UNIVERSITY OF WASHINGTON
BOTHELL

School of
Science, Technology, Engineering, and Mathematics (STEM)

Department of Electrical Engineering
B EE 495/496
June 2019

Arduino Muscle

Students

Minh Ho	(Electrical Engineering)
Evan Wansa	(Electrical Engineering)
Garrett Padilla	(Electrical Engineering)
Landon Kruse	(Electrical Engineering)

Advisors

Faculty Advisor:	Dr. Randy Kong
Industrial Advisor:	Joe Justice

Abstract

The Arduino Muscle Project is an Arduino Shield ESC developed for 1 kW motors. The shield is designed for brushless DC motors (BLDC) running high currents and voltages powered from LiPo batteries. The project is aimed at providing hobbyists with a motor controller that provides motor speed control and current sensing capabilities. Other sensing capabilities include light, noise, humidity, and temperature. The final product has a target price of less than $100 per board in volumes greater than 10.

Table of Contents

1. List of Acronym Definitions

BLDC- Brushless DC Motor - Commonly used 3 phase electric motor.

Emf - Electromotive force- Internal electrical potential.

ESC - Electronic Speed Controllers

GPIO - General Purpose Input Output - Input and output terminals for the microcontroller for transmitting and receiving data.

IDE - Integrated Development Environment - a software application that provides comprehensive facilities to computer programmers for software development.

LUX - Luminous flux per unit area is a measurement of light intensity visible to the human eye.

MOSFET - Metal Oxide Semiconductor Field Effect Transistor - Voltage controlled switch

NMOS - N-Channel MOSFET

PCB - Printed Circuit Board

PWM- Pulse-width modulation is a modulation process or technique used in most communication systems for encoding the amplitude of a signal right into a pulse width or duration of another signal, usually a carrier signal, for transmission. Although PWM is also used in communications, its main purpose is actually to control the power that is supplied to various types of electrical devices, most especially to inertial loads such as AC/DC motors.

RH - Relative humidity is the amount of water vapor present in air expressed as a percentage of the amount needed for saturation at the same temperature.

PCB - Surface Mount Device

SPI - Serial Peripheral Interface is a form of serial communication. It is how the driver and arduino communicate.

2.Introduction

This project is to create a credit card sized arduino motor controller capable of pushing up to 50 amps or 1 kW of power to a 3-phase motor and read back to the user the amount of current and power draw from the motor. In order to expand the capabilities for use the controller will also sense things such as light, temperature from 0-50 °C, (RH)relative humidity reading between 20-80% with 5% accuracy, and noise from 20-20KHz. And when ordered in bulk of 10+ units, the motor controller will cost less than $100.

2.1 Background and Motivation

In a relatively short time, the cost of powerful and efficient three-phase motors has dropped rapidly. Three phase motors are more power efficient than the equivalent single phase brushed motors. These motors have many applications such as use in drones, robotics, and even small scale manufacturing. Control of these powerful motors exists in the market today in the form of ESCs, or Electronic Speed Controllers. These are generally hobby level for drones or RC cars, and are set up for control from a radio receiver. ESCs are compact and effective at what they do, but require external control and direct interfacing with the device can require extra accessories to change settings.

Another product that has gained wide popularity and a similar drop in price is microcontrollers. Microcontrollers allow for the control of many devices and circuits through input and output pins and a range of communication protocols. The flexibility and low cost of microcontrollers allows for creation of task specific devices. Arguably the most popular and approachable microcontroller on the market today is the Arduino.

Low cost, high power motors and inexpensive microcontrollers together would be a good thing for rapid prototyping, small to medium robotics, or drones. To bring these products together would require an interface to control and respond to the motor with a microcontroller. This is similar to a motor controller board called the ODrive V3.5 however, this projects price will be slightly lower and have the capabilities of driving more power. This project is meant to bridge the gap between powerful motors and cheap microcontrollers by creating a hobby priced, high power motor shield. The direct connection with a well known microcontroller would set our product apart and make the use of high power motors much more accessible.

3.Project Design

In this section, the project will be described at a high level including discussion about the phases of the project.

3.1 Project Block Diagram

The motor controller driver acts as a modular component to the Arduino microcontroller as a shield. The Arduino passes serial output commands to the shield that drives the motor.

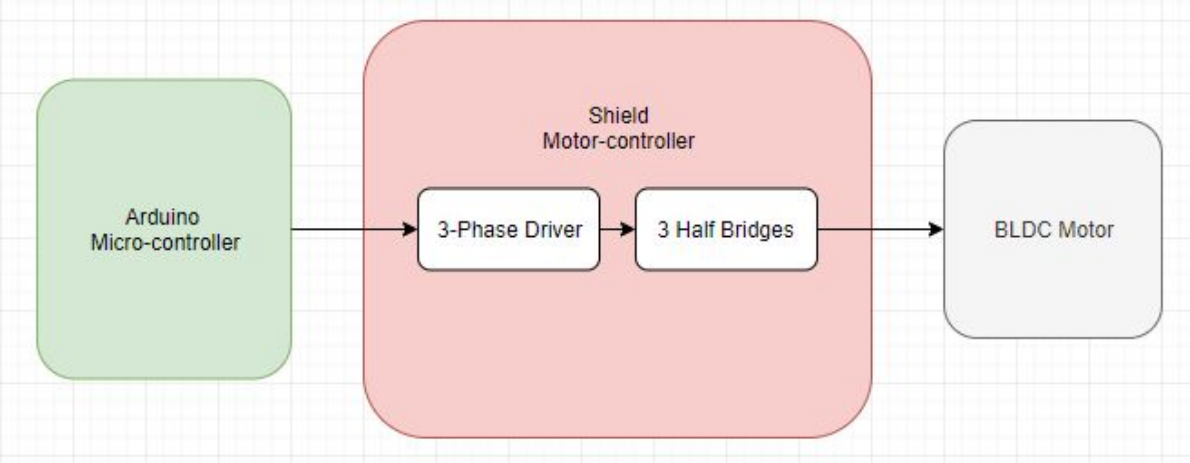

Figure 3.1: Top level diagram of driving a BLDC motor

The motor controller's main functionality is to control the motor and measure it's current and power draw. Controlling the motor means to control the speed and direction of rotation. Sensors read environmental data and feed it back to the Arduino. The sensors directly interact with the Arduino, but are mounted onto the motor-controller PCB to provide additional general functionality for future design paths.

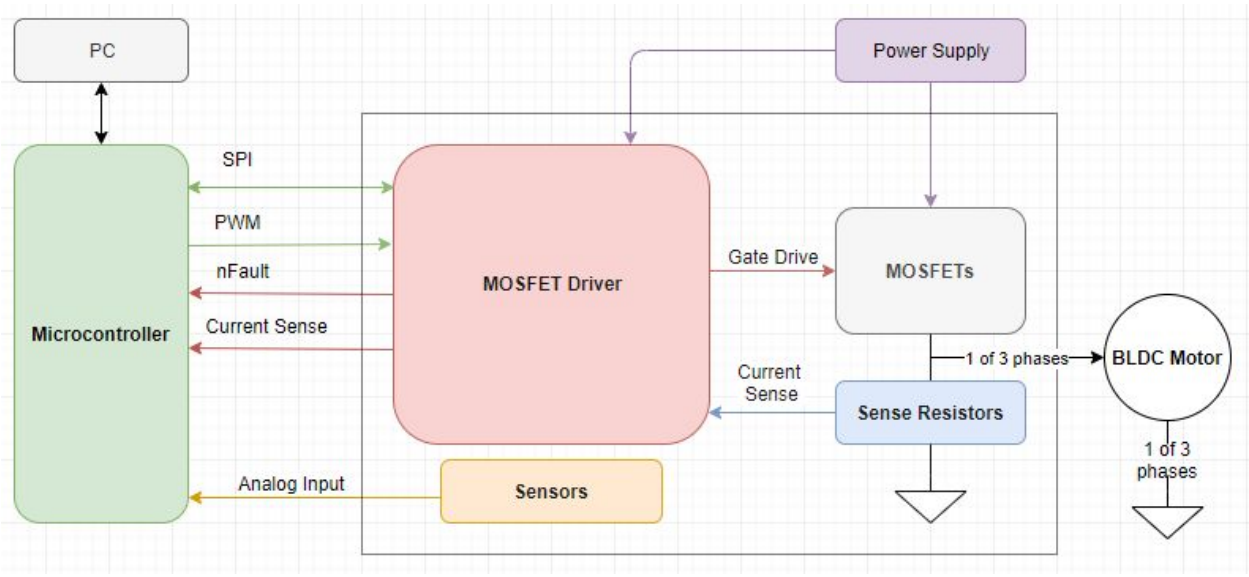

Figure 3.2 : Overall project block diagram

3.2 Arduino Mega

The Arduino Mega is a microcontroller board that includes 54 digital input/output pins. It is the foundation to this project and determines the variety of capabilities that can be done with the motor

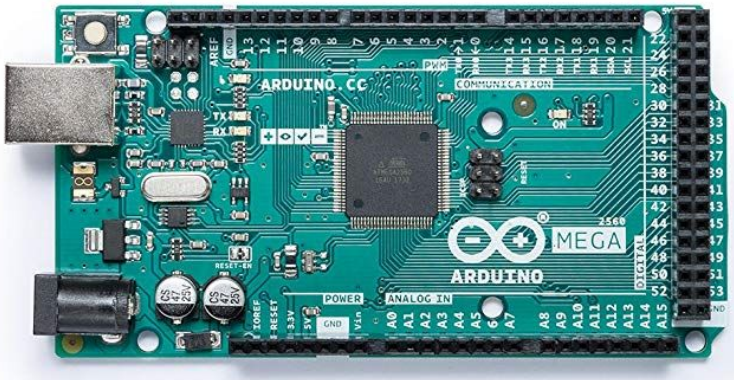

Figure 3.3: Arduino Mega 2560 REV3 board

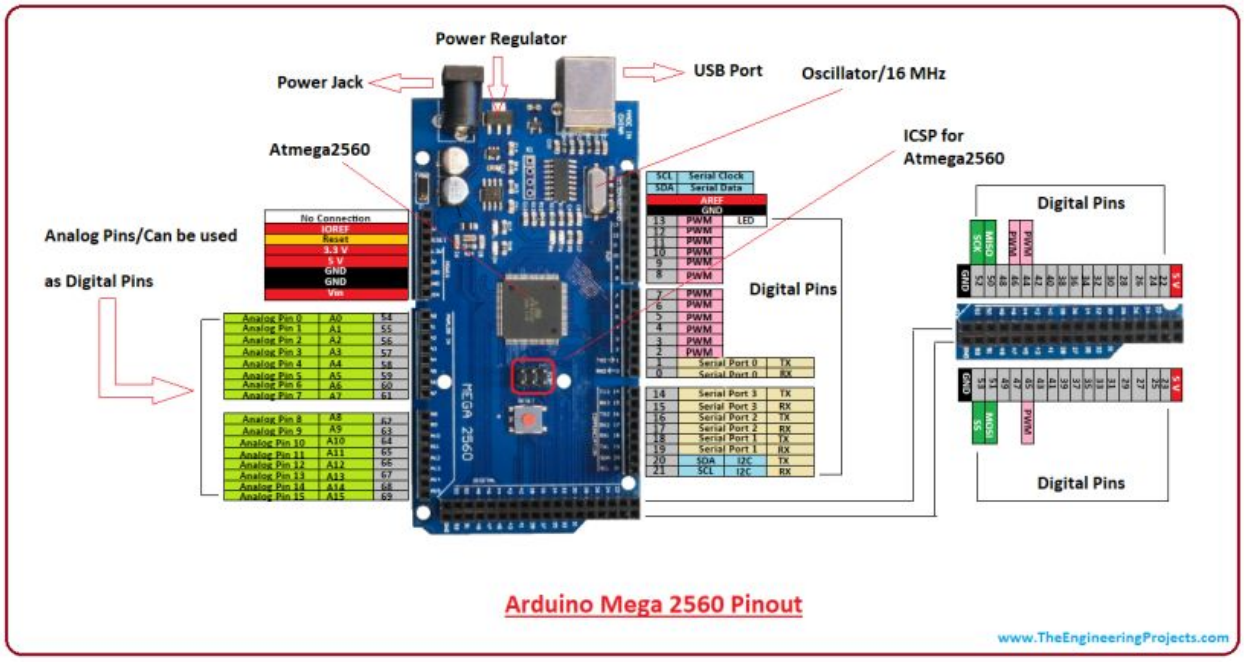

Figure 3.4: Arduino Mega 2560 Pinout

3.3 3 Phase BLDC

The 3 phase BLDC motor assigned for this project by the customer is the KB 44-65. It has no data sheet, manufacturer name, or really any information on its specifications. However, the project still managed to navigate these complications. The key information given was that the max power was over 1kW, the max current was 51A, and it had no hall effect sensors.The measured resistance of the motor between two phases is ~3 mΩ.

Figure 3.5: KB 44-65 Brushless Inrunner 1520KV

Kv: **1520 rpm/v**
Max Current: **51A**
Max Power: **1130W**
Cell Count: **4 ~ 6S LiPoly**

3.4 Hardware Description

Driving the BLDC motor requires a half bridge for each phase of the motor. Two of the three phases will be active at any instance in time that the motor is running, one high side for the first phase and low side for the second. At the end of each phase is a sense resistor that the driver uses to measure current through the motor.

