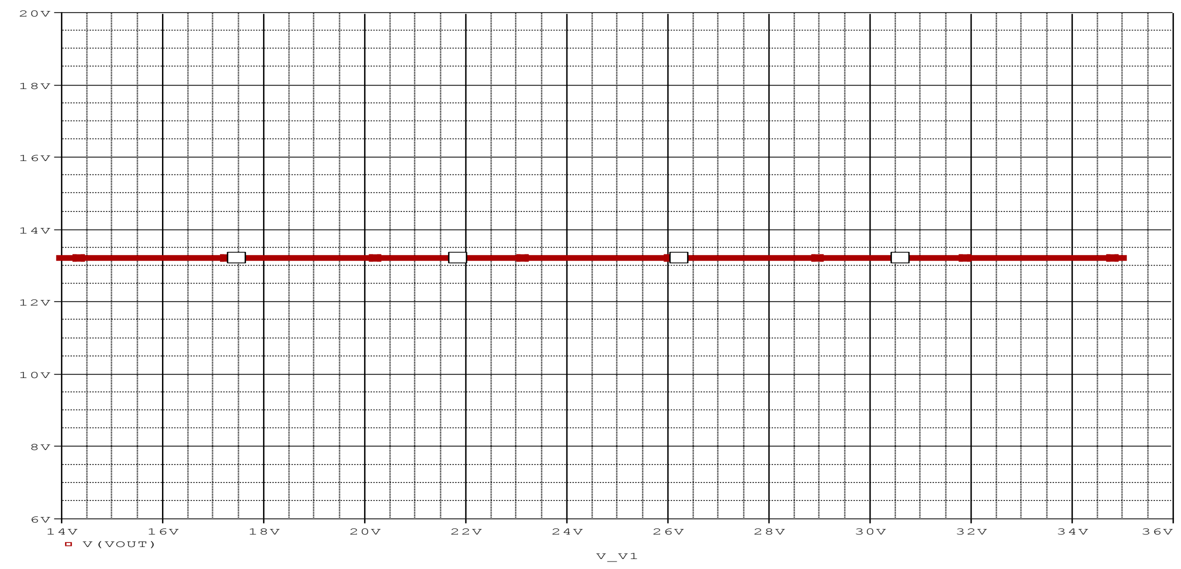

Figure 9: Output Voltage

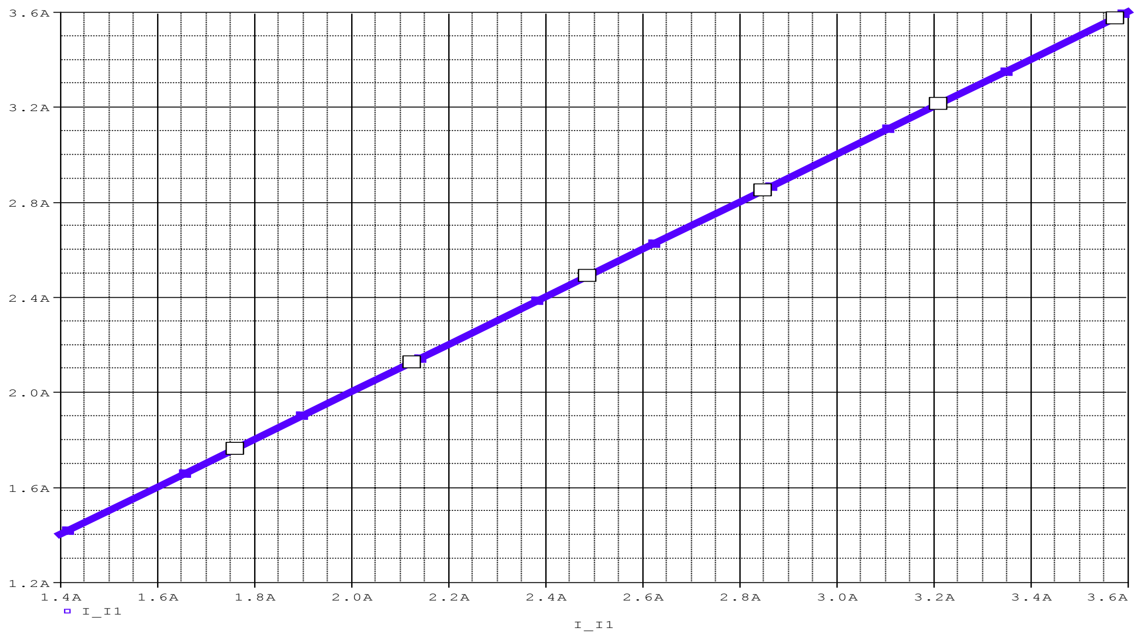

Figure 10: Output Current

# 4. PHYSICAL IMPLEMENTATION

## 4.1 Method of Implementation

In order to facilitate testing and troubleshooting of the prototype MPPT, a PCB of the circuit was created. The parts were soldered on and testing began on different parts of the tracker. To begin with, the power circuitry was built and tested. The Arduino was hooked up, and a simple blink program was used to ensure the proper output of the power circuitry was seen. Next, the load circuitry was constructed and tested. A simple duty cycle program was uploaded to the Arduino, and the output of the controller was measured against the Arduino input to ensure proper function. A load was then added to test voltage drop of adding a load. Finally, the sensing circuitry was added to the controller, and was tested by seeing if the Arduino picked up specific voltages and currents introduced into the system.

