Formula SAE Power Distribution Controller

by

Daniel Baron

Advisor: Professor Bridget Benson

Electrical Engineering Department

California Polytechnic State University

San Luis Obispo

2017

Table of Contents *Page*

List of Tables *Page*

List of Figures *Page*

Abstract:

The Power Distribution Controller senior project aims to replace the current Power Distribution Module (PDM) on Cal Poly's Formula SAE internal combustion vehicle with a more advanced, configurable, and reliable system. The Power Distribution Controller is a MOSFET based system that can be controlled through either controller area network or a parallel interface to distribute and control power from the vehicle's battery to the several electrical devices on the car. The Power Distribution Controller also collects output voltage and current data, and communicates this data to the vehicle's data logger. This will allow engineers to troubleshoot electrical problems on the car by looking through log files, and also detect possible sources of electrical failure before the devices fail. Finally, the Power Distribution Controller is highly configurable and adaptable to any surrounding electrical system by being updatable through firmware rather than hardware.

1. Background and Introduction

Background:

Formula SAE is an engineering competition in which "teams are to assume that they work for a design firm that is designing, fabricating, testing and demonstrating a prototype vehicle for the nonprofessional, weekend, competition market [1]". Cal Poly's Formula SAE car from 2016 is shown in Figure 1.1. As an engineering project, Formula SAE teams have extensive electronics systems to control the engine, as well as collect kinematic data to tune and improve designs. At the center of the electronics systems is power distribution circuitry to deliver power to several electronic devices.

Figure 1.1: Cal Poly Racing 2016 Formula SAE car

Historically, Cal Poly Racing has used relays and fuses to distribute power and protect circuits as seen in figure 1.2. While this is the typical method of power distribution for production vehicles due to simplicity and cost, this is an atypical solution for racecars. Racecars generally use power distribution controllers which have digital inputs as well as solid state switching outputs in order to control, distribute, and protect circuits as seen in figure 1.3.

Figure 1.2: Typical Production Vehicle power distribution circuitry

Figure 1.3: Typical racecar power distribution controller

Racecars use digitally controlled, solid state power controllers as depicted in figure 1.3 for several reasons. First, the digital input and output controller allows for a much more controllable and versatile system than its analog counterpart as seen in figure 1.2. Secondly, when circuits fail and draw too much current with a traditional relay and fuse style power distribution system, the fuse blows, and the circuit is unusable until the fuse is replaced. This could be detrimental to a racecar's operation and performance in the middle of a race. A digital power distribution controller is able to attempt to restart the circuit after the overcurrent condition, and can determine if a circuit should be left on or shut down after analyzing the failing circuit's impact to the car's operation, safety, and potential electrical system damages. Finally, racecars utilize digital power distribution controllers because racecar electrical systems undergo several modifications throughout a racing season, and major modifications between seasons. With analog power distribution circuity, when modifications are made to the surrounding electrical system architecture, hardware such as relays and fuses are required to be added to the power distribution circuity to support the new system architecture. With a digital power distribution controller, when a change is made to the surrounding electrical system architecture, a wire may need to be added going to the power controller, but toggling and controlling the output(s) can be modified in firmware, leading to a much faster and more reliable implementation.

Product Description:

The Formula SAE Power Distribution Controller (PDC) is a solid-state power controller for prototype racecars. Racecars aim for lightweight, small, reliable, and efficient products. This power controller takes advantage of lightweight solid state electronics to replace power inefficient, large, and heavy relays and fuses. The CAN interface of this product also contributes to a very clean device implementation. The primary customer of the product, Cal Poly Racing Formula SAE, sees the greatest improvement in lap times with more testing time for the car. When electrical devices fail on the vehicle, engineers have the painstaking process of continuity testing, checking fuses, and testing relays to determine the source of the problem, all of which takes away time from testing. The PDC solves this issue by logging output currents and voltages to detect problems before devices fail, and locate the source of the electrical problem quickly when devices do fail. This will decrease the amount of time troubleshooting, allowing for more time on the track.

Racecar electrical systems can best be described as prototypes that undergo several modifications and revisions within a season. Because power distribution circuity sits at the center point of this architecture to deliver power to all electrical devices, it needs to be changed frequently to work with the modifications and revisions of the surrounding system. Traditional fuses and relays require hardware changes in order to support changes in the vehicle's electrical system. This also requires revalidation of hardware reliability, and often compromises in reliability. The PDC solves this issue by being able to change input and output relationships through firmware updates, making this product highly configurable. Finally, Formula SAE teams cannot afford programmable solid state power controllers from well-known suppliers such as Motec. These products have a significant amount of development cost and are highly priced in the market due to the convenient user interface they have for configuring inputs and outputs. The Formula SAE power controller lowers cost by acknowledging that the customers have technical knowledge to program firmware into the power controller. This removes development cost of the user interface, and decreases the price of the product in the market, ultimately saving customers thousands of dollars.

Table 1.1: Market research and competing products [2] [3] [4]

Picture	Product Name	Price	Pros	Cons
	Motec PDM15	$1636.53	• Small • Light weight • Developed User interface for modifying settings	• Expensive • Limited configurations based on user interface
	Motec PDM16	$2463.55	• Small and light weight • Durable • High quality components • Developed user interface	• Expensive • Expensive mating connectors (~$150 per connector) • Limited Configurations based on user interface
	Cartek Power Distribution Module	$272.74	• Inexpensive and affordable • Very small and light weight	• Too few outputs (may require purchase of 2 modules) • Outputs too low of current rating • Not configurable

2. Customer Archetype, Needs, Requirements, and Specifications

The customers for the Formula SAE Power Distribution Controller are Formula SAE teams. These teams consist of full time students, mostly engineers, who also build Formula SAE cars in their spare time. The users and influencers on the team are typically electrical or computer engineering students who are in charge of designing the electrical systems for the teams' vehicle. These students typically have technical aptitude and programming knowledge. However, the short timeline to design, manufacture, and test the Formula SAE car, in combination with the limited amount of spare time an engineering student has, often leads to the users and influencers looking for suitable off-the-shelf solutions that can be implemented quickly. The buyer on the team is typically a student and the team manager. This individual controls a very tight budget, as Formula SAE teams are often reliant on donations and sponsorships. The predesigned hardware and open source firmware of the Power Distribution Controller is an excellent solution for the users and influencers because the product is incredibly adaptable and offers the user full control over the inputs and outputs, while also saving time on the design and implementation. This product appeals to the buyer on the teams because it is reasonably priced. When the influencer asks the buyer to purchase a product, the more expensive the product is, the more the influencer must convince the buyer that the money spent will improve the overall performance of the vehicle at competition. Teams that will be seeking to purchase this power distribution controller will generally be newer teams, small teams, and teams just starting out in developing electronics. Teams that are well established in the Formula SAE competition series often have a custom designed power controller already

Pain relievers and gains:

The Formula SAE power distribution module solves both technical and management pains for Formula SAE teams. From a technical standpoint, because the Power Controller utilizes solid state electronics, it is more robust and more reliable than traditional fuse and relay systems. Additionally, because of microcontroller on the power controller, electrical circuits will only be turned off in competition if the circuit is a risk to safety, or any other criteria set by the

team, not just a simple overcurrent criteria set by traditional fuses. The microcontroller will also allow for analysis of the electrical system and enable faster troubleshooting when problems do arise. The more time spent testing the vehicle and less time troubleshooting problems, directly translates to a better performing car at competition. From a management standpoint, this product solves the problem of teams having to choose to spend a significant amount of time designing a power distribution system, or spending a significant amount of money purchasing one. This product offers a nice compromise to the two solutions by providing low cost hardware, but also requiring a small amount of time configuring the open source firmware so that the hardware integrates nicely with each team's car.

For teams that have previously used relays and fuses for power distribution, teams will see significant gains in electrical system control, and see gains in optimizing electrical systems by collecting data from the power distribution controller. These teams will also have gains by obtaining more reliable system while having a faster design and implementation than designing their own fuse and relay system or custom power controller.

Market Leaders:

Motec is the market leader for intelligent power distribution modules for motorsports. They mostly serve professional and Semiprofessional markets

Littelfuse is the market leader for relay and fuse based power distribution modules. Littelfuse sells modular systems in which different configurations of relays and fuses can be used to distribute power. They mostly serve the aftermarket truck, commercial, and industrial markets.

Limitations of Competitors

The two key competitors for this product are Littelfuse and Motec. Motec does not have limitations on their product that would be problematic to a Formula SAE team. Their systems are designed to power electrical systems for professional and semiprofessional racecars that

are much more extensive than a Formula SAE car's electrical system. These features come at a price that is far too expensive for the vast majority of Formula SAE teams. Littelfuse is the leader in providing analog power distribution modules. While these systems are affordable, they often lack the feature set desired by Formula SAE teams. The limitations of Littelfuse systems is that once a fuse blows, the electrical circuit cannot be reset immediately, which could lead to a poorly performing racecar. Additionally, multiple criteria cannot toggle an output on the Littlefuse system without the addition of external controllers.

Key Areas of Strength

The Formula SAE Power Controller has two key areas of leverage. First, the system is affordable for most Formula SAE teams and other amateur race teams. This is in contrast to the high priced Motec system. The second area of leverage is that the Formula SAE Power Controller is versatile, configurable, and adaptable through open source firmware. This is in contrast to the Littelfuse system. Overall, the combination of these two strengths offers leverage over Motec and Littelfuse in the target market.

Customer Needs

Table 2.1: Customer Needs

Feature	Importance	Customer Need
High Current Outputs	High	Customers need to be able to drive and control fuel pumps, fans, and ignition systems that draw a lot of current
Low Current Outputs	High	Customers need to control power to low current control systems such as shifting and engine control
Digital Inputs	Medium	Customers without CAN communication on their vehicles need to interface this product through digital inputs
CAN Receiver	High	Customers with CAN on their vehicle must be able to control the Power Controller through CAN communication
CAN Transmitter	Low	Communicate output current and voltage data to external data logger
Digital “Fuses”	High	Customers must be able to set current limits on outputs through firmware to protect electronics
Water resistant/Durable Build	High	Power Distribution Controller must be able to operate in rain, wet road conditions, and dusty/dirty conditions

Marketing Requirements

Table 2.2: Marketing requirements

Marketing Requirement	**Rationale**
Product sold for $400	This price point is low enough where Formula SAE teams would buy the product and low enough to significantly disrupt the market when the product arrives at the market. This price is also high enough to make a profit
4 High Current Outputs	The Formula SAE car currently uses three high current outputs. The 4th high current output would be to add another high current device if necessary.
8 Low Current Outputs	Low current outputs are the most common output on the car, and the Formula SAE car currently uses two of these outputs. This is being expanded to eight to more properly protect various electronic devices from overcurrent conditions.
6 Digital Inputs	6 digital inputs are needed because if CAN fails, three inputs would be enough to control the fuel pump, cooling fan, and ignition, and then the other three inputs could be used for various inputs such as a pressure sensor for a brake light.
Capable of delivering 600 Watts of power steady state for extended periods of time	The Formula SAE car currently draws 396 Watts. Requiring the device to supply 600 Watts allows for room to improve the electrical system by adding more electronics.
All outputs can individually have current limits set in firmware	Different outputs require different current limits. Users need to be able to choose any current limit they desire for a particular output.
Product must interface easily with surrounding electrical system	The customer must not be frustrated with the implementation or use of this product, in order for the business model to be sustainable.
Build must be durable and resistant to impact, pressure, and heat	The device is being used in a high vibration and hot environment. Furthermore, people may hit the device by accident while working on the car.
Product must be user friendly, adaptable, and easily configurable through firmware	The device must be easy to use for the customer to be satisfied. The system must also be adaptable to give the race car's electrical systems designer flexibility in his or her design.

Engineering Requirements

Table 2.3: Marketing requirements translated to engineering requirements

Marketing Requirement	Engineering Requirement
Affordable	• Prototype Costs $200 (max)
Meets FSAE Rules	• Follows the FSAE rule book regarding emergency cutoff switch circuitry
4 High Current Outputs	• Outputs must source 20A max at 14.4 V
8 Low Current Outputs	• Outputs must source 5A max at 14.4 V
6 Inputs (Analog and Digital)	• Inputs must accept up to 14.4 V • Inputs must be over voltage protected
Capable of delivering 600 W of power steady state for extended periods of time (~40 A continuous current)	• Satisfy marketing wattage and current requirements • Deliver maximum power for at least 1 hour intervals • Size PCB traces to have a 10 °F maximum temperature rise
All Outputs can individually have current limits set in firmware	• Current limits must have 0.1 A precision
Turning of a circuit in reaction to the current limit must be faster than a fuse	• In the case of an overcurrent condition, the output must shut down in less than 10 ms • Analog redundancy must limit current in case of digital failure
Product must interface easily with surrounding electrical system	• Use inexpensive connectors to interface with the rest of the electrical system: < $10 per connector • Terminals must accept 22 AWG wire
Build must be durable and resistant to impact, pressure, and heat	• Connectors must be automotive rated and water resistant • Enclosure must be water resistant • Product must operate between 40 °F and 120 °F
Product must be user friendly, adaptable, and easily configurable through software	• Sends data regarding output voltages and currents over CAN • Current limits for outputs set in software • All outputs can be toggled based on CAN or input data • Electrical system warnings must be sent over CAN

Project Deliverables

Table 2.4: Power Distribution Controller Project Deliverables

Delivery Date	**Description**
March 22, 2017	EE 461 Alpha Demo
May 8, 2017	EE 462 Beta Demo
May 8, 2017	EE 462 Report Rough Draft
June 2, 2017	Senior Project Expo
June 14, 2017	EE 462 Final Demo
June 14, 2017	Final Report

3. Functional Decomposition

Level 0 Block Diagram Input/Output Descriptions

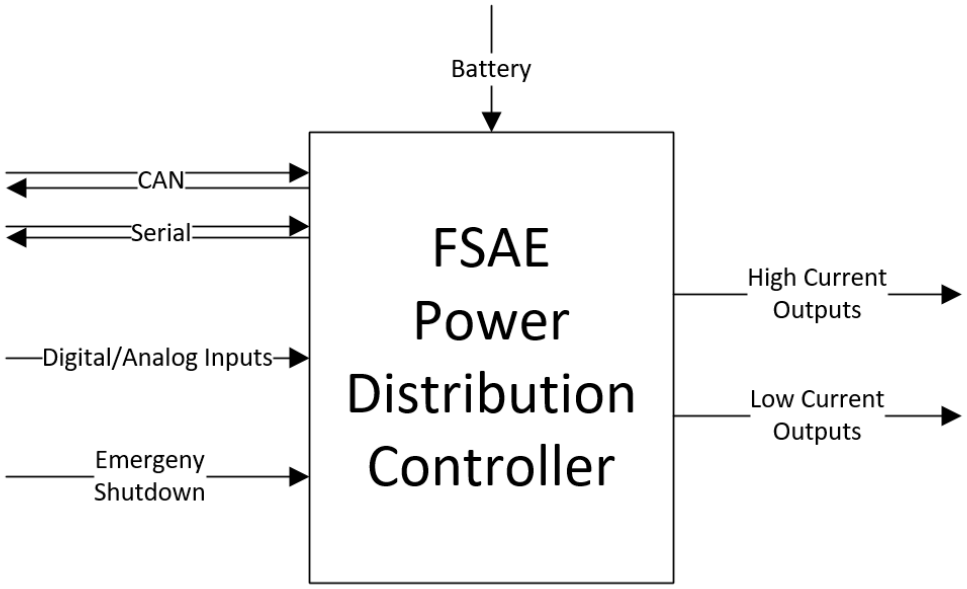

Figure 3.1: Level 0 Block Diagram of the FSAE Power Distribution Controller

High Current Outputs: Four high current outputs that are current limited at 15 A through software, and 20 A through the analog redundancy provide the FSAE vehicle power for high current loads such as ignition coils, cooling fans, and fuel pumps.

Low Current Outputs: Eight low current outputs that are current limited at 5 A each through software and 6 A through the analog redundancy, provide power for low current draw loads such as data loggers and sensors.

Digital and Analog Inputs: Six digital inputs can be configured to control and toggle the high current outputs and/or the low current outputs. Two of the inputs can optionally be selected to read analog inputs.

Controller Area Network (CAN): CAN will be used for the communication between the Power Distribution Controller, and other controllers in the FSAE car. Channels received over CAN will be able to toggle the high current and low current outputs. CAN data will

also be transmitted from the power controller so that information on current draws and output voltages can be logged in the car's data logger.

Emergency Shutdown: In accordance with FSAE competition rules, power must be turned off to critical engine components through two switches in the case of an emergency. Additionally, these switches are required to act through analog means (ie. not software dependent). Due to these switches requiring analog circuitry, they are distinctly different from the digital inputs.

Level 1 Functional Block Description

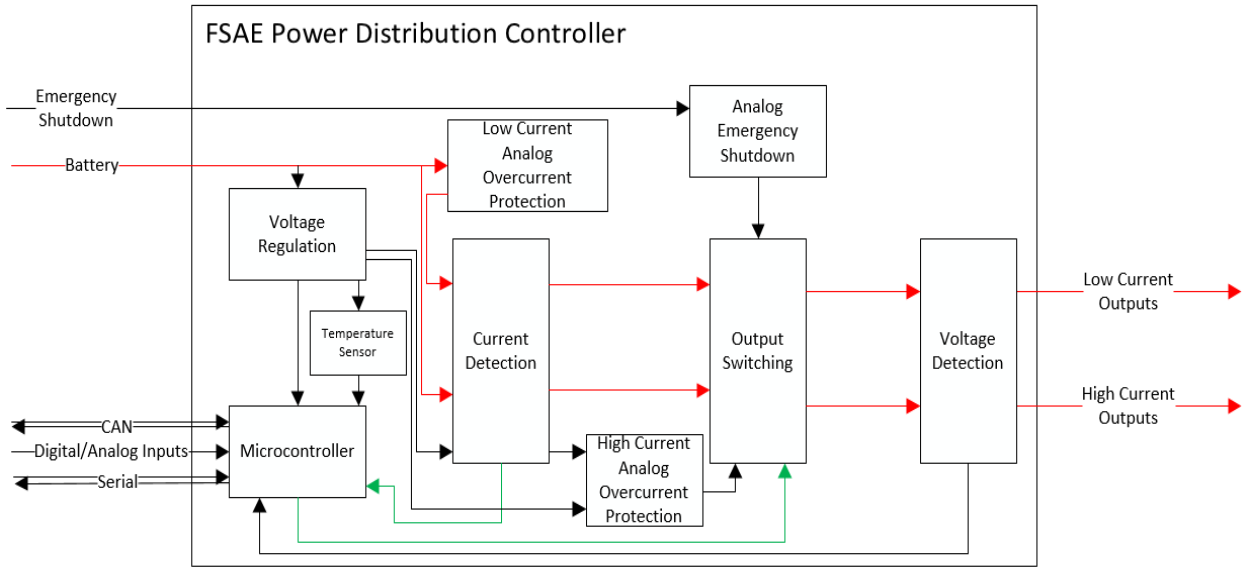

Figure 3.2: Level 1 Block Diagram of the FSAE Power Distribution Controller. The red arrows indicate the main power flow, while the green arrows indicate the control loop for "fusing" through software

Microcontroller: The microcontroller is at the center of the power controller and analyzes information from the digital and analog inputs, CAN communication, serial interface, and voltage and current information from the outputs, and controls the high current and low current outputs accordingly.

Output Switching: The microcontroller is unable to provide the outputs with sufficient driving voltage and current. The output switching block receives signals from the microcontroller, and then amplifies these signals to the appropriate levels for the output. The switching block must make the output signal approximately the same as the battery voltage, and source current up to 20 amps for the high current outputs, and 6 amps for the low current outputs

Analog Emergency Shutdown: The analog emergency shutdown circuit is required by FSAE rules. This circuit accepts input from an external switch, and if activated, the circuit disables power to the high current outputs. The circuit is required to bypass the microcontroller so that it still operates in the case that the code in the microcontroller fails.

Low Current Redundant Overcurrent Protection: The analog overcurrent protection consists of PTC resettable fuses that will “blow” if the current detection and/or microcontroller fail and the output current exceeds the rated output current. Under normal operation, the user will set a current limit, and this value will be lower than the fusing value of the PTC fuse in this block.

High Current Redundant Overcurrent Protection: The analog overcurrent protection consists of a feedback loop from the current detection block to disable high current outputs in case of the software failing to shut down the output before exceeding 20 A.

Voltage and Current Detection: This block detects the voltage and current of the output and feeds this information back to the microcontroller. If the current exceeds the user set current limit, the microcontroller will turn the output off. If the output voltage is lower than expected, the microcontroller will send error messages over CAN in order to notify the driver.

Temperature Sensing: The power controller will have an internal temperature sensor because the significant amount of power passing through the circuit board will result in a significant amount of heat. If the heat exceeds a particular temperature, the microcontroller will turn off all outputs that do not stop the car from driving.

4. Project Planning

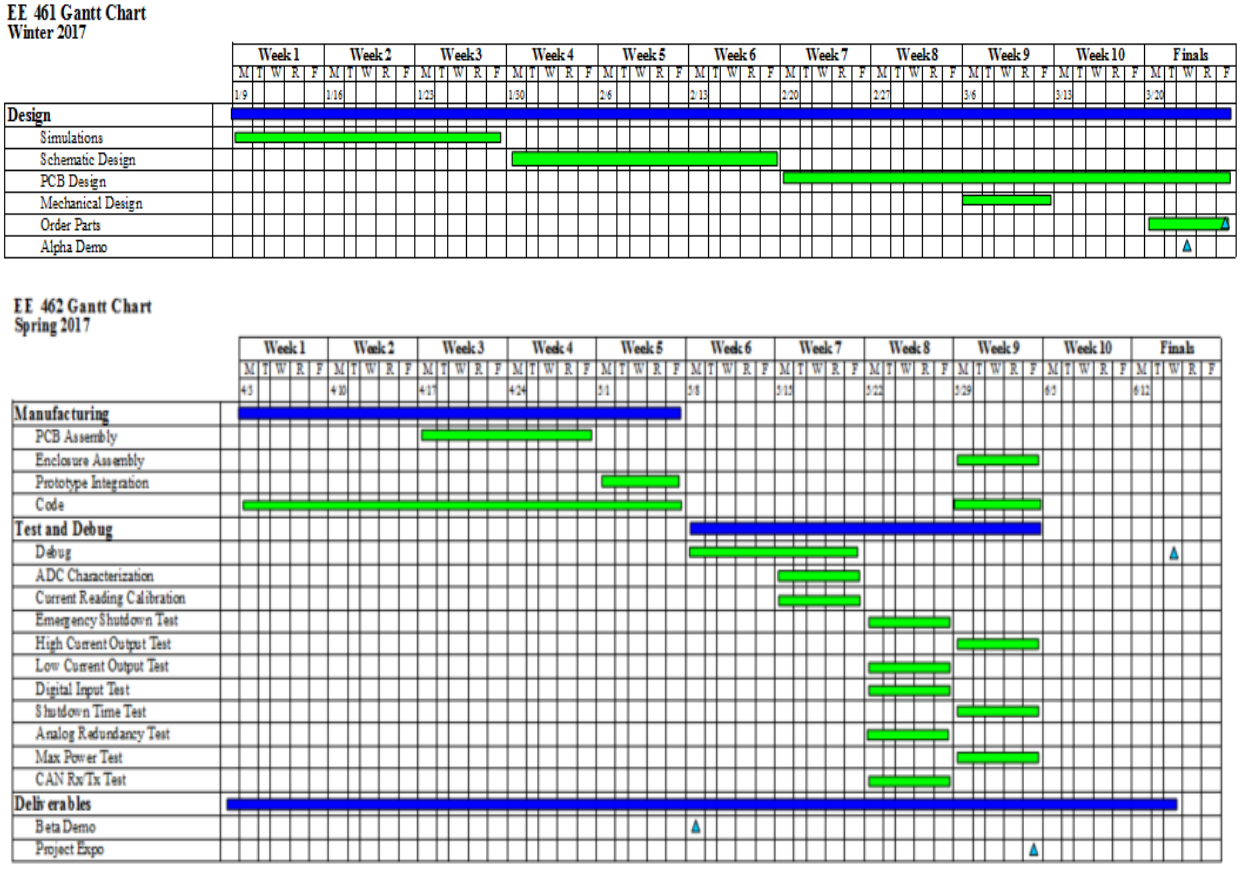

Figure 4.1: Gantt Chart for EE 461 and EE 462

Cost

Table 4.1: General Cost breakdown

Item	Quantity	Cost
Circuit Board Components	1	$120
Printed Circuit Board	1	$0
Connectors, Enclosure, and Hardware	-	$2
Extra Expenses (Shipping, re-orders, extra components)	-	$80
Total		$202

Table 4.1 shows a general categorized cost breakdown for the power distribution controller. A more detailed bill of materials and associated costs may be found in Appendix C. If all

materials were to be bought outright for this project, total cost would near $300 before additional expenses such as shipping, re-orders, and extra components, however, this project has received support from many industry partners who have aided this project with parts and materials. These donations have lowered the cost of this project significantly.