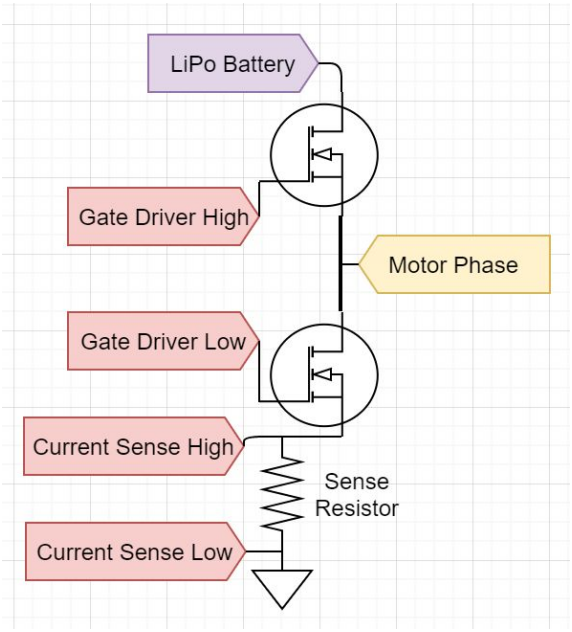

Figure 3.6: Diagram of a half bridge for each phase of the motor

The sensors for the shield take readings from the environment and convert it to an analog or digital output to the microcontroller GPIO ports.

3.5 Software Description

The software configures the Driver, Arduino, and various sensors to communicate and run as a complete system. Beyond initializing the various arduino pins, it also programs the driver's functionality. In order for proper driver programming, SPI (serial communication) between the arduino and driver had to be configured and fine tuned. And lastly, the motors position had to be simulated in the software in place of hall effect sensors. Cycling through the six different motor states or positions allowed the motor to make full rotations.

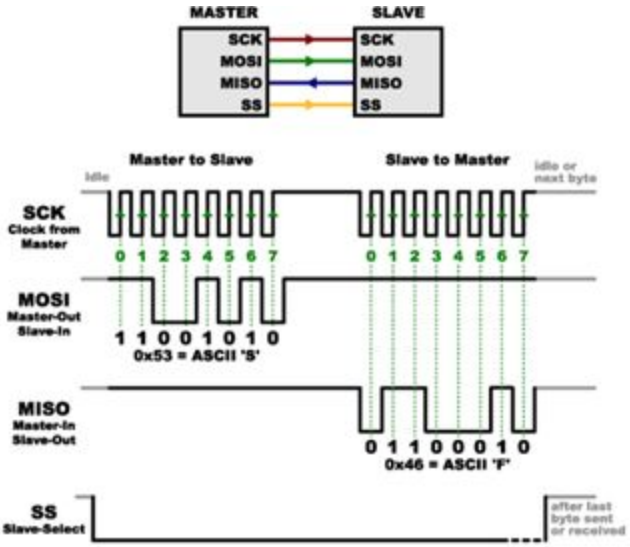

Figure 3.6: Block Diagram of the serial communication lines

SPI was the key to the software. Communication with the driver would not have happened if this was not configured. Without the driver, driving any motor would have been near impossible.

4.Hardware Design

In this section, the project hardware will be described with the circuit schematic and PCB for the driver prototype board and the final motor controller prototype.

The heart of the project is the driver selected to drive the motor, the DRV8323RS 6-60 V Three-Phase Smart Gate Driver from Texas Instruments. The driver is capable of driving 6 N-Channel MOSFETs for driving the BLDC motor with current sensing for each phase. A charge pump is used to drive 3 High-Side and 3 Low-Side NMOS.

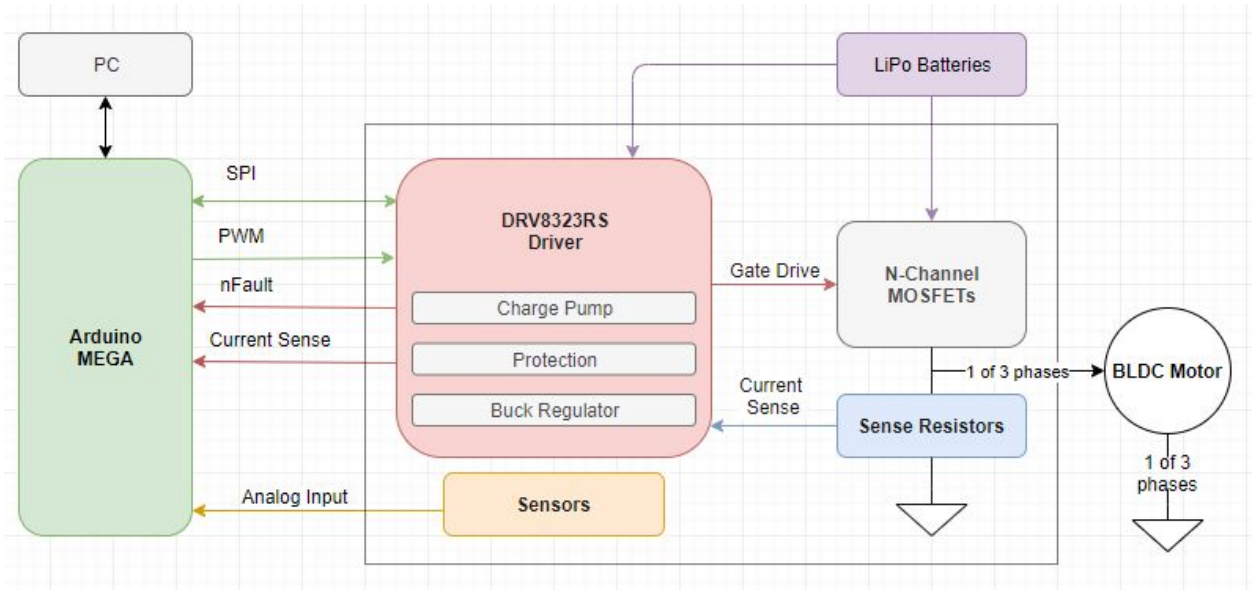

Figure 4.1: Overall Project Block Diagram

The driver supports 100% PWM duty cycle for maximum speed control. It also comes with integrated protection features including ones necessary for dealing with high voltages and currents:

- VM Undervoltage Lockout
- Charge Pump Undervoltage
- MOSFET Overcurrent Protection
- Gate Driver Fault
- Thermal Warming and Shutdown
- Fault Condition Indicator

Taking into consideration high voltages and currents, the project was divided into 2 prototypes for development. The first prototype was an implementation for driving low voltages and currents. This design was used to test and debug the circuit and code when interfaced with the project software as well as any sensors. The second prototype was designed to run high currents from 20-50 A to reach the 1 kW target.

4.1 Driver Prototype

The first prototype is designed with testing the driver functionality in mind. The MOSFETs were built on a separate breadboard to make it easier to test each one. Also the prototype was designed to use the bench power supply in lab to limit the variable of controlling/ limiting voltage and current to the board.

4.1.1 Block Diagram

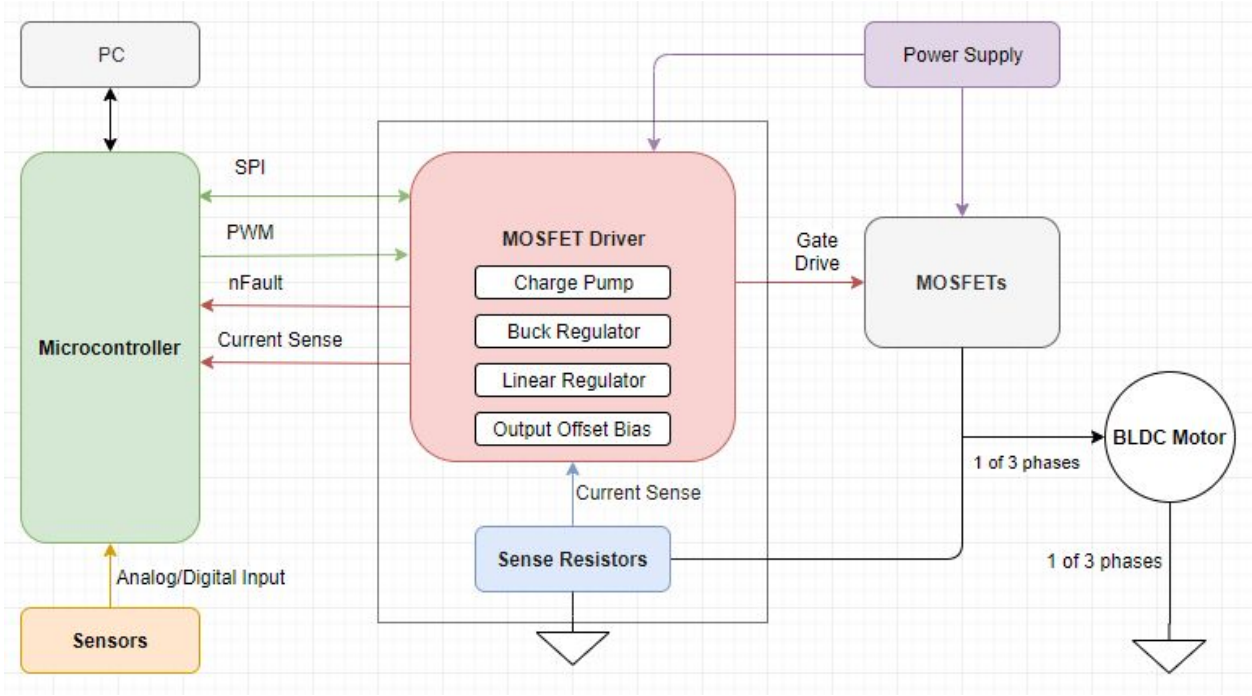

Figure 4.2: Prototype board with sensors & MOSFETs off PCB

The first prototype is designed and assembled with the:

- Charge Pump
- Buck Regulator
- Linear Regulator
- Output Offset Bias

4.1.2 Schematic

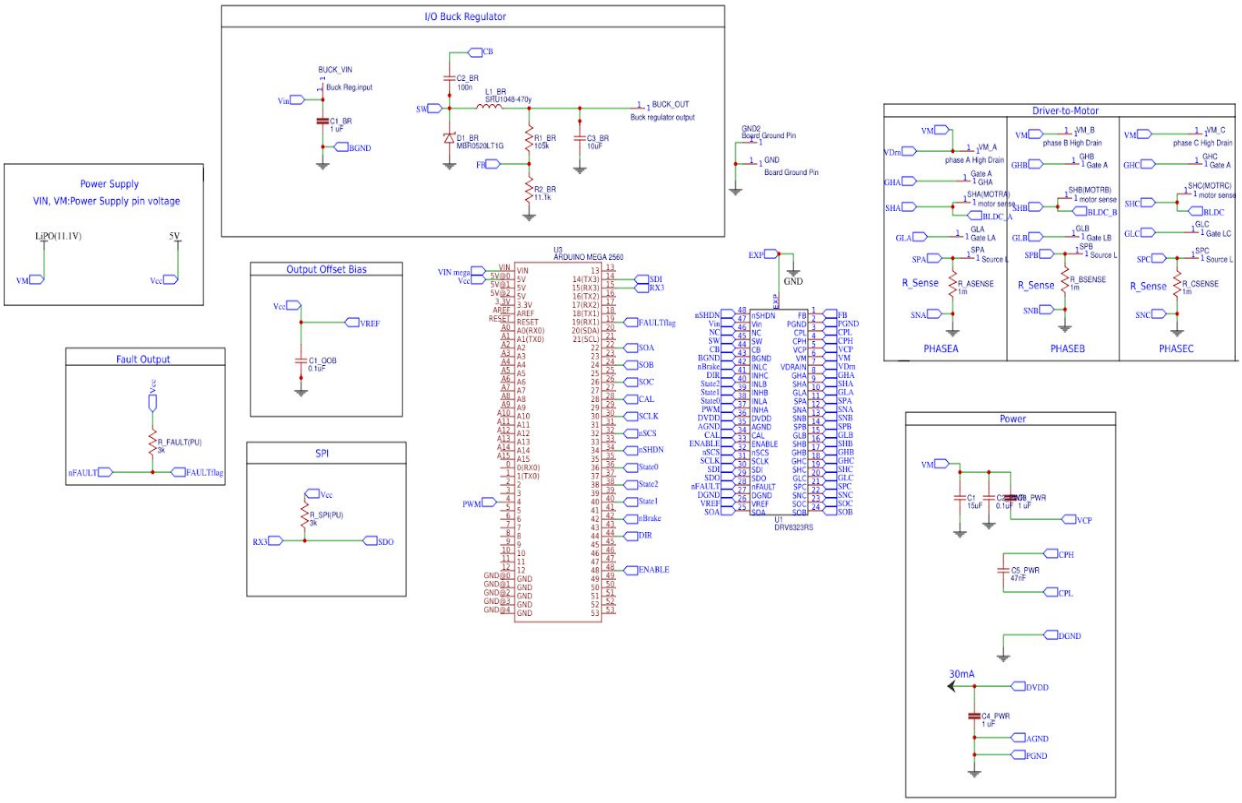

Figure 4.3: Complete schematic of the lower power driver

4.1.3 PCB Layout

The low power PCB features numerous test pins for testing and debugging seen on the left side of the schematic. Having the MOSFETs off of the PCB (figure 4.5) eliminated any issues we would have with SMD. All of the connections to the gates, sense resistors and motors were made with jumper cables to a breadboard.