After testing the different parts of the tracker, the system was tested as a whole. The code was uploaded to the Arduino, and the solar panel and battery were hooked up to test. The current and voltage between the panel and tracker were monitored, and the output current and voltage of the tracker were also monitored. A battery charging circuit was placed between the battery and tracker to ensure that any sort of fault or mishap would not harm the tracker.

After ensuring the controller worked as desired, the process began to fabricate the circuit on a matching Arduino protoboard. The protoboard is made to function with an Arduino Uno microcontroller, and it takes up very little space in comparison to the printed circuit board previously made. The protoboard was not physically, just laid out to ensure the circuit would fit in the desired space. Once the layout was finished and the PCB was completely tested, the project was ready to be financially analyzed.

## 4.2 Results of Implementation

When testing the power circuitry, a simple square wave was desired on the oscilloscope. With the blink program in the Arduino, the high and low outputs were alternatively triggered. By connecting the oscilloscope to the Source of M2, the waveform in Figure 11 was captured, showing that the power circuitry worked as expected.

To ensure the load circuitry operated correctly, the oscilloscope was connected to the Gate of M2 and the output of the controller. The results of this can be seen in Figure 12, showing that the buck converter worked correctly. After checking the load circuitry, a load was added to see the voltage drop and the output impedance of the charger. The output voltage dropped by 0.5V, and the output impedance was measured at 0.45Ω. Using this, the Arduino code could be modified to account for the voltage drop and impedance of the controller.

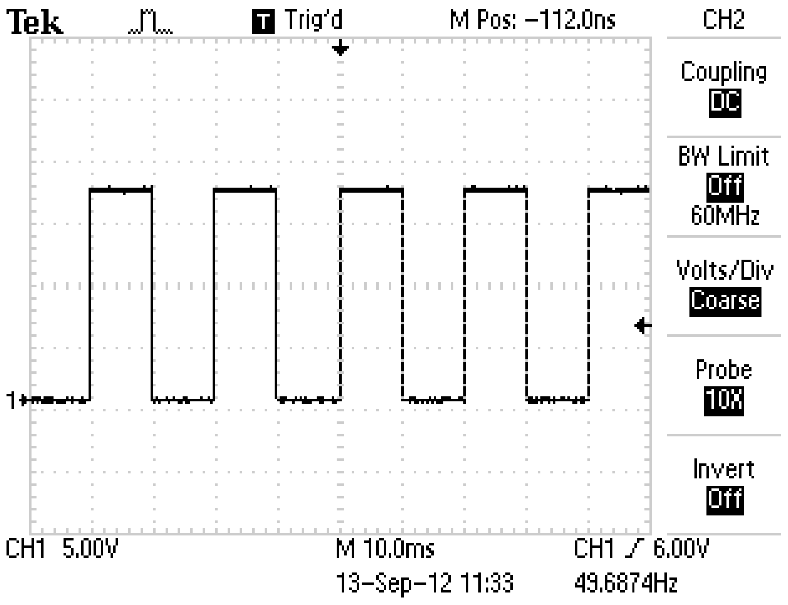

Figure 11: Power Circuitry Output

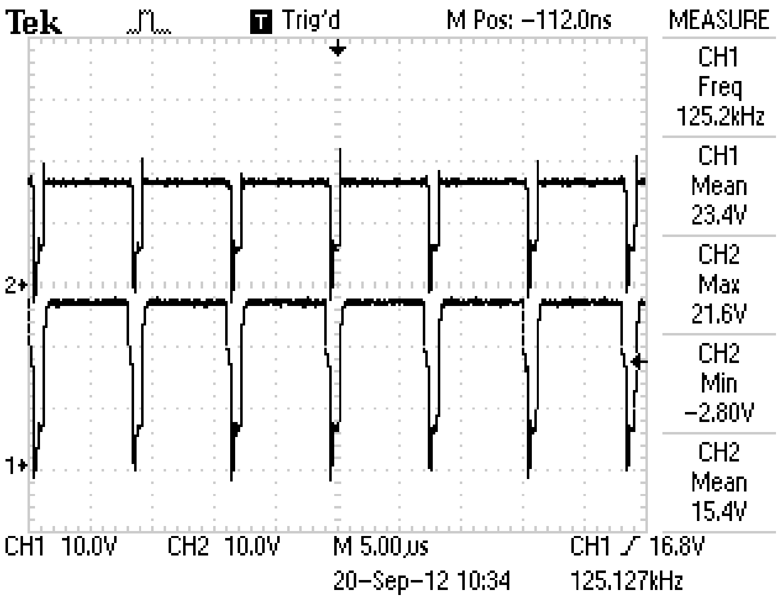

Figure 12: Load Testing and Output

Finally, the sensing circuitry was tested. In Tables 1 and 2 below, the input voltages and currents can be seen, with the Arduino readings associated with the inputs. The percent difference is also seen. Since all these percent differences were relatively low, the monitoring circuitry was deemed to work, and the charger was determined to be complete.