Finance

This project is primarily financed through the electrical engineering department's $200 grant per student. However, this project will also be financed through part donations from suppliers and Cal Poly Racing. While Cal Poly Racing will not contribute to the upfront costs of the prototype, the team will be responsible for the cost of maintaining the product as well as building more of these products for backups. This project will work with Cal Poly Racing industry partners in order to obtain free and discounted parts.

Resources

People: Advisors will be very important to the success of this project. I will consult with Professor Benson, and Cal Poly graduate and former Formula Electric SAE team lead Thomas Wilson in order to review design decisions, schematics, circuit boards, and overall design. I will also be consulting with the current Cal Poly racing leadership and electronics subsystem lead in order to ensure that the end product will be compatible with next year's vehicle and vehicles for future years.

Campus Resources: This project will require access to the Senior Project Lab and the Electrochemistry lab. These two labs contain the equipment needed to construct and test this project. Furthermore, this project will require access to the Hangar and Mustang 60 shops in order to construct the enclosure for the project.

Skills: This project requires programming, analog circuit design, digital circuit design, mixed signal printed circuit board design, soldering, and debugging skills. Furthermore, in order to complete this project on time, this project will require superb organizational and time management skills.

5. Project Design

Hardware Design

(Refer to Appendix B for hardware schematics)

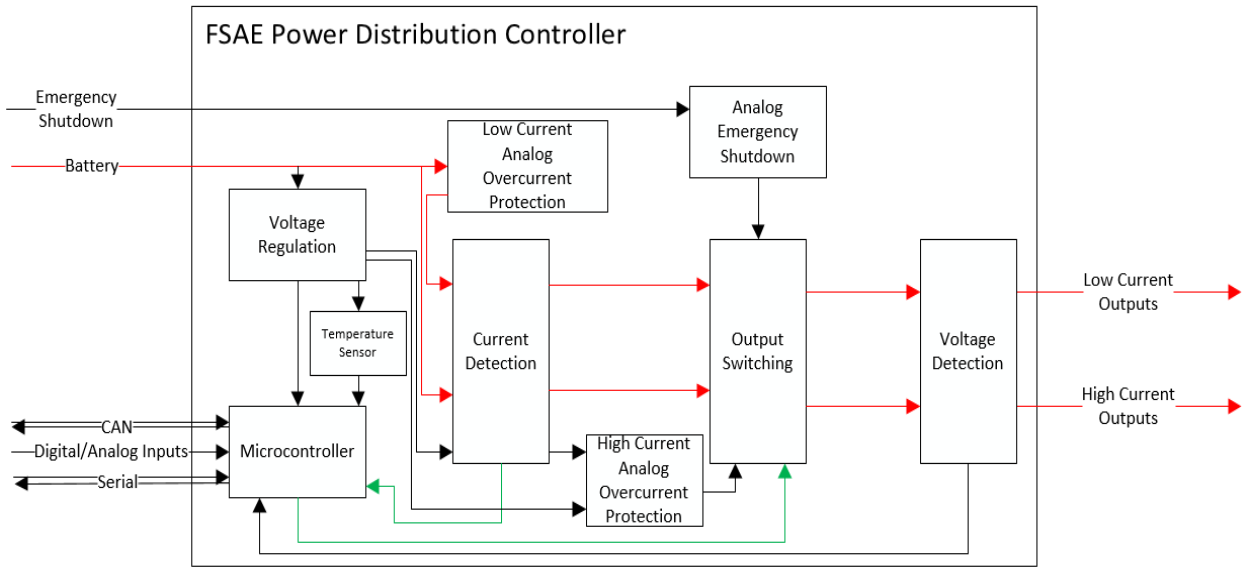

Figure 5.1: Level 1 Block Diagram of the FSAE Power Distribution Controller

Microcontroller

The AT90CAN128 microcontroller is at the center of the Power Distribution Controller. The AT90CAN128 is an 8-bit AVR microcontroller with 128 kilobytes of ISP flash and a CAN controller. This microcontroller also contains a 10 bit SAR ADC with eight ADC input channels. Furthermore, this microcontroller contains four hardware timers that can time events and provide outputs with PWM if needed. Most importantly, this microcontroller is used on all Cal Poly Racing circuit boards meaning that they are readily available to the Formula SAE team, and the people who will be using the Power Distribution Controller will be familiar with programming this microcontroller.

Additional circuitry that supports the microcontroller is the 16MHz external clock, the FTDI232RL USB to UART interface, and the MCP2551 CAN Transceiver. The MCP2551 CAN transceiver supports up to 1Mb/s.

High Current Outputs

The high current outputs provide power for loads drawing up to 15 A. The main power path through the high current outputs begin at main power input from the battery, then travels through a 0.002 Ω resistor for current measurement, and then passes through two power N-MOSFETs in parallel which controls the on/off state of the output.

To measure current, the LT6100 current sense amplifier reads the voltage across the 0.002 Ω resistor, and then amplifies that voltage by 40 V/V. The output of the amplifier then is sent through a 4:1 mux that selects which of the four current readings, from the four high current outputs, should be sent to the ADC. The output of the current sense amplifier is also sent to the high current output redundancy circuit as described below.

The two MOSFETs in parallel are the Nexperia PSMN2R2-25YLC. Two MOSFETs are used in parallel because it significantly reduces the amount of power dissipated by each MOSFET and reducing the amount of resulting heat while carrying high currents as seen in figure 5.2. Two MOSFETs in parallel also do not take up too much of the available circuit board space.

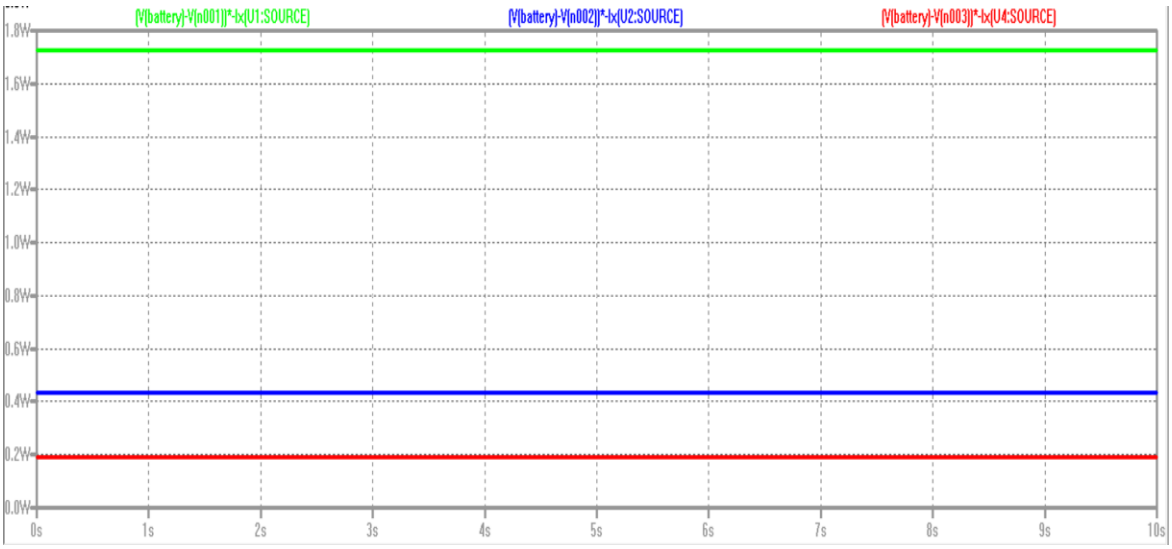

Figure 5.2: This simulation shows the amount of power dissipated for each PSMN2R2-YLC in three different configurations. The green line is a single MOSFET, the blue line is two MOSFETs in parallel, and the red line is three MOSFETs in parallel.

MOSFET Gate Driving

In order to reduce drain to source resistance and voltage drop in the MOSFETs, NMOS are being used to high side drive the loads on the outputs. In order to high side drive with a NMOS and reduce the drain to source resistance, the NMOS must operate in the linear mode of operation with ideally the highest gate to source voltage allowed as per the electrical specifications of the MOSFET. In the case of the PSMN2R2-YLC, this value is approximately 20 V.

The microcontroller outputs a 5 V signal to the LT1161 High-Side MOSFET Driver which converts the 5 V signal to a 20 V signal that is able to drive the MOSFET gate. The signal from the microcontroller to the gate driver passes through the analog emergency shut down circuitry for the high current outputs. This connection is direct for the low current outputs.

High Current Output Redundancy

The high current output redundancy prevents damage to the Power Distribution Controller in the case that the software fails to shut down a circuit in an overcurrent condition. This circuit operates by feeding the output of the current sense amplifier to a hysteretic comparator that compares the amplifier output to 2 V, the voltage at which output current equals 20 A. When the voltage from the current sense amplifier exceeds 2 V, the comparator triggers low to high, and enables a BJT which then drains the gate of the power MOSFET. Because the gate voltage is drained, the output is disabled.

High current outputs commonly drive inductive loads that have sharp current spikes when turning on. For this reason the analog redundancy is disabled for 10 milliseconds after an output is turned on. The 2 V reference described above is provided by a digital to analog converter. To disable the redundancy circuit, the reference is changed from 2 V to 5 V momentarily, therefore prevent the comparator from triggering low to high.

Analog Emergency Shut Down

Analog emergency shutdown consists of a BJT that has its gate pulled high to 5 V via a pullup resistor. When the base of the BJT is shorted to ground via an emergency stop switch, the microcontroller's signal can pass to the gate driver, however, when the emergency stop is pushed and the base is passively pulled high, the microcontroller signal is blocked from passing through to the gate driver, therefore disabling the high current outputs.

Low Current Outputs

The eight low current outputs drive loads that are less than or equal to 5 amps. The main power path through these outputs start at the main power input, then flow through a PTC resettable fuse for the analog overcurrent redundancy. From the PTC fuse, the current flows through a 0.02 Ω resistor for current measurement, and then flows to a single PSMN2R2-25YLC N-MOSFET. Current is sensed in the same manner as the high current outputs, however, this time the amplifier reads the voltage across the 0.02 Ω resistor, and amplifies the value by 50 V/V.

Voltage Measurement

The voltage is measured at each low current and high current output. In order to ensure voltages being sent to the ADC are on the 0 V to 5 V scale, a resistive divider made of a 26.1 kΩ and a 10 kΩ resistor is used. The center node of the two resistors is then fed through two 8:1 muxes to determine which signal is to be sent to the ADCs.

Connector

The Super Seal 1.0, 60 position connector allows for the Power Distribution Controller to connect with the rest of the Formula car's electronics. The Super Seal connector is inexpensive, has a high pin density, is automotive rated, and is water resistant. Furthermore, the formula team uses the Super Seal connector on their engine control unit with a different key pattern, meaning that the contacts for the Super Seal connector are readily available to the team.

Printed Circuit Board

The printed circuit board is a 2 ounce copper board that was designed to have a maximum 10 °F temperature rise. The corners of the board were rounded to a quarter inch diameter to offer the option of CNC milling an enclosure that can have a snug fit with the circuit board. The printed circuit board contains 3 main sections: the power path (figure 5.3 white box), the controls and driving block (figure 5.3 yellow box), and the digital block (figure 5.3 blue box). The power path contains the components described in the high current and low current output sections. Furthermore, the current sense amplifiers are placed directly underneath their respective current sense resistors in order to reduce the influence of noise and voltage drop. The controls section contains gate drivers, multiplexers, analog redundancy circuitry, and emergency shutdown circuitry. The digital block contains all elements described in the microcontroller section.

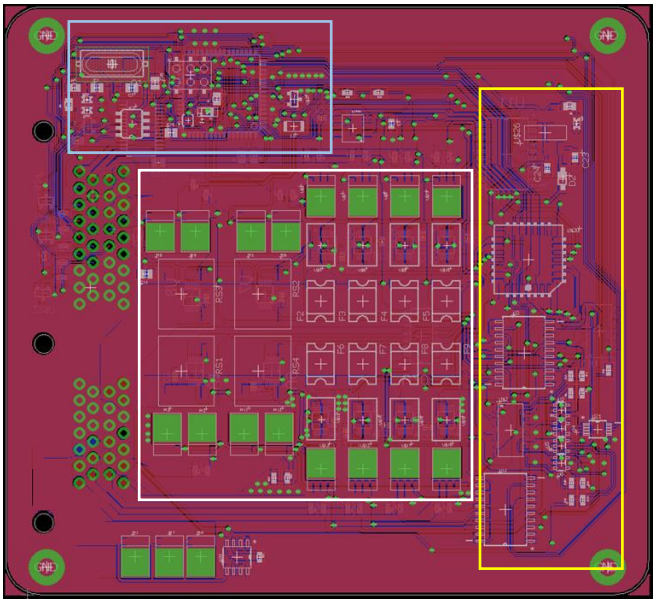

Figure 5.3: Printed Circuit Board

Enclosure

Figure 5.4 shows the enclosure CAD

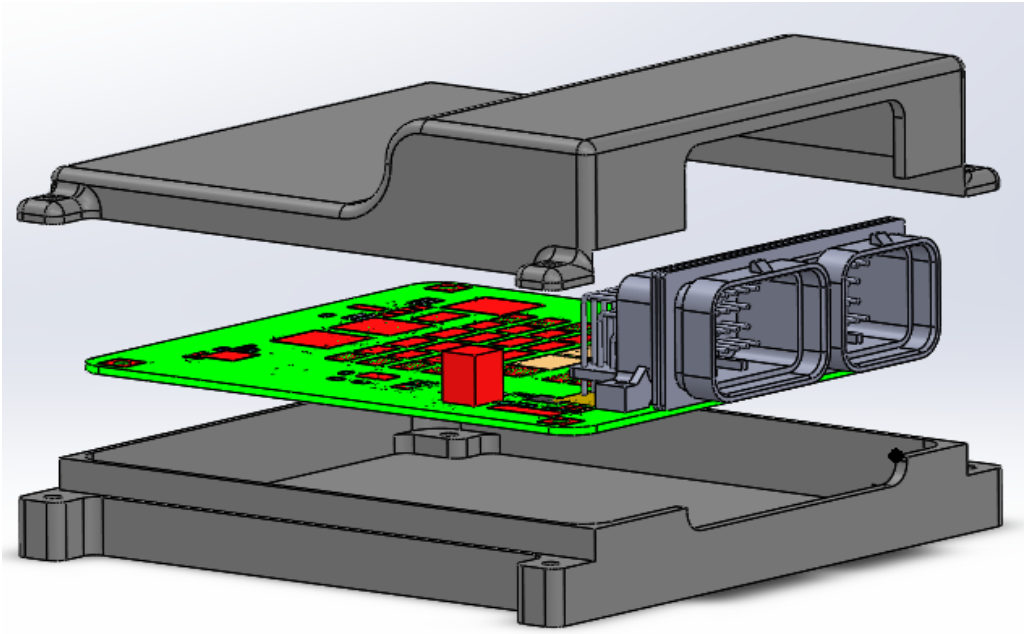

Figure 5.4: PDC Enclosure CAD

There are two key features to the design of this enclosure. First, when the top and bottom halves are joined together, the two halves overlap on three sides of the enclosure. This overlap prevents water from seeping into the box when it is splashed or sprayed on the box. On the connector side of the enclosure, the cutout is made to lock the PCB in place via the connector, and to make a tight seal with the connector's flange. The second key feature is that interior corners of the enclosure are rounded to a quarter inch diameter. The design intent with the filleted corners is that this enclosure could be made on the CNC mills in Cal Poly's shop. The first revision of the enclosure was 3D printed using PLA plastic, however, in order to assist with heat dissipation, this enclosure could be milled out of aluminum and act as a heatsink for the printed circuit board.

Software Design

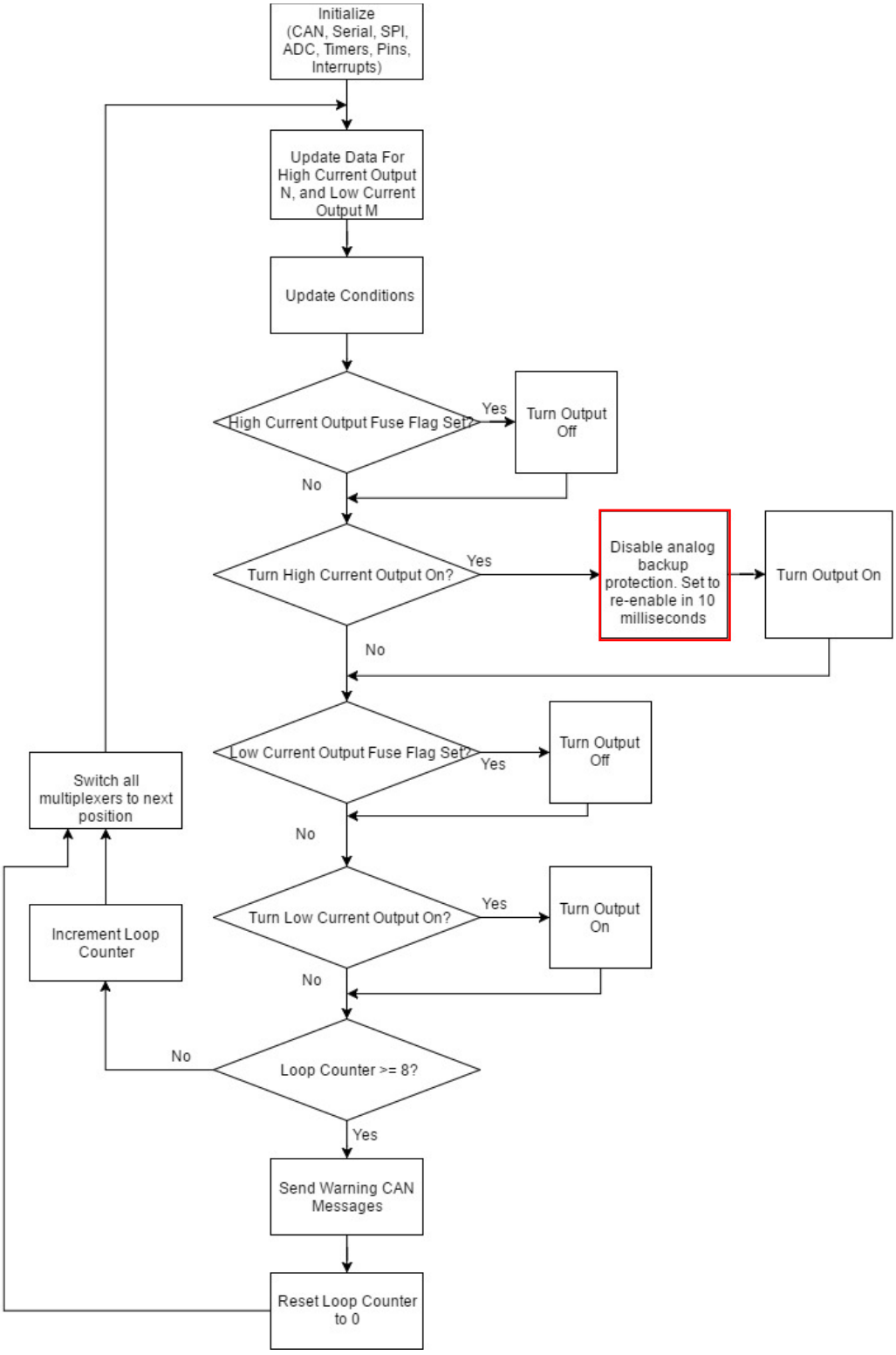

Figure 5.5: Software flow diagram

Code Summary

There are three major sections to the code. The first section is the initialization, the second section is the data collection and output switching, and the third section is the CAN data transmission. Figure 5.6 shows the main function. This section of the report will summarize the three major sections of code. For the full code, please refer to Appendix F.

```
int main(){
        pdc.InitializePDC(&canRxIntFunc);
        OutputCheckCounter = 0;
        pdc.SetOutputSettings();

        while(1){
                while(OutputCheckCounter < 8){
                        pdc.UpdateData(OutputCheckCounter);
                        pdc.OutputIO();
                        OutputCheckCounter++;
                }

                OutputCheckCounter=0;
                pdc.TxCANdata();
        }
        return 0;
}
```

Figure 5.6: Main Function

Initialization

The initialization first consists of CAN, GPIO, serial, SPI, ADC, timer, and interrupt initialization in pdc.InitializePDC(&canRxIntFunc). The argument to this function is the interrupt vector for the CAN receive interrupt. The timer initialization consists of Timer 1 which interrupts every 10 milliseconds to check if the analog redundancy for the high current outputs should be re-enabled after the output turns on (this section of logic can be found in the red box in Figure 5.5) and Timer 2 which delays the activation of the firmware fuse if a delay is set by the user.

The second part of the initialization consists of setting up the outputs. Each output contains two key registers: the setup register, and the control register. These two registers for low current output 1 are shown in figure 5.7.

```
LC1_Setup.OutName = OutputNames::LC1;
LC1_Setup.OutputEnable = false;
LC1_Setup.CurrentLimit = 5; //in Amps
LC1_Setup.FusingTime = 10;
LC1_Setup.ResetEnable = false;
LC1_Setup.ResetAttempts = 10; //Must be non zero
LC1_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

LC1Control.FuseEnabled = false;
LC1Control.OutCurrent  = 0;
LC1Control.OutVoltage  = 0;
LC1Control.ResetAttemptsRemaining = LC1_Setup.ResetAttempts;
LC1Control.ResetTimeEnable = true;
LC1Control.FuseBuffer = false;
LC1Control.timecount = 0;
```

Figure 5.7: Setup and Control Registers

The user configures the power distribution controller by modifying the setup register. In the setup register:

"OutputEnable" sets if the output is used or not.

"CurrentLimit" sets the firmware current limit for the output.

"FusingTime" sets an additional delay in milliseconds between detection of an overcurrent condition and output shutdown.

"ResetEnable" determines if an output should attempt to turn on again after an overcurrent condition shut down the output.

"ResetAttempts" determines how many times the output should attempt to restart.

"ConditionNChannel" are references to Boolean values determined from the digital, analog, or CAN inputs.

The Control register tracks the state of the output, and based on the values in this register, the power distribution controller determines if the output should be on or off. In the control register:

“FuseEnabled,” is true if an over current condition was detected and the fusing time set in the setup register has passed since the over current detection.

“Fuse Buffer” is true if an overcurrent condition is detect but the fusing time has not yet passed.

“timecount” in the control register tracks the amount of time that has passed since the overcurrent condition was detected.

”OutCurrent” and “OutVoltage” hold the most recent output current and voltage reading respectively.

“ResetAttemptsRemaining” tracks how many more times an output could be reset after an overcurrent condition. Once this number reaches zero, the output will remain off.

“ResetTimeEnable” determines how long after an output is shut down due to an overcurrent condition, that the output should be reset.

Data Collection and Output Switching

The pdc.UpdateData(OutputCheckCounter) function in figure 5.6 collects output voltage and current readings for the outputs determined by OutputCheckCounter in figure 5.6, and then stores those values in the outputs’ respective control registers. The function then collects values for the circuit board temperature, battery voltage, and analog inputs, and stores those readings in other global registers. This function then calls a function called “UpdateConditions().” The “UpdateConditions()” function compares the current readings with the current limits set by the user and then sets the “FuseBuffer” flag to true if the current limit has been surpassed. This function also updates other Boolean values for battery voltage thresholds, received CAN data, and

digital input values. If the output setup register references any of these Boolean values in the “ConditionNChannel” elements, these updated values will later be used to determine if the output should be on or off.

After the “UpdateData()” function is called in main(), the “OutputIO()” function is called. In “OutputIO()”, between the setup and control register, if “FuseEnable” is false, “OutputEnable” is true, all of the “ConditionNChannel” references are true, the “ResetAttemptsRemaining” is greater than zero and “ResetTimeEnable” is true, the output turns on. If any of these stated conditions are not met, the output turns off.

CAN Transmission

After data has been collected and the output state of each output has been determined, all output voltages and currents are packaged into CAN messages, and sent over the CAN bus to the Formula SAE Vehicle’s data logger. Other data such as fuse states, input states, and circuit board temperature are also sent to the data logger at this time. After all data has been sent over CAN, the loop counter is reset to zero, and the data collection and output switching segment of the code is repeated.

6. Testing and Integration

Test and Integration Approach

The power distribution controller has several different elements both in hardware and software. The testing and integration approach was to first test individual elements of the controller, and then test the integration of those elements. The individual elements tested are the ADC reading accuracy, SPI communication, CAN communication, serial communication, digital to analog converter output, current reading accuracy, and input switch reading. Between these individual tests and the integration tests, all engineering specifications have been tested. Table 6.1 shows a summary of all engineering requirements being tested and their results. For further test information and analysis, refer to the “Test Summary” section. For test data that does not pertain directly to testing engineering requirements, refer to Appendix E.