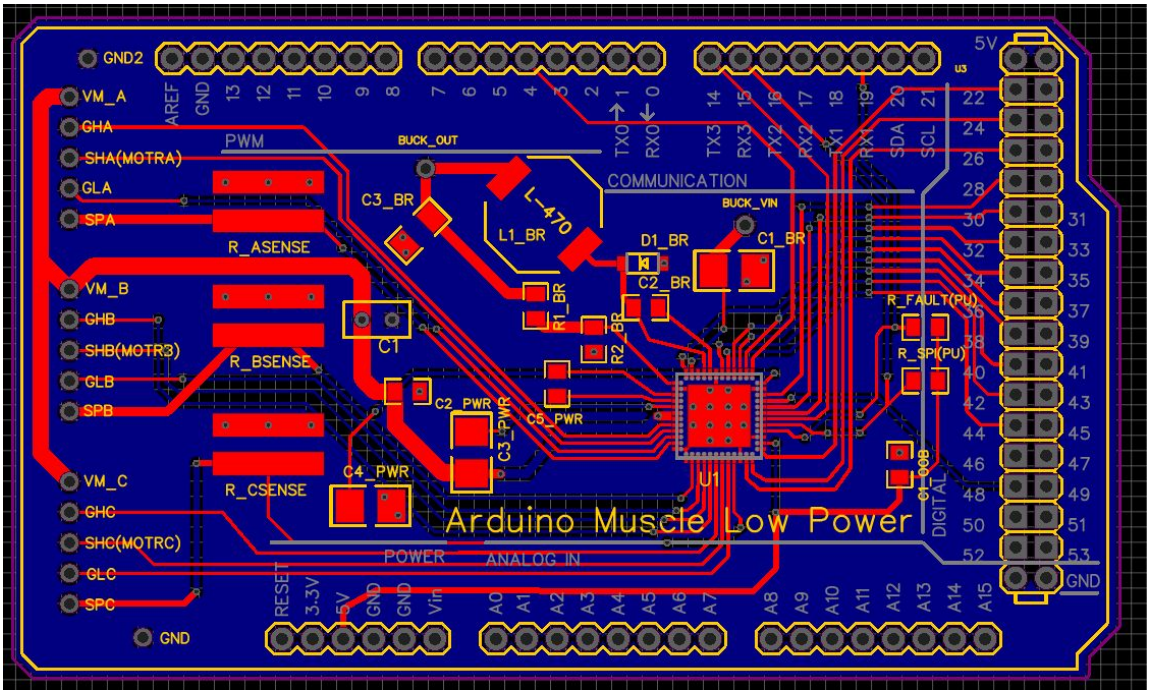

Figure 4.4: Driver test prototype PCB

This was designed in EasyEDA and then manufactured via JLCPCB. The overall dimensions are 3.63x2.15 inches.

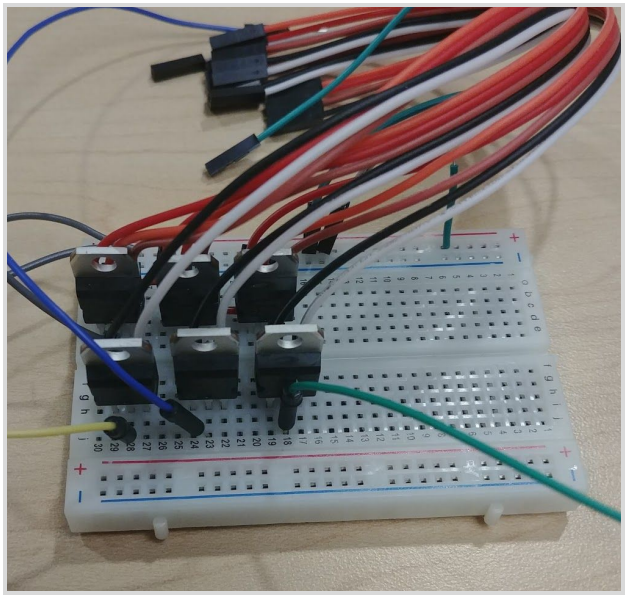

Figure 4.5: Driver test prototype 3 phase inverter

4.1.4 DRV8323RS

The DRV8323RS is an integrated gate driver for three-phase applications. It provides three half-bridge gate drivers that drive high-side and low-side N-channel power MOSFETs. The device generates the gate drive voltages by an integrated charge pump that drives the high-side MOSFETs and a linear regulator for the low-side MOSFETs. Based off figure x.xx, the controller (Arduino Mega) will communicate configured instruction to the driver using PWM to run the smart gate driver to the N-Channel Mosfet bridge and run the three-phase BLDC motor. The driver will deliver back to the controller current and power consumed by the motor and if any faults occurred while it's running.

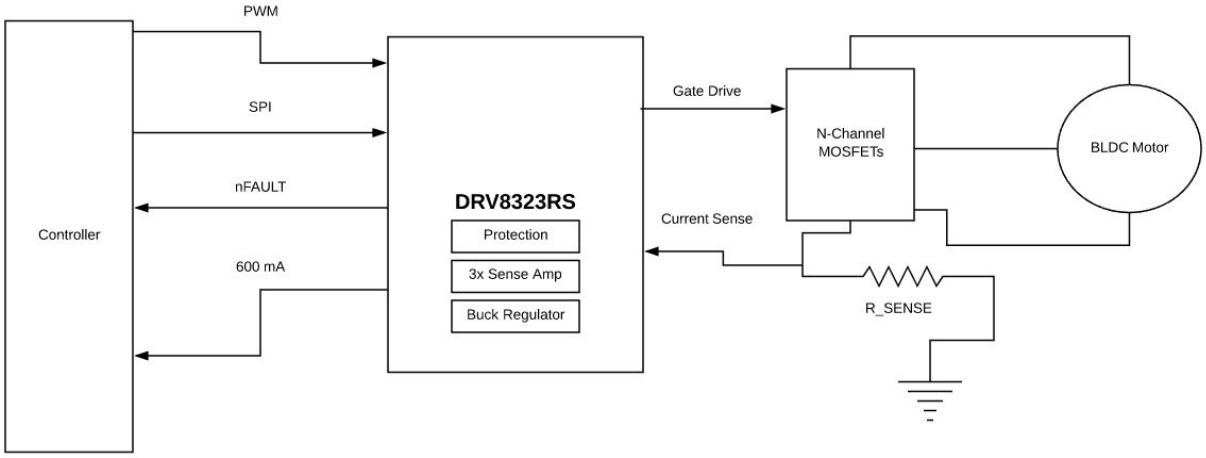

Figure 4.6: DRV8323RS Simplified schematic

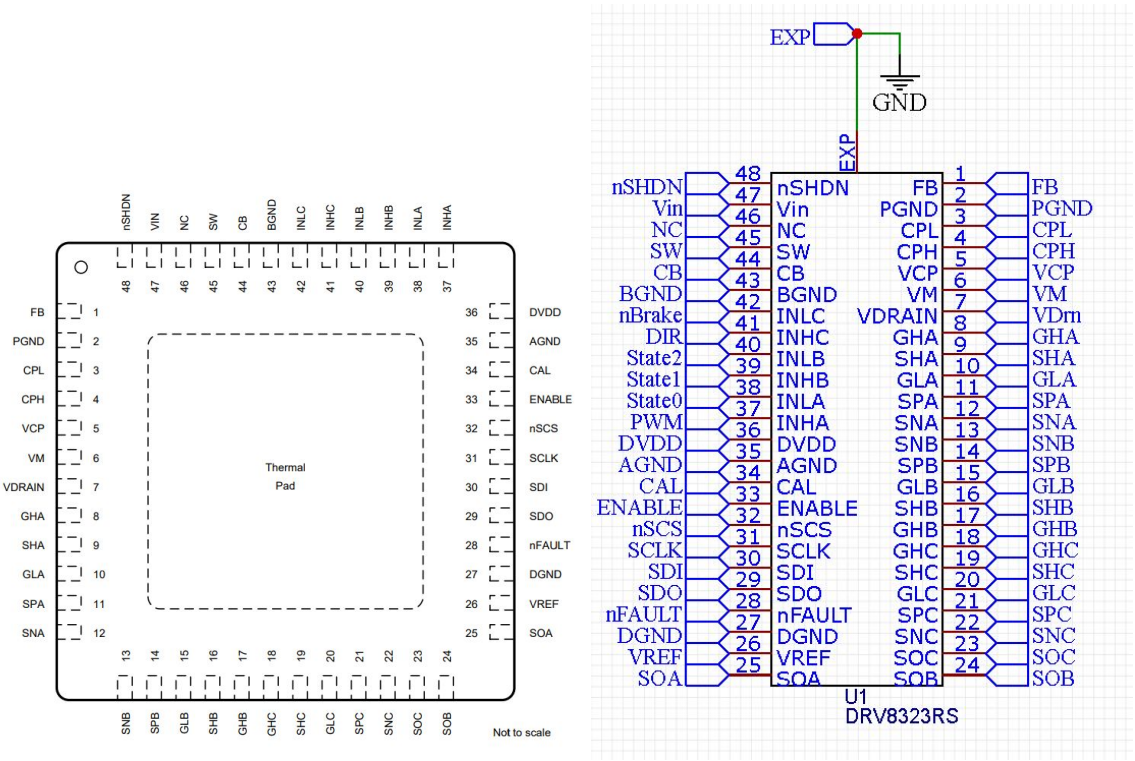

Figure 4.7: DRV8323RS Driver pin output

Pin #	Abbrev	Description
48	nSHDN	Buck shutdown input. Connect with the Arduino Mega.
47	VIN	Buck regulator power supply input.
46	NC	No internal connection.
45	SW	Buck switch node. The pin is connected to the inductor, diode, and CB bootstrap capacitor.
44	CB	Buck regulator bootstrap input.
43	BGND	Buck regulator ground.
42	INLC	Low-side gate driver control input (nBrake) from the Arduino Mega pin #42. This pin controls the output of the low-side gate driver for C.
41	INHC	High-side gate driver control input (DIR) from the Arduino Mega pin #44. This pin controls the output of the high-side gate driver for C.
40	INLB	Low-side gate driver control input (State2) from the Arduino Mega pin #38. This pin controls the output of the low-side gate driver for B.
10	INHB	High-side gate driver control input (State1) from the Arduino Mega pin #40. This pin controls the output of the high-side gate driver for B.
11	INLA	Low-side gate driver control input (State0) from the Arduino Mega pin #36. This pin controls the output of the low-side gate driver for A.

12	INHA	High-side gate driver control input (PWM) from the Arduino Mega pin #4. This pin controls the output of the high-side gate driver for A.
13	DVDD	3.3-V internal regulator output. It can source up to 30 mA.
14	AGND	Device analog ground.
15	CAL	Amplifier calibration input. Set logic high to internally short amplifier inputs and perform auto offset calibration (not used).
16	ENABLE	Gate driver enable. The Arduino Mega pin #48 send a pulse to the driver for activation.
17	nSCS	Serial chip select.This pin connects with the Arduino Mega pin #32 for serial interface communication.
18	SCLK	Serial clock input. Serial data is shifted out and captured on corresponding rising and falling edge with the Arduino Mega pin #30.
19	SDI	Serial data input. Serial data is captured on the falling edge of the SCLK pin. Also ties with the Arduino Mega (TX3) for serial communication.
20	SDO	Serial data output. Data is shifted out on the riding edge of the SCLK pin. The pin is tie with an external pullup resistor and the Arduino Mega (RX3) for serial communication.
21	nFAULT	Fault indicator output. This pin is pulled logic low during a fault condition with an external pullup resistor. the pin connected to the Arduino Mega pin # 19 (FAULTflag).
22	DGND	Device ground.
23	VREF	Current sense amplifier power supply input and the reference. *
24	SOA	Current sense amplifier output. This pin connects with the Arduino Mega pin #22.
25	FB	Buck feedback input.
26	PGND	Device power ground.
27	CPL	Charge pump switching node.
28	CPH	Charge pump switching node.
29	VCP	Charge pump output.
30	VM	Gate driver power supply input.
31	VDRAIN	High-side MOSFET drain sense input.
32	GHA	High-side gate driver output.
33	SHA	High-side source sense input.
34	GLA	Low-side gate driver output.
35	SPA	Low-side current shunt amplifier input.
36	SNA	Current sense amplifier input.
37	SNB	High-side source current input.

38	SPB	Low-side current shunt amplifier input.
39	GLB	Low-side gate driver output.
40	SHB	High-side source sense input.
41	GHB	High-side gate driver output.
42	GHC	High-side gate driver output.
43	SHC	High-side source sense input.
44	GLC	Low-side gate driver output.
45	SPC	Low-side current shunt amplifier input.
46	SNC	Current sense amplifier input.
47	SOC	Current sense amplifier output.This pin connects with the Arduino Mega pin #26.
48	SOB	Current sense amplifier output.This pin connects with the Arduino Mega pin #24.

Table 4.1: DRV8323RS Driver pin output described

Charge Pump

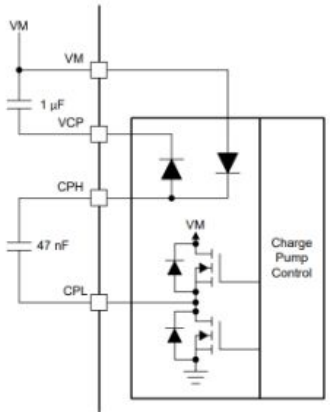

Figure 4.8: Charge Pump Architecture

Charge pumps are DC-to-DC converters used to raise or lower voltage. The charge pump in the DRV8323RS has a double charge pump which operates from the power supply input. It will let the gate driver bias the high-side MOSFET gate by the source across the input supply voltage range. It is also regulated to fix the power supply input plus 11V and set the output current of 25 mA. If the input power supply is less than 12 V, the charge pump begins to operate in full doubler mode which generates two times of the decremented 1.5 V of the input power supply when unloaded. The pump will monitor for any events of under voltage to prevent any under-driven MOSFET conditions.

It is required that the charge pump has a 1-uF, 16-V ceramic capacitor between the VM and VCP pins to act as the storage. A 47-nF ceramic capacitor is required between the CPH and CPL pins to act as a flyer capacitor. The low-side gate creates the voltage supply using a linear regulator from the input power supply. The linear regulator output is fixed at 11V and support 25 mA for output.

Buck Regulator

A buck regulator is used in the design to step down the voltage from our motor power supply from 22.2V to 5V. This is already integrated to the MOSFET driver. The buck regulator is used to power external logic circuits. It can improve performance during line and load transients by implementing a constant-frequency current-mode control scheme. This feature is optional.