Table 1: Measured vs. Arduino Voltages

| Measured Input Voltage (V) | Measured Battery Voltage (V) | Arduino Input Voltage (V) | Arduino Battery Voltage (V) | Voltage Difference (% | Battery Difference (%) |
|---|---|---|---|---|---|
| 16.8 | 13.3 | 16.5 | 13.3 | 1.79 | 0 |
| 15.5 | 11.6 | 15.3 | 11.4 | 1.29 | 1.72 |
| 17.2 | 12.1 | 17.0 | 12.0 | 1.16 | 0.83 |

Table 2: Measured vs. Arduino Currents

| Measured Input Current (A) | Arduino Input Current (A) | Percent Difference (%) |
|---|---|---|
| 3.2 | 3.1 | 3.13 |
| 2.8 | 3.0 | 7.14 |
| 3.5 | 3.4 | 2.86 |

Once the charger's components were working, the entire system was tested. This was done first by connecting the charger to a power supply and the battery. To determine the charger was working, the output current and voltage of the MPPT were constantly monitored. The input current and voltage of the tracker was also monitored, and the power into and out of the system was calculated. These power calculations were used to measure the efficiency of the MPPT. A picture of the setup can be seen in Figures 12 and 13, while the results can be seen in Table 3.

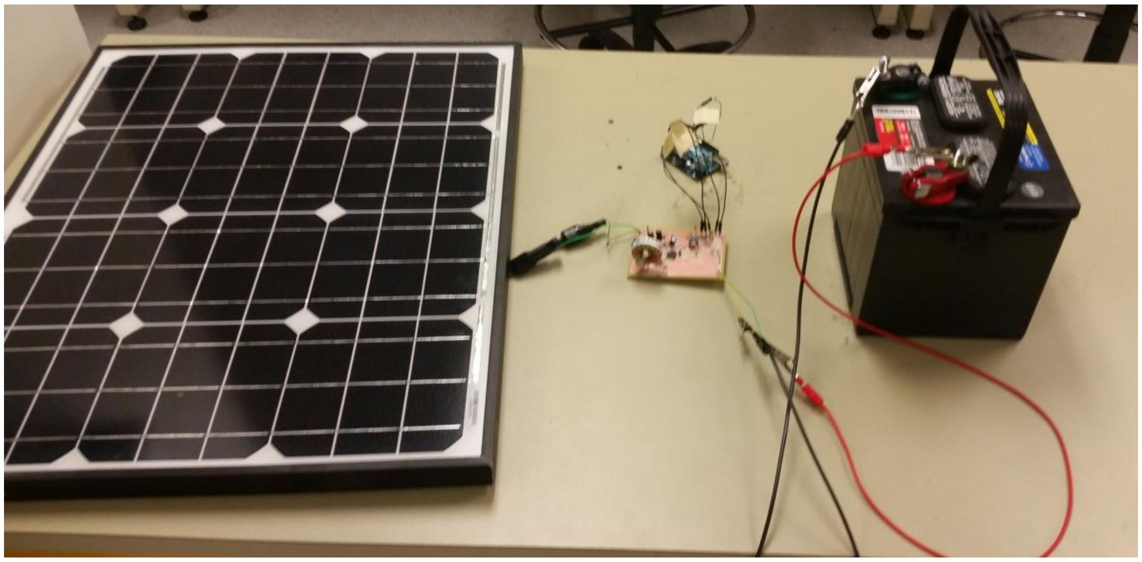

Figure 13: Full Testing Setup

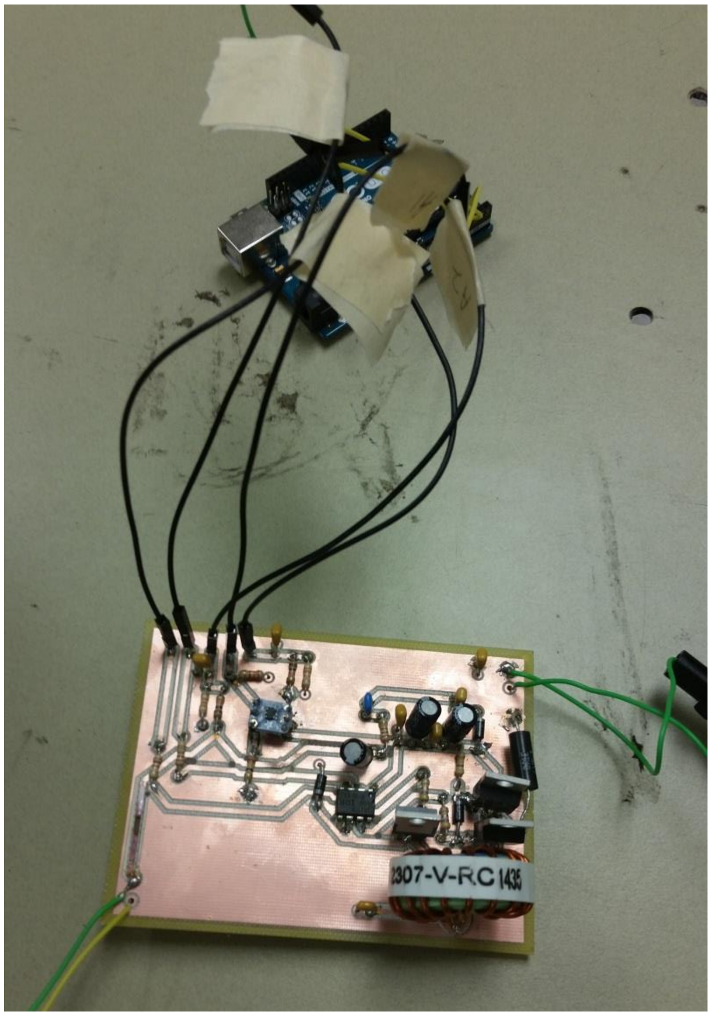

Figure 14: MPPT and Arduino Setup

This set up is just an example of the system. Most times the system was tested outside with the same connections. I picture could not be obtained while outside that was easily viewable, so a dummy version of the setup was connected inside for display purposes

Table 3: MPPT Testing Results

| Time (min) | Input Current (A) | Input Voltage (V) | Input Power (W) | Output Current (A) | Output Voltage (V) | Output Power (W) | Efficiency (%) |
|---|---|---|---|---|---|---|---|
| 1 | 0.739 | 16.830 | 12.437 | 0.958 | 13.130 | 12.579 | 101.135 |
| 2 | 0.746 | 16.310 | 12.167 | 0.931 | 13.060 | 12.159 | 99.931 |
| 3 | 0.740 | 17.220 | 12.743 | 0.920 | 13.060 | 12.015 | 94.290 |
| 4 | 0.735 | 17.040 | 12.524 | 0.888 | 13.240 | 11.757 | 93.874 |
| 5 | 0.710 | 17.510 | 12.432 | 0.907 | 13.210 | 11.981 | 96.375 |
| 6 | 0.694 | 17.630 | 12.235 | 0.856 | 13.080 | 11.196 | 91.510 |
