Spring 2019

Cycle Assist

Tyler Matthews
trm84@zips.uakron.edu

Cycle Assist

Final Design Report

Design Team 18

Ryan Applebee

Nick Darash

Tyler Matthews

Ethan Wesel

Dr. Yilmaz Sozer

11/29/2018

Table of Contents:

List of Figures:

List of Tables:

Abstract

Through thorough research on today’s exercise trends, studying both trainers and amateurs alike, it was noticed that cycling was among the most prescribed forms of cardiovascular exercise. The goal of this project is to modernize this ancient form of exercise to today’s standard by utilizing heart rate monitoring and motor control technologies that have only recently become mainstream. By measuring the user’s heart rate it is possible to gain a thorough understanding of how hard the user is working. If it was possible to keep the user’s heart rate constant for extended periods of time, it would then be possible to plan workouts where the outcome (calories burned) is known to a precise degree. This project takes the user’s heart rate and feeds it to an algorithm that controls a motor tied to a bicycle. This motor then constantly adjusts its output torque to force the user to work hard enough to keep their heart rate constant. This report follows the design process that will allow for this project to come to fruition.

1. Project Statement

1.1 Need

Biking is one of the most common types of exercise subscribed by physical therapists and athletic trainers. It's considered a great way to work on one's cardio, increase the strength in their lower body muscles, and burn substantial amounts of calories. It manages to do all of this while minimizing the chance of injuries that the user cold incur by limiting the amount of impact their joints receive. When riding a bike trainers typically want to keep the trainee's heart rate in a specific area, or intensity zone. This allows the exercise to be sustained longer and maximizes the consistency and productivity of the workout. In order to do this, someone has to continuously monitor their heart rate and manually adjust the resistance on the bike, causing them to lose focus on their workout. This means that someone else has to always be there, typically a personal trainer, or the trainee has to manage this themselves which causes them to lose the focus required to sustain a productive workout. This loss of focus, along with the cost of a personal trainer and gym membership, mean that many people do not have the means to obtain the highly productive workouts that they want. There is a need to autonomously adjust one's workout so that they can maintain a selected workout intensity for an extended period of time, thus maximizing the gains that are sustained from their workout.

1.2 Objective

The objective of this project is to provide customers with an easy-to-use bicycle that allows them to select a workout intensity and keeps them working within a given range of intensities centered around the one selected. Users will be riding a bicycle that will vary intensity autonomously based on their heart rate. As the user becomes more exhausted the bike will adjust pedal resistance, thus allowing the user to regain stamina and keep them in the selected intensity zone. By adjusting pedal resistance automatically, the user is free to work at their own pace and be assured that they are still getting the workout they desire. Also, if the user isn’t getting enough of a workout, the bike’s electric motor can be oriented as a generator, creating an extra load for the user to drive (causing them to work harder and increase their heart rate) while recharging the batteries.

1.3 Background

The intent of this device is to make a continuously variable resistance bicycle that will keep the user within a specific range of workout intensity without the user having to intervene. The resistance will be entirely continuous, always correcting for the users level of fatigue, and keep them from going outside their selected zone of intensity. To accomplish this there will be a DC motor assisting, or hindering, the user's pedaling causing the intensity of the workout to increase, or decrease, as necessary. The DC motor will be able to assist the user by adding torque to the drive wheel when they are getting over fatigued or adding the resistance to the drive wheel when the user is not working hard enough. When the motor is adding resistance it will act as a generator, recharging the battery pack that is being used to run the motor when it is assisting the user.

In September 1997 Saturo Kumagai and the Yazaki Corp patented an idea for a regenerative braking system for an electric bicycle. "A regenerative braking device 2 provided on an electric power-assisted motor fitted to a bicycle 1 controls the regenerative braking action by a regenerative braking means A fitted to a brake lever supporting point, and the regenerative braking is performed with the motor connected to a rear wheel in the regenerative charging mode. The regenerative braking device 2 is operated only in the traveling condition where the braking or deceleration is required by turning on/turning off a switch, and a bicycle driver feels no sense of incompatibility in the inertia traveling."[7] This is similar to the design that will be utilized to create the device described within this paper. The difference being that instead of switching to regenerative braking through the use of a switch, the change will be done autonomously based on the user's heart rate. If the user's heart rate is starting to slow down, that means they require more resistance and the bike will begin to impede the user's pedaling causing the motor to act as a generator and the energy they are producing to charge a battery. Vijyalakshmi, S. Sandhiya and C. Bhuvaneswari have an article that showcases the current design parameters of electric bikes with regenerative braking induced by a DC motor [4]. In this article they speak about utilizing a microcontroller to not only run the motor, but to also control the other vital features of the bike. The idea of utilizing a microcontroller to run the various functions of the bike will be vital to this design because it will limit the complexities of the system to a manageable amount (no need to deal with cross-board communications).

For this design, the user's heart rate will be used to keep track of their current zone of intensity. To do this a heart rate sensor is needed: N. Constant, T. Wang, and K. Mankodiya came up with a unique design for a simplistic heart rate sensor: "The technique places a light and photodetector on the surface of the skin. The light transmits through the skin and blood but is largely reflected by the bone thereby traveling back to the photodetector. As the light journeys through the skin, the photodetector monitors fluctuations in light absorption caused by the variations in blood volume passing through the arteries. This allows us to calculate the duration of the cardiac cycle."[3] The key principle of the design of their heart rate monitor is to monitor the change in light being absorbed, and thus emitted, from the skin which is directly proportional to the volume of blood being passed through the arteries. This is the same idea that will be utilized in the design of the heart rate monitor for this project.

The design of the bike is intended to utilize a heart rate monitor to read and control the fatigue level of the user. The most efficient way to do this is to have the user put on a heart rate monitor when riding the bike that will wirelessly transmit information about the user's heart rate back to the control system. Modern heart rate sensors use a light source pointed at the skin and measure the feedback of how much light is being absorbed in the skin from the presence of blood, which changes when the heart pumps. "The present invention relates to heart rate monitor for measuring a heart rate of a user, including at least one artificial light source for emitting light into skin of the user, a first sensor for sensing light reflected through the skin of the user and for generating a first sensor-signal in response to sensed light, and an optical high-pass filter for filtering out infrared light, wherein the optical high-pass filter is arranged in an optical path before the first sensor for filtering out infrared light from light travelling to the first sensor."[8] This basic design of a heart rate monitor as described here will be a good simple design for the concept of the bike. The information will be relayed back to the bike and the information will be used to apply a torque on the drive wheel. This can either assist the user in strenuous situations or make it harder for the user in less strenuous situations.

Currently there are a few common ways bikes are used to convert mechanical energy into electrical energy. One particular approach that can be used as a model for this project was that taken by Hudson Harr, founder of ReRev based out of Clearwater, Florida. He purchased many elliptical machines and began tearing the equipment apart. What he found was that some of these machines used DC generators to vary the resistance the user feels while pedaling. "The current generated creates a magnetic force that opposes the motion that creates this current. By adjusting the amount of current created, the user can vary the resistance he or she feels."[2] Normally the generators use a bank of resistors to dissipate the extra energy, but Harr decided to remove the internal resistance of the DC generator and instead hook it up to an external load. Using an inverter, the next step was to transform the DC power generated into AC and finally feeding it into the electrical grid.[2] For this project, the implementation of a generator that can create pedal resistance will be approached in a similar way. Applying this method of power generation to power an external source will be utilized for an actual bicycle. It will create more resistance when the user has a light load and assist the user when the load is strenuous. A control system that alters a variable resistance to autonomously control the current being sent to the motor or drawn from it will be implemented to realize this design, depending upon the users heart rate and specified range in which they wish to operate.[2]

To keep the user in the desired heart rate range the control system will need to obtain certain inputs from the bicycle and the user. For this particular application one of the inputs will be the users heart rate which is obviously proportional to their fatigue level. Other inputs may be the torque created, the frequency of peddling, the slope of the riding surface, or the bike speed. Depending upon these values the control system will determine the assisted torque that needs generated from the motor.[1] To electrically-assist the rider, implementation of a constant proportion-assisted power controller (PAPC) could be used in which the assistance provided by the motor is proportional to the provided torque and speed of the bike. Another implementation could be a fuzzy logic controller (FLC) which instead of the torque, it takes the peddle frequency as an input to determine the amount of assistance needed.

Another integral part of the bike will be the batteries that will be charged/discharged during use. “Battery management systems are of paramount importance to ensure safe functioning and optimal usage of the battery pack.”[6] In order to keep the user safe the bike will have to have a sophisticated water proof battery management system that will allow the batteries to be charged and discharged while ensuring that they are not damaged. It is also vital that the batteries do not overheat while in use. Part of the battery management system will have to continuously monitor the temperature of the batteries and adjust load if needed. Safety should be the most important aspect of the bike and ensuring that the batteries are safe to operate is a huge step in maintaining safe operation of the system. If the bike battery is not generating enough energy for sufficient assistance, the use of a super capacitor could be used. “Super-capacitor modules are used to provide the high current required during starting and acceleration, and eventfully will help increasing lifespan of battery. A secondary source, like regenerative braking or a small solar panel module could be availed onboard so as to charge battery/super capacitor.”[5]

Limitations of current designs stem from the fact that feedback to the motor control are typically coming from the bike itself. Such inputs can be the speed of the bike or the torque applied to the pedals. In this design the motor control system will take an input directly from the user, in this case, the user’s heart rate. This improvement will make the bike much more practical for exercise purposes, because it will be able to directly measure the fatigue of the user and keep them in the ideal workout zone.

1.4 Marketing Requirements

1. Be able to vary pedal resistance autonomously
2. Assist user on inclines
3. Accurately read user's heart rate
4. Attempt to keep user's heart rate in a selected intensity zone
5. Be able to assist the user for the duration of a typical cycling workout
6. Utilize a portable and rechargeable energy storage device
7. Have a safe and reliable energy management system
8. Have a user activated shutdown
9. Display health and ride parameters
10. Transmit user's heart rate to the rest of the system wirelessly
11. The additional weight of the bicycle's electronics won't overload the user if they are pedaling without assistance

1.5 Objective Tree

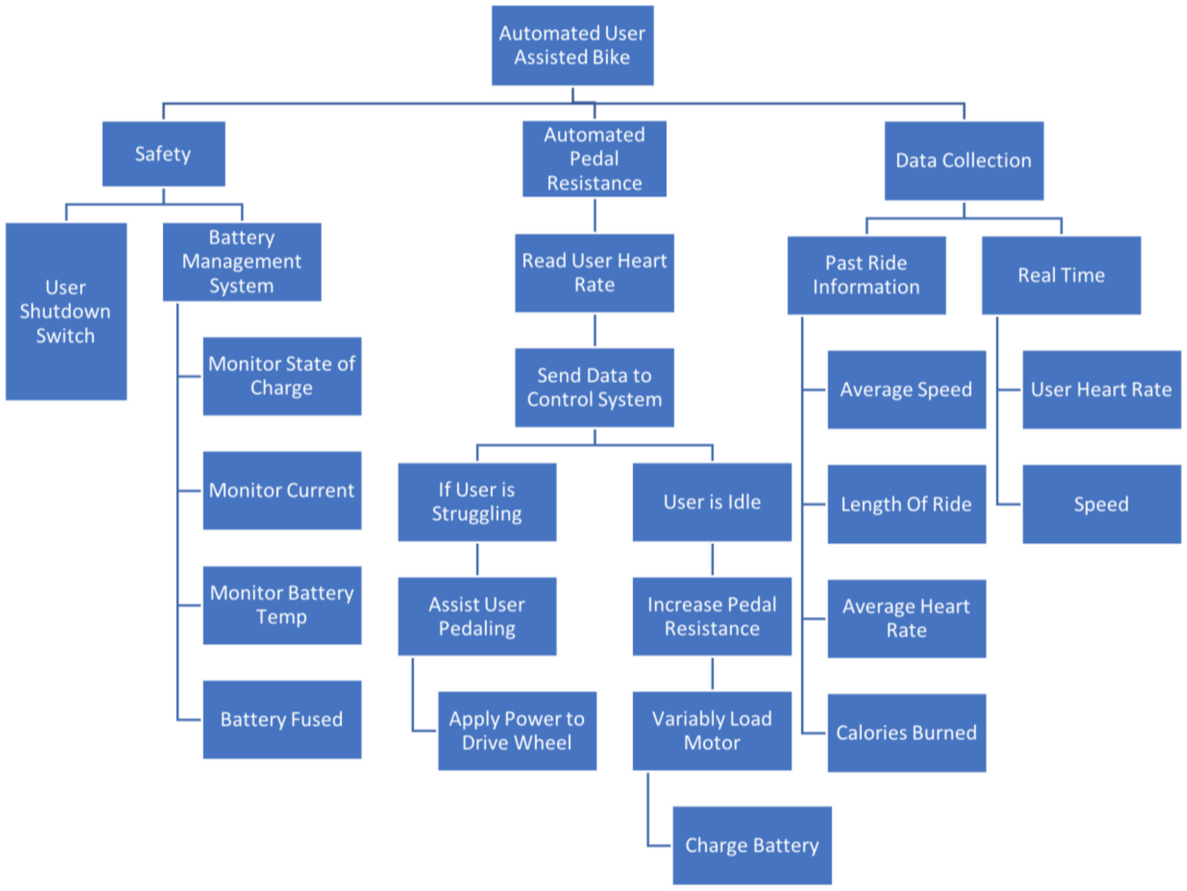


Figure 1 - Objective Tree

2. Design Requirement Specification

Marketing Requirements	Engineering Requirements	Justification
1, 2	The bicycle will utilize at least a 400W DC hub motor to drive one of the wheels.	It takes slightly less than 400W to drive a 260lbs load up a 6 degree incline without assistance.
5, 6, 7	The bicycle will be able to assist the user for at least one hour.	The average length of a typical cardiovascular workout is less than one hour and the bicycle should be able to assist for longer than a typical workout.
3	The embedded system will take measurements from the heart rate sensor at least every tenth of a second.	By taking a reading every 0.1 seconds, the embedded system will be able to make minor adjustments to the motor's assistance.
4	The bicycle will be able to keep the user's heartrate within ±10% of a selected value.	A range of 10% is large enough that the system will be able to keep the user within it with minor adjustments, but small enough to keep a uniform workout.
9	An LCD display and associated controller will be used to display the user's heart rate, speed, and ride duration in real time.	Heart rate, speed, and ride duration will be the most important information for a user and by updating it in real time the information shown will always be accurate.
2	The bike will be able to assist the user up a 6 degree (10 % grade) slope.	6 degree slopes are the largest that are typically seen on bike paths.
7, 8	The energy management system will interrupt power flow to, or from, the motor in less than 0.1 seconds.	By isolating the batteries in under 0.1 seconds, the amount of damage done to the batteries and other electronics on the bicycle will be limited.
10	The heart rate monitor will be able to transmit data at least 10 feet.	10 feet is an appropriate distance to ensure the data will reach the controller.
11	The bicycle's associated electronics will weigh less than 60lbs.	If the electronics weight less than 60lbs, then the bike will still be usable without electronic assistance.
1	The motor controller must be able to send 400W of power from the batteries to the motor.	The assist, or resist function, of the bike can occur at any time and the motor must be able to transition continuously.
1	The motor controller must be able to send power back to the batteries from the motor.	In order to increase the load the user is driving, thus increasing the user's heart rate, the motor must act like a generator and produce power which will be routed back to the batteries.

Table 1 – Design Requirement Specifications

2.1 Marketing Requirements

1. Be able to vary pedal resistance autonomously
2. Assist user on inclines
3. Accurately read user's heart rate
4. Attempt to keep user's heart rate in a selected intensity zone
5. Be able to assist the user for the duration of a typical cycling workout
6. Utilize a portable and rechargeable energy storage device
7. Have a safe and reliable energy management system
8. Have a user activated shutdown
9. Display health and ride parameters
10. Transmit user's heart rate to the rest of the system wirelessly
11. The additional weight of the bicycle's electronics won't overload the user if they are pedaling without assistance

3. Accepted Technical Design

3.1.1 Level 0 Hardware Theory of Operation

A typical electronic bicycle takes input from a throttle and a DC hub motor. The hub motor has a controller which increases or decreases its outputted torque according to what the throttle commands. In this system, the bike will utilize the user's heart rate as a throttle. This will allow the bike to adjust its output accordingly without the user having to consciously input anything into the system.

3.1.2 Level 0 Hardware Diagram

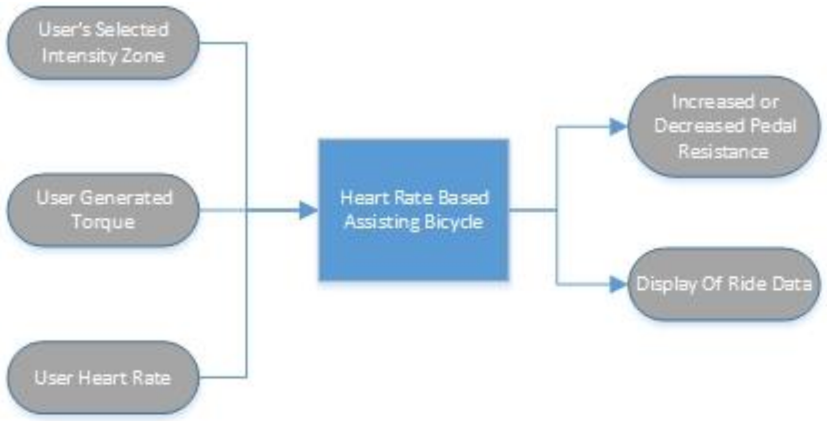

User's Selected Intensity Zone
User Generated Torque
User Heart Rate
Heart Rate Based Assisting Bicycle
Increased or Decreased Pedal Resistance
Display Of Ride Data

Figure 2 – Level 0 Hardware Diagram

A heart rate based user assisting bicycle, as seen in *Figure 2*, will be designed and built so that it will be able to assist a user pedaling their bicycle. It will utilize the user's heart rate as a throttle and attempt to keep their heart rate in a specified intensity zone. As the user rides it will display pertinent data to them.

Module	Heart Rate Based Assisting Bicycle
Inputs	• User selected intensity zone • User generated torque • User heart rate
Outputs	• Display heart rate, speed, and ride length • Torque to the driveshaft
Description	The user's heart rate, intensity zone, and generated torque will be used to decide how to drive the motor. Various ride data will be displayed to the user throughout the ride.

Table 2 – Functional Requirements of Heart Rate Based Assisting Bicycle

3.2.1 Level 1 Hardware Theory of Operation

A heart rate monitor attached to the user will be used to measure their heart rate and send that data to the embedded system of the bicycle. The embedded system will use this data, along with the drive train's rotations per minute, the batteries state of charge and the motor phase, to decide what the motor should do. The controller then produces the corresponding control signal to send to the motor controller. Along with that control signal the embedded system will send the data to be displayed to the display. The entire system, excluding the heart rate monitor, will be powered by a set of batteries that are controlled and monitored by a battery management system.

3.2.2 Level 1 Hardware Diagram

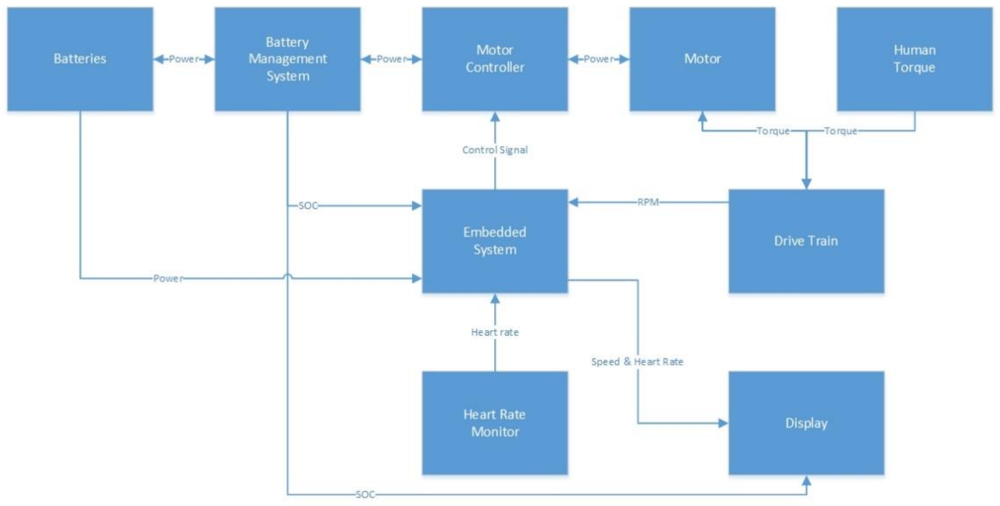


Figure 3 – Level 1 Hardware Diagram

The level 1 hardware diagram, as seen in *Figure 3*, encompasses the entire system that will be required to create a heart rate based assisting bicycle. A 48V battery pack will be utilized to power the entire system and be controlled by the battery management system. The embedded system will collect all of the appropriate data (heart rate, rpm, state of charge, etc.) and utilize it to route the appropriate amount of power through the motor controller to the motor. The motor will then drive the drive train and assist the user the amount needed to keep them in the specified intensity zone.

Module	Heart Rate Monitor
Designer	Ethan Wesel & Ryan Applebee
Inputs	• User's heart rate • Battery power (1.5V)
Outputs	• Bluetooth signal that corresponds to the user's heart rate
Description	The heart rate monitor will measure the user's heart rate and then convert that to a Bluetooth signal which will be sent to the rest of the system.

Table 3 – Level 1 Heart Rate Monitor Functional Requirements

Module	Batteries
Designer	Tyler Matthews
Inputs	• 48V Power Signal
Outputs	• 48V Power Signal
Description	The batteries will either supply power to the motor, and the rest of the bike's electronics, or the motor will act as a generator and supply power to the batteries.

Table 4 – Level 1 Battery Functional Requirements

Module	Battery Management System
Designer	Tyler Matthews
Inputs	• 48V Power
Outputs	• 48V Power • Battery State Of Charge
Description	The battery management system will ensure the batteries are safely monitored. While the bike is running, it will route power to the rest of the electronics and send the batteries SOC to the embedded system.

Table 5 – Level 1 Battery Management System Functional Requirements

Module	Motor
Designer	Ryan Applebee, Ethan Wesel
Inputs	• Power
Outputs	• Torque • Power
Description	The motor will be attached to the drivetrain and can act to assist the user by adding torque. Alternatively, the motor can act as a generator and send power to charge the batteries.

Table 6 – Level 1 Motor Functional Requirements

Module	Drivetrain
Inputs	• Torque
Outputs	• Torque • RPM
Description	The drivetrain will either use torque provided by the motor, to assist the user, or provide torque to the motor which will charge the batteries. Also, it will output its rotations per minute to the embedded system.

Table 7 – Level 1 Drivetrain Functional Requirements

Module	Display
Designer	Nick Darash
Inputs	• Power • Battery State of Charge • Drivetrain RPM
Outputs	• Displays data to the user.
Description	The display will take power from the batteries and data from the embedded system to display data to the user.

Table 8 – Level 1 Display Functional Requirements

Module	Motor Controller
Designer	Ryan Applebee, Ethan Wesel
Inputs	• Pulse Width Modulated Control Signal • Regenerative Braking Power
Outputs	• Motor Power • 48V Regenerative Breaking Power
Description	The Motor Controller will act as a bi-directional power supply to either power the motor or recharge the batteries. When powering the motor, it will utilize a control signal from the embedded system to control the power flow

Table 9 – Level 1 Controller Functional Requirements

Module	Embedded System
Designer	Nick Darash
Inputs	• Power • Battery State Of Charge • RPM • Heart Rate • Motor Position • Phase Currents
Outputs	• Display data • Control signal to the motor
Description	The Embedded System is the brains behind the entire bike. It will take in all the data from the other systems and decide how the motor will operate. It will also interact with the user by receiving the heart rate signal and display appropriate information on the screen.

Table 10 – Level 1 Embedded System Functional Requirements

3.4.1 Level 2 Embedded System Hardware Theory of Operation

The embedded system will take in signals from the batteries such as power and state of charge. It will also use signals from the heart rate monitor and the Hall Effect sensor from the drivetrain and feed the information to a microcontroller. The microcontroller will then use the data gathered and an algorithm we write in software to command the motor more or less torque. This controller will also be used as a motor controller. The controller will need to control the switches in the inverter using six different amplified PWM signals. The 6 PWM signals will be sent into the inverter to control the switches. In order make the motor impede the user we need to drive a negative current into the motor. This is done through closing and opening different switches, through the switching operation we can cause the current to go –180 degrees out of phase and thus create the negative current that is needed. This will apply toque on the wheel and make it more difficult to pedal. We will be using three Hall Effect sensors to sense the phase of the motor. The phase of the motor will be used to indicate the timings of which the switches will be activated.