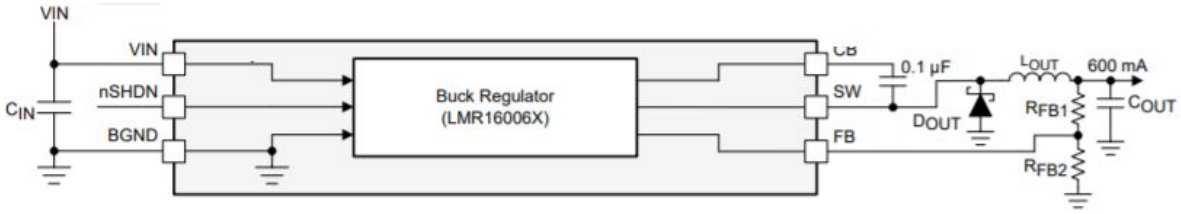

Figure 4.9: Buck Regulator Block Diagram

Auto Offset Calibration

Auto Offset Calibration minimizes DC offset by the current sense calibrating through the CAL pin. When it is enabled, the inputs to the amplifiers are shorted, the load is disconnected, and the gain of the amplifier is changed to the 40 V/V setting.The amplifier goes through automatic trim routine to minimize input offset. After 100 microseconds, the inputs of the amplifier still shortens, the load still disconnected, and the gain stays at 40 V/V if further offset calibration is done by external controller. The offset completes the calibration when the CAL pin is pulled low, the gain then returns to the original gain setting.

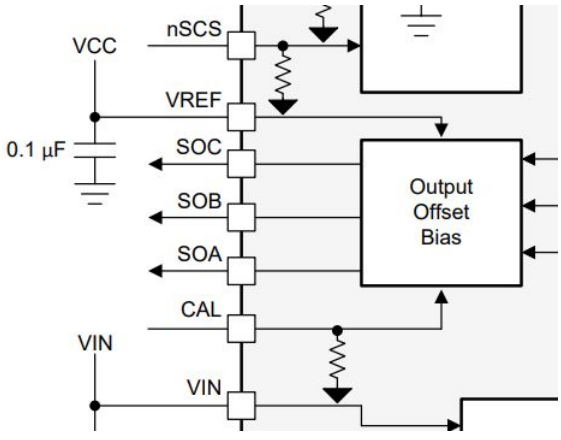

Figure 4.10: Output Offset Bias

Power Schematic

This part of the schematic applies the components and power supply needed to run the VCP Charge Pump, VGLS Linear Regulator, and DVDD Linear Regulator off the DRV8323RS. The description below are the list of components functionality for the power circuit schematic:

- 0.1 uF capacitor: X5R, 50-V Rated and 0805 surface mount.
- 1 uF capacitor (x2): X7R, 50-V Rated and 1210 surface mount.
- 15 uF capacitor: X7R, 16-V Rated and through hole capacitor.
- 47 nF capacitor: X7R and 0805 surface mount.
- Lug: 2 AWG High AMP PCB Wire Lug 2-14 AWG.

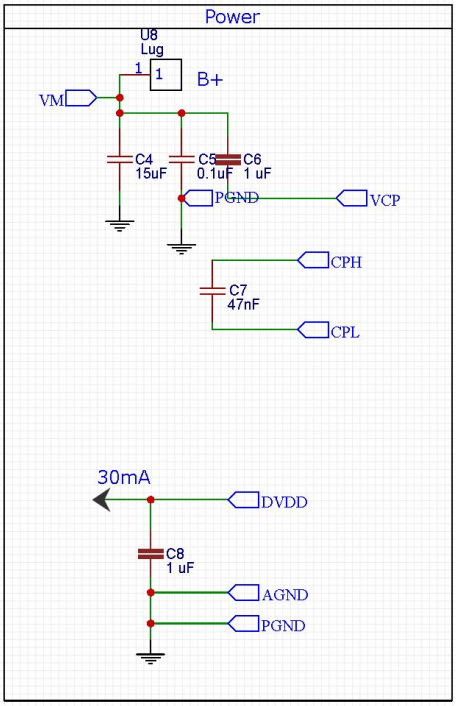

Figure 4.11: Power Block

SPI Schematic

The SPI schematic is designed by tying RX3 pin from the Arduino, 3k resistor and the SDO pin from the DRV8323RS. The SDO pin is the data output to the RX3 which includes 3k as a pull up resistor.The pull-up resistor to ensure functionality for the SPI is:

- 3kΩ resistor: .125-W Rated and 0805 surface mount

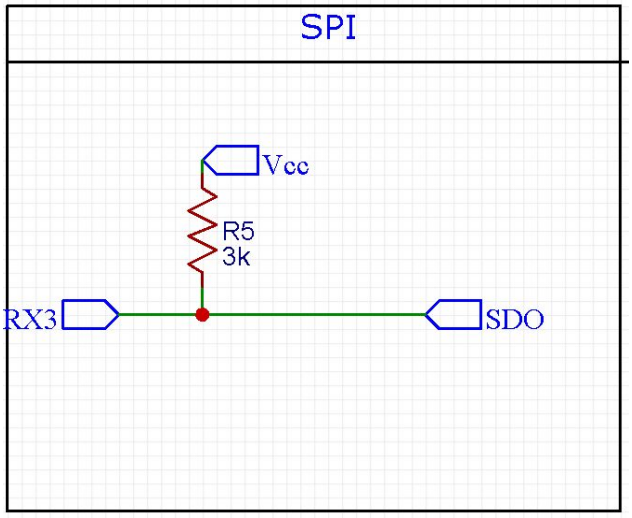

Figure 4.12: SPI Input Block

Output Offset Bias Schematic

The output offset bias ties VREF with VCC and a 0.1 uF capacitor. The VREF pin is a current sense amplifier supply input and reference. The component used for the output offset bias schematic:

- 0.1 uF capacitor: X5R, 50-V Rated and 0805 surface mount.

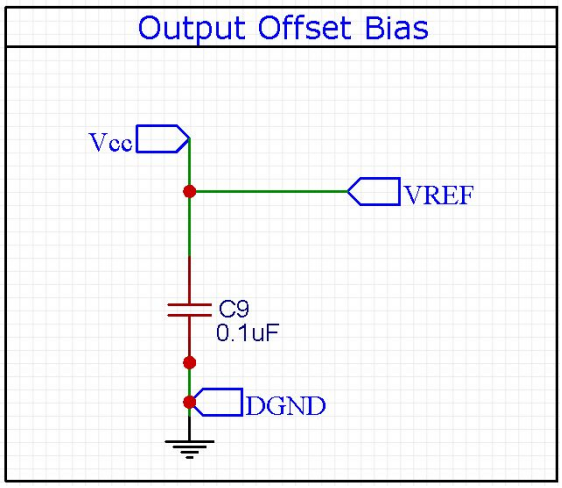

Figure 13: Output Offset Bias Block

I/O Buck Regulator Schematic

The buck regulator, LMR16006, is integrated in the MOSFET driver. The components were picked by using a customized tool that Texas Instruments provides for optimization based off the input and output for our design.The description below is the list of components for the I/O buck regulator schematic:

- 1 uF capacitor: X7R, 1210 package.
- 100 nF capacitor: 0805 package.
- MBR0520LT1G diode: Bootstrap recharge diode.
- SRU1048-470y inductor: 47 uH.
- 10 uF capacitor: 1206 package.
- 105kΩ resistor: 0805 package.
- 11.1kΩ resistor: 0805 package.

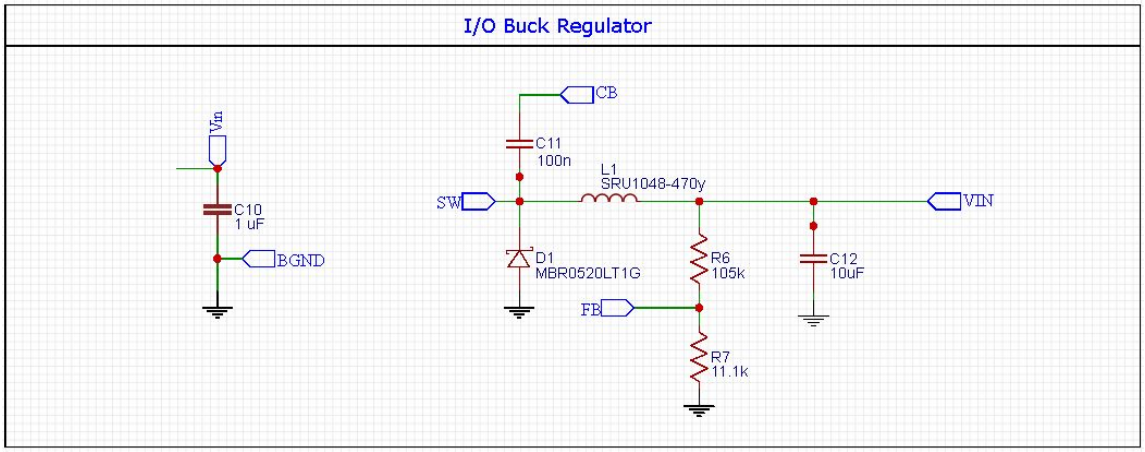

Figure 4.14: Input & Output Buck Regulator Block

4.2 Final Prototype

4.2.1 Block Diagram

The 2nd prototype PCB was designed with high current in mind from 20-50 A. Taking this design constraint into consideration, high AMP PCB wire lugs rated for 115 A are used to deliver high current through 10 AWG wire rated for up to 55 A. The power now comes from two 3S LiPo batteries in series for 22.2 V connected through xt60 connectors. In this design the buck regulator is removed for safety concerns.

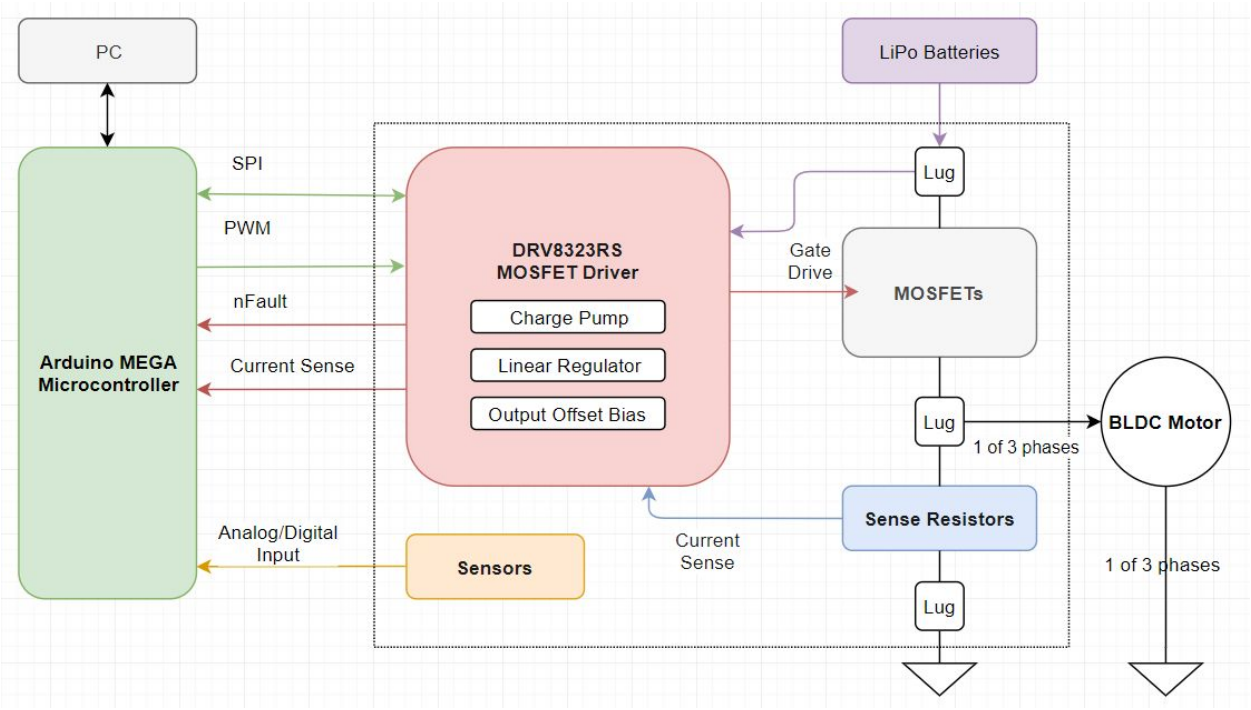

Figure 4.14: 2nd Prototype Block Diagram

4.2.2 Schematic Changes From Driver Prototype

Bypass capacitors are added from the power supply (+) from the lug on the PCB to the negative power supply (-) through the ground plane. Lugs are added to connect to each phase of the motor using 10 AWG wire. They are also added for connections from the positive and negative of the LiPo batteries. The power MOSFETs used are changed to SMD and are rated for high current.

- Power MOSFET N-channel (x6): 40 Vds, 3 mΩds, 120 I_D.
- 2mΩ resistor (x3): 6-W Rated, Current Sense.
- Lug (x5): 2 AWG High AMP PCB Wire Lug 2-14 AWG.
- 10uF capacitor: Bypass capacitor, X5R, 50-V Rated and 0805 surface mount.

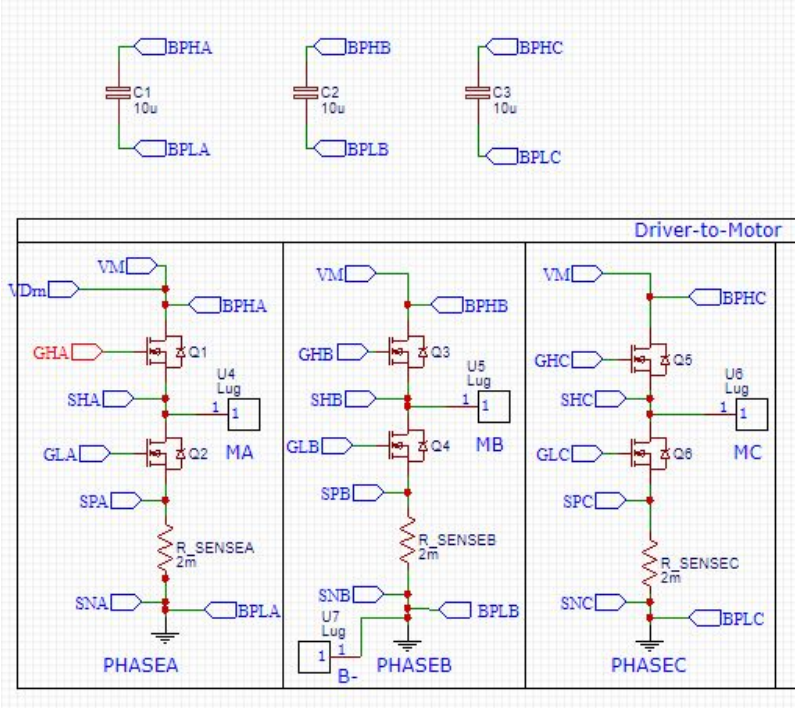

Figure 4.15: Bypass capacitors and lugs added for each phase

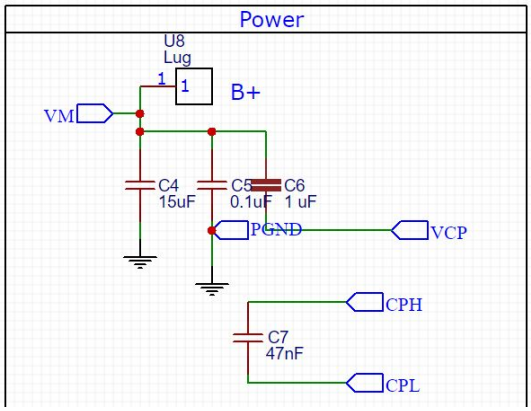

Figure 4.16: Lugs also added for LiPo battery power supply

A reset switch was added in the bottom right corner of the final prototype board. This is an added safety feature which allows the user to quickly wipe or stop the driver. When changing power supplies from the bench power supply to the LiPo batteries this reset switch added ease and safety when wiping the PWM signal so as that too much PWM wasn't applied immediately when hooking up the new power source.