| 7 | 0.453 | 16.980 | 7.692 | 0.563 | 13.060 | 7.353 | 95.591 |
| 8 | 0.812 | 17.160 | 13.934 | 1.021 | 13.210 | 13.487 | 96.796 |
| 9 | 0.733 | 16.850 | 12.351 | 0.911 | 13.120 | 11.952 | 96.772 |
| 10 | 0.769 | 16.920 | 13.011 | 0.933 | 13.240 | 12.353 | 94.939 |

These results prove that the charger worked as desired. With constantly changing input voltage and current, the tracker maintained a steady output voltage and maximized the power it could use. The efficiency of the tracker was astoundingly good, and changed very little over the course of the testing.

In order to test the charger to ensure the different modes were working correctly, the battery voltage and input power were measured and the mode that the charger was running in was checked. These results can be seen in Table 4. The charger worked as desired, changing states as needed to ensure proper battery charging.

Table 4: State Change Measurements

| Battery Voltage (V) | Input Power (W) | Desired State | Measured State |
|---|---|---|---|
| 10.5 | 0.8 | Off | Off |
| 11.2 | 0.5 | Off | Off |
| 10.7 | 4 | On | On |
| 12.2 | 3 | On | On |
| 12.5 | 14 | Bulk | Bulk |
| 13.1 | 34 | Bulk | Bulk |
| 12.8 | 41 | Bulk | Bulk |
| 11.2 | 29 | Bulk | Bulk |
| 11.8 | 5 | Bulk | Bulk |
| 13.6 | 37 | Float | Float |
| 13.8 | 45 | Float | Float |

# 5. FINANCIAL ANALYSIS

## 5.1 Summary

The point of this charger was to make a tracker that was financially possible. Basic components were chosen; resistors, capacitors, diodes, terminal blocks and MOSFETs were all chosen for their functionality and low cost. The inductor, current sense amplifier, and power driver are all mass produced, but a little more expensive due to their specialization. The most expensive objects were the Arduino and protoboard, but both were needed in order to create a portable and reasonably sized product. Table 5 shows specific parts used in the build. Table 6 shows the cost to build the prototype, while Table 7 shows the cost per unit if 1000 units were built. All parts were ordered on Digikey [2], and prices were current as of March 18th, 2015.

