HYBRID'S ANGELS

Final Report

Hybrid Electric Motorcycle

Stephen Govea;Eric Schafer;Jean Whitney;Ravi Fernando;Jeffrey F. Wolanin

4/30/2009

Table of Contents

1 INTRODUCTION

From August 2007 to May 2008, a group of senior electrical engineering majors at the University of Notre Dame began construction on a series hybrid electric motorcycle. Over the course of the year, they successfully converted a 1983 Yamaha Seca into a battery powered vehicle; however, they were unable to meet the ambitious goal of mechanical power system hybridization. In this improved design, Hybrid's Angels have realized this goal in addition to fixing and enhancing current issues with the motorcycle's previous design.

1.1 PROBLEM STATEMENT

The phrase "going green" has gained widespread popularity in recent years. As uncertainty mounts about the effects of carbon emissions on the earth's future, increased pressure is falling upon individuals around the world to reduce their carbon footprints. This can be accomplished in a wide variety of ways from an increased devotion to recycling to simply turning off a light when not in use. Energy conservation and decreased emissions are more important now than ever because evidence of the negative effect of the world's hitherto wasteful nature is finally beginning to manifest itself. One of the major sources of this waste is a product that has become fundamental to Americans' daily activities – the automobile. Automobiles have been a staple of transportation for decades, and the carbon dioxide and carbon monoxide that they billow through their exhausts is a colossal part of the problem. A great deal of stress has been placed upon auto manufacturers to trim their vehicles' emissions, and thus far the most popular response has been the introduction of the hybrid vehicle. Although certainly not the solution to all of the energy ills, hybrid technology will act as a critical transition technology until cleaner, more efficient sources are implemented.

1.2 PROPOSED SOLUTION

While hybrid vehicles are not completely independent of fossil fuels, they are much more "green" than gasoline powered engines. The electricity used to power the batteries in a hybrid is likely to come from a power plant that burns fossil fuels, but even that situation is much more efficient and environmentally friendly than the combustion engine of a car. In addition, a hybrid vehicle will not produce any emissions while running off the batteries. So why not just build an electric vehicle? Range and charging time of current electric automobile technology cannot compete with traditional gasoline powered engines used in most cars. A hybrid is therefore the best solution because it is a compromise between current gasoline powered automobiles and the emission-free vehicles of the future.

Last year, a group of senior electrical engineering majors, the Lightning Riders, built a prototype electric motorcycle. For this group's project, they have modified and improved the electric motorcycle built by the Lightning Riders by turning it into a hybrid motorcycle.

1.3 SYSTEM DESCRIPTION

Hybrid's Angels' hybrid motorcycle utilizes a series hybrid configuration. This configuration begins with power from a standard 120VAC wall outlet being sent into a specially designed circuit. This circuit will be created to convert the AC waveform into a DC signal of approximately 90V. This 90V of direct current

will be used to charge the 72V stack of batteries. When the batteries are fully charged, the group will take the wall input plug and insert it into the generator.

The generator produces the same waveform as the wall output but it is mobile because it runs off of gasoline power. Therefore, we can use the same charging circuit to charge the batteries while the motorcycle is running. Ideally the generator will be electric start, meaning that we would be able to start it via a signal from the microcontroller. In order for this functionality however, the generator would also have to have an electric, or choke-less, start. Due to financial constraints, this proved not to be possible. Therefore the generator will have to be manually started during operation.

In operational mode, the batteries will be used to power two things. First and foremost, the 72V battery stack will drive the analog motor controller. This motor controller drives the electric motor, which in turn rotates the wheel via a gear shaft. It is operated by the bike's throttle. This entire system was put in place by the Lightning Riders and will not be our biggest concern. Since the original design suffered a slight flaw when the electronic circuitry was being powered by two batteries in the stack, the group will need to explore a more efficient way to power these onboard electronics, which include the microcontroller, LED display, sensors, and other necessary onboard electronic components.

Of more interest to the group will be the electronic system. There will be a system of sensors placed all over the bike – temperature on the batteries, current from the batteries, and voltage remaining on the batteries. All of these data will be fed into the microcontroller. Based on these data, the control module will make real-time decisions about ways to increase the bike's energy efficiency. A monitoring module will also receive the data from the sensors, but its primary focus is to store that data for future analysis.

1.4 SYSTEM REQUIREMENTS

1.4.1 OVERALL SYSTEM

Hybrid's Angels' overarching goal was to make a working hybrid motorcycle. Still, the group intended to fix and improve some of the features, which were implemented to varying degrees of success in the Lightning Riders' model. In May, Hybrid's Angels expect to demonstrate a fully functional hybrid motorcycle with top speeds of at least 50 miles per hour and a range of at least 20 miles before refueling and/or recharging.

1.4.2 SUBSYSTEM REQUIREMENTS

1.4.2.1 GENERATOR

The generator has to be gas-powered and mobile. Hybrid's Angels decided to use a single generator mounted on the back rather than the dual-generator saddle-bag configuration, so size and balance are of the utmost importance. In addition, the group's initial goal was to use an electric start generator; this was so that the microcontroller can send a signal to start the gas engine inside. After further investigation, the group decided that implementing an electric choke in order to start the engine would

be too costly, both financially and time-wise. Therefore although the generator is electric start, it must be turned on manually. Finally, it has to output 120VAC, and the group will have to be able to draw a continuous 20 amps in order to charge the batteries to the ideal specifications.

1.4.2.2 CHARGING CIRCUITRY

Ultimately, the overlying goal in the creation of the charging circuitry was to develop a system that charges the 72V battery stack as quickly as possible while maintaining an appropriate level of safety. Appropriate measures were taken not only to protect the user but also the integrity of the batteries. As a goal of this project is to show that a series hybrid system is a ready, viable solution, the charging circuitry must interface with a common electrical socket (120 VAC, 60 Hz). While initially it was thought to double the circuitry in order to maximize power and efficiency, testing proved this unnecessary resulting in a single charging system.

From the outlet, a mechanism, such as a fuse and or transformer, must be present so as to prevent the high voltage AC signal from destroying the circuitry. This circuitry then required conversion circuitry to transform the AC signal to a stable DC one which can ultimately be used to charge the 72V battery stack. It would have been nice if the group could only output one DC voltage as this would have greatly simplified the process; however, the group chose to design a circuit that can output variable levels of voltage and current to the battery stack. The group's desire to create a quick and efficient charge necessitated this more complicated requirement. As a result, it is imperative to have a way to receive feedback from the battery stack and make a decision based on the temperature, voltage, and or current in the battery. Thus, the charging circuitry required a way to interface with the microcontroller, which reads the appropriate sensors at a given interval, in order to make the correct decisions in a timely manner.

Charging the 72V battery stack is not the only on board device that the group must power. The design must also power onboard electronic circuitry on the motorcycle—LED display, microcontroller, etc. As last year's group experienced some issues with the battery system being heavily taxed by the onboard electronics, a separate battery was added instead of drawing the power from the 72V stack, which serves as the primary source of energy to the motorcycle's motor.

1.4.2.3 CONTROL SYSTEM

For this project, the control system is the brains behind the operation. It is extremely important to write robust, functional code as the microcontrollers has to shoulder many important tasks without user intervention. First, it should be able to process the incoming data from various onboard sensors, which include temperature sensors, voltage sensors, and current sensors. Based on the information that is read into the system, this design must give feedback to the appropriately adjust the DC voltage and or current into the 72V battery stack so that the batteries will charge efficiently. Besides making decisions, it must inform the user of the state of the vehicle via the LED display. Of course, this design must not only be able to make real time decisions and display them but it also must be able to output formatted data and save it to an external storage for later analysis and use.

1.4.2.4 MONITORING SYSTEM

1.4.2.4.1 DATA COLLECTION SUBSYSTEM

The finished product has a number of sensors at key locations to measure critical data points such as voltages and currents. Some of these are vital for successful real-time operation while others are of more interest in on a historical basis. The data collection subsystem contains appropriate hardware and software to periodically sample the data from the onboard sensors and record it in an onboard storage device for future retrieval. Both the hardware and software have specific requirements in this subsystem:

Hardware – it must be capable of capturing both analog and digital signals; many of the sensors generate an analog signal between 0 and 5 volts. In addition, it must sample all data inputs quickly to maintain a high sampling rate while not tying up the system resources for extended periods of time. In the same way, the storage scheme must be simple and swift to execute, once again to avoid tying up the system processor during real-time operation. Finally, the subsystem storage capacity needs to be of sufficient size to hold data from a reasonable length ride.

Software – while important, data collection must be a secondary concern to safe and efficient real-time operation. The data collection routines should seamlessly integrate with the critical software functions and should not significantly affect the overall system speed. Wherever possible the system should use standard communication protocols that are easy to understand and debug, such as SPI. Finally, the storage scheme should use the available storage space efficiently while avoiding an extremely complicated methodology.

1.4.2.4.2 ANALYSIS SUBSYSTEM

Data is not much use until it is transformed into useful information, which is accomplished in the analysis subsystem. Data is transferred from the onboard storage system to a PC where statistical and graphical analysis can take place. There are both hardware and software components to this subsystem:

Hardware – primary component is the physical data link between the microcontroller and the PC. Ideally this would be a wireless data link but a hardware backup is in place to ensure a reliable connection. The data link is based on the standard RS-232 protocol

Software – there are two software components to this subsystem. First, on the microcontroller end the software must interface with the storage system and retrieve all the data in an organized fashion and send it through the data link. In addition, the microcontroller software must communicate back and forth with the PC based software to reliably transfer the data. On the PC side, the software must capture the incoming data, store it in more permanent hard drive based storage (database) and enable the user to perform statistical and graphical analysis.

1.4.3 FUTURE ENHANCEMENT REQUIREMENTS

Although of minor importance, the Hybrid's Angels would like to suggest a few features to either enhance the safety or mission of the hybrid motorcycle. First and foremost, the group would like to

implement a functional headlight, which the user could turn on or off from the control console. To make the design fully street legal, it will also be necessary to install front brakes, a taillight, brake lights, mirrors, and a horn. These devices will require a neat, orderly connection and wiring scheme. Another additional feature would be to digitize the motorcycle's speedometer instead of the analog sensor that is currently present. In order to further enhance the operation safety, the Lightning Riders indicated that it may be useful to create a physical disconnect on the battery stack, allowing for safer maintenance.

In the spirit of energy economy, the group would also like to suggest an alternative energy source to power one of the electronic components. Preliminary thoughts on such an implementation would include utilizing solar technology or better harnessing the energy from the existing regenerative braking system. Incorporating an alternative source will be no small challenge; it will require a proper design to achieve the necessary ratings to power the given electronic device as well as a storage scheme when peak conditions are not present.

Last year, team member Steve Govea received a wireless module that simulates an RS232 connection with a computer. Since this part is already present, he intends to implement this part when the group designs its board for the project. Although a nice feature, this part is certainly redundant as the group will already be able to program and download information from the system via a USB to serial converter, which is described below.

1.5 HIGH LEVEL SOLUTION DESCRIPTION

In order to solve the problem described above, Hybrid's Angels have continued to pursue the series hybrid configuration first established by the Lightning Riders. Although the general hybrid concept has remained consistent, the exact solution varies widely from that attempted last year. As a general overview, the final design contains three major subsystems: the control system, the charging system, and the monitoring system. A high level description of each is described below.

1.5.1 CONTROL CIRCUIT SYSTEM

For this project, the control system is the brains behind the operation. It was extremely important to write robust, functional code as the microcontroller had to shoulder many important tasks without user intervention. First, it was able to quickly and accurately process the incoming data from various onboard sensors, which include temperature sensors, voltage sensors, and current sensors. Besides making decisions, it informed the user of the state of the vehicle via the LCD display. Of course, this design is also able to store information for later analysis in design problems.

1.5.2 CHARGING CIRCUIT SYSTEM

The charging circuitry is responsible for the primary conversion of the generator voltage to a usable DC input to charge the batteries. The circuitry itself is composed of three smaller circuits: triac circuitry, the buck converter, and a pi filter.

The triac circuitry functions to reduce the AC voltage coming from the battery. In essence, the functionality is similar to that of a digital AC dimmer switch used for common home lighting applications. A zero-cross detection circuit is implemented to determine when the AC waveform crosses the 0V mark during each cycle. This information is then sent to the triac circuit's microcontroller. Software decisions are then made to determine when to fire the triac. This will delay the signal that is passed by the circuit, thereby rectifying the original signal from the generator. Visually, the idea can be seen as follows in Figure 1.5.2.

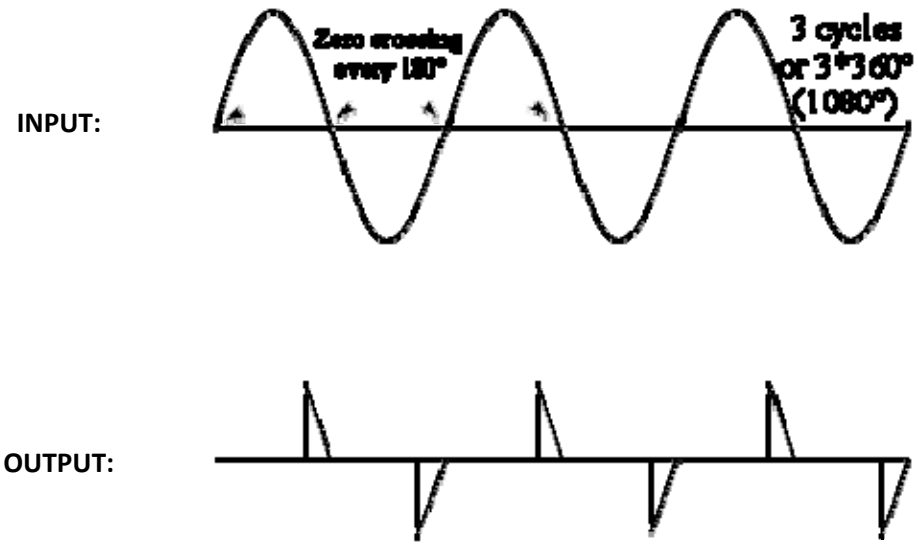

Figure 1.5.2: AC Wave Dimming

This dimmed signal that is passed is then rectified using a full bridge rectifier. The result is a DC voltage that can be varied by varying the firing time of the triac, controlled via the microcontroller. In essence, this triac circuitry functions as a variable AC-DC converter.

1.6 REVIEW OF FINAL RESULTS

The final result of the series hybrid design employed by Hybrid's Angels proved successful. It was demonstrated that the batteries could be charged using the implemented circuitry. While charging at 8A was observed for a short period of time, the group maintained a charging current of 3-4A continuously to charge the batteries from a very low charge state to maximum charge.

Additionally, successful control of the charging process was observed. The control circuitry effectively monitored the voltage and temperature of each battery. An effective charging algorithm was implemented using this feedback information to charge the battery stack the most efficiently.

User interfaces were also observed to function properly. A user-friendly output was programmed to the LCD making monitoring and control relatively easy for the operator. Also, a functioning LCD backlight was implemented in order to make the screen more visible.

A final point to note was the success in mounting the generator itself. Given the size and weight of this piece of equipment, mounting proved to be no small task. An effective support system was developed to secure the generator behind the rear wheel of the motorcycle. This process was successfully completed, making the motorcycle a completely self-contained hybridized unit.

2 DETAILED PROJECT DESCRIPTION

2.1 SYSTEM THEORY OF OPERATION

The Hybrid's Angels were able to successfully design and build a relatively sophisticated hybrid-electric motorcycle. This vehicle employs high-power circuitry to charge the battery stack, and this charging is acutely controlled with a custom algorithm written by Hybrid's Angels for this express purpose.

At first glance, purely electric vehicles seem devoid of flaws. They are noise-free, emissions free, and it is drastically cheaper to produce a unit of electric energy for vehicle use that it is to produce an equivalent unit of liquid fuel. However, electric vehicles do carry many problems, the majority of which stem from their batteries.

Current battery technology is the major limiting factor in electric vehicle design. Electric motors with efficiencies upwards of 95% have been invented, and ultra-capacitors that store charge from the batteries are under development, but problems still plague the batteries themselves. One main issue is time to charge – many batteries take many hours to charge now. Another main issue is the life cycle of batteries, or the number of deep discharges that a battery can sustain before it is unusable. New batteries are being developed that are starting to really address this problem, but in the meantime they are much too expensive.

The theory behind hybrid-electric vehicles was created around the idea that batteries are still too primitive to be used in day-to-day applications. This would allow vehicles to use the best features of internal combustion engines in conjunction with the best features of electric motors. In particular, the series hybrid architecture that the Hybrid's Angels' motorcycle utilizes the fact that internal combustion engines operate at their peak efficiency in a small rpm band.

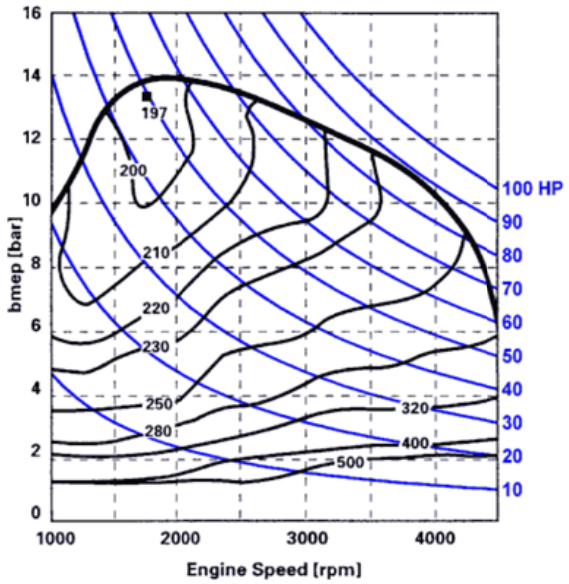

Figure 2.1: Brake-Specific Fuel Consumption Curve for a Typical IC Engine

As shown in the above chart, the sample internal combustion (IC) engine has its peak efficiency, measured by its brake-specific fuel consumption rate of 197, at a single output power for a single rpm. Varying the rpm just a little bit along that power curve severely changes the efficiency of the engine. The primary benefit of the series-hybrid architecture is that it allows the IC engine to operate at that peak efficiency for the entire duration of operation. The IC engine is used to charge the batteries and prevent the sorts of deep discharges that are so damaging and harmful to their lifetimes.

This is the general theory behind the motorcycle's design. The generator is meant to be run at around its peak level in order to charge the batteries and improve the vehicle's run time. The charging is completed in two stages – the triac circuit and the buck converter.

The triac circuit achieves AC-to-DC conversion and its theory is quite simple. A standard 120Vrms signal from the wall is fed into the circuit as the input. In order to output a desired DC value from this waveform, the sine wave is cut off when it reaches said value. This creates a saw-tooth type signal which is fed into a smoothing capacitor. The buck converter, on the other hand, achieves DC-to-DC conversion. This is done with a switching MOSFET which continuously charges and discharges a capacitor through an inductor. The varying current through the inductor creates a voltage drop which causes the original DC voltage to be stepped down. Both of these systems will be explained more fully with diagrams later.

The theory behind the user interface (UI) is also very straightforward. The control circuit mounted on the motorcycle features a microcontroller that constantly gathers data such as temperature, voltage and current on the batteries. This information is fed in real time to an LCD screen which is featured prominently between the bike's handlebars.

2.2 SYSTEM BLOCK DIAGRAM

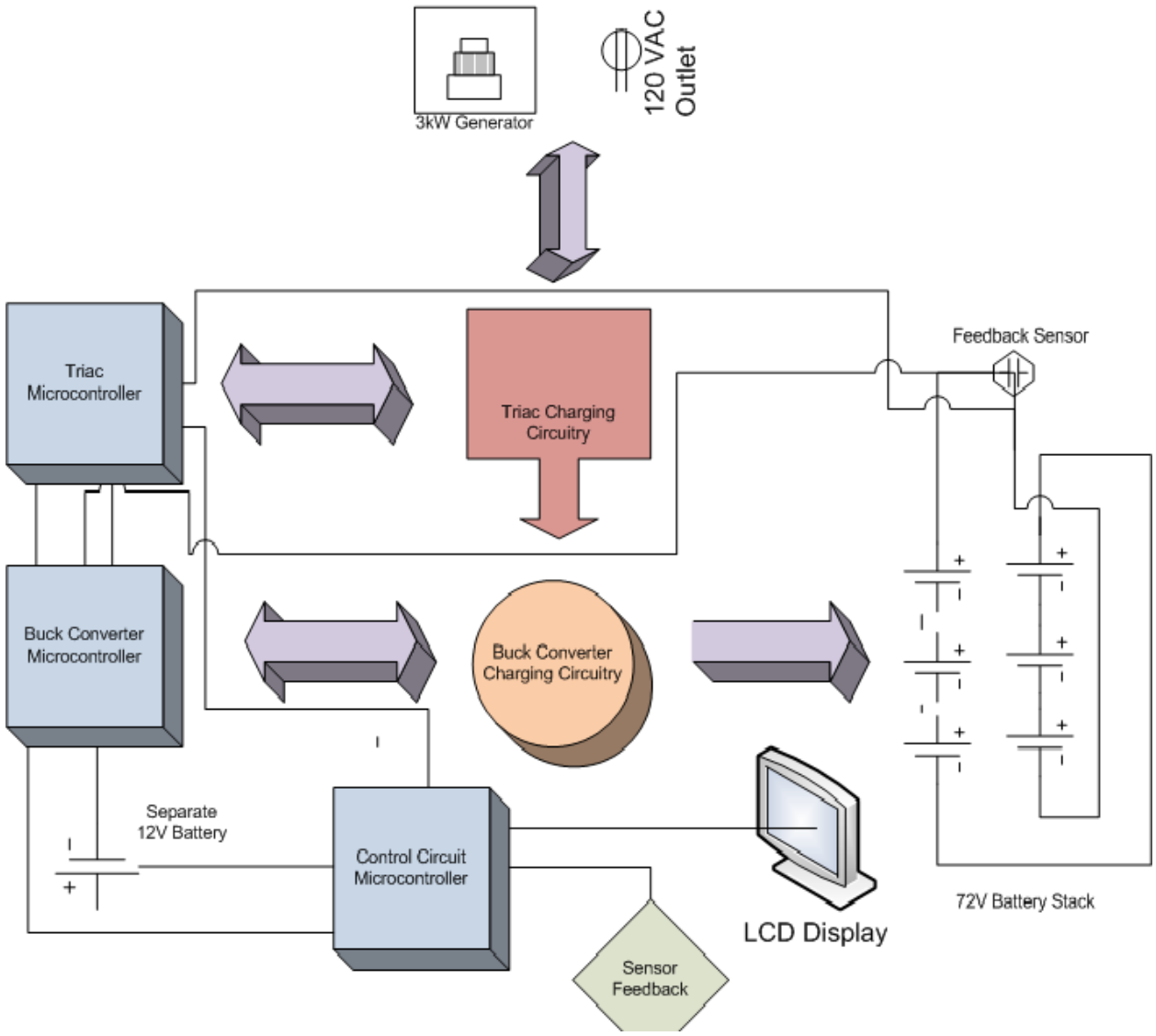

Figure 2.2: System Block Diagram

2.3 DETAILED OPERATION OF CHARGING CIRCUITRY

The function of the control circuitry at the highest level is to convert the signal from the generator or a wall socket to a usable input into the 72V battery stack for charging. This is accomplished by first using a triac circuitry connected to the generator, followed by a buck converter. These two subsystems are controlled via their own microcontroller and act separately. In essence, the triac circuit functions as a variable digital AC-DC converter. The buck converter, by convention, functions as a DC-DC converter.