4.2.2 PCB Layout

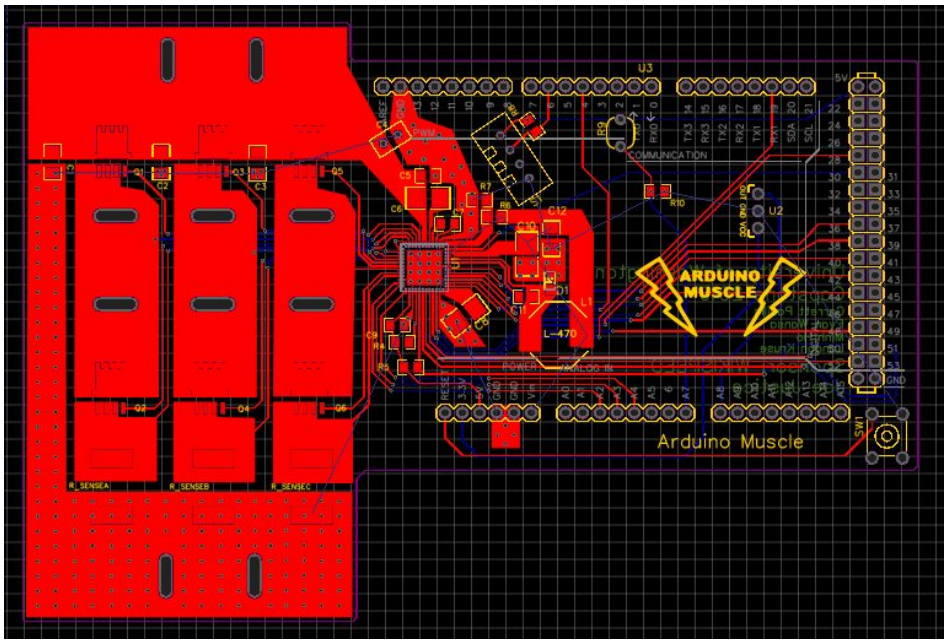

Figure 4.17: Final Prototype PCB layout

The final prototype PCB layout incorporates the MOSFETs onto the board as well as lugs for battery +/- as well as outputs for each phase of the motor. All sensors (Temperature/humidity, light and sound) are integrated into the PCB as well. Connections were fixed from the driver prototype board as well. As thermal management was a concern with the higher current version, areas where heat could flow into the ground plane and dissipate were crucial. Figure 4.18 shows the ground plane and the traces that run through it, the traces were laid out to avoid isolating the driver thermal pad and vias. This connection layout lets heat flow to other cooler portions of the PCB to help keep the driver chip cooler as it operates.

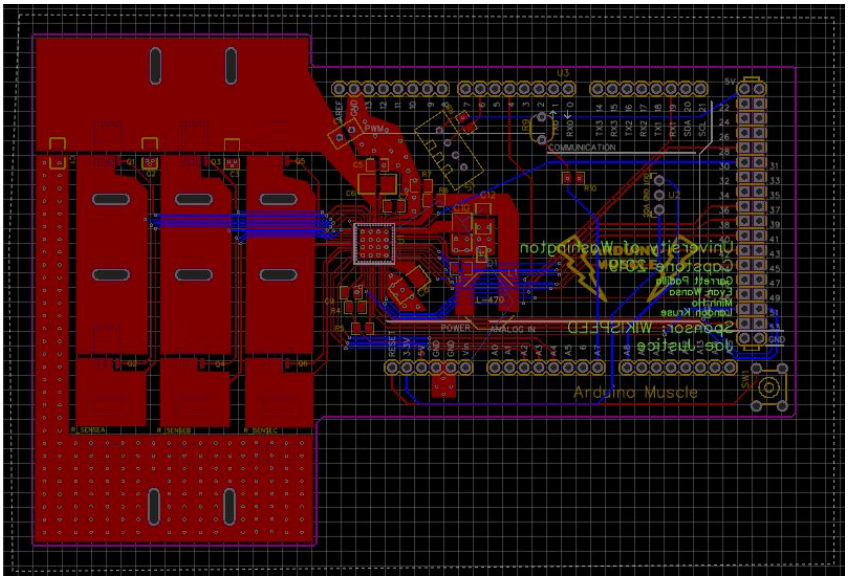

Figure 4.17: Final Prototype PCB layout (bottom layer - no copper fill)

Table 4.1 - Resistivity calculation (predicted worst case on each phase trace)

Material	**Resistivity**	**Area**	**Length**	**Resistance**	**Power dissipated (@ 50A)**
Copper	1.68E-8	1302mil^2	295mil	3.806nΩ	9.515μW
Solder	1.759E-7	1302mil^2	295mil	39.85nΩ	99.64μW

Even though solder is ten times worse as a conductor in terms of resistance (and therefore power loss) than copper, a cheap and fast way to increase the current load of a trace is to expose the copper trace and add a layer of solder. All high current traces were exposed and solder was added to increase the max current load. Table 4.1 shows the worst case scenario (as we expect more solder than this) of a 2.8mil layer added to the 2.8mil layer of copper.

4.2.3 Component Changes From Driver Prototype

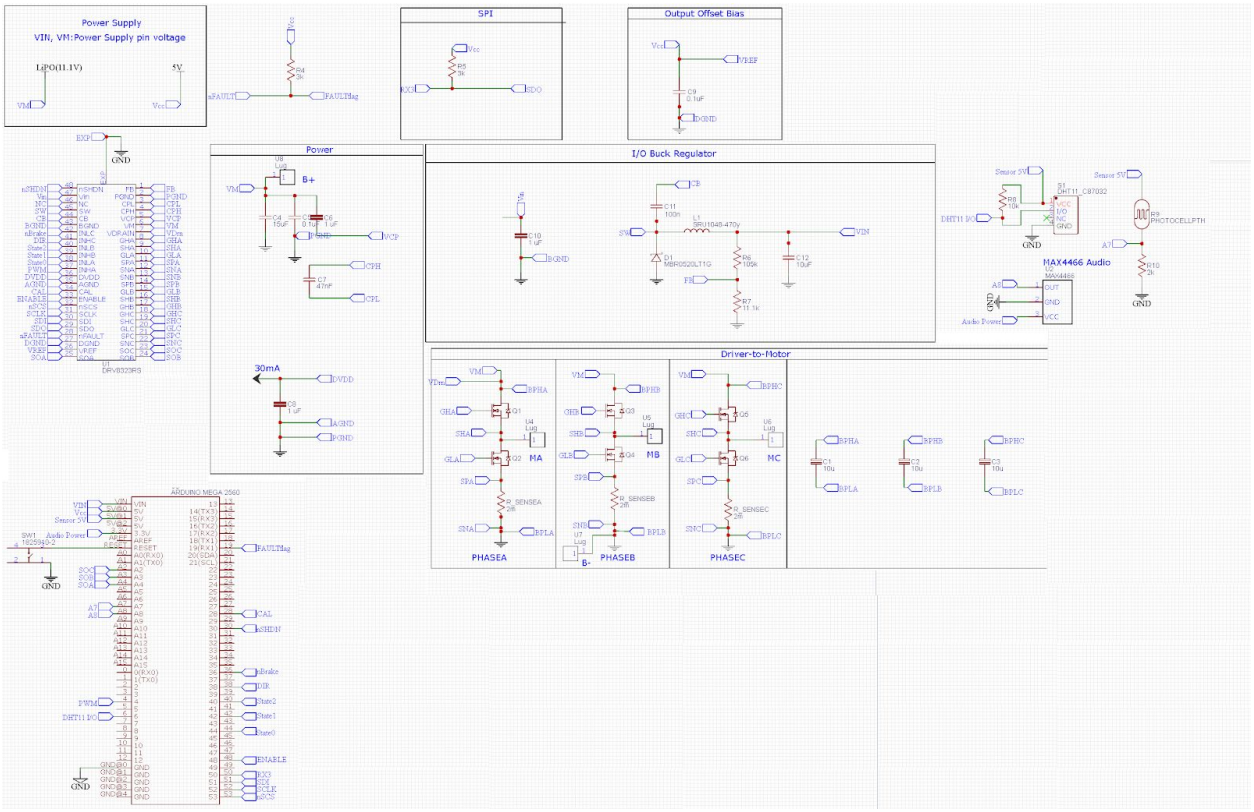

Figure 4.19: Complete schematic of the High power driver

DHT11

The DHT sensor will measure both temperature and humidity. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and outputs a digital signal on the data pin.The a humidity sensor make real time measurements of relative humidity between 20-80% with 5% accuracy. The thermosensor make real time measurements of temperature between 0-50℃ The sensor can be used based off what the user wants to do with it. The description below are the list of components functionality for the DHT11 schematic:

- 10kΩ resistor:Surface mount 0805 package.
- DHT11

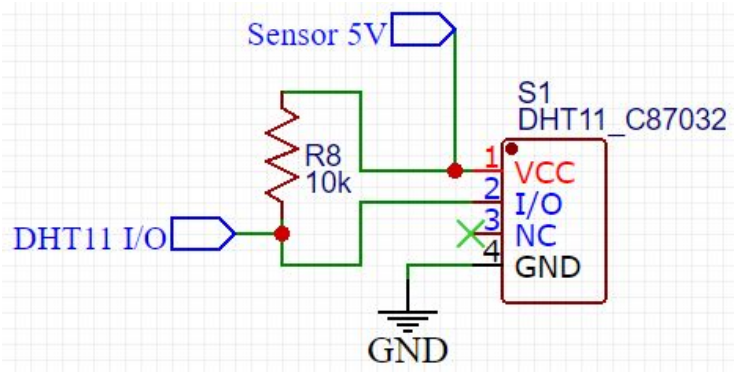

Figure 4.20: DHT11 Schematic

MAX4466 Audio Sensor

The MAX4466 audio is a micropower op amps optimized for microphone preamplifier. The microphone make real time measurements of noise levels between 20-20kHz. The description below are the list of components functionality for the MAX4466 schematic:

- MAX4466

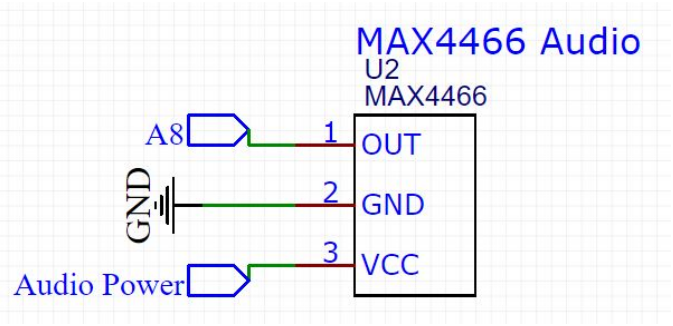

Figure 4.21: MAX4466 Schematic

Photocell

The photo sensor circuit utilizes a photoresistor to make real time measurements of ambient light by sensing light from 400 to 700 nm with a peak sensitivity of 520 nm; these are the ranges the human eyes can pick up (visible light spectrum). The photo sensor circuit is set up as a voltage divider to detect from 0 lux (no light, max resistance for the photoresistor) to above 4000 lux (the power of a cell phone flashlight shone directly into the photoresistor). This provides a swing between 0V and 5V for the arduino to convert into a value.

- 2kΩ resistor:Surface mount 0805 package.
- Photocells

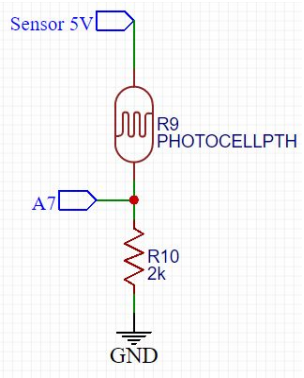

Figure 4.22: Photocell Schematic

2x(3S) LiPo Batteries

The power supply for the final prototype was chosen to be ~22V based on the motor specification due to our specification to drive a 1 kW motor. LiPo batteries are very energy dense and capable of driving much more current than required for this project. LiPo batteries are built from cells each ~3.7V and the number of cells is designated by the battery “S” value. LiPo batteries also have a “C” value which is the safe discharge current value. The chosen battery configuration was picked to reduce cost (a 6S battery is expensive vs 2 3S batteries) and was easily configured to establish a voltage of 22V-24V by connecting the batteries in series. The batteries were also chosen for their capacity (designated in mAh) to ensure ample supply of power for testing.