Table 5: Parts Used in MPPT

| Part | Description | Digi-Key Part Number |
|---|---|---|
| Arduino | Arduino UNO | 1050-1024-ND |
| Protoboard | Revision 3 Protosheild-Assembled | 1050-1034-ND |
| Terminal Block | Output and Input Blocks (J1, J2) | ED2703-ND |
| IC1 | Current Sense Amp (IC1) | LTC6101HVAIS5#TRPBF |
| IC2 | Power Driver IC (IC2) | IR2104PBF-ND |
| Current Sense Resistor | 2W Current Sense Resistor | 12FR005E-ND |
| 470k Resistor | R2 | CF14JT470KTR-ND |
| 10k Resistor | R3, R6 | CF14JT10KTRND |
| 3.3k Resistor | Rout | CF14JT3300KTRND |
| 2.2k Resistor | R5, R8 | CF14JT22KTRND |
| 200 Resistor | R1, R9, R10 | CF14JT200KTRND |
| 25 Resistor | Rin | CF14JT25KTRND |
| 5 Resistor | R4, R7 | CF14JT5KTRND |
| 0.1uF mono Capacitor | C1 | 478-3192-ND |
| 1uF Ceramic Capacitor | C2, C4, C6, C7, C9, C10, C11 | 478-7342-2-ND |
| 100uF Radial Capacitor | C3, C5, C8 | P12392ND |
| Sil Rec Diode | D1, D3 | 641-1310-3-ND |
| Rectifier Diode | D2 | UF4007-TPMSCT-ND |
| MOSFET | M1, M2, M3 | IRLZ34NPBF |
| Inductor- Toroid 33uH | L1 | 2100HT-330-V-RC-ND |

Table 6: Price to Build Prototype

| Part | Number Needed | Price Per Unit ($) | Total Price ($) |
|---|---|---|---|
| Arduino | 1 | 28.28 | 28.28 |
| Protoboard | 1 | 11.98 | 11.98 |
| Terminal Block | 2 | 0.24 | 0.48 |
| IC1 | 1 | 4.84 | 4.84 |
| IC2 | 1 | 2.60 | 2.60 |
| Current Sense Resistor | 1 | 2.59 | 2.59 |
| 470k Resistor | 1 | 0.01 | 0.01 |
| 10k Resistor | 2 | 0.01 | 0.02 |
| 3.3k Resistor | 1 | 0.01 | 0.01 |
| 2.2k Resistor | 2 | 0.01 | 0.02 |
| 200 Resistor | 3 | 0.01 | 0.03 |
| 25 Resistor | 1 | 0.01 | 0.01 |
| 5 Resistor | 2 | 0.01 | 0.02 |
| 0.1uF mono Capacitor | 1 | 0.35 | 0.35 |
| 1uF Ceramic Capacitor | 7 | 0.97 | 6.79 |
| 100uF Radial Capacitor | 3 | 0.45 | 1.35 |
| Sil Rec Diode | 2 | 0.02 | 0.04 |
| Rectifier Diode | 1 | 0.35 | 0.35 |
| MOSFET | 3 | 1.58 | 4.74 |
| Inductor- Toroid 33uH | 1 | 1.51 | 1.51 |
| **Total** | | | 66.02 |

Table 7: Cost per Unit for 1000 Units

| Part | Number Needed | Price Per Unit ($) | Total Price ($) |
|---|---|---|---|
| Arduino | 1 | 28.28 | 28.28 |
| Protoboard | 1 | 8.98 | 8.98 |
| Terminal Block | 2 | 0.12 | 0.12 |
| IC1 | 1 | 2.48 | 2.48 |
| IC2 | 1 | 1.10 | 1.10 |
| Current Sense Resistor | 1 | 1.14 | 1.14 |
| 470k Resistor | 1 | 0.01 | 0.01 |
| 10k Resistor | 2 | 0.01 | 0.02 |
| 3.3k Resistor | 1 | 0.01 | 0.01 |
| 2.2k Resistor | 2 | 0.01 | 0.02 |
| 200 Resistor | 3 | 0.01 | 0.03 |
| 25 Resistor | 1 | 0.01 | 0.01 |
| 5 Resistor | 2 | 0.01 | 0.02 |
| 0.1uF mono Capacitor | 1 | 0.09 | 0.09 |
| 1uF Ceramic Capacitor | 7 | 0.22 | 1.54 |
| 100uF Radial Capacitor | 3 | 0.09 | 0.27 |
| Sil Rec Diode | 2 | 0.02 | 0.04 |
| Rectifier Diode | 1 | 0.07 | 0.07 |
| MOSFET | 3 | 0.59 | 1.77 |
| Inductor- Toroid 33uH | 1 | 1.51 | 1.51 |
| **Total** | | | 47.51 |

While the prototype was slightly more expensive to build, to mass produce the charger would cost $47.51 per unit. This price is exactly $27.49 less than the desired goal of a $75 charger. Adding in taxes, fees, overhead, labor and shipping costs, the total cost of the charger would be right around $75. In comparison, the average price of an MPPT on the market is almost $100 to do the same job. The savings here can make solar power a reality in many other parts of the world that would be unable to use renewable energy due to price. The next step would be to talk to an electronics distributer to see if, by putting a kit of required parts together, the price could be reduced even further for either a single (prototype) build or for mass production. This is frequently done to great effect, and can be applied to this project.