2.3.1 TRIAC CIRCUIT SYSTEM

The triac circuit functions as an AC-DC converter. The subsystem itself is composed of a zero cross detection circuit, a separate microcontroller, and the triac circuit itself. These systems are explained in detail below.

2.3.1.1 OVERALL SCHEMATIC

The diagram below shows the overall hardware schematic of the triac subsystem as well as some additional features necessary for the circuit board design. Each portion of the circuitry will be described below.

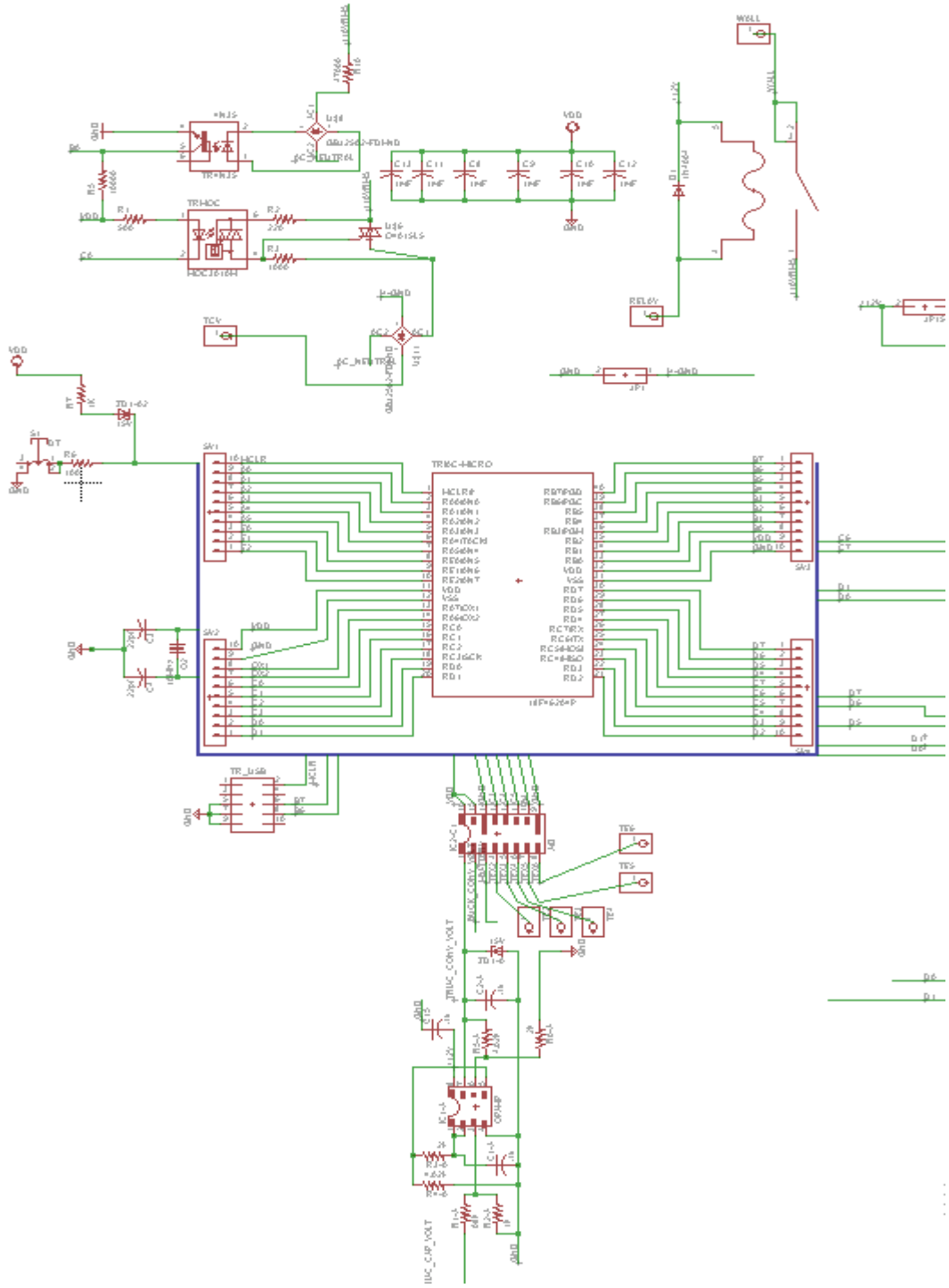

Figure 2.3.1.1: Triac Circuit System

2.3.1.2 ZERO-CROSS DETECTION CIRCUIT

Pictured in the top left of the main schematic is the zero-cross detection circuitry. The role of this circuit is to sense when the incoming AC signal has crossed the 0V mark during each AC cycle. The circuit is reproduced below for convenience.

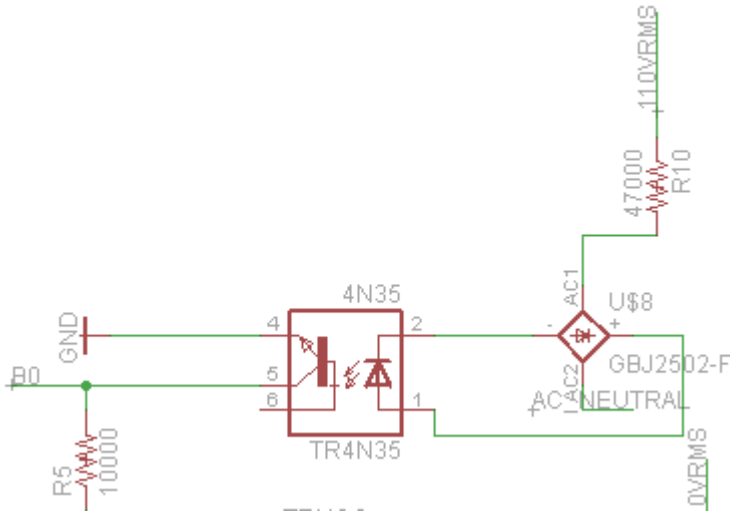

Figure 2.3.1.2: Zero-Cross Detection Circuit

The zero-cross detection circuit functions as follows. The AC voltage is passed through a full bridge rectifier, as pictured on the right of the schematic. Then the output of the rectifier is applied to an optocoupler. The optocoupler consists of an LED on the input side and a BJT on the across the output terminals. When current is flowing across the LED, the emitted light turns on the BJT. Since the LED will be conducting current any time the rectified signal is above 0V, the BJT will always be conducting. In this situation, the input to the microcontroller (B0) will be low.

When the AC rectified input signal to the optocoupler hits 0V however, the LED will not emit light and the BJT will therefore not conduct. This will bring the input to the microcontroller (B0) high, signifying that a zero cross has been detected.

2.3.1.3 MICROPROCESSOR

Clearly pictured in the center of the above schematic is the 18F46204P microcontroller the group chose for the design. Although the capability of this processor may be excessive for this particular application, this part was chosen for multiple reasons. The processing power and capability made this microcontroller attractive given that precision and robustness are critical for this application. Additionally, the group's experience with and access to the 18F46204P during the fall semester made it the best choice for the final design.

The primary function of the triac system microcontroller is to make decisions as to when to "turn on" the AC signal from the generator or the wall. During each AC cycle, the microcontroller first recognizes the zero-cross signal from the detection circuitry as an interrupt. Then based on the desired output

signal, the microcontroller delays a certain amount on time and “fires” the triac itself by setting pin C0 low. This will produce a rectified and cropped AC signal.

One important feature of the digital control provided by the microcontroller is the level of precision that is available. Given a 60 Hz input signal, there will be 8.33 ms between each zero-cross. Given the clock speed of the microcontroller is 10MHz/4 (Fosc/4), there will be an accuracy level of 400ns per count. In theory, this means that there are 20,833 counts in between each zero-cross of a 60 Hz input signal. This level of accuracy is desirable in that it gives a precise level of control.

Additionally, the microcontroller takes information about the proceeding buck converter circuitry into consideration when making decisions about the voltage to output from the triac. This is done via the voltage sensor circuit pictured below the microcontroller in the schematic. In essence, the buck converter will try to meet a set current to deliver to the batteries (Section 2.3.2). In doing so, the buck converter will adjust its output voltage to the 72V stack accordingly. Then this information level will be passed into the triac microcontroller via the A/D converter (MCP3208) and a voltage sensor. The triac microcontroller will then try to set its output voltage to approximately 3V above that outputted by the buck converter.

This is done in order to minimize the power dissipated across the buck converter. By making the voltage differential from the output of the triac to the buck convert and the output of the buck converter to the battery stack as small as possible, the power dissipated is minimized. For example, if there is 5A being delivered to the battery and a total of 3V is being dissipated across the buck converter then a total of 75W must be dissipated by the MOSFET in the buck converter. This value should be as small as possible and is done so by this functionality in the output of the triac to the buck converter.

2.3.1.3.1 SOFTWARE FLOW CHART

Below is a chart depicting the overall flow of the code in the triac microcontroller. For the exact code listing, see the Appendix.

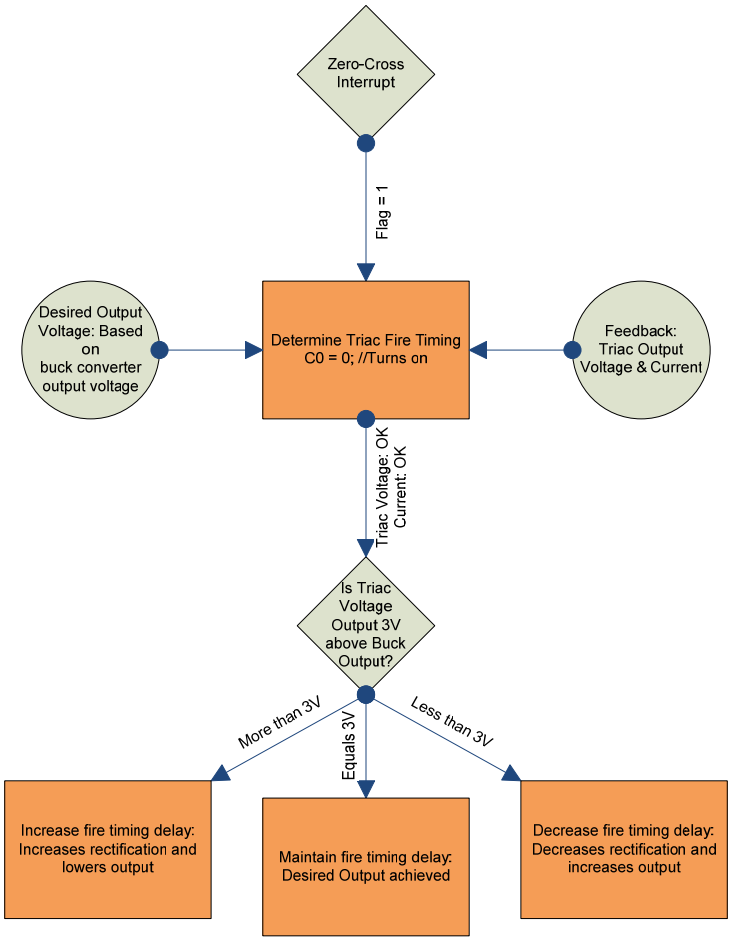

Figure 2.3.1.3.1: Triac Microcontroller Software Flow Chart

The above software flow diagram shows the general flow of code in the microcontroller for the triac circuitry. The process begins when an interrupt occurs due to the detection of a zero cross. This interrupt is recognized by the microcontroller. Then the microcontroller uses the information about the desired output voltage and feedback from the triac's present voltage and current outputs. The desired output voltage is set to be approximately 3V about the output of the buck converter in order to minimize power loss as previously discussed.

The microcontroller then checks to make sure the triac voltage is within its defined bounds and that the current being output is at the desired level. If these safety precautions are met, then the microcontroller compares whether or not the present output voltage is 3V above the output of the buck converter.

For a complete software listing, refer to the Appendix.

2.3.1.4 TRIAC CIRCUIT

The triac circuitry itself is the primary controller of the rectification of the AC waveform. As depicted in the reproduced schematic below, the AC signal is applied to the triac (Q4015L5). Then using an optocoupler (MOC3010M), the microcontroller determines when to fire the triac. By setting pin C0 low, current flows through the LED and turns on the internal triac in the optocoupler. This in turn fires the external trial and allows the remaining portion of the AC signal to pass (See Figure 1.5.2).

This cropped AC signal is then passed through a full bridge rectifier as shown in the schematic. The resulting waveform in then passed to the triac capacitors (21.6 millifarads), resulting in a DC signal.

As an additional note, a 2KΩ resistor is connected across the triac capacitor bay. This bay has such a large capacitance (21.6 millifarads) in order to smooth the desired DC voltage and drive the desired current as efficiently as possible on to the buck converter and eventually the batteries. Therefore, the resistor is connected in order to create a time constant that will dissipate the voltage at a rate that allows for decent responsiveness when the desired output voltage is changed. For example, if the desired output voltage changes from 90V to 85V, the resistor needs to dissipate the 5V at a rapid enough rate to achieve effective responsiveness.

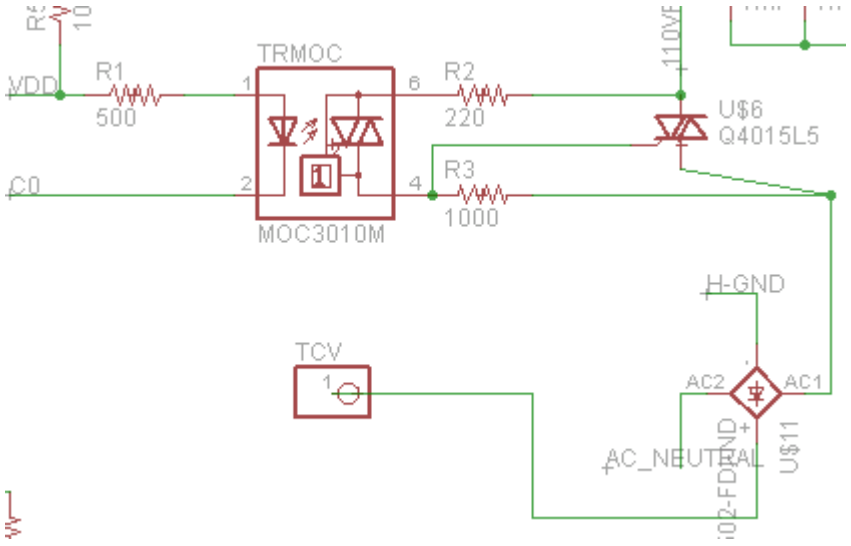

Figure 2.3.1.4: Triac Circuit Schematic

2.3.2 BUCK CONVERTER SYSTEM

The buck converter system functions as a DC-DC converter. The input to the circuit comes from the Triac Circuit System output, and the output of the buck converter is used to charge the batteries. The buck converter is composed of a microcontroller, switching power MOSFET, and smoothing circuitry. These subsystems are described below.

A buck converter functions as a DC-DC converter by varying the switching speed and duty cycle of a MOSFET. An initial DC voltage is applied to the drain of a power MOSFET and the output is taken at the

source. By varying the duty cycle, the output DC voltage can theoretically be controlled from 0 to 100% of the DC voltage on the drain. If the duty cycle was 100% (always on), then the output would theoretically be 0V. If the duty cycle were 0% (always off), then the output would theoretically be that voltage applied to the gate. Therefore a desired DC output can be obtained by picking the proper duty cycle and switching frequency of the power MOSFET.

Through testing and analysis, an ideal switching frequency and duty cycle range was found. For this specific DC-DC application, a switching frequency of 26.1kHz was optimal in terms of output and efficiency in control. Additionally, varying the duty cycle between 16% and 100% seemed to be necessary to meet the desired output from the buck converter.

2.3.2.1 OVERALL SCHEMATIC

The diagram below shows the overall hardware schematic of the buck converter subsystem as well as some additional features necessary for the circuit board design. Each portion of the circuitry will be described below.

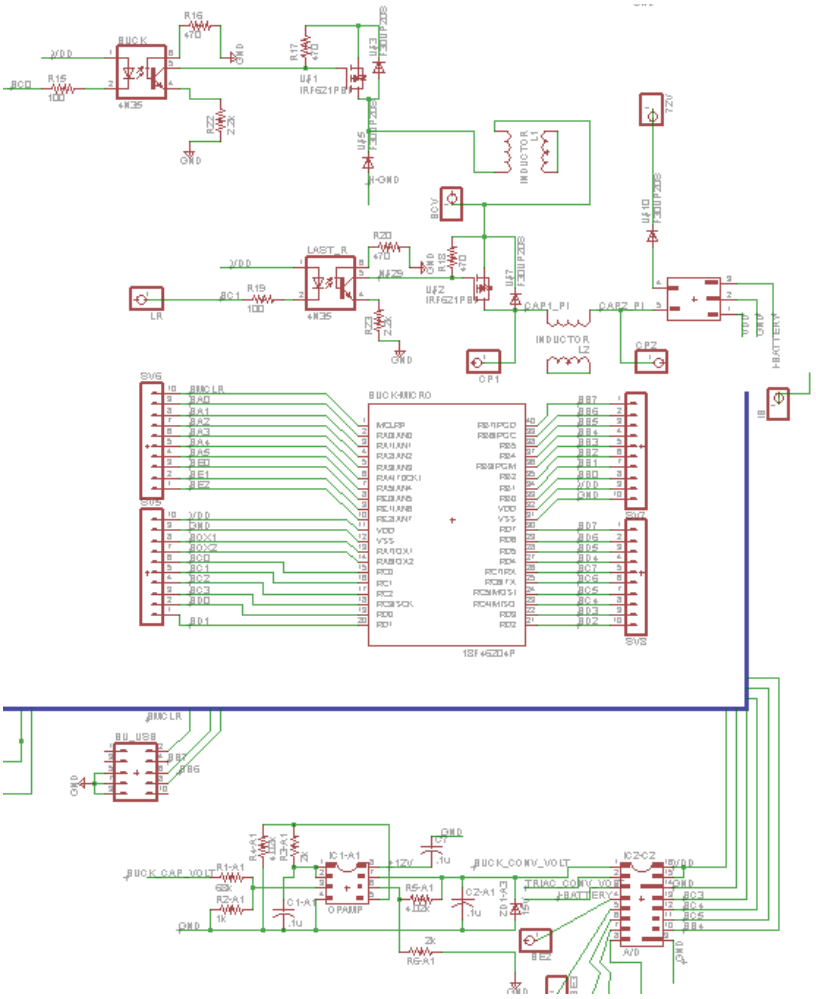

Figure 2.3.2.1: Buck Converter System Schematic

2.3.2.2 MICROPROCESSOR

Clearly pictured in the center of the above schematic is the 18F46204P microcontroller used for the design. Although the capability of this processor may be excessive for this particular application, this part was chosen for multiple reasons. The processing power and capability made this microcontroller attractive given that precision and robustness are critical for this application. Additionally, the group's experience with and access to the 18F46204P during the fall semester made it the best choice for the final design.

The primary function of the microcontroller is to control the voltage and current being output to the 72V battery stack. A desired charging current and an allowable voltage range (65 to 89V) is programmed into the processor and the microcontroller. The main goal is to continuously deliver a constant current within the allowable voltage range applied to the batteries. The voltage range is 65V to 89V, meaning that any voltage in between is allowed to be put on the batteries. The microcontroller then attempts to deliver the desired charging current. If more current is needed, the duty cycle of the control signal to the buck converter is decreased in order to increase the voltage applied and therefore the current sourced. If less current is needed, the duty cycle is increased in order to decrease the voltage applied.

The microcontroller will continue this process until the desired current is reached or until the voltage being applied to reach this desired current reaches one of the defined limits (65V or 89V). If and when the voltage does reach the upper limit, then the voltage is held constant at this 89V limit. This will inherently result in a constant voltage charging state, with a likely decrease in charging current.

This overall control procedure is characteristic of commonly used charging algorithms today. The most efficient algorithms involve an initial constant current mode followed by a constant voltage mode. In this design's case, the constant current mode is implemented by programming a desired current to be delivered for charging. If the batteries are depleted of charge enough, the voltage output by the buck converter will be able to obtain a value that will deliver this current. Then as the batteries are charging, the current will remain constant as the voltage increases within the limited range (65V to 89V) to deliver this current. When the voltage reaches the upper limit of 89V, the voltage will remain at this value and the current will therefore begin to decrease. This will be the transition into the constant voltage mode.