Figure 4.23: 3S LiPo Battery for testing the final prototype

5.Software Design

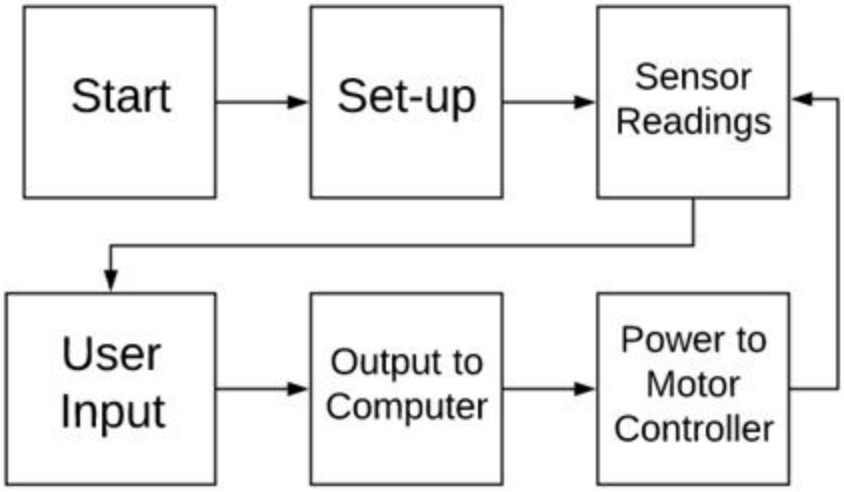

Figure 5.1: Block Diagram of the software design

The software can be broken up into several main blocks. Each block concerning the software will be covered more in depth in the following sections.

5.1 Set-up

The software setup is what had to be configured first in order for the rest of the code to function properly. This consisted of configuring the different digital and analog connections between the various hardware on the shield and Arduino and the setup for the Driver.

5.1.1 SPI

The major piece of software in the set-up was configuring the SPI. This consisted of four digital communication pins on the Driver connecting to the appropriate communication pins on the Arduino Mega. The SPI is used to set device configurations, operating parameters, and read out diagnostic information from the MOSFET driver to the Arduino. The MOSFET driver will be the slave for the Arduino MEGA which will be the master. The SPI input data (SDI) is a 16-bit word with a 5-bit command and address and 11 bits of data and the SPI output data (SDO) consists of 11-bit register data. There's conditions that needs to be followed in order for the SPI communicate efficiently between the Arduino and the MOSFET driver:

- The SCLK pin should be low while the nSCS pin transitions from high-to-low and vise versa.
- The nSCS pin has to be pulled high for at least 400 ns between words.
- Data is captured on the falling edge of the SCLK pin and data is propagated on the rising edge of the SCLK pin.

- The most significant bit (MSB) is shifted in and out first.
- A full 16 SCLK cycles must occur for transaction to be valid.

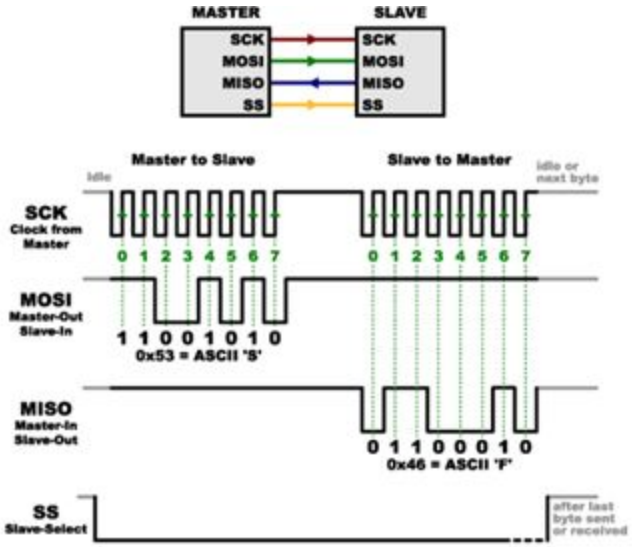

Figure 5.2: Block Diagram of the serial communication lines

With the wide capabilities of the arduino, configuring the settings for serial communication was made very easy. The arduino had several different modes to choose from based on how the driver sends and receives data.

```
84 ////////////////////////SPI interface//////////////////////////////
85
86   SPI.setClockDivider(SPI_CLOCK_DIV128);
87   SPI.setBitOrder(MSBFIRST);
88   SPI.setDataMode(SPI_MODE1);
89   digitalWrite(nSCS, LOW);
90   //Comunicate data
91   digitalWrite(nSCS, HIGH);
92   delay(100);
```

Figure 5.3: SPI setup sample code

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)	Output Edge	Data Capture
SPI_MODE0	0	0	Falling	Rising
SPI_MODE1	0	1	Rising	Falling
SPI_MODE2	1	0	Rising	Falling
SPI_MODE3	1	1	Falling	Rising

Table 5.1: Arduino SPI modes

This code slows the clock signal down for an acceptable speed for communication. It also tells the arduino that code will be transmitted Most Significant Bit first. Lastly one of the Arduino's SPI modes (Mode 1) configures the Arduino to send data on the falling edge of the clock and capture data on the rising edge of the clock.

5.1.2 Configuring the Driver

The next step of setup was configuring the Driver's addresses. The DRV8323RS has seven registers; first two fault status registers and the rest are control registers. The control registers consists of the Input Driver Control Fields, Gate Drive HS Field, CSA Control, OCP Control, and Gate Drive LS.

Table 8. SDI Input Data Word Format

R/W	ADDRESS				DATA										
B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
W0	A3	A2	A1	A0	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Table 9. SDO Output Data Word Format

DON'T CARE BITS					DATA										
B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
X	X	X	X	X	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

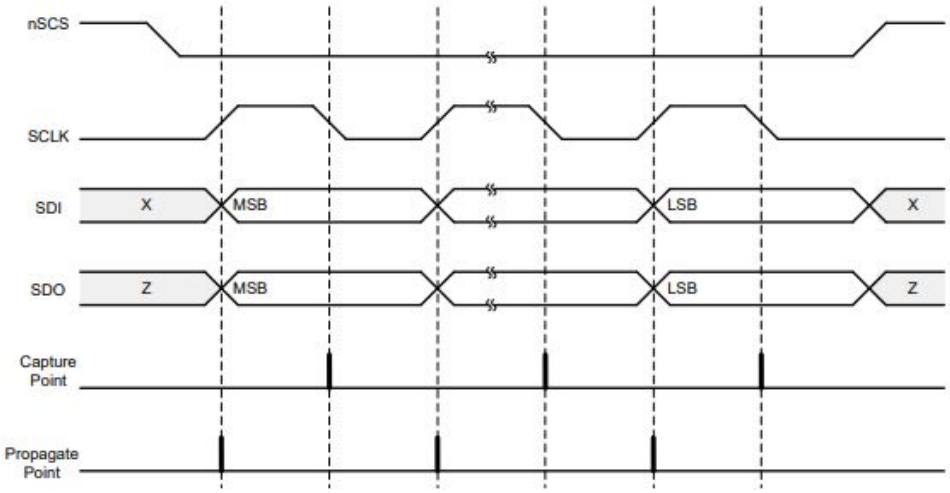

Figure 5.4: SDI and SDO Data Word Format

Name	10	9	8	7	6	5	4	3	2	1	0	Type	Address
DRV8320S and DRV8320RS													
Fault Status 1	FAULT	VDS_OCP	GDF	UVLO	OTSD	VDS_HA	VDS_LA	VDS_HB	VDS_LB	VDS_HC	VDS_LC	R	0h
VGS Status 2	SA_OC	SB_OC	SC_OC	OTW	CPUV	VGS_HA	VGS_LA	VGS_HB	VGS_LB	VGS_HC	VGS_LC	R	1h
Driver Control	Reserved	DIS_CPUV	DIS_GDF	OTW_REP	PWM_MODE		1PWM_COM	1PWM_DIR	COAST	BRAKE	CLR_FLT	RW	2h
Gate Drive HS	LOCK			IDRIVEP_HS				IDRIVEN_HS				RW	3h
Gate Drive LS	CBC	TDRIVE		IDRIVEP_LS				IDRIVEN_LS				RW	4h
OCP Control	TRETRY	DEAD_TIME		OCP_MODE		OCP_DEG		VDS_LVL				RW	5h
Reserved	Reserved											RW	6h
Reserved	Reserved											RW	7h
DRV8323S and DRV8323RS													
Fault Status 1	FAULT	VDS_OCP	GDF	UVLO	OTSD	VDS_HA	VDS_LA	VDS_HB	VDS_LB	VDS_HC	VDS_LC	R	0h
VGS Status 2	SA_OC	SB_OC	SC_OC	OTW	CPUV	VGS_HA	VGS_LA	VGS_HB	VGS_LB	VGS_HC	VGS_LC	R	1h
Driver Control	Reserved	DIS_CPUV	DIS_GDF	OTW_REP	PWM_MODE		1PWM_COM	1PWM_DIR	COAST	BRAKE	CLR_FLT	RW	2h
Gate Drive HS	LOCK			IDRIVEP_HS				IDRIVEN_HS				RW	3h
Gate Drive LS	CBC	TDRIVE		IDRIVEP_LS				IDRIVEN_LS				RW	4h
OCP Control	TRETRY	DEAD_TIME		OCP_MODE		OCP_DEG		VDS_LVL				RW	5h
CSA Control	CSA_FET	VREF_DIV	LS_REF	CSA_GAIN		DIS_SEN	CSA_CAL_A	CSA_CAL_B	CSA_CAL_C	SEN_LVL		RW	6h
Reserved	Reserved											RW	7h

Table 5.2: DRV8323RS Register Map

The Driver Control register contains the main driver functionality modes such as the type of PWM mode the gates will be receiving, coasting the motor, and braking the motor. Synchronous 1x PWM mode is what these prototypes were built around. And for testing purposes, braking and coasting the motor were ignored for future iterations.

Bit	Field	Type	Default	Description
10	Reserved	R/W	0b	Reserved
9	DIS_CPUV	R/W	0b	**0b = Charge pump UVLO fault is enabled** 1b = Charge pump UVLO fault is disabled
8	DIS_GDF	R/W	0b	**0b = Gate drive fault is enabled** 1b = Gate drive fault is disabled
7	OTW_REP	R/W	0b	**0b = OTW is not reported on nFAULT or the FAULT bit** 1b = OTW is reported on nFAULT and the FAULT bit
6-5	PWM_MODE	R/W	00b	**00b = 6x PWM Mode** 01b = 3x PWM mode 10b = 1x PWM mode 11b = Independent PWM mode
4	1PWM_COM	R/W	0b	**0b = 1x PWM mode uses synchronous rectification** 1b = 1x PWM mode uses asynchronous rectification (diode freewheeling)
3	1PWM_DIR	R/W	0b	In 1x PWM mode this bit is ORed with the INHC (DIR) input
2	COAST	R/W	0b	Write a 1 to this bit to put all MOSFETs in the HI-Z state
1	BRAKE	R/W	0b	Write a 1 to this bit to turn on all three low-side MOSFETs in 1x PWM mode. This bit is ORed with the INLC (BRAKE) input.
0	CLR_FLT	R/W	0b	Write a 1 to this bit to clear latched fault bits. This bit automatically resets after being written.

Table 5.3: Drive Control Register explained

The Gate High and Low side addresses were very similar. They control the gate drive currents to the 3 phase inverter. Using the NMOS power MOSFEt datasheets and the following equations from the DRV8323RS datasheet, the appropriate gate currents can be calculated.

$$I_{DRIVEP} > \frac{Q_{gd}}{t_r}$$

$$I_{DRIVEN} > \frac{Q_{gd}}{t_f}$$

The OCP (Overcurrent Protection) register is very important to this project specifically as High current management is a major concern. This register is where the over current protection is configured. By telling the Driver where its measuring for overcurrent and how much is too much current, the chance for blowing parts due to too much current is greatly minimized. This is based on measuring the voltage across the 3 phase inverter MOSFETs based on the internal resistance of these MOSFETs, a calculation for the max current that the 3 phase inverter should see can be found.

Bit	Field	Type	Default	Description
10	TRETRY	R/W	0b	**0b = VDS_OCP and SEN_OCP retry time is 4 ms** 1b = VDS_OCP and SEN_OCP retry time is 50 µs
9-8	DEAD_TIME	R/W	01b	00b = 50-ns dead time **01b = 100-ns dead time** 10b = 200-ns dead time 11b = 400-ns dead time
7-6	OCP_MODE	R/W	01b	00b = Overcurrent causes a latched fault **01b = Overcurrent causes an automatic retrying fault** 10b = Overcurrent is report only but no action is taken 11b = Overcurrent is not reported and no action is taken
5-4	OCP_DEG	R/W	01b	00b = Overcurrent deglitch time of 2 µs **01b = Overcurrent deglitch time of 4 µs** 10b = Overcurrent deglitch time of 6 µs 11b = Overcurrent deglitch time of 8 µs
3-0	VDS_LVL	R/W	1001b	0000b = 0.06 V 0001b = 0.13 V 0010b = 0.2 V 0011b = 0.26 V 0100b = 0.31 V 0101b = 0.45 V 0110b = 0.53 V 0111b = 0.6 V 1000b = 0.68 V **1001b = 0.75 V** 1010b = 0.94 V 1011b = 1.13 V 1100b = 1.3 V 1101b = 1.5 V 1110b = 1.7 V 1111b = 1.88 V

Table 5.3: Overcurrent Protection Register explained

The last control register is the CSA (Current Sense Amplifier) register. This supports the use of sense resistors on each phase of the motor. Based on the range of currents that this motor requires and the resolution of data that can be transmitted on a 5V analog signal, current sense resistor size can be calculated, and the amount of gain the driver needs.