# 6. CONCLUSION

## 6.1 Summary

This thesis focused on creating a Maximum Power Point Tracker to help encourage the use of renewable energy in the form of solar power throughout the world. The goal was to create a 50W device that could be easily understood, modified, and repaired. This was to be done at a cost of less than $75, yet would still be robust enough to handle the power and rigors of its duty. A buck converter was decided on with a perturb and observe algorithm implementing the tracking. The base of the tracker was chosen to be an Arduino, with common electronic components ordered through Digikey making up the rest of the system

The controller was first tested in OrCAD PSpice, and then built piece by piece. Testing on each separate component commenced, with the final product being tested all at once by connecting a solar panel and a battery load. Once the functionality of the tracker was proven, it was ported over to a protoboard made for the Arduino Uno. This helped ensure simplicity and portability. Finally, a financial analysis of building one tracker and mass producing the tracker was done, showing that the tracker could be built under financial restrictions.

The success of a simple and affordable charge controller has a many applications in developing and developed countries. Creating an easy to use and understand product that is cheap and easily modified encourages the use of renewable energy to save money and help the environment. In developed countries, people in their own homes can begin converting to solar power with a do it themselves attitude. They will not have to spend an exorbitant amount of money, and will save money in the long run. Projects like this will continue to help the general populace begin to save both money and the environment one small step at a time.

## APPENDIX

### A. References

[1]S.  Kalogirou, *Solar energy engineering*. Burlington, MA: Elsevier/Academic Press, 2009.

[2] Digikey.com, 'DigiKey Electronics - Electronic Components Distributor', 2015. [Online]. Available: http://digikey.com. [Accessed: 18- Mar- 2015].

[3]C.  Troutner, 'Introducing the V4 Charge Controller', *The Solar Power Expert*, 2012. .

[4]J.  Twidell and T.  Weir, *Renewable Energy Resources*. London: Spon Press, 2005.