Additionally, the buck converter microcontroller is responsible for controlling the last resort MOSFET after the buck converter circuitry. By setting the appropriate pin as described in Section 2.3.2.5, the MOSFET is able to open circuit the connection to the 72V battery stack in the event of a potentially dangerous condition.

2.3.2.2.1 SOFTWARE FLOW CHART

Below is a chart depicting the overall flow of the code in the buck converter microcontroller. For the exact code listing, see the Appendix.

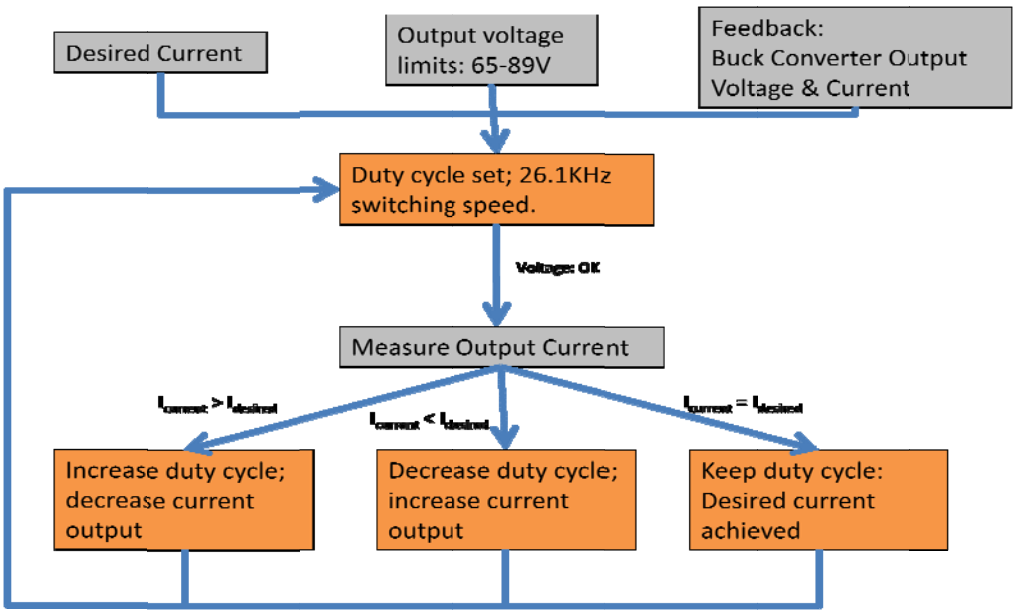

Figure 2.3.2.2.1: Buck Converter Microcontroller Software Flow Chart

2.3.2.3 POWER MOSFET

For the buck converter application, a p-channel power MOSFET, IXTP36P15P, was chosen. This part is ideal from both a functionality and cost standpoint. The MOSFET is rated for 36 amps and 150 volts maximum. Additionally it is able to dissipate 300W of power provided the proper heat sink and ventilation. It is also capable of handling the switching speeds implemented in the buck converter design (26.1 KHz).

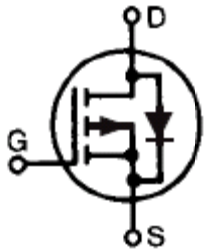

Figure 2.3.2.3: Power MOSFET Schematic

Looking at the above power MOSFET configuration used in the IXTP36P15P, the general schematic can be visualized. Note the standard p-MOSFET configuration. Additionally, the power MOSFET employs a diode allowing any back current to flow from drain to source without damaging the device. This design is common to most power MOSFETs.

2.3.2.4 SMOOTHING CIRCUITRY

The smoothing circuitry, reproduced below, follows the power MOSFET in the buck converter system. This circuit uses a fast-switching diode, an inductor, and a capacitor bay in order to smooth the voltage and produce a smooth DC output.

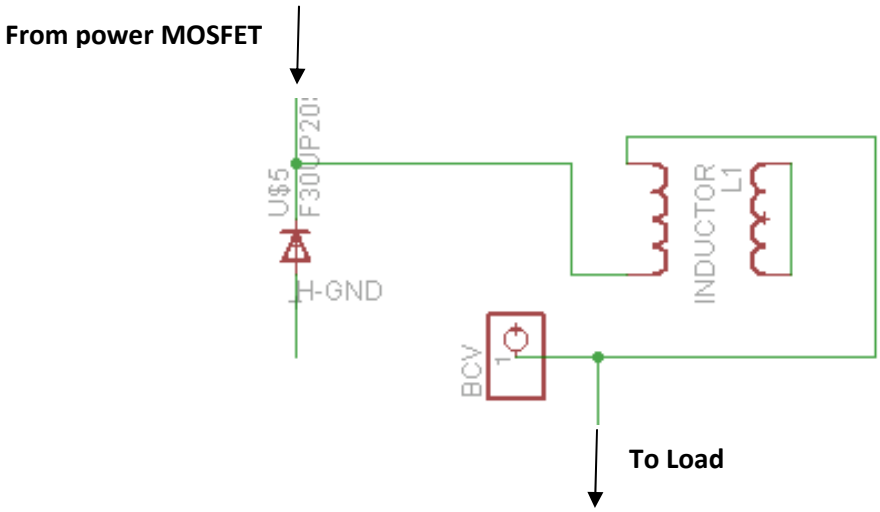

Figure 2.3.2.4: Smoothing Circuitry

The smoothing circuitry functions as follows. When the MOSFET is switched on, current is allowed to pass through the inductor to charge the capacitors (BCV). Then when the MOSFET is turned off, current is no longer flowing into the smoothing circuitry. In this case, the inductor and capacitor attempt to drive current and voltage respectively to the load. The diode directs this flow on to the load by preventing backflow.

The FFPF30UP20STU diode was chosen for multiple reasons. First, its ultra fast switching capability makes it ideal for a switching circuit like the buck convert. Additionally, the part is rated for 200V allowing for an adequate buffer against voltage spikes from the neighboring inductor.

An inductor value of 2.3 millihenries and a capacitance of 20 millifarads were found to be best for this design as well. A high inductor and capacitor value function to effectively and strongly drive energy into the load. Additionally, the large capacitance functions to provide the most stable and constant DC output possible.

2.3.2.5 LAST RESORT MOSFET AND FINAL CIRCUITRY

As a final precaution, a last resort MOSFET was added between the output of the smoothing circuitry from the buck converter circuit and the 72V battery stack itself. This MOSFET functions as a form of last resort protection, directly controlled by the buck converter's microcontroller. Its sole function is to monitor the voltage and current being delivered to the battery stack, shutting off if those values exceed certain programmed limits. This protects against any major power surges that could be pumped into the batteries by providing a simple, yet effective, means of terminating the connection.

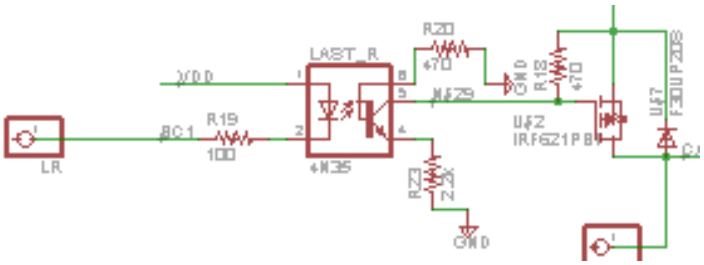

Figure 2.3.2.5(a): Temperature Sensor Configuration

The MOSFET chosen is the same as that used in the buck converter, IXTP36P15P. This is a p-MOSFET, meaning that with a low signal, or no signal, applied it will remain on letting current flow to the batteries. If a limit is exceeded, the microcontroller will turn the MOSFET off, protecting the batteries.

The buck converter microcontroller is connected to the last resort MOSFET via an optocoupler, as pictured in the above schematic. The anode of the optocoupler diode is then connected to port C0 from the microcontroller. Therefore when C0 is set low, current flows from VDD to the microcontroller and turns on the BJT in the optocoupler. Turning on the BJT causes a low voltage to be placed on the gate of the last resort MOSFET, thereby turning it on and allowing current to be passed to the batteries. If a dangerous voltage or current is detected by the microcontroller, C0 is then set high which turns off the BJT in the optocoupler and applies a high voltage to the gate of the last resort MOSFET. This turns off the MOSFET and protects the batteries from the dangerous signal.

After the last resort MOSFET, some additional miscellaneous circuitry exists prior to power delivery to the batteries. This schematic is shown below.

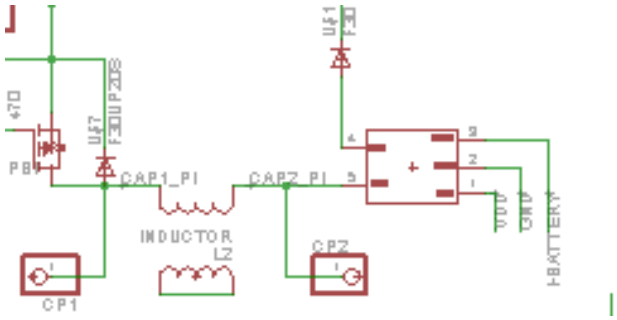

Figure 2.3.2.5(b): Miscellaneous Circuitry Following Buck Converter

Shown directly following the MOSFET, is a pi-filter design that was intended to filter any remaining ripple from the DC signal before the signal was delivered to the batteries. However after extensive testing, this filter proved more harmful than beneficial. It stores a great deal of charge and was observed to have the ability to spike the output voltage through the entire Charging Circuit System to dangerous levels (150V to 160V), destroying some components. Therefore, the inductor was short circuited and the capacitors were left in place as a further smoothing component.

Following the capacitors is the current sensor, ACS756. This part is a 5 pin Hall Effect current sensor. It was chosen because of its sensitivity (20-40 mV/A), and it is powered by 5V. This current sensor

monitors the final current being delivered to the 72V battery stack. This information is then fed back to the buck converter and triac microcontrollers as well as the control circuit.

Finally, the signal is passed through a diode as protection between the batteries and the circuit. If the charging circuitry was not sourcing current, this diode would prevent any backflow that would cause damage to the circuitry. The part used here is the diode FFPF30UP20STU. It is rated for 200V which is more than enough to prevent current backflow.

2.4 DETAILED OPERATION OF CONTROL SYSTEM

This section describes the power supply, microcontroller, software, user interface and communication between the control and charging boards. While many of the sensors are located on the control circuit board, they are not discussed in this section. The user interface is also not covered in the control circuit section. Please refer to "Interfaces and Sensors" for descriptions of these subsystems.

2.4.1 POWER SUPPLY

In the previous configuration, the Lightning Riders tapped two batteries in the 72V stack to power the control system circuitry. Naturally, this voltage was stepped down from 24V to more useable levels, such as 12V, 10V, 5V, and 3.3V. As a result, last year's group perceived some irregularities on the overall voltage stack as a result of this technique. Because the current charging circuitry is no longer functional, it would be a hefty task to reconstruct the previous group's circuitry to test the validity of their theory. For better or worse, the group must accept this evaluation as it is. In this light, the Hybrid's Angels have decided to power the control circuitry by mounting an additional 12V battery to the front of the motorcycle.

All components in the control system cannot be powered at 12V. In fact, most components, such as the microcontroller, the analog to digital converters, and USB to serial converter, require a 5V input. This voltage is generated by utilizing the LM1117-5.0 voltage regulator that produces the necessary output to power these devices. Although most devices need 5V to operate, six components require 12V. These components are the LM392 Comparator/Operational Amplifier chips used to measure the voltage on each of the batteries.

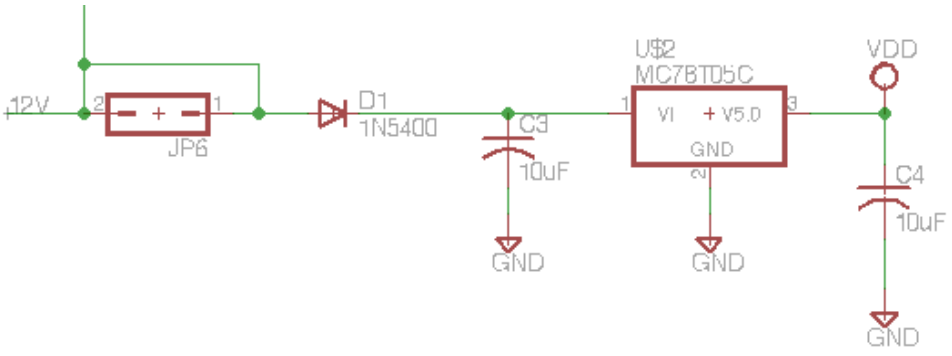

Figure 2.4.1: Power Supply Schematic

2.4.2 PROCESSOR AND PROGRAMMING

Due to the importance of a fast response time to the charging system, the group has opted to use three separate microcontrollers in this project. Two microcontrollers are programmed to monitor and control the charging circuitry, while the remaining one is used to manage the control system circuitry.

For this project, space is certainly not a concern, at least not as it has to do with the onboard electronic circuitry. Thus, the group decided to stick with the seeming EE 41430 standard, 18F4620P microcontroller from MicroChip. This microcontroller provides the group with sufficient speed and port availability to accomplish the goals of this project. Data storage for this device is not an issue as the group will include a MicroSD chip on the order of a couple gigabytes. Admittedly, the decision to utilize this microcontroller was aided by the presence of eight of these chips left over from the year before.

Selection of the microcontroller is just one portion of the decision making process for the control system. In addition, the group needed to allocate ports on the microcontroller. This decision making process was performed mainly by meeting functionality and availability. For instance, certain ports on the device are utilized for Serial Peripheral Interface (SPI) communication; whereas others are strictly reserved for VDD or GND. Listed in the table below is a summary of the allocation of microcontroller ports.

µcontroller pin:	Other End:	Description:
MCLR	Reset Button	Input
A0	#RI	Ring Indicator on USB to Serial
A1	Chip select	Data Storage Chip Select
A2		
A3		
A4		
A5		
E0	Main LED display	Output
E1	Main LED display	Output
E2		
Vdd	5V	
GND	0V	
OX1/A7	Oscillator/timer	I/O
OX2/A6	Oscillator/timer	I/O
C0	Button	Display Button
C1	Button	Display Button
C2		
C3	SPI CLK	SPI Clock (A/D, Storage Card)
D0	Main LED display	Output
D1	Main LED display	Output

B7		
B6		
B5		
B4	Chip select	A/D 3 Chip Select
B3	Chip select	A/D 2 Chip Select
B2	Chip select	A/D 1 Chip Select
B1	CTS	Control Signals
B0	RTS	Control Signals
Vdd	5V	
GND	0V	
D7	Main LED display	Output
D6	Main LED display	Output
D5	Main LED display	Output
D4	Main LED display	Output
C7	TXD	Serial Output
C6	RXD	Serial Input
C5	Data Input	Data Input (A/D, Storage Card)
C4	Data Output	Data Output (A/D, Storage Card)
D3	Main LED display	Output
D2	Main LED display	Output

Figure 2.4.2: Control Circuit Microcontroller Pin out

In this project, the microcontroller plays a key role in running the entire system. From reading important data values to making decisions for the system, the microcontroller is at the heart of this project. To make sure that the microcontroller knows what to do in a given situation, it is of paramount importance to have intelligent, yet robust code to allow the system to function as quickly, efficiently, safely, and effectively as possible.

In the subsequent pages, pseudo-code is given. Whenever the bike is appropriately powered, this code will run on the microcontroller to check the status of the system. Checking the system includes constantly ensuring that voltages, currents, and temperatures remain at proper levels on each of the motorcycle's subsystems. If these values fall outside the specified constraints, the microcontroller is instructed to take action until the system properly stabilizes. The microcontroller will further be responsible for storing important system variables and providing system information to the LCD screen via low priority interrupts.

2.4.3 SOFTWARE FLOW CHART

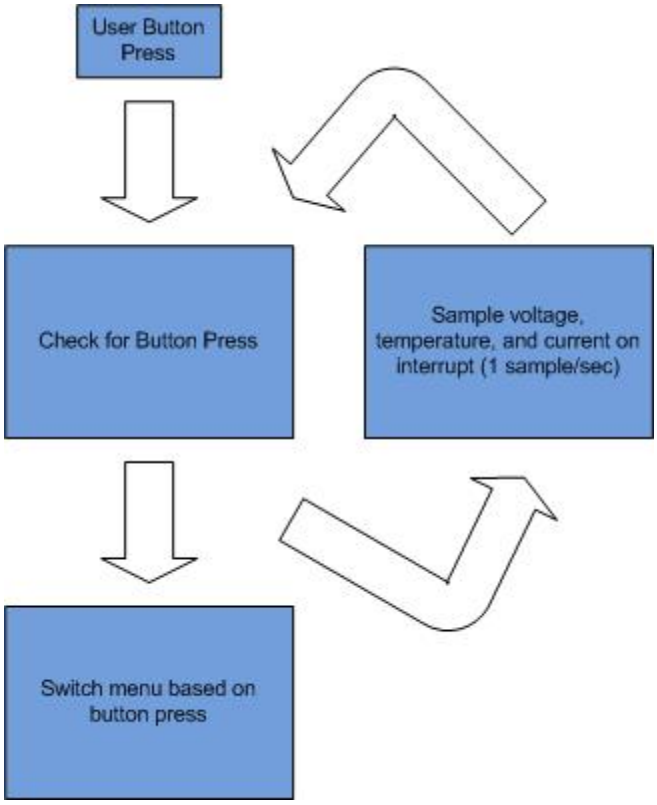

Figure 2.4.2: Control Code Flow

2.4.4 LCD USER INTERFACE

The LCD user interface plays a very important role in the control system, as it is the means by which the microcontroller can communicate intelligibly with the rider of the motorcycle. It is the source of important information on the cycle such as warnings, critical errors, and general information.

When the ignition key is turned on, the LCD screen first displays the message “Hybrid’s Angles, Revision 4/30/2009”. It then reverts to the main menu screen where the user can scroll between the following options: System Status, Battery Voltage, Current Info, and Temp Sensors. Using the buttons on the left handlebar, the rider can scroll up and down through the main menu. The right handlebar buttons control the “enter/select” and “back” functions.

When “System Status” is selected, the LCD screen will display “Start Generator. Instant SOC” followed by a percentage. This percentage is an estimation of the instantaneous state of charge on the batteries.

When the “Battery Voltage” menu is selected, the user is taken to a screen that displays the sensed voltage on each of the six batteries on the stack. This voltage displayed for each battery is not the voltage on each specific battery, but instead is the voltage on the stack up to that battery. For instance,

if Battery 1 had a voltage of 12.3V and Battery 2 had a voltage of 12.5V, then the displayed voltage on the User Interface for Battery 2 would be 24.8V.

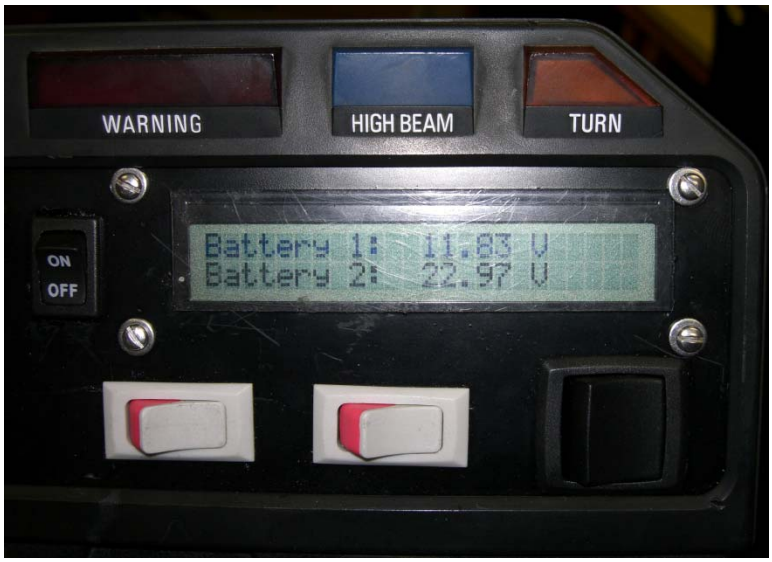

Figure 2.4.4: User Interface Displaying Batter Voltage

The Current Info menu displays information about the current going through the batteries and also the current being drawn from the wall outlet. This is very important when determining whether the current is close to tripping a circuit breaker or whether it is about to exceed the physical limits of our components.

In the Temp Sensors menu, six temperatures are displayed. Each of these temperatures indicates the temperature of a battery. The sensors were mounted directly onto the batteries and they constantly monitor the amount of heat that they are dissipating. This is useful when performing analysis about the efficiency of our system.

2.5 DETAILED OPERATION OF GENERATOR SYSTEM

As originally anticipated, a single generator was used for the series hybrid design. The generator chosen was the PowerMax XP4400E. This model was decidedly the best choice from both a functionality and a cost perspective. The generator is a single-phase, key start model rated to output 3500W. Additionally, the relatively compact size and light weight made it ideal for mounting directly to the motorcycle frame,

The generator was mounted on a metal frame customized to fit the back of the motorcycle. The metal frame was constructed from ¼ inch thick, right-angled iron beams. These beams were then appropriately bent and cut to form a support cage. The cage itself sits low behind the rear wheel. The generator can then be placed inside the metal support cage and fastened with four U-bolts at each of the respective corners.

The major concerns with mounting the generator in this fashion were balance and stability. While the significant weight of the generator does cause the motorcycle's center of gravity to be shifted backward, test runs and experimentation proved that this was not a major issue. The bike experienced no major front to back imbalances due to the added rear weight. As far as side to side stability is concerned, the added weight did cause a noticeable effect. While riding at lower speeds, taking turns proved to be more difficult. The added weight does require that the operator be more cautious when making these turns and other similar maneuvers.