3.4.2 Level 2 Embedded System Hardware Diagram

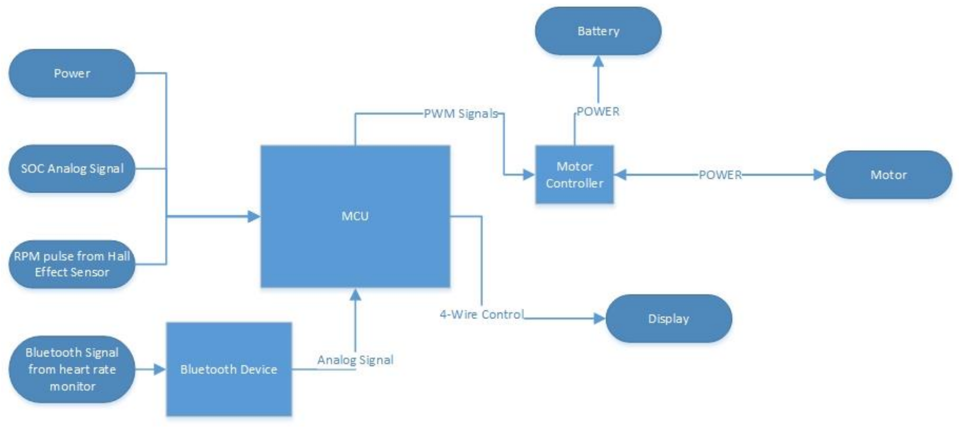


Figure 4 – Level 2 Embedded System Hardware Diagram

The embedded system will receive power from the batteries along with a state of charge signal from the battery management system. It will display state of charge on the display along with the speed the bike is travelling. The speed of the bike is calculated by taking in pulses from a Hall Effect sensor that pulses once for every revolution of the wheel. Lastly the embedded system gets the users heart rate via Bluetooth. It uses this information to the set the duty cycle and the switching speed of 6 PWM signals to send to the motor controller.

Module	MCU
Designer	Nick Darash
Inputs	• Heart Rate Signal • Drivetrain RPM • Battery State Of Charge Signal • Motor position • Motor Current
Outputs	• Motor Control Signals • Display Data
Description	Reads the user's heart rate, drivetrain rpm, Motor Position, Motor Current, and battery SOC and outputs different signals accordingly. Will have an algorithm to control the motor and display data on the screen

Table 11 – Embedded System MCU Functional Requirements

Module	Motor Controller
Designer	Nick Darash
Inputs	• 6 PWM Control Signals • Back EMF from Motor
Outputs	• Battery Power • 24 Volt Regen Power
Description	When the bike is assisting, the motor controller will be controlled by the MCU and deliver the battery power to the motor. In regen mode power from the motor will be used to recharge the battery.

Table 12 – Embedded System Motor Controller Functional Requirements

3.5.1 Level 2 Battery Management System Hardware Theory of Operation

The battery management system will read voltages from each series connection in the battery in order to manage them properly. The system will use these voltages to calculate an accurate state of charge of the batteries and send that signal to the embedded system. The battery management system will also be able to detect faults on the system and shut down the batteries in case of an emergency. If the batteries are in a safe operating range, the system will also allow them to charge safely and use power resistors to bleed off energy if the cells become unbalanced. The battery management system will also ensure the batteries remain within a sustainable temperature range, and turn off power if they get too hot. Most importantly the system will protect against overcurrent, overvoltage and under voltage when the cells are being charged or discharged.

3.5.2 Level 2 Battery Management System Hardware Diagram

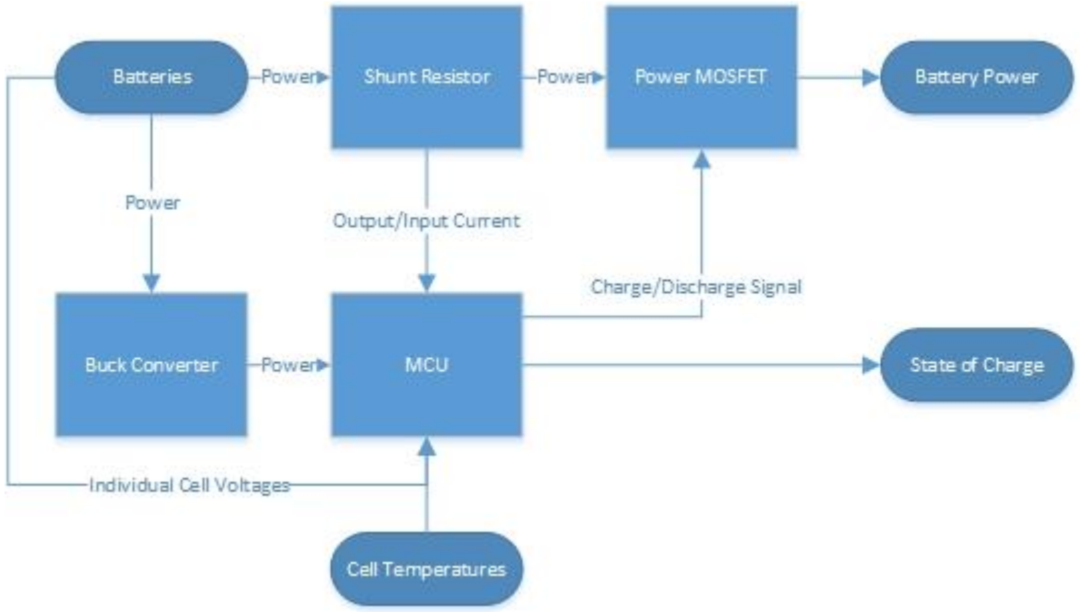


Figure 5 – Level 2 Battery Management System Hardware Diagram

The microcontroller will read voltage across a shunt resistor in order to accurately read current. It will control power flow to the drivetrain with a power MOSFET / MOSFET driver and read cell temperature directly using thermistors. All this information will be used by an algorithm in the microcontroller to ensure the batteries are operating safely. Then the microcontroller will send a state of charge signal to the embedded system to be displayed on the screen.

Module	MCU
Designer	Tyler Matthews & Ethan Wesel
Inputs	• Battery0 Voltage • Battery Current • Battery Temperatures
Outputs	• Charge / Discharge Signal • Battery SOC
Description	The microcontroller will take in the individual cell voltages, temperatures, and the overall pack's current to make decisions regarding the battery operation. It will also output the batteries SOC to the embedded system.

Table 13 – Battery Management System MCU Functional Requirements

Module	Buck Converter
Designer	Tyler Matthews & Ethan Wesel
Inputs	• 24V Battery Power
Outputs	• 3.3V Power
Description	The buck converter will take power from the batteries and step it down to a reasonable level to power the MCU.

Table 14 – Battery Management System Buck Converter Functional Requirements

Module	Power MOSFET
Designer	Tyler Matthews & Ethan Wesel
Inputs	• 24V Battery Power • Charge / Discharge Signal
Outputs	• 24V Battery Power
Description	The power MOSFET will be used to control the flow of power to / from the batteries to the motor controller. It will need to be able to handle high voltage and current (24V, 20A) from the batteries while having a low impedance path from its source to drain.

Table 15 – Battery Management System Power MOSFET Functional Requirements

Module	Shunt Resistor
Designer	Tyler Matthews & Ethan Wesel
Inputs	• 24V Battery Power
Outputs	• Voltage differential that corresponds to the current flowing
Description	The voltage drop across the shunt resistor, will be measured by the MCU and will be proportional to the voltage flowing through it. It will need to be low impedance and be able to hand high currents (20A).

Table 16 – Battery Management System Shunt Resistor Functional Requirements

Module	Batteries
Designer	Tyler Matthews & Ethan Wesel
Inputs	• N/A
Outputs	• 24V Power
Description	The batteries will be able to supply 24V and high currents to power the rest of the electronics.

Table 17 – Battery Management System Batteries Functional Requirements

3.6.1 Software Theory of Operation for the Embedded System

The embedded system will manage the operations of the entire bike and motor. It will be required to keep track of the user's heart rate and keep the motor operating at proper torque and speed. It will then be required to display pertinent data on a display so that the user can have an understanding of the system conditions. The embedded system will look at the speed of the bike, the user's heart rate, and the state of charge from the battery, in order to operate correctly. Using all this data the bike will be able to operate the motor in a safe and effective manner.

3.6.2 Software Diagram for the Embedded System

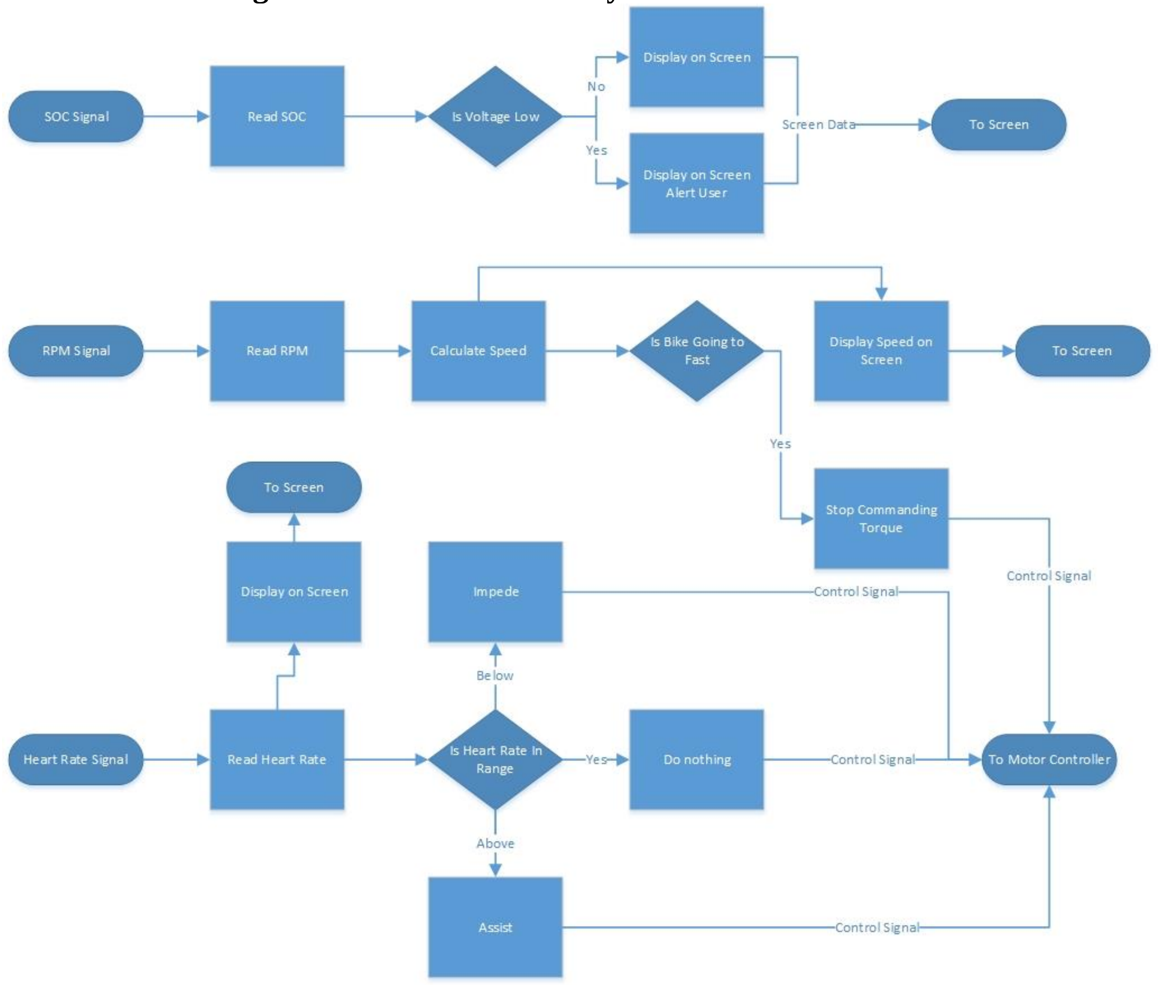


Figure 6 – Embedded System Software Diagram

The software diagram, as seen in *Figure 6*, encompasses the motor control and display. This system should be able to take state of charge, drivetrain rpm, user heart rate and motor position. It will then create a control signal that will tell the switches to turn either open or close and weather the motor should start adding resistance to make it harder to pedal, or assist the rider. Lastly, it should be able to use the collected data to create a signal to tell the display what to show the user.

3.7.1 Software Theory of Operation for the Battery Management System

The battery management system will be responsible for monitoring the batteries, ensuring the batteries are operating within safe parameters, and allow the batteries to power the rest of the electronics. This system will take measurements of the battery voltage, temperature, and input/output current to determine if the batteries are operating correctly.

3.7.2 Software Diagram for the Battery Management System

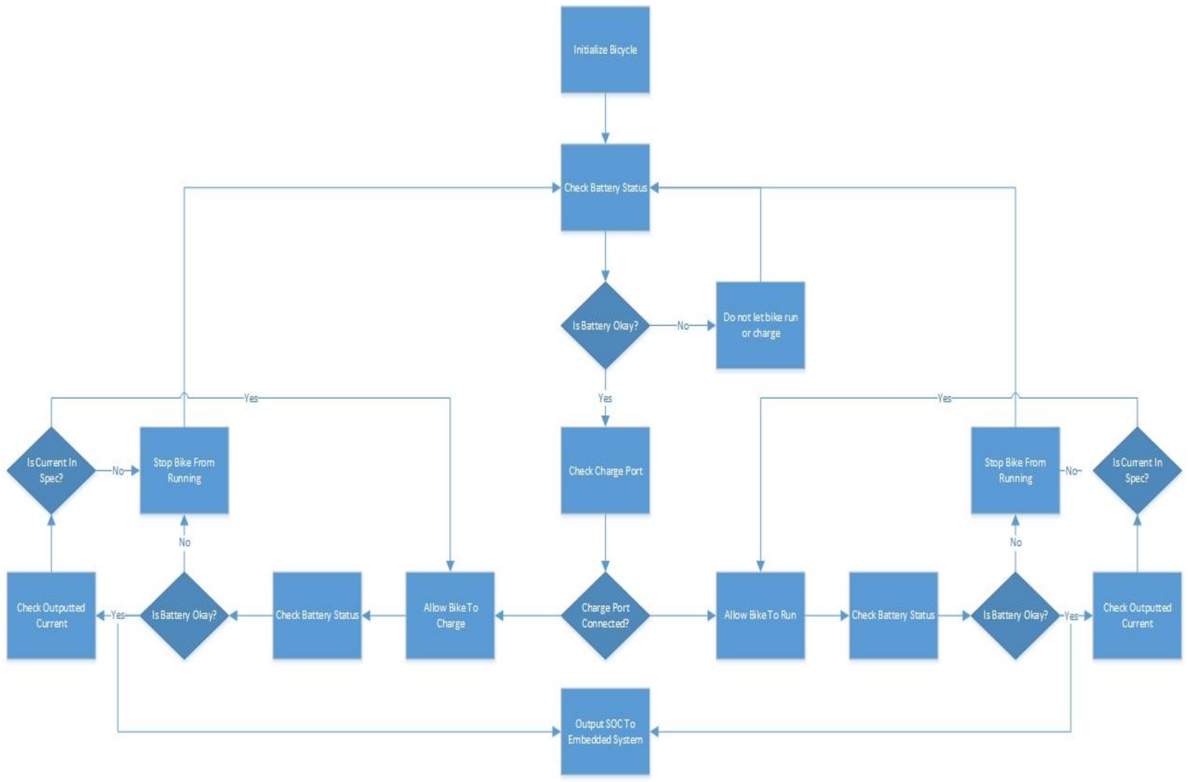


Figure 7 – Battery Management System Software Diagram

The level 1 software diagram, as seen in *Figure 5*, showcases the operation of the battery management system. This system constantly takes measurements of the battery voltages, temperatures, and current. If any of these values appear out of the ordinary then the system will stop the bike from starting or, if the bike had already started, it will stop the bike from continuing to run. If the batteries are being charged, then the battery management system will allow them to continue charging as long as the batteries are working with spec. Finally, the battery management will calculate the battery’s state of charge and send it to the embedded system.

3.8.1 Level 3 Battery Management System Hardware Theory of Operation [TRM]

As previously discussed, the three main objectives of a battery management system is to ensure that the batteries' voltage, current, and temperatures are within a safe operating zone. In order to do this, the battery management has three separate subsystems that each monitor one of above parameters and sends the measured values to a central MCU. If the MCU gets any value that's outside of the safe operating zone, it disconnections the batteries from the rest of the bicycle.

The first requirement for the battery management system is to monitor the battery cells' voltages. Since the battery pack for this project is a high voltage (44.4V nominal), connecting the voltage sense lines directly to the MCU ADCs would destroy it. One way to get around this is to use a voltage divider circuit to minimize the voltage sampled by the ADCs and then scaling the measured value back to what it should be in software. The issue with this is that it constantly burns power off through the resistors and a lot of resolution is measured when the values are scaled down. To get around these issues, a battery monitoring IC (LTC6804) was selected to measure the voltages. It then sends the values it measures to the central MCU through the SPI communication protocol. On top of that, the IC has built in FET drivers that allow it to passively balance (passive balancing keeps battery cells that are connected in series at the same voltage level by burning off excess power through a load resistor) the battery cells.

The second requirement for the battery management system is to monitor the current flowing into, or out of, the battery. Typically when measuring currents in a circuit, one would use a shunt resistor (resistor with a small impedance) and then measure the voltage drop across the shunt to derive the current flowing through that line (using I = V/R). In the case of the battery management system, the peak current flowing from the batteries to the rest of the bicycle is expected to be around 30 amps. This means that even a 0.1 ohm shunt resistor will cause losses in excess of 90W (P = I^2 * R) which is unacceptable. To mitigate this issue, a hall-effect based current sensor was selected. A hall-effect based current sensor measures the magnetic field generated by the current through the conductor and then derives the current based on that magnetic field. This allows the current to be measured with minimal losses, since the only added resistance is the length of conductor required to connect the sensor to the circuit. The sensor that was selected for this application, MLX91210, is a bi-directional hall-effect based current sensor that out puts a voltage that corresponds to the current it measures. It can handle up to 50 amps in either direction with a resolution of 50mV outputted for every amp flowing through it.

The third requirement for the battery management system is to monitor the battery temperature while it is being operated. Because the battery is going to be rather large (5P12S = 60 individual cells), the temperature throughout it won't necessarily be uniform. To combat this, the temperature will be measured at multiple points throughout it. Typically, when measuring temperatures, either diodes or thermistors is used. The voltage drop across the PN junction of a diode, or transistor, has an inversely proportional linear relationship with temperature. Meaning, that as temperatures increase, the voltage drop across that junction decreases by a set amount (ie: 0.05V / 20 degrees C). Also by paralleling the diodes, or transistors, only one reading needs to be taken because the diode with the smallest voltage drop is the only one that will be "active" and that corresponds to the diode with the highest temperature. Unfortunately, to get this to work, the diodes have to be perfectly matched and the precise measuring techniques are needed to get accurate values. Because of this, the battery management system is implementing thermistor based temperatures sensors instead. The thermistors chosen are 10K NTC (negative thermal coefficient), which have a nominal impedance of 10K ohms and the impedance decreases as their temperature increases. By putting each of these in series with a 10K resistor, whose temperature should be stable, and measuring the voltage between them, the temperature of the NTC can be easily derived (the difference in voltage across the NTC can be used to calculate it's impedance, and then a lookup table for its resistance at very temperatures can be used to find its temperature). One of the main drawbacks of this system is that it is constantly burning power through the resistors and NTCs. To combat this, the temperature sense circuit is fed through an N-channel MOSFET that will only allow current to flow when the temperatures are going to be measured.

If anything goes wrong, the MCU has to be able to disconnect the battery from the rest of the bicycle, or charger if the batteries are being charged. Because the battery will be delivering a lot of power, a power transistor or relay is needed to connect and disconnect them. A relay would work, but they typically act slower than transistors and, assuming the use of normally open non-latching relays, they would constantly be burning power to keep the battery connected. By using a MOSFET instead, the circuit is able to switch quickly and minimize the current draw (MOSFET draw little to no current when they are not switching). N-Channel MOSFET's require a voltage at its gate larger than that present on its source in order to switch it on and since that battery management system has no control on what is happening outside of its circuitry (MOSFET's source would be the output to the rest of the bike) it's impossible to predict what the voltage at its source would be. So, it was decided to use a P-Channel MOSFET since they only require the gate voltage to be lower than the source voltage to turn them on. Unfortunately this means, to turn it off, its gate voltage has to be almost equal to its source voltage. This is easily remedied with a resistor tied between its source and its gate, so that whenever its gate is not pulled to ground it'll be forced to the same voltage as its source. To turn the MOSFET on, its gate then needs to be pulled significantly lower than its source voltage. To do this, an N-Channel MOSFET is trigger by the MCU and then it pulls the gate of the P-Channel closer to ground. The gate of the P-Channel is not pulled directly to ground, because it can only handle a given difference (about 20V) between its gate and its source before it fails. Also, the MCU cannot directly pulled the P-Channel to ground because it lacks the current sinking capabilities required to do so.

3.8.2 Level 3 Battery Management System Schematic Diagrams [TRM][ELW]

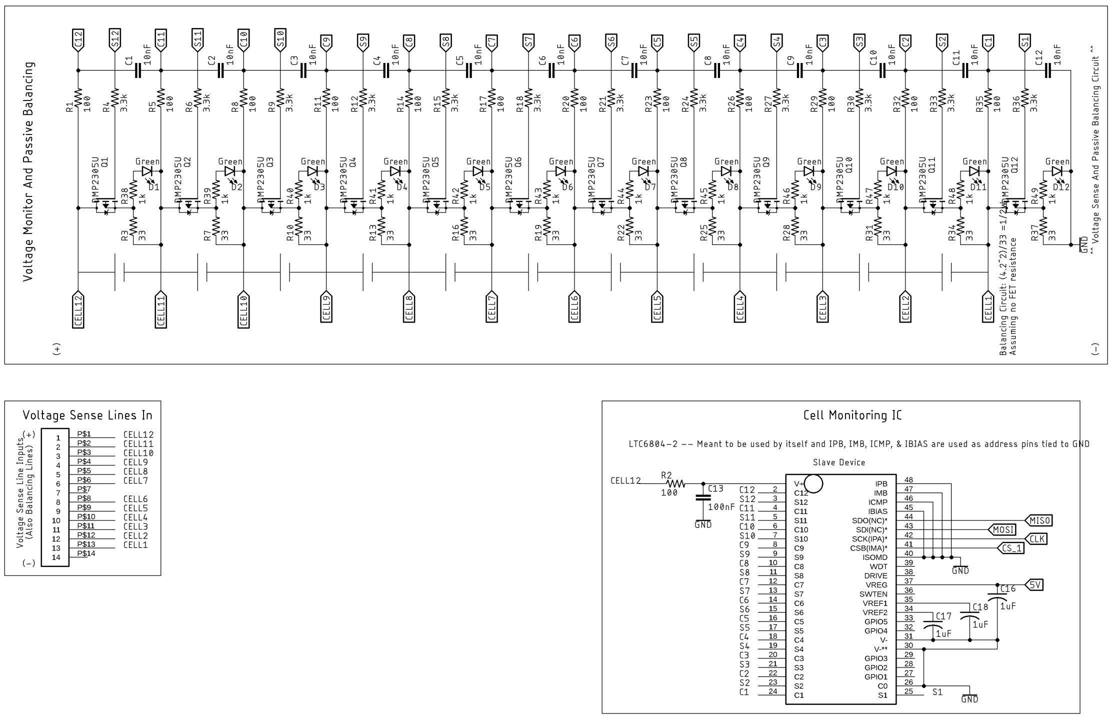

Voltage Monitor And Passive Balancing
Voltage Sense And Passive Balancing Circuit
Balancing Circuit: (4.2^2)/33 =1/2W Assuming no FET resistance
Voltage Sense Lines In
Voltage Sense Line Inputs (Also Balancing Lines)
Cell Monitoring IC
LTC6804-2 -- Meant to be used by itself and IPB, IMB, ICMP, & IBIAS are used as address pins tied to GND
Slave Device
MISO
MOSI
CLK
CS_1
5V
GND

Figure 8 – Voltage Measuring Circuity

Shown in *Figure 8* is the schematic for the voltage sensing circuitry. As can be seen all 12 battery cells that are in series have a voltage sense line coming from them. They get sent to the LTC6804 “C” pins through a 100 ohm resistor. Then, if the IC senses that the voltage of any one of the cells is too high, it turns on the balancing circuitry with the corresponding “S” pin and the excess power is burned off through the 33 ohm load resistors (the other resistor and led attached to the FETs is just for visual verification that balancing is occurring). Also, the SPI communication lines (MOSI, MISO, CLK, and CS_1) are connected to the IC so that it can send the measured voltages to the central MCU.