## B. Arduino P&O Code

```
 //----------------------------------------------------------------------
//
// Arduino Peak Power Tracking Solar Code     by Chris Plate   3/1/15
Honors Thesis Spring 2015
//
//----------------------------------------------------------------------

#include "TimerOne.h"          // using Timer1 library from
http://www.arduino.cc/playground/Code/Timer1

//----------------------------------------------------------------------
// definitions

#define SOL_AMPS_CHAN 1                // read solar amps
#define SOL_VOLTS_CHAN 0               // read solar volts
#define BAT_VOLTS_CHAN 2               // read battery volts
#define AVG_NUM 8                      // number of iterations to
average readings
#define SOL_AMPS_SCALE 12              // the scaling value for raw adc
reading to get solar amps scaled by 100
#define SOL_VOLTS_SCALE 27            // the scaling value for raw adc
reading to get solar volts scaled by 100
#define BAT_VOLTS_SCALE 27            // the scaling value for raw adc
reading to get battery volts scaled by 100
#define PWM_PIN 9                    // the output pin for the pwm
#define PWM_ENABLE_PIN 8            // pin used to control the IR2104
MOSFET driver
#define PWM_FULL 1023                // Timer1 value for 100% pwm duty
cycle
#define PWM_MAX 100                  // pwm duty cyle 0-100%
#define PWM_MIN 60                  // pwm duty cyle 0-100%
#define PWM_START 90                // pwm duty cyle 0-100%
#define PWM_INC 1                    //the value the increment to the
pwm value for the algorithm
#define TRUE 1
#define FALSE 0
#define ON TRUE
#define OFF FALSE
#define TURN_ON digitalWrite(PWM_ENABLE_PIN, HIGH)      // enable
MOSFET driver
#define TURN_OFF digitalWrite(PWM_ENABLE_PIN, LOW)      // disable
MOSFET driver
#define ONE_SECOND 50000             //count for number of interrupt in
1 second on interrupt period of 20us
#define LOW_SOL_WATTS 500            //value of solar watts scaled by
100 so this is 5.00 watts
#define MIN_SOL_WATTS 100            //value of solar watts scaled by
100 so this is 1.00 watts
#define MIN_BAT_VOLTS 1100          //value of battery voltage scaled
by 100 so this is 11.00 volts
```

```
#define MAX_BAT_VOLTS 1410          //value of battery voltage scaled
by 100 so this is 14.10 volts
#define HIGH_BAT_VOLTS 1300          //value of battery voltage scaled
by 100 so this is 13.00 volts
#define OFF_NUM 10                  //number of iterations of off
charger state

//------------------------------------------------------------------------
// global variables

int count = 0;
int pwm = 0;                          // pwm duty cycle 0-100%
int sol_amps;                         // solar amps scaled by 100
int sol_volts;                        // solar volts scaled by 100
int bat_volts;                        // battery volts scaled by 100
int sol_watts;                        // solar watts scaled by 100
int old_sol_watts = 0;                // solar watts from previous time
through ppt routine scaled by 100
unsigned int seconds = 0;             // seconds from timer routine
unsigned int prev_seconds = 0;        // seconds value from previous
pass
unsigned int interrupt_counter = 0;    // counter for 20us interrrupt
int delta = PWM_INC;                  // variable used to modify pwm
duty cycle for the ppt algorithm

enum charger_mode {off, on, bulk, bat_float} charger_state;    //
enumerated variable, holds state for charger state machine

//------------------------------------------------------------------------
// Powerup/Reset Routine
//------------------------------------------------------------------------
void setup()                            // run once, when the sketch
starts
{
  pinMode(PWM_ENABLE_PIN, OUTPUT);     // sets the digital pin as
output
  Timer1.initialize(20);               // initialize timer1, and set a
20uS period
  Timer1.pwm(PWM_PIN, 0);              // setup pwm on pin 9, 0% duty
cycle
  TURN_ON;                             //turn on MOSFET driver chip
  Timer1.attachInterrupt(callback);    // attaches callback() as a
timer overflow interrupt
  Serial.begin(9600);                  // open the serial port at 38400
bps:
  pwm = PWM_START;                     //starting value for pwm
  charger_state = on;                  // start with charger state as
on
}
//------------------------------------------------------------------------
// This is interrupt service routine for Timer1 that occurs every 20uS.
//------------------------------------------------------------------------
void callback()
{
  if (interrupt_counter++ > ONE_SECOND) {        //increment
interrupt_counter until one second has passed
    interrupt_counter = 0;
    seconds++;                                    //then increment
seconds counter
  }
}
//----------------------------------------------------------------------
// Reads and averages the analog inputs.
//----------------------------------------------------------------------
int read_adc(int channel){

  int sum = 0;
  int temp;
  int i;

  for (i=0; i<AVG_NUM; i++) {            // loop through reading raw
adc values AVG_NUM number of times
    temp = analogRead(channel);          // read the input pin
    sum += temp;                        // store sum for averaging
    delayMicroseconds(50);              // pauses for 50 microseconds
  }
  return(sum / AVG_NUM);                // divide sum by AVG_NUM to get
average and return it
}
//----------------------------------------------------------------------
// This routine uses Timer1.pwm function to set the pwm duty cycle.
//----------------------------------------------------------------------
void set_pwm_duty(void) {

  if (pwm > PWM_MAX) {                                 // check limits of PWM
duty cyle and set to PWM_MAX
    pwm = PWM_MAX;
  }
  else if (pwm < PWM_MIN) {                            // if pwm is less than
PWM_MIN then set it to PWM_MIN
    pwm = PWM_MIN;
  }
  if (pwm < PWM_MAX) {
    Timer1.pwm(PWM_PIN,(PWM_FULL * (long)pwm / 100), 20); // use Timer1
routine to set pwm duty cycle at 20uS period
  }
  else if (pwm == PWM_MAX) {                           // if pwm set to 100%
it will be on full
    Timer1.pwm(PWM_PIN,(PWM_FULL - 1), 1000);          // keep
switching so set duty cycle at 99.9% and slow down to 1000uS period
  }
}

//----------------------------------------------------------------------
// Puts all values into real values instead of scaled values
//----------------------------------------------------------------------