Overall however, mounting the generator in the rear is still thought to be the best option for the time being. Other options could include making a sidecar or putting the generator into a tow cart. While a side-car may prove feasible, it would require a significant amount of design attention as it would shift the side to side center of gravity as well as adding significantly more weight. The idea of a tow cart was discarded due to complications with balance and acceleration. In general, motorcycles are not made for towing as the need for side to side stability is easily compromised at low speeds. Since the bike will be running at these low speeds (likely at or under 40mph), this would not have been a useful design. In the end, mounting the generator to the back provided the best solution at the time.

2.6 SOLUTION FOR POWERING CIRCUIT BOARDS

In order to power the two circuit boards (charging circuit and the buck converter circuit), an on board 12V chargeable lead-acid battery was mounted on the motorcycle. The battery chosen for this application was the Werker WKA12-5F model as pictured below. This battery proved ideal for the application because all the circuitry required 12V or less for power. Additionally, the entire control and charging circuitry was tested and found to draw approximately 40mA of continuous current, making this battery a feasible solution.

This battery was mounted in the front of the motorcycle, above the 72V stack and beneath the circuit boards. It was mounted using basic zip ties to hold it in place. Additionally, the battery comes with a simple trickle charger that allows for charging when the bike is not in use.

2.7 INTERFACES AND SENSORS

As monitoring and communication among subsystems of the utmost importance for this hybrid motorcycle design, interfaces and sensors perform a pivotal role in the design. These components are described in more detail here.

2.7.1 INTERFACES

A USB to serial converter, the FT232RL from FTDI chip, is included in the group's design of the control system circuitry. This device provides a valuable interface to not only program the microcontroller but also download data from the device. Although one of the future enhancements, staunchly proposed by

team member Stephen Govea, was to install a wireless interface to download data, and even program the microcontroller, the USB to serial converter provides a simple, yet powerful way to interact with the microcontroller in case the wireless connection did not come to fruition.

2.7.2 SENSORS

Three basic sensor types were used on the motorcycle in order to measure voltage, current, and temperature. Additionally, the analog to digital conversion of circuitry is described here as it involves the sensing and processing of a signal.

2.7.2.1 VOLTAGE SENSORS: LM350P

Measuring the voltage on each of the batteries in the 72V stack is of obvious importance for the charging circuitry to react properly to the change in state of the batteries. Thus, the voltage on each of the batteries, as well as the entire stack, must readily be available to both protect and enhance the charging circuitry as much as possible. To accomplish this objective, the group decided to utilize the method that the Lightning Riders used last year with some minor alterations to correct perceived problems with the design.

Ultimately, this design renders a voltage between 0V and approximately 2.5V corresponding to the charge level on the specific battery. This method provides a voltage, which can safely be translated to the microcontroller via the analog to digital converter circuitry. Then, the microcontroller works backwards to determine the true voltage on each of the batteries in the stack. With this information, the microcontroller is able to inform the user of the charge state as well as make the necessary alterations to the voltage level (and current level) being output by the charging circuitry.

Naturally, calibration is an important element in this process; however, due to the proportionality of the constructed system, this calibration was easily tested and accomplished.

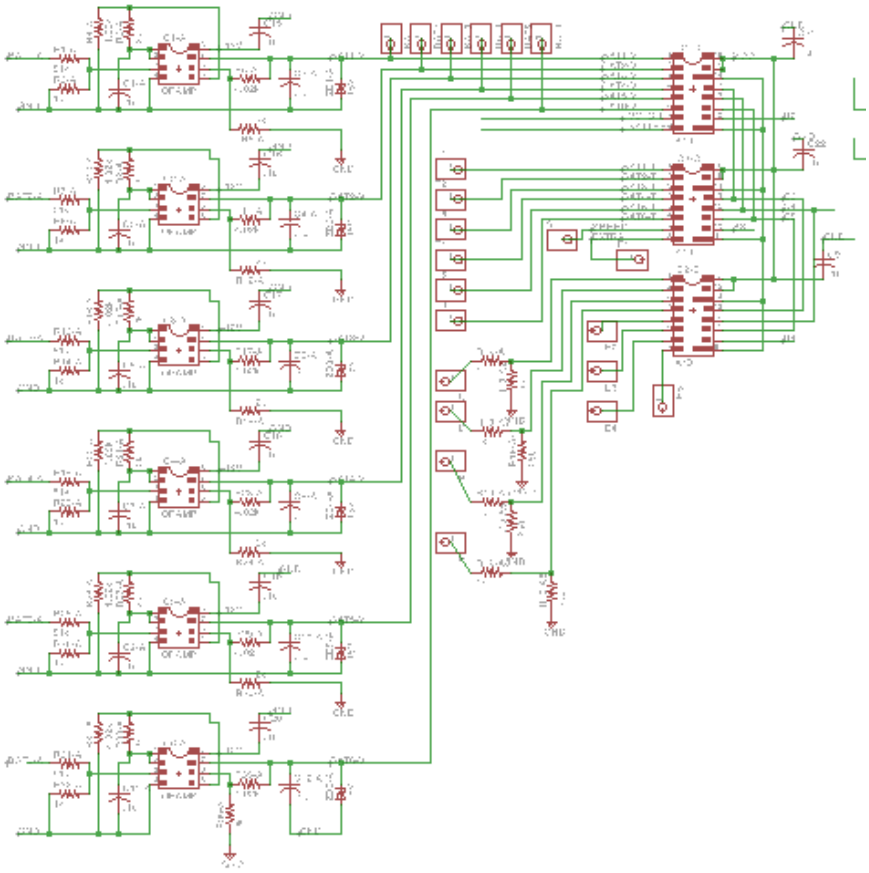

Figure 2.7.2.1: Voltage Sensor Schematic

Each battery's high voltage terminal is fed into the control system circuitry and stepped down to approximately one tenth of its initial value. This voltage is fed into a comparator, whereby it is compared with its previous value, allowing it to be amplified to that voltage at any given time. This voltage is once again stepped down to roughly two thirds of its value and fed into an operational amplifier. Configuring the operational amplifier in a non-inverting configuration allows for the voltage to be solved as a value between 0V and 2.5V.

Additional features of the voltage sensor circuitry include decoupling capacitors to better stabilize the important voltages that are fed into the operational amplifiers as well as a Zener diode to help protect the system from short circuits.

2.7.2.2 CURRENT SENSOR: ACS756

The current sensor chosen for this design was the Allegro Microsystems part, ACS756. This is a Hall Effect linear current sensor with a 3kVrms isolation rating. Additionally, the sensor is provides a feedback with an accuracy of 20-40mV/A. Finally given its small size and low power requirement, this sensor proved ideal for the needed application.

This current sensor is utilized after the buck converter circuitry to measure the direct current that is being sourced to the battery stack for charging, as pictured below. As shown, current enters the sensor through Pin 5 and exits via Pin 4. Power and GND are connected to Pin 1 and Pin 2 respectively. The output voltage corresponding to the current passed is output via Pin 3.

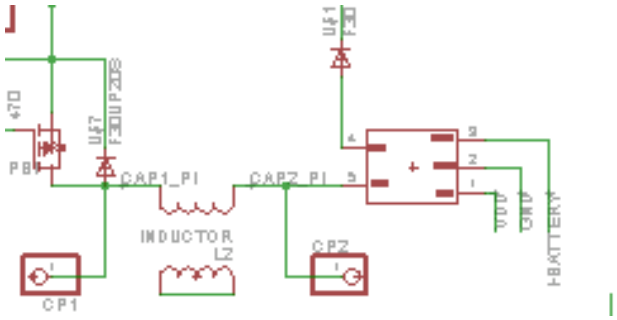

Figure 2.8.2.2: Current Sensor Placement Schematic

2.7.2.3 TEMPERATURE SENSORS: LM35DT

Temperature sensor circuitry is implemented in the same fashion as the Lightning Riders began to do. At present, the motorcycle has temperature sensors mounted on all of the six batteries. As with the voltage sensors, these devices will have to be calibrated accordingly.

As five of the six sensors were already mounted, the group naturally decided to stick with the same technology chosen by last year's group, the LM35DT. This sensor is constructed in the basic configuration outlined in the manufacturer's data sheet and shown below. This configuration allows for temperatures to be measured from 2°C to 150°C. This range should be more than sufficient for the temperatures that occur in the battery.

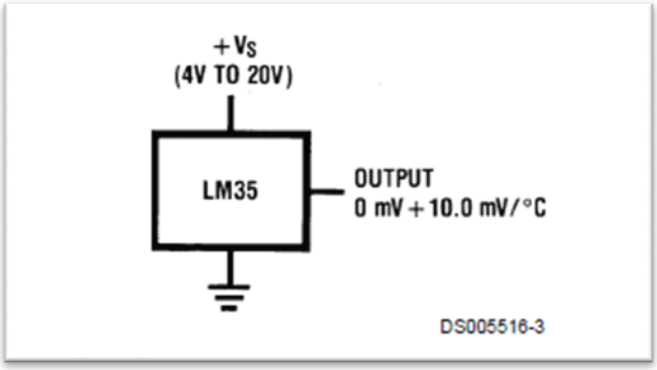

Figure 2.8.2.3: Temperature Sensor Configuration

It can also be seen in the diagram above that the output voltage is proportional to 10mV per degree Centigrade. This output voltage is fed into an analog to digital converter, which passes the voltage on to the microcontrollers. At this point, the Control System's microcontroller can both inform the user of the temperature on the batteries as well as make decisions to remedy a high temperature if necessary.

2.7.2.4 ANALOG TO DIGITAL CONVERSION OF SIGNALS: MCP3208

Proper analog to digital conversion is essential for the microcontroller to properly receive feedback regarding currents, voltages, and temperatures that various components of the system may be expecting. Once again, the group utilized the analog to digital converters purchased by the Lightning Riders.

These analog to digital converters, which are part number MCP3208, have eight channels which interface with the microcontroller via SPI. Basically, the microcontroller selects one of the eight channels on the analog to digital converter and the device outputs the voltage on that channel in binary. This output value is based on the following digital output code:

$$Digital\ Output\ Code = \frac{4096 \times V_{IN}}{V_{REF}}$$

Where:

V_{IN} = analog input voltage

V_{REF} = reference voltage

Figure 2.8.2.4: Digital Output Code for Analog to Digital Converter

Since the reference voltage is 5V, a temperature sensor outputting a voltage of 0.5V sends the following binary values to the microcontroller: 110011010, corresponding to 410. This value allows the microcontroller to know that the temperature sensor is measuring a temperature of 50°C.

Obviously, each analog to digital converter requires its own distinct chip select port on the microcontroller. By periodically cycling through all of the channels, the microcontroller is able to read critical information from the system's sensors and process decisions accordingly.

To increase the accuracy of the device, a decoupling capacitor is included on the reference voltage. This decoupling capacitor will stabilize the reference voltage lending to better accuracy in the overall system.

3 SYSTEM INTEGRATION TESTING

3.1 INTERGRATION TESTING

The integrated set of subsystems was tested in the following manner. Initially each system (control system, charging system, and monitoring system) were tested separately. Next, the control system and the charging system were integrated and tested together. Finally all three of the major subsystems were connected together and a complete test routine was performed. The following provides a more in depth description of the test procedure.

3.1.1 CHARGING SYSTEM TESTING

The charging system circuitry was tested across each individual subsystem as well. First the triac circuitry was tested. Then the buck converter circuit was added and tested. A more in depth explanation of the testing is given here.

3.1.1.1 TRIAC CIRCUIT TESTING

The triac circuit was tested by programming the microcontroller to control the output to be 92V. Once this was done, an AC variac transformer was used to gradually increase the signal applied to the triac from the wall outlet. The AC voltage from the wall could then be manually and gradually increased in order to slowly observe the accuracy and robustness of the triac system design.

It was demonstrated that a full wall voltage of 110Vrms could be applied to the triac system, resulting in a 92V DC output varying from +/- 1 V. This output was desired and the variation actually proved to be less prevalent than was previously anticipated from other prototype testing. The system was then tested to output other regulated DC voltages. These tests proved successful with the same degree of accuracy.

3.1.1.2 BUCK CONVERTER TESTING

Following successful implementation and testing of the triac system, the buck converter system was added to the output of the triac and additional testing was performed. This was done in the following manner. First, a desired output voltage was programmed to the microcontroller of the buck converter system. Additionally, the microcontroller for the triac circuit was programmed to output approximately 3V above the desired output of the buck converter. The reasoning behind this was to minimized power dissipation across the switching MOSFET in the buck converter, as previously described.

Initially the buck converter was programmed to output 90VDC. This setup was implemented and executed resulting in 90VDC appearing on the output of the buck converter. This in turn resulted in approximately 93VDC being output into the buck converter by the preceding triac system. The buck converter was then reprogrammed to output a variety of other DC voltages, proving it was functional and versatile.

3.1.1.3 OVERALL CHARGING SYSTEM TESTING

The next testing step was to test the entire charging system by charging the batteries. This was accomplished by connecting the output of the buck converter to the 72V battery stack. This step was necessary to test the fact that current could actually be delivered from the charging circuitry to the battery stack for charging.

The buck converter microcontroller was then programmed to deliver a low amount of current (250mA) to the battery stack. This was accomplished through reading the current being delivered using the current sensor. This current was then increased or decreased by changing the desired voltage output from the buck converter automatically in the code. The triac circuitry continued to output 3V above the buck converter output as before.

The desired current was then systematically increased in order to test the current limit that could be supplied to the battery stack. Eventually, 8A was able to be delivered for a relatively short period of

time. Current delivery at this level caused a great amount of heat across the MOSFET and the associated heat sink of the buck converter given the increased amount of power dissipation. After further testing and manual temperature sensing, the team chose to maintain a 3A – 4A charging circuitry as it resulted in a steady and controllable charge.

The final charging system design was then tested by charging the 72V battery stack from a stack of close to zero charge to full charge. This was done by programming the buck converter microcontroller to output 3A to the batteries. Given that the batteries together are rated for 38AH, this 3A charging current was applied for approximately 13 hours continuously.

3.1.2 CONTROL SYSTEM TESTING

After receiving the fabricated board, the various components were soldered and tested in the different circuits. The first circuits tested were the voltage sensors. With six voltage sensor circuits on the board, after each circuit was soldered the necessary source voltages and grounds were applied in addition to a test voltage; in this way we were able to verify that the sensors were outputting the required analog voltage.

The next important circuit to test was the power supply to the board. The various resistors and voltage regulators were systematically attached to the board and tested to make sure that the proper voltage was being supplied to the microcontrollers and voltage sensors.

Once able to power the microcontroller, all the components related to it and its output ports such as resistors and decoupling capacitors were attached. Test programs were then used to verify the functionality of the ports.

3.2 MEETING DESIGN REQUIREMENTS

Looking at the above tests and the associated results, it is clear that the design has met the primary design requirements outlined in the project proposal and further project plans.

The primary requirement was to build a system capable of charging a 72V battery stack. As noted above, this was accomplished. It was demonstrated that the circuitry is able to deliver 3-4A of constant charging current to the batteries. This overall functionality is a result of the successful adherence to design requirements of the subsystems.

The triac was demonstrated to perform its desired function of acting as a variable AC-DC transformer. This circuit successfully transforms the 120Vrms wall voltage to a usable DC voltage that is 3V above the output of the following buck converter. The buck converter circuitry was demonstrated to meet its requirements as well. A desired current can be programmed within a given voltage range. The buck converter will attempt to output this current, resulting in a matching of the desired current or a maximum voltage condition during which the voltage is held at its maximum limit. This constitutes a successful demonstration of a Constant Current – Constant Voltage charging algorithm.

In addition to the successful demonstration of the primary design requirement, this project also successfully met many secondary requirements. A user-friendly interface was implemented. Also, the

process of attaching the generator to the back of the motorcycle was no small feat. All of these add to the overall success in meeting design requirements.

4 USERS MANUAL

4.1 HOW TO CHARGE THE MOTORCYCLE

Hybrid's Angels have designed this motorcycle to charge from a standard 120VAC wall outlet. Simply plug the cord into the nearest outlet and let the vehicle charge for 7 hours to ensure maximum energy storage. When the bike has been charged, remove the cord from the wall outlet and bring it around to the mounted generator. Plug the cord into the generator outlet.

4.2 HOW TO OPERATE THE MOTORCYCLE

4.2.1 HOW TO PREPARE THE MOTORCYCLE FOR USE

Figure 4.2.1: Generator

Make sure microcontroller battery is connected.

4.2.1.1 HYBRID MODE

The generator must be started first and allowed to warm up for a period of a few minutes with no load attached. To do so, close the choke all the way and turn the key to the start position. When the motor starts, slowly adjust the choke to the open position to obtain the maximum running speed. Allow the generator to run for a few minutes. Then flip the circuit breaker switch located on the right hand side of the plastic casing in front of the rider to begin charging the batteries.

The motorcycle is then ready to ride. Insert the key into the ignition located under the main display. Turn it right to the "on" position and observe the motor-controller. This will be a grey box located under the charging circuitry and between the batteries. Wait approximately 20 seconds for a red LED to light up on the top of the motor-controller before mounting the vehicle and pushing off of the kickstand.

4.2.2 HOW TO RIDE THE MOTORCYCLE

Mount the motorcycle with one leg over the seat. Do not let the motorcycle lean too far to either side lest it become unbalanced and fall. The motor is operated primarily with the throttle found on the right handle and the brake found on the left handle. The brake mechanism is regenerative so when braking, take extra care to start very early before the intended stop point.

There is also an auxiliary foot-brake located underneath the right foot. Simply depress the pedal to brake the motorcycle.

Figure 4.2.2: ***Foot-Brake located under the right foot***

4.2.3 HOW TO OPERATE THE USER INTERFACE

The Hybrid's Angels have designed, programmed, and installed a fully-functional user interface on the motorcycle which provides a wealth of data about key performance statistics. This interface is navigated

by the bi-directional switches on each handle. On the left handle, the (left) switch moves up a given menu and the (right) switch moves down a given menu. On the right handle, the (right) switch selects a menu and the (left) switch moves back to the previous menu. When the ignition key is turned on, the LCD screen first displays the message "Hybrid's Angles, Revision 4/30/2009". It then reverts to the main menu screen where the user can scroll between the following options: System Status, Battery Voltage, Current Info, and Temp Sensors.

Figure 4.2.3(a): *Left Handlebar*

Figure 4.2.3(b): Right *Handlebar*

4.2.3.1 HOW TO READ BATTERY VOLTAGE

When the "Battery Voltage" menu is selected, the user is taken to a screen that displays the sensed voltage on each of the six batteries on the stack. This voltage displayed for each battery is not the voltage on each specific battery, but instead is the voltage on the stack up to that battery. For instance, if Battery 1 had a voltage of 12.3V and Battery 2 had a voltage of 12.5V, then the displayed voltage on the User Interface for Battery 2 would be 24.8V.

Figure 4.2.3.1: Displaying Battery Voltage

4.2.3.2 HOW TO READ BATTERY CURRENT

The Current Info menu displays information about the current going through the batteries and also the current being drawn from the wall outlet. This is very important when determining whether the current is close to tripping a circuit breaker or whether it is about to exceed the physical limits of our components.

Figure 4.2.3.2: Current

4.2.3.3 HOW TO READ BATTERY TEMPERATURE

In the Temp Sensors menu, six temperatures are displayed. Each of these temperatures indicates the temperature of a battery. The sensors were mounted directly onto the batteries and they constantly monitor the amount of heat that they are dissipating. This is useful when performing analysis about the efficiency of our system.

Figure 4.2.3.3: Battery Temperatures

4.2.3.4 SYSTEM STATUS

When “System Status” is selected, the LCD screen will display “Start Generator. Instant SOC” followed by a percentage. This percentage is an estimation of the instantaneous state of charge on the batteries. It will additionally give other status updates on the system.

Figure 4.2.3.4: System Status

5 CONCLUSIONS

Overall, the Hybrid's Angels believe that their final design was successful. As the primary goal was to charge the 72V battery stack, the team has met the major design challenge presented in the original problem statement from September 2008. In designing this solution, the Hybrid's Angels were able to break down and divide a much more complex challenge into many smaller, more understandable subsystems. Additionally, the design incorporated many additional other features such as an updated user interface and the successful mounting of a large generator to the rear of the motorcycle. Although the team was not able to fully meet the elaborate goals outlines in the original project proposal at the beginning of the design process, overall the final status of the project can be considered a success. The Hybrid's Angels were able to convert an electric motorcycle to a hybrid.