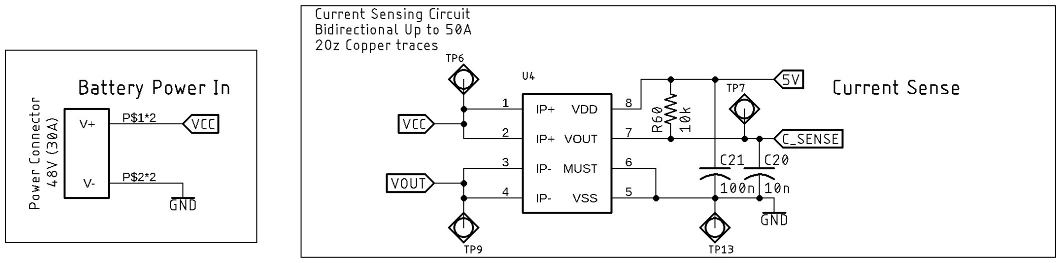


Figure 9 – Current Measuring Circuitry

Shown in *Figure 9* is the schematic for the current sensing circuitry. The battery power comes in from the connector on the left and is then immediately routed to the current monitor (MLX91210). This monitor measures the power following from IP+ to IP- and outputs a corresponding voltage (40mV / A) through the VOUT pin. The C_SENSE net (connected to the VOUT pin) is then routed to the MCU so that it can make sure that a safe amount of current pushed to, or pulled from, the battery.

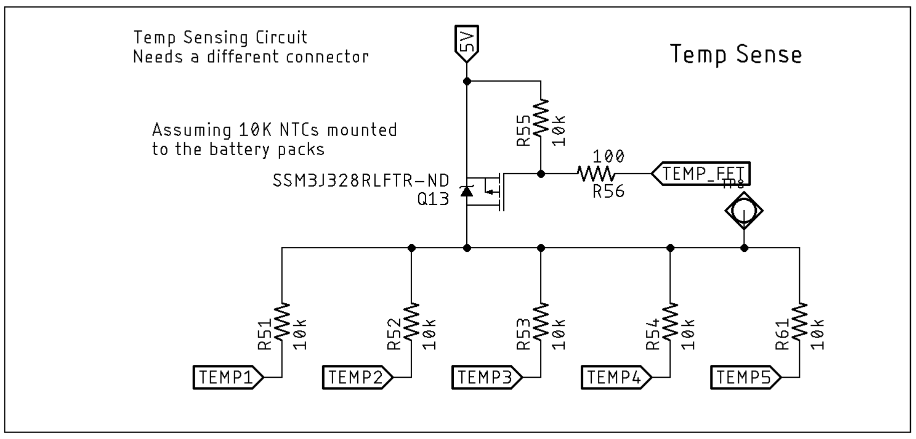


Figure 10 – Temperature Measuring Circuitry

Shown in *Figure 10* is the schematic for the temperature measuring circuitry. This circuitry consists of the in-series 10k resistors, that create the voltage divider with the 10k NTCs mounted on the battery, and the N-Channel MOSFET that controls whether or not this circuit has power. The TEMP1 through TEMP5 nets get routed back to the MCU so it can cut off power if the temperatures are no longer within safe operating parameters. The TEMP_FET net gets routed back to the MCU so that it can turn the temperature measurement circuitry on and off.

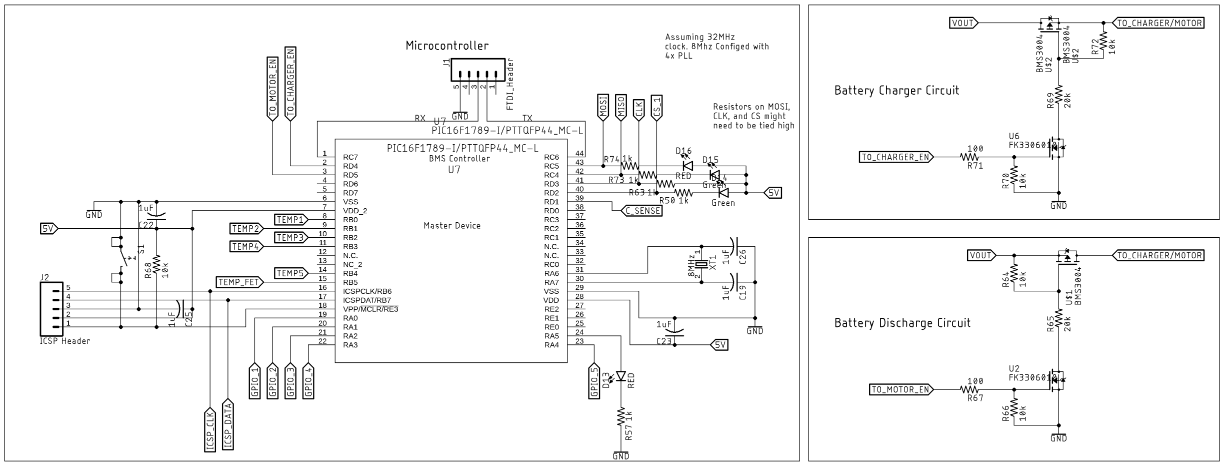


Figure 11 – MCU and Power Circuitry

Shown in *Figure 11* is the schematic for the MCU and the circuits that enable charging, and discharging, of the batteries. On the left of the MCU is the circuitry required to program it using a PIC KIT and a reset button. Attached to that circuitry are the I2C lines (ICSP_CLK, ICSP_DATA) that will be used to send the battery SOC to the embedded system. On the top is the circuitry that allows the MCU to communicate using UART (using a UART to USB adapter data can be sent to a computer and viewed with a terminal, allowing for an easier way to debug the software). Around the MCU inputs for temperature measuring (TEMP1 – TEMP5), current measuring (C_SENSE), the SPI communication lines for voltage measuring (MOSI, MISO, CLK, CS_1), and the general purpose pins (GPIO_ 1 – GPIO_5) which will be used for the kill switch, start switch, and charge switch can all be seen. The two pins that will be used to enable charging, and discharging, of the batteries can be seen on the top left of the MCU. To the right are the corresponding circuits that will actually connect the batteries to the charger or the other electronics on the bicycle. Each of these circuits consist of a power P-Channel MOSFET that has their gate tied to their source through a 10k resistor. Then, when the N-Channel MOSFETS have their gates pushed to 4.95V ($5 * \frac{10000}{10000+1000}$), the P-Channel gates are pulled to 2/3 of their source voltages using a simple voltage divider ($V_{Gate} = V_{Source} * [\frac{20000}{20000+10000}]$). This connects the P-Channel MOSFET's source and drain thus allowing power to flow to, or from, the battery.

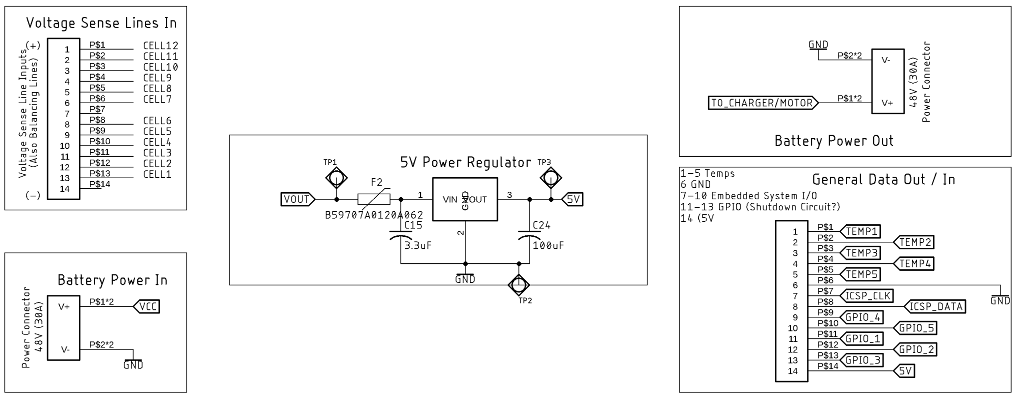


Figure 12 – 5V Regulator and Connectors

Shown in *Figure 12* is the schematic for the 5V switching regulator needed to power the battery management circuitry along with all of the connectors that are on the board. This 5V regulator (SRH05S05) is capable of taking a range of 9-72V and output a constant 5V for up to 500mA. This high input range means this can take the battery voltage directly, without needing to step it down. And it being able to output 500mA, means that it will be able to easily power all of the battery management electronics as well any external electronics that might need a 5V rail that are not currently accounted for. On the input of this regulator is a PTC (positive thermal coefficient) rated for the battery pack voltage. If a short circuit occurs somewhere and too much current is being pulled, the temperature of the PTC will rise and limit the current draw until the short is fixed (essentially acting as a resettable fuse).

3.8.3 Level 3 Battery Management System Software Theory of Operation [TRM]
As discussed above, the battery management system has three subsystems for measuring the various values needed to understand if the battery is operating safely or not. So, the software needs to be set up so that the MCU can interact with each of these subsystems in a timely manner. To do this, timers will be setup so that the MCU can take measurements at constant time intervals. The temperature and voltage measurements will be tracked by one timer, while the current measurement will be tracked by another. The current is tracked by itself because to accurately implement coulomb counting, for SOC calculations, the time interval between current measurements has to be precise. And the amount of time needed to take voltage and current measurements will be enough to make the coulomb counting SOC calculation inaccurate.

The battery management system also needs to be able to calculate the battery's SOC (state-of-charge) and send it to the embedded system whenever it is requested. To calculate a battery's SOC a few values need to be known: the most recent battery no-load voltage and the amount of charge it's given/received since the most recent no-load voltage measurement. The battery management takes the last known no load voltage and converts that to a percentage of battery charge, and then multiplies that by the total ampacity of the battery. This gives the amount of charge that the battery had left when the no-load voltage measurement was taken. Then, at a given time interval, the current flowing out of the battery is measured and subtracted from the total previous calculated. Finally, by dividing this value by the total battery ampacity, the battery's SOC can be calculated and sent to the embedded system. This technique for calculating battery SOC is known as coulomb counting and is the industry standard.

3.8.4 Level 3 Battery Management System Pseudo Code [TRM][ELW]

```
//Pseudo Code -- Tyler Matthews 11/16/2018
//Configuration Bits will be set in a config.h file

//Global Variables
*****************************************************************
***********************************************************

boolean batteries_are_safe = FALSE;
boolean battery_is_charging = FALSE;    //This will be changed to
true based on interrupts caused by the GPIO pins (charging button
pressed / connection made)
boolean battery_is_discharging = FALSE; //This will be changed to
true based on interrupts caused by the GPIO pins (start button
pressed / connection made)
float voltage_array[];
float temperature_array[];
float battery_SOC;
float max_ampacity = ; //will be intialized as the batteries
ampacity at full charge in Amps-time_between_measurements (ie:
amp-seconds or amp-milliseconds)
float current_value; //the most recent current value (in Amps)

//End Of Global Variables
*****************************************************************
****************************************************


//Main Function
*****************************************************************
**************************************************************

main(){
  system_initialize();
  initial_measurement();

  while(1){
      if(batteries_are_safe == TRUE){
        if(battery_is_discharging == True{
           start_discharging();

           if(Timer1 interrupt is thrown){
         measure_temperatures();
             read_voltages();
        }
```

```
          if(Timer2 interrupt is thrown){
        measure_current();
            calculate_soc();
      }

          if(embedded system requests data){
            send_soc();
          }

          battery_check();
        }
        else if(battery_is_charging == TRUE){
          start_charging();

          if(Timer1 interrupt is thrown){
        measure_temperatures();
            read_voltages();
        }

          if(Timer2 interrupt is thrown){
        measure_current();
            calculate_soc();
      }

          battery_check();
        }

      else if(batteries_are_safe == FALSE){
        //Set RD4 and RD5 to 0, turning off the
Charging/Discharging Circuits
        //Blink the LED connected to RA5
        //Do nothing else -- requiring a manual reset before the
system will run again
        //This ensures that the system is checked before trying
to run again
      }
    }
}

//End Of Main Function
*****************************************************************
*******************************************************
```

```
//Custom Functions
*****************************************************************
***********************************************************

//Initializing functions
*****************************************************************
**********************************
void system_initialize(){
  i2c_setup();
  spi_setup();
  uart_setup();
  timer_setup();
  adc_setup();
  pin_setup();
}

void inital_measurement(){
  read_voltages();
  inital_soc();
  measure_temperatures();
  initial_battery_check();
}

void inital_soc(){
  //Loop through the individual cell voltages array and add them
all together
  //Divide the value calculated above by the maximum pack voltage
(50.4 V) to get current SOC
  //Save this value as batter_SOC()
}

void initial_battery_check(){
  //Checks voltage and temperature arrays to ensure that all
measured values are within spec
  //If they are within spec, set batteries_are_safe equal to
TRUE, allowing the battery to charge / discharge
  //If they are not within spec, set batteries_are_safe equal to
FALSE, not allowing the battery to charge / discharge
}
//End Of Initializing Functions
*****************************************************************
***************************
```

```
//Charge / Discharge Functions
*****************************************************************
****************************
void start_discharging(){
  //Set RD5 high, causing the FET connecting the batteries to the
rest of the bike to close (allowing the batteries to discharge)
  //Set RD4 low, should already be low but this ensures that only
discharging circuit is on
}

void start_charging(){
  //Set RD4 high, causing the FET connecting the batteries to the
charger to close (allowing the batteries to charge)
  //Set RD5 low, should already be low but this ensures that only
charging circuit is on
}
//End Of Charge / Discharge Functions
*****************************************************************
*********************

//Measurement Functions
*****************************************************************
***********************************
void read_voltages(){
  //Send a command to the LTC6804 to intiate a measurement of the
battery cell voltages
  //Wait until SPI_interrupt is thrown and cell voltages can be
read from the SPI buffer
  //Put cell voltages into the array voltage_array[]
}

void measure_temperatures(){
  //Set RB5 high to enable temperature measurements
  //Read Temperatures using ADCs on RB0, RB1, RB2, RB3, and RB4
  //Put temperatures into the array temperature_array[]
}

void measure_currents(){
  //Measures the current value (in Amps) at the instant this
function is called
  //Samples ADC on RD1 and then converts that value from volts to
Amps based on the current sensor's relationship
}

void battery_check(){
  //Checks the voltage_array, temperature_array, and
current_value to ensure all measured values are within spec
  //If they are within spec, set batteries_are_safe = TRUE;
  //If they are not within spec, set batteries_are_safe = FALSE;
}
//End Of Measurement Functions
*****************************************************************
****************************


//State-Of-Charge Functions
*****************************************************************
*******************************
void calculate_soc(){
  //Calculates the batteries current SOC with the formula below:
  //battery_soc = battery_soc - (current_value / max_ampacity)
}

void send_SOC(){
  //Sends battery_soc to the embedded system using i2c
}
//End Of State-Of-Charge Functions
*****************************************************************
************************

//SETUP FUNCTIONS
*****************************************************************
*****************************************
i2c_setup(){
  //Setup I2C module to use RB6 & RB7 as data and clock lines,
respectively
  //Setup I2C module to be used as Slave (Embbeded system's MCU
is the master)
  //Setup I2C interrupt to be thrown when data is requested from
the embedded system MCU
}

spi_setup(){
  //Pull RD2 low -- this is the slave select pin and only one
device is ever being communicated with
  //Setup SPI module to use RC5 as MOSI, RC4 as MISO, and RD3 as
the clock
  //Setup SPI module to be active low (all connections have
pullup resistors)
```

```
  //Setup SPI module to be the master device (LTC6804-2 is the
slave device)
  //Setup SPI interrupt to be thrown when data is recieved
}

uart_setup(){
//NOTE: UART module is only for testing -- allows data to be sent
and viewed on PC terminal
//Setup UART module to use RC7 as RX and RC8 a TX
//Setup UART module to use a baud rate of 9600 (not much data
being sent -- this is more than sufficient)
}

timer_setup(){
  //Timer's will be used to measure battery data at certain time
intervals (ie: 0.1ms)
  //Timer1 will be used to poll battery voltage and temperatures
  //Timer2 will be used to poll current (current is used for SOC
calculation and will be on a different interval)
}

adc_setup(){
  //ADC's will be used to measure temperatures (RB0, RB1, RB2,
RB3, RB4), current (RD1), and the GPIO pins (RA0, RA1, RA2, RA3,
RA4)
  //NOTE: GPIO pins will be used on an "as needed basis" --
things like the start switch, kill switch, etc...
}

pin_setup(){
  //RC7, RD4, RB5, and RA4 will be set as digital outputs
  //RC7 and RD4 will be used to enable/disable battery charging
and discharging
  //RB5 enables temperature measuring (enabling and disabling
this saves power)
  //RA5 is an LED indicator for testing purposes
}
//END OF SETUP FUNCTIONS
*****************************************************************
**********************************

//End Of Custom Functions
*****************************************************************
****************************************************
```

Figure 13 – Battery Management System Pseudo Code

3.8.5 Level 3 Battery Management System Calculations [TRM]

Current Measurements:

The MLX91210 has a resolution of 40mV/A. So, for the maximum current expected to be pulled from the battery (30A) the MCU should get a reading of 1.2V (30A * 40mV/A = 1.2V).

Voltage Measurements:

The voltage measurements are done by the LTC6804, so there are no calculations for it. But, the MCU will constantly make sure that the individual cells do not exceed 4.2V or drop below 3.2V.

Passive Balancing Circuit:

The LTC6804 comes with the ability to do passive balancing of the battery. This is where the voltages of the individual cells that are in series are constantly kept the same by burning off excess in the cells with a higher voltage level. To do this, the LTC6804 sends a signal from one of its "S" pins to the gate of a P-Channel MOSFET, through a 3.3k resistor. When this MOSFET is completely turned on, the battery cell is shorted to itself through a 33ohm load resistor. Knowing that the maximum voltage of any 18650 cell is 4.2V, the calculation for power dissipated through that load resistor can be calculated as 0.5 W ($P = \frac{4.2^2}{33}$). This value is large enough to quickly balance the cells, but not so large as to destroy components from overheating (especially since the balancing is only ever on for very short periods of time).

Temperature Measurements:

The battery management system use 10K NTCs whose resistance are dictated by the following formula:

$$R = R_0 * e^{B*[\frac{1}{T} - \frac{1}{T_0}]}$$

where B is 3940, R0 is 10k, T0 is 30 C (298 Kelvin), and R is the NTC resistance at temperature T. Seeing as though the temperature of the batteries should not exceed 60 C (333 Kelvin), the minimum resistance the NTC should have is 2494 ohms ($R = 10000 * e^{3940*[\frac{1}{333} - \frac{1}{298}]}$). From this, the voltage measured by the MCU that indicates battery issues can be calculated via a simple voltage divider. By running the calculations, it can be seen that any value less than 1V ($V_{Failure} = 5 * [\frac{2949}{2949+ 10000}]$) indicates battery issues. When this problem is sensed the battery discharging switch will open and stop operation.

Battery Charging and Discharging Circuits:

The circuits the enable charging and discharging of the batteries are exactly the same, except the power P-Channel MOSFET is flipped. As such, the calculations for both of the circuits are the same.

The P-Channel MOSFET has its gate tied to its source with a 10K resistor, when closer the voltage at the gate and source of the MOSFET get to each other the closer to being “off” the MOSFET is. Since the MOSFET has an internal capacitance (CISS) that is roughly 13400pF, that resistor creates an RC time constant with it. This time constant is 0.134s ($\tau = R * C$), which means it would take roughly 1/10 of a seconds for the MOSFET to go from completely on to completely off. This value might seem high but as the MOSFET transitions from one state to another, the impedance between its source and drain changes. So, although it might take 1/10 of a second to completely cutoff current flow, it'll take significantly less to start limiting it. When the N-Channel MOSFET turns on, grounding the gate of the P-Channel through a 20K resistor, the voltage at the gate of the P-Channel is set to 2/3 of the source voltage through a voltage divider ($V_{Gate} = V_{Source} * [\frac{20000}{20000+10000}]$). This ensures that the gate is sufficiently biased to turn on the P-Channel MOSFET completely, while not destroying the MOSFET by having an overly large voltage difference between the source and gate.

The N-Channel MOSET works by pulling its gate high than its source. So, when the MCU puts 0V on its gate, through the 100 ohm resistor, the value at the gate is essentially the same as that on the source (0V). When the N-Channel needs to turn on, the MCU puts 5V on the gate, through the 100 ohm resistor, and the voltage divider network sets the gate to 4.95V ($V_{Gate} = 5 * [\frac{10000}{10000+100}]$). This is more than sufficient to turn on the MOSFET completely.

State of Charge Estimation:

To understand how much capacity a battery pack has left, it is necessary to estimate its SOC, or State-Of-Charge. To estimate this, the following formula is used:

$$SOC_{\%} = \left(\frac{\left(\left(\left(\frac{V_{NL}}{V_{Total}} \right) * I_{Total} \right) - \int I_t \right)}{I_{Total}} \right) * 100$$

where $SOC_{\%}$ is the percentage of battery life left, V_{NL} is the last no-load voltage measurement that was taken, V_{Total} is the battery voltage at full charge (50.4V), I_{Total} is the total ampacity of the battery pack (12500mAH), and I_t is the battery current at time "t". By taking the no-load voltage as a fraction of the total battery voltage and then multiplying it by the total pack ampacity, it is possible to find the ampacity the pack had left when the no-load measurement was taken. By then taking the integral of the current over a set period of time, one can calculate what percentage of the battery has been used during that period of time. Finally, by subtracting the amount used during the period of time since no-load measurement was taken from the no-load amount that was measured, the ampacity of the pack at any time “t” can be solved for. If this value is divided by the total ampacity of the pack, and multiplied by 100, then the percentage of battery life left is calculated.

3.8.6 Level 3 Battery Management System Parts List [TRM]

Qty	Value	Package	Parts	Description
1	TACTILE-SWITCH	TACTILE-SWITCH	S1	PIC Reset Switch
17	1k	R0805	R38, R39, R40, R41, R42, R43, R44, R45, R46, R47, R48, R49, R50, R57, R63, R73, R74	Resistors
8	1uF	C0805	C16, C17, C18, C19, C22, C23, C25, C26	Capacitors
1	SSM3J328RLFTR-NO	SOT23-3	Q13	P-Channel MOSFET
12	3.3k	R0805	R4, R6, R9, R12, R15, R18, R21, R24, R27, R30, R33, R36	Resistors
1	3.3uF	C0805	C15	Capacitor
1	8MHz	HC49UP	XT1	Oscillator
12	10k	R0805	R51, R52, R53, R54, R55, R60, R61, R64, R66, R68, R70, R72	Resistors
1	10n	C0805	C20	Capacitor
12	10nF	C0805	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12	Capacitors
2	20k	R0805	R65, R69	Resistors
12	33	R0805	R3, R7, R10, R13, R16, R19, R22, R25, R28, R31, R34, R37	Resistors
16	100	R0805	R1, R2, R5, R8, R11, R14, R17, R20, R23, R26, R29, R32, R35, R56, R67, R71	Resistors
1	100n	C0805	C21	Capacitor
1	100nF	C0805	C13	Capacitor
1	100uF	C0805	C24	Capacitor
1	B59707A0120A062	1210	F2	PTC

2	BMS3004	TO218V	U$1, U$2	Power P-Channel MOSFETS
12	DMP2305U	SOT-23	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12	P-Channel MOSFET
2	FK3306010L	SSSMINI3-F2-B	U2, U6	N-Channel MOSFET
1	FTDI_Header	1X05	J1	FTDI Header
14	Green	LED-0603	D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D14, D15	LEDs
1	ICSP Header	1X05	J2	ICSP Header
1	LTC6804	SSOP48	U1	Battery Monitoring IC
1	MLX91210	SOIC-8	U4	Current Sensor
2	MOLEX_42840-2224	MOLEX_42820-2224	X2, X4	Connector
2	MOLEX_105314-1114	MOLEX_105314-1114	X1, X5	Connector
1	PIC16F1789-I/PTTQFP44_MC-L	TQFP44_MC-L	U7	MCU
2	RED	LED-0603	D13, D16	LEDs
1	SRH05S05	SIP3	U5	5V Regulator
8	TP5015	TP5015	TP1, TP2, TP3, TP6, TP7, TP8, TP9, TP13	Test Points
5	NXRT15XV103FA1B040	Through Hole	Not On Schematics	10k NTC

Table 18 – Battery Management System Parts List

3.9.1 Level 3 Embedded System Hardware Theory of Operation [NMD]

The level 3 hardware diagram is an actual schematic of the system itself. While the schematic relatively close to the final product with the possibility of changes only occurring after testing has proved that changes to the design are needed. The final schematic will be split into six main parts each serving its own function in the final product.