Table 6.1: Engineering Requirements Test Summary

Test Name	Engineering Requirement Tested	Results
Product Cost	Prototype should be less than $200	Requirement Met
Emergency Shutdown	Emergency shutdown circuit operates in accordance to FSAE rules	Requirement met
High Current Output Power	High Current Outputs must source 15 A max at 14.4 V	Requirement Met
Low Current Power	Low Current Outputs must source 5 A max at 14.4 V	Requirement Met
Input Maximum Voltage	Inputs must accept up to 14.4 V	Requirement Met
Max Power	Deliver 600 W of power steady state at 40 A for one hour intervals with maximum 10 °F temperature rise	Requirement Not Met
Current Limit Accuracy	Current Limits must have 0.1 A precision	Requirement Met
Output Shutdown Time	Outputs must shutdown in less than 10 ms in an overcurrent condition	Requirement Met
Analog Redundancy Test	In the case firmware fails to shutdown the circuit in an overcurrent condition, analog redundancy circuit must shutdown the circuit	Requirement partially met
Interface Connector Cost	Interface connector must be less than $10 per connector	Requirement Met
Connector Properties Test	Terminals must accept 22 AWG wire, connector must be automotive rated and must be water resistant	Requirement Met
Operating Temperature Range	Product must operate between 40 °F and 120 °F	Requirement not Met
Firmware Current Limits	Current limits for outputs are set in firmware	Requirement Met
Output Toggle Conditions	All outputs can be toggled based on CAN or digital inputs	Requirement Met
Transmit CAN	Electrical system data and warnings must be sent over CAN	Requirement Met

Test Summary

Product Cost

If all parts and components were to be purchased directly, the total cost of the project would be $291.57 which would not meet the $200 specification limit. The cost of the PCB which is $5 per square inch, is the major item that makes the project over budget. However, many of the parts used in this project are manufactured by partners of Cal Poly Racing, the primary customer of this project. Therefore, after discounts for the customer are taken into account, this product costs approximately $102.25 to manufacture

Emergency Shutdown

Figure 6.1 shows the emergency shutdown circuit. When the section circled in red is grounded, the signal passes from the microcontroller to the MOSFET gate driver, however, when the output is left floating, the signal cannot pass between the microcontroller and the MOSFET gate driver, effectively disabling the high current outputs.

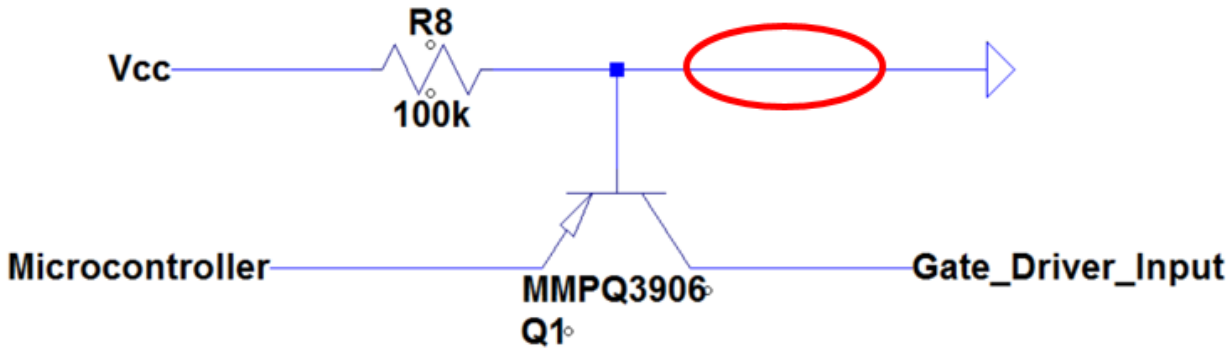

Figure 6.1: Emergency shutdown circuit

The emergency shutdown test initially failed. When the base of the BJT in figure 6.1 was grounded and the microcontroller signal was high, the voltage at the gate driver input was only 1 volt, which was not a high enough input to turn on the MOSFET. The expected input into the gate driver was 5 volts. The gate driver input voltage was so low because of the lack of a base resistor on the BJT leading to an excessive base

current and insufficient collector voltage. The wire circled in red in figure 6.1 is connected to a connector terminal and is grounded externally to the circuit board. The wire harness to this connector terminal was modified to include an inline 10 kΩ resistor. Figure 6.2 reflects this modification. Once this modification was made, the emergency shutdown circuit operated as expected.

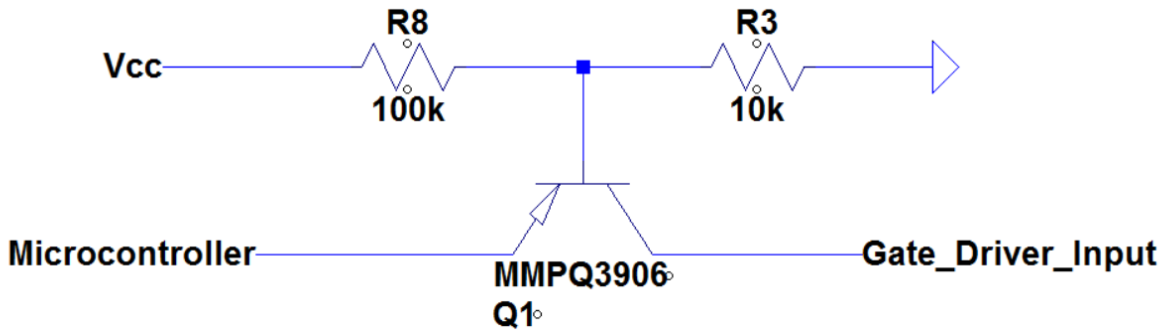

Figure 6.2: Modified Emergency shutdown circuit

High Current Output Power Test

The high current outputs were each individually connected to an electronic load with a 15 A constant current limit. The power distribution controller was powered by a DC power source set to 14.4 Volts and a 16 A current limit. Power distribution controller effectively delivered 15 A to the load at 14.2 Volts (200 mV drop across components). PCB traces heated approximately 15 °F as measured by a thermal camera, however, to the touch, the trace did not feel warm.

Low Current Power Test

The low current outputs were each individually connected to an electronic load with a 5 A constant current limit. The power distribution controller was powered by a DC power source set to 14.4 V and a 7 A current limit. Power distribution controller effectively delivered 5 A to the load at 14.2 Volts (200 mV drop across components). PCB traces heated approximately 15 °F.

Input Maximum Voltage Test

A voltage of 14.4 volts was applied to each digital input. With a voltmeter, the voltage at the microcontroller input was measured and found to be 5 V. This is the expected clamping voltage, and therefore, this test was successful.

Max Power Test

High current outputs 1, 2, and 3 were each connected to an electronic load set to 13 A constant current. The power distribution controller was then connected to DC power source set to deliver 14.4 volts and a current limit at 40 A. After 10 minutes of testing, PCB traces were extremely hot to the touch, and measured to be approximately 180 °F. Figure 6.3 shows a thermograph of the power distribution module during this test.

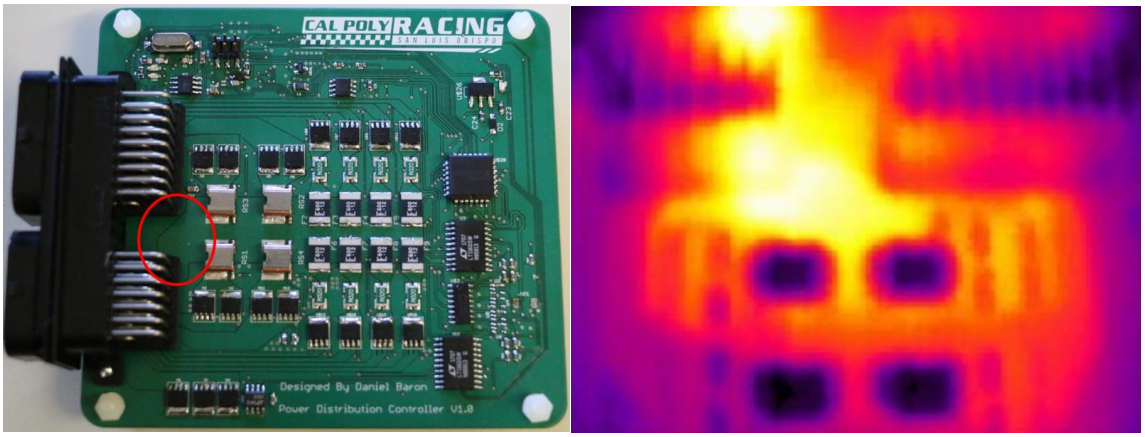

Figure 6.3: Thermograph of Power distribution controller

The thermograph on the right in figure 6.3 shows extreme heating in the region of the circuit board circled in the picture on the left. The trace that is heating excessively is the main power trace that is carrying 39 A during this test. While the output traces did not heat excessively when they were carrying 15 A during the High Current Output Power Test, the main power trace heated so significantly that the heat conducted through the current sense resistors (black boxes in the thermograph in figure 6.3) and into the output traces. In attempt to decrease heating of the circuit board, a small heatsink was added to this section of the circuit board. This heatsink had limited success with the traces heating to 165 °F instead of 180 °F. Figure 6.4 shows a thermograph of the test with a heatsink. While the heatsink (circled) cools the board

slightly, there is still a large section of the trace that is bright yellow showing significant heat.

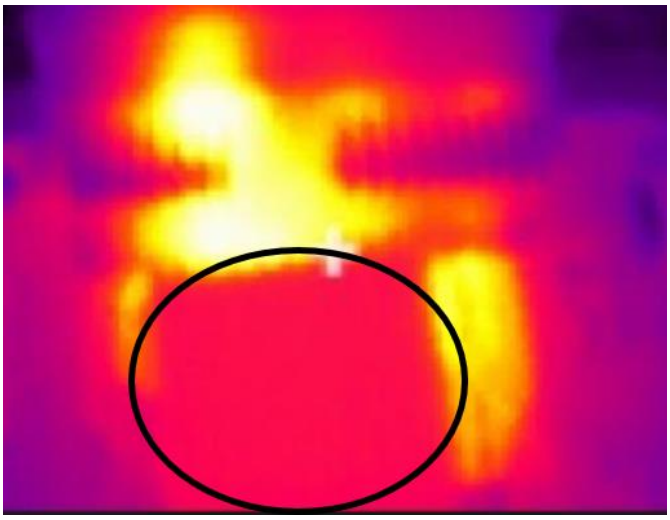

Figure 6.4: Thermograph of Max Power Test with heatsink. Circled section is the heatsink.

While this heating issue could not be solved on this circuit board, there are two solutions that are likely to rectify this issue. The first is to remove the solder mask on top of this trace and add solder to it in order to thicken the trace. The second solution would be to print the circuit board using a 4 ounce copper pour. The trace is approximately 0.6 inches wide, and if made using 4 oz copper, the expected temperature rise would be 16 °F [7].

Current Limit Accuracy Test

For this test a low current output was connected to a constant current load. The power distribution controller was powered by a 14.4 volt source with a 7 amp current limit. The load was stepped in 0.1 amp increments from 0 amps to 5 amps. The power distribution then sent the output current reading to the computer terminal via serial communication. This value was compared to the electronic load setting. The power distribution controller output current readings matched the electronic load settings for all data points. This test was then repeated for the high current output with the power distribution controller being powered by a 14.4 volt source with a 20 amp

current limit, and load currents were stepped in 0.1 amp increments from 0 amps to 15 amps. In this test, the power distribution showed zero current on the output between 0 amps and 0.7 amps, however, beyond 0.7 amps, the current reading was accurate. Because the high current outputs are designed for loads drawing more than 5 amps, the lack of readings between 0 amps and 0.7 amps is insignificant to the performance of the power distribution controller.

Output Shutdown Time Test

For this test, a high current output was connected to a constant current load of 2 amps, and the power distribution controller was powered by a 14.4 volt DC power supply with a 7 amp current limit. A firmware current limit of 5 amps was set for the output. An oscilloscope was connected to the output of the current sense amplifier on channel 1, and was then connected to the output on channel 2. The oscilloscope was placed in single sweep mode with a rising edge trigger. After starting the test, the constant current load was changed from 2 amps to 5.2 amps. After making this change in the load, the current sense amplifier output had a rising edge which triggered the single sweep, and channel 2 then showed the falling edge of output turning off. The time from the current sense amplifier rising edge to the output turning off falling edge is the output shutdown time which was determined to be 7 ms. Figure 6.5 shows the results of this test on an oscilloscope.

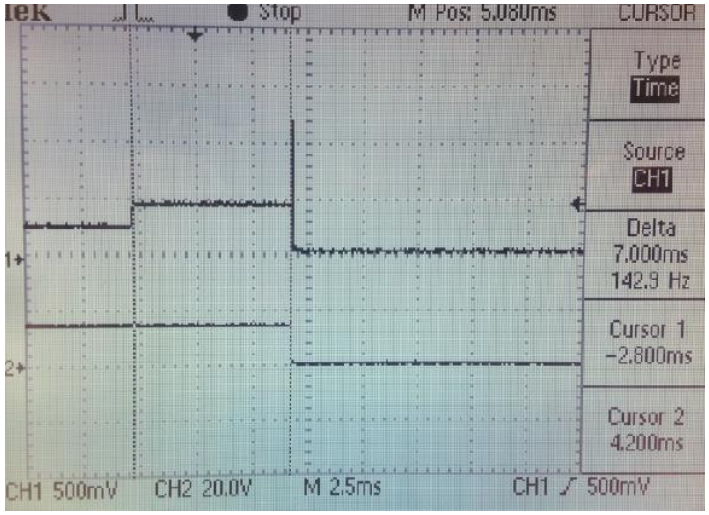

Figure 6.5: Oscilloscope image of the output shutdown test time.

Analog redundancy Test

The power distribution module has two analog redundancy circuits, one for the low current outputs, and one for the high current outputs. The analog redundancy test was successful for the low current outputs, but the requirement was not met for the high current output redundancy. The low current redundancy simply uses a PTC resettable fuse that fuses at 6 amps after two seconds. The high current output redundancy uses a more complex feedback circuit that assesses the output of the current sense amplifier via a hysteretic comparator, and if the comparator detects that the output has surpassed 20 amps, the comparator turns on, which then drains the gates of the output MOSFETs.

Figure 6.6 shows the high current output redundancy circuit. Due to the lack of a base resistor on the output of the hysteretic comparator and base of the BJT, the gate of the output MOSFET was only reaching approximately 1 volt when it was turned on during normal operation. In order to test the rest of the high current output operation, the high current redundancy circuit had to be removed from the circuit board by removing the BJT array that contains Q2 in figure 6.6.

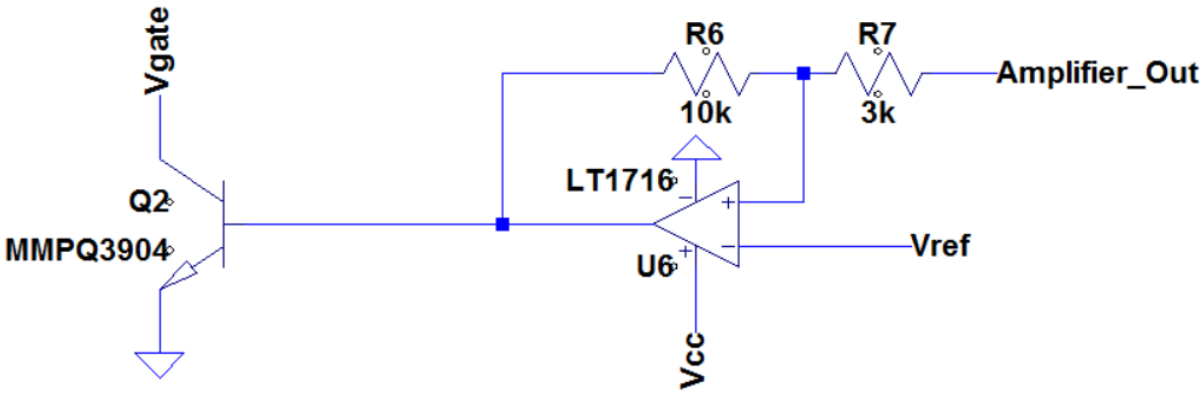

Figure 6.6: High current output analog redundancy circuit

The reference voltage for the hysteretic comparator in figure 6.6 is provided by a digital to analog converter. The digital to analog converter provides a 5 volt reference to the comparator to disable it when the MOSFET turns on, in order to allow an inrush

current through the output for inductive loads, and then re-enables after 10 ms. With the removal of the BJT array, the digital to analog converter and hysteretic comparator operated as expected.

Connector Cost and Properties

The connectors cost $8.75 for the 26 position connector and $7.16 for the 34 position connector. The Super Seal connectors are sealed connectors, and are designed for automotive use.

Operating Temperature Range Test

The operating temperature range was not explicitly tested, however, due to the failure of the maximum power test, the operating temperature range test inherently fails. During the maximum power test, the room temperature was 74 °F. At this ambient temperature, the main power PCB trace overheated which indicates that the product cannot operate properly at 74 °F, and therefore will not operate properly at 120 °F.

Firmware Current Limit Test

For this test, low current and high current outputs were connected to constant current loads one at a time, and current limits were set in firmware for the outputs. The constant current load was set to draw 0.1 amps below the firmware current limit, and then after the test began, the load was set to draw 0.1 amps above the current limit. During all tests, the firmware successfully shutdown the outputs when the firmware current limit was exceeded.

Output Toggle Conditions Test

This test ensured that analog, digital, and CAN, inputs into the system, can toggle a Boolean value, and that the Boolean value can toggle an output. To test this, a low current output was set to toggle based on the digital 1 input. A toggle switch was then

connected to the digital 1 input. A voltmeter read the voltage on the low current output. When the toggle switch was toggled to the on position, the output turned on as indicated on the voltmeter. When the toggle switch was toggled to the off position, the output turned off. This test was then repeated, however, instead of using a toggle switch as the input, the power distribution controller was connected to the Formula SAE vehicle's CAN bus. The data logger on the Formula SAE car was set to send a simulated RPM value to the power distribution controller. The power distribution controller was set to keep a Boolean value at false if the RPM was below 100 RPM, and then change to true when the RPM value exceeded 100 RPM. The low current output was then set to toggle based on this Boolean value. When a RPM of 0 was sent to the power distribution controller, the output was off, however, when 3000 RPM was sent to the power distribution controller, the output turned on.

Transmit CAN Test

For this test, the power distribution controller was connected to the Formula SAE CAN bus. The data logger was set up to read output current and voltage data from low current output 1 and high current output 2. The power distribution controller was turned on with both outputs enabled. The data logger data was monitored live and the current showed 0 amps and the output voltage showed 13.2 volts as expected. The test was run again with both outputs disabled, and both outputs read 0 amps and 0 volts on the data logger readout. With this successful CAN transmission, the power distribution controller is able to send both data and warnings to the data logger.

7. Conclusions

This project has had a mix of successes as well as limitations. The current, voltage, temperature, and input detection as well as the CAN communication, code, and overall build quality of the power distribution controller are very successful. However, the two major drawbacks to this product now is the inability to deliver maximum power and the nonfunctioning analog redundancy for the high current outputs. While these two drawbacks do pose an issue for implementing this controller on the Formula SAE car, this first revision of the power distribution controller has provided an excellent testbed for testing hardware as well as the logic in the software at lower currents than the maximum 40 A specification. In a second revision board, only these two drawbacks must be solved. The analog redundancy issue can likely be solved by adding a base resistor to the BJT in figure 6.6, and the overheating issue at maximum power can be solved by either using a thicker copper pour such as four ounce copper, or adding solder on top of the main power input trace circled in figure 6.3. With just a couple corrections on a revision two printed circuit board, the power distribution controller can be fully functioning and high performing controller on the formula SAE vehicle.

References

[1] "2016 Formula SAE Rules," in *Formula SAE*, SAE International, 2015. [Online]. Available: http://www.fsaeonline.com/content/2016_FSAE_Rules.pdf. Accessed: Oct. 24, 2016.

[2] Tim, "prices_ecu_motec," in *Capa Performance*, 2016. [Online]. Available: http://www.capa.com.au/prices_ecu_motec.pdf. Accessed: Oct. 24, 2016.

[3] "MoTeC," in *MoTeC Engine Management and Data Acquisition Systems*, 2008. [Online]. Available: http://www.motec.com/home. Accessed: Oct. 24, 2016.

[4] "CARTEK Power Distribution Modules," in *CARTEK Motorsport electronics*, 2016. [Online]. Available: http://www.cartekmotorsport.com/pdm.html. Accessed: Oct. 24, 2016.

[5] "SAE Collegiate Chapters," in *SAE International*, 2016. [Online]. Available: http://www.sae.org/servlets/collegiate?PAGE=getCollegiateMainPage&OBJECT_TYPE=CollegiateChapInfo. Accessed: Oct. 24, 2016.

[6] "Official Results," in *Formula Student*, Institution of Mechanical Engineers, 2016. [Online]. Available: http://formulastudent.imeche.org/docs/default-source/default-document-library/download-the-final-overall-class-1-results.pdf?sfvrsn=0. Accessed: Oct. 24, 2016.

[7] B. Suppanz, “Trace Width Website Calculator,” in Advanced Circuits, 2007. [Online]. Available: http://www.4pcb.com/trace-width-calculator.html, accessed Jun. 10, 2017. Accessed: Jun. 10, 2017.

[8] “M400/M600/M800/M880 User’s Manual,” in Motec Pty Ltd, 2003. [Online]. Available: http://www.motec.com/m400/m400downloads/. Accessed: Jun. 10, 2017.

Appendix A: Senior Project Analysis

Project Title: Formula SAE Power Distribution Controller

Student Name: Daniel Baron

Advisor's Name: Professor Bridget Benson

Summary of Functional Requirements:

This project accepts a power input from the Formula SAE car's battery, and distributes the power to the various electronics on the vehicle. Current limits for each output are set in firmware. If and when an over current condition, as defined by the user, occurs on an output, the device must immediately turn off the output on which the overcurrent condition occurs. Inputs into the system include CAN and digital inputs. Outputs will be able to toggle on and off based upon input values. Output current and voltage data will also be sampled and sent over CAN to the car's data logger.

Primary Constraints:

First and foremost, this project must abide by the Formula SAE rulebook. This especially affects the emergency shutdown circuitry that disables the high current outputs of the system. The system must also be designed to dissipate heat from the high currents flowing through PCB traces, and be able to operate in ambient temperatures up to 120°F. Finally, the entire system must be light weight, as low mass is important to a competitive racecar. Finally, the system must cost less than $200 to build.

Economic:

What economic impacts result?

Human Capital: This device will create jobs in engineering, software development, manufacturing, sales, and technical support. Additionally, customer support jobs will be created to teach race teams how to implement and use the product.

Financial Capital: This product will create minimal business for suppliers due to a limited market, however, this product will have a large financial impact for the customer. Customers currently pay $3000 to $5000 dollars for a similar product. Undercutting the market price significantly will create a large market for this product.

Manufactured or Real Capital: The manufactured capital will be the inventory of this product.

Natural Capital: The ICs used on this device require a significant amount of water while being manufactured. Additionally the raw materials that create the components, circuit boards, and enclosures come from natural resources.

When and where do costs and benefits accrue throughout the project lifecycle?

Costs to the project team occurs during the design and manufacturing stages of the project lifecycle through research and development, labor costs, and material costs. When the product is sold, the customer has an upfront purchase cost. The customer will then see the benefits through the rest of the life of the product before for the product is ultimately disposed of.

What inputs does the experiment require? How much does the project cost? Who pays?

Table A.1: Original cost estimate

Item	Quantity	Cost
Circuit Board Components	1	$120
Printed Circuit Board	1	$0
Connectors, Enclosure, and Hardware	-	$2
Extra Expenses (Shipping, re-orders, extra components)	-	$80
Total		$202

While different parties will be assuming the costs at different points in the product lifecycle, ultimately, the cost of this project will come from the EE department through the $200 grant they give to senior projects. If manufactured commercially, the customer would be assuming all the costs when they buy the product.

How much does the product earn? Who profits?

This project is being funded through the EE department grants, and donations. Furthermore, this project is being given to Cal Poly Racing, a nonprofit organization. Therefore, this project does not profit, however, Cal Poly Racing profits from new equipment, and the parts suppliers profit from purchasing parts from them.

Timing

Products would emerge after 20 weeks combined of design, manufacturing, and testing. The products are then expected to last 10 years at a minimum. In order for the product to last this long, some soldering and components may need to be repaired over time due to the amount of vibration and heat generated by the Formula SAE car. After the project is complete, the project will be given to Cal Poly Racing in order to implement on the 2018 vehicle. Product training for integration and use will occur after the 2017 Formula SAE competition.