Once the correct driver configurations are calculated and converted into hex numbers, the driver can be programmed over SPI.

```
29
30  word reg1_write = 0x1040; //Write to the Driver Control Register
31  word reg2_write = 0x1B43; //Write to the Gate Drive HS Register
32  word reg3_write = 0x2743; //Write to the Gate Drive LS Register
33  word reg4_write = 0x2952; //Write to the OCP Control Register
34  word reg5_write = 0x32C0; //Write to the CSA Control Register
35
36  word fault1_read = 0x8000; //Read from the Fault Status Register 1
37  word fault2_read = 0x8800; //Read from the Fault Status Register 2
38
39  word reg1_read = 0x9040; //Read from the Driver Control Register
40  word reg2_read = 0x9B43; //Read from the Gate Drive HS Register
41  word reg3_read = 0xA743; //Read from the Gate Drive LS Register
42  word reg4_read = 0xA952; //Read from the OCP Control Register
43  word reg5_read = 0xC2C0; //Read from the CSA Control Register
44
```

```
86
87   //Configure Driver Control Register
88   SPI.setClockDivider(SPI_CLOCK_DIV128);
89   SPI.setBitOrder(MSBFIRST);
90   SPI.setDataMode(SPI_MODE1);
91   digitalWrite(nSCS, LOW);
92   delay(100);
93   SPI.begin();
94   delay(100);
95   SPI.transfer16(reg1_write);
96   delay(100);
97   SPI.end(); //Ends the SPI process
98   delay (100); //delay 1mS
99   digitalWrite(nSCS, HIGH);
100  delay(100);
101
102  //Configure Gate Drive HS Register
103  digitalWrite(nSCS, LOW);
104  delay(100);
105  SPI.begin();
106  delay(100);
107  SPI.transfer16(reg2_write);
108  delay(100);
```

Figure 5.5: Programming the drivers addresses

5.1.3 Configuring the A/D pins

First, the different arduino pins were defined as the names of the ports or hardware that they were connecting to. This made understanding the logic of the code easier. Due to the various sensor and driver outputs and inputs, the arduino pins needed to be declared as inputs. This allowed for proper sending and receiving of important data.

```
6  //Analog pins
7  #define SOC A2 //phase A current sense
8  #define SOB A3 //phase B current sense
9  #define SOA A4 //phase C current sense
10 #define Light A7 //photo cell sensor
11 #define Audio A8 //Audio sensor
12
13 //Digital pins
14 #define PWM 4 //PWM is sent into the Driver and applied on individual Power MOSFET gates
15 #define Temp 6 //Humidity/Temperature sensor
16 #define nFAULT 19 //driver faults are read off this pin
17 #define CAL 28 //Sense resistor amplification
18 #define nSHDN 30 //Shutdwon pin for the buck regulator
19 #define nBRAKE 36 //Brakes the motor
20 #define DIR 38 //controls direction of motor
21 #define STATE2 40 //communicates the location of the motor
22 #define STATE1 42 //communicates the location of the motor
23 #define STATE0 44 //communicates the location of the motor
24 #define ENABLE 48 //Driver operating mode select
25 #define SDO 50 //MISO (Master In Slave Out) serial communication
26 #define SDI 51 //MOSI (Master Out Slave In) serial communication
27 #define SCLK 52 //Serial communication clock
28 #define nSCS 53 //SS (Slave Select) serial communication
29
```

Figure 5.6: Defining the different Arduino pins

```
51  //Declare input pins
52  pinMode(SOA, INPUT);
53  pinMode(SOB, INPUT);
54  pinMode(SOC, INPUT);
55  pinMode(Light, INPUT);
56  pinMode(Audio, INPUT);
57  pinMode(Temp, INPUT);
58  pinMode(nFAULT, INPUT);
59  pinMode(SDO, INPUT);
60
61  //Declare output pins
62  pinMode(PWM, OUTPUT);
63  pinMode(CAL, OUTPUT);
64  pinMode(nSHDN, OUTPUT);
65  pinMode(nBRAKE, OUTPUT);
66  pinMode(DIR, OUTPUT);
67  pinMode(STATE2, OUTPUT);
68  pinMode(STATE1, OUTPUT);
69  pinMode(STATE0, OUTPUT);
70  pinMode(ENABLE, OUTPUT);
71  pinMode(SDI, OUTPUT);
72  pinMode(SCLK, OUTPUT);
73  pinMode(nSCS, OUTPUT);
74
```

Figure 5.7: Configuring the different arduino inputs and outputs

5.2 Sensor Readings

5.2.1 DHT11 Temperature and Humidity Sensor

The DHT11 was polled for the temperature and humidity every 2 seconds to ensure that a new reading had been taken. Taking a reading at shorter times was found to give a fault response from the DHT11. The code was written for the output temperature to be displayed in either celsius or, after a conversion, fahrenheit. The relative humidity is calculated internally by the device and output to the Arduino. The communication with the sensor was handled by the DHT11 library available from Arduino.

```
19   int chk = DHT.read11(DHT11_PIN);
20
21   Serial.print("Temperature = ");
22   tempC = DHT.temperature;
23   tempF = tempConversion(tempC);
24   Serial.print(tempC);
25   Serial.print(" C, ");
26   Serial.print(tempF);
27   Serial.println(" F");
28
29   Serial.print("Humidity = ");
30   Serial.println(DHT.humidity);
31   Serial.println();
32   Serial.print("Light Level: ");
33   Serial.println(analogRead(PhotoPin));
34   Serial.println();
35   delay(2000);
36 }
```

Figure 5.8: Temperature and Humidity test code

```
38 float tempConversion (float degC)
39 {
40   float degF = 0;
41
42   degF = float(degC * 9 / 5);
43   degF = degF + 32;
44   return degF;
45 }
```

Figure 5.9: Celcius to Farenheit conversion function

5.2.2 Photoresistor

The photoresistor detects light levels due to changes in resistance, by creating a voltage divider with another resistor in series an analog pin of the arduino can be attached between the two resistors and the voltage level read. A bright area (high lux reading) will output a value of close to 1023 (as the Arduino has 10-bit ADC) and a dim or completely dark environment will read near 0 indicating a very low lux reading.

```
32   Serial.print("Light Level: ");
33   Serial.println(analogRead(PhotoPin));
```

Figure 5.10: Test code for photoresistor

5.2.3 MAX4466 Audio Sensor

The audio sensor is powered by the 3.3V pin on the arduino and the output is a voltage signal picked up by the integrated microphone and amplified in the sensor before being output to an analog pin on the Arduino. The code for reading the audio sensor is simply reading the input analog signal and getting a digital value (0-1023).

```
1  const int audioPin = A8;
2
3  void setup()
4  {
5    Serial.begin(9600);
6    pinMode(audioPin, INPUT);
7  }
8
9  void loop()
10 {
11   Serial.println(analogRead(audioPin));
12   delay(10);
13 }
```

Figure 5.11: Audio sensor test code

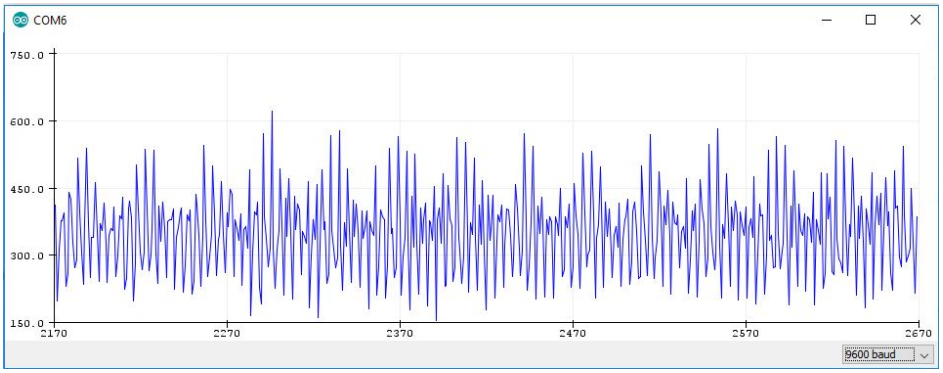

Figure 5.12: 20Hz sample test

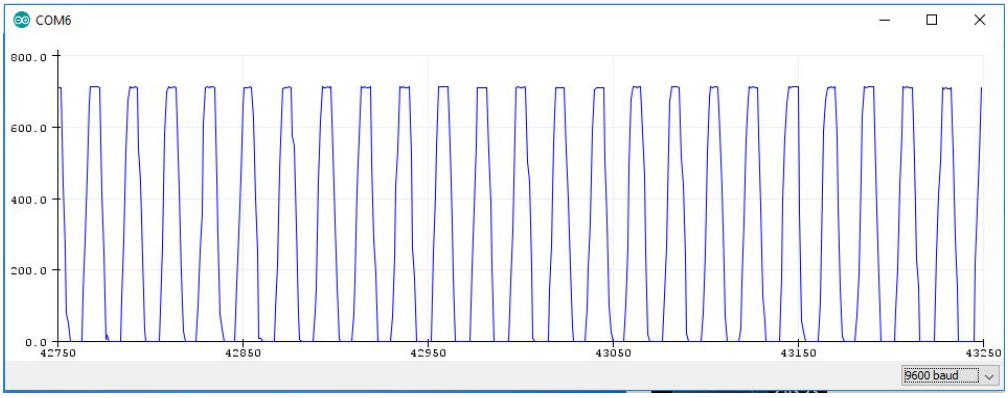

Figure 5.13: 200Hz sample test

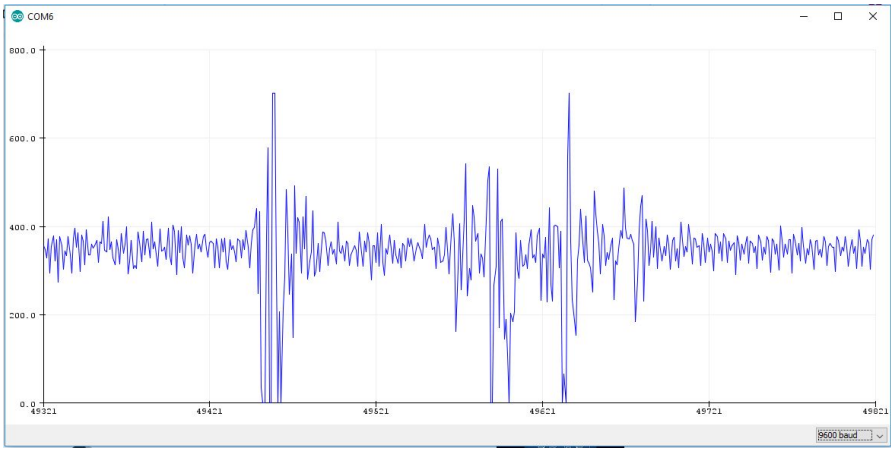

Figure 5.14: Sample speech test (Winsotn Churchill: "This was their finest hour")

5.3 User Input

The User has two main controls, Speed and power of the motor. The following sections describe how each control works.

5.3.1 Power (PWM)

The power of the motor is controlled by a PWM signal Fed to the Driver from the Arduino. The arduino's PWM duty cycle is on a scale of 0-255 (0 off, 255 full duty cycle). As the duty cycle is increased, more power is supplied to the motor.

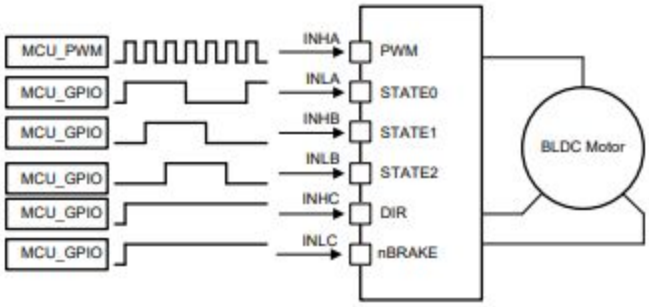

Figure 5.15: 1x PWM - Simple Controller

The arduino makes it very easy to supply a PWM signal as it has specific PWM dedicated pins such as digital pin 4). When activating these pins, a duty cycle value is fed in and the arduino then supplies the PWM signal.

```
239  pinMode(PWM, OUTPUT); //sets PWM as an output
240  analogWrite(PWM,4); //sets PWM duty cycle
```

Figure 5.16: Configuring PWM signal code

5.3.2 Speed (delay/ state simulation)

The second control the user has is the speed or RPM of the motor. The speed of the motor is controlled by the delay of the state simulation loop.

The state simulation loop is how the motor position is simulated in code and then fed to the driver. The driver needs to know the position of the motor as it has to fire up individual MOSFETs in sequence to get a complete revolution out of the motor. A 3 phase BLDC motor has 6 states/ positions. These positions are 60° apart to create a full 360°. Typically these positions are known from hall effect sensors on the motor sending back data on its position however, the 3 phase BLDC motor being used in this project does not have hall effect sensors.

```
242   digitalWrite(STATE0, LOW);
243   digitalWrite(STATE1, LOW);
244   digitalWrite(STATE2, LOW);
245 }
246
247 void loop() {
248   int i = 12;
249   digitalWrite(STATE0, HIGH); //sets state 2 on first loop and state 1 after first loop
250    delay(i);
251   digitalWrite(STATE1, LOW); //sets state 2 after first loop
252    delay(i);
253   digitalWrite(STATE2, HIGH); //sets state 3
254    delay(i);
255   digitalWrite(STATE0, LOW); //sets state 4
256    delay(i);
257   digitalWrite(STATE1, HIGH); // sets state 5
258    delay(i);
259   digitalWrite(STATE2, LOW); //sets state 6
260    delay(i);
```

Figure 5.17: State simulation code

By increasing or decreasing the delay of the simulation, the motor either spins faster or slower.

5.4 Output to motor controller

The software is setup to output certain data to the user while running such as; a fault if it occurs, the current draw, and the power draw from the motor. The following section describes in detail how each one of these outputs work.

5.4.1 Fault Codes

The driver has 21 different faults that it is constantly monitoring for. When a fault occurs, a fault flag is activated and then the fault registers are read to the user. The fault codes are read in hex but can be easily interpreted by using the following tables.