3.9.2 Level 3 Embedded System Schematic Diagrams [NMD]

This first main portion of the schematic shows how each component on the board is powered. There are three main power rails on this board that takes the 48-Volt battery power as an input and delivers 3.3-Volts, 5-Volts, and 12-Volts to the appropriate portions of the circuit. The schematic for this circuit is shown below in Figure X and utilizes a simple switching regulator to provide appropriate voltages. Only 1 circuit is shown in Figure 14 but the each voltage rail is very similar to this circuit. The only difference being that the switching regulator device is a different model that creates another voltage value.

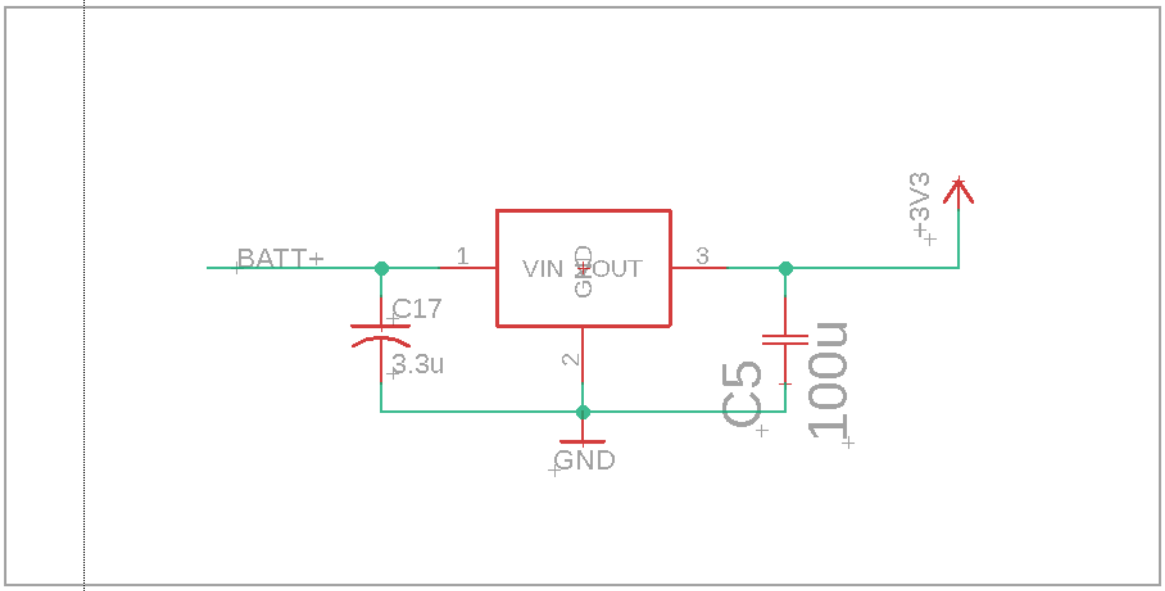


Figure 14 – DC Voltage Regulator Circuit

After power can now be properly distributed throughout the system, the next main part of the system to be discussed is the main controller. The design block for the main controller can be seen in Figure 15 below and shows that the main microcontroller is a dsPIC33F that will be running at a speed of 40MIPS. Certain associated passive devices can be seen in this block as well that the microcontroller requires to run properly. Finally the main purpose of the microcontroller is to control the functionality of the entire system. Nets for each input and output signal can be seen that will send PWM signals to the motor driver IC, read the values of the Hall effect sensors of the motor, and read the currents in each phase of the motor. Another aspect of these signals is the communication signals that are going to be used to read heart rate, throttle input, and state of charge from the battery management system along with fault data. An I2C bus is implemented for communication with each of these peripherals because it allows for quick communication with a large number of devices while only requiring two lines for communication. This will later allow for the addition of a display, and if time permits, a Wi-Fi module which will then allow for the creation of a simple website.

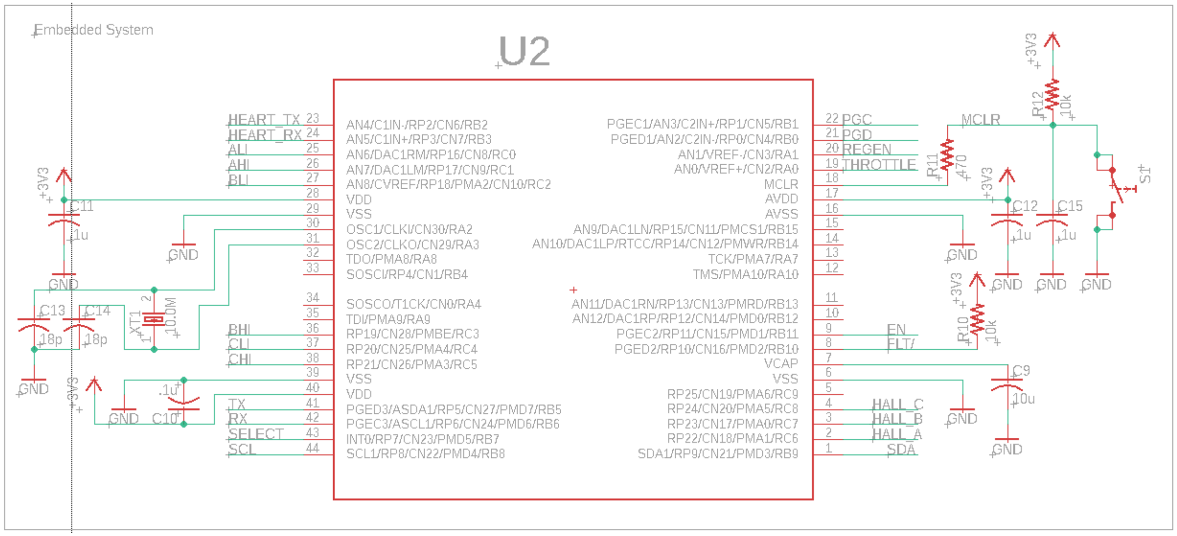


Figure 15 – Embedded System Main Controller

The last main portion of the embedded system is the motor driver IC and 3-Phase H-bridge design. The design block can be seen below in Figure 16 and shows both the driver IC which is used to help drive the gates of the large power MOSFETs in the bridge and then the 3-Phase half bridge. The driver IC utilizes three bootstrap capacitors that are charged during the off time of each top FET in the bridge which help drive the gate of the FET high when the source is floating. This allows for quick and reliable switching of the FET as needed.

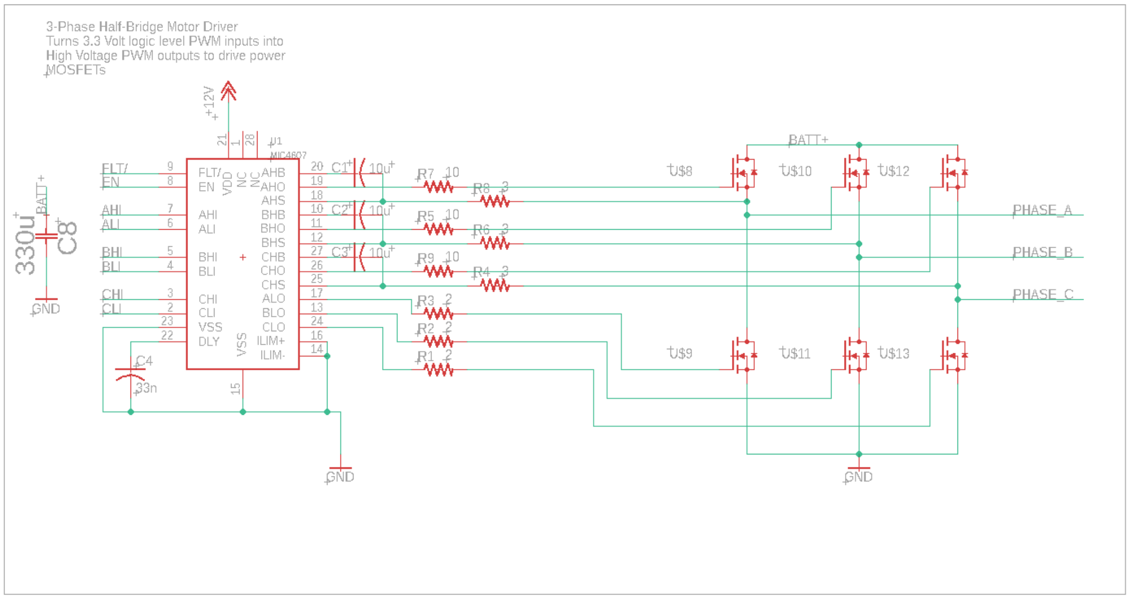


Figure 16 – 3 Phase Half Bridge and Driver IC

The last three blocks of the schematic shown are simple blocks depicting a series of test points that will be placed on the board in Figure 17 below. The purpose of these test points is to help with testing and debugging the final prototype. The following block depicts a series of LEDs that will be used for indication of the signals being transmitted on the board, such as power, power FET gates, Hall sensors, and communications among others. The very last design block shown is the connectors that will transfer signals from this board to the BMS, peripheral boards and finally the motor.

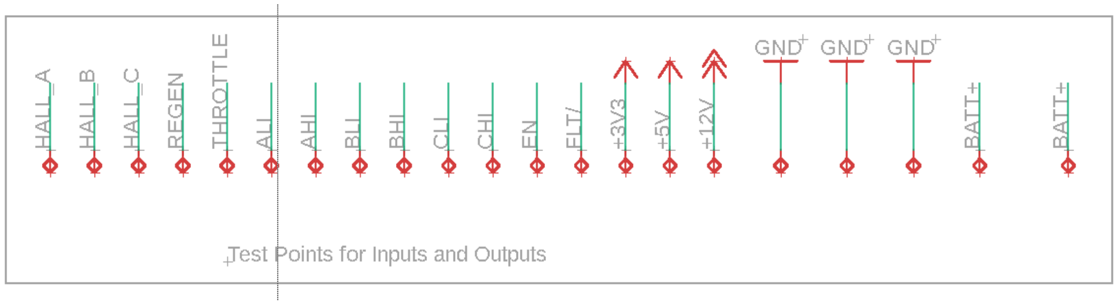


Figure 17 – Test Points and Signals

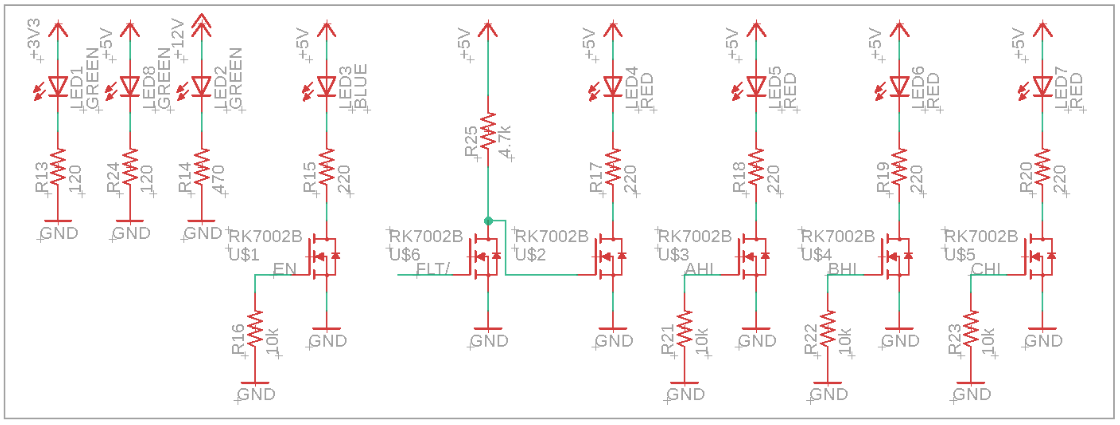


Figure 18 – Signal Indication LEDs

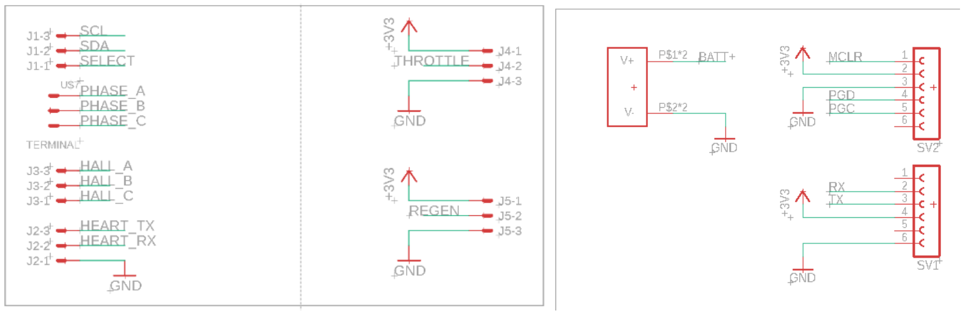


Figure 19 – Connector Circuits

The next step taken after creating these schematics is to layout the individual parts onto a PCB to have printed. The first revision of this board can be seen in Figure 20 below and depicts the PCB that will be created to control the bicycle. The board itself is subject to change however it suffices as a good model for future board layouts.

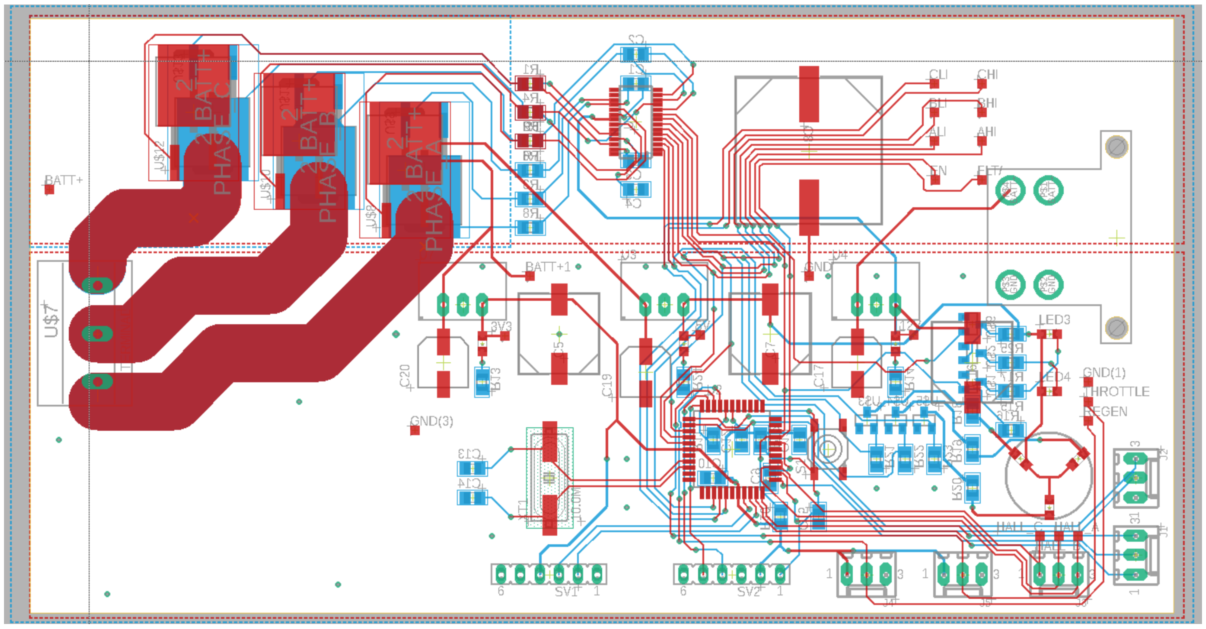


Figure 20 – First Revision of the Embedded System Board Layout

3.9.3 Level 3 Embedded System Software Theory of Operation [NMD]

After the hardware has been simulated and schematics have been made the next step is to write the actual code that will control all the hardware shown above. Seen below is a simplified version of pseudo code that will be implemented on the dsPIC. The code running on the dsPIC will have similar functionality to the code running in MATLAB however it will also have to keep track of all sensor data and communicate frequently with other devices on the bike. Another key difference will be how the sine and cosine of the motor angle will be calculated. In MATLAB the sine and cosine are calculated directly using Taylor series expansion, while this method can give a very accurate value of these two functions it is far to computationally demanding to occur in real time. Therefore lookup tables will be implemented in software to help extrapolate values for these functions that are not as precise but done much more quickly. All pseudo code can be seen below in Figure 21.

3.9.4 Level 3 Embedded System Pseudo Code [NMD]

```
/* Pseudo code for motor controller
* Created by Nick Darash for use by DT18
* Cycle Assist
*/

////////////////////////
//     Directives     //
////////////////////////
#define Q1
#define Q2
#define Q3
#define Q4
#define Q5
#define Q6

enum {false,true};
enum {OFF,ON};

//////////////////////
// Global Variables //
//////////////////////
int theta;           // Position of motor in radians
int w;               // Speed of motor in radians per second
int delta;           // Time since previous hall effect signal
int time;            // Time since start of program
int delta_A;         // Value of HALL A one step prior
int delta_B;         // Value of HALL B one step prior
int delta_C;         // Value of HALL C one step prior
int Ia,Ib,Ic;        // Values of Phase A, B, and C currents

void main(){
    int integral_Q;
    int integral_D;
    int Id,Iq;          // Direct and Quadrature current values
    int U_q,U_d;        // Output after PI controller
    int command;        // Value of torque command from user
    oscSetup();         // Set up PIC to work at 40 MIPS

    /*
     * Set up interrupts for appropriate peripherals
     */
    interruptSetup();
    /*
     * Set up UART peripheral for debugging purposes
     */
    UARTSetup(baudrate);

    /*
     * Set up I2C bus for add on stuff later and to communicate
     * with battery management system
     */
    I2CSetup(baudrate);

    /*
     * Set up timers for switching and for communication purposes
     */
    timerSetup();

    /*
   * Set up out put compare module for Trapezoid control
     */
    OCSetup();
    while(1){
        if(ADC done triggered a flag){
            update();   // Update all 3 phase current values
            U_q = PI(&integral_Q, command, &Iq);
            U_d = PI(&integral_D, command, &Id);
        }
        if()
    }
}

////////////////////////
//       Timers       //
////////////////////////
void timerSetup(){
    // Timer 1 Timer 2 will be used for switching and
communication
    // respectively Timer 3 will be used for ms_delay function
and Timer 4
    // will be used for us_delay function
}
/*
* Timer 1 ISR main timer for switching
* this timer will be used to switch gates of the power stage
*/
void T1ISR(){
    // Ideally set up for 20kHz switching frequency
}
/*
* Timer 2 ISR will be used to initiate comms with other
peripherals
* ie. I2C communication for heart rate, display, and BMS
*/
void T2ISR(){
}

/////////////////////////
// External Interruprs //
/////////////////////////
/*
* External interrupts will be used to read the 3 hall effect
sensors
*/
void int0ISR(){
    // Read rising and falling edge of Hall A
}
void int1ISR(){
    // Read rising and falling edge of Hall B
}
void int2ISR(){
    // Read rising and falling edge of Hall C
}
///////////////////
//      I2C      //
///////////////////
void I2CSetup(int baudrate){
    // setup at 10020kHz
    // master device
    // 7-bit addressing
}
void I2CStart(){
    // send start condition
}
void I2Csend(unsigned char data){
    // send byte of data
}
unsigned char I2Creceive(){
    // return data from I2CRXREG
}
void I2CISR(){
    // set flag when I2CRXbuffer is full to pull all values
```

```
}

//////////////////
//     UART     //
//////////////////
void UARTSetup(int baudrate){
    // UART is used for debugging purposes only
}
void UARTsend(unsigned char data){
}

////////////////////
//      ADCs      //
////////////////////
void ADCSetup(){
    // set up all four ADCs for 3 current sensors and for
throttle input
    // Should be enough
}

void ADCread(){
    // read all four ADCs at once
}

void ADCISR(){
    // interrupt when ADCs have finished sampling
}

////////////////////
//   PI Control   //
////////////////////
int PI(int *integral, int command, int I){
    int error = command - I;
    *integral = (*integral + error*(time since last calc))
    return((*integral << 9) + (error << 9));
}
```

Figure 21 – Embedded System Pseudo Code

3.9.5 Level 3 Embedded System Simulation [NMD][RKA]

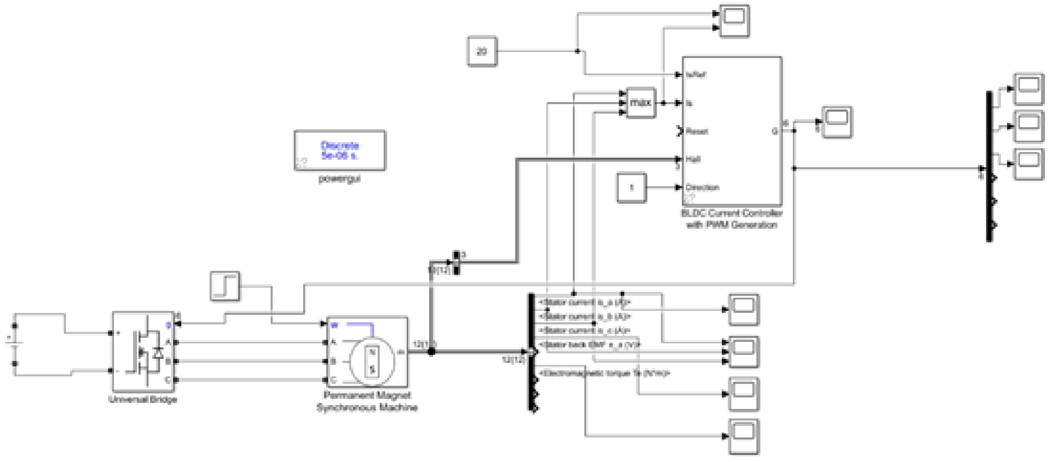
Motor Controller

Motor control can be accomplished in many different ways depending on the needs of the motor and the type of motor being used. When creating the circuit for a motor controller used to run an e-bike many variables had to be taken into account. The purpose of the bike is to change the resistance felt by the user while riding to force the user into a specific training zone. In order to accomplish this, the controller will work best when it is able to command torque from the motor instead of speed. This way the bike can both assist, and resist, the user with a constant force. While many different types of control exists for both speed and torque based control, only the two that were studied the most will be discussed.

First Simulation

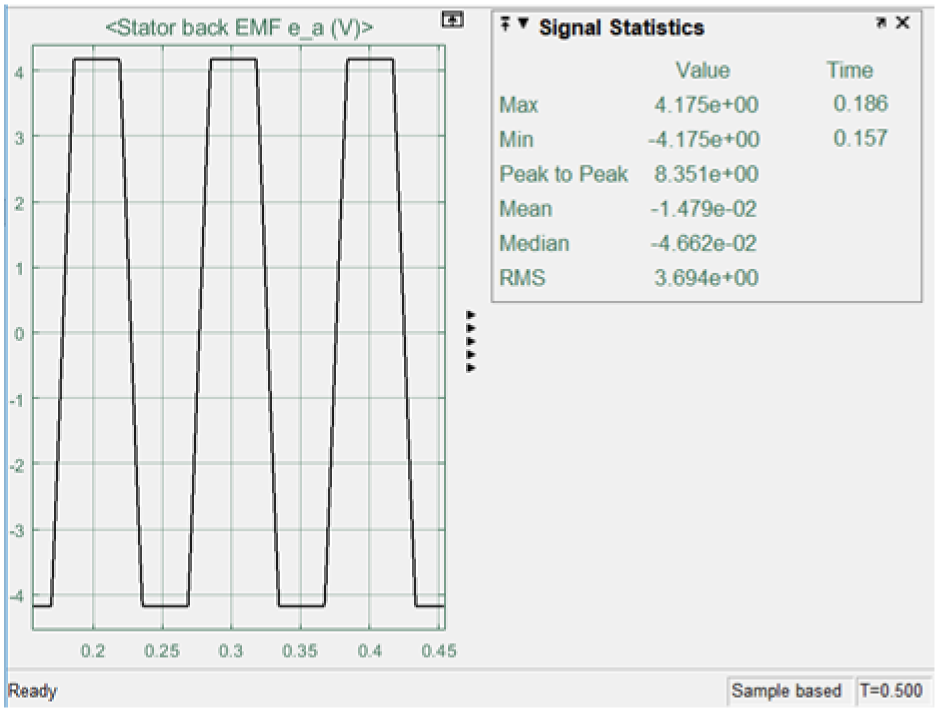
Using Simulink, simulations representing the three-phase BLDC motor operation and control can be implemented. The first simulation can be seen below in Figure 22. This simulation utilizes multiple blocks from the Simscape Power Systems library which is a downloadable add-on to the Simulink program. The block labeled “Permanent Magnet Synchronous Machine” is used to model the three-phase synchronous BLDC motor. Trapezoidal mode is selected which means this machine will operate assuming the flux established by the permanent magnets on the rotor produces three (one per phase) trapezoidal back emf waveforms. Figure 23 below shows the trapezoidal back emf for a single phase (phase A) of the motor spinning at 5.3 rad/s or about 50rpm. The total amplitude of the signal is 8.35V, and when rotating the motor in the lab at a rate of about

50rpm the total amplitude of the signal was 8.20V – a small enough difference to assume the back emf model is reliable.