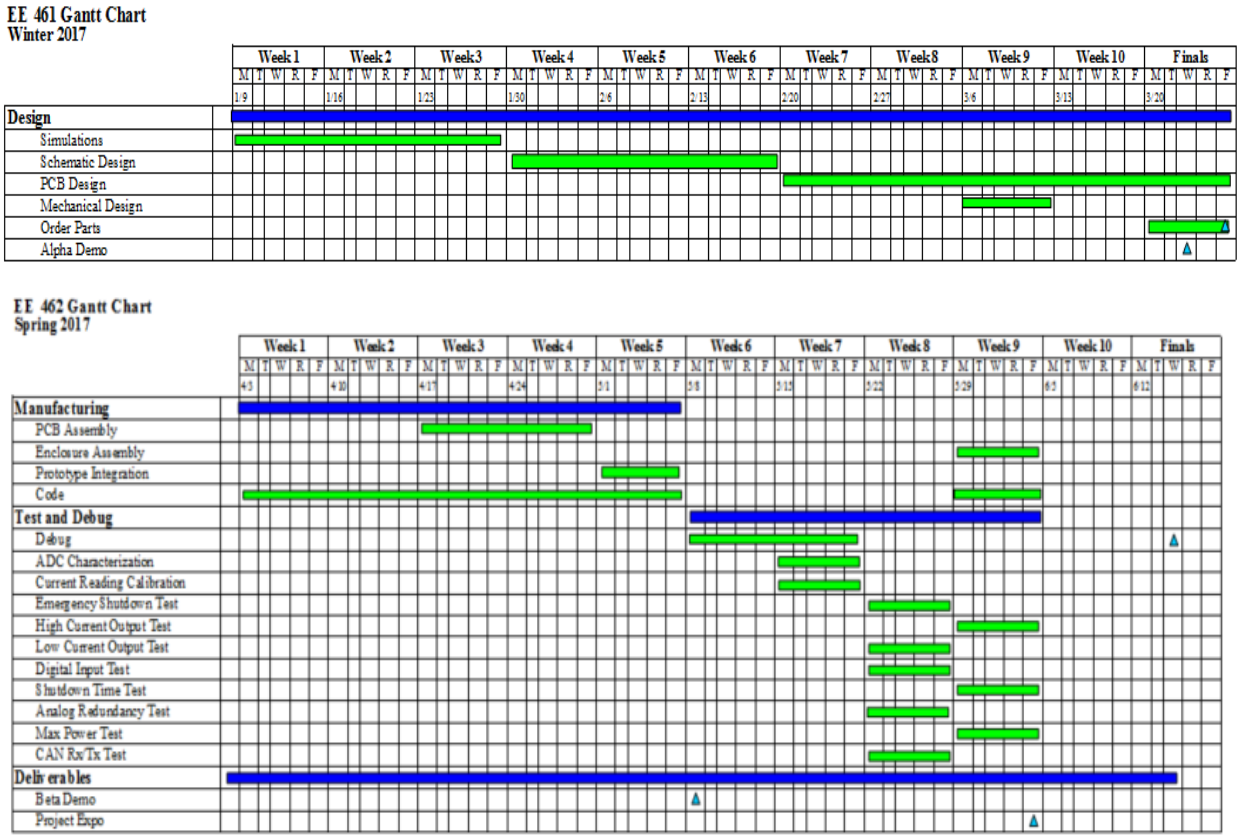

Figure A.1: Estimated development time

If Manufactured on a Commercial Basis:

Estimated number of devices sold per year: 200

Estimated manufacturing cost for each device: $110 (this considers economies of scale and labor)

Estimated Purchase price for each device: $400

Estimated profit per year: $29,000

Estimated cost for user to operate device: Operating the device for one hour will cost the user approximately $0.072 per hour. This assumes $0.12 per Kilowatt-hour.

Environmental:

This product mostly affects the environment at the beginning and end of the product lifecycle, and has a minimal environmental impact during its use. At the beginning of the lifecycle, energy, raw materials, and consumable materials put into manufacturing the ICs, circuit boards, and enclosures for the product taxes upon the environment in water

usage, waste, and emissions. During the use of the product, the product indirectly contributes to air pollution because its intended use is to help control internal combustion vehicles. At the end of the lifecycle, the product is disposed of, and affects the cleanliness of the environment because it would most likely end up in a landfill. While this product does cause some harm to the environment, it does also improve the environment. This product will be more effective at protecting electronics on the Formula SAE vehicles, so there will be less damage to the electronics. The electronics on the vehicles then will have extended lifecycles, and less electronic waste will be in the landfills. For the areas of the environment that this product negatively impacts, the species residing in those areas will see some impact in their habitat as the impact of this project combines with the impact of other projects on the environment.

Manufacturability:

Manufacturing will be difficult due to the environment the product will be used in. All solder joints must be of very high quality due to significant amount of vibration that the circuit board will be experiencing under normal conditions. Furthermore, the enclosure needs to be manufactured so that it is water and dust resistant which will be difficult due to the small tolerances this specification requires.

Sustainability

Describe any issues or challenges associated with maintaining the completed device or system.

The environment in which the device is operated will pose as a challenge to maintaining the device. This device must be designed to withstand vibration, heat, impact, water, and dust. Furthermore, due to the significant amount of power flowing through this device, this device must effectively dissipate the generated heat to ensure the longevity of the internal components, however, this device must also be lightweight due to it being used on racecars.

Describe how the project impacts the sustainable use of resources.

While this product taxes upon resources upfront through the manufacturing process, overall, this product improves the sustainable use of resources. This product not only distributes power to various electronics, but it also protects those electronics from overcurrent and overvoltage conditions. This will ultimately result in fewer damaged electronic devices that will then result in fewer electronic devices being disposed of, and fewer electronic devices being manufactured to replace the older devices.

Describe any upgrades that would improve the design of the project.

This project can be upgraded by making the internal circuit boards and components very accessible. This will allow the device to be serviceable, and would result in a longer

product life cycle. Furthermore, reducing product weight and size would be advantageous because weight is a very big consideration when choosing devices to put on a racecar, and a smaller size will help with packaging the device in the racecar. Finally, a graphical user interface would be an advantageous addition to this project so that users do not need to interact with the code directly in order to operate this project.

Describe any issues or challenges associated with upgrading the design:

Upgrading the design to be modular would pose a challenge because even when the device is designed to be taken apart easily, it must still be water resistant and durable. Upgrading the design to be smaller and lighter would pose issues because the circuit board would need to still fit all of the same components. Secondly, the circuit board must be large enough so that the trace width could be wide enough for the device to stay cool. Finally, making the device lighter would prove to be difficult because the device must be cooled, and a significant amount of the weight of the system is designated to cooling the device.

Ethical

This project has several ethical implications. First, the device must be designed to follow the Formula SAE competition rules, even if simpler design choices could be made by not following the rules. Secondly, the device must be reliable and thoroughly tested. Not only is this device being used in a competition, and the customer expects a reliable product, this product also contains key safety shutdown circuitry that disables the engine of the vehicle in the case of an emergency, including, but not limited to a fire, crash, or loss of brake pressure. The current limiting circuitry must also work reliably because the customer is relying on this device to protect $10,000 to $15,000 worth of electronics from damage. Another important ethical consideration is that this product is documented properly. When the customer receives this product, the customer must have the documentation on how to implement the device into the vehicle and how to use it. Without that documentation and knowledge, the customer would invest time and money into this product, and then would be unable to use it. Finally, while this product is designed to be used in a motorsports application, the designers have no control over how the customer uses the devices. A customer could choose to misuse the product, and use it on a harmful system that causes damage to people and/or property.

Health and Safety

The safety concerns of this product revolve primarily around the design and use of the device. Specifically, the emergency shutdown circuitry must always work flawlessly as it is responsible for disabling the engine of the car in the case of an emergency. If this circuitry ever fails, it could put the driver and surrounding spectators in serious danger of injury.

Social and Political

This product has social implications associated with it because this product will seriously affect the motorsports electronics industry. This product impacts both motorsports electronics customers, and competing companies and the employees at these companies. This product significantly undercuts the price of competing products, and the customers will greatly benefit from the lower product cost. Likewise, competitors may lose business do to their products now being too expensive. This may affect the pay and livelihood of employees at the other companies. While all customers would pay equally for this product, this product does create inequities. This device requires programming knowledge in order to successfully implement this solution on a racecar. Therefore, individuals who have programming knowledge would be able to buy this low cost solutions, while individuals without programming knowledge would probably have to purchase the more expensive versions the competitors offer.

Development

This project will require development in knowledge of thermal dissipation of electronic components, thermal transfer and cooling, and how, quantitatively, heat affects the operation of various electronic components.

Appendix B: Hardware Schematics

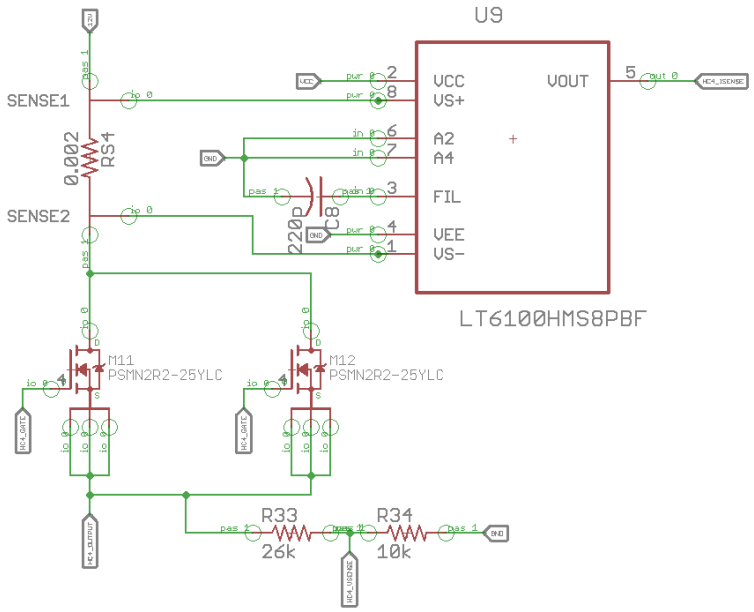

Figure B.1: High Current Output circuit containing the current sense resistor (RS4), two MOSFETs in parallel (M11 and M12) and the current sense amplifier (U9). There are four of these circuits total on the circuit board.

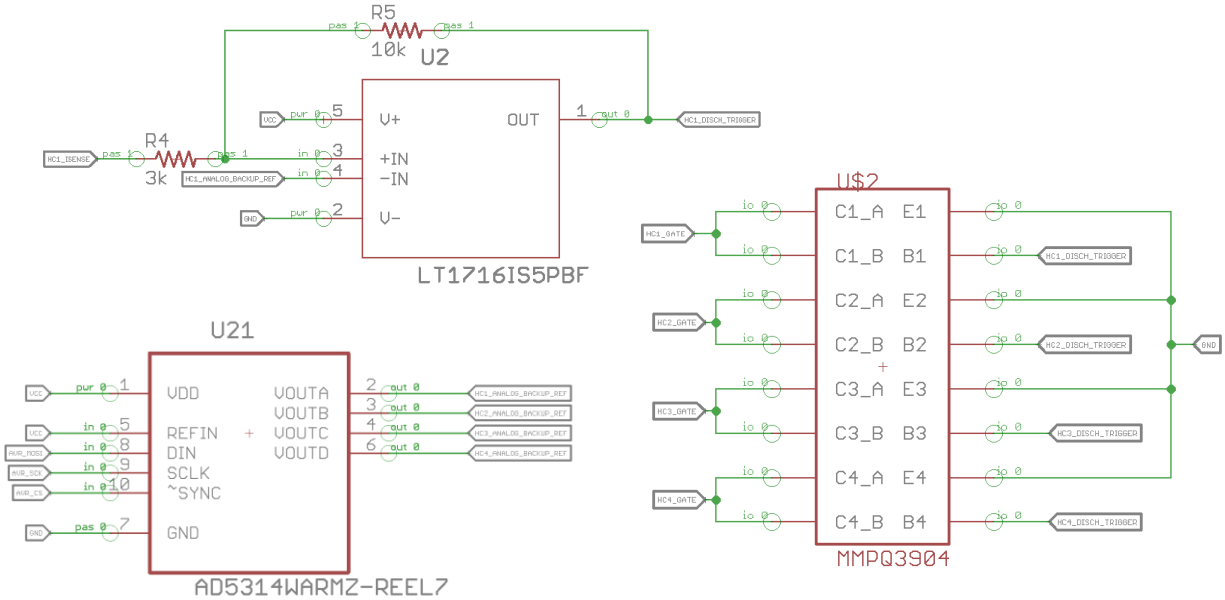

Figure B.2: High Current Analog Redundancy Circuit. The output of the current sense amplifier from Figure B.1 is input into the hysteretic comparator based on (U2). A reference voltage provided by the DAC (U21) is also sent to the hysteretic comparator. If the comparator is triggered, a BJT in the BJT array (U$2) turns on and drains the gait voltage from the MOSFETs (M11 and M12 in Figure B.1).

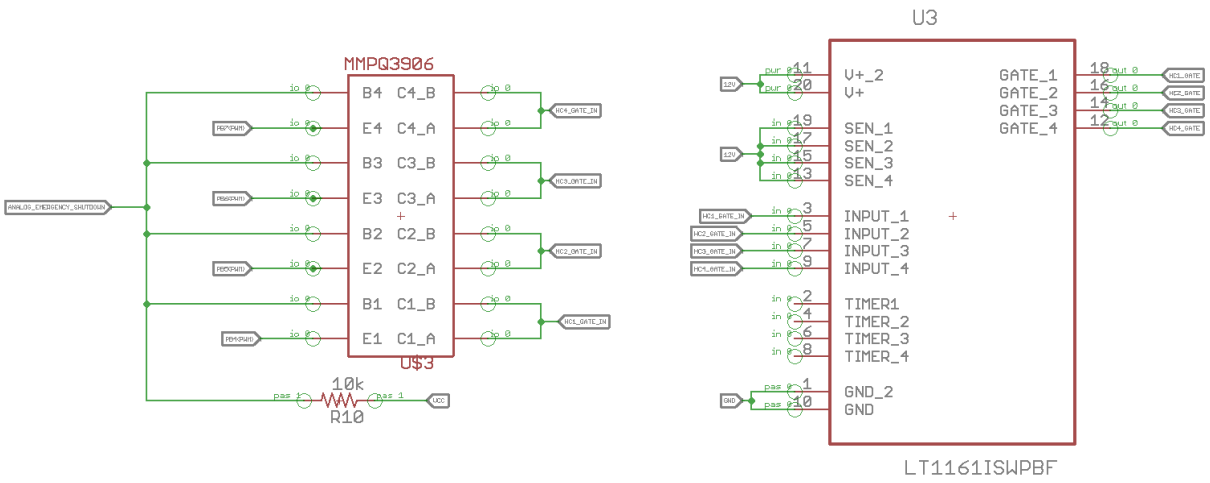

Figure B.3: Analog shut down circuitry (U$3) and gate driver for high current output (U3)

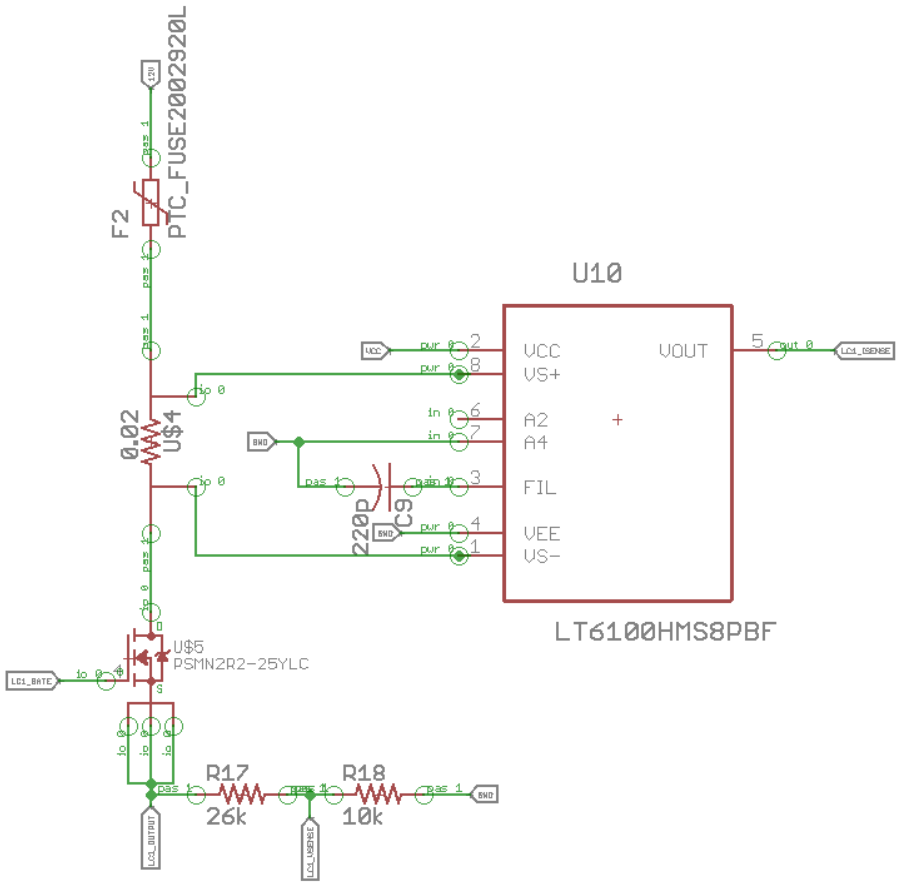

Figure B.4: Low current output containing PTC fuse (F2), current measuring resistor (U$4), output MOSFET (U$5), and current sense amplifier (U10)

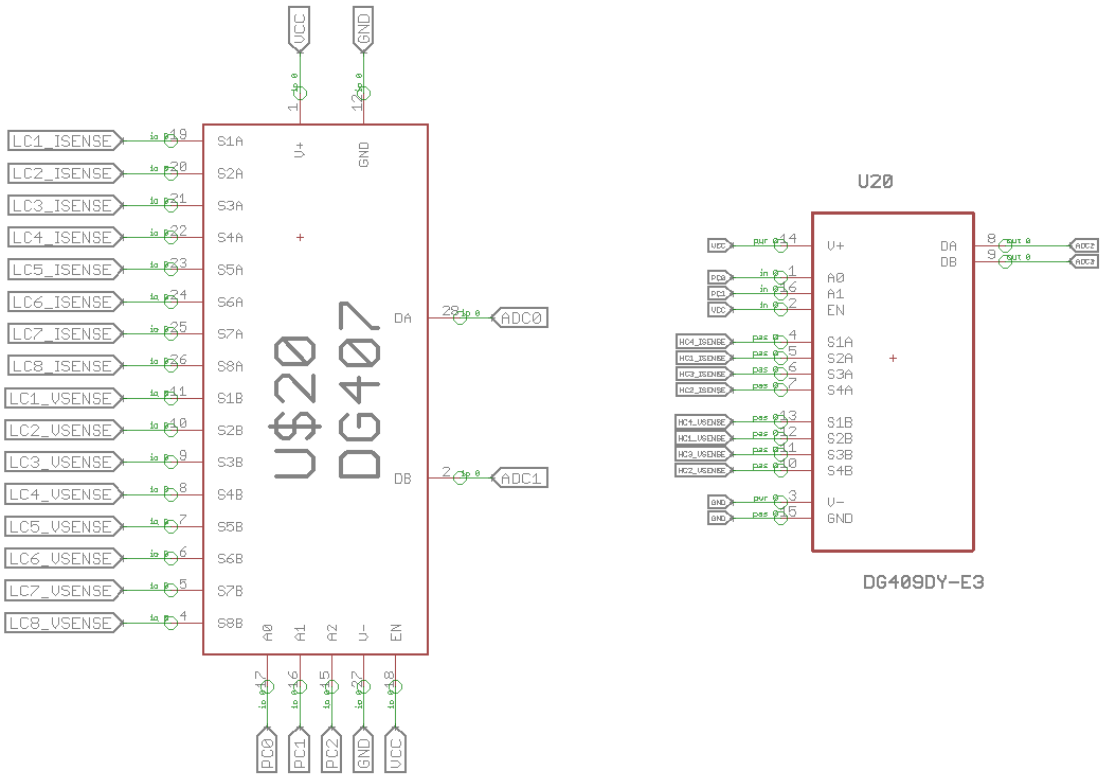

Figure B.5: Signal to ADC multiplexing for high current outputs (U20) and low current outputs (U$20)

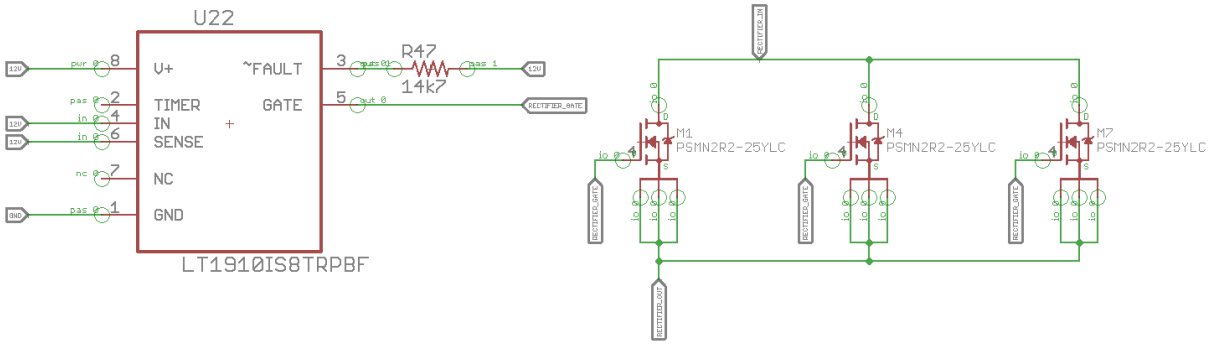

Figure B.6: Charging rectifier shutdown circuit containing gate driver (U22) and output MOSFETS (M1, M4, M7)

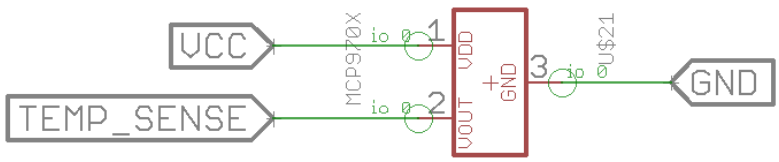

Figure B.7: Temperature sensing

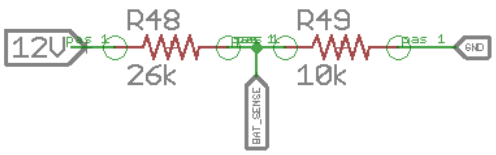

Figure B.8: Battery voltage measurement circuit

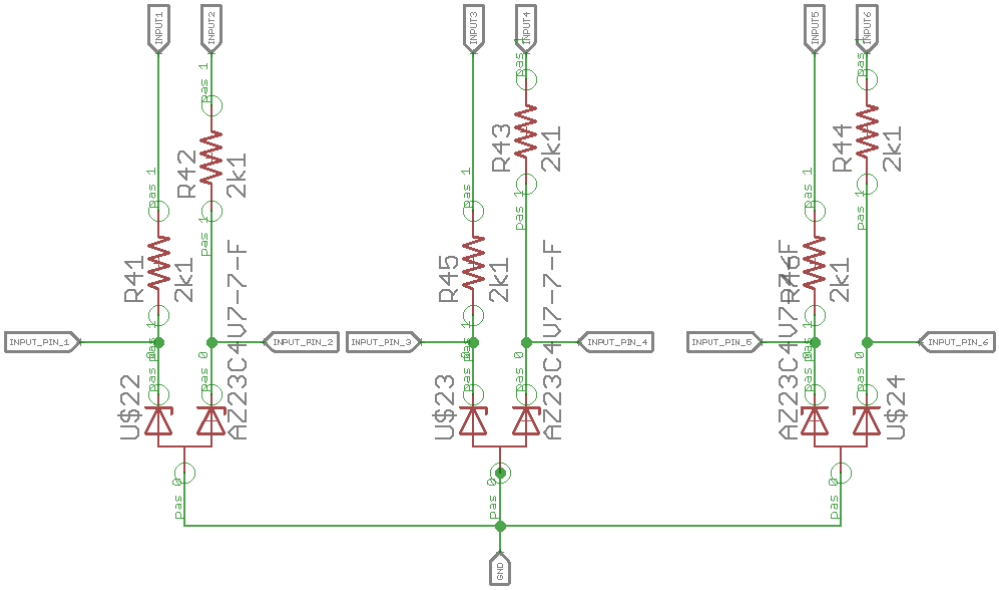

Figure B.9: Input current and voltage limiting

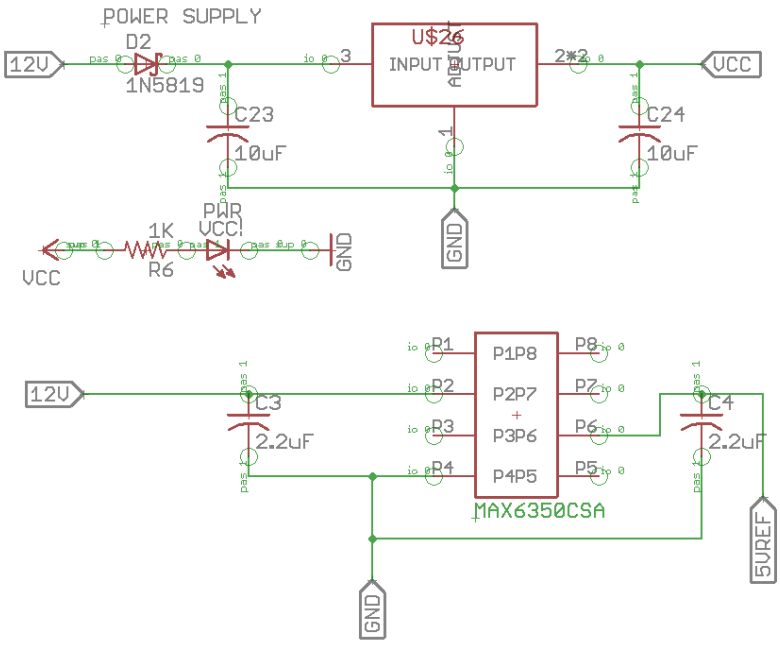

Figure B.10: 5 V linear regulator (U$26) and the voltage reference for the ADC (MAX6350CSA). The PWR LED indicates when the Power Distribution Controller is on.