void read_data(void) {

  sol_amps =  ((read_adc(SOL_AMPS_CHAN)  * SOL_AMPS_SCALE) + 5) / 10;
//input of solar amps result scaled by 100
  sol_volts = ((read_adc(SOL_VOLTS_CHAN) * SOL_VOLTS_SCALE) + 5) / 10;
//input of solar volts result scaled by 100
  bat_volts = ((read_adc(BAT_VOLTS_CHAN) * BAT_VOLTS_SCALE) + 5) / 10;
//input of battery volts result scaled by 100
```

```
  sol_watts = (int)((((long)sol_amps * (long)sol_volts) + 50) / 100);
//calculations of solar watts scaled by 10000 divide by 100 to get
scaled by 100
}
//----------------------------------------------------------------------
// This is the charger state machine.
// Four states: On, Off, Bulk and Float.
//
//  On - Happens when the solar watts are less than the watts required
to charge but greater than the minimum watts
//      needed to keep the charger on. In this state we just set the
pwm = 100% to get the most of the power available
//
//  Bulk - Happens when the solar watts are greater than the minimum
needed watts.Where the Peak Power Tracking algorithm
//      is run. Runs the maximum amount of current possible into the
battery.
//
//  Float - Happens when the battery's voltage reaches its maximum
level.Keeps the battery voltage at max level by
//      adjusting the pwm value.
//
//  Off - Happens when the panel is not providing enough power to the
charger. The MOSFETs are turned
//      off in this state so that power from the battery doesn't leak
back into the solar panel.
//----------------------------------------------------------------------
void run_charger(void) {

  static int off_count = OFF_NUM;

  switch (charger_state) {
    case on:
      if (sol_watts < MIN_SOL_WATTS) {              //If watts input
from the solar panel is less than
        charger_state = off;                        //the minimum solar
watts go to off state
        off_count = OFF_NUM;
        TURN_OFF;
      }
      else if (bat_volts > MAX_BAT_VOLTS) {        //If the battery
voltage has gotten above the float
        charger_state = bat_float;                 //go to the charger
battery float state
      }
      else if (sol_watts < LOW_SOL_WATTS) {        //If the solar input
watts is less than low solar watts
        pwm = PWM_MAX;                             //go to on state
        set_pwm_duty();
      }
      else {
        pwm = ((bat_volts * 10) / (sol_volts / 10)) + 5;  //If there is
more power than low solar watts find0 the pwm
        charger_state = bulk;                             //value
should be and change the charger to bulk state
      }
      break;
    case bulk:
      if (sol_watts < MIN_SOL_WATTS) {
        charger_state = off;                        //If watts input
from the solar panel is less than
        off_count = OFF_NUM;                        //the minimum solar
watts go to off state
        TURN_OFF;
      }
      else if (bat_volts > MAX_BAT_VOLTS) {         //If the battery
voltage has gotten above the float
        charger_state = bat_float;                  //go to the charger
battery float state
      }
      else if (sol_watts < LOW_SOL_WATTS) {        //If the solar input
watts is less than low solar watts
        charger_state = on;                        //go to on state
        TURN_ON;
      }
      else {                                      // Begin MPPT
Algorithm
        if (old_sol_watts >= sol_watts) {         //  if previous watts
are greater change the value of
          delta = -delta;                  // delta to make pwm increase
or decrease to maximize watts
        }
        pwm += delta;                            // add delta to change
PWM duty cycle for PPT algorythm
        old_sol_watts = sol_watts;               // load old_watts with
current watts value for next time
        set_pwm_duty();                    // set pwm duty cycle to pwm
value
      }
      break;
    case bat_float:
      if (sol_watts < MIN_SOL_WATTS) {           //If watts input from
the solar panel is less than
        charger_state = off;                     //the minimum solar
watts go to off stat
        off_count = OFF_NUM;
        set_pwm_duty();
        TURN_OFF;
      }
      else if (bat_volts > MAX_BAT_VOLTS) {      //If the battery
voltage is above the float voltage
        pwm -= 1;                                //lower the pwm to
lower voltage
        set_pwm_duty();
      }
      else if (bat_volts < MAX_BAT_VOLTS) {    //else if the battery
voltage is less than the float voltage
        pwm += 1;                               //increment the pwm to
get it back up to the float voltage
        set_pwm_duty();
        if (pwm >= 100) {                       //If pwm gets up to
100%,  battery cannot keep at float
          charger_state = bulk;                 //Go to charger bulk
state to charge the battery
```

```
          }
        }
        break;
      case off:                                  // When going to off
state, off_count is set with OFF_NUM
        if (off_count > 0) {                     //Iterate off 10 times
          off_count--;
        }
        else if ((bat_volts > HIGH_BAT_VOLTS) && (bat_volts <
MAX_BAT_VOLTS) && (sol_volts > bat_volts)) {
          charger_state = bat_float;              //If battery voltage
and solar volts are high
          set_pwm_duty();                        //change charger state to
battery float
          TURN_ON;
        }
        else if ((bat_volts > MIN_BAT_VOLTS) && (bat_volts <
MAX_BAT_VOLTS) && (sol_volts > bat_volts)) {
          pwm = PWM_START;                        //if battery volts
aren't quite high enough but solar volts
          set_pwm_duty();                    //are greater than battery
volts then
          charger_state = on;                     //change charger state
to on
          TURN_ON;
        }                                          //else stay in the off
state
        break;
      default:
        TURN_OFF;
        break;
    }
  }
//----------------------------------------------------------------------
// Main loop.
//----------------------------------------------------------------------
void loop()
{
  read_data();                          //read data from inputs
  run_charger();                       //run the charger state machine
  if ((seconds - prev_seconds) > 0)
  {
    prev_seconds = seconds;         // do this stuff once a second
    read_data();                          //read data from inputs
    run_charger();
  }
}
//----------------------------------------------------------------------
```