Bit	Field	Type	Default	Description
10	FAULT	R	0b	Logic OR of FAULT status registers. Mirrors nFAULT pin.
9	VDS_OCP	R	0b	Indicates VDS monitor overcurrent fault condition
8	GDF	R	0b	Indicates gate drive fault condition
7	UVLO	R	0b	Indicates undervoltage lockout fault condition
6	OTSD	R	0b	Indicates overtemperature shutdown
5	VDS_HA	R	0b	Indicates VDS overcurrent fault on the A high-side MOSFET
4	VDS_LA	R	0b	Indicates VDS overcurrent fault on the A low-side MOSFET
3	VDS_HB	R	0b	Indicates VDS overcurrent fault on the B high-side MOSFET
2	VDS_LB	R	0b	Indicates VDS overcurrent fault on the B low-side MOSFET
1	VDS_HC	R	0b	Indicates VDS overcurrent fault on the C high-side MOSFET
0	VDS_LC	R	0b	Indicates VDS overcurrent fault on the C low-side MOSFET

Bit	Field	Type	Default	Description
10	SA_OC	R	0b	Indicates overcurrent on phase A sense amplifier (DRV8323xS)
9	SB_OC	R	0b	Indicates overcurrent on phase B sense amplifier (DRV8323xS)
8	SC_OC	R	0b	Indicates overcurrent on phase C sense amplifier (DRV8323xS)
7	OTW	R	0b	Indicates overtemperature warning
6	CPUV	R	0b	Indicates charge pump undervoltage fault condition
5	VGS_HA	R	0b	Indicates gate drive fault on the A high-side MOSFET
4	VGS_LA	R	0b	Indicates gate drive fault on the A low-side MOSFET
3	VGS_HB	R	0b	Indicates gate drive fault on the B high-side MOSFET
2	VGS_LB	R	0b	Indicates gate drive fault on the B low-side MOSFET
1	VGS_HC	R	0b	Indicates gate drive fault on the C high-side MOSFET
0	VGS_LC	R	0b	Indicates gate drive fault on the C low-side MOSFET

Table 5.4: DRV8323RS fault registers map

These built in fault codes are very helpful in the scope of this project. One of the biggest concerns about this project initially was thermal and current management. This Driver monitors for overcurrent at 9 different points on the shield and watches for over thermal conditions too.

In order for the fault register to be read, the correct SPI sequences have to be called because the driver has to communicate this data through serial communication to the arduino. Once the fault registers are read to the user, it is up to the user to interpret them and ultimately how to handle each fault. For future iterations, it would be very easy to translate the data before being read to the user for exactly what faults are being triggered. This would make the project a lot more user friendly.

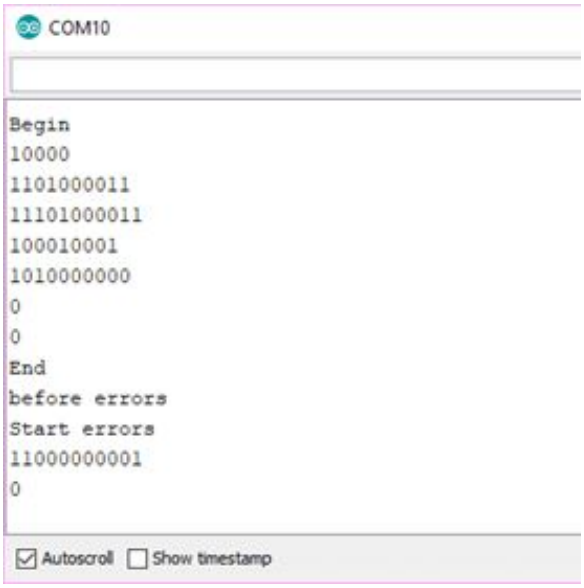

Figure 5.18: Fault register output example

5.4.2 Current and Power Draw

The last output to the computer is the current and power draw from the motor. This is being monitored by a single current sense resistor on each phase of the motor. Programming this system was made very easy by the Driver as it has a built in gain for sense resistors. It also provided the equation for how to interpret the analog signal into a current.

How it works; first the voltage is measured across a known resistor to ground. Then this voltage is given a gain by the diver and is translated to an analog signal that has 1024 different voltages between 0 and 5V used for communicating different information. Once the analog signal is received, using some basic calculations and the gain of the driver, we can calculate what current created that signal. Once the current is known, the power is then calculated from that current. This shield runs at a constant 22v and P = V * I.

Unfortunately, the range of currents that the sense resistors are measuring is 0-50A. This means that for currents less than about 10A, the measurements are pretty inaccurate due to the amount of resolution in the signal and the size of the sense resistors.

6.TESTING

6.1 Driver Prototype Testing

To reduce the potential problems that could come from designing and building one high power version of the PCB it was decided that a low power version would be designed to test interfacing of the arduino and the driver.

The low power version of the PCB was laid out in a way to facilitate easy testing. The inputs and outputs of all three of the motor phases and the gates that controlled them were connected to pins that could then be used as connection points for a test circuit. This way there was access to the intermediate nodes between components and readings could be taken to troubleshoot. The mosfets for the three phases were connected on a breadboard and a low power motor was connected to that.

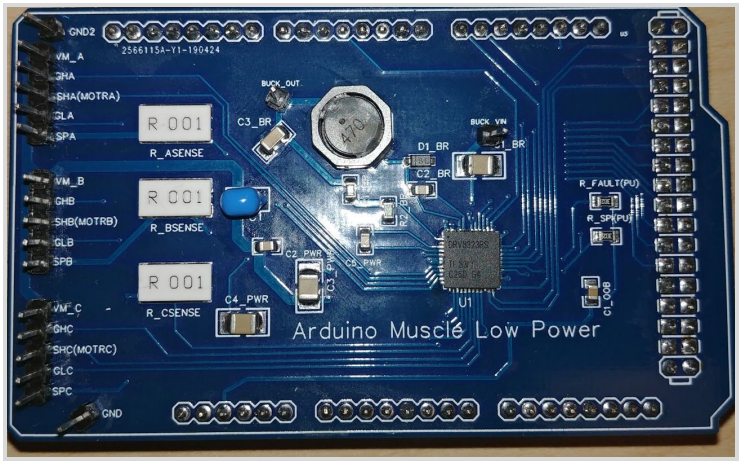

Figure 6.1 Driver prototype board with header pins for each phase

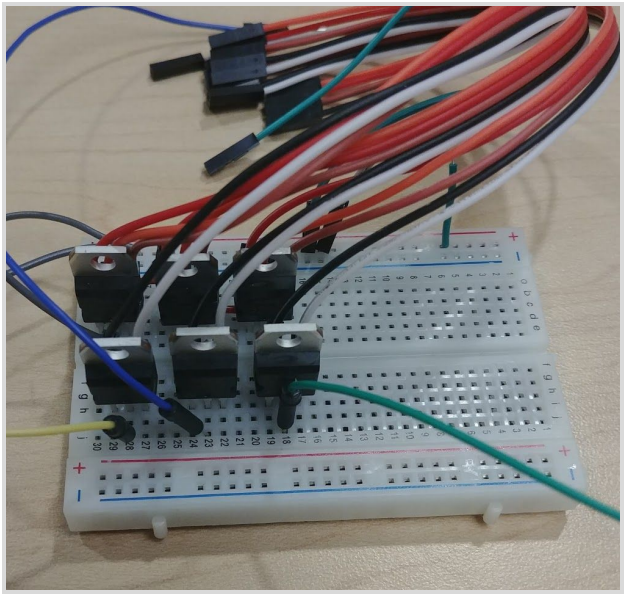

Figure 6.2: Driver prototype 3 phase inverter test circuit

Figure 6.3: Three phase test motor

Table 6.1.1 - Driver Prototype Test plan

Test	Expected	Observed	Action
Vin to Ground resistance	∞	Very high (MΩ range)	
Confirm Driver pins and connections	Complete and uniform connections	All pin connections were uniform	
Confirm different ground plane continuity	Resistance between arduino and shield grounds are low or 0	Resistance was found to be very high	Short the two ground planes with solder and wire
Correct Driver to Arduino A/D pin connections	Driver to arduino A/D pin connections agree with datasheet	State 0, 1, 3 and SPI lines do not connect to the correct pins	Cut header pins and solder appropriate pins together to complete the traces.
SPI signals	Proper signaling viewed on O-scope	SCLK signal was not timed right	Review arduino serial communication configuring
PWM signal (test at different duty cycles)	Signal from arduino ranges 0-5v with correct duty cycle	Observed on the O-scope the correct signal	
Gate drive functionality	The correct Gates are high when correlating states	The gate activation followed that of	

	are simulated (see DRV8323RS datasheet)	the table in the data sheet	
Motor position vs State simulation	The motor adjusts position based on the state simulated	The motor moved for each state would toggle between 3 positions	Increase the power on the bench supply to increase power to the motor
Motor drive functionality	The motor spins while simulating all 6 states in a loop	We were able to spin the motor and adjust speed by adjusting the PWM and state simulation delay	
Limit current with PWM	As the PWM decreases, the current draw off the bench power supply should decrease	Observed the correct PWM to current draw relationship	
Current sensors	Read back from the driver the current draw from the sensors	Current sensors were found to be too small to accurately measure current	Incorporate appropriate current sensors into the next board

6.2 Final Prototype Testing

The final prototype testing was carried out similar to the driver prototype testing. First a visual inspection of the solder joints was carried out to spot any possible issues. Then the board was tested for shorts and continuity without powering it up by use of a multimeter. After no shorts or misconnections was found, the board was tested by attaching the controlled bench power supply and limiting the current (100mA) to ensure the driver could be programmed successfully.

Table 6.2.1 - Final Prototype Test Plan

Test	Expected	Observed	Action
Vin to Ground	∞	Very high (MΩ	

resistance		range)	
Confirm Driver pins and connections	Complete and uniform connections	All pin connections were uniform	
Confirm different ground plane continuity	Resistance between arduino and shield grounds are low or 0	Grounds are connected	
Correct Driver to Arduino A/D pin connections	Driver to arduino A/D pin connections agree with datasheet	All pins are connected correctly	
SPI signals	Proper signaling viewed on O-scope		
PWM signal (test at different duty cycles)	Signal from arduino ranges 0-5v with correct duty cycle	Observed on the O-scope the correct signal	
Gate drive functionality	The correct Gates are high when correlating states are simulated (see DRV8323RS datasheet)	Gates were systematically tested and found to not be behaving as expected. Acting independently	Review driver register programming and set 1x PWM mode instead of 6x independent PWM mode
Motor drive functionality (bench power supply, final prototype board, small motor)	Full rotation with steps and speed based on the PWM and delay settings in the program	Motor behaves as expected.	
Motor drive functionality (bench power supply, final prototype board, big motor)	Full rotation limited holding current, slower rotation due to limited current	Motor rotates, bench power supply compensating for full current draw and higher voltage.	
Motor drive functionality (Lipo batteries , final prototype	Full rotation, full power based on PWM and delay settings	Limited holding strength and constant speed despite PWM and	

board, big motor motor)		delay settings.	
Current sense on each phase	An analog voltage read by the arduino and current calculated based on datasheet formula	Raw readings out of the driver near 5V, very low resolution at low currents	PWM increase was attempted to increase current but current remained unchanged (confirmed by DC current clamp meter)
Board thermals at higher power supply	As PWM is increased component temperature increases	Confirmed by laser thermometer	
Temperature/ Humidity sensor	Correct Temperature confirmed by thermometer	Temperature readings are within datasheet limits	
Light sensor	Correct reading related to Lux meter app	Confirmed with Lux meter app	
Sound sensor	Confirm signal on the sensor based on known sound frequencies	Example frequencies are confirmed with the Arduino Mega serial plotter to	

7. Summary and Evaluation

7.1 Summary of Results

Most of the customer requirements were met. The ones that were not met, were close to being met. The 1kW motor was driven using the final prototype board. However, it was not driven at a high enough current to meet the 1kW goal. Due to this, accurate current and power sense measurements could not be taken as the current and power sense system was setup for measuring much higher currents. As for the size of the final prototype, if multiple boards/ layers were incorporated into this project, the size requirement could easily be met.

Table 7.1.1 - Project Requirements Results

Objectives	Customer Requirements	Actual Results
PCB size	Arduino Mega dimensions: 4 x 2.1 inches	Shield dimensions: Roughly 81% larger
Cost	$100/each (Bulk, 10+ units)	Cost of Parts: just under $62 for one board.
Driving the 1KW motor	Selected motor runs at around 1KW	Drove the motor at 22V and roughly 1A this is about 2.2% full power
Motor Sensing	Current Draw	Were able to read current draw but not accurately due to the low current usage
Motor Sensing	Power Usage	due to inaccurate current readings, power draw could not be measured accurately

All of the various environmental sensors were incorporated into the final prototype and functioned as expected meeting the additional customer requirements.

Table 7.1.2 - Project Sensors Results

Objective	Customer Requirements	Results	Comments
Noise Sense	noise from 20-20KHz	20Hz - ~15kHz, Picked up speech test audio	Has adjustable gain for user application
Light sense	0 to ~4000 Lux	0 to ~4000 Lux	Ambient Room light: center of scale
Temperature Sense	temperature from 0-50°C	Correct temp. +3% error	Consistently overestimates by 3-4%
Humidity Sense	relative humidity between 20-80%	+/-4.6% error	Better than the datasheet value of +/-5% Based off of NOAA weather data

7.2 Future Iterations and Work

The next iteration of this project would ideally be able to drive up to the max current of 50A that was specified for this version. This would enable the use of the sense resistors in this configuration as the gain settings and the values of the sense resistors would allow for a fine reading of the output current on each phase of the motor and tell the user what kind of load that the motor was attempting to move. With the driven current being known, the power in each phase could be calculated and the output force and accurate battery drain could be calculated. Other improvements could be fully utilizing the buck regulator to act as a “battery elimination circuit” and step down the full voltage of the battery to a level that the Arduino could run off of without the need for a USB tether for power. Another potential change could be setting up for a sensorless drive of the chosen motor or using a brushless motor with hall effect sensors for positional sensing.

8.List of Industry Standards

8.1 UART Serial Communication

The Arduino communicates to a PC terminal via a USB connection. It uses a UART communication protocol. To receive information from the microcontroller on a PC via USB, a serial monitor needs to be installed and launched on the PC. The correct COM ports need to be selected, and the baud rate needs to be set to 9600 pulses per second. The PC assigns the COM port.

8.2 USB Serial Communication

The Arduino uses a male mini-B USB to male type-A USB cable to interface the Arduino with a PC. The USB cable transmits a UART