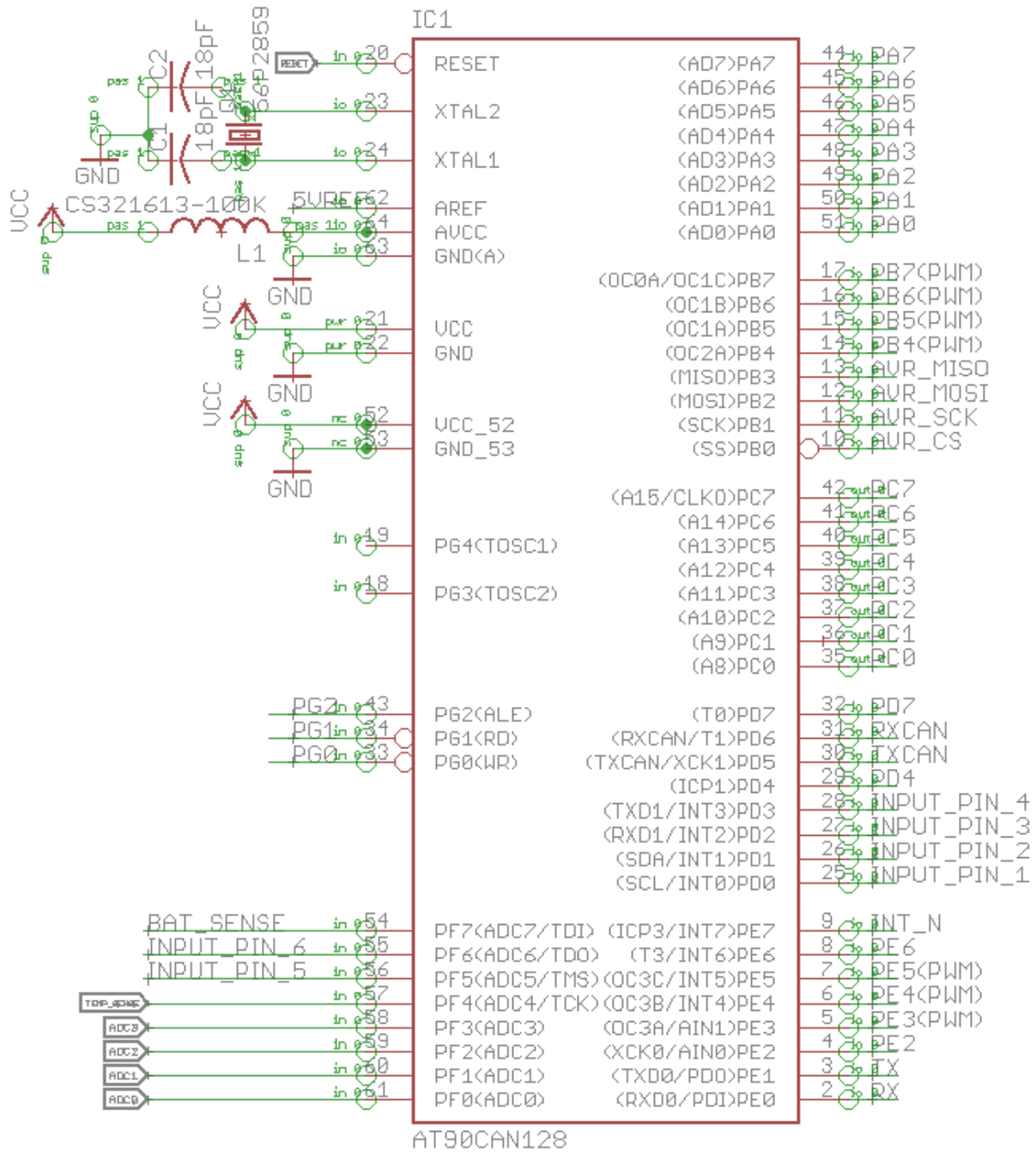

Figure B.11: AVR connections

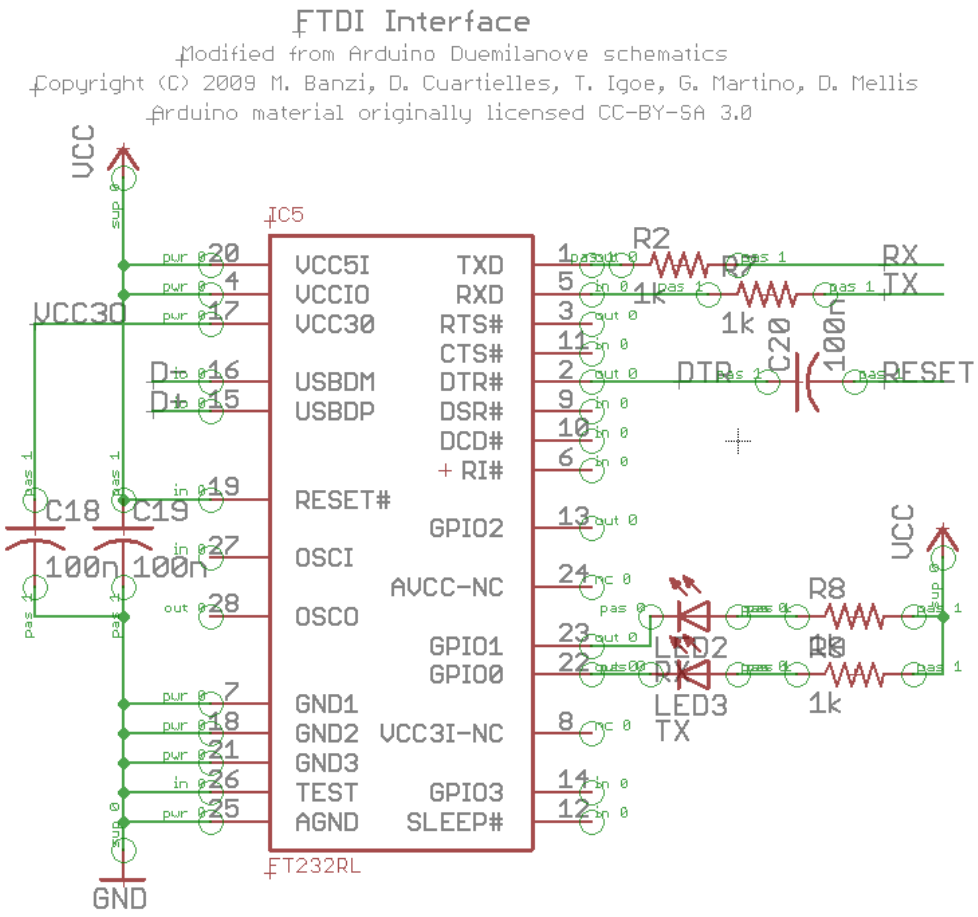

Figure B.12: FTDI USB to UART interface

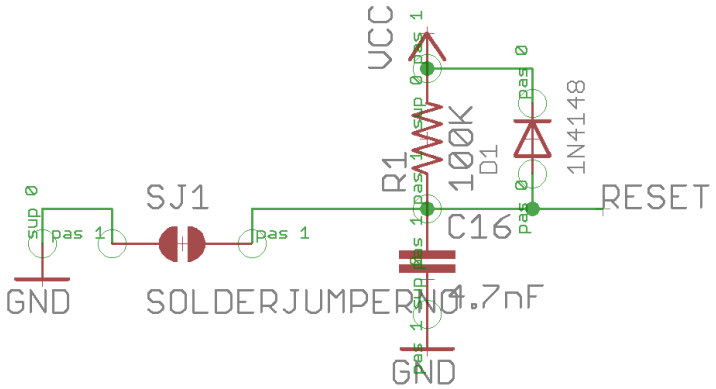

Figure B.13: AVR Rest Circuit

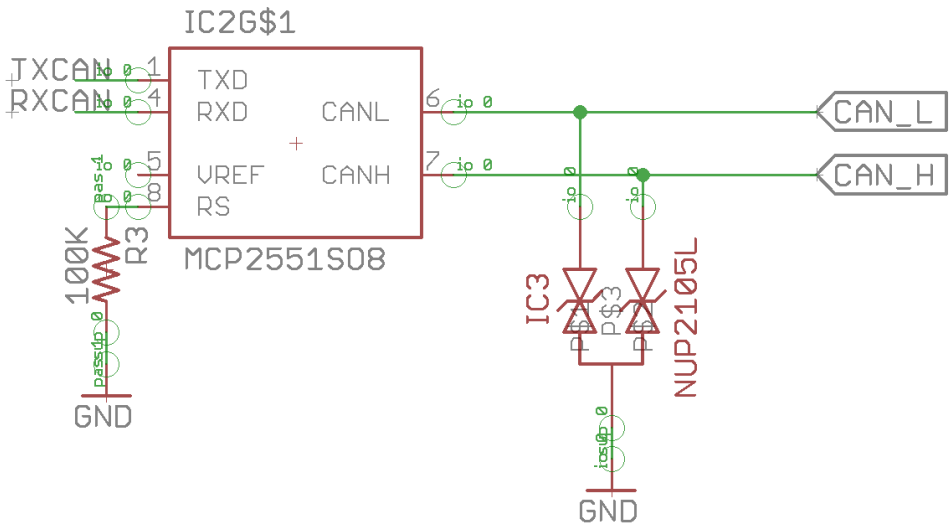

Figure B.14: CAN transceiver

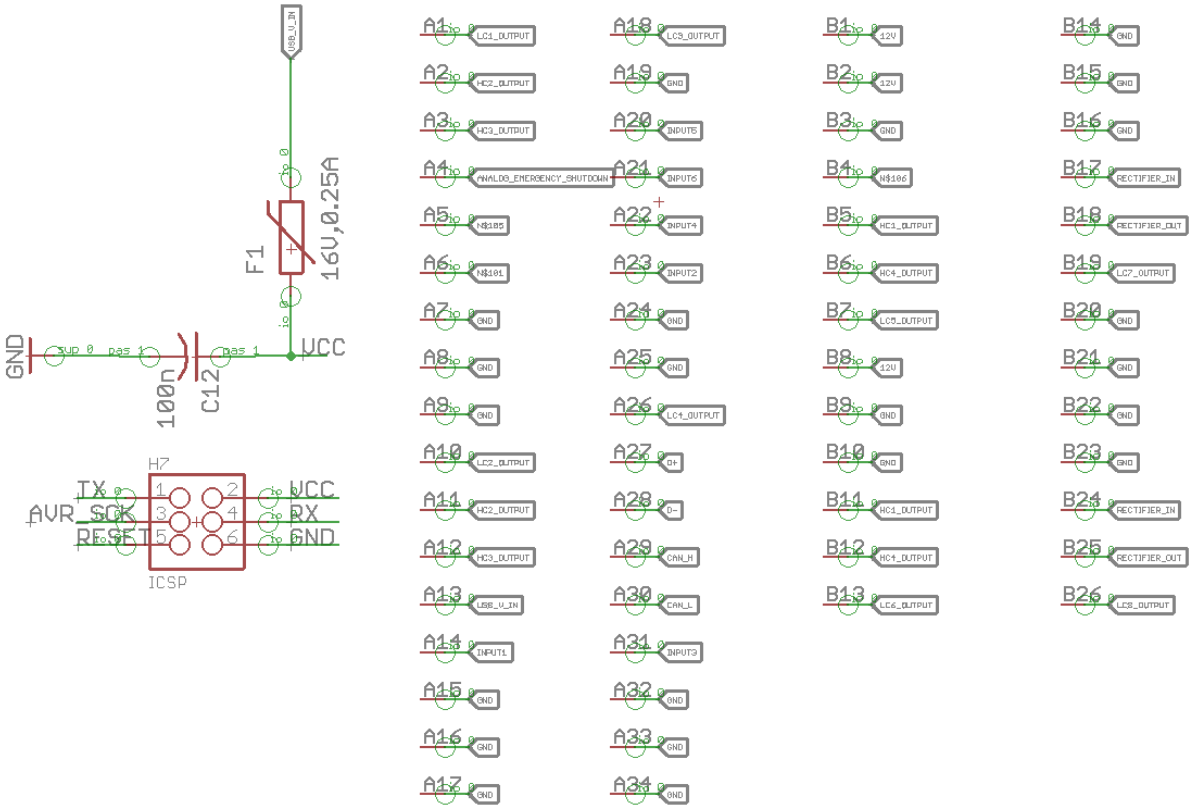

Figure B.15: Connectors and USB circuit protection

Appendix C: Bill of Materials

Table C.1: Bill of Materials

Qty	Part	Manufacturer's Part Number	Price per Unit	Total Price
4	WSL2726_RES	WSL2762L000FEB	$2.39	$9.56
8	LVK_SERIES	LVK20R020DER	$1.19	$9.55
2	R-US_R0805	ERJ-P06F-1003V	$0.19	$0.38
4	C-USC0805	C0805C104K5RACAUTO	$0.15	$0.60
18	R-US_R0603	RC0603RF-0710KL	$0.02	$0.27
2	C-USC0603	GRT188R61C106ME13D	$0.47	$0.94
1	R-US_R0603	RC0603FR-0714K7L	$0.10	$0.10
1	PTCSMD	FUSE PTC RESET 16 V .25 A 1206	$0.50	$0.50
2	C-USC0805	CL21C180JBANNNC	$0.10	$0.20
5	R-US_R0805	RC0402FR-071KL	$0.01	$0.07
1	DIODESOD-123	1N4148W-TP	$0.14	$0.14
1	1N5819-BSOD123	1N5819HW-7-F	$0.49	$0.49
2	C-USC0603	EMK107BJ225KA-T	$0.12	$0.24
12	C-USC0603	C0603C221F5GACTU	$0.18	$2.16
13	R-US_R0603	RT0603BRE0726k1L	$0.28	$3.61
6	R-US_R0603	RC0402FR-072K1L	$0.01	$0.08
4	R-US_R0603	RC0603JR-73KL	$0.01	$0.04
1	CAP0805	CL21B472KB6WPNC	$0.10	$0.10
1	CRYSTALSMD	ABLS-16.000MHZ-B4-T	$0.25	$0.25
1	AD5314WARMZ-REEL7	AD5314ARMZ-REEL7	$7.06	$7.06
1	AT90CAN128	AT90CAN128-16AUR	$7.66	$7.66
3	AZ23C5V6-7-F	AZ23C5V6-7-F	$0.44	$1.32
1	L-USL3216C	CS321613-100K	$0.34	$0.34
1	DG407	DG407DN-E3	$8.04	$8.04
1	DG409DY-E3	DG409DY-E3	$3.31	$3.31
1	FT232RL	FT232RL-REEL	$4.50	$4.50
1	M03X2SMD_FCI	20021121-00006C4LF	$0.53	$0.53
3	LT1161ISWPBF	LT1161ISW#PBF	$7.84	$23.52
4	LT1716IS5PBF	LT1716IS5#PBF	$2.93	$11.72
1	LT1910IS8TRPBF	LT1910IS8#TRPBF	$4.52	$4.52
12	LT6100HMS8PBF	LT6100HMS8#PBF	$2.96	$35.52
1	MAX6350CSA	MAX6350CSA+	$9.78	$9.78
1	MCP2551SO8	MCP2551T-E/SN	$1.22	$1.22
1	MCP970X	MCP9700T-E/TT	$0.26	$0.26
1	MMPQ3904	MMPQ3904	$1.55	$1.55
1	MMPQ3906	MMPQ3906	$1.69	$1.69
1	NCP1117	NCP1117ST50T3G	$0.50	$0.50
1	NUP2105L	NUP2105LT1G	$0.44	$0.44
19	PSMN2R2-25YLC	PSMN2R2-25YLC, 115	$0.90	$17.10
8	PTC_FUSE2002920L	2920L600	$0.85	$6.80
1	LEDCHIPLED_0805	LG R971-KN-1	$0.25	$0.25
1	LEDCHIPLED_0805	LY R976-PS-36	$0.35	$0.35
1	SUPERSEAL1.0_60POS	2-6437285-2	$15.95	$15.95
1	LEDCHIPLED_0805	LY R976-PS-36	$0.35	$0.35
1	PCB	PCB	$96.00	$96.00
1	Enclosure	Enclosure	$0.00	$0.00
1	8X 4-40 Bolt, 8X Nut		$2.00	$2.00

Appendix D: Wiring Guide and Device Setup

Table D.1: Pin Functions List

Connector A: 34 Position		**Connector B: 26 Position**	
Pin Number	Function	Pin Number	Function
1	Low Current 1 Output	1	Battery +
2	High Current 2 Output	2	Battery +
3	High Current 3 Output	3	Rectifier Ground
4	Emergency Shutdown	4	No Connect
5	No Connect	5	High Current 1 Output
6	No Connect	6	High Current 4 Output
7	Low Current 1 Ground	7	Low Current 5 Output
8	Low Current 2 Ground	8	Battery +
9	High Current 1 Ground	9	Rectifier Ground
10	Low Current 2 Output	10	Low Current Output 8 Ground
11	High Current 2 Output	11	High Current 1 Output
12	High Current 3 Output	12	High Current 4 Output
13	USB Voltage	13	Low Current 6 Output
14	Digital Input 1	14	Battery- (Ground)
15	Low Current 3 Ground	15	High Current 4 Ground
16	Low Current 4 Ground	16	High Current 3 Ground
17	High Current 1 Ground	17	Rectifier In
18	Low Current 3 Output	18	Rectifier Out
19	USB Ground (Ground)	19	Low Current 7 Output
20	Digital/Analog Input 5	20	Battery- (Ground)
21	Digital/Analog Input 6	21	Battery- (Ground)
22	Digital Input 4	22	High Current 4 Ground
23	Digital Input 2	23	High Current 3 Ground
24	Low Current 5 Ground	24	Rectifier In
25	High Current 2 Ground	25	Rectifier Out
26	Low Current 4 Output	26	Low Current 8 Output
27	USB D+		
28	USB D-		
29	CAN High		
30	CAN Low		
31	Digital Input 3		
32	Low Current 6 Ground		
33	Low Current 7 Ground		
34	High Current 2 Ground		

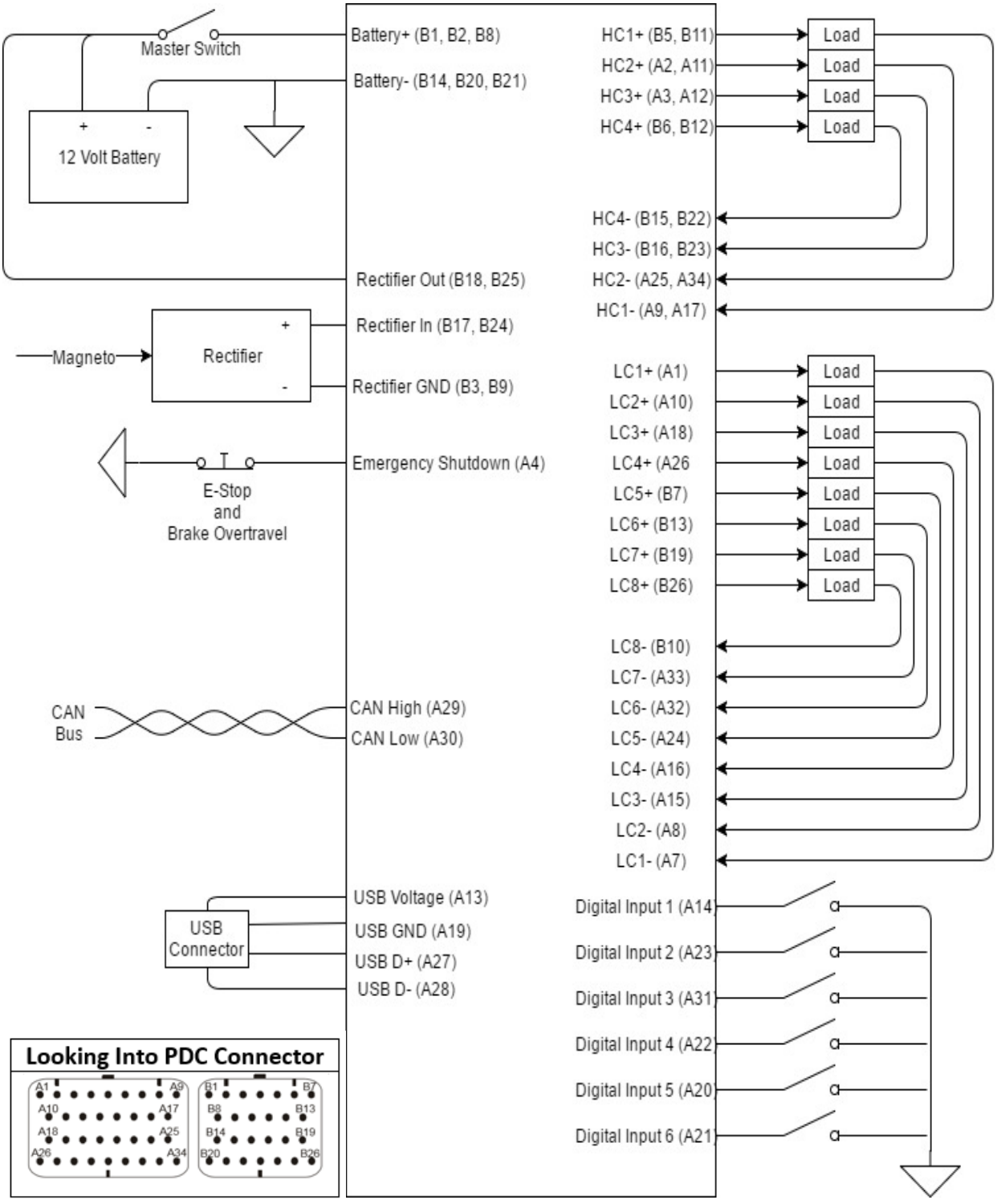

Figure D.1: Wiring Diagram [8]

Output Setup Instructions:

1) Open PDC.h, and find the section “Set Output Settings” near the top of the file.
2) Find the output that you would like to set up. Outputs are designated as “HC” for high current output and “LC” for low current outputs. The number following HC or LC is the output number. Figure D.2 shows the setup parameters for high current output 1. All further instructions refer to output HC1, however, in order to set up another output, replace HC1 with the output name you want to set up.

```
HC1_Setup.OutputEnable = true;
HC1_Setup.CurrentLimit = 12;
HC1_Setup.FusingTime = 0; //in [ms]should be set higher than inrush current time
HC1_Setup.ResetEnable = false;
HC1_Setup.ResetAttempts = 240; //Must be non zero
HC1_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
HC1_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
HC1_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
HC1_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
HC1_Setup.Condition5Channel = &(ConditionReg.defaultCondition);
```

Figure D.2: HC1 setup parameters

3) To enable the output, set HC1_Setup.OutputEnable to true. If this value is false, the output will not turn on.
4) Set the firmware current limit in HC1_Setup.CurrentLimit. This value has 0.1 amp accuracy. For high current outputs, this value should not exceed 15 amps, and for low current outputs this value should not exceed 5 amps.
5) To set a fusing delay, set HC1_Setup.FusingTime to the number of milliseconds after an over current condition the output should shutdown. If the current drops below the current limit at any time between the initial overcurrent detection and the time at which the output is delayed to turn off, the output will remain on until either a second overcurrent condition activates this delay again, or another condition shutdown the output. HC1_Setup.FusingTime should be set slightly higher than the expected inrush current time for the load.
6) If you would like the output to turn the output back on after an overcurrent condition to reattempt to power the load, set HC1_Setup.ResetEnable to true. Then, set HC1_Setup.ResetAttempts to the number of times the output should try to restart before remaining off. If you set HC1_Setup.ResetEnable to false, HC1_Setup.ResetAttempts must be set to a non-zero number. As long as HC1_Setup.ResetEnable is set to false, the output will not attempt to restart after an overcurrent condition.
7) In order to toggle an output based on a digital, analog, or CAN receive value, set the HC1_Setup.ConditionNChannel variables to the reference of the Boolean value you want the output to be toggled on. Predefined references are shown in table D.2.