6 APPENDICIES

6.1 HARDWARE SCHEMATICS

6.1.1 CHARGING CIRCUITRY

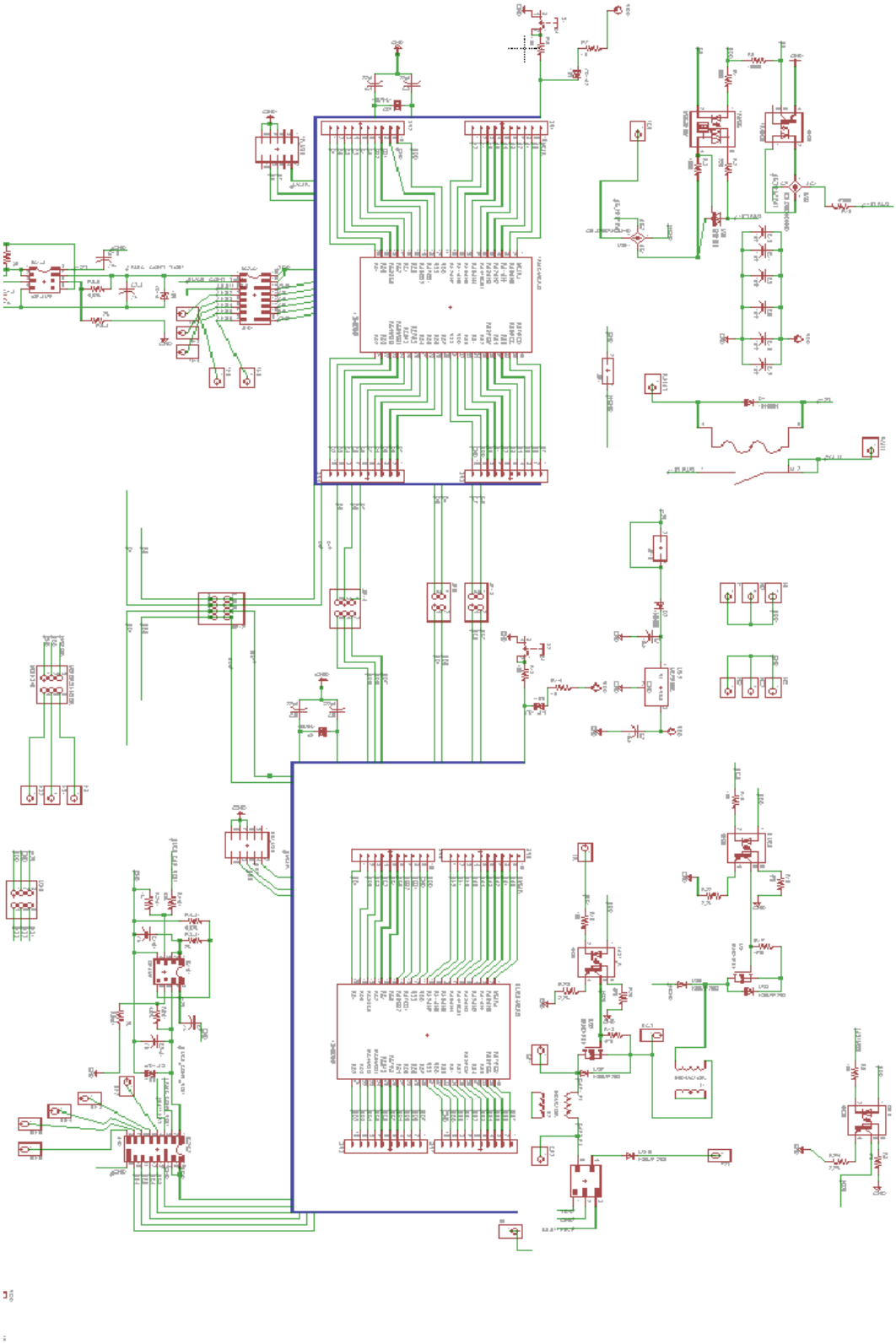

Figure 6.1.1: Charging Circuitry Schematic

6.1.2 CONTROL CIRCUITRY

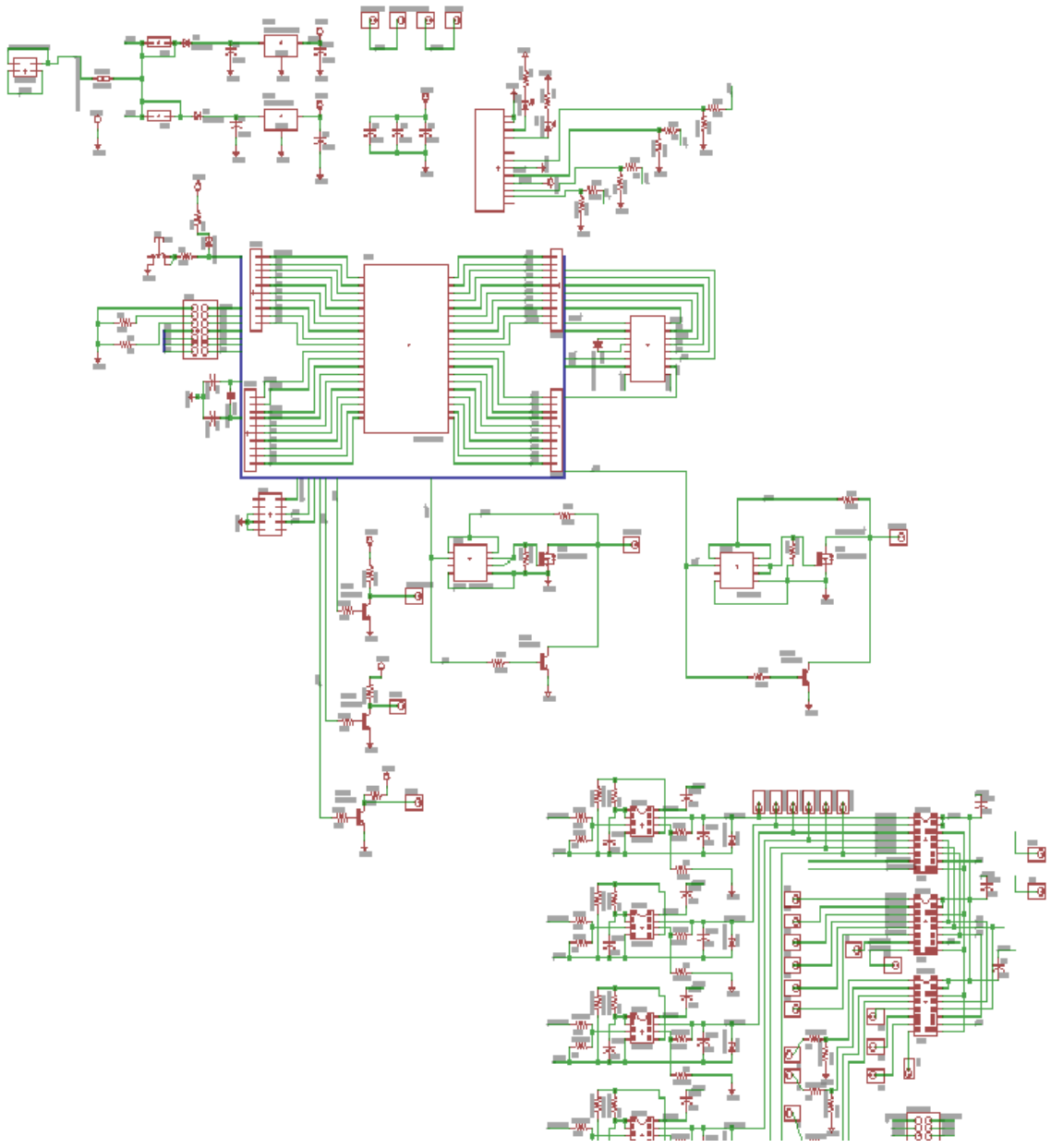

Figure 6.1.2: Control Circuit Schematic

6.2 SOFTWARE LISTINGS

6.2.1 TRIAC CIRCUIT MICROCONTROLLER

```
/************************************************
 * This is a routine to test the basic functions
 * of the board used in 330/340 in 05-06.
 * It does the following:
 *  COunts up and down on the LED's twice
 *    Writes a message to the LCD
 *    runs an 57600 a terminal program which
 *  echoes character to the screen and to the LCD
 *
 * ***********************************************/

#include <system.h>

#include "EESD.h"

#pragma DATA _CONFIG1H, _OSC_HS_1H //10 mhz
#pragma DATA _CONFIG2H, _WDT_OFF_2H
#pragma DATA _CONFIG4L, _LVP_OFF_4L & _XINST_OFF_4L
#pragma DATA _CONFIG3H, _MCLRE_ON_3H

#pragma CLOCK_FREQ 10000000

#define min_val 1000
#define max_val 9000

void modulation(void);

unsigned int dim_on;
unsigned int dim_off;
unsigned int val = 1800;
unsigned char mode;  // the mode
unsigned short tmr0_reg@TMR0L;

volatile bit fire@PORTC.0 = 1;
      long temp;
      long temp2;
      long avg=0;
      long avg2=0;
      long des_volt = 7500;
      long prev_volt = 0;
      unsigned int data;
      unsigned int data2;
      int counter = 0;
      int dat_count = 0;
      int flag =0;

void interrupt(void)
{
      // zero crossing interrupt
      // zero crossing interrupt
      if(intcon.INT0IF) {
            //fire = 1;
            dim_on = val;
            dim_off = 44702 + dim_on + 1000;
            dim_on = 65535 - dim_on;
            intcon.INT0IF = 0;
```

```
            mode = 0;
            t0con = 0x8f;
            tmr0_reg = dim_on;
      }
      // timer 0 expired
      if(intcon.TMR0IF)
      {
            if (mode == 0)
            {
                  fire = 0;
                  t0con = 0x8f;
                  tmr0_reg = dim_off;
                  mode = 1;
            }
            else
            {
                  fire = 1;
                  t0con = 0x0f;
            }
            intcon.TMR0IF = 0;
      }
}

void main(void)
{
      trisb = 0x01;
      portb = 0x00;

      trisc.0 = 0;
      trisc.1 = 0;

      // set up timer and interrupts
      t0con = 0x0f;  // timer 0 off, 8 bit, instruction clock, prescaler off
      intcon = 0xF0;
      intcon2 = 0x85;  // pullups off, int0 falling edge
      rcon.IPEN = 0;  // disable priority levels
      adcon1 = 0x03;


      val = min_val; // select value between 1700 and 20833.333/2 (10400)
      //dim_off = 65535-val;
      //dim_on = 20830 - 2*val;
      //dim_on = 65535 - dim_on;
      //dim_on = dim_on + val;

      SPI_bat_init();

      while(1)
      {
            modulation();
            //val = 3300;
      }

}

void modulation()
{
            AD_sample(0,1,3,data);
            AD_sample(1,1,3,data2);
            counter++;
            avg = data + avg;
            avg2 = data2 + avg2;
            if (counter >=50)
            {
```

```
while(1)
{
	//LCD_setpos(0,0);
	//LCD_dec(data);
	data = avg/counter;
	if (data <= 22)
	{
		AD_sample(0,1,3,data);
		dat_count++;
		if (dat_count >= 3)
		{
			val = min_val;
			dat_count = 0;
			break;
		}
	}
	else
	{
		break;
	}
}
	if (data <= 40)
	{
		data = 40;
	}
	//data = 22;
	temp = (long)data;
	temp = temp * 500 / 29 * 1000 / 4096;

if (temp <= 1000)
{
	counter++;
	if (counter >=5)
	{
		val = min_val;
		counter = 0;
	}
	else
	{}
}
else
{
	if (des_volt >= temp)
	{
		val++;
	}
	else
	{
		val--;
	}
	/*if (des_volt >= temp && des_volt > prev_volt)
	{
		val++;
	}
	else if (des_volt >= temp && des_volt < prev_volt)
	{
		val++;
	}
	else if (des_volt < temp && des_volt < prev_volt)
	{
		if ((temp - des_volt) >= 250)
		{
			//val = val/2;
			//val = val - val/10;
```

```
                        val--;
                    }
                    else
                    {
                        val--;
                    }
                }
                else if (des_volt < temp && des_volt > prev_volt)
                {
                    val--;
                }
                else {}*/

                if (val >= max_val)
                {
                    val = max_val;
                }
                else if (val <= min_val)
                {
                    val = min_val;
                }
                else
                {
                    val = val;
                }
            }
            prev_volt = temp;

            data2 = avg2/counter + 25;
            temp2 = (long)data2 *500/39*1000/4096;
                if (temp2 <= 7000)
                {
                    des_volt = 7200;
                }
                else if (temp2 >= 9000)
                {
                    des_volt = 9200;
                }
                else
                {
                    des_volt = temp2 + 300;
                    //des_volt = 8000;
                }
            avg2 = 0;
            avg = 0;
            counter = 0;
            }
            //des_volt = 8200;

            if (des_volt > 9000 && temp2 >= 9100)
            {
                val = min_val;
            }
}
```

6.2.2 BUCK CONVERTER MICROCONTROLLER

```
/************************************************
 * This is a routine to test the basic functions
 * of the board used in 330/340 in 05-06.
```

```
 * It does the following:
 *  COunts up and down on the LED's twice
 *    Writes a message to the LCD
 *    runs an 57600 a terminal program which
 *  echoes character to the screen and to the LCD
 *
 * ***********************************************/

#include <system.h>

#include "EESD.h"

#pragma DATA _CONFIG1H, _OSC_HS_1H //10 mhz
#pragma DATA _CONFIG2H, _WDT_OFF_2H
#pragma DATA _CONFIG4L, _LVP_OFF_4L & _XINST_OFF_4L
#pragma DATA _CONFIG3H, _MCLRE_ON_3H

#pragma CLOCK_FREQ 10000000

#define max_current 3250

void set_pwm(int freq , int duty)
 {
  //Basic PWM module

      trisc.1 = 0; //set CCP2 to output

      //pr2 controls the frequency
      pr2 = freq;

      ccpr2l.7 = duty.9;
      ccpr2l.6 = duty.8;
      ccpr2l.5 = duty.7;
      ccpr2l.4 = duty.6;
      ccpr2l.3 = duty.5;
      ccpr2l.2 = duty.4;
      ccpr2l.1 = duty.3;
      ccpr2l.0 = duty.2;
  //ccpr21 and bits 4 and 5 control the duty cycle
      ccp2con.5 = duty.1;
      ccp2con.4 = duty.0;

  t2con = 0x6; //turn timer 2 on and set prescale to 16 (110)

  ccp2con = 0xc; // 00001100 to set to PWM mode

 }

volatile bit LRMOS@PORTC.0;

void main(void)
{
      int duty = 10;
      int freq = 5;
      unsigned int data;
      unsigned int data2;
      long conv;
      long conv2;
      long current;
      long des_volt = 5000;
      long h_volt = 0;
      long l_volt = des_volt;
      int count;
      long avg;
```

```
int lmoscount = 0;

trisc.0 = 0;

SPI_bat_init();
//LCD_init();
LRMOS = 0;

while(1)
{
      AD_sample(0,1,3,data);
      //data = 0;
      if (data <= 10)
      {
            data = -40;
      }
      data = data + 40;
      conv = (long)data * 500 * 1000 / 39 / 4096;
      if (conv >= (des_volt+0))
      {
            duty++;
      }
      else if (conv < (des_volt - 0))
      {
            duty--;
      }
      else
      {}

      if (duty <= 4)
      {
            duty = 4;
      }
      else if (duty > 1023)
      {
            duty = 1023;
      }
      else
      {
      }

      if (conv > h_volt)
      {
            h_volt = conv;
      }

      if (conv < l_volt)
      {
            l_volt = conv;
      }

      AD_sample(2,1,3,data2);
      avg = avg + data2;
      count++;
      if (count >= 100)
      {
            data2 = avg/count;
            if (data <= 8)
            {
                  data = 8;
            }
            data2 = data2 - 8;
            conv2 = (long)data2 * 5000 /4096;
            conv2 = 2506 - conv2;
```

```
                conv2 = conv2*1000/40;
                conv2 = conv2 - 500;

                if (conv2 >= max_current)
                {
                      des_volt = des_volt - 5;
                }
                else
                {
                      des_volt = des_volt + 5;
                }
                if (conv2 >= max_current + 1000)
                {
                      des_volt = des_volt - 50;
                }
                else
                {
                }

                if (des_volt >= 8000)
                {
                      des_volt = 8000;
                }
                else if (des_volt <= 6500)
                {
                      des_volt = 6500;
                }
                count = 0;
                avg = 0;
          }

          set_pwm(freq, duty);
     }


}
```

6.2.3 CONTROL CIRCUIT MICROCONTROLLER

```
/************************************************
 * This is a routine to test the basic functions
 * of the board used in 330/340 in 05-06.
 * It does the following:
 *  COunts up and down on the LED's twice
 *    Writes a message to the LCD
 *    runs an 57600 a terminal program which
 *  echoes character to the screen and to the LCD
 *
 * ***********************************************/

#include <system.h>

#include "EESD.h"

#pragma DATA _CONFIG1H, _OSC_HS_1H //10 mhz
#pragma DATA _CONFIG2H, _WDT_OFF_2H
#pragma DATA _CONFIG4L, _LVP_OFF_4L & _XINST_OFF_4L
#pragma DATA _CONFIG3H, _MCLRE_ON_3H
```

```
#pragma CLOCK_FREQ 10000000

unsigned short tmr0_reg@TMR0L;
bool check_voltage = 1;

void interrupt(void)
{
	//if(intcon.1==1) //high priority external interrupt
	//{
		//intcon.1 = 0; //reset external interrupt flag
	//}
	if (intcon.2==1) //tmr0 low priority interrupt, once every second
		{
		tmr0_reg = 0x6769; //put initial value into register so interrupts
at 1 sec
		intcon.2 = 0;	//reset tmr0 interrupt flag
		check_voltage = 1;
		}
	//if(intcon3.0 == 1) //low priority external interrupt
	//{
	//	critical_flag = 1;
	//	intcon3.0 = 0; //reset external low priority interrupt flag
	//}
}

void main(void)
{
	//int j = 0;
	rcon.7 = 1; //enables priority levels on interrupts
	intcon = 0xB0; //set preferences for  treatment of interrupts
	//intcon3.6 = 0; //set INT1 as a low priority interrupt
	//intcon3.3 = 1; //INT1 enabled
	intcon3 = 0x08;
	adcon1 = 0x0f; //set preferences for tmr0 interrupt
	t0con = 0x85;  //set preferences for tmr0 interrupt

	SPI_init();
	LCD_init();
	INTRO();

	while(1)
	{
		//intcon3.0 = critical();
		//if (critical_flag == 1)
		//{
		//	critical_flag = 0;
		//	break;
		//}
		//critical();
		if (check_voltage == 1)
		{
			sample_all();
			check_voltage = 0;
		}
		else
		{
			check_menu();
			//test();
		}
	}
}
```