20
max
IsRef
Is
Reset
Hall
Direction
G
1
BLDC Current Controller with PWM Generation
Discrete 5e-06 s.
powergui
Universal Bridge
Permanent Magnet Synchronous Machine
<Stator current is_a (A)>
<Stator current is_b (A)>
<Stator current is_c (A)>
<Stator back EMF e_a (V)>
<Electromagnetic torque Te (N*m)>

Figure 22 – First BLDC Motor Simulation in Simulink



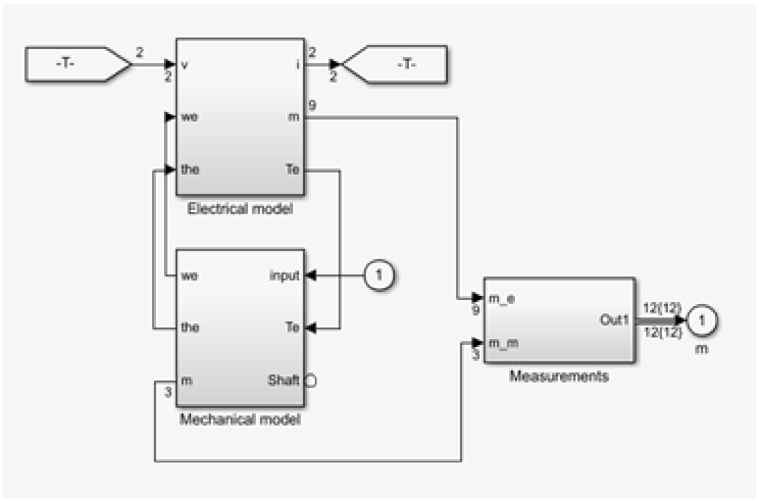
Signal Statistics

	Value	Time
Max	4.175e+00	0.186
Min	-4.175e+00	0.157
Peak to Peak	8.351e+00	
Mean	-1.479e-02	
Median	-4.662e-02	
RMS	3.694e+00	

Ready Sample based T=0.500

Figure 23 – Trapezoidal Back-EMF Waveform at 50 RPM

Both the electrical and mechanical parts of the machine are governed by second-order state-space equations given in the rotor reference frame (q and d axis) and quantities are referred to the stator for simplicity. Looking under the blocks mask two other blocks can be seen; one corresponding to mechanical rotation and the other corresponding to the electrical operation. The second-order state-space equations and a brief description of each variable along with the electrical and mechanical block models are shown in Figure 24. There are 12 different parameters represented at the output of the motor block: each phase's current (i_a, i_b, and i_c [A]), each phase's trapezoidal back emf (e_a, e_b, and e_c [V]), electromagnetic Torque (T_e [N*m]), the three hall effect sensors (h_a, h_b, and h_c [unitless]), the corresponding rotor angle (θ[rad]) and angular speed (w_m [rad/s]).

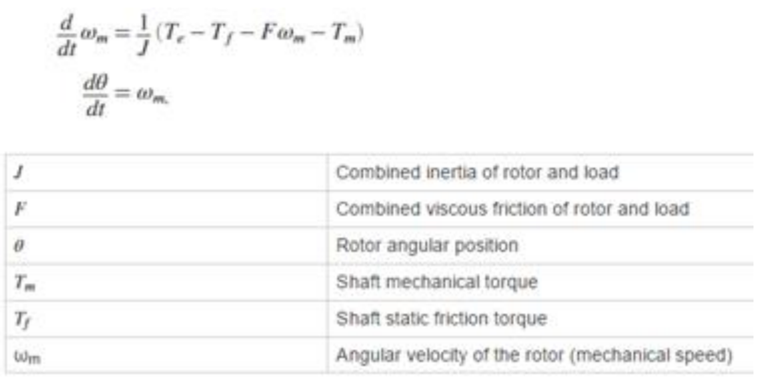


$$\frac{d}{dt}i_d = \frac{1}{L_d}v_d - \frac{R}{L_d}i_d + \frac{L_q}{L_d}p\omega_m i_q$$

$$\frac{d}{dt}i_q = \frac{1}{L_q}v_q - \frac{R}{L_q}i_q - \frac{L_d}{L_q}p\omega_m i_d - \frac{\lambda p\omega_m}{L_q}$$

$$T_e = 1.5p[\lambda i_q + (L_d - L_q)i_d i_q]$$

L_q, L_d	q and d axis inductances
R	Resistance of the stator windings
i_q, i_d	q and d axis currents
v_q, v_d	q and d axis voltages
ω_m	Angular velocity of the rotor
λ	Amplitude of the flux induced by the permanent magnets of the rotor in the stator phases
p	Number of pole pairs
T_e	Electromagnetic torque



$$\frac{d}{dt}\omega_m = \frac{1}{J}(T_e - T_f - F\omega_m - T_m)$$

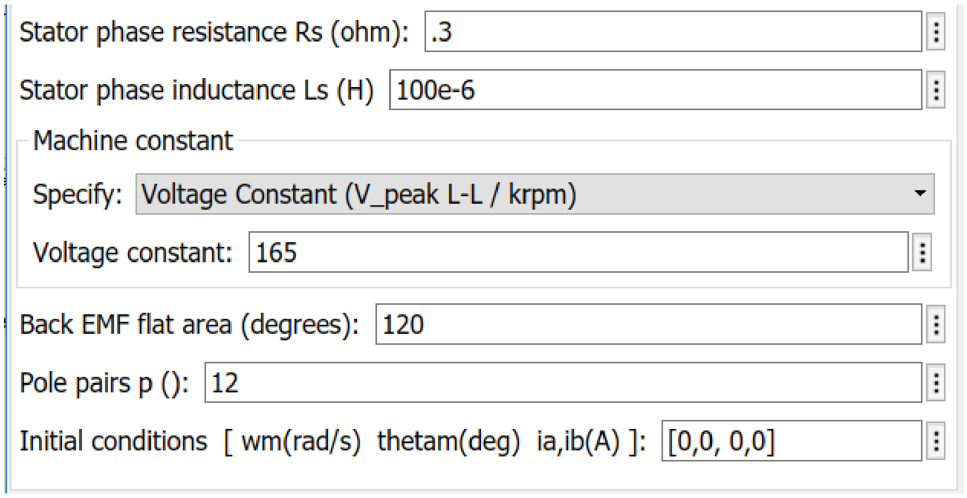
$$\frac{d\theta}{dt} = \omega_{m.}$$

J	Combined inertia of rotor and load
F	Combined viscous friction of rotor and load
θ	Rotor angular position
T_m	Shaft mechanical torque
T_f	Shaft static friction torque
ω_m	Angular velocity of the rotor (mechanical speed)

Figure 24 – Electrical and Mechanical Model and Their Governing Equations

The motor also takes a mechanical input - in this case it is mechanical speed applied to the motor shaft in rad/s and is applied using a stepped input starting from 0 and stepping up to a final value. Now that the basic operation of this block is understood selection of the motor parameters to reflect the actual motor are selected to create an accurate model. These parameter values include the per phase stator resistance and inductance, the machine constant or voltage constant defined as the line-to-line voltage per 1000rpm, the back emf flat area (left at the default 120 degrees), the number of pole pairs of the rotor magnets, and the initial conditions of the angular rotation, position, and phase currents which are all set to 0. The motor was provided by the university and a data sheet cannot be found for this motor, so measurements of the stator phase inductance, resistance, and voltage constant were made using an LCR meter and volt meter.

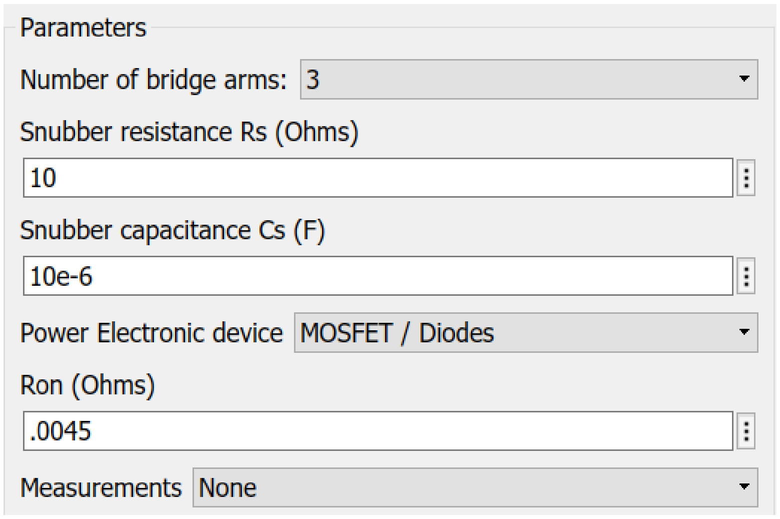
The number of pole pairs was determined by observing a single hall effect sensor on the oscilloscope and mechanically rotating the motor one rotation. The hall sensor pulsed 12 times during this single rotation, so the motor has 12 pole pairs. To determine the voltage constant the peak line-to-line voltage is monitored as the motor is spun one mechanical rotation while being timed. Converting to krpm and dividing the peak line-to-line voltage by this value gives the voltage constant. All motor parameters used can be seen below in Figure 25. Setting the sample time parameter to -1 allows the block to inherit the sample time specified in the powergui block (5µs in this application). The powergui block must be implemented in any Simulink model that uses Simscape Power Systems blocks as it stores the equivalent Simulink circuits that represent the state-space equations.



Stator phase resistance Rs (ohm):	.3
Stator phase inductance Ls (H)	100e-6
Machine constant	
Specify:	Voltage Constant (V_peak L-L / krpm)
Voltage constant:	165
Back EMF flat area (degrees):	120
Pole pairs p ():	12
Initial conditions [wm(rad/s) thetam(deg) ia,ib(A)]:	[0,0, 0,0]

Figure 25 – Motor Parameters

Each of the three phases of the motor are fed by a half bridge of MOSFETs – one connected to the high voltage rail and one connected to the low voltage rail (6 MOSFETs in total). Each phase will either provide current to the motor or return current from the motor, dependent upon whether the high or low MOSFET is conducting, respectfully. Another Simscape Power System block is utilized to simulate the operation of this three phase converter and is labeled above in Figure 22 as “Universal Bridge”. As shown, we connect a 48V (provided battery voltage) source across the half bridge circuit. This block requires the following parameter inputs: Number of bridge arms (3), snubber resistance and capacitance, power electronic device (MOSFET/Diodes), and the internal resistance of the power electronic devices. The value of internal resistance is taken directly from the data sheet for the power MOSFET devices (4.5 mΩ). The snubber resistance and capacitance are connected in series and then connected in parallel to each MOSFET gate. The snubber capacitor is the same as the bootstrap capacitor which was calculated to be around 10uF. The chosen values for all input parameters of this block are shown below in Figure 26. The block also takes 6 separate pulse values. The duration of these pulses will determine the current flowing through each phase of the half bridge to the motor.


Parameters

Number of bridge arms: 3

Snubber resistance Rs (Ohms): 10

Snubber capacitance Cs (F): 10e-6

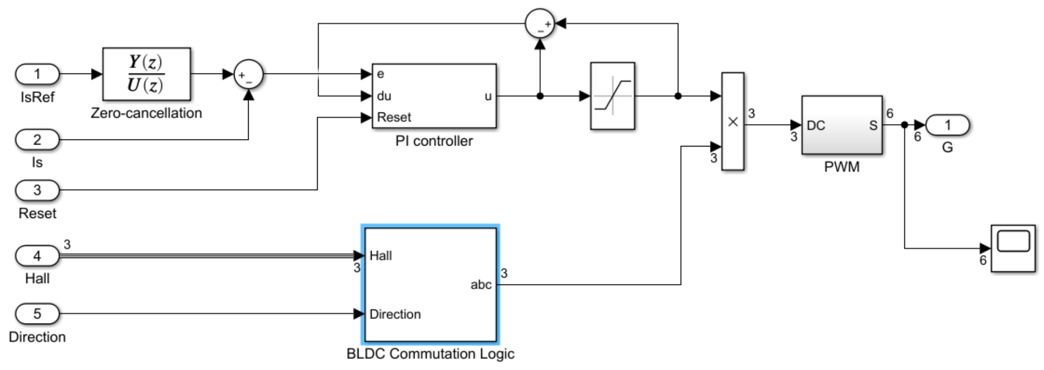
Power Electronic device: MOSFET / Diodes

Ron (Ohms): .0045

Measurements: None

Figure 26 – Universal Bridge Parameters

To determine the duty cycle for the pulse widths sent to each gate we use a Simulink block named “BLDC Current Controller With PWM Generation” which can be seen in Figure 22 above. This block takes a reference current, the maximum value of the phase current, a reset, three hall sensors from the motor, and a direction. The reset input is left alone as it is not needed here, and the direction input is sent a constant value of 1 which denotes clockwise rotation. The block uses an internal PI controller and summation blocks to calculate the error between the reference current and actual phase current. Another block uses the hall effect sensors and direction input to provide the commutation logic for switching. Using the outputs of these two blocks, a pulse width modulation block implements the actual duty cycle and sends the pulses to the respective gate. The equation used to determine the duty cycle with short explanation of variables, the transfer function used by the PI controller (the Zero-cancellation block) with short explanation of variables, and the model used to implement its operation is shown below in Figure 27. The calculated duty cycle is multiplied by the commutation signals to give three output values. Although for certain implementations, only three signals are needed as the current is regulated by determining the duty cycle for only the high or low side MOSFETs, it is more accurate for this particular model (and in general) to control both high and low side MOSFETs. The block shown in the model below in Figure 27 labeled "PWM" takes the three signals representing three duty cycles and creates 6 separate output duty cycles which are then used to control each MOSFET separately.



$$D = \left(K_p + K_i \frac{T_s z}{z - 1}\right)(I_{s\_ref} - I_s)$$

$$G_{ZC}(z) = \frac{\frac{T_s K_i}{K_p}}{z + \left(\frac{T_s - \frac{K_p}{K_i}}{\frac{K_p}{K_i}} \right)}.$$

- D is the duty cycle.
- K_p is the proportional gain.
- K_i is the integral gain.
- T_s is the time period.
- I_{s_ref} is the reference current.
- I_s is the measured current.
- G_{zc} is the zero cancellation polynomial.

Figure 27 – Model for Current Controller with PWM and its Governing Equations

Trapezoidal Control

The first type of motor control that was considered is a form of speed control known as Trapezoidal Control. This form of control algorithm utilizes PWM signals with a variable duty cycle that sets the voltage of all three phases. These PWM signals create a 3-phase trapezoid shaped wave form as the motor revolves**.** The duty cycle sets the voltage of the phases, but deciding which phases are supposed to be switching to high voltage and which phases switch to ground is determined by the internal Hall effect sensors of the motor. The controller utilizes three half bridges to switch each phase high or low. This design can be seen in Figure 28 below where q1 through q6 are the inputs to each gate of each power FET. PHASE_A, PHASE_B, and PHASE_C output the power to the permanent magnet machine.

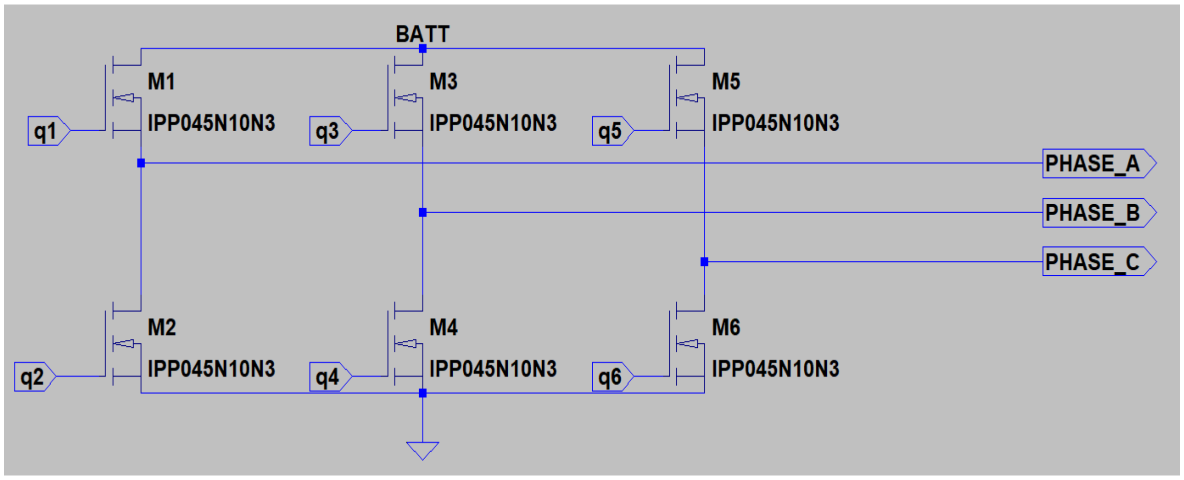


Figure 28 – 3 Half Bridges

The depiction of the three half bridges in Figure 28 show how each phase is either connected to the positive terminal of the battery or ground. When controlled by a trapezoid wave we get wave forms similar to those in Figure 29 and 30. Figure 29 shows the overall commutation of each phase while Figure 30 shows a close up of each phase and a trapezoid wave with a 60 percent duty cycle.

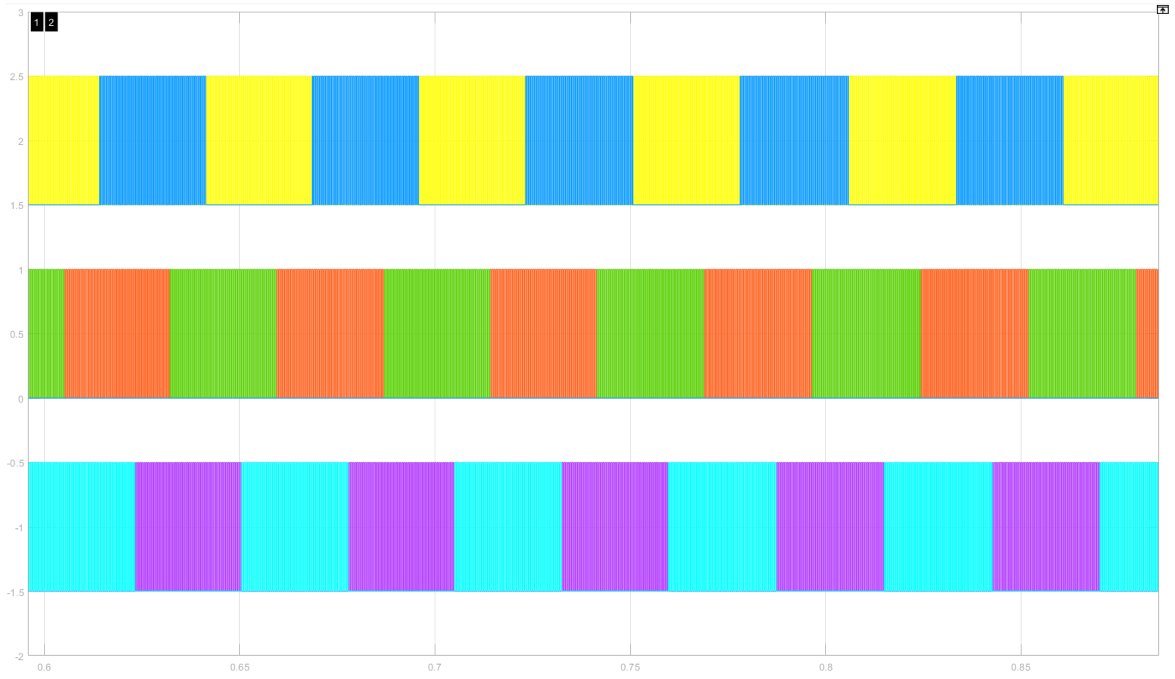


Figure 29 – 3 Phase Commutation

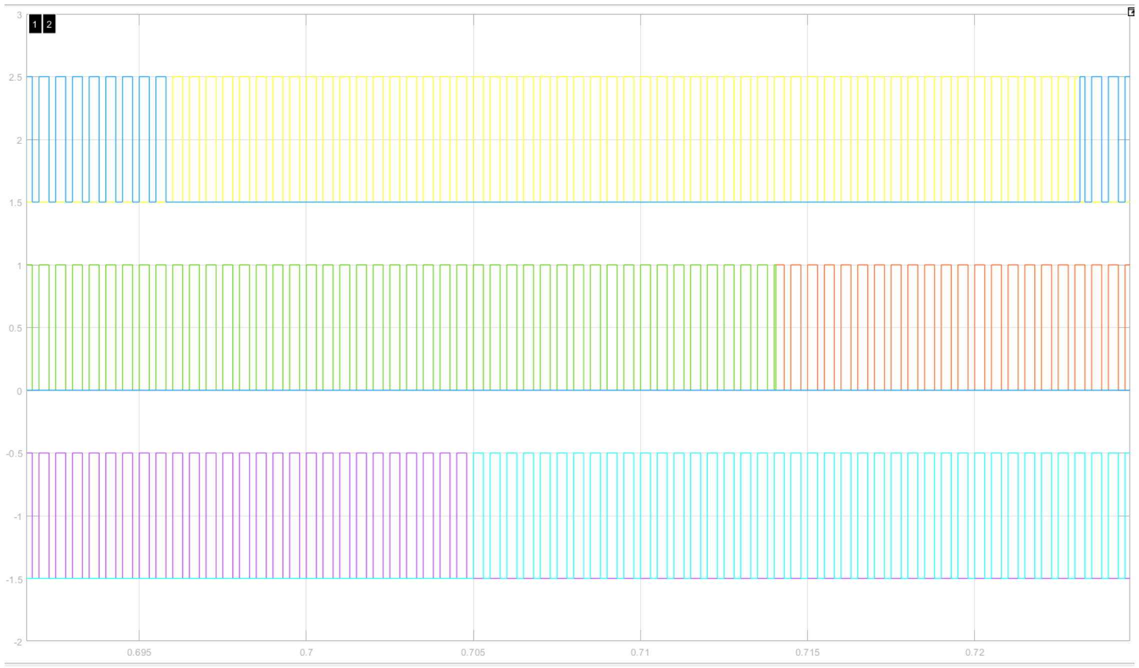


Figure 30 – 60 Percent Duty Cycle

These waveforms were simulated using Simulink with the Simscape Power Systems add on which is able to simulate a BLDC motor as described previously. In Figure 29 the blue and yellow waveforms represent gates q1 and q2, which switches between pulling phase A high or low. Similarly phase B is represented by the red and green waveforms and phase C is represented by the teal and purple waveforms. Figure 30 takes a closer look at these waveforms to show that 60 percent of the time they are either pulled high or low and 40 percent of the time they are left floating. This allows the voltage across each phase to be adjusted by a program and command a certain speed from the motor.

Figure 31 contains the MATLAB code that is responsible for setting these inputs to the 6 gates. The MATLAB function takes the inputs of each Hall Effect sensor, and each phase current from the motor.

```
function [X,Y,Z]=control(Ia,Ib,Ic,HALL_A,HALL_B,HALL_C,command)
global q1
global q2
global q3
global q4
global q5
global q6
global counter

counter = counter + 1;
if counter <= command
    if HALL_A && ~HALL_B && HALL_C
        q1 = 1;
        q2 = 0;
        q3 = 0;
        q4 = 1;
        q5 = 1;
        q6 = 0;

    elseif HALL_A && ~HALL_B && ~HALL_C
        q1 = 1;
        q2 = 0;
        q3 = 0;
        q4 = 1;
        q5 = 0;
        q6 = 1;

    elseif HALL_A && HALL_B && ~HALL_C
        q1 = 1;
        q2 = 0;
        q3 = 1;
        q4 = 0;
        q5 = 0;
        q6 = 1;

    elseif ~HALL_A && HALL_B && ~HALL_C
        q1 = 0;
        q2 = 1;
        q3 = 1;
        q4 = 0;
        q5 = 0;
        q6 = 1;

    elseif ~HALL_A && HALL_B && HALL_C
        q1 = 0;
        q2 = 1;
        q3 = 1;
        q4 = 0;
        q5 = 1;
        q6 = 0;

    elseif ~HALL_A && ~HALL_B && HALL_C
        q1 = 0;
        q2 = 1;
        q3 = 0;
        q4 = 1;
        q5 = 1;
        q6 = 0;
    end
else
    q1 = 0;
    q2 = 0;
    q3 = 0;
    q4 = 0;
    q5 = 0;
    q6 = 0;
end
if counter >= 100
    counter = 0;
end
```

Figure 31 – Trapezoidal Control Matlab Code

The MATLAB code above takes in the Hall effect sensor signals and changes the gates of all six power FETs. Each change in the gates represents the direction the force is applied to the motor and if done correctly allows the motor to commutate appropriately. The program also keeps track of counters that help to implement the duty cycle of the PWM signals which control the speed.