Table D.2: Output Condition References

Input	Reference
Default Value (true)	&(ConditionReg.defaultCondition)
Digital Input 1	&(InputReg.dInput1)
Digital Input 2	&(InputReg.dInput2)
Digital Input 3	&(InputReg.dInput3)
Digital Input 4	&(InputReg.dInput4)
Digital Input 5	&(InputReg.dInput5)
Digital Input 6	&(InputReg.dInput6)
Engine RPM (CAN)*	&(ConditionReg.EngineRPMthresh)
Engine Temp (CAN)*	&(ConditionReg.EngineTempthresh

*Note that Engine RPM and Engine temperature are configured for a CAN message with CAN ID 0x0E0, and Engine Temp as the first two data bytes, and Engine RPM as the second two data bytes.

To create more conditions, first created a Boolean value in ConditionReg in PDC.h, then in UpdateConditions() under the comment "//CAN condition check," write a conditional statement that assigns a value to the Boolean variable you created. Then, to have an output toggle based on that condition, set HC1_Setup.ConditionNChannel to &(ConditionReg.YourBooleanVariable).

8) Save the file and build the project. Flash the new hex file to the power distribution controller.

Appendix E: Testing Details and Data

This section contains the full datasets for the ADC error test and calibration, as well as the data set for the output current calibrations.

ADC error test and calibration

Table E.1 shows the ADC error pre calibration, and the ADC error post calibration. For this test, analog input 5 was connected to a DC voltage source, and the ADC reading was sent to a serial monitor via UART. The AT90CAN128 microcontroller contains 8 ADC inputs, however, they are multiplexed into a single ADC. Therefore, performing the calibration on analog input 5, actually calibrates all ADC inputs.

In Table E.1, "Input Voltage" is the input into analog input 5, "ADC Return" is the raw 10 bit ADC value, "ADC Voltage Pre-Calibration" is the ADC voltage reading based on the equation E.1, "Error Pre-Calibration" Is the difference between "ADC Voltage Pre-Calibration" and "Input Voltage," "Voltage Post Calibration" is the "Voltage Pre-Calibration" minus the trend line generated in figure E.1, and "Error Post Calibration" is the difference between "Voltage Post Calibration" and "Input Voltage."

$$ADC\ Voltage\ Pre\ Calibration = (\text{ADC Return}) * \frac{ADC\ Reference\ Voltage}{10\ Bit\ ADC} = (ADC\ Return) * \frac{5\ Volts}{2^{10}}$$

Equation E.1: ADC Voltage Calculation

Figure E.1 shows the amount of absolute error in the ADC readings before the calibration, and Figure E.2 shows the amount of absolute error in the ADC readings after the ADC calibration. Notice that in Figure E.2, the error is corrected to be less than on least significant bit of the ADC for all points.

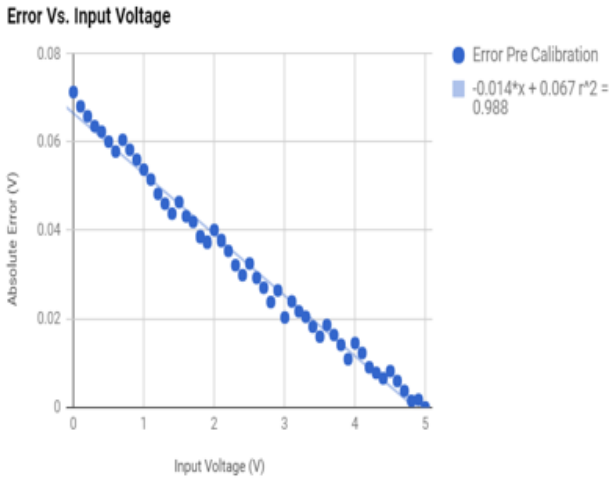

Figure E.1: ADC Error Pre-Calibration

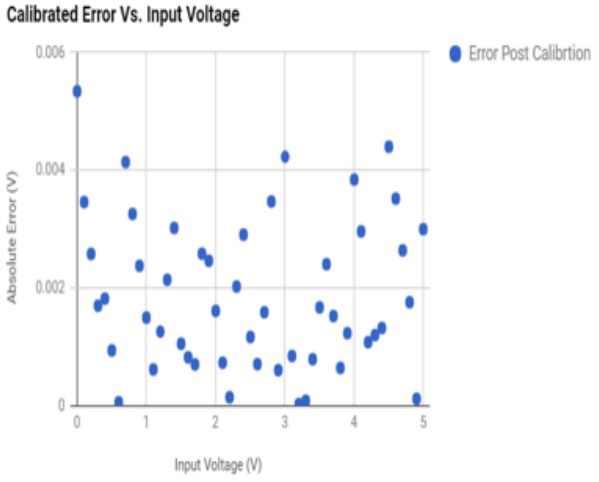

Figure E.2: ADC Error Post-Calibration

Table E.1: ADC Calibration Raw Dataset

Input Voltage (V)	ADC Return	ADC Voltage Pre-Calibration (V)	Error Pre Calibration (V)	Voltage Post Calibration (V)	Error Post Calibration (V)
5	1023	5	0	5.003	0.003
4.904	1003	4.902248289	0.001751710655	4.903879765	0.0001202346041
4.803	983	4.804496579	0.00149657869	4.804759531	0.001759530792
4.703	963	4.706744868	0.003744868035	4.705639296	0.002639296188
4.603	943	4.608993157	0.00599315738	4.606519062	0.003519061584
4.503	923	4.511241447	0.008241446725	4.507398827	0.004398826979
4.402	902	4.408602151	0.006602150538	4.403322581	0.001322580645
4.303	882	4.31085044	0.007850439883	4.304202346	0.001202346041
4.204	862	4.213098729	0.009098729228	4.205082111	0.001082111437
4.103	842	4.115347019	0.01234701857	4.105961877	0.002961876833
4.003	822	4.017595308	0.01459530792	4.006841642	0.003841642229
3.904	801	3.914956012	0.01095601173	3.902765396	0.001234604106
3.803	781	3.817204301	0.01420430108	3.803645161	0.0006451612903
3.703	761	3.71945259	0.01645259042	3.704524927	0.001524926686
3.603	741	3.62170088	0.01870087977	3.605404692	0.002404692082
3.503	720	3.519061584	0.01606158358	3.501328446	0.001671554252
3.403	700	3.421309873	0.01830987292	3.402208211	0.0007917888563
3.303	680	3.323558162	0.02055816227	3.303087977	0.00008797653959
3.204	660	3.225806452	0.02180645161	3.203967742	0.00003225806452
3.104	640	3.128054741	0.02405474096	3.104847507	0.0008475073314
3.005	619	3.025415445	0.02041544477	3.000771261	0.004228739003
2.906	600	2.93255132	0.02655131965	2.906607038	0.0006070381232
2.806	579	2.829912023	0.02391202346	2.802530792	0.003469208211
2.705	559	2.732160313	0.02716031281	2.703410557	0.001589442815
2.605	539	2.634408602	0.02940860215	2.604290323	0.0007096774194
2.504	519	2.536656891	0.0326568915	2.505170088	0.001170087977
2.404	498	2.434017595	0.03001759531	2.401093842	0.002906158358
2.304	478	2.336265885	0.03226588465	2.301973607	0.002026392962
2.203	458	2.238514174	0.035514174	2.202853372	0.000146627566

2.103	438	2.140762463	0.03776246334	2.103733138	0.0007331378299
2.003	418	2.043010753	0.04001075269	2.004612903	0.001612903226
1.903	397	1.940371457	0.0373714565	1.900536657	0.002463343109
1.804	377	1.842619746	0.03861974585	1.801416422	0.002583577713
1.703	357	1.744868035	0.04186803519	1.702296188	0.0007038123167
1.604	337	1.647116325	0.04311632454	1.603175953	0.0008240469208
1.503	317	1.549364614	0.04636461388	1.504055718	0.001055718475
1.403	296	1.446725318	0.04372531769	1.399979472	0.003020527859
1.303	276	1.348973607	0.04597360704	1.300859238	0.002140762463
1.203	256	1.251221896	0.04822189638	1.201739003	0.001260997067
1.102	236	1.153470186	0.05147018573	1.102618768	0.0006187683284
1.002	216	1.055718475	0.05371847507	1.003498534	0.001498533724
0.902	196	0.9579667644	0.05596676442	0.9043782991	0.00237829912
0.802	176	0.8602150538	0.05821505376	0.8052580645	0.003258064516
0.702	156	0.7624633431	0.06046334311	0.7061378299	0.004137829912
0.602	135	0.6598240469	0.05782404692	0.6020615836	0.00006158357771
0.502	115	0.5620723363	0.06007233627	0.502941349	0.0009413489736
0.402	95	0.4643206256	0.06232062561	0.4038211144	0.00182111437
0.303	75	0.366568915	0.06356891496	0.3047008798	0.001700879765
0.203	55	0.2688172043	0.06806549365	0.2055806452	0.002580645161
0.103	35	0.1710654936	0.07131378299	0.1064604106	0.003460410557
0.002	15	0.07331378299	0	0.007340175953	0.005340175953

Output Current Calibration

This test determined the calibration between the current sense amplifier output voltage for the high current and low current outputs, and the corresponding output current. Figure E.3 shows the calibration curve for the low current output while Figure E.4 shows the calibration curve for the high current outputs. The trend line generated in Figures E.3 and E.4 are used in the code to determine the output current. The curves are referenced to raw ADC return values rather than voltage in order to reduce processing time in the code. Table E.2 shows the raw dataset for the low current output calibration, and Table E.3 shows the raw dataset for the high current output calibration. In each table, the “ADC Value” is an average of the ADC values observed during each data point.

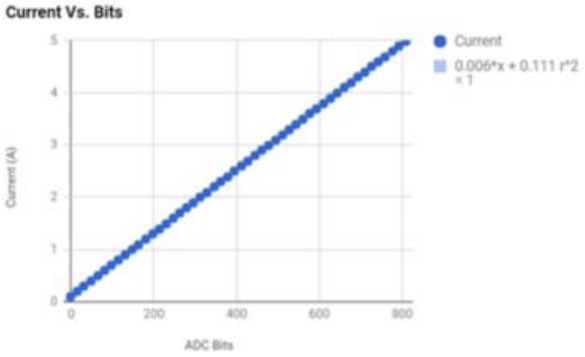

Figure E.3: Low Current Output ADC Calibration

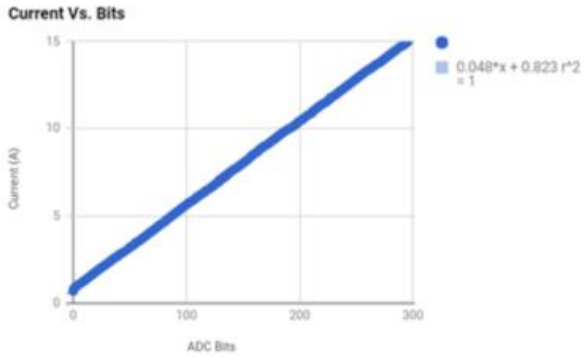

Figure E.4: High Current Output ADC Calibration

Table E.2: Raw Dataset for Low Current Output Current Calibration

ADC Value	Load Current (A)	Output Voltage (V)	Load Power (W)	PDC Dissipated Power (W)
	0.02	12	0.24	0
0	0.1	11.99	1.199	0.001
15	0.2	11.97	2.394	0.006
30.5	0.3	11.96	3.588	0.012
48.5	0.4	11.94	4.776	0.024
65	0.5	11.93	5.965	0.035
81	0.6	11.91	7.146	0.054
97	0.7	11.89	8.323	0.077
114	0.8	11.88	9.504	0.096
131	0.9	11.86	10.674	0.126
147.5	1	11.85	11.85	0.15
163	1.09	11.83	12.8947	0.1853
180.5	1.2	11.82	14.184	0.216
197.5	1.3	11.8	15.34	0.26
213.5	1.39	11.79	16.3881	0.2919
230	1.49	11.77	17.5373	0.3427
247	1.6	11.75	18.8	0.4
262.5	1.7	11.74	19.958	0.442
279	1.8	11.72	21.096	0.504
295	1.89	11.71	22.1319	0.5481

311.5	2	11.69	23.38	0.62
328.5	2.09	11.68	24.4112	0.6688
346	2.2	11.66	25.652	0.748
361.5	2.3	11.64	26.772	0.828
378.5	2.39	11.63	27.7957	0.8843
394.5	2.5	11.61	29.025	0.975
412	2.6	11.6	30.16	1.04
427	2.69	11.58	31.1502	1.1298
444.5	2.8	11.56	32.368	1.232
461	2.9	11.55	33.495	1.305
477.5	2.99	11.53	34.4747	1.4053
494.5	3.09	11.52	35.5968	1.4832
511	3.19	11.5	36.685	1.595
527	3.29	11.48	37.7692	1.7108
543.5	3.39	11.47	38.8833	1.7967
560	3.49	11.45	39.9605	1.9195
576.5	3.6	11.43	41.148	2.052
593	3.69	11.42	42.1398	2.1402
610	3.79	11.4	43.206	2.274
626.5	3.89	11.39	44.3071	2.3729
643	3.99	11.37	45.3663	2.5137
660	4.1	11.35	46.535	2.665
677	4.19	11.34	47.5146	2.7654
693	4.3	11.32	48.676	2.924
710	4.39	11.3	49.607	3.073
726	4.5	11.28	50.76	3.24
742	4.59	11.27	51.7293	3.3507
758.5	4.68	11.25	52.65	3.51
776.5	4.79	11.23	53.7917	3.6883
792	4.89	11.21	54.8169	3.8631
809	4.98	11.19	55.7262	4.0338

Table E.3: Raw Dataset for High Current Output Current Calibration

ADC Value	Current (A)	Output Voltage (V)	Load Power (W)	PDC Output Power (W)
0	0	12	0	0
0	0.11	12	1.32	0
0	0.21	11.99	2.5179	0.0021
0	0.31	11.99	3.7169	0.0031
0	0.4	11.98	4.792	0.008
0	0.51	11.98	6.1098	0.0102
0	0.6	11.97	7.182	0.018
0	0.7	11.96	8.372	0.028
0.5	0.81	11.96	9.6876	0.0324
1.5	0.9	11.95	10.755	0.045
3.5	1	11.95	11.95	0.05
5.5	1.1	11.94	13.134	0.066
8	1.2	11.94	14.328	0.072
10	1.3	11.93	15.509	0.091
12	1.4	11.93	16.702	0.098
14	1.5	11.92	17.88	0.12
16	1.6	11.91	19.056	0.144
18	1.7	11.91	20.247	0.153
20	1.8	11.9	21.42	0.18
22	1.9	11.9	22.61	0.19
24	2	11.89	23.78	0.22
26	2.1	11.89	24.969	0.231
28.5	2.2	11.88	26.136	0.264
30.5	2.3	11.87	27.301	0.299
32.5	2.4	11.87	28.488	0.312
34.5	2.5	11.86	29.65	0.35
36.5	2.6	11.86	30.836	0.364
39	2.69	11.85	31.8765	0.4035
41	2.8	11.85	33.18	0.42
43.5	2.9	11.84	34.336	0.464
46	2.99	11.84	35.4016	0.4784

48	3.09	11.83	36.5547	0.5253
50	3.2	11.82	37.824	0.576
52	3.29	11.82	38.8878	0.5922
54.5	3.4	11.81	40.154	0.646
55.5	3.49	11.81	41.2169	0.6631
58.5	3.59	11.8	42.362	0.718
60.5	3.7	11.8	43.66	0.74
62.5	3.79	11.79	44.6841	0.7959
64.5	3.89	11.79	45.8631	0.8169
66.5	4	11.78	47.12	0.88
68.5	4.09	11.78	48.1802	0.8998
70.5	4.19	11.77	49.3163	0.9637
72.5	4.29	11.76	50.4504	1.0296
74.5	4.39	11.76	51.6264	1.0536
76.5	4.49	11.75	52.7575	1.1225
78.5	4.59	11.74	53.8866	1.1934
81	4.69	11.74	55.0606	1.2194
82.5	4.79	11.73	56.1867	1.2933
84.5	4.89	11.73	57.3597	1.3203
86.5	4.99	11.72	58.4828	1.3972
88.5	5.09	11.72	59.6548	1.4252
90.5	5.19	11.71	60.7749	1.5051
92.5	5.29	11.71	61.9459	1.5341
94.5	5.39	11.7	63.063	1.617
96.5	5.49	11.7	64.233	1.647
98.5	5.59	11.69	65.3471	1.7329
100.5	5.69	11.68	66.4592	1.8208
102.5	5.79	11.68	67.6272	1.8528
105	5.88	11.67	68.6196	1.9404
107.5	5.99	11.67	69.9033	1.9767
109	6.09	11.66	71.0094	2.0706
111.5	6.19	11.66	72.1754	2.1046
113.5	6.29	11.65	73.2785	2.2015

116	6.39	11.64	74.3796	2.3004
117.5	6.48	11.64	75.4272	2.3328
120.5	6.59	11.63	76.6417	2.4383
122.5	6.69	11.63	77.8047	2.4753
124.5	6.79	11.62	78.8998	2.5802
126.5	6.89	11.62	80.0618	2.6182
128.5	6.98	11.61	81.0378	2.7222
130	7.08	11.6	82.128	2.832
132.5	7.19	11.6	83.404	2.876
134.5	7.29	11.59	84.4911	2.9889
136.5	7.38	11.59	85.5342	3.0258
138.5	7.48	11.58	86.6184	3.1416
140.5	7.59	11.57	87.8163	3.2637
143	7.68	11.57	88.8576	3.3024
145.5	7.78	11.56	89.9368	3.4232
147.5	7.88	11.56	91.0928	3.4672
149.5	7.99	11.55	92.2845	3.5955
151.5	8.08	11.54	93.2432	3.7168
153.5	8.18	11.54	94.3972	3.7628
155.5	8.28	11.53	95.4684	3.8916
157	8.39	11.53	96.7367	3.9433
158.5	8.48	11.52	97.6896	4.0704
161	8.58	11.52	98.8416	4.1184
163	8.68	11.51	99.9068	4.2532
164.5	8.78	11.5	100.97	4.39
166.5	8.88	11.5	102.12	4.44
168.5	8.98	11.49	103.1802	4.5798
171.5	9.08	11.49	104.3292	4.6308
173.5	9.18	11.48	105.3864	4.7736
175.5	9.28	11.48	106.5344	4.8256
177.5	9.37	11.47	107.4739	4.9661
179.5	9.48	11.46	108.6408	5.1192
181.5	9.58	11.46	109.7868	5.1732

183.5	9.68	11.45	110.836	5.324
185.5	9.78	11.44	111.8832	5.4768
188	9.88	11.44	113.0272	5.5328
190.5	9.98	11.43	114.0714	5.6886
193.5	10.08	11.43	115.2144	5.7456
195.5	10.18	11.42	116.2556	5.9044
197.5	10.28	11.41	117.2948	6.0652
199	10.38	11.41	118.4358	6.1242
201.5	10.48	11.4	119.472	6.288
203.5	10.58	11.4	120.612	6.348
205.5	10.68	11.39	121.6452	6.5148
207.5	10.78	11.38	122.6764	6.6836
210.5	10.88	11.38	123.8144	6.7456
211.5	10.98	11.37	124.8426	6.9174
213.5	11.08	11.37	125.9796	6.9804
215.5	11.18	11.36	127.0048	7.1552
217.5	11.28	11.36	128.1408	7.2192
219.5	11.38	11.35	129.163	7.397
222.5	11.48	11.34	130.1832	7.5768
224.5	11.58	11.34	131.3172	7.6428
226.5	11.68	11.33	132.3344	7.8256
227.5	11.78	11.33	133.4674	7.8926
230.5	11.88	11.32	134.4816	8.0784
232.5	11.98	11.31	135.4938	8.2662
234.5	12.08	11.31	136.6248	8.3352
236.5	12.18	11.3	137.634	8.526
238.5	12.28	11.3	138.764	8.596
240.5	12.38	11.29	139.7702	8.7898
242.5	12.48	11.28	140.7744	8.9856
244.5	12.58	11.28	141.9024	9.0576
246.5	12.68	11.27	142.9036	9.2564
248.5	12.78	11.27	144.0306	9.3294
250.5	12.88	11.26	145.0288	9.5312

252.5	12.98	11.25	146.025	9.735
254.5	13.08	11.25	147.15	9.81
256.5	13.18	11.24	148.1432	10.0168
258.5	13.28	11.24	149.2672	10.0928
261	13.38	11.23	150.2574	10.3026
263.5	13.48	11.22	151.2456	10.5144
265.5	13.58	11.22	152.3676	10.5924
267.5	13.68	11.21	153.3528	10.8072
270.5	13.78	11.2	154.336	11.024
272.5	13.88	11.2	155.456	11.104
274.5	13.98	11.19	156.4362	11.3238
276.5	14.08	11.19	157.5552	11.4048
278.5	14.18	11.18	158.5324	11.6276
280.5	14.28	11.17	159.5076	11.8524
282.5	14.38	11.17	160.6246	11.9354
284.5	14.48	11.16	161.5968	12.1632
286.5	14.58	11.16	162.7128	12.2472
288.5	14.68	11.15	163.682	12.478
291.5	14.78	11.14	164.6492	12.7108
293.5	14.88	11.14	165.7632	12.7968
295.5	14.98	11.13	166.7274	13.0326

Appendix F: Code

main.cpp

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <AVRLibrary/arduino/Arduino.h>
#include <AVRLibrary/CPFECANLib.h>
#include "PDC.h"

PDC pdc;
testPDC test;
uint8_t OutputCheckCounter;

SIGNAL(TIMER1_COMPA_vect) {//Analog Redundancy Interrupt
	pdc.AnalogRedEnableLOGIC();
}

SIGNAL(TIMER2_COMP_vect){//Fusing Delay Interrupt
	pdc.timer2_int_handler();
}

void canRxIntFunc(CPFECANLib::MSG *msg, uint8_t mobNum) {//CAN Receive Interrupt
	pdc.CAN_RX_Int(msg, mobNum);
}

int main(){
	pdc.InitializePDC(&canRxIntFunc);
	OutputCheckCounter = 0;
	pdc.SetOutputSettings();

	while(1){
		while(OutputCheckCounter < 8){
			pdc.UpdateData(OutputCheckCounter);
			pdc.OutputIO();
			OutputCheckCounter++;
		}

		OutputCheckCounter=0;
		pdc.TxCANdata();
	}
	return 0;
}
```

PDC.h

```
#include <stdio.h>
#include <stdint.h>
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <util/delay.h>

#include <AVRLibrary/CPFECANLib.h>
#include <AVRLibrary/arduino/Arduino.h>

#define MINIMUM_BATTERY_VOLTAGE 11
#define MAX_TEMPERATURE 49 //degrees C

class testPDC;

/*****************************************************************************/
/*ECU CAN Message Setup*/

static constexpr uint8_t ECU_CAN_ID = 0x0E0;

static const CPFECANLib::MSG PROGMEM ECU1_MSG =
{       {       ECU_CAN_ID}, 8, 0, 0, 0};

static const CPFECANLib::MSG PROGMEM ECU1_MASK = { {0xFFF}, 8, 1, 1, 0};

/*****************************************************************************/

class PDC {
        friend class testPDC;
protected:
        enum class OutputNames
                : uint8_t {
                        HC1, HC2, HC3, HC4, LC1, LC2, LC3, LC4, LC5, LC6, LC7, LC8
        };

public:
/*****************************************************************************/
/*                          Set Output Settings                              */
/*****************************************************************************/
/*
 * This Function sets the output settings
 */

        void SetOutputSettings(void){
                //Output Settings

                GlobalControl.GlobalEnable = true;
                ConditionReg.defaultCondition =true;