6.2.4 EESDLIB.C (FUNCTION LISTING)

```
#include    <system.h>
#include    "EESD.h"

battery bat[7]; // struct for battery information
currents cur[3]; // struct for current information; Current 1 = motor current;
Current 2 = battery current
unsigned int index = 1; //index for battery information (vals 1-6)
unsigned int cur_index = 1; //index for current information (vals 1-2)
deb debounce[5];
option_menu menu; //Globally Defining Menu data
int charg_status; //0 refers to a non-charging state
int gen_status; // 0 refers to the generator being off
int soc; // 0 refers to completely discharged; 100 refers to completely
charged
int crit_status;
int crit_count[20];

void LCD_init(void)
{
  trisd = trisd & 0x0f;  //Top 4 pins configured for output
  trise = trise & 0xf8;  //Bottom 3 pins configured for output
  adcon1 |= 0x0f;             // port E digital mode

  delay_ms(40);

  LCD_icmd( 0x30 ) ;          /*  8 Bit mode  */
  delay_ms(5);                // DH - min delay here of 4.1 ms
  LCD_icmd( 0x30 ) ;          /*  8 Bit mode  */
  LCD_icmd( 0x30 ) ;          /*  8 Bit mode  */

  LCD_icmd( 0x20 ) ;          /*  4-BIT Mode  */
  LCD_cmd( 0x28 );            /*  Function Set: 4-bit,2-line,5X7 */

  LCD_cmd( 0x0C );            /*  Display on, Cursor off         */
  LCD_cmd( 0x06 );            /*  Entry mode: INC addr, NO SHIFT display */

  LCD_cmd( 0x01 );            /*  Clear Display */
  delay_ms( 20 );
}


/*******************************************************
 *  LCD_cmd - Write ASCII character to LCD peripheral  *
 *      configured as a 4 bit interface                *
 *******************************************************/
void LCD_char(char data)
{
  char dsave;
  dsave = latd;
  LCD_DATA = (data & 0xF0) | (LCD_DATA & 0x0F);         /* Write HI nibble
word to LCD */
  write_data;
  LCD_DATA = ((data << 4) & 0xF0) | (LCD_DATA & 0x0F);  /* Write LO nibble
word to LCD */
  write_data;
  delay_ms(1);
  latd = dsave;
}
/*******************************************************
 *  LCD_icmd - Write control word to LCD peripheral  *
 *******************************************************/
```

```
void LCD_icmd(char data)
{
  char dsave;
  dsave = portd;
  LCD_DATA = data | (LCD_DATA & 0x0F) ;
  write_cmd;

  delay_ms(1);
  portd = dsave;
  return;
}

/*************************************************
 *  LCD_cmd - Write control word to LCD peripheral *
 *      configured as a 4 bit interface            *
 *************************************************/
void LCD_cmd(char data)
{
  char dsave;
  dsave = portd;
  LCD_DATA = (data & 0xF0) | (LCD_DATA & 0x0F);         /* Write HI nibble
word to LCD */
  write_cmd;
  LCD_DATA = ((data << 4) & 0xF0) | (LCD_DATA & 0x0F);  /* Write LO nibble
word to LCD */
  write_cmd;
  if (data < 0x03) delay_ms(20); else delay_ms(1);
  portd = dsave;
  return;

}
/* set the cursor position */

void LCD_setpos(char line, char pos)
{
  LCD_cmd(0x80 + line * 0x40 + pos);
  return;
}

/**********************
 * routine to clear display screen
 * */

void LCD_clear(void)
{
  LCD_cmd( 0x01 );            /*  Clear Display */
  delay_ms(20);
  return;
}

/*******************************
 * Display char in HEX - format *
 *******************************/
void LCD_hex(char data)
{
  char n;
  n = ((data >> 4) & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  n = (data & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  return;
}
```

```
/*******************************
 * Display unsigned short in HEX - format *
 *******************************/
void LCD_hex(unsigned short   data)
{
  char n;
  n = ((data >> 12) & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  n = ((data >> 8) & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  n = ((data >> 4) & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  n = (data & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  return;
}
/*******************************
 * Display  short in HEX - format *
 *******************************/
void LCD_hex(short data)
{
  char n;
  n = ((data >> 12) & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  n = ((data >> 8) & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  n = ((data >> 4) & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  n = (data & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  return;
}
/*******************************
 * Display low nibble in HEX - format *
 *******************************/
void LCD_nib(char data)
{
  char n;
  n = (data & 0x0F) + 0x30;
  if (n > 0x39) n = n+7;
  LCD_char(n);
  return;
}

/*******************************
 * Display byte in binary - format *
 *******************************/
void LCD_bin(char data)
{
  char n;
  char temp;
  temp =0x80;
  for( n=1; n<=8; ++n)
  {
    if(temp & data)
      LCD_char(0x31);
```

```
    else
      LCD_char(0x30);
    temp = temp >> 1;
  }
  return;
}
/*******************************
 * Display unsigned short in binary - format *
 *******************************/
void LCD_bin(unsigned short data)
{
  char n;
  unsigned short temp;
  temp =0x8000;
  for( n=1; n<=16; ++n)
  {
    if(temp & data)
      LCD_char(0x31);
    else
      LCD_char(0x30);
    temp = temp >> 1;
  }
}

void LCD_dec(short dat)
{
  unsigned short val;   // ascii results
  unsigned short temp;
  unsigned short div;
  unsigned short data;
  char i;
  char digit;
  data = dat;  // make it unsigned
  div = 10000;
  for(i=0; i <= 4; ++i)   // get all 5 digits
  {
    val = data/div;       // get most signif. digit
    LCD_char(val + '0');    // print digit
    data -= val * div;   // what we've printed
    div=div/10;          // adjust divisor
  }
  return;
}

void LCD_dec(unsigned short dat)
{
  unsigned short val;   // ascii results
  unsigned short temp;
  unsigned short div;
  unsigned short data;
  char i;
  char digit;
  data = dat;  // make it unsigned
  div = 10000;
  for(i=0; i <= 4; ++i)   // get all 5 digits
  {
    val = data/div;       // get most signif. digit
    LCD_char(val + '0');    // print digit
    data -= val * div;   // what we've printed
    div=div/10;          // adjust divisor
  }
  return;
}
/*******************************
```

```
 * Display byte as decimal number *
 *********************************/
void LCD_dec(char data)
{
  LCD_char(((data /100) & 255) + 0x30);
  data = data % 100;
  LCD_char( ((data / 10) & 255) + 0x30 );
  LCD_char( ((data % 10) & 255) + 0x30 );
  return;
}
/*
 * display char as a signed int
 * */

void LCD_int(char data)
{
  char val;   // ascii results
  char temp;
  char div;
  char i;
  bool dozero;
  dozero = false;
  div = 100;
  if(data == 0)  // always print 0
  {
    LCD_char('0');
    return;
  }
    /*
     * adjust for sign
     * */
  if(data & 10000000b)
  {
    LCD_char('-');
    data = ~data + 1;
  }
  for(i=0; i <= 2; ++i)   // get all 3 digits
  {
    val = data/div;       // get most signif. digit
    if(val != 0 || dozero)
    {
      LCD_char(val + '0');    // print digit
      dozero = true;
    }
    temp = val * div;   // subtract off
    data = data - temp;   // what we've printed
    div=div/10;         // adjust divisor
  }
}
/*
 * display unsigned short as a signed int
 * */

void LCD_int(short dat)
{
  short val;   // ascii results
  short temp;
  short div;
  char i;
  unsigned short data;
  bool dozero;
  dozero = false;
  data = dat;
  if(dat == 0)  // always print 0
```

```
  {
    LCD_char('0');
    return;
  }
  div = 10000;
    /*
     * adjust for sign
     * */
  if(dat < 0)
  {
    LCD_char('-');
    data = ~data + 1;
  }
  for(i=0; i <= 4; ++i)   // get all 3 digits
  {
    val = data/div;       // get most signif. digit
    if(val != 0 || dozero)
    {
      LCD_char(val + '0');    // print digit
      dozero = true;
    }
    data -= val*div;   // subtract off what we've printed
    div=div/10;         // adjust divisor
  }
}

void LCD_printf( const char* text )
{
  char i = 0;
  while( text[i] != 0 )
    LCD_char( text[i++] );
  return;
}

void INTRO(void)
{
      const char* l1;
      const char* l2;

      menu.position = 1; // initialization of menu variables
      menu.active_program = 0;      // initialization of menu variables
      menu.active_loc = 0; //Top
      crit_status = 0;

      l1 = "Hybrid's Angels";
      l2 = "Revision: 04/29/09";

      LCD_setpos(0,0);
      LCD_printf(l1);
      LCD_setpos(1,0);
      LCD_printf(l2);
      for (int i = 0; i<=20; i++)
      {
            delay_ms(200);
            LCD_cmd(0x18);  // shift display left
      }
      LCD_clear();
}

void LCD_Voltage(int seg1, int seg2)
{
      int i = 0;
      int j = 0;
      int stpos = 12;
```

```
	if (seg1 == 6)
	{
		seg2 = 1;
	}
	else
	{
		seg2 = seg1+1;
	}

	LCD_setpos(0,0);
	LCD_printf("Battery ");
	LCD_setpos(0,8);
	LCD_char('0' + seg1);
	LCD_setpos(0,9);
	LCD_char(':');
	LCD_setpos(0,10);
	LCD_char(' ');
	LCD_setpos(0,11);
	LCD_char(' ');

	for (j = 0; j<=5; j++) //Displays voltage while taking out insignificant
digits
	{
		if (j == 0 || j == 1)
		{
			if (bat[seg1].volt[j] == '0')
			{
				if (j == 1)
				{
					if (bat[seg1].volt[0] == '0')
					{
						LCD_setpos(0,stpos+5-j);
						LCD_char(' ');
					}
					else
					{
						LCD_setpos(0,stpos+0);
						LCD_char(bat[seg1].volt[0]);
						LCD_setpos(0,stpos+1);
						LCD_char(bat[seg1].volt[1]);
						i = 2;
					}
				}
				else
				{
					LCD_setpos(0,stpos+5-j);
					LCD_char(' ');
				}
			}
			else
			{
				LCD_setpos(0,stpos+i);
				LCD_char(bat[seg1].volt[j]);
				i++;
			}
		}
		else
		{
			LCD_setpos(0,stpos+i);
			LCD_char(bat[seg1].volt[j]);
			i++;
		}
	}
```

```
	if (bat[seg1].volt[0] == '0')
	{
		if (bat[seg1].volt[1] == '0')
		{
			LCD_setpos(0,stpos+4);
			LCD_char(' ');
			LCD_setpos(0,stpos+5);
			LCD_char('V');
			LCD_setpos(0,stpos+6);
			LCD_char(' ');
			LCD_setpos(0,stpos+7);
			LCD_char(' ');
		}
		else
		{
			LCD_setpos(0,stpos+5);
			LCD_char(' ');
			LCD_setpos(0,stpos+6);
			LCD_char('V');
			LCD_setpos(0,stpos+7);
			LCD_char(' ');
		}
	}
	else
	{
		LCD_setpos(0,stpos+6);
		LCD_char(' ');
		LCD_setpos(0,stpos+7);
		LCD_char('V');
	}

	LCD_setpos(1,0);
	LCD_printf("Battery ");
	LCD_setpos(1,8);
	LCD_char('0' + seg2);
	LCD_setpos(1,9);
	LCD_char(':');
	LCD_setpos(1,10);
	LCD_char(' ');
	LCD_setpos(1,11);
	LCD_char(' ');
	i = 0;

	for (j = 0; j<=5; j++) //Displays voltage while taking out insignificant
digits
	{
		if (j == 0 || j == 1)
		{
			if (bat[seg2].volt[j] == '0')
			{
				if (j == 1)
				{
					if (bat[seg2].volt[0] == '0')
					{
						LCD_setpos(1,stpos+5-j);
						LCD_char(' ');
					}
					else
					{
						LCD_setpos(1,stpos+0);
						LCD_char(bat[seg2].volt[0]);
						LCD_setpos(1,stpos+1);
						LCD_char(bat[seg2].volt[1]);
```

```
                                    i = 2;
                              }
                        }
                        else
                        {
                              LCD_setpos(1,stpos+5-j);
                              LCD_char(' ');
                        }
                  }
                  else
                  {
                        LCD_setpos(1,stpos+i);
                        LCD_char(bat[seg2].volt[j]);
                        i++;
                  }
            }
            else
            {
                  LCD_setpos(1,stpos+i);
                  LCD_char(bat[seg2].volt[j]);
                  i++;
            }
      }

      if (bat[seg2].volt[0] == '0')
      {
            if (bat[seg2].volt[1] == '0')
            {
                  LCD_setpos(1,stpos+4);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+5);
                  LCD_char('V');
                  LCD_setpos(1,stpos+6);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+7);
                  LCD_char(' ');
            }
            else
            {
                  LCD_setpos(1,stpos+5);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+6);
                  LCD_char('V');
                  LCD_setpos(1,stpos+7);
                  LCD_char(' ');
            }
      }
      else
      {
            LCD_setpos(1,stpos+6);
            LCD_char(' ');
            LCD_setpos(1,stpos+7);
            LCD_char('V');
      }
}

void LCD_Temp(int seg1, int seg2)
{
      int i = 0;
      int j = 0;
      int stpos = 12;

      if (seg1 == 6)
      {
```

```
            seg2 = 1;
      }
      else
      {
            seg2 = seg1+1;
      }

      LCD_setpos(0,0);
      LCD_printf("Battery ");
      LCD_setpos(0,8);
      LCD_char('0' + seg1);
      LCD_setpos(0,9);
      LCD_char(':');
      LCD_setpos(0,10);
      LCD_char(' ');
      LCD_setpos(0,11);
      LCD_char(' ');

      for (j = 0; j<=5; j++) //Displays voltage while taking out insignificant
digits
      {
            if (j == 0 || j == 1)
            {
                  if (bat[seg1].temp[j] == '0')
                  {
                        if (j == 1)
                        {
                              if (bat[seg1].temp[0] == '0')
                              {
                                    LCD_setpos(0,stpos+5-j);
                                    LCD_char(' ');
                              }
                              else
                              {
                                    LCD_setpos(0,stpos+0);
                                    LCD_char(bat[seg1].temp[0]);
                                    LCD_setpos(0,stpos+1);
                                    LCD_char(bat[seg1].temp[1]);
                                    i = 2;
                              }
                        }
                        else
                        {
                              LCD_setpos(0,stpos+5-j);
                              LCD_char(' ');
                        }
                  }
                  else
                  {
                        LCD_setpos(0,stpos+i);
                        LCD_char(bat[seg1].temp[j]);
                        i++;
                  }
            }
            else
            {
                  LCD_setpos(0,stpos+i);
                  LCD_char(bat[seg1].temp[j]);
                  i++;
            }
      }

      if (bat[seg1].temp[0] == '0')
      {
```

```
            if (bat[seg1].temp[1] == '0')
            {
                  LCD_setpos(0,stpos+4);
                  LCD_char(' ');
                  LCD_setpos(0,stpos+5);
                  LCD_char('F');
                  LCD_setpos(0,stpos+6);
                  LCD_char(' ');
                  LCD_setpos(0,stpos+7);
                  LCD_char(' ');
            }
            else
            {
                  LCD_setpos(0,stpos+5);
                  LCD_char(' ');
                  LCD_setpos(0,stpos+6);
                  LCD_char('F');
                  LCD_setpos(0,stpos+7);
                  LCD_char(' ');
            }
      }
      else
      {
            LCD_setpos(0,stpos+6);
            LCD_char(' ');
            LCD_setpos(0,stpos+7);
            LCD_char('F');
      }

      LCD_setpos(1,0);
      LCD_printf("Battery ");
      LCD_setpos(1,8);
      LCD_char('0' + seg2);
      LCD_setpos(1,9);
      LCD_char(':');
      LCD_setpos(1,10);
      LCD_char(' ');
      LCD_setpos(1,11);
      LCD_char(' ');
      i = 0;

      for (j = 0; j<=5; j++) //Displays voltage while taking out insignificant
digits
      {
            if (j == 0 || j == 1)
            {
                  if (bat[seg2].temp[j] == '0')
                  {
                        if (j == 1)
                        {
                              if (bat[seg2].temp[0] == '0')
                              {
                                    LCD_setpos(1,stpos+5-j);
                                    LCD_char(' ');
                              }
                              else
                              {
                                    LCD_setpos(1,stpos+0);
                                    LCD_char(bat[seg2].temp[0]);
                                    LCD_setpos(1,stpos+1);
                                    LCD_char(bat[seg2].temp[1]);
                                    i = 2;
                              }
                        }
```

```
                        else
                        {
                              LCD_setpos(1,stpos+5-j);
                              LCD_char(' ');
                        }
                  }
                  else
                  {
                        LCD_setpos(1,stpos+i);
                        LCD_char(bat[seg2].temp[j]);
                        i++;
                  }
            }
            else
            {
                  LCD_setpos(1,stpos+i);
                  LCD_char(bat[seg2].temp[j]);
                  i++;
            }
      }

      if (bat[seg2].temp[0] == '0')
      {
            if (bat[seg2].temp[1] == '0')
            {
                  LCD_setpos(1,stpos+4);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+5);
                  LCD_char('F');
                  LCD_setpos(1,stpos+6);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+7);
                  LCD_char(' ');
            }
            else
            {
                  LCD_setpos(1,stpos+5);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+6);
                  LCD_char('F');
                  LCD_setpos(1,stpos+7);
                  LCD_char(' ');
            }
      }
      else
      {
            LCD_setpos(1,stpos+6);
            LCD_char(' ');
            LCD_setpos(1,stpos+7);
            LCD_char('F');
      }
}

void LCD_Current()
{
      int i = 0;
      int j = 0;
      int stpos = 11;

      LCD_setpos(0,0);
      LCD_printf("Motor:     ");

      cur_index = 1;
```

```
      for (j = 0; j<=6; j++) //Displays voltage while taking out insignificant
digits
      {
            if (j == 0 || j == 1)
            {
                  if (cur[cur_index].current[j] == '0')
                  {
                        if (j == 1)
                        {
                              if (cur[cur_index].current[0] == '0')
                              {
                                    LCD_setpos(0,stpos+6-j);
                                    //LCD_char(' ');
                              }
                              else
                              {
                                    LCD_setpos(0,stpos+0);
                                    LCD_char(cur[cur_index].current[0]);
                                    LCD_setpos(0,stpos+1);
                                    LCD_char(cur[cur_index].current[1]);
                                    i = 2;
                              }
                        }
                        else
                        {
                              LCD_setpos(0,stpos+6-j);
                              //LCD_char(' ');
                        }
                  }
                  else
                  {
                        LCD_setpos(0,stpos+i);
                        LCD_char(cur[cur_index].current[j]);
                        i++;
                  }
            }
            else
            {
                  LCD_setpos(0,stpos+i);
                  LCD_char(cur[cur_index].current[j]);
                  i++;
            }
      }

      if (cur[cur_index].current[0] == '0')
      {
            if (cur[cur_index].current[1] == '0')
            {
                  LCD_setpos(0,stpos+5);
                  LCD_char(' ');
                  LCD_setpos(0,stpos+6);
                  LCD_char('A');
                  LCD_setpos(0,stpos+7);
                  LCD_char(' ');
                  LCD_setpos(0,stpos+8);
                  LCD_char(' ');
            }
            else
            {
                  LCD_setpos(0,stpos+6);
                  LCD_char(' ');
                  LCD_setpos(0,stpos+7);
                  LCD_char('A');
                  LCD_setpos(0,stpos+8);
```

```
                LCD_char(' ');
            }
        }
        else
        {
            LCD_setpos(0,stpos+7);
            LCD_char(' ');
            LCD_setpos(0,stpos+8);
            LCD_char('A');
        }

        LCD_setpos(1,0);
        LCD_printf("Battery:   ");

        cur_index = 2;
        i=0;

        for (j = 0; j<=6; j++) //Displays voltage while taking out insignificant
digits
        {
            if (j == 0 || j == 1)
            {
                if (cur[cur_index].current[j] == '0')
                {
                    if (j == 1)
                    {
                        if (cur[cur_index].current[0] == '0')
                        {
                            LCD_setpos(1,stpos+6-j);
                            //LCD_char(' ');
                        }
                        else
                        {
                            LCD_setpos(1,stpos+0);
                            LCD_char(cur[cur_index].current[0]);
                            LCD_setpos(1,stpos+1);
                            LCD_char(cur[cur_index].current[1]);
                            i = 2;
                        }
                    }
                    else
                    {
                        LCD_setpos(1,stpos+6-j);
                        //LCD_char(' ');
                    }
                }
                else
                {
                    LCD_setpos(1,stpos+i);
                    LCD_char(cur[cur_index].current[j]);
                    i++;
                }
            }
            else
            {
                LCD_setpos(1,stpos+i);
                LCD_char(cur[cur_index].current[j]);
                i++;
            }
        }

        if (cur[cur_index].current[0] == '0')
        {
            if (cur[cur_index].current[1] == '0')
```

```
            {
                  LCD_setpos(1,stpos+5);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+6);
                  LCD_char('A');
                  LCD_setpos(1,stpos+7);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+8);
                  LCD_char(' ');
            }
            else
            {
                  LCD_setpos(1,stpos+6);
                  LCD_char(' ');
                  LCD_setpos(1,stpos+7);
                  LCD_char('A');
                  LCD_setpos(1,stpos+8);
                  LCD_char(' ');
            }
      }
      else
      {
            LCD_setpos(1,stpos+7);
            LCD_char(' ');
            LCD_setpos(1,stpos+8);
            LCD_char('A');
      }
      cur_index = 1;
}

void relay_init(void)
{
      trisa.2 = 1;
      trisa.4 = 1;
      trisb.7 = 0;

      volatile bit relay@PORTB.7 = 0;
}

//*************************************************\\
//             SPI Section of the Program          \\
//*************************************************\\

void SPI_init(void)
{
  // Prepare SSPSTAT (SPI status register)
  sspstat.7 = 0;        // SMP sample time, 1 samples at end of data output
time
  sspstat.6 = 1;        // SPI Clock Select, transmit on transition from
active to idle

  sspcon1 = 0x20;

  // Prepare RC3,RC4,RC5 for SPI use
  trisc.3 = 0;   // SCLK, SPI mode
  trisc.4 = 1;     // SDO (serial data out), SPI mode
  trisc.5 = 0;   // SDI (serial data in), SPI mode

  //Initialize SPI (CS!)
  trisb.2 = 0;    // (CS!) for A/D Converter #1; set B2 as output
  volatile bit b2high@PORTB.2 = 1;  //Set B2 high
  trisb.3 = 0;    // (CS!) for A/D Converter #2; set B3 as output
  volatile bit b3high@PORTB.3 = 1;  //Set B3 high
  trisb.4 = 0;    // (CS!) for A/D Converter #2; set B4 as output
```

```
  volatile bit b4high@PORTB.4 = 1;  //Set B4 high
}

void SPI_bat_init(void)
{
  // Prepare SSPSTAT (SPI status register)
  sspstat.7 = 0;        // SMP sample time, 1 samples at end of data output
time
  sspstat.6 = 1;        // SPI Clock Select, transmit on transition from
active to idle

  sspcon1 = 0x20;

  // Prepare RC3,RC4,RC5 for SPI use
  trisc.3 = 0;   // SCLK, SPI mode
  trisc.4 = 1;     // SDO (serial data out), SPI mode
  trisc.5 = 0;   // SDI (serial data in), SPI mode

  //Initialize SPI (CS!)
  trisb.4 = 0;    // (CS!) for A/D Converter #1; set B4 as output
  volatile bit b4high@PORTB.4 = 1;  //Set B4 high

  //Initialize State Signals
  trisd.1 = 1; //Initializes input signal from control circuit
  trisd.0 = 0; //Initializes output signal to control circuit
}

void SPI_CS(char micro, char device, bool choice)
{
  // By default, every time this function is called, set all chip
  // select pins high.  Only one pin can be set low (only one device selected)
  // at any given time.
  unsigned char cs_delay = 1;

  switch (micro)
  {
    case 1:

      volatile bit AD_conv1@PORTB.2 = 1;
      volatile bit AD_conv2@PORTB.3 = 1;
      volatile bit AD_conv3@PORTB.4 = 1;

      // device = 1, selects A/D converter #1
      // device = 2, selects A/D converter #2
      // device = 3, selects A/D converter #3

      switch (device)
      {
        case 1:
          AD_conv1 = choice;
          break;
        case 2:
          AD_conv2 = choice;
          break;
            case 3:
                  AD_conv3 = choice;
                  break;
      }
  }
  delay_us(cs_delay);         // Allow for !CS setup time
}

void SPI_send(char data)
{
```

```
  sspcon1.7 = 0;
  volatile bit spi_bf@SSPSTAT = 0x0f;
  nop();
  nop();
  nop();
  sspbuf = data;                          // write input data to SSPBUF
  while (!spi_bf) {}
  nop();
  delay_us(1);
}

//*****************************************************\\
//          Analog to Digital Channel Select           \\
//*****************************************************\\

void AD_sample(char channel, char micro, char device, unsigned int &data)
{
  unsigned char FIRST_bit = 6;
  unsigned char SECOND_bit = 0;
  unsigned char THIRD_bit = 0;

  data = 0;

  switch (channel)
  {
    case 0:
      break;
    case 1:
      SECOND_bit.6 = 1;
      break;
    case 2:
      SECOND_bit.7 = 1;
      break;
    case 3:
      SECOND_bit.7 = 1;
      SECOND_bit.6 = 1;
      break;
    case 4:
      FIRST_bit.0 = 1;
      break;
    case 5:
      FIRST_bit.0 = 1;
      SECOND_bit.6 = 1;
      break;
    case 6:
      FIRST_bit.0 = 1;
      SECOND_bit.7 = 1;
      break;
    case 7:
      FIRST_bit.0 = 1;
      SECOND_bit.7 = 1;
      SECOND_bit.6 = 1;
      break;
  }
  SPI_CS(micro, device, 0);               // Select the A/D converter
  SPI_send(FIRST_bit);                    // Send the first 8 bits
  SPI_send(SECOND_bit);                   // Send the second 8 bits
  data.11 = sspbuf.3;                     // Read SSPBUF data into result
variable
  data.10 = sspbuf.2;                     // Read SSPBUF data into result
variable
  data.9 = sspbuf.1;                      // Read SSPBUF data into result
variable
```

```
  data.8 = sspbuf.0;                        // Read SSPBUF data into result
variable
  SPI_send(THIRD_bit);
  /*data.7 = sspbuf.7;                      // Read SSPBUF data into result
variable
  data.6 = sspbuf.6;                        // Read SSPBUF data into result
variable
  data.5 = sspbuf.5;                        // Read SSPBUF data into result
variable
  data.4 = sspbuf.4;                        // Read SSPBUF data into result
variable
  data.3 = sspbuf.3;                        // Read SSPBUF data into result
variable
  data.2 = sspbuf.2;                        // Read SSPBUF data into result
variable
  data.1 = sspbuf.1;                        // Read SSPBUF data into result
variable
  data.0 = sspbuf.0;                        // Read SSPBUF data into result
variable
  */
  data = data + sspbuf;
  //delay_us(5);
  SPI_CS(micro, device, 1);                 // Finished with channel
  //delay_us(5);
}

//*************************************************\\
//             Battery Analysis Section            \\
//*************************************************\\

void bat_volt_convert(void)
{
      long conv[7];
      long temp[7];
      long dataval;
      unsigned int i = 0;
      long dec = 100; // will indicate number of decimal places; i.e. 100 => 2
decimals
      long opamp_num = 39;
      long opamp_den = 1000;
      battery work;
      //index = 3; // testing the function call

      if (bat[index].data >= 4095)
      {
            work.data = 2;
      }
      else if (bat[index].data >=10)
      {
            work.data = bat[index].data -10;
      }
      else
      {
            work.data = 0;
      }

      dataval = (long)work.data;
      conv[i] = dataval*5*dec*opamp_den/opamp_num/4096; // always multiply
before dividing
      work.voltage = conv[i];
      bat[index].volt_val = conv[i];

      for (char d=0;d<=6;d++)
      {
```

```
            if (d > 5)
            {
                  work.volt[6] = 0;
            }
            else if (d!=2)
            {
                  temp[d] = conv[d];
                  conv[d] = conv[d] - (int)(conv[d]/10)*10;
                  work.volt[5-d] = '0' + (int)conv[d];
                  conv[d+1] = temp[d]/10;
            }
            else
            {
                        work.volt[5-d] = '.';
                        conv[d+1] = conv[d];
            }

      }
      bat[index] = work;
}

void bat_temp_convert(void)
{
      long conv[7];
      long temp[7];
      long dataval;
      long conv_val = 100; // 10 mV/degree C
      unsigned int i = 0;
      long dec = 100; // will indicate number of decimal places; i.e. 100 => 2
decimals

      if (bat[index].temp_data >= 4095)
      {
            bat[index].temp_data = 2;
      }
      else if (bat[index].temp_data >=10)
      {
            bat[index].temp_data = bat[index].temp_data - 2;
      }
      else
      {
            bat[index].temp_data = 0;
      }

      dataval = (long)bat[index].temp_data;
      conv[i] = dataval*(9*dec)*conv_val/4096+3200; // always multiply before
dividing
      bat[index].temperature = conv[i];
      bat[index].temp_val = conv[i];

      for (char d=0;d<=6;d++)
      {
            if (d > 5)
            {
                  bat[index].temp[6] = 0;
            }
            else if (d!=2)
            {
                  temp[d] = conv[d];
                  conv[d] = conv[d] - (int)(conv[d]/10)*10;
                  bat[index].temp[5-d] = '0' + (int)conv[d];
                  conv[d+1] = temp[d]/10;
            }
            else
```

```
            {
                        bat[index].temp[5-d] = '.';
                        conv[d+1] = conv[d];
            }

      }
}

void ind_sample(void)
{
      unsigned int z = 0;
      int count = 0;
      long temp = 0;
      unsigned int helper = 0;
      int pos;

      temp = 0;
      helper = 0;
      pos = index - 1;
      while (1)
      {
            AD_sample(pos,1,1,helper);
            if (helper << 30)
            {
                  AD_sample(pos,1,1,helper);
                  z++;
            }
            else
            {
                  z++;
            }
            if (z >=2)
            {
                  z = 0;
                  break;
            }
      }
      bat[index].data = helper;
      bat_volt_convert();

      while (1)
      {
            AD_sample(pos,1,2,helper);
            if (helper << 30)
            {
                  AD_sample(pos,1,2,helper);
                  z++;
            }
            else
            {
                  z++;
            }
            if (z >=2)
            {
                  z = 0;
                  break;
            }
      }
      bat[index].temp_data = helper;
      bat_temp_convert();
}

/*void critical(void)
{
```

```
	if (bat[6].voltage >= 9000)
	{
		crit_status = 8;
		crit_count[8]++;
	}
	else if (bat[6].voltage <= 6500)
	{
		crit_status = 7;
		crit_count[7]++;
	}
	else if ((bat[6].voltage - bat[5].voltage) <= 1000)
	{
		crit_status = 6;
		crit_count[6]++;
	}
	else if ((bat[5].voltage - bat[4].voltage) <= 1000)
	{
		crit_status = 5;
		crit_count[5]++;
	}
	else if ((bat[4].voltage - bat[3].voltage) <= 1000)
	{
		crit_status = 4;
		crit_count[4]++;
	}
	else if ((bat[3].voltage - bat[2].voltage) <= 1000)
	{
		crit_status = 3;
		crit_count[3]++;
	}
	else if ((bat[2].voltage - bat[1].voltage) <= 1000)
	{
		crit_status = 2;
		crit_count[2]++;
	}
	else if ((bat[1].voltage) <= 1000)
	{
		crit_status = 1;
		crit_count[1]++;
	}
	else
	{
		crit_status = 0;
	}
	if (crit_status != 0 && crit_count[crit_status] >= 60000)
	{
			crit_count[status] = 0;
			menu.active_program = 5;
	}
}*/

void sample_all(void)
{
	index = 1;
	ind_sample();
	index = 2;
	ind_sample();
	index = 3;
	ind_sample();
	index = 4;
	ind_sample();
	index = 5;
	ind_sample();
	index = 6;
```

```
	ind_sample();
	index = 1;
	current_sample();
}

void current_sample(void)
{
	int z = 0;
	unsigned int helper = 0;

	while (1) // sample motor current
	{
		AD_sample(6,1,1,helper);
		if (cur[1].data << 30)
		{
			AD_sample(6,1,1,helper);
			z++;
		}
		else
		{
			z++;
		}
		if (z >=2)
		{
			z = 0;
			break;
		}
	}
	cur_index = 1;
	cur[cur_index].data = helper;
	current_convert();
	helper = 0;

	while (1) // sample battery current
	{
		AD_sample(7,1,1,helper);
		if (cur[1].data << 30)
		{
			AD_sample(7,1,1,helper);
			z++;
		}
		else
		{
			z++;
		}
		if (z >=2)
		{
			z = 0;
			break;
		}
	}
	cur_index = 2;
	cur[cur_index].data = helper;
	current_convert();
	helper = 0;
	cur_index = 1;
}

void current_convert(void)
{
	long conv[7];
	long temp[7];
	long dataval;
	long conv_val = 400; // 40 mV/A
```

```
      unsigned int i = 0;
      long dec = 1000; // will indicate number of decimal places; i.e. 100 =>
2 decimals
      long def_cur = 2500; // Voltage at 0A (2.5V)


      dataval = (long)cur[cur_index].data;
      conv[i] = dataval*dec/819;
      conv[i] = def_cur - conv[i];
      conv[i] = conv[i]*1000/40;
      cur[cur_index].current_val = conv[i];
      cur[cur_index].cur_val = conv[i];



      for (char d=0;d<=6;d++)
      {
            if (d!=3)
            {
                  temp[d] = conv[d];
                  conv[d] = conv[d] - (int)(conv[d]/10)*10;
                  cur[cur_index].current[6-d] = '0' + (int)conv[d];
                  conv[d+1] = temp[d]/10;
            }
            else
            {
                        cur[cur_index].current[6-d] = '.';
                        conv[d+1] = conv[d];
            }
      }
}

int check_soc(void)
{
      long bat6volt;
      bat6volt = bat[6].voltage;
      long min_volt = 6500;
      long max_volt = 9000;
      int percent;

      if (charg_status == 1)
      {
            percent = 101;
      }
      else
      {
            if (bat6volt <= 6500)
            {
                  percent = 0;
            }
            else if (bat6volt >= 9000)
            {
                  percent = 100;
            }
            else
            {
                  percent = (percent - min_volt)*100/(max_volt - min_volt);
            }
      }
      //percent = 30;
      return percent;
}

//************************************************\\
```

```
//               User Interface Section                \\
//*****************************************************\\

void check_ui(option_menu &menu)
{
      int j = 1;
      unsigned int data;
      int def_level = 2000;
      deb test[5];

      for (j = 1; j <=4; j++)
      {
            debounce[j].prev = debounce[j].status;
            debounce[j].status = 0;
      }

      AD_sample(0,1,3,data); // Sample Up
      debounce[1].data[0] = data;
      AD_sample(1,1,3,data); // Sample Down
      debounce[2].data[0] = data;
      AD_sample(2,1,3,data); // Sample Back
      debounce[3].data[0] = data;
      AD_sample(3,1,3,data); // Sample Return
      debounce[4].data[0] = data;

      delay_ms(20);

      AD_sample(0,1,3,data); // 2nd Sample Up
      debounce[1].data[1] = data;
      AD_sample(1,1,3,data); // 2nd Sample Down
      debounce[2].data[1] = data;
      AD_sample(2,1,3,data); // 2nd Sample Back
      debounce[3].data[1] = data;
      AD_sample(3,1,3,data); // 2nd Sample Return
      debounce[4].data[1] = data;

      if (debounce[1].data[0] >= def_level && debounce[1].data[1] >=def_level)
      {
            debounce[1].status = 1;
      }
      if (debounce[1].prev == 1 && debounce[1].status == 0)
      {
            if (menu.active_loc == 0)
            {
                  menu.active_loc = 0;
                  menu.position--;
                  if (menu.position == 0)
                  {
                        menu.position = menu.total;
                  }
                  else
                  {
                  }
            }
            else
            {
                  menu.position--;
                  menu.active_loc = 0;
                  if (menu.position == 0)
                  {
                        menu.position = menu.total;
                  }
                  else
                  {
```

```
                }
            }
        }

        if (debounce[2].data[0] >= def_level && debounce[2].data[1] >=def_level)
        {
            debounce[2].status = 1;
        }
        if (debounce[2].prev == 1 && debounce[2].status == 0)
        {
            if (menu.active_loc == 0)
            {
                menu.position++;
                menu.active_loc = 1;
                if (menu.position >> menu.total)
                {
                    menu.position = 1;
                }
                else
                {
                }
            }
            else
            {
                menu.active_loc = 1;
                menu.position++;
                if (menu.position >> menu.total)
                {
                    menu.position = 1;
                }
                else
                {
                }
            }

        }

        if (debounce[3].data[0] <= def_level && debounce[3].data[1] <=def_level)
        {
            debounce[3].status = 1;
        }
        if (debounce[3].prev == 1 && debounce[3].status == 0)  // Detect Back
        {
                menu.position = menu.active_program;
                menu.active_program = 0;
                menu.prev_program = 0;
                menu.active_loc = menu.prev_loc;
        }

        if (debounce[4].data[0] <= def_level && debounce[4].data[1] <=
def_level)
        {
            debounce[4].status = 1;
        }
        if (debounce[4].prev == 1 && debounce[4].status == 0) // Detect
Enter/Return
        {
            if (menu.prev_program == 0 && menu.active_program == 0)
            {
                menu.prev_program = menu.active_program;
                menu.active_program = menu.position;
                menu.position = 1;
                menu.prev_loc = menu.active_loc;
                menu.active_loc = 0;
```

```
            }
            else
            {
            }
      }
}

void main_menu(void)
{
      const char* men0 = "Main Menu";
      const char* men1 = "1) System Status   ";
      const char* men2 = "2) Battery Volt.   ";
      const char* men3 = "3) Current Info    ";
      const char* men4 = "4) Temp. Sensors   ";

      menu.total = 4; // total number of options in this menu

      switch(menu.position)
      {
            case 1:
                  if (menu.active_loc == 0)
                  {
                        LCD_setpos(0,0);
                        LCD_printf(men1);
                        LCD_setpos(1,0);
                        LCD_printf(men2);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc+1,19);
                        LCD_char(' ');
                  }
                  else
                  {
                        LCD_setpos(1,0);
                        LCD_printf(men1);
                        LCD_setpos(0,0);
                        LCD_printf(men4);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc-1,19);
                        LCD_char(' ');
                  }
                  break;
            case 2:
                  if (menu.active_loc == 0)
                  {
                        LCD_setpos(0,0);
                        LCD_printf(men2);
                        LCD_setpos(1,0);
                        LCD_printf(men3);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc+1,19);
                        LCD_char(' ');
                  }
                  else
                  {
                        LCD_setpos(0,0);
                        LCD_printf(men1);
                        LCD_setpos(1,0);
                        LCD_printf(men2);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc-1,19);
```

```
                        LCD_char(' ');
                }
                break;
        case 3:
                if (menu.active_loc == 0)
                {
                        LCD_setpos(0,0);
                        LCD_printf(men3);
                        LCD_setpos(1,0);
                        LCD_printf(men4);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc+1,19);
                        LCD_char(' ');
                }
                else
                {
                        LCD_setpos(0,0);
                        LCD_printf(men2);
                        LCD_setpos(1,0);
                        LCD_printf(men3);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc-1,19);
                        LCD_char(' ');
                }
                break;
        case 4:
                if (menu.active_loc == 0)
                {
                        LCD_setpos(0,0);
                        LCD_printf(men4);
                        LCD_setpos(1,0);
                        LCD_printf(men1);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc+1,19);
                        LCD_char(' ');
                }
                else
                {
                        LCD_setpos(0,0);
                        LCD_printf(men3);
                        LCD_setpos(1,0);
                        LCD_printf(men4);
                        LCD_setpos(menu.active_loc,19);
                        LCD_char('<');
                        LCD_setpos(menu.active_loc-1,19);
                        LCD_char(' ');
                }
                break;
        default:
                menu.position = 1;
                break;
        }
}

void check_menu(void)
{
        check_ui(menu);

        switch(menu.active_program)
        {
                case 0:
```

```
                        main_menu();
                        break;
                case 1:
                        status_menu();
                        break;
                case 2:
                        bv_menu();
                        break;
                case 3:
                        current_menu();
                        break;
                case 4:
                        temp_menu();
                        break;
                //case 5:
                        //error_menu();
                        //break;
                default:
                        main_menu();
                        break;
        }
}

void bv_menu(void)
{
        menu.total = 6; // total number of options in this menu

        switch (menu.position)
        {
                case 1:
                        if (menu.active_loc == 0)
                        {
                                LCD_Voltage(1,2);
                        }
                        else
                        {
                                LCD_Voltage(6,1);
                        }
                        break;
                case 2:
                        if (menu.active_loc == 0)
                        {
                                LCD_Voltage(2,3);
                        }
                        else
                        {
                                LCD_Voltage(1,2);
                        }
                        break;
                case 3:
                        if (menu.active_loc == 0)
                        {
                                LCD_Voltage(3,4);
                        }
                        else
                        {
                                LCD_Voltage(2,3);
                        }
                        break;
                case 4:
                        if (menu.active_loc == 0)
                        {
                                LCD_Voltage(4,5);
                        }
```

```
                else
                {
                        LCD_Voltage(3,4);
                }
                break;
        case 5:
                if (menu.active_loc == 0)
                {
                        LCD_Voltage(5,6);
                }
                else
                {
                        LCD_Voltage(4,5);
                }
                break;
        case 6:
                if (menu.active_loc == 0)
                {
                        LCD_Voltage(6,1);
                }
                else
                {
                        LCD_Voltage(5,6);
                }
                break;
        default:
                menu.position = 1;
                break;
    }
}

void status_menu(void)
{
    const char* g0c0 = "Start Generator     ";
    const char* goff = "Generator Off       ";
    const char* soc_title = "Instant. SOC:  ";
    const char* charging = "Currently Charging  ";
    const char* viagen = "from the Generator  ";
    const char* viawall = "from the Wall      ";
    int soc_offset = 15;
    int temp[3];
    int soc_stat[3];

    soc = check_soc();

    if (charg_status == 0)
    {
        LCD_setpos(1,0);
        LCD_printf(soc_title);
        LCD_setpos(1,19);
        LCD_char('%');
        LCD_setpos(1,18);
        LCD_char(' ');
        soc_stat[0] = soc;

        for (int i = 0; i <=2; i++)
        {
            temp[i] = soc_stat[i];
            soc_stat[i] = soc_stat[i] - (soc_stat[i]/10)*10;
            LCD_setpos(1,soc_offset+i);
            if (soc_stat[i] == 0 && i !=2)
            {
                LCD_char(' ');
            }
```

```
                  else
                  {
                        LCD_char('0' + soc_stat[i]);
                  }
                  soc_stat[i+1] = temp[i]/10;
            }

            if (gen_status == 0)
            {
                  if (soc >= 66)
                  {
                        LCD_setpos(0,0);
                        LCD_printf(goff);
                  }
                  else
                  {
                        LCD_setpos(0,0);
                        LCD_printf(g0c0);
                  }
            }
            else
            {
            }
      }
      else
      {
            LCD_setpos(0,0);
            LCD_printf(charging);
            LCD_setpos(1,0);
            if (gen_status == 1)
            {
                  LCD_printf(viagen);
            }
            else
            {
                  LCD_printf(viawall);
            }
      }
}

void current_menu(void)
{
      LCD_Current();
}

void temp_menu(void)
{
      menu.total = 6; // total number of options in this menu

      switch (menu.position)
      {
            case 1:
                  if (menu.active_loc == 0)
                  {
                        LCD_Temp(1,2);
                  }
                  else
                  {
                        LCD_Temp(6,1);
                  }
                  break;
            case 2:
                  if (menu.active_loc == 0)
                  {
```

```
                        LCD_Temp(2,3);
                }
                else
                {
                        LCD_Temp(1,2);
                }
                break;
        case 3:
                if (menu.active_loc == 0)
                {
                        LCD_Temp(3,4);
                }
                else
                {
                        LCD_Temp(2,3);
                }
                break;
        case 4:
                if (menu.active_loc == 0)
                {
                        LCD_Temp(4,5);
                }
                else
                {
                        LCD_Temp(3,4);
                }
                break;
        case 5:
                if (menu.active_loc == 0)
                {
                        LCD_Temp(5,6);
                }
                else
                {
                        LCD_Temp(4,5);
                }
                break;
        case 6:
                if (menu.active_loc == 0)
                {
                        LCD_Temp(6,1);
                }
                else
                {
                        LCD_Temp(5,6);
                }
                break;
        default:
                menu.position = 1;
                break;
    }
}

/*void error_menu(void)
{
    const char* error = "Error Found. Code: ";
    const char* msg;
    LCD_setpos(0,0);
    LCD_printf(error);
    LCD_setpos(0,19);
    LCD_char('0'+crit_status);

    switch(crit_status)
    {
```

```
                case 0:
                msg = "No errors found     ";
                break;
                case 1:
                msg = "<10 V on 12V Battery";
                break;
                case 2:
                msg = "<10 V on 24V Battery";
                break;
                case 3:
                msg = "<10 V on 36V Battery";
                break;
                case 4:
                msg = "<10 V on 48V Battery";
                break;
                case 5:
                msg = "<10 V on 60V Battery";
                break;
                case 6:
                msg = "<10 V on 72V Battery";
                break;
                case 7:
                msg = "<65 V on Stack      ";
                break;
                case 8:
                msg = ">90 V on Stack      ";
                break;
                default:
                msg = "Unknown Error       ";
                break;
        }
        LCD_setpos(1,0);
        LCD_printf(msg);
}*/

void test(void)
{
        //check_ui(menu);
        charg_status = 1;
        gen_status = 1;
        status_menu();
}
```