While this form of control works and is relatively easy to implement we can see from Figure 32 that the current delivered to the motor is quite messy and shows many harmonics which would cause the ride to be unsteady. In addition to this there is no way to control the amount of torque being generated by the motor.

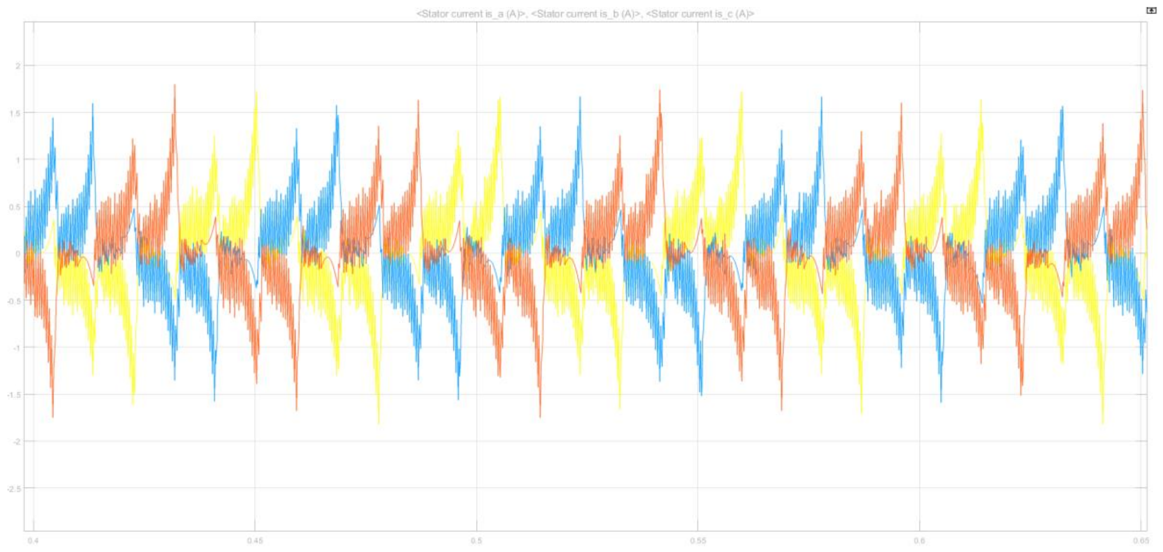


Figure 32 – Trapezoidal Control 3 Phase Currents

Now to help get an overview of the entire simulation there is Figure 33 that shows the entirety of the model. The block on the left that looks like a MOSFET represents three half bridges that control the motor. This block has three inputs, one positive and one negative terminal input for the battery voltage, and then one bus input for the six gates of the FETs. The block also has three outputs; one for phase A, phase B, and phase C. These outputs then go to the input of the motor block along with an external torque signal that represents another force being applied to the motor externally. The output of the motor is a bus connection that contains many signals such as: phase currents, back EMF voltages, speed, position, and Hall Effect sensors. Then the big block where all the signals are going into is a MATLAB function block that contains the code seen in Figure 31. The powergui block sets parameters for the simulation such as the continuity of the simulation and the time step. The remaining blocks in the simulation are either scopes to graph the values of the signals or memory blocks that contain global variables.

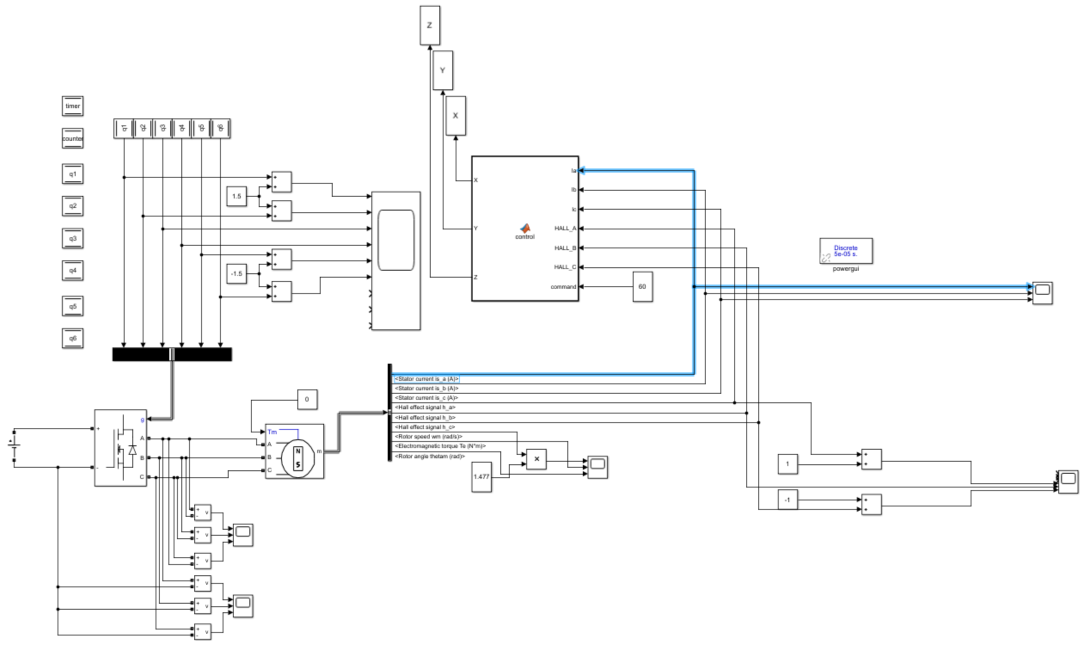


Figure 33 – Entire Simulink Model for Trapezoidal Control

Field Oriented Control

A more appropriate control scheme is what is referred to as Field Oriented Control. Field Oriented Control is a way to control the current in an electric machine rather than the voltage. This directly correlates to controlling speed or controlling torque, as torque is proportional to current and voltage is proportional to speed. However one drawback of FOC is its complexity, while Trapezoidal is relatively easy to implement FOC is quite the opposite. Requiring that the currents be monitored along with the precise position of the rotor arm. With these values in hand a microcontroller is then required to do very challenging calculations to control the current.

The Simulink model for this controller is more complex than the model for the Trapezoidal Controller, it employs more global variables and requires more in depth calculations. It also uses two separate PI controllers to regulate current. The current pushed to the motor using the this Method of control is purely sinusoidal, and varies continuously when contrasted with the current in the Trapezoidal control. 3-Phase sinusoidal current allows for a much smoother ride and direct control of the torque. In addition it also helps with regenerative breaking which is vital to the project to act as resistance to the rider while also increasing the length of the workout.

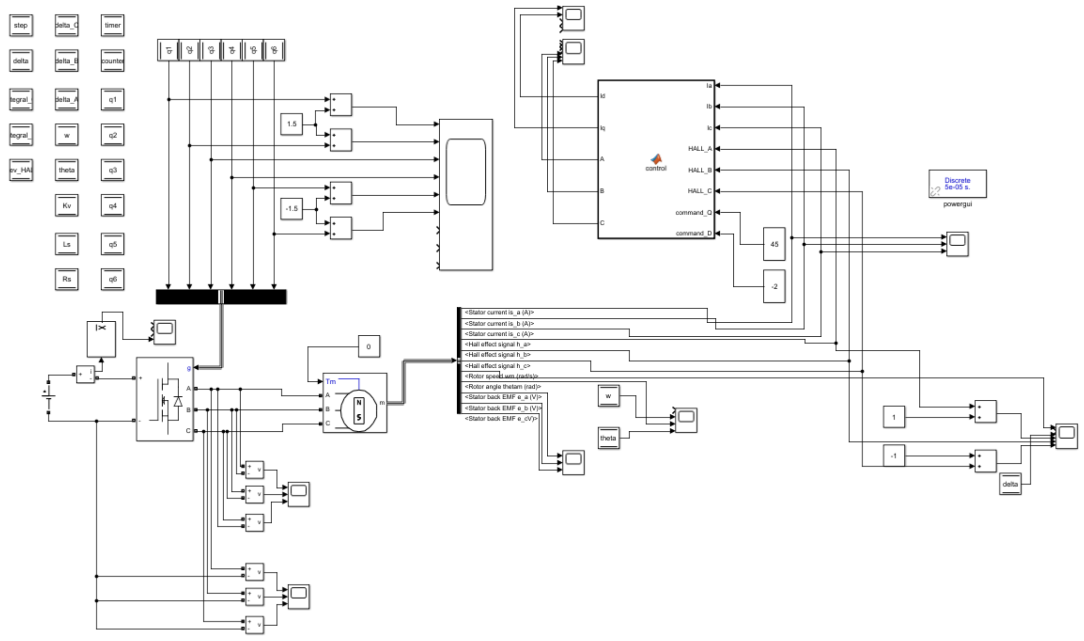


Figure 34 – Entire Simulink Model for Field Oriented Control

Figure 34 above shows the Simulink model for Field Oriented Control for the motor block. The model looks very similar to the Trapezoid model, and in many ways is, but the code that runs the main control block is very different. Instead of looking at the

Hall effect sensors and a command input to change the duty cycle the, it reads the currents from the motor using current sensors and along with the Hall effect sensors. This control function first calculates the position of the rotor arm using the Hall effect sensors and reads the current in each phase of the motor. Using this information it runs a Clarke transformation which changes 3-Phase currents into 2-Phase currents while not changing the amplitude. These two phases, assuming a balanced system, will always be 90 degrees out of phase with each other, hence one can be represented as a sine wave while the other can be represented as a cosine wave. After this is complete the Park transformation is used to transform from a stationary reference frame to a rotating reference frame, that rotates in sync with the motor. This is where the angle of the rotor arm comes into play, since the value of this angle is known the sine and cosine currents now appear to be DC currents that can be much more easily controlled and transformed back to the useful 3-Phase currents needed by the motor.

```
function [Id,Iq,A,B,C] =
control(Ia,Ib,Ic,HALL_A,HALL_B,HALL_C,command_Q,command_D)

% most of the global variables that are used throughout the
program are
% created and listed here
global counter     % Counter for duty cycle
global delta       % Time since previous HALL sensor
global step        % Sample period
global delta_A     % Value of HALL A one step prior
global delta_B     % Value of HALL B one step prior
global delta_C     % Value of HALL C one step prior
global w           % Speed in radians per second
global theta       % Angle in radians
global timer       % Time since start of program
global integral_Q  % Value of discrete integral for Q current
global integral_D  % Value of discrete integral for D current
```

```
A = 0; % Sets initial value for phase A voltage
B = 0; % Sets initial value for phase B voltage
C = 0; % Sets initial value for phase C voltage

% Using current readings from the current sensors we calculate
the alpha
% and beta currents of the motor
[I_alpha,I_beta] = clarke(Ia,Ib,Ic);

% Continuously tracking the amount of time since the previous
Hall effecto
% sensor reading to keep track of speed and position of motor
delta = delta + step;

% Continuously keeps track of the time passed since the beginning
of the
% program to change when we switch from trapezpoidal to FOC
timer = timer + step;

if timer >= 0.0001 % Problem in function fix later
    read_pos(HALL_A,HALL_B,HALL_C); % Keeps track of the position
of the motor
end


delta_A = HALL_A; % Saves previous state of Hall effect sensor A
delta_B = HALL_B; % Saves previous state of Hall effect sensor B
delta_C = HALL_C; % Saves previous state of Hall effect sensor C

theta = theta + w*step; % Integrating to keep track of angle

[Id,Iq] = parke(I_alpha,I_beta,4*theta); % Calculate direct and
quadrature currents

counter = counter + 1; % Keeping track of duty cycle

% Runs trapezoidal control for .1 seconds then flips to FOC
if timer < .1
    trapezoid(command_Q,HALL_A,HALL_B,HALL_C);

% Flipping Field Oriented Control
else
    % PI control for quadrature current
    error_Q = ((command_Q/100)*30) - Iq;
    integral_Q = integral_Q + (error_Q*step);
    u_Q = 1000*integral_Q + 1000*error_Q;

    % PI control for direct current
    error_D = (command_D - Id);
    integral_D = integral_D + (error_D*step);
    u_D = 1000*integral_D + 1000*error_D;

    % Calculate inverse parke transformation for PI control
output
    [I_alpha,I_beta] = inverse_parke(u_D,u_Q,4*theta);

    % Calculate inverse clarke transformation for PI control
output
    [A,B,C] = inverse_clarke(I_alpha,I_beta);
    % Raises or lowers voltages based on coontrol signal
    regulate(A,B,C,Ia,Ib,Ic);
end

      end
```

Figure 35 – Field Oriented Control Matlab Code

The code in Figure 35 above contains the main function that the control block runs during every iteration of the simulation. First all global variables are specified which are fairly well explained in the comments but will also be touched on as they are used in the function. The first main function that is called is the Clarke transform function that requires all three phase currents that are read from the current sensors and returns and alpha current and a beta current which will always be 90 degrees out of phase. Next the delta and timer variables are incremented which is just housekeeping for the function. Next the read_pos function is called which uses the values given by the Hall effect sensors to precisely keep track of the motor position and speed. Using the motor position calculated from this function the Park transformation is started which is moving the reference from a stationary position to a rotating one. The three phase currents will be shown later however the alpha and beta currents are shown below in

Figure 36 and then the currents after the Park transformation are located in Figure 37. These currents are named the Direct current and the Quadrature current and will referred to as I_d current and I_q from now on.

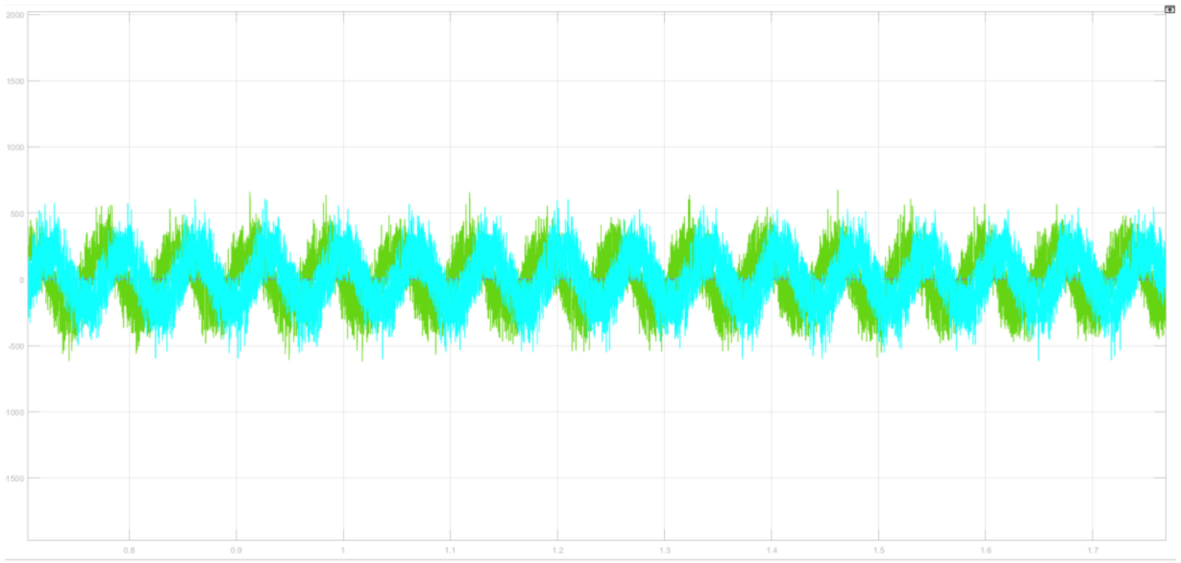


Figure 36 – Alpha and Beta Currents

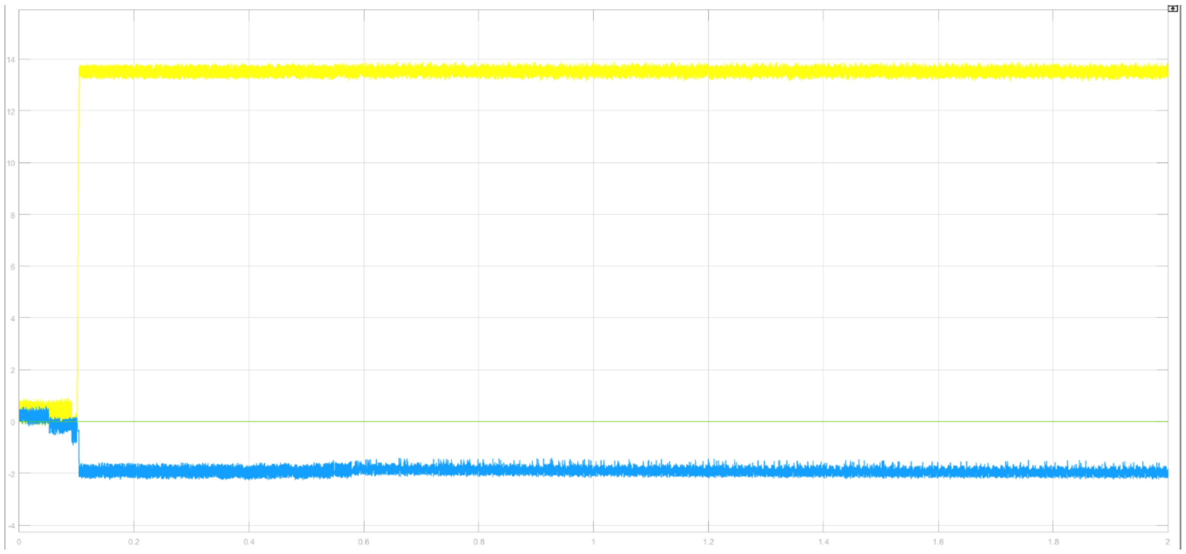


Figure 37 – Direct and Quadrature Currents

I_d and I_q shown in Figure 37 are the currents that will be commanded to the motor where I_q, in yellow, is 90 degrees out of phase with the motor and I_d, in blue is in phase with the motor. So commanding I_q to a certain value while commanding I_d to zero provides the most efficient torque possible to the motor. However it is sometimes necessary to command a negative I_d to help slow down the motor to prevent the back emf from getting too large. This method is referred to as flux weakening and helps to get more stable control to the motor.

After I_d and I_q are calculated from the Clarke and Park transforms, two separate Proportional Integral controllers are used to command each current separately. The PI controllers are implemented directly in the main function with a few simple lines of code. First the error is found by subtracting the actual value of current from the commanded value. Also, a simple discrete integrator is used to keep track of the integral of the error. Then the integral and the error are multiplied by large gain values of 1000 and 500 respectively and added to give the output of the controller. One of these controllers is used to control the current I_d and one is used to control the current I_q. Using these to values and the position of the motor in the inverse Park transform, control currents I_alpha and I_beta are created. Then the inverse Clarke transform creates control currents Ia, Ib, and Ic for each phase of the motor. In Figure 38 the three currents can be seen which resemble a near perfect 3-Phase sine wave.

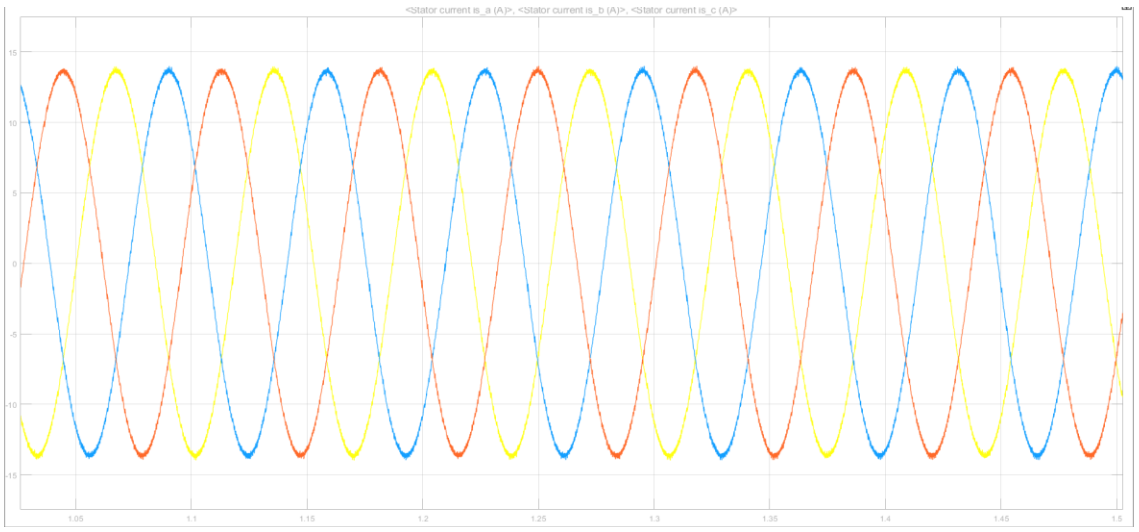


Figure 38 – Field Oriented Control 3 Phase Currents

The main function calls upon many different functions to accomplish this method of control. Below a few of the more important functions are discussed in detail while the code for all is given. The first function is called clarke which can be seen below and is passed the three phase currents and returns alpha and beta currents. As discussed previously the Clarke transform maintains the amplitude of the currents but transforms them to be two currents instead of three that are 90 degrees out of phase. The MATLAB function simply takes the phase A current as the alpha current and the beta current and the difference between phase B and phase C currents multiplied by a constant. The Clarke transformation implements the Equation X and Equation Y below to turn A phase, B phase, and C phase currents into alpha and beta currents.

$$\text{alpha} = a$$

Equation 1

$$\text{beta} = (1/\text{sqrt}(3))*(b-c)$$

Equation 2

```
% This function takes in a/b/c voltages/currents
% and computes the alpha/beta voltages/currents
function [alpha,beta] = clarke(a,b,c)
    alpha = a;
    beta = (1/(sqrt(3))*(b-c));
End
```

Figure 39 – Clarke Transform Function

Next using these newly found alpha and beta currents along with the position of the motor calculated using the function read_pos, discussed later, the Park transform creates two DC currents I_d and I_q that are much more easily controlled via PI controllers. The Park transformation implements the Equation X1 and Equation Y1 below to turn alpha and beta currents into direct quadrature currents.

$$d = \cos(theta)*alpha + \sin(theta)*beta$$

Equation 3

$$q = \cos(theta)*beta - \sin(theta)*alpha$$

Equation 4

```
% This function takes in alpha/beta voltages/currents and the
angle of the
% motor computes the direct/quadrature voltages/currents
function [d,q] = park(alpha,beta,theta)
    d = cos(theta)*alpha + sin(theta)*beta;
    q = cos(theta)*beta - sin(theta)*alpha;
End
```

Figure 40 – Park Transform Function

The best way to understand how the read_pos function is to look at the code and try to follow along with how it adds or subtracts position based on Hall effect sensor inputs. Table 19 below shows the order of the Hall effect sensors, rising and falling edges, and whether or not the motor is moving backward or forward. That along with the time passed since the previous change in Hall effect sensor readings the speed of the motor can be calculated and with it the position. The Arrow next to the letter represents which Hall effect sensor is changing and whether it is a rising edge of the Hall effect signal or the falling edge. For example Event 1 is A↑ and Event 2 is B↓ meaning that the rising edge of Hall effect sensor a occurs followed a falling edge of Hall effect sensor B. Then the minus sign indicates if this series of sensor inputs occurs the motor is moving in reverse.