                HC1_Setup.OutName = OutputNames::HC1;
                HC1_Setup.OutputEnable = false;
                HC1_Setup.CurrentLimit = 3;
                HC1_Setup.FusingTime = 5000; //in [ms]should be set higher than inrush current time
                HC1_Setup.ResetEnable = false;
                HC1_Setup.ResetAttempts = 240; //Must be non zero
```

```
HC1_Setup.Condition1Channel = &(InputReg.dInput1);
HC1_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
HC1_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
HC1_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
HC1_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

HC2_Setup.OutName = OutputNames::HC2;
HC2_Setup.OutputEnable = false;
HC2_Setup.CurrentLimit = 4; //in Amps
HC2_Setup.FusingTime = 0;
HC2_Setup.ResetEnable = false;
HC2_Setup.ResetAttempts = 240; //Must be non zero
HC2_Setup.Condition1Channel = &(InputReg.dInput2);
HC2_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
HC2_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
HC2_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
HC2_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

HC3_Setup.OutName = OutputNames::HC3;
HC3_Setup.OutputEnable = false;
HC3_Setup.CurrentLimit = 5; //in Amps
HC3_Setup.FusingTime = 0;
HC3_Setup.ResetEnable = false;
HC3_Setup.ResetAttempts = 240; //Must be non zero
HC3_Setup.Condition1Channel = &(InputReg.dInput1);
HC3_Setup.Condition2Channel = &(InputReg.dInput2);
HC3_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
HC3_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
HC3_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

HC4_Setup.OutName = OutputNames::HC4;
HC4_Setup.OutputEnable = false;
HC4_Setup.CurrentLimit = 3; //in Amps
HC4_Setup.FusingTime = 10;
HC4_Setup.ResetEnable = false;
HC4_Setup.ResetAttempts = 240; //Must be non zero
HC4_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
HC4_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
HC4_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
HC4_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
HC4_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

LC1_Setup.OutName = OutputNames::LC1;
LC1_Setup.OutputEnable = true;
LC1_Setup.CurrentLimit = 5; //in Amps
LC1_Setup.FusingTime = 10;
LC1_Setup.ResetEnable = false;
LC1_Setup.ResetAttempts = 10; //Must be non zero
LC1_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
LC1_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

LC2_Setup.OutName = OutputNames::LC2;
LC2_Setup.OutputEnable = false;
```

```
LC2_Setup.CurrentLimit = 3; //in Amps
LC2_Setup.FusingTime = 10;
LC2_Setup.ResetEnable = false;
LC2_Setup.ResetAttempts = 240; //Must be non zero
LC2_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
LC2_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
LC2_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
LC2_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
LC2_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

LC3_Setup.OutName = OutputNames::LC3;
LC3_Setup.OutputEnable = false;
LC3_Setup.CurrentLimit = 3; //in Amps
LC3_Setup.FusingTime = 10;
LC3_Setup.ResetEnable = false;
LC3_Setup.ResetAttempts = 240; //Must be non zero
LC3_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
LC3_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
LC3_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
LC3_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
LC3_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

LC4_Setup.OutName = OutputNames::LC4;
LC4_Setup.OutputEnable = false;
LC4_Setup.CurrentLimit = 3; //in Amps
LC4_Setup.FusingTime = 10;
LC4_Setup.ResetEnable = false;
LC4_Setup.ResetAttempts = 240; //Must be non zero
LC4_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
LC4_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
LC4_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
LC4_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
LC4_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

LC5_Setup.OutName = OutputNames::LC5;
LC5_Setup.OutputEnable = false;
LC5_Setup.CurrentLimit = 3; //in Amps
LC5_Setup.FusingTime = 10;
LC5_Setup.ResetEnable = false;
LC5_Setup.ResetAttempts = 240; //Must be non zero
LC5_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
LC5_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
LC5_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
LC5_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
LC5_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

LC6_Setup.OutName = OutputNames::LC6;
LC6_Setup.OutputEnable = false;
LC6_Setup.CurrentLimit = 3; //in Amps
LC6_Setup.FusingTime = 10;
LC6_Setup.ResetEnable = false;
LC6_Setup.ResetAttempts = 240; //Must be non zero
LC6_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
LC6_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
LC6_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
LC6_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
```

```
                LC6_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

                LC7_Setup.OutName = OutputNames::LC7;
                LC7_Setup.OutputEnable = false;
                LC7_Setup.CurrentLimit = 3; //in Amps
                LC7_Setup.FusingTime = 10;
                LC7_Setup.ResetEnable = false;
                LC7_Setup.ResetAttempts = 240; //Must be non zero
                LC7_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
                LC7_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
                LC7_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
                LC7_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
                LC7_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

                LC8_Setup.OutName = OutputNames::LC8;
                LC8_Setup.OutputEnable = false;
                LC8_Setup.CurrentLimit = 3; //in Amps
                LC8_Setup.FusingTime = 10;
                LC8_Setup.ResetEnable = false;
                LC8_Setup.ResetAttempts = 240; //Must be non zero
                LC8_Setup.Condition1Channel = &(ConditionReg.defaultCondition);
                LC8_Setup.Condition2Channel = &(ConditionReg.defaultCondition);
                LC8_Setup.Condition3Channel = &(ConditionReg.defaultCondition);
                LC8_Setup.Condition4Channel = &(ConditionReg.defaultCondition);
                LC8_Setup.Condition5Channel = &(ConditionReg.defaultCondition);

/*******************************************************************************************/
                //Set Control Register Initialization
                //DO NOT EDIT THE REMAINDER OF THIS FUNCTION

                HC1Control.FuseEnabled = false;
                HC1Control.InrushFlag  = false;
                HC1Control.OutCurrent  = 0;
                HC1Control.OutVoltage  = 0;
                HC1Control.ResetAttemptsRemaining = HC1_Setup.ResetAttempts;
                HC1Control.ResetTimeEnable = true;
                HC1Control.FuseBuffer = false;
                HC1Control.timecount = 0;

                HC2Control.FuseEnabled = false;
                HC2Control.InrushFlag  = false;
                HC2Control.OutCurrent  = 0;
                HC2Control.OutVoltage  = 0;
                HC2Control.ResetAttemptsRemaining = HC2_Setup.ResetAttempts;
                HC2Control.ResetTimeEnable = true;
                HC2Control.FuseBuffer = false;
                HC2Control.timecount = 0;

                HC3Control.FuseEnabled = false;
                HC3Control.InrushFlag  = false;
                HC3Control.OutCurrent  = 0;
                HC3Control.OutVoltage  = 0;
                HC3Control.ResetAttemptsRemaining = HC3_Setup.ResetAttempts;
                HC3Control.ResetTimeEnable = true;
                HC3Control.FuseBuffer = false;
                HC3Control.timecount = 0;
```

```
HC4Control.FuseEnabled = false;
HC4Control.InrushFlag  = false;
HC4Control.OutCurrent  = 0;
HC4Control.OutVoltage  = 0;
HC4Control.ResetAttemptsRemaining = HC4_Setup.ResetAttempts;
HC4Control.ResetTimeEnable = true;
HC4Control.FuseBuffer = false;
HC4Control.timecount = 0;

LC1Control.FuseEnabled = true;
LC1Control.InrushFlag  = false;
LC1Control.OutCurrent  = 0;
LC1Control.OutVoltage  = 0;
LC1Control.ResetAttemptsRemaining = LC1_Setup.ResetAttempts;
LC1Control.ResetTimeEnable = true;
LC1Control.FuseBuffer = false;
LC1Control.timecount = 0;

LC2Control.FuseEnabled = false;
LC2Control.InrushFlag  = false;
LC2Control.OutCurrent  = 0;
LC2Control.OutVoltage  = 0;
LC2Control.ResetAttemptsRemaining = LC2_Setup.ResetAttempts;
LC2Control.ResetTimeEnable = true;
LC2Control.FuseBuffer = false;
LC2Control.timecount = 0;

LC3Control.FuseEnabled = false;
LC3Control.InrushFlag  = false;
LC3Control.OutCurrent  = 0;
LC3Control.OutVoltage  = 0;
LC3Control.ResetAttemptsRemaining = LC3_Setup.ResetAttempts;
LC3Control.ResetTimeEnable = true;
LC3Control.FuseBuffer = false;
LC3Control.timecount = 0;

LC4Control.FuseEnabled = false;
LC4Control.InrushFlag  = false;
LC4Control.OutCurrent  = 0;
LC4Control.OutVoltage  = 0;
LC4Control.ResetAttemptsRemaining = LC4_Setup.ResetAttempts;
LC4Control.ResetTimeEnable = true;
LC4Control.FuseBuffer = false;
LC4Control.timecount = 0;

LC5Control.FuseEnabled = false;
LC5Control.InrushFlag  = false;
LC5Control.OutCurrent  = 0;
LC5Control.OutVoltage  = 0;
LC5Control.ResetAttemptsRemaining = LC5_Setup.ResetAttempts;
LC5Control.ResetTimeEnable = true;
LC5Control.FuseBuffer = false;
LC5Control.timecount = 0;

LC6Control.FuseEnabled = false;
```

```
                LC6Control.InrushFlag  = false;
                LC6Control.OutCurrent  = 0;
                LC6Control.OutVoltage  = 0;
                LC6Control.ResetAttemptsRemaining = LC6_Setup.ResetAttempts;
                LC6Control.ResetTimeEnable = true;
                LC6Control.FuseBuffer = false;
                LC6Control.timecount = 0;

                LC7Control.FuseEnabled = false;
                LC7Control.InrushFlag  = false;
                LC7Control.OutCurrent  = 0;
                LC7Control.OutVoltage  = 0;
                LC7Control.ResetAttemptsRemaining = LC7_Setup.ResetAttempts;
                LC7Control.ResetTimeEnable = true;
                LC7Control.FuseBuffer = false;
                LC7Control.timecount = 0;

                LC8Control.FuseEnabled = false;
                LC8Control.InrushFlag  = false;
                LC8Control.OutCurrent  = 0;
                LC8Control.OutVoltage  = 0;
                LC8Control.ResetAttemptsRemaining = LC8_Setup.ResetAttempts;
                LC8Control.ResetTimeEnable = true;
                LC8Control.FuseBuffer = false;
                LC8Control.timecount = 0;

        }

/*****************************************************************************/
/*                              CAN                                          */
/*****************************************************************************/

        //MOB Numbers
        static constexpr uint16_t CAN0id = 0x0A0;
        static constexpr uint16_t CAN1id = 0x0A1;
        static constexpr uint16_t CAN2id = 0x0A2;
        static constexpr uint16_t CAN3id = 0x0A3;
        static constexpr uint16_t CAN4id = 0x0A4;
        static constexpr uint16_t CAN5id = 0x0A5;
        static constexpr uint16_t CAN6id = 0x0A6;
        static constexpr uint16_t CAN7id = 0x0A7;

        static constexpr uint8_t sendcanMOB0 = 0;
        static constexpr uint8_t sendcanMOB1 = 1;
        static constexpr uint8_t sendcanMOB2 = 2;
        static constexpr uint8_t sendcanMOB3 = 3;
        static constexpr uint8_t sendcanMOB4 = 4;
        static constexpr uint8_t sendcanMOB5 = 5;
        static constexpr uint8_t sendcanMOB6 = 6;
        static constexpr uint8_t sendcanMOB7 = 7;
        static constexpr uint8_t RX_ECU1MOB = 8;

        void TxCANdata(void) {
                CANMessageData messageData = {0, 0, 0, 0};

                messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend0.HCCurrent);
```

```
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend0.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend0.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend0.LCVoltage);
		txCAN(CAN0id, &messageData, sendcanMOB0); //Send Message For Check 0

		messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend1.HCCurrent);
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend1.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend1.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend1.LCVoltage);
		txCAN(CAN1id, &messageData, sendcanMOB1);//Send Message for Check 1

		messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend2.HCCurrent);
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend2.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend2.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend2.LCVoltage);
		txCAN(CAN2id, &messageData, sendcanMOB2);//Send Message for Check 2

		messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend3.HCCurrent);
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend3.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend3.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend3.LCVoltage);
		txCAN(CAN3id, &messageData, sendcanMOB3);//Send Message for Check 3

		messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend4.HCCurrent);
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend4.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend4.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend4.LCVoltage);
		txCAN(CAN4id, &messageData, sendcanMOB4);//Send Message for Check 4

		messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend5.HCCurrent);
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend5.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend5.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend5.LCVoltage);
		txCAN(CAN5id, &messageData, sendcanMOB5);//Send Message for Check 5

		messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend6.HCCurrent);
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend6.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend6.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend6.LCVoltage);
		txCAN(CAN6id, &messageData, sendcanMOB6);//Send Message for Check 6

		messageData.chan1 = CPFECANLib::hton_uint16_t(CANsend7.HCCurrent);
		messageData.chan2 = CPFECANLib::hton_uint16_t(CANsend7.HCVoltage);
		messageData.chan3 = CPFECANLib::hton_uint16_t(CANsend7.LCCurrent);
		messageData.chan4 = CPFECANLib::hton_uint16_t(CANsend7.LCVoltage);
		txCAN(CAN7id, &messageData, sendcanMOB7); //Send Message for Check 7
}

void CAN_RX_Int(CPFECANLib::MSG *msg, uint8_t mobNum) {

		switch (msg->identifier.standard) {
		case ECU_CAN_ID:
			memcpy((void *) &ECU1data, msg->data, sizeof(ECU1data));
			RX_ECU1(true);
			break;
		}
```

```
        }

/*****************************************************************************/
/*                          InitializePDC                                    */
/*****************************************************************************/
/*
 * This function Initializes inputs and outputs, Serial Interface, SPI, ADCs,
 * timers and interrupts
 */
        void InitializePDC(CPFECANLib::CAN_MSG_RX canRxIntFunc){
                CPFECANLib::init(CPFECANLib::CAN_BAUDRATE::B1M, canRxIntFunc);
                initCAN_RX();
                PinInit();
                Serial.begin(115200);
                Initialize_SPI_Master();
                initADC();
                TimersInit();
                RedInit();
                sei();
        }
/*****************************************************************************/
/*                          UpdateData                                       */
/*****************************************************************************/
/*
 * This Function updates data for selected outputs and all inputs
 */

        void UpdateData(uint8_t CheckNumber){
                volatile OutputControl *pHCControlReg;
                volatile OutputControl *pLCControlReg;
                CANoutputData *pSendCANdata;

                SwitchMUX(CheckNumber);
                OutputRegisterTracker(CheckNumber);

                pSendCANdata = OutReg.pCANoutputData;

                //Check Outputs
                pHCControlReg = OutReg.pHCControlReg;//Set HC Control Reg to current HC Reg
                pLCControlReg = OutReg.pLCControlReg;//Set LC Control Reg to current LC Reg

                pHCControlReg->OutCurrent = ReadHC_Current();

                //Read 16 bit ADC value for output voltage CAN tx
                pSendCANdata->HCVoltage = highPrecisionRead(3);

                //Read/calculate HC output voltage
                pHCControlReg->OutVoltage = (pSendCANdata->HCVoltage) * 3.61 * 0.00488;
                pLCControlReg->OutCurrent = ReadLC_Current();

                //Read 16 bit ADC value for output voltage CAN tx
                pSendCANdata->LCVoltage = highPrecisionRead(1);

                //Read/calculate LC output voltage
                pLCControlReg->OutVoltage = pSendCANdata->LCVoltage * 3.61 * 0.00488;
```

```
        //Check Battery Voltage and PDC Temperature

                //Read Input Battery Voltage
                 GlobalControl.BatteryVoltage = highPrecisionRead(7) * 3.61 * 0.00488;

                //Read board temperature
                 GlobalControl.Temperature = ((highPrecisionRead(4) * 0.00488)-0.4)/0.0195;

                 //Check Analog Inputs
                 InputReg.aInput5 = highPrecisionRead(5) * 0.00488; //0.00488 = volts per bit for adc
                 InputReg.aInput6 = highPrecisionRead(6) * 0.00488;

                 //Update Received CAN messages
                 CAN_RX.EngineRPM = (motecToFloat(ECU1data[1]))*100;// Engine RPM Value

                 UpdateConditions();
        }

/*****************************************************************************/
/*                          OutputIO                                         */
/*****************************************************************************/
/*
 * This Function Decides if an output should be on or off
 */
        void OutputIO(void){

                 volatile OutputControl *pHCControlReg;
                 OutputSetup *pHCSetupReg;
                 volatile OutputControl *pLCControlReg;
                 OutputSetup *pLCSetupReg;

                 pHCControlReg = OutReg.pHCControlReg;
                 pHCSetupReg   = OutReg.pHCSetupReg;
                 pLCControlReg = OutReg.pLCControlReg;
                 pLCSetupReg   = OutReg.pLCSetupReg;

        if(~(pHCControlReg->FuseEnabled)&&(pHCSetupReg->OutputEnable)&&
           *(pHCSetupReg->Condition1Channel)&&*(pHCSetupReg->Condition2Channel)&&
           *(pHCSetupReg->Condition3Channel)&&*(pHCSetupReg->Condition4Channel)&&
           *(pHCSetupReg->Condition5Channel)&&(pHCControlReg->ResetTimeEnable)&&
            (pHCControlReg->ResetAttemptsRemaining>0)&&GlobalControl.GlobalEnable){

                                 OutputON(pHCSetupReg->OutName);
                        }
                        else{
                                 OutputOFF(pHCSetupReg->OutName);
                        }

        if(~(pLCControlReg->FuseEnabled)&&(pLCSetupReg->OutputEnable)&&
           *(pLCSetupReg->Condition1Channel)&&*(pLCSetupReg->Condition2Channel)&&
           *(pLCSetupReg->Condition3Channel)&&*(pLCSetupReg->Condition4Channel)&&
           *(pLCSetupReg->Condition5Channel)&&(pLCControlReg->ResetTimeEnable)&&
            (pLCControlReg->ResetAttemptsRemaining>0)&&GlobalControl.GlobalEnable){

                        OutputON(pLCSetupReg->OutName);
                        }
```

```
                        else{
                                OutputOFF(pLCSetupReg->OutName);
                        }
                }
/*************************************************************************/
/*                Analog Re-Enable Determination                         */
/*************************************************************************/
/*
 * This Function is called based on timer 1 interrupt when activated. This
 * re-enables HC analog redundancy 10ms after an output is turned on.
 */
        void AnalogRedEnableLOGIC(void){
                cli();//Disable interrupts

        if((RedundancyStatusBuffer.RedundancyHC1)&& (RedundancyStatus.RedundancyHC1)&&
          (RedundancyStatusBuffer.RedundancyHC2)&& ~(RedundancyStatus.RedundancyHC2)&&
          (RedundancyStatusBuffer.RedundancyHC3)&& ~(RedundancyStatus.RedundancyHC3)&&
          (RedundancyStatusBuffer.RedundancyHC4)&& ~(RedundancyStatus.RedundancyHC4)){
                        TIMSK1 &= ~(1<<OCIE1A);//disable compare interrupt Timer 1
                }

        if((RedundancyStatusBuffer.RedundancyHC1)&& ~(RedundancyStatus.RedundancyHC1)){
                        EnableRedundancy(0);//Enable HC1 Redundancy
                }
        if((RedundancyStatusBuffer.RedundancyHC2)&& ~(RedundancyStatus.RedundancyHC2)){
                        EnableRedundancy(1);//Enable HC2 Redundancy
                }
        if((RedundancyStatusBuffer.RedundancyHC3)&& ~(RedundancyStatus.RedundancyHC3)){
                        EnableRedundancy(2);//Enable HC3 Redundancy
                }
        if((RedundancyStatusBuffer.RedundancyHC4)&& ~(RedundancyStatus.RedundancyHC4)){
                        EnableRedundancy(3);//Enable HC4 Redundancy
                }

                RedundancyStatusBuffer.RedundancyHC1 = true;
                RedundancyStatusBuffer.RedundancyHC2 = true;
                RedundancyStatusBuffer.RedundancyHC3 = true;
                RedundancyStatusBuffer.RedundancyHC4 = true;

                LC1Control.InrushFlag = false;
                LC2Control.InrushFlag = false;
                LC3Control.InrushFlag = false;
                LC4Control.InrushFlag = false;
                LC5Control.InrushFlag = false;
                LC6Control.InrushFlag = false;
                LC7Control.InrushFlag = false;
                LC8Control.InrushFlag = false;
                HC1Control.InrushFlag = false;
                HC2Control.InrushFlag = false;
                HC3Control.InrushFlag = false;
                HC4Control.InrushFlag = false;

                sei(); //Enable Interrupts
        }
```

```
/*****************************************************************************************/
/*                              Delay Fusing Time                                        */
/*****************************************************************************************/
                void timer2_int_handler(void){
                        if(HC1Control.FuseBuffer){
                                (HC1Control.timecount)++;
                        }
                        if(HC2Control.FuseBuffer){
                                (HC2Control.timecount)++;
                        }
                        if(HC3Control.FuseBuffer){
                                (HC3Control.timecount)++;
                        }
                        if(HC4Control.FuseBuffer){
                                (HC4Control.timecount)++;
                        }
                        if(LC1Control.FuseBuffer){
                                (LC1Control.timecount)++;
                        }
                        if(LC2Control.FuseBuffer){
                                (LC2Control.timecount)++;
                        }
                        if(LC3Control.FuseBuffer){
                                (LC3Control.timecount)++;
                        }
                        if(LC4Control.FuseBuffer){
                                (LC4Control.timecount)++;
                        }
                        if(LC5Control.FuseBuffer){
                                (LC5Control.timecount)++;
                        }
                        if(LC6Control.FuseBuffer){
                                (LC6Control.timecount)++;
                        }
                        if(LC7Control.FuseBuffer){
                                (LC7Control.timecount)++;
                        }
                        if(LC8Control.FuseBuffer){
                                (LC8Control.timecount)++;
                        }
                }

protected:
/*****************************************************************************/
/*                              CAN                                          */
/*****************************************************************************/
        volatile uint16_t ECU1data[4];//Received data for ECU1data

        void RX_ECU1(bool interruptMode) {
                CPFECANLib::enableMOBAsRX_PROGMEM(RX_ECU1MOB, &ECU1_MSG,
                                                        &ECU1_MASK, interruptMode);
        }

        void initCAN_RX() {
                RX_ECU1(false);
        }
```

```
	typedef struct {
		uint16_t chan1;
		uint16_t chan2;
		uint16_t chan3;
		uint16_t chan4;
	} CANMessageData;

	static void txCAN(uint16_t ID, CANMessageData *data, uint8_t MOB) {
		CPFECANLib::MSG msg; //comes from CPECANLib.h
		msg.identifier.standard = ID; //set for standard.  for extended use identifier.extended
		msg.data = (uint8_t *)data;
		msg.dlc = 8; //Number of bytes of data
		msg.ide = 0; //Set to 0 for standard identifier.  Set to 1 for extended address
		msg.rtr = 0;
		CPFECANLib::sendMsgUsingMOB(MOB, &msg);
		}

	//add all variables received from CAN in this struct
	typedef struct CANrx{
		float EngineRPM;
		float EngineTemp;
	}CANrx;

	CANrx CAN_RX;

	uint16_t swap(uint16_t d) {
		uint16_t a;
		unsigned char *dst = (unsigned char *) &a;
		unsigned char *src = (unsigned char *) &d;
		dst[0] = src[1];
		dst[1] = src[0];
		return a;
	}

	float motecToFloat(uint16_t value, float scaler=100.0) {
		return (float)swap(value) / scaler;
	}
/*************************************************************************/
/*		Initialization Functions				*/
/*************************************************************************/
	void PinInit(void){
		//Setup Low Current Outputs
		DDRA |= (1<<PA0)|(1<<PA1)|(1<<PA2)|(1<<PA3)|(1<<PA4);
		DDRE |= (1<<PE3)|(1<<PE4)|(1<<PE5);

		//Setup High Current Outputs
		DDRB |= (1<<PB4)|(1<<PB5)|(1<<PB6)|(1<<PB7);

		//Setup Inputs with Pullups
		PORTD |= (1<<PD0)|(1<<PD1)|(1<<PD2)|(1<<PD3);

		//Setup SPI Pins as Outputs
		DDRB |= (1<<PB0)|(1<<PB1)|(1<<PB2); //slave select, SCLK, MOSI
```