6.3 REVELANT COMPONENT DATA SHEETS

NUMBER	DESCRIPTION	CHARGING BRD	CONTROL BRD	DATA SHEET
18F46204P	Microcontroller	y	y	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=PIC18F4620-I/P-ND
TC1411NCPA-ND	IC MOSFET DVR 1A HS 8DIP	y		http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=TC1411NCPA-ND
FFPF30UP20STU-ND	DIODE ULTRA FAST 200V TO-220F	y		http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=FFPF30UP20STU-ND
GBJ2502-FDI-ND	RECT BRIDGE GPP 25A 200V GBJ	y		http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=GBJ2502-FDI-ND
620-1237-ND	IC HALL EFFECT SENSOR MOD 5PIN	y		http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=620-1237-ND

4N35MFS-ND	OPTOCOUPLER TRANS-OUT 6-DIP	y		http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=4N35MFS-ND
LM35DT-ND	IC SENSOR PREC CENT TEMP TO-220		y	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=LM35DT-ND
MC78T05CTFS-ND	REGULATOR POS 5V 3A 4% TO220	y	y	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=MC78T05CTFS-ND
MCP3208-CI/P-ND	IC ADC 12BIT 2.7V 8CH SPI 16-DIP	y	y	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=MCP3208-CI/P-ND
Q4015L5-ND	GATE TRIAC ISOLATED 15.0A 400V	y		http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=Q4015L5-ND
IXTP26P20P-ND	MOSFET P-CH 200V 26A TO-220	y		http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=IXTP26P20P-ND
296-1395-5-ND	IC OPAMP GP 700KHZ DUAL 8DIP	y	y	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=296-1395-5-ND