Event 1	**Event 2**	**Motor Direction**	**Event1**	**Event 2**	**Motor Direction**	**Event 1**	**Event 2**	**Motor Direction**
A↑	**B↑**		**B↑**	**A↑**		**C↑**	**A↑**	
A↑	**B↓**	**-**	**B↑**	**A↓**	**+**	**C↑**	**A↓**	**-**
A↑	**C↑**		**B↑**	**C↑**		**C↑**	**B↑**	
A↑	**C↓**	**+**	**B↑**	**C↓**	**-**	**C↑**	**B↓**	**+**
A↓	**B↑**	**-**	**B↓**	**A↑**	**+**	**C↓**	**A↑**	**-**
A↓	**B↓**		**B↓**	**A↓**		**C↓**	**A↓**	
A↓	**C↑**	**+**	**B↓**	**C↑**	**-**	**C↓**	**B↑**	**+**
A↓	**C↓**		**B↓**	**C↓**		**C↓**	**B↓**	

Table 19 – Order of the Hall Effect Sensors

```
% This function takes the three Hall effect sensor outputs
% along with the amount of time that has passed and integrates to
keep
% track of the position of the motor for use in the parke
transformations
% and inverse parke transformation
function read_pos(HALL_A,HALL_B,HALL_C)
    global prev_HALL
    global delta_A
    global delta_B
    global delta_C
    global delta
    global w

    if((HALL_A-delta_A ~= 0 || HALL_B-delta_B ~= 0 || HALL_C-
delta_C ~= 0))
        if HALL_A-delta_A == -1
        if prev_HALL == 3
            w = -1*((2*pi)/24)/delta;
        elseif prev_HALL == 2
            w = ((2*pi)/24)/delta;
        else
            w = 0;
        end
        prev_HALL = 1;
    elseif HALL_A-delta_A == 1
        if prev_HALL == 3
            w = -1*((2*pi)/24)/delta;
        elseif prev_HALL == 2
            w = ((2*pi)/24)/delta;
        else
            w = 0;
        end
        prev_HALL = 1;
    elseif HALL_B-delta_B == -1
        if prev_HALL == 1
            w = -1*((2*pi)/24)/delta;
        elseif prev_HALL == 3
            w = ((2*pi)/24)/delta;
        else
            w = 0;
        end
        prev_HALL = 2;
    elseif HALL_B-delta_B == 1
        if prev_HALL == 1
            w = -1*((2*pi)/24)/delta;
        elseif prev_HALL == 3
            w = ((2*pi)/24)/delta;
        else
            w = 0;
        end
        prev_HALL = 2;
    elseif HALL_C-delta_C == -1
        if prev_HALL == 2
            w = -1*((2*pi)/24)/delta;
        elseif prev_HALL == 1
            w = ((2*pi)/24)/delta;
        else
            w = 0;
        end
        prev_HALL = 3;
    elseif HALL_C-delta_C == 1
```

```
            if prev_HALL == 1
                w = ((2*pi)/24)/delta;
            elseif prev_HALL == 2
                w = -1*((2*pi)/24)/delta;
            else
                w = 0;
            end
            prev_HALL = 3;
        end
        delta = 0;
    end
end
```

Figure 41 – Matlab Function to Read Motor Speed and Position

Following these functions is the functions to transform the two DC currents I_d and I_q back into I_alpha and I_beta. These sinusoidal currents can then be transformed back into 3-phase currents that will provide torque to the motor. So, in Figure 42 and Figure 43 below the MATLAB code responsible for doing these transformations can be seen.

```
% This function takes in direct/gaudrature voltages/currents
% and the angle of the motor and performs the reverse parke
transformation
% giving alpha/beta voltages/currents
function [alpha,beta] = inverse_parke(d,q,theta)
    alpha = d*cos(theta) - q*sin(theta);
    beta = q*cos(theta) + d*sin(theta);
end
```

Figure 42 – Matlab Function for Inverse Park Transformation

```
% This function takes in alpha/beta voltages/currents and
% performs the reverse clarke transformation and returns a/b/c
% voltages/currents
function [a,b,c] = inverse_clarke(alpha,beta)
    a = alpha;
    b = (1/2)*((-alpha) + (sqrt(3)*beta));
    c = (1/2)*((-alpha) - (sqrt(3)*beta));
end
```

Figure 43 – Matlab Function for Inverse Clarke Transform

Finally the last function that is vital to the Field Oriented Control model is the regulate function. This function takes in the actual currents read from the three phases of the motor and the newly found control signals from the PI controllers and tries to force the control currents through each phase. The method for doing this is quite simple, if the current through one phase is lower than the control current the top FET for that phase is turned on and the bottom phase is turned off. Otherwise the bottom FET is turned on and the top FET is turned off.

```
% regulate is a function that comapares the values of the
% voltages on the line and tries to raise them if they are below
% the value that the PI controller is asking for
% or lowers them if the actual value is less than what the PI
controller
% asks for
function regulate(command_A,command_B,command_C,A,B,C)
    global q1
    global q2
    global q3
    global q4
    global q5
    global q6

    if A < command_A
        q1 = 1;
        q2 = 0;
    else
        q1 = 0;
        q2 = 1;
    end
    if B < command_B
        q3 = 1;
        q4 = 0;
    else
        q3 = 0;
        q4 = 1;
    end
    if C < command_C
        q5 = 1;
        q6 = 0;
    else
        q5 = 0;
        q6 = 1;
    end
end
```

Figure 44 – Regulate Function that Drives Phase Currents

3.9.6 Level 3 Embedded System Calculations [RKA]

A bootstrap capacitor is needed for each MOSFET that is connected to the upper rail and raises the gate bias of the top MOSFET so that it is always at a higher value than the phase node (or the middle point in between the upper and lower MOSFETs on the bridge). When the low side MOSFET is on, the IC driving circuit will charge this capacitor up to roughly 12V (when low side MOSFET conducts, a path to the ground node of phase A is completed to allow for this charging). When the low side MOSFET is turned off, almost all the current provided at the gate of the high side MOSFET will be sourced from the bootstrap capacitor. It is important to size this capacitor correctly because it needs to be able to charge up to a sufficient value to bias the gate correctly but also must discharge quickly to turn the gate on. An even more critical issue is that the IC driver circuit has undervoltage lockout (a protective measure) so the ripple caused by the charging/discharging of this capacitor must be limited to make sure this protective measure is not triggered. The following equation is the starting point for calculation:

$$Q_{CB} = Q_G + (D * t_{cyc} * I_B)$$

Where Q_{CB} is the total charge that needs delivered to bootstrap capacitor at maximum duty cycle, Q_G is the total gate charge needed to turn on the MOSFET, D is the duty cycle, and I_B is the VDDA IC driver circuit pin biasing current. From the datasheet Q_G = 88nC, assuming a max duty cycle of 90%, t_{cyc} = 50us, and I_B = 3mA. This gives us a value of Q_{CB} = 223nC. Next, we use the following equation:

$$C_B \geq \frac{Q_{CB}}{\Delta V_{CB}}$$

Where C_B is the bootstrap capacitor and ΔV_{CB} is the maximum allowable ripple voltage which we choose To be 5% of the VDD value, 12V. This gives us a value of $C_B \geq 0.5$uF.

After talking to the faculty advisor we choose to use a 10uF capacitor which is sufficiently larger than the calculated 0.5uF.

We also place a resistor in series with the capacitor which forms an RC circuit. The time constant of this circuit is calculated using:

$$\tau = RC$$

With a resistance value of 10Ω, we have τ = 100us. This is the time it will take to charge the capacitor and in this application is sufficient considering the maximum rotations per minute and corresponding off time for a single MOSFET.

3.9.7 Level 3 Embedded System Parts List [TRM][NMD]

Qty	Value	Package	Parts	Description
3	2	R0805	R1, R2, R3	Resistors
3	3	R0805	R4, R6, R8	Resistors
3	10	R0805	R5, R7, R9	Resistors
2	120	R0805	R13, R24	Resistors
5	220	R0805	R15, R17, R18, R19, R20	Resistors
2	470	R0805	R11, R14	Resistors
4	.1u	C0805	C10, C11, C12, C15	Capacitors
1	8MHz	HC49UP	XT1	Oscillator
3	100u	CAP_ECEV_G	C5, C7, C16	Capacitors
9	10k	R0805	R10, R12, R16, R21, R22, R23, R26, R27, R28	Resistors
3	10n	C0805	C6, C18, C21	Capacitors
4	10u	C0805	C1, C2, C3, C9	Capacitors
2	18p	C0805	C13, C14	Capacitors
5	22-27-2031-03	6410-03	J1, J2, J3, J4, J5	Connectors
3	3.3u	C0805	C17, C19, C20	Capacitors
1	330u	CAP_ECEV_G	C8	Capacitors
1	33n	C0805	C4	Capacitors
1	4.7k	R0805	R25	Resistors
3	ACS770	5CB	U14, U15, U16	
1	Blue	0805	LED3	LEDs
3	Green	0805	LED1, LED2, LED8	LEDs
6	IPP045N10N3 GXKSA1	TO220	FET1, FET2, FET3, FET4, FET5, FET6	MOSFETs
1	MIC4607	TSSOP28	U1	
1	MOLEX_42840-2224	MOLEX_42820-2224	X1	Connectors
4	RED	0805	LED4, LED5, LED6, LED7	LEDs

6	RK7002B	SOT23	U$1, U$2, U$3, U$4, U$5, U$6	
3	SRH05S05	SIP3	U3, U4, U7	
1	Terminal	3_TERMINAL	U$7	

3.10.1 Motor Overview [TRM][RKA][ELW]

Seeing as though this project is supposed to adjust the torque of a motor to try and keep a person's heartrate constant, it is important to spec out an appropriately sized motor. The following showcases the calculations that were used to size the motor:

Definition of Terms:

1. F_h = Normal force on a hill in Lbs
2. F_f = Force of friction in Lbs
3. W_{tot} = Total weight in Lbs
4. Θ_s = Angle of slope in degrees
5. K_f = Coefficient of static friction
6. F_{tot} = Total force in Lbs
7. T_{tot} = Total torque in Lb-in
8. W_r = Wheel Radius in Feet
9. M_p = Motor Power in Kilowatts
10. S_{rpm} = Speed in RPM
11. S_{mph} = Speed in MPH
12. $Batt_{num}$ = Number of batteries
13. $Batt_{Voltage}$ = Nominal battery voltage in volts
14. $Batt_{Ampacity}$ = Battery ampacity in amp-hours
15. $Motor_{voltage}$ = Motor voltage in volts
16. $Batt_{Type}$ = Type Of Battery Used

Assumptions:

1. W_{tot} = 235.90 Lbs
2. $\Theta_s = 6°$
3. K_f = 0.004
4. W_r = 1 ft
5. S_{mph} = 7.5 mph
6. $Motor_{voltage}$ = 24 V
7. Ride Duration = 1 hour
8. $Batt_{Type}$ = Lithium Ions
    a. $Batt_{Voltage}$ = 3.7 V
    b. $Batt_{Ampacity}$ = 2500 Ah

Equations:

1. $F_f = W_{Tot} * \text{Cos}(\Theta_s) * K_f$
2. $F_h = W_{tot} * \text{Sin}(\Theta_s)$
3. $F_{Tot} = F_h + F_f$
4. $T_{Tot} = F_{Tot} * 12 * W_r$
5. $M_P = ((T_{Tot} * S_{rpm}) / 63025) * 0.75$
6. $S_{rpm} = ((S_{mph}/60)*5280) / (2 * PI * W_r)$
7. $Batt_{Type}$ = Li-Ion Batteries

Calculations

1. S_{rpm} = ((7.5/60)*5280) / (2 * PI * W_r) = 105
2. F_f = 235.90 * Cos(6*PI/180) * 0.004 = 0.938
3. F_h = 235.9 * Sin(6*PI/180) = 24.658
4. F_{Tot} = 24.658 + 0.938 = 25.6
5. T_{Tot} = 25.6 * 12 * 1 = 307.16
6. M_P = ((307.16 * S_{rpm}) / 63025) * 0.75 = 0.384

Following the above calculations it can be seen that, at a minimum, a 384W motor capable of delivering 307 lb-in of torque is needed to drive the expected load (235.9 lbs) up a 6 degree incline. After realizing this, a motor that the university already had was found and the specs for it more that met the above requirements. This motor is a Phoenix Racer II BLDC hub motor, capable of handling more than 3000W of power with more than 1000 lb-in of torque. Because the motor is a lot larger than necessary for this application, it will never have to be run at its maximum capacity. This ensures that this project will never over duty the motor and cause it to fail.

Sensing Motor Position:

To adequately derive the position of the motor at any time “t”, Hall Effect sensors that are placed inside of the motor are constantly being measured. Hall effect sensors work based on the principles of magnetic fields. As a permanent magnets move pass the sensors, a voltage is induced across the sensors which can then be measured. By knowing what sensors have a voltage induced on them at any given time, the position of the motor can be derived. A graph of the outputs of the Hall Effect sensors can be seen below:

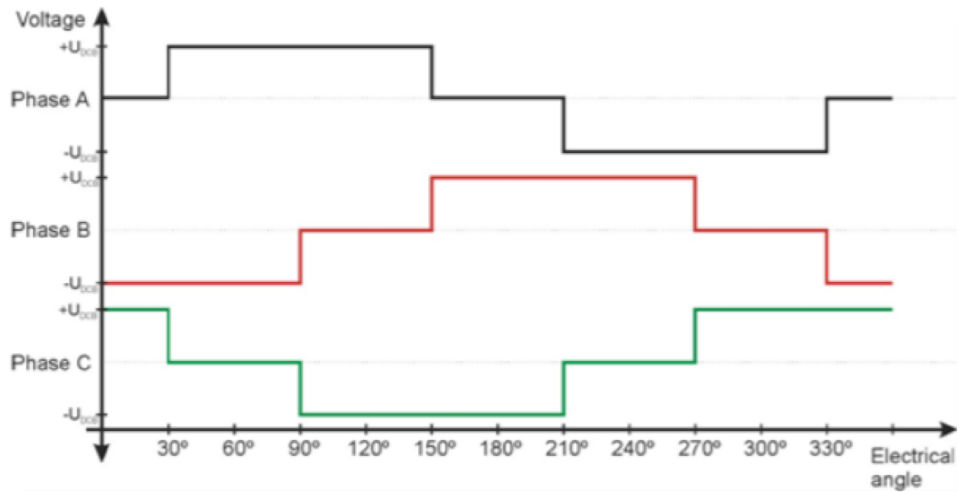


Figure 45 – Hall Effect Sensors' Outputs

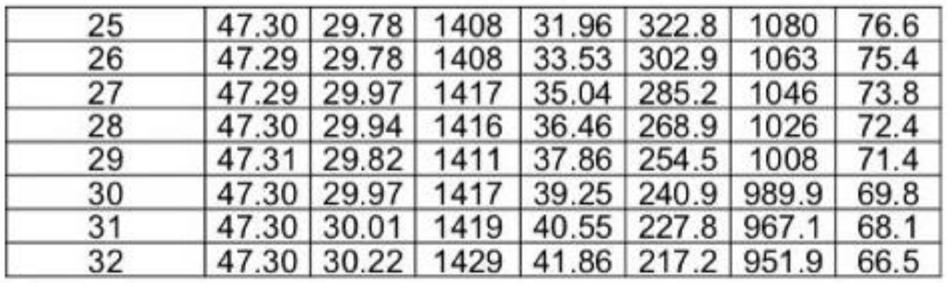
Motor Datasheet:

After calculating the values that were required from a motor to successfully complete this project, the motor that the university had was analyzed. In order to analyze the motor, the datasheet from the manufacturing company was acquired. By acquiring this datasheet from the manufacturer, less time was needed to be spent analyzing the motor and there were less inaccuracies in the measurements. Below is the datasheet that was supplied from the motor manufacturer.

Crystalyte 5403 Motor

Descripton	U	I	P1	M	N	P2	Eff
	(V)	(A)	(W)	(N.m)	(rpm)	(W)	(%)
Unload	48.15	2.518	121.2	0.06	522.0	3.27	2.7
Max-Eff	47.41	22.97	1089	18.68	456.5	892.8	81.9
Max-Pout	47.18	30.18	1424	25.55	435.0	1163	81.6
Max-Torque	47.30	30.22	1429	41.86	217.2	951.9	66.5
End	47.30	30.22	1429	41.86	217.2	951.9	66.5

NO:	U	I	P1	M	N	P2	Eff
	(V)	(A)	(W)	(N.m)	(rpm)	(W)	(%)
1	48.15	2.518	121.2	0.06	522.0	3.27	2.7
2	48.15	2.623	126.2	0.16	521.9	8.74	6.9
3	48.13	2.923	140.7	0.45	521.2	24.55	17.4
4	48.11	3.475	167.1	0.94	518.8	51.05	30.5
5	48.08	4.190	201.5	1.53	517.0	82.81	41.0
6	48.05	5.118	245.9	2.42	513.5	130.1	52.8
7	48.00	6.235	299.3	3.35	510.1	178.9	59.7
8	47.95	7.518	360.5	4.51	505.7	238.7	66.2
9	47.90	8.961	429.2	5.83	501.7	306.2	71.3
10	47.84	10.47	501.3	7.22	496.5	375.3	74.8
11	47.78	12.15	580.7	8.73	491.0	448.7	77.2
12	47.72	13.87	662.3	10.32	485.0	524.0	79.1
13	47.65	15.63	745.1	11.81	480.4	594.0	79.7
14	47.59	17.47	831.5	13.50	474.5	670.6	80.6
15	47.53	19.32	918.8	15.31	467.8	749.8	81.6
16	47.47	21.16	1004	16.93	462.6	819.9	81.6
17	47.41	22.97	1089	18.68	456.5	892.8	81.9
18	47.35	24.87	1178	20.40	451.1	963.4	81.7
19	47.29	26.67	1261	22.13	445.3	1031	81.7
20	47.24	28.55	1349	23.85	439.2	1096	81.2
21	47.18	30.18	1424	25.55	435.0	1163	81.6
22	47.30	29.66	1403	27.24	389.6	1111	79.1
23	47.31	29.33	1388	28.89	363.2	1098	79.1
24	47.30	29.70	1405	30.50	337.9	1079	76.7



25	47.30	29.78	1408	31.96	322.8	1080	76.6
26	47.29	29.78	1408	33.53	302.9	1063	75.4
27	47.29	29.97	1417	35.04	285.2	1046	73.8
28	47.30	29.94	1416	36.46	268.9	1026	72.4
29	47.31	29.82	1411	37.86	254.5	1008	71.4
30	47.30	29.97	1417	39.25	240.9	989.9	69.8
31	47.30	30.01	1419	40.55	227.8	967.1	68.1
32	47.30	30.22	1429	41.86	217.2	951.9	66.5

Figure 46 – Phoenix Racer II Datasheet

3.11.1 Heart Rate Sensor Overview [RKA][ELW]

The user will be wearing a heart rate sensor that will represent a heartbeat with an electrical pulse. This pulsed signal will be transmitted wirelessly over Bluetooth to the embedded system. Integrating these pulses over time will allow us to calculate beats per minute. Communication will be done with a Bluetooth module hooked up to a DsPIC-33. The heart rate sensor is powered by coin cell battery.

3.12.1 Wiring Overview [TRM]

This project has multiple subsystem, that all have to interact in order to work reliably. In order to ensure this happens, multiple wires and buses will need to be used to interconnect the system. Below is a diagram showcasing the wiring of this system:

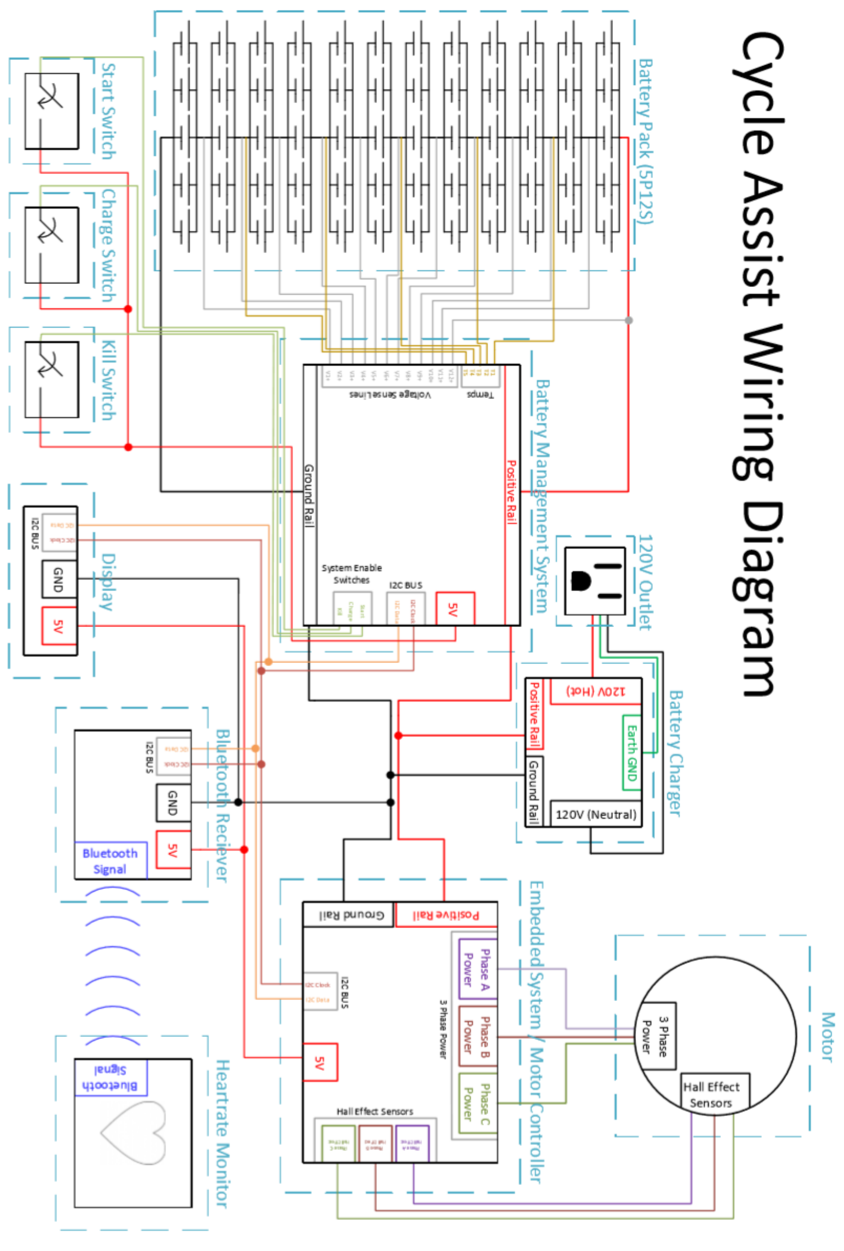


Figure 47 – Wiring Diagram

4. Parts List [TRM][NMD][RKA][ELW]

Qty	Value	Package	Parts	Total Price
1	TACTILE-SWITCH	TACTILE-SWITCH	S1	$0.33
17	1k	R0805	R38, R39, R40, R41, R42, R43, R44, R45, R46, R47, R48, R49, R50, R57, R63, R73, R74	$1.70
8	1uF	C0805	C16, C17, C18, C19, C22, C23, C25, C26	$1.60
1	SSM3J328RLFTR-NO	SOT23-3	Q13	$0.52
12	3.3k	R0805	R4, R6, R9, R12, R15, R18, R21, R24, R27, R30, R33, R36	$1.20
1	3.3uF	C0805	C15	$0.20
1	8MHz	HC49UP	XT1	$0.33
12	10k	R0805	R51, R52, R53, R54, R55, R60, R61, R64, R66, R68, R70, R72	$1.20
1	10n	C0805	C20	$0.20
12	10nF	C0805	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12	$2.40
2	20k	R0805	R65, R69	$0.20
12	33	R0805	R3, R7, R10, R13, R16, R19, R22, R25, R28, R31, R34, R37	$1.20
16	100	R0805	R1, R2, R5, R8, R11, R14, R17, R20, R23, R26, R29, R32, R35, R56, R67, R71	$1.60
1	100n	C0805	C21	$0.20
1	100nF	C0805	C13	$0.20
1	100uF	C0805	C24	$0.20

1	B59707A0120A062	1210	F2	$1.10
2	BMS3004	TO218V	U$1, U$2	$8.34
12	DMP2305U	SOT-23	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12	$6.24
2	FK3306010L	SSSMINI3-F2-B	U2, U6	$0.86
1	FTDI_Header	1X05	J1	$0.00
14	Green	LED-0603	D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D14, D15	$4.34
1	ICSP Header	1X05	J2	$0.00
5	LTC6804	SSOP48	U1	$95.35
1	MLX91210	SOIC-8	U4	$3.91
2	MOLEX_42840-2224	MOLEX_42820-2224	X2, X4	$5.00
2	MOLEX_105314-1114	MOLEX_105314-1114	X1, X5	$5.00
1	PIC16F1789-I/PTTQFP44_MC-L	TQFP44_MC-L	U7	$2.38
2	RED	LED-0603	D13, D16	$0.62
1	SRH05S05	SIP3	U5	$8.00
8	TP5015	TP5015	TP1, TP2, TP3, TP6, TP7, TP8, TP9, TP13	$3.52
5	NXRT15XV103FA1B040	Through Hole	Not On Schematics	$2.70
3	2	R0805	R1, R2, R3	$0.30
3	3	R0805	R4, R6, R8	$0.30
3	10	R0805	R5, R7, R9	$0.30
2	120	R0805	R13, R24	$0.20
5	220	R0805	R15, R17, R18, R19, R20	$0.50
2	470	R0805	R11, R14	$0.20
4	.1u	C0805	C10, C11, C12, C15	$1.20