```
                //MUX Select Pins Set as Outputs
                DDRC |= (1<<PC0)|(1<<PC1)|(1<<PC2);
        }

        void Initialize_SPI_Master(void)
        {
           SPCR = (0<<SPIE) |          //No interrupts
           (1<<SPE) |                  //SPI enabled
           (0<<DORD) |                 //send MSB first
           (1<<MSTR) |                 //master
           (1<<CPOL) |                 //clock idles low
           (1<<CPHA) |                 //sample leading edge
           (0<<SPR1) | (0<<SPR0) ;     //clock speed
           SPSR = (0<<SPIF) |          //SPI interrupt flag
           (0<<WCOL) |                       //Write collision flag
           (0<<SPI2X) ;                //Doubles SPI clock
           PORTB = 1 << PB0;           // make sure SS is high
        }

        void initADC(){
                        ADCSRA = 0x87;//Turn On ADC and set prescaler (CLK/128)
                        ADCSRB = 0x00;//turn off autotrigger
                        ADMUX = 0x00; //Set ADC channel ADC0
        }

        void TimersInit(void){
                //Timer 1 used for Analog redundancy re-enable
                TCCR1B |= (1 << WGM12)|(1 << CS11); //set timer 1 to CTC mode and prescaler 8
                TCNT1 = 0; //initialize counter
                OCR1A = 20000;//This sets interrupt to 10ms
                //Timer1 Interrupt is enabled and disabled In analog redundancy functions as needed

                //Timer 2 used to delay the fuse after overcurrent condition if needed.
                TCCR2A |= (1<<WGM21) | (1<<CS22); //CTC, 64 prescaler
                OCR2A = 250;//interrupt every 1ms
                TIMSK2 = (1 << OCIE2A); //Timer 2 CTC interrupt enable
        }

/*****************************************************************************/
/*****************************************************************************/
/****************SwitchMUX(uint8_t)*********************************/
/*
 * This function changes the MUX outputs based on a uint8_t input called CheckNumber
 *
 * CheckNumberValue                        Outputs Sent to ADCs
 *          0                                  LC1, HC4
 *          1                                  LC2, HC1
 *          2                                  LC3, HC3
 *          3                                  LC4, HC2
 *          4                                  LC5, HC4
 *          5                                  LC6, HC1
 *          6                                  LC7, HC3
 *          7                                  LC8, HC2
*/
```

```
	void SwitchMUX(uint8_t CheckNumber){
		switch(CheckNumber){
		case 0:
			PORTC &= ~((1<<PC2) | (1<<PC1) | (1<<PC0));//Clear PC2, PC1, PC0
			break;
		case 1:
			PORTC &= ~((1<<PC2)|(1<<PC1));
			PORTC |= (1<<PC0);
			break;
		case 2:
			PORTC &= ~((1<<PC2)|(1<<PC0));
			PORTC |= (1<<PC1);
			break;
		case 3:
			PORTC &= ~(1<<PC2);
			PORTC |= (1<<PC1)|(1<<PC0);
			break;
		case 4:
			PORTC &= ~((1<<PC1)|(1<<PC0));
			PORTC |= (1<<PC2);
			break;
		case 5:
			PORTC &= ~(1<<PC1);
			PORTC |= (1<<PC2)|(1<<PC0);
			break;
		case 6:
			PORTC &= ~(1<<PC0);
			PORTC |= (1<<PC2)|(1<<PC1);
			break;
		case 7:
			PORTC |= (1<<PC2)|(1<<PC1)|(1<<PC0);
			break;
		}
	}
/*************************************************************************/
/*		Output Control Registers				*/
/*************************************************************************/
	typedef struct OutputControl{
		bool FuseEnabled;//Sets the output as enabled
		uint8_t ResetAttemptsRemaining;//Counter for restart attempts
		bool ResetTimeEnable;//Prevents restart when time condition not met
		bool FuseBuffer; //set to true when overcurrent condition is met, and wait x ms before
				shutdown
		float OutVoltage;
		float OutCurrent;
		bool InrushFlag;
		uint16_t timecount; //counts time for how long over current condition is present.
					compared to fusing time in setup register
	}OutputControl;

	volatile OutputControl HC1Control;
	volatile OutputControl HC2Control;
	volatile OutputControl HC3Control;
	volatile OutputControl HC4Control;
	volatile OutputControl LC1Control;
	volatile OutputControl LC2Control;
```

```
        volatile OutputControl LC3Control;
        volatile OutputControl LC4Control;
        volatile OutputControl LC5Control;
        volatile OutputControl LC6Control;
        volatile OutputControl LC7Control;
        volatile OutputControl LC8Control;

        typedef struct Globalcontrol{
                float BatteryVoltage;
                float Temperature;
                bool GlobalEnable;
        }Globalcontrol;

        Globalcontrol GlobalControl;

/*****************************************************************************/
/*                  Setup Outputs                                            */
/*****************************************************************************/
        typedef struct OutputSetup{
                        OutputNames OutName;
                        bool OutputEnable; //enable the output
                        float CurrentLimit; //Current Limit for the output
                        uint16_t FusingTime; //Time of over current condition before output disable in
                                             ms
                        bool ResetEnable; //if true, output will try to restart
                        uint8_t ResetAttempts;//Number of restart attempts allowed
                        bool *Condition1Channel;//Pointer to a conditional channel
                        bool *Condition2Channel;//Pointer to a conditional channel
                        bool *Condition3Channel;//Pointer to a conditional channel
                        bool *Condition4Channel;//Pointer to a conditional channel
                        bool *Condition5Channel;//Pointer to a conditional channel
                }OutputSetup;

                OutputSetup HC1_Setup;
                OutputSetup HC2_Setup;
                OutputSetup HC3_Setup;
                OutputSetup HC4_Setup;
                OutputSetup LC1_Setup;
                OutputSetup LC2_Setup;
                OutputSetup LC3_Setup;
                OutputSetup LC4_Setup;
                OutputSetup LC5_Setup;
                OutputSetup LC6_Setup;
                OutputSetup LC7_Setup;
                OutputSetup LC8_Setup;

        typedef struct CANoutputData{
                uint16_t LCCurrent;
                uint16_t LCVoltage;
                uint16_t HCCurrent;
                uint16_t HCVoltage;
        }CANoutputData;

                CANoutputData CANsend0;
                CANoutputData CANsend1;
                CANoutputData CANsend2;
```

```
CANoutputData CANsend3;
CANoutputData CANsend4;
CANoutputData CANsend5;
CANoutputData CANsend6;
CANoutputData CANsend7;

/*****************************************************************************************/
/*                    OutputRegisterTracker                                              */
/*****************************************************************************************/
/*
 * This function points to the Control Register and the Setup Register given the Check
 * Number to be performed.
 */

typedef struct OutputTracker{
        volatile OutputControl *pHCControlReg;
        OutputSetup *pHCSetupReg;
        volatile OutputControl *pLCControlReg;
        OutputSetup *pLCSetupReg;
        CANoutputData *pCANoutputData;
}OutputTracker;

OutputTracker OutReg;

void OutputRegisterTracker(uint8_t CheckNumber){
        switch(CheckNumber){
        case 0:
                OutReg.pHCControlReg = &HC4Control;
                OutReg.pHCSetupReg = &HC4_Setup;
                OutReg.pLCControlReg = &LC1Control;
                OutReg.pLCSetupReg = &LC1_Setup;
                OutReg.pCANoutputData = &CANsend0;
                break;
        case 1:
                OutReg.pHCControlReg = &HC1Control;
                OutReg.pHCSetupReg = &HC1_Setup;
                OutReg.pLCControlReg = &LC2Control;
                OutReg.pLCSetupReg = &LC2_Setup;
                OutReg.pCANoutputData = &CANsend1;
                break;
        case 2:
                OutReg.pHCControlReg = &HC3Control;
                OutReg.pHCSetupReg = &HC3_Setup;
                OutReg.pLCControlReg = &LC3Control;
                OutReg.pLCSetupReg = &LC3_Setup;
                OutReg.pCANoutputData = &CANsend2;
                break;
        case 3:
                OutReg.pHCControlReg = &HC2Control;
                OutReg.pHCSetupReg = &HC2_Setup;
                OutReg.pLCControlReg = &LC4Control;
                OutReg.pLCSetupReg = &LC4_Setup;
                OutReg.pCANoutputData = &CANsend3;
                break;
        case 4:
                OutReg.pHCControlReg = &HC4Control;
```

```
                                        OutReg.pHCSetupReg = &HC4_Setup;
                                        OutReg.pLCControlReg = &LC5Control;
                                        OutReg.pLCSetupReg = &LC5_Setup;
                                        OutReg.pCANoutputData = &CANsend4;
                                        break;
                                case 5:
                                        OutReg.pHCControlReg = &HC1Control;
                                        OutReg.pHCSetupReg = &HC1_Setup;
                                        OutReg.pLCControlReg = &LC6Control;
                                        OutReg.pLCSetupReg = &LC6_Setup;
                                        OutReg.pCANoutputData = &CANsend5;
                                        break;
                                case 6:
                                        OutReg.pHCControlReg = &HC3Control;
                                        OutReg.pHCSetupReg = &HC3_Setup;
                                        OutReg.pLCControlReg = &LC7Control;
                                        OutReg.pLCSetupReg = &LC7_Setup;
                                        OutReg.pCANoutputData = &CANsend6;
                                        break;
                                case 7:
                                        OutReg.pHCControlReg = &HC2Control;
                                        OutReg.pHCSetupReg = &HC2_Setup;
                                        OutReg.pLCControlReg = &LC8Control;
                                        OutReg.pLCSetupReg = &LC8_Setup;
                                        OutReg.pCANoutputData = &CANsend7;
                                        break;
                                }
                        }
/*****************************************************************************************/
/*                      Input Control Register                                           */
/*****************************************************************************************/
                typedef struct Inputreg{
                        bool dInput1;
                        bool dInput2;
                        bool dInput3;
                        bool dInput4;
                        bool dInput5;
                        bool dInput6;
                        float aInput5;
                        float aInput6;
                }Inputreg;

                Inputreg InputReg;//digital and analog input values
/*****************************************************************************************/
/*                      ADC Functions                                                    */
/*****************************************************************************************/
                void setADMUX(uint8_t ADCnum){//select ADC input to read
                        switch(ADCnum){
                        case 0:
                                ADMUX &= ~((1<<MUX2)|(1<<MUX1)|(1<<MUX0));
                                break;
                        case 1:
                                ADMUX|= (1<<MUX0);
                                ADMUX &= ~((1<<MUX2)|(1<<MUX1));
                                break;
                        case 2:
```

```
                ADMUX|= (1<<MUX1);
                ADMUX &= ~((1<<MUX2)|(1<<MUX0));
                break;
        case 3:
                ADMUX|= (1<<MUX0)|(1<<MUX1);
                ADMUX &= ~(1<<MUX2);
                break;
        case 4:
                ADMUX|= (1<<MUX2);
                ADMUX &= ~((1<<MUX1)|(1<<MUX0));
                break;
        case 5:
                ADMUX|= (1<<MUX2)|(1<<MUX0);
                ADMUX &= ~(1<<MUX1);
                break;
        case 6:
                ADMUX|= (1<<MUX2)|(1<<MUX1);
                ADMUX &= ~(1<<MUX0);
                break;
        case 7:
                ADMUX|= (1<<MUX2)|(1<<MUX1)|(1<<MUX0);
                break;
        }
}


uint16_t highPrecisionRead(uint8_t ADCnum){//read ADC
        setADMUX(ADCnum);
        int voltage;
        int CalibratedVoltage;
        ADCSRA = 0xC7;                  // start conversion
        while(ADCSRA &(1<<ADSC));
        voltage = ADC & 0x3FF;                  // read voltage
        CalibratedVoltage = voltage+((0.014*voltage)- 13.8);


        return CalibratedVoltage;
}

float ReadLC_Current(void){//Read and average low current output current
        CANoutputData *pSendCANdata;
        pSendCANdata = OutReg.pCANoutputData;

        float Current;
        int Currentb;
        int ReadValues[5];
        int Average2[5];
        int Average3[10];
        uint8_t i = 0;
        uint8_t j = 0;
        uint8_t k = 0;

        while(k<10){
          while(j<5){
            while(i<5){
                ReadValues[i] = highPrecisionRead(0);
```

```
                                    i++;
                                }

                                    Average2[j] = ReadValues[0] + ReadValues[1]+
                                                  ReadValues[2] +ReadValues[3] +
                                                  ReadValues[4];
                                    Average2[j] = Average2[j]/5;
                                    j++;
                                }

                                     Average3[k] = Average2[0] + Average2[1] + Average2[2] +
                                                   Average2[3] + Average2[4];
                                     Average3[k] = Average3[k]/5;
                                     k++;
                                }

                        Currentb = Average3[0] + Average3[1] + Average3[2] + Average3[3] +
                                   Average3[4] + Average3[5] + Average3[6] + Average3[7] +
                                   Average3[8] + Average3[9];

                        Currentb = Currentb/10;
                        pSendCANdata->LCCurrent = static_cast<uint16_t>(Currentb);
                        Current = (0.00603*Currentb) +0.111;

                        if(Current<0){
                                Current = 0;
                        }
                        return Current;
}

float ReadHC_Current(void){//read and average high current output current
                        CANoutputData *pSendCANdata;
                        pSendCANdata = OutReg.pCANoutputData;

                        float Current;
                        int Currentb;
                        int ReadValues[5];
                        int Average2[5];
                        int Average3[10];
                        uint8_t i = 0;
                        uint8_t j = 0;
                        uint8_t k = 0;

                        while(k<10){
                          while(j<5){
                            while(i<5){
                                ReadValues[i] = highPrecisionRead(2);
                                i++;
                                    }

                               Average2[j] = ReadValues[0] + ReadValues[1] +ReadValues[2] +
                                             ReadValues[3] + ReadValues[4];
                                             Average2[j] = Average2[j]/5;
                                             j++;
                                    }
                              Average3[k] = Average2[0] + Average2[1] + Average2[2] +
```

```
                                        Average2[3] + Average2[4];
                                        Average3[k] = Average3[k]/5;
                                        k++;
                                }

                        Currentb = Average3[0] + Average3[1] + Average3[2] + Average3[3] +
                                Average3[4] + Average3[5] + Average3[6] + Average3[7] +
                                 Average3[8] + Average3[9];
                        Currentb = Currentb/10;
                        pSendCANdata->HCCurrent = static_cast<uint16_t>(Currentb);

                        Current = (0.048*Currentb) +0.826;

                        if(Current<0){
                                Current = 0;
                        }
                        return Current;
                }
/*********************************************************************************************/
/*                      UpdateConditions                                                     */
/*********************************************************************************************/
/*
 * This function updates all conditions based on input values, output currents and voltages
 * and other global conditions. These conditions are then stored in boolean type to be used
 * in other functions to determine if an output should be on or off.
 */
                typedef struct Conditionreg{
                        bool defaultCondition;
                        bool EngineRPMthresh;
                        bool EngineTempthresh;
                        //Add more bool conditions here from CAN
                }Conditionreg;

                Conditionreg ConditionReg;


                void UpdateConditions(void){
                        volatile OutputControl *pHCControlReg;
                        OutputSetup *pHCSetupReg;
                        volatile OutputControl *pLCControlReg;
                        OutputSetup *pLCSetupReg;


                        //Check Global Conditions First
                        if(GlobalControl.BatteryVoltage < MINIMUM_BATTERY_VOLTAGE){
                                GlobalControl.GlobalEnable = false;
                        }

                        if(GlobalControl.Temperature > MAX_TEMPERATURE){
                                GlobalControl.GlobalEnable = false;
                        }

                        //Check Output Values

                        pHCControlReg = OutReg.pHCControlReg;
                        pHCSetupReg   = OutReg.pHCSetupReg;
```

```
                pLCControlReg = OutReg.pLCControlReg;
                pLCSetupReg   = OutReg.pLCSetupReg;

                //Determine if overcurrent event has occurred
                if(((pHCControlReg->OutCurrent) > (pHCSetupReg->CurrentLimit))&&
                  ~(pHCControlReg->InrushFlag)){
                        pHCControlReg->FuseBuffer = true;
                        if(pHCControlReg->timecount >= pHCSetupReg->FusingTime){
                                pHCControlReg->ResetTimeEnable = false;
                                (pHCControlReg->ResetAttemptsRemaining)--;
                                pHCControlReg->FuseEnabled = true;
                                pHCControlReg->FuseBuffer = false;
                                pHCControlReg->timecount = 0;
                        }
                }
                else{
                        pHCControlReg->FuseBuffer = false;
                        pHCControlReg->timecount = 0;
                }

                if(((pLCControlReg->OutCurrent) > (pLCSetupReg->CurrentLimit))&&
                  ~(pHCControlReg->InrushFlag)){
                        pLCControlReg->FuseBuffer = true;
                        if(pLCControlReg->timecount >= pLCSetupReg->FusingTime){
                                (pLCControlReg->ResetAttemptsRemaining)--;
                                pLCControlReg->ResetTimeEnable = false;
                                pLCControlReg->FuseEnabled = true;
                                pLCControlReg->FuseBuffer = false;
                                pLCControlReg ->timecount = 0;
                        }
                }
                else{
                        pLCControlReg->FuseBuffer = false;
                        pLCControlReg->timecount = 0;
                }

                //Determine status of digital inputs

                InputReg.dInput1 = (PIND & (1<<PD0)) ? false:true;
                InputReg.dInput2 = (PIND & (1<<PD1)) ? false:true;
                InputReg.dInput3 = (PIND & (1<<PD2)) ? false:true;
                InputReg.dInput4 = (PIND & (1<<PD3)) ? false:true;
                InputReg.dInput5 = (PINF & (1<<PF5)) ? false:true;

                //CAN Condition Check
                ConditionReg.EngineRPMthresh = (CAN_RX.EngineRPM > 100) ?
                                                        true:false; //If RPM is greater than 100
}
```

```
/*****************************************************************************************/
/*                  Enable and Disable Analog Redundancy                                 */
/*****************************************************************************************/
                void Transmit_SPI_Master(uint8_t Data1, uint8_t Data2)
                {
                        PORTB &= ~(1<<PB0);  // assert the slave select
                        SPDR = Data1;     // Start transmission, send high byte first. Send upper 8 bits
                        while (!(SPSR & (1<<SPIF)));// Wait (poll) for transmission complete
                        SPDR = Data2;    // Start transmission, send high byte first. Send lower 8 bits
                        while (!(SPSR & (1<<SPIF)));// Wait (poll) for transmission complete
                        PORTB |= (1<<PB0);     // deassert the slave select
                }

                void RedInit(void){ //Initialization values for Analog Redundancy DAC
                        Transmit_SPI_Master(0x32, 0x64);
                        Transmit_SPI_Master(0x72, 0x64);
                        Transmit_SPI_Master(0xB2, 0x64);
                        Transmit_SPI_Master(0xF2, 0x64);
                }


                typedef struct RedStatus{
                        bool RedundancyHC1;
                        bool RedundancyHC2;
                        bool RedundancyHC3;
                        bool RedundancyHC4;
                }RedStatus;

                RedStatus RedundancyStatusBuffer;
                RedStatus RedundancyStatus;

                void EnableRedundancy(uint8_t Output){
                        uint8_t data1;
                        switch(Output){
                        case 0:
                                data1 = 0x32;
                                RedundancyStatus.RedundancyHC1 = true;
                                break;
                        case 1:
                                data1 = 0x72;
                                RedundancyStatus.RedundancyHC2 = true;
                                break;
                        case 2:
                                data1 = 0xB2;
                                RedundancyStatus.RedundancyHC3 = true;
                                break;
                        case 3:
                                data1 = 0xF2;
                                RedundancyStatus.RedundancyHC4 = true;
                                break;
                        }

                        Transmit_SPI_Master(data1, 0x64);//set DAC output to 2 V
                }
```

```
void DisableRedundancy(uint8_t Output){
	uint8_t data1;
	switch(Output){
	case 0:
		data1 = 0x3F;
		RedundancyStatusBuffer.RedundancyHC1 = false;
		RedundancyStatus.RedundancyHC1 = false;
		break;
	case 1:
		data1 = 0x7F;
		RedundancyStatusBuffer.RedundancyHC2 = false;
		RedundancyStatus.RedundancyHC2 = false;
		break;
	case 2:
		data1 = 0xBF;
		RedundancyStatusBuffer.RedundancyHC3 = false;
		RedundancyStatus.RedundancyHC3 = false;
		break;
	case 3:
		data1 = 0xFF;
		RedundancyStatusBuffer.RedundancyHC4 = false;
		RedundancyStatus.RedundancyHC4 = false;
		break;
	}

	Transmit_SPI_Master(data1, 0xFF);//set DAC output to 5 V

	if(~(TIMSK1&(1<<OCIE1A))){
		TIMSK1 |= (1<<OCIE1A);//enable compare interrupt Timer 1 to
                                      re-enable Redundancy
	}
}

/*****************************************************************************************/
/*                  Output Switching                                                     */
/*****************************************************************************************/
/*
 * These functions turn the outputs on and off
 */
void OutputON(OutputNames OutputName){
	switch(static_cast<uint8_t>(OutputName)){
	case 0: //HC1
		//DisableRedundancy(0);
		PORTB |= (1<<PB4);
		HC1Control.InrushFlag = true;
		break;
	case 1: //HC2
		DisableRedundancy(1);
		PORTB |= (1<<PB5);
		HC2Control.InrushFlag = true;
		break;
	case 2: //HC3
		DisableRedundancy(2);
		PORTB |= (1<<PB6);
		HC3Control.InrushFlag = true;
		break;
```

```
                case 3: //HC4
                        DisableRedundancy(3);
                        PORTB |= (1<<PB7);
                        HC4Control.InrushFlag = true;
                        break;
                case 4: //LC1
                        PORTE |= (1<<PE3);
                        LC1Control.InrushFlag = true;
                        break;
                case 5: //LC2
                        PORTE |= (1<<PE4);
                        LC2Control.InrushFlag = true;
                        break;
                case 6: //LC3
                        PORTE |= (1<<PE5);
                        LC3Control.InrushFlag = true;
                        break;
                case 7: //LC4
                        PORTA |= (1<<PA0);
                        LC4Control.InrushFlag = true;
                        break;
                case 8: //LC5
                        PORTA |= (1<<PA1);
                        LC5Control.InrushFlag = true;
                        break;
                case 9: //LC6
                        PORTA |= (1<<PA2);
                        LC6Control.InrushFlag = true;
                        break;
                case 10: //LC7
                        PORTA |= (1<<PA3);
                        LC7Control.InrushFlag = true;
                        break;
                case 11: //LC8
                        PORTA |= (1<<PA4);
                        LC8Control.InrushFlag = true;
                        break;
                }

        }

        void OutputOFF(OutputNames OutputName){
                switch(static_cast<uint8_t>(OutputName)){
                case 0: //HC1
                        PORTB  &= ~(1<<PB4);
                        break;
                case 1: //HC2
                        PORTB  &= ~(1<<PB5);
                        break;
                case 2: //HC3
                        PORTB  &= ~(1<<PB6);
                        break;
                case 3: //HC4
                        PORTB  &= ~(1<<PB7);
                        break;
                case 4: //LC1
```

```
                                PORTE  &= ~(1<<PE3);
                                break;
                        case 5: //LC2
                                PORTE  &= ~(1<<PE4);
                                break;
                        case 6: //LC3
                                PORTE  &= ~(1<<PE5);
                                break;
                        case 7: //LC4
                                PORTA  &= ~(1<<PA0);
                                break;
                        case 8: //LC5
                                PORTA  &= ~(1<<PA1);
                                break;
                        case 9: //LC6
                                PORTA  &= ~(1<<PA2);
                                break;
                        case 10: //LC7
                                PORTA  &= ~(1<<PA3);
                                break;
                        case 11: //LC8
                                PORTA  &= ~(1<<PA4);
                                break;
                        }
                }


};//end class PDC

class testPDC{//test code for PDC
PDC pdcClass;
public:
        void RedOn(void){//redundancy test
                        Serial.printf("Red On\n");
                        pdcClass.Transmit_SPI_Master(0x2F, 0xFF);//**
                        pdcClass.Transmit_SPI_Master(0x6F, 0xFF);
                        pdcClass.Transmit_SPI_Master(0xAF, 0xFF);//**
                        pdcClass.Transmit_SPI_Master(0xEF, 0xFF);

                }

        void RedOff(void){//redundancy test

                Serial.printf("Red off\n");
                pdcClass.Transmit_SPI_Master(0x20, 0x00);
                pdcClass.Transmit_SPI_Master(0x60, 0x00);
                pdcClass.Transmit_SPI_Master(0xA0, 0x00);
                pdcClass.Transmit_SPI_Master(0xE0, 0x00);
        }

        void RedTest(void){
                RedOn();

                _delay_ms(200);
                _delay_ms(200);
                _delay_ms(200);
```

```
                RedOff();
                _delay_ms(200);
                _delay_ms(200);
                _delay_ms(200);
        }

        void ADCtest(void){
                uint16_t adcRetvalue;
                adcRetvalue = pdcClass.highPrecisionRead(7);
                Serial.printf("Raw ADC value is %d\n",adcRetvalue);
        }

        void OutOn(void){
                pdcClass.OutputON(PDC::OutputNames::LC1);
        }

        void UpdateInfo(uint8_t Check){
                pdcClass.UpdateData(Check);
        }
};
```