6.4 PARTS LIST

6.4.1 CONTROL CIRCUIT

Part Value Device Package Library Sheet

12V-BAT PINHD-1X1 1X01 pinhead 1
12V-BAT-2 PINHD-1X1 1X01 pinhead 1
B PINHD-1X1 1X01 pinhead 1
BAT1 PINHD-1X1 1X01 pinhead 1
BAT2 PINHD-1X1 1X01 pinhead 1
BAT3 PINHD-1X1 1X01 pinhead 1
BAT4 PINHD-1X1 1X01 pinhead 1
BAT5 PINHD-1X1 1X01 pinhead 1
BAT6 PINHD-1X1 1X01 pinhead 1
BAT_VOLT EDGECONNECTOR-7 EDGECONNECTOR-7 EDGECONNECTOR lightning_riders 1
C1 22pf C-US025-024X044 C025-024X044 rcl 1
C1-A .1u C-US075-032X103 C075-032X103 rcl 1
C2 22pf C-US025-024X044 C025-024X044 rcl 1
C2-A .1u C-US075-032X103 C075-032X103 rcl 1
C3 10uF C-US025-024X044 C025-024X044 rcl 1
C3-A .1u C-US075-032X103 C075-032X103 rcl 1
C4 10uF C-US025-024X044 C025-024X044 rcl 1
C4-A .1u C-US075-032X103 C075-032X103 rcl 1
C5 10uF C-US025-024X044 C025-024X044 rcl 1
C5-A .1u C-US075-032X103 C075-032X103 rcl 1
C6 1mF C-US025-024X044 C025-024X044 rcl 1
C6-A .1u C-US075-032X103 C075-032X103 rcl 1
C7 1mF C-US025-024X044 C025-024X044 rcl 1
C7-A .1u C-US075-032X103 C075-032X103 rcl 1
C8 1mF C-US025-024X044 C025-024X044 rcl 1
C8-A .1u C-US075-032X103 C075-032X103 rcl 1
C9 .1u C-US050-035X075 C050-035X075 rcl 1
C9-A .1u C-US075-032X103 C075-032X103 rcl 1
C10 10uF C-US025-024X044 C025-024X044 rcl 1
C10-A .1u C-US075-032X103 C075-032X103 rcl 1
C11-A .1u C-US075-032X103 C075-032X103 rcl 1
C12-A .1u C-US075-032X103 C075-032X103 rcl 1
C15 .1u C-US050-035X075 C050-035X075 rcl 1

C16 .1u C-US050-035X075 C050-035X075 rcl 1
C17 .1u C-US050-035X075 C050-035X075 rcl 1
C18 .1u C-US050-035X075 C050-035X075 rcl 1
C19 .1u C-US050-035X075 C050-035X075 rcl 1
C20 .1u C-US050-035X075 C050-035X075 rcl 1
C21 .1u C-US050-035X075 C050-035X075 rcl 1
C22 .1u C-US050-035X075 C050-035X075 rcl 1
CHARGING MOLEX-34045-8 MOLEX-34045-8 MOLEX-34045-8 rmslib 1
CONTROLS MOLEX-6 MOLEX-6 MOLEX-6 lightning_riders 1
D PINHD-1X1 1X01 pinhead 1
D1 1N5400 1N5400 DO201-15 diode 1
D2 1N5400 1N5400 DO201-15 diode 1
E PINHD-1X1 1X01 pinhead 1
E1 PINHD-1X1 1X01 pinhead 1
E2 PINHD-1X1 1X01 pinhead 1
E3 PINHD-1X1 1X01 pinhead 1
E4 PINHD-1X1 1X01 pinhead 1
E5 PINHD-1X1 1X01 pinhead 1
EX-SPEED MOLEX-6 MOLEX-6 MOLEX-6 lightning_riders 1
FUSE F2332 F2332 F2332 mylib 1
GENS PINHD-1X1 1X01 pinhead 1
GND PINHD-1X1 1X01 pinhead 1
GND-2 PINHD-1X1 1X01 pinhead 1
I-B PINHD-1X1 1X01 pinhead 1
I-M PINHD-1X1 1X01 pinhead 1
IC1-A OPAMP OPAMP OPAMP lightning_riders 1
IC1-C A/D A/D A/D lightning_riders 1
IC2 18F46204P 18F46204P DIL40 mylib 1
IC2-A OPAMP OPAMP OPAMP lightning_riders 1
IC2-C A/D A/D A/D lightning_riders 1
IC2-C1 A/D A/D A/D lightning_riders 1
IC3-A OPAMP OPAMP OPAMP lightning_riders 1
IC4-A OPAMP OPAMP OPAMP lightning_riders 1
IC5-A OPAMP OPAMP OPAMP lightning_riders 1
IC6-A OPAMP OPAMP OPAMP lightning_riders 1
JP5 PINHD-2X7 2X07 pinhead 1
JP6 JP1Q JP1 jumper 1
JP7 JP5Q JP5Q jumper 1
JP8 JP1Q JP1 jumper 1
LASTMOS PINHD-1X1 1X01 pinhead 1
LED1 .7 LED5MM LED5MM led 1
LED2 .7 LED5MM LED5MM led 1
MD TC1411N TC1411N-GOOD DIL8 mylib 1
MD1 TC1411N TC1411N-GOOD DIL8 mylib 1
OKB PINHD-1X1 1X01 pinhead 1
OKT PINHD-1X1 1X01 pinhead 1
POWER JP2Q JP2Q jumper 1
POWER-EX MOLEX-6 MOLEX-6 MOLEX-6 lightning_riders 1
Q1 32.768kHZ CRYSTALHC49U-V HC49U-V crystal 1
Q2 10MHz CRYSTALHC49U-V HC49U-V crystal 1
Q3 FQP50N06 FQP50N06 TO220BV mylib 1
Q4 FQP50N06 FQP50N06 TO220BV mylib 1
R1 100 R-US_0207/7 0207/7 rcl 1
R1-A 51k R-US_0207/10 0207/10 rcl 1
R2 1K R-US_0207/7 0207/7 rcl 1
R2-A 1k R-US_0207/10 0207/10 rcl 1
R3 1K R-US_0207/7 0207/7 rcl 1

R3-A 2k R-US_0207/10 0207/10 rcl 1
R4 270 R-US_0207/7 0207/7 rcl 1
R4-A 4.02k R-US_0207/10 0207/10 rcl 1
R5 220 R-US_0207/7 0207/7 rcl 1
R5-A 4.02k R-US_0207/10 0207/10 rcl 1
R6 220 R-US_0207/7 0207/7 rcl 1
R6-A 2k R-US_0207/10 0207/10 rcl 1
R7 470 R-US_0207/7 0207/7 rcl 1
R7-A 51k R-US_0207/10 0207/10 rcl 1
R8 150 R-US_0207/7 0207/7 rcl 1
R8-A 1k R-US_0207/10 0207/10 rcl 1
R9 150 R-US_0207/7 0207/7 rcl 1
R9-A 2k R-US_0207/10 0207/10 rcl 1
R10 470 R-US_0207/7 0207/7 rcl 1
R10-A 4.02k R-US_0207/10 0207/10 rcl 1
R11 150 R-US_0207/7 0207/7 rcl 1
R11-A 4.02k R-US_0207/10 0207/10 rcl 1
R12 470 R-US_0207/7 0207/7 rcl 1
R12-A 2k R-US_0207/10 0207/10 rcl 1
R13 150 R-US_0207/7 0207/7 rcl 1
R13-A 51k R-US_0207/10 0207/10 rcl 1
R14 470 R-US_0207/7 0207/7 rcl 1
R14-A 1k R-US_0207/10 0207/10 rcl 1
R15 220 R-US_0207/7 0207/7 rcl 1
R15-A 2k R-US_0207/10 0207/10 rcl 1
R16 220 R-US_0207/7 0207/7 rcl 1
R16-A 4.02k R-US_0207/10 0207/10 rcl 1
R17 220 R-US_0207/7 0207/7 rcl 1
R17-A 4.02k R-US_0207/10 0207/10 rcl 1
R18 470 R-US_0207/7 0207/7 rcl 1
R18-A 2k R-US_0207/10 0207/10 rcl 1
R19 470 R-US_0207/7 0207/7 rcl 1
R19-A 51k R-US_0207/10 0207/10 rcl 1
R19-A1 1k R-US_0207/10 0207/10 rcl 1
R19-A2 250 R-US_0207/10 0207/10 rcl 1
R19-A3 1k R-US_0207/10 0207/10 rcl 1
R19-A4 250 R-US_0207/10 0207/10 rcl 1
R19-A5 1k R-US_0207/10 0207/10 rcl 1
R19-A6 250 R-US_0207/10 0207/10 rcl 1
R19-A7 1k R-US_0207/10 0207/10 rcl 1
R19-A8 250 R-US_0207/10 0207/10 rcl 1
R20 100 R-US_0207/7 0207/7 rcl 1
R20-A 1k R-US_0207/10 0207/10 rcl 1
R21 100 R-US_0207/7 0207/7 rcl 1
R21-A 2k R-US_0207/10 0207/10 rcl 1
R22 220 R-US_0207/7 0207/7 rcl 1
R22-A 4.02k R-US_0207/10 0207/10 rcl 1
R23 220 R-US_0207/7 0207/7 rcl 1
R23-A 4.02k\ R-US_0207/10 0207/10 rcl 1
R24 220 R-US_0207/7 0207/7 rcl 1
R24-A 2k R-US_0207/10 0207/10 rcl 1
R25-A 51k R-US_0207/10 0207/10 rcl 1
R26 100 R-US_0207/7 0207/7 rcl 1
R26-A 1k R-US_0207/10 0207/10 rcl 1
R27 470 R-US_0207/7 0207/7 rcl 1
R27-A 2k R-US_0207/10 0207/10 rcl 1
R28-A 4.02k R-US_0207/10 0207/10 rcl 1

R29-A	4.02	R-US_0207/10	0207/10	rcl	1
R30-A	2k	R-US_0207/10	0207/10	rcl	1
R31-A	51k	R-US_0207/10	0207/10	rcl	1
R32-A	1k	R-US_0207/10	0207/10	rcl	1
R33-A	2k	R-US_0207/10	0207/10	rcl	1
R34-A	4.02k	R-US_0207/10	0207/10	rcl	1
R35-A	4.02k	R-US_0207/10	0207/10	rcl	1
R36-A	2k	R-US_0207/10	0207/10	rcl	1
RELAY		PINHD-1X1	1X01	pinhead	1
S		PINHD-1X1	1X01	pinhead	1
S1	DT	DT	PBSWITCH	mylib	1
SV1		MA10-1	MA10-1	con-lstb	1
SV2		MA10-1	MA10-1	con-lstb	1
SV3		MA10-1	MA10-1	con-lstb	1
SV4		MA10-1	MA10-1	con-lstb	1
T1		PINHD-1X1	1X01	pinhead	1
T2		PINHD-1X1	1X01	pinhead	1
T3		PINHD-1X1	1X01	pinhead	1
T4		PINHD-1X1	1X01	pinhead	1
T5		PINHD-1X1	1X01	pinhead	1
T6		PINHD-1X1	1X01	pinhead	1
TEMP1-2	MOLEX-6	MOLEX-6	MOLEX-6	lightning_riders	1
TEMP3-4	MOLEX-6	MOLEX-6	MOLEX-6	lightning_riders	1
TEMP5-6	MOLEX-6	MOLEX-6	MOLEX-6	lightning_riders	1
U		PINHD-1X1	1X01	pinhead	1
U$1	USD-SOCKETUSD	USD-SOCKETUSD	USD-SOCKET-PP	SparkFun-3-2-07	1
U$2	MC78T05C	MC78T05C	TO220(V)	mylib	1
U$3	LD1585CV33	LD1585CV33	TO220(V)	mylib	1
U$4	DS1305	DS1305	DIL16	mylib	1
U$8	2N4401	2N4401	TO92-CBE	mylib	1
U$9	2N4401	2N4401	TO92-CBE	mylib	1
U$11	2N4401	2N4401	TO92-CBE	mylib	1
U$12	2N4401	2N4401	TO92-CBE	mylib	1
U$15	2N4401	2N4401	TO92-CBE	mylib	1
ZD1-A	15V	1N4728	DO41Z10	diode	1
ZD1-A2	15V	1N4728	DO41Z10	diode	1
ZD2-A	15V	1N4728	DO41Z10	diode	1
ZD3-A	15V	1N4728	DO41Z10	diode	1
ZD4-A	15V	1N4728	DO41Z10	diode	1
ZD5-A	15V	1N4728	DO41Z10	diode	1
ZD6-A	15V	1N4728	DO41Z10	diode	1

6.4.2 CHARGING CIRCUIT

Part	Value	Device	Package	Library	Sheet
72V		PINHD-1X1	1X01	pinhead	1
BCV		PINHD-1X1	1X01	pinhead	1
BE2		PINHD-1X1	1X01	pinhead	1
BE3		PINHD-1X1	1X01	pinhead	1
BE4		PINHD-1X1	1X01	pinhead	1
BE5		PINHD-1X1	1X01	pinhead	1

BE6 PINHD-1X1 1X01 pinhead 1
BEX MOLEX-6 MOLEX-6 MOLEX-6 lightning_riders 1
BS PINHD-1X1 1X01 pinhead 1
BUCK 4N35 4N35 DIL06 optocoupler 1
BUCK-MICRO 18F46204P 18F46204P DIL40 mylib 1
BU_USB JP5Q JP5Q jumper 1
C1 10uF C-US025-024X044 C025-024X044 rcl 1
C1-A .1u C-US075-032X103 C075-032X103 rcl 1
C1-A1 .1u C-US075-032X103 C075-032X103 rcl 1
C2 10uF C-US025-024X044 C025-024X044 rcl 1
C2-A .1u C-US075-032X103 C075-032X103 rcl 1
C2-A1 .1u C-US075-032X103 C075-032X103 rcl 1
C3 22pf C-US025-024X044 C025-024X044 rcl 1
C4 22pf C-US025-024X044 C025-024X044 rcl 1
C5 22pf C-US025-024X044 C025-024X044 rcl 1
C6 22pf C-US025-024X044 C025-024X044 rcl 1
C7 .1u C-US050-035X075 C050-035X075 rcl 1
C8 1mF C-US025-024X044 C025-024X044 rcl 1
C9 1mF C-US025-024X044 C025-024X044 rcl 1
C10 1mF C-US025-024X044 C025-024X044 rcl 1
C11 1mF C-US025-024X044 C025-024X044 rcl 1
C12 1mF C-US025-024X044 C025-024X044 rcl 1
C13 1mF C-US025-024X044 C025-024X044 rcl 1
C15 .1u C-US050-035X075 C050-035X075 rcl 1
CP1 PINHD-1X1 1X01 pinhead 1
CP2 PINHD-1X1 1X01 pinhead 1
D1 1N4004 1N4004 DO41-10 diode 1
D2 1N5400 1N5400 DO201-15 diode 1
EX PINHD-1X1 1X01 pinhead 1
EX1 PINHD-1X1 1X01 pinhead 1
EX2 PINHD-1X1 1X01 pinhead 1
F1 PINHD-1X1 1X01 pinhead 1
F5 PINHD-1X1 1X01 pinhead 1
F52 PINHD-1X1 1X01 pinhead 1
FG PINHD-1X1 1X01 pinhead 1
FG1 PINHD-1X1 1X01 pinhead 1
FG2 PINHD-1X1 1X01 pinhead 1
IB PINHD-1X1 1X01 pinhead 1
IC1-A OPAMP OPAMP OPAMP lightning_riders 1
IC1-A1 OPAMP OPAMP OPAMP lightning_riders 1
IC2-C1 A/D A/D A/D lightning_riders 1
IC2-C2 A/D A/D A/D lightning_riders 1
J1 MTA09-156 1X9MTA con-amp 1
JP1 JP1Q JP1 jumper 1
JP5 PINHD-2X2 2X02 pinhead 1
JP12 PINHD-2X4 2X04 pinhead 1
JP13 PINHD-2X2 2X02 pinhead 1
JP14 PINHD-2X3 2X03 pinhead 1
JP15 JP1Q JP1 jumper 1
L1 INDUCTOR INDUCTOR INDUCTOR lightning_riders 1
L2 INDUCTOR INDUCTOR INDUCTOR lightning_riders 1
LAST_R 4N35 4N35 DIL06 optocoupler 1
LR PINHD-1X1 1X01 pinhead 1
MOTORSENSOR MOLEX-6 MOLEX-6 MOLEX-6 lightning_riders 1
OK5 4N35 4N35 DIL06 optocoupler 1
Q1 10MHz CRYSTALHC49U-V HC49U-V crystal 1
Q2 10MHz CRYSTALHC49U-V HC49U-V crystal 1

R1 500 R-US_0207/10 0207/10 resistor 1
R1-A 68k R-US_0207/10 0207/10 rcl 1
R1-A1 68k R-US_0207/10 0207/10 rcl 1
R2 220 R-US_0207/10 0207/10 resistor 1
R2-A 1k R-US_0207/10 0207/10 rcl 1
R2-A1 1k R-US_0207/10 0207/10 rcl 1
R3 1000 R-US_0207/10 0207/10 resistor 1
R3-A 2k R-US_0207/10 0207/10 rcl 1
R3-A1 2k R-US_0207/10 0207/10 rcl 1
R4-A 4.02k R-US_0207/10 0207/10 rcl 1
R4-A1 4.02k R-US_0207/10 0207/10 rcl 1
R5 10000 R-US_0207/10 0207/10 resistor 1
R5-A 4.02k R-US_0207/10 0207/10 rcl 1
R5-A1 4.02k R-US_0207/10 0207/10 rcl 1
R6 100 R-US_0207/7 0207/7 rcl 1
R6-A 2k R-US_0207/10 0207/10 rcl 1
R6-A1 2k R-US_0207/10 0207/10 rcl 1
R7 1K R-US_0207/7 0207/7 rcl 1
R8 470 R-US_0207/7 0207/7 rcl 1
R9 100 R-US_0207/7 0207/7 rcl 1
R10 47000 R-US_0207/10 0207/10 resistor 1
R11 100 R-US_0207/7 0207/7 rcl 1
R12 470 R-US_0207/7 0207/7 rcl 1
R13 100 R-US_0207/7 0207/7 rcl 1
R14 1K R-US_0207/7 0207/7 rcl 1
R15 100 R-US_0207/7 0207/7 rcl 1
R16 470 R-US_0207/7 0207/7 rcl 1
R17 470 R-US_0207/7 0207/7 rcl 1
R18 470 R-US_0207/7 0207/7 rcl 1
R19 100 R-US_0207/7 0207/7 rcl 1
R20 470 R-US_0207/7 0207/7 rcl 1
R22 2.2k RESISTOR RESISTOR lightning_riders 1
R23 2.2k RESISTOR RESISTOR lightning_riders 1
R24 2.2k RESISTOR RESISTOR lightning_riders 1
R26 100 R-US_0207/7 0207/7 rcl 1
R27 470 R-US_0207/7 0207/7 rcl 1
RELAY PINHD-1X1 1X01 pinhead 1
S1 DT DT PBSWITCH mylib 1
S2 DT DT PBSWITCH mylib 1
SV1 MA10-1 MA10-1 con-lstb 1
SV2 MA10-1 MA10-1 con-lstb 1
SV3 MA10-1 MA10-1 con-lstb 1
SV4 MA10-1 MA10-1 con-lstb 1
SV5 MA10-1 MA10-1 con-lstb 1
SV6 MA10-1 MA10-1 con-lstb 1
SV7 MA10-1 MA10-1 con-lstb 1
SV8 MA10-1 MA10-1 con-lstb 1
TCV PINHD-1X1 1X01 pinhead 1
TE2 PINHD-1X1 1X01 pinhead 1
TE3 PINHD-1X1 1X01 pinhead 1
TE4 PINHD-1X1 1X01 pinhead 1
TE5 PINHD-1X1 1X01 pinhead 1
TE6 PINHD-1X1 1X01 pinhead 1
TEX MOLEX-6 MOLEX-6 MOLEX-6 lightning_riders 1
TR4N35 4N35 4N35 DIL06 optocoupler 1
TRIAC-MICRO 18F46204P 18F46204P DIL40 mylib 1
TRMOC MOC3010M MOC3010M DIL06 optocoupler 1

TR_USB JP5Q JP5Q jumper 1
TS PINHD-1X1 1X01 pinhead 1
U$1 IRF621PBF IRF621PBF TO220(B) mylib 1
U$2 IRF621PBF IRF621PBF TO220(B) mylib 1
U$3 F30UP20S F30UP20S TO220(X2) mylib 1
U$4 RELAY RELAY RELAY lightning_riders 1
U$5 F30UP20S F30UP20S TO220(X2) mylib 1
U$6 Q4015L5 Q4015L5 TO220(B) mylib 1
U$7 F30UP20S F30UP20S TO220(X2) mylib 1
U$8 GBJ2502-FDI-ND GBJ2502-FDI-ND GBJ mylib 1
U$9 CURRENT-SENSOR2 CURRENT-SENSOR2 CURRENT-SENSOR2 lightning_riders 1
U$10 F30UP20S F30UP20S TO220(X2) mylib 1
U$11 GBJ2502-FDI-ND GBJ2502-FDI-ND GBJ mylib 1
U$12 MC78T05C MC78T05C TO220(V) mylib 1
U$13 MOLEX-34045-8 MOLEX-34045-8 MOLEX-34045-8 rmslib 1
U$14 2N4401 2N4401 TO92-CBE mylib 1
U$15 2N4401 2N4401 TO92-CBE mylib 1
U$16 MOLEX-6 MOLEX-6 MOLEX-6 lightning_riders 1
WALL PINHD-1X1 1X01 pinhead 1
ZD1-A 15V 1N4728 DO41Z10 diode 1
ZD1-A2 15V 1N4728 DO41Z10 diode 1
ZD1-A3 15V 1N4728 DO41Z10 diode 1
ZD1-A4 15V 1N4728 DO41Z10 diode 1