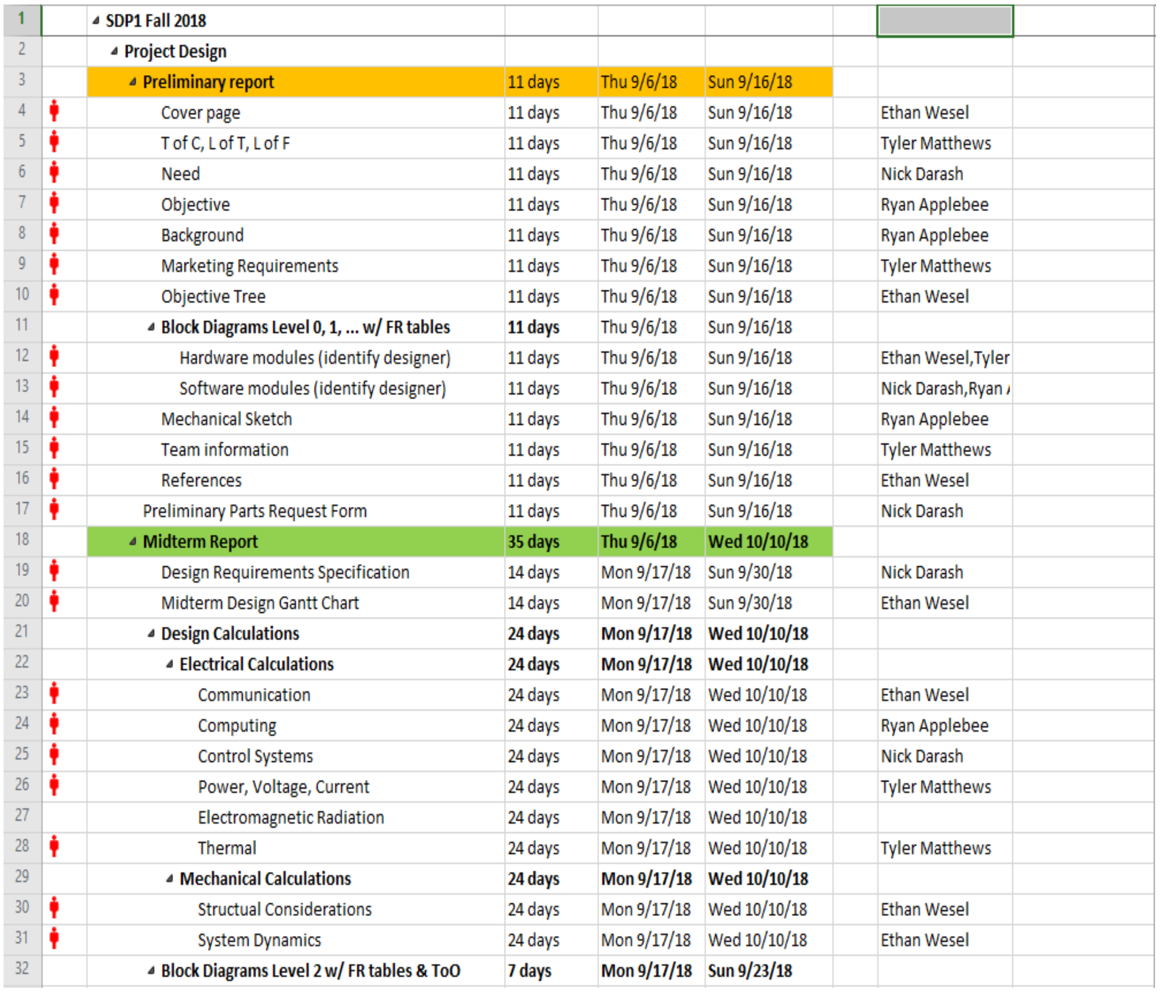
1	8MHz	HC49UP	XT1	$0.33
3	100u	CAP_ECEV_G	C5, C7, C16	$1.41
9	10k	R0805	R10, R12, R16, R21, R22, R23, R26, R27, R28	$0.90
3	10n	C0805	C6, C18, C21	$0.30
4	10u	C0805	C1, C2, C3, C9	$0.40
2	18p	C0805	C13, C14	$0.20
5	22-27-2031-03	6410-03	J1, J2, J3, J4, J5	$0.37
3	3.3u	C0805	C17, C19, C20	$0.30
1	330u	CAP_ECEV_G	C8	$0.47
1	33n	C0805	C4	$0.30
1	4.7k	R0805	R25	$0.10
3	ACS770	5CB	U14, U15, U16	$25.14
1	Blue	0805	LED3	$0.31
3	Green	0805	LED1, LED2, LED8	$0.93
6	IPP045N10N3GXKSA1	TO220	FET1, FET2, FET3, FET4, FET5, FET6	$17.67
1	MIC4607	TSSOP28	U1	$6.27
1	MOLEX_42840-2224	MOLEX_42820-2224	X1	$5.00
4	RED	0805	LED4, LED5, LED6, LED7	$1.24
6	RK7002B	SOT23	U$1, U$2, U$3, U$4, U$5, U$6	$1.86

3	SRH05S05	SIP3	U3, U4, U7	$24.00
1	Terminal	3_TERMINAL	U$7	$0.00
1	Polar T34 Heart Rate Transmitter	Heart Rate Sensor with Bluetooth		$65.00
1	SSD1306 OLED	Display		$10.99
5	BMS PCBs from JLC PCB			$100.00
5	Embedded System / Motor Controller PCBs from JLC PCB			$100.00
			Total Cost:	**$527.13**

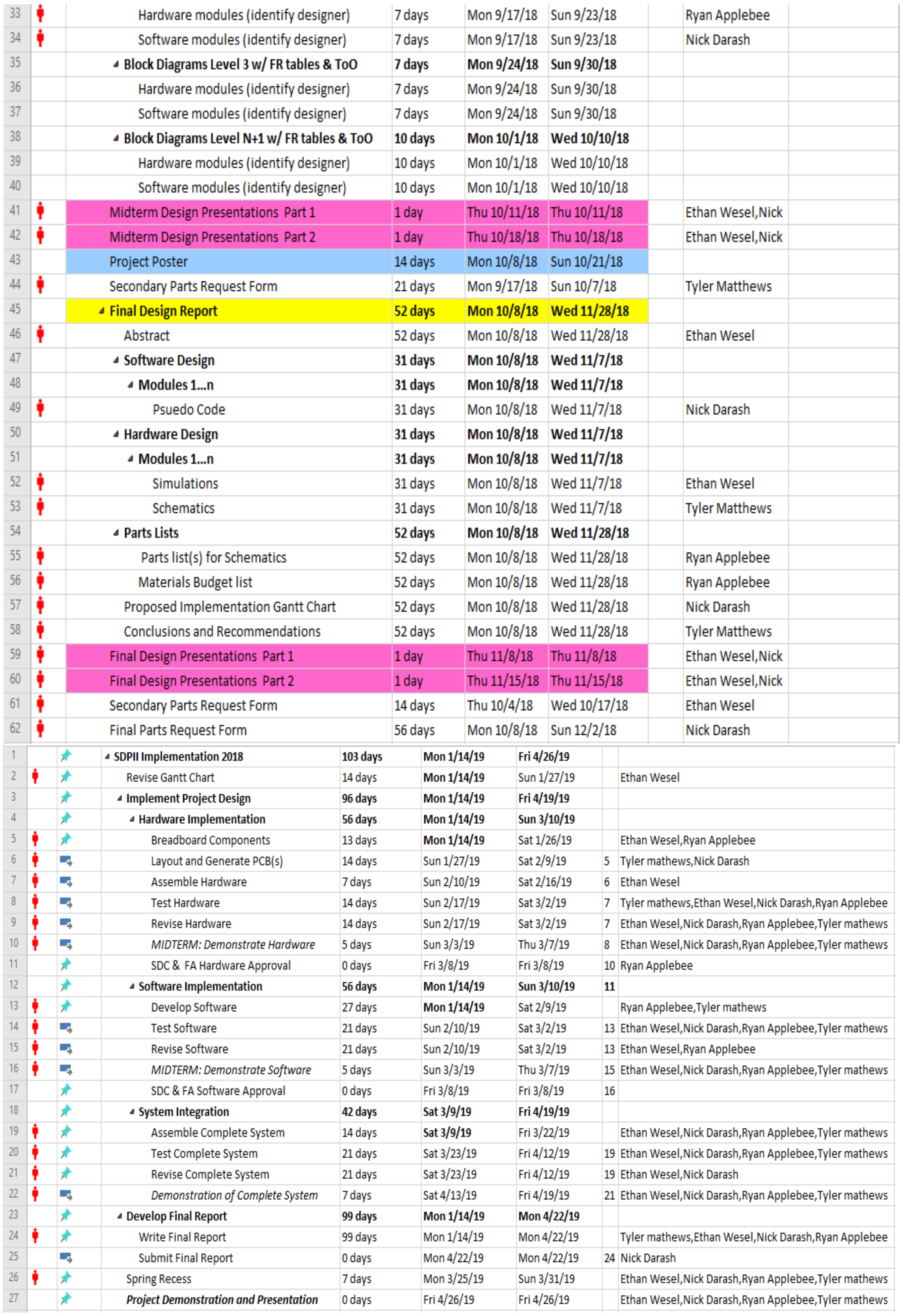
Table 20 –Master Parts List

5. Project Schedule [TRM][NMD][RKA][ELW]

The following Gantt chart represents the projected team schedule. It shows the various deadlines and who is responsible for facilitating the progress of each. It is important to outline the timeline to make sure all vital parts of the project are completed in a timely manner.



1		**SDP1 Fall 2018**						
2		**Project Design**						
3		**Preliminary report**	11 days	Thu 9/6/18	Sun 9/16/18			
4	🧍	Cover page	11 days	Thu 9/6/18	Sun 9/16/18		Ethan Wesel	
5	🧍	T of C, L of T, L of F	11 days	Thu 9/6/18	Sun 9/16/18		Tyler Matthews	
6	🧍	Need	11 days	Thu 9/6/18	Sun 9/16/18		Nick Darash	
7	🧍	Objective	11 days	Thu 9/6/18	Sun 9/16/18		Ryan Applebee	
8	🧍	Background	11 days	Thu 9/6/18	Sun 9/16/18		Ryan Applebee	
9	🧍	Marketing Requirements	11 days	Thu 9/6/18	Sun 9/16/18		Tyler Matthews	
10	🧍	Objective Tree	11 days	Thu 9/6/18	Sun 9/16/18		Ethan Wesel	
11		**Block Diagrams Level 0, 1, ... w/ FR tables**	**11 days**	Thu 9/6/18	Sun 9/16/18			
12	🧍	Hardware modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18		Ethan Wesel,Tyler	
13	🧍	Software modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18		Nick Darash,Ryan	
14	🧍	Mechanical Sketch	11 days	Thu 9/6/18	Sun 9/16/18		Ryan Applebee	
15	🧍	Team information	11 days	Thu 9/6/18	Sun 9/16/18		Tyler Matthews	
16	🧍	References	11 days	Thu 9/6/18	Sun 9/16/18		Ethan Wesel	
17	🧍	Preliminary Parts Request Form	11 days	Thu 9/6/18	Sun 9/16/18		Nick Darash	
18		**Midterm Report**	**35 days**	**Thu 9/6/18**	**Wed 10/10/18**			
19	🧍	Design Requirements Specification	14 days	Mon 9/17/18	Sun 9/30/18		Nick Darash	
20	🧍	Midterm Design Gantt Chart	14 days	Mon 9/17/18	Sun 9/30/18		Ethan Wesel	
21		**Design Calculations**	**24 days**	**Mon 9/17/18**	**Wed 10/10/18**			
22		**Electrical Calculations**	**24 days**	**Mon 9/17/18**	**Wed 10/10/18**			
23	🧍	Communication	24 days	Mon 9/17/18	Wed 10/10/18		Ethan Wesel	
24	🧍	Computing	24 days	Mon 9/17/18	Wed 10/10/18		Ryan Applebee	
25	🧍	Control Systems	24 days	Mon 9/17/18	Wed 10/10/18		Nick Darash	
26	🧍	Power, Voltage, Current	24 days	Mon 9/17/18	Wed 10/10/18		Tyler Matthews	
27		Electromagnetic Radiation	24 days	Mon 9/17/18	Wed 10/10/18			
28	🧍	Thermal	24 days	Mon 9/17/18	Wed 10/10/18		Tyler Matthews	
29		**Mechanical Calculations**	**24 days**	**Mon 9/17/18**	**Wed 10/10/18**			
30	🧍	Structual Considerations	24 days	Mon 9/17/18	Wed 10/10/18		Ethan Wesel	
31	🧍	System Dynamics	24 days	Mon 9/17/18	Wed 10/10/18		Ethan Wesel	
32		**Block Diagrams Level 2 w/ FR tables & ToO**	**7 days**	**Mon 9/17/18**	**Sun 9/23/18**			



33	Hardware modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18		Ryan Applebee
34	Software modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18		Nick Darash
35	**Block Diagrams Level 3 w/ FR tables & ToO**	**7 days**	**Mon 9/24/18**	**Sun 9/30/18**		
36	Hardware modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18		
37	Software modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18		
38	**Block Diagrams Level N+1 w/ FR tables & ToO**	**10 days**	**Mon 10/1/18**	**Wed 10/10/18**		
39	Hardware modules (identify designer)	10 days	Mon 10/1/18	Wed 10/10/18		
40	Software modules (identify designer)	10 days	Mon 10/1/18	Wed 10/10/18		
41	Midterm Design Presentations Part 1	1 day	Thu 10/11/18	Thu 10/11/18		Ethan Wesel,Nick
42	Midterm Design Presentations Part 2	1 day	Thu 10/18/18	Thu 10/18/18		Ethan Wesel,Nick
43	Project Poster	14 days	Mon 10/8/18	Sun 10/21/18		
44	Secondary Parts Request Form	21 days	Mon 9/17/18	Sun 10/7/18		Tyler Matthews
45	**Final Design Report**	**52 days**	**Mon 10/8/18**	**Wed 11/28/18**		
46	Abstract	52 days	Mon 10/8/18	Wed 11/28/18		Ethan Wesel
47	**Software Design**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
48	**Modules 1…n**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
49	Psuedo Code	31 days	Mon 10/8/18	Wed 11/7/18		Nick Darash
50	**Hardware Design**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
51	**Modules 1…n**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
52	Simulations	31 days	Mon 10/8/18	Wed 11/7/18		Ethan Wesel
53	Schematics	31 days	Mon 10/8/18	Wed 11/7/18		Tyler Matthews
54	**Parts Lists**	**52 days**	**Mon 10/8/18**	**Wed 11/28/18**		
55	Parts list(s) for Schematics	52 days	Mon 10/8/18	Wed 11/28/18		Ryan Applebee
56	Materials Budget list	52 days	Mon 10/8/18	Wed 11/28/18		Ryan Applebee
57	Proposed Implementation Gantt Chart	52 days	Mon 10/8/18	Wed 11/28/18		Nick Darash
58	Conclusions and Recommendations	52 days	Mon 10/8/18	Wed 11/28/18		Tyler Matthews
59	Final Design Presentations Part 1	1 day	Thu 11/8/18	Thu 11/8/18		Ethan Wesel,Nick
60	Final Design Presentations Part 2	1 day	Thu 11/15/18	Thu 11/15/18		Ethan Wesel,Nick
61	Secondary Parts Request Form	14 days	Thu 10/4/18	Wed 10/17/18		Ethan Wesel
62	Final Parts Request Form	56 days	Mon 10/8/18	Sun 12/2/18		Nick Darash

1	**SDPII Implementation 2018**	**103 days**	**Mon 1/14/19**	**Fri 4/26/19**		
2	Revise Gantt Chart	14 days	**Mon 1/14/19**	Sun 1/27/19		Ethan Wesel
3	**Implement Project Design**	**96 days**	**Mon 1/14/19**	**Fri 4/19/19**		
4	**Hardware Implementation**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**		
5	Breadboard Components	13 days	**Mon 1/14/19**	Sat 1/26/19		Ethan Wesel,Ryan Applebee
6	Layout and Generate PCB(s)	14 days	Sun 1/27/19	Sat 2/9/19	5	Tyler mathews,Nick Darash
7	Assemble Hardware	7 days	Sun 2/10/19	Sat 2/16/19	6	Ethan Wesel
8	Test Hardware	14 days	Sun 2/17/19	Sat 3/2/19	7	Tyler mathews,Ethan Wesel,Nick Darash,Ryan Applebee
9	Revise Hardware	14 days	Sun 2/17/19	Sat 3/2/19	7	Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
10	*MIDTERM: Demonstrate Hardware*	5 days	Sun 3/3/19	Thu 3/7/19	8	Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
11	SDC & FA Hardware Approval	0 days	Fri 3/8/19	Fri 3/8/19	10	Ryan Applebee
12	**Software Implementation**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**	**11**	
13	Develop Software	27 days	**Mon 1/14/19**	Sat 2/9/19		Ryan Applebee,Tyler mathews
14	Test Software	21 days	Sun 2/10/19	Sat 3/2/19	13	Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
15	Revise Software	21 days	Sun 2/10/19	Sat 3/2/19	13	Ethan Wesel,Ryan Applebee
16	*MIDTERM: Demonstrate Software*	5 days	Sun 3/3/19	Thu 3/7/19	15	Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
17	SDC & FA Software Approval	0 days	Fri 3/8/19	Fri 3/8/19	16	
18	**System Integration**	**42 days**	**Sat 3/9/19**	**Fri 4/19/19**		
19	Assemble Complete System	14 days	**Sat 3/9/19**	Fri 3/22/19		Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
20	Test Complete System	21 days	Sat 3/23/19	Fri 4/12/19	19	Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
21	Revise Complete System	21 days	Sat 3/23/19	Fri 4/12/19	19	Ethan Wesel,Nick Darash
22	*Demonstration of Complete System*	7 days	Sat 4/13/19	Fri 4/19/19	21	Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
23	**Develop Final Report**	**99 days**	**Mon 1/14/19**	**Mon 4/22/19**		
24	Write Final Report	99 days	Mon 1/14/19	Mon 4/22/19		Tyler mathews,Ethan Wesel,Nick Darash,Ryan Applebee
25	Submit Final Report	0 days	Mon 4/22/19	Mon 4/22/19	24	Nick Darash
26	Spring Recess	7 days	Mon 3/25/19	Sun 3/31/19		Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews
27	***Project Demonstration and Presentation***	0 days	Fri 4/26/19	Fri 4/26/19		Ethan Wesel,Nick Darash,Ryan Applebee,Tyler mathews

Figure 48 – Gantt Chart

6. Design Team Information

- **Tyler Matthews**
  - Electrical Engineer – Team Leader
- **Nick Darash**
  - Electrical Engineer – Software Manager
- **Ethan Wesel**
  - Electrical Engineer – Hardware Manager
- **Ryan Applebee**
  - Electrical Engineer – Engineering Data Manager

7. Conclusions and Recommendations [TRM][NMD][RKA][ELW]

To conclude, the design for a heart rate based user assisting bicycle may seem simple on the surface but is actually quite complex. The three main modules for this project – the battery management system, embedded system, and motor controller all require hardware tailored to this specific application. Each of these main components must receive and/or send the necessary signals for reliable, predictable operation and then utilize software to make complex decisions as fast as possible with this data. As always, consistent, predictable operation to ensure user safety is of the utmost importance. If the battery management system doesn't act quickly enough, catastrophic damage can occur to both the batteries and the various electronics powered by them. If the embedded system doesn't tell the motor controller what to do at exactly the right time, the motor may grind to a halt, speed up uncontrollably, or turn on a MOSFET at the wrong time short circuiting the batteries. Any of these possibilities can have dire consequences which is why this project must be executed with high precision, and every possibility must be considered to guarantee fail-safe operation.

8. References

[1]J. S. Lee, J. W. Jiang and Y. H. Sun, "Design and simulation of control systems for electric-assist bikes," *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, Hefei, 2016, pp. 1736-1740.
doi: 10.1109/ICIEA.2016.7603866
URL:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7603866&isnumber=76

03539

[2]T. Gibson, "Turning sweat into watts," in *IEEE Spectrum*, vol. 48, no. 7, pp. 50-55, July 2011.
doi: 10.1109/MSPEC.2011.5910449,
URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5910449&isnumber=5910416

[3]N. Constant, T. Wang and K. Mankodiya, "Pulseband: A hands-on tutorial on how to design a smart wristband to monitor heart-rate," *2015 IEEE Virtual Conference on Applications of Commercial Sensors (VCACS)*, Raleigh, NC, 2015, pp. 1-3.
doi: 10.1109/VCACS.2015.7439565
URL:

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7439565&isnumber=7439560

[4]K. Vijyalakshmi, S. Sandhiya and C. Bhuvaneswari, "High efficiency bi-directional converter with regenerative concept for E-bike," *2017 International Conference on Computation of Power, Energy Information and Commuincation (ICCPEIC)*, Melmaruvathur, India, 2017, pp. 668-671.
doi: 10.1109/ICCPEIC.2017.8290445
URL:

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8290445&isnumber=8290323

[5]N. Hatwar, A. Bisen, H. Dodke, A. Junghare and M. Khanapurkar, "Design approach for electric bikes using battery and super capacitor for performance improvement," *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, The Hague, 2013, pp. 1959-1964.
doi: 10.1109/ITSC.2013.6728516

URL:

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6728516&isnumber=6728201

[6]A. S. Badrinath and S. V. Udupa, "Protection circuitry and passive balancing for battery management systems part II," *2017 International Conference on Computation of Power, Energy Information and Commuincation (ICCPEIC)*, Melmaruvathur, India, 2017, pp. 530-535.
doi: 10.1109/ICCPEIC.2017.8290423
URL:

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8290423&isnumber=8290323

[7]Kumagai, Satoru. Electric Power-assisted Bicycle. Yazaki Corp矢崎総業株式会社, assignee. Patent JPH09254861A. 30 Sept. 1997. Print.

[8]Tsang, Peter, and Christian Nicolae Presura. Heart Rate Monitor for Measuring a Heart Rate of a User. Original Assignee Koninklijke Philips Electronics N.V., assignee. Patent WO2013042070A1. 28 Mar. 2013. Print.

9. Appendix [ELW][RKA]

Part	Datasheet
B59707A0120A062	https://product.tdk.com/info/en/documents/data_sheet/55/db/PTC/PTC_OC_SMD_0402_0603_1210_24V_230V.pdf
BMS3004	https://www.onsemi.com/pub/Collateral/ENA1908-D.PDF
DMP2305U	https://www.diodes.com/assets/Datasheets/ds31737.pdf
FK3306010	https://industrial.panasonic.com/content/data/SC/ds/ds4/FK3306010L_E.pdf
LTC6804	https://www.analog.com/media/en/technical-documentation/data-sheets/680412fc.pdf
MLX91210	https://www.melexis.com/-/media/files/documents/.../mlx91210-datasheet-melexis.pdf
PIC16F1789-I/PTTQFP44_MC-L	http://ww1.microchip.com/downloads/en/DeviceDoc/40001675C.pdf
SRH05S05	https://www.xppower.com/Portals/0/pdfs/SF_SRH05.pdf
TP5015	https://www.datasheets360.com/pdf/-9169934687069640514
NXRT15XV103FA1B040	https://www.murata.com/en-us/products/productdata/8796838297630/S0423E.pdf
SSM3J328RLFTR-NO	https://toshiba.semicon-storage.com/info/docget.jsp?did=2429&prodName=SSM3J328R
22-27-2031-03	https://www.molex.com/pdm_docs/sd/022272031_sd.pdf
HC49UP	https://www.digchip.com/datasheets/parts/datasheet/922/HC49UP-pdf.php
ACS770	https://www.allegromicro.com/~/media/Files/Datasheets/ACS770-Datasheet.ashx
IPP045N10N3GXKSA1	https://www.infineon.com/dgdl/Infineon-IPP045N10N3%20G-DS-v02_09-EN.pdf?fileId=5546d4625d5945ed015d9885241104ce
MIC4607	http://ww1.microchip.com/downloads/en/DeviceDoc/MIC4607-85V-Three-Phase-MOSFET-Driver-DS20005610C.pdf
SSD1306 OLED	https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
Polar T34 Heart Rate Transmitter	https://www.marutsu.co.jp/contents/shop/marutsu/ds/1077_Web.pdf

SRH05S05	http://www.farnell.com/datasheets/2563627.pdf?_ga=2.103957001.864562035.1543440146-10472548.1543440146
RK7002B	https://datasheet.octopart.com/RK7002BT116-Rohm-datasheet-68401982.pdf
MOLEX_42840-2224	https://www.mouser.com/datasheet/2/276/0428202224_PCB_HEADERS-772469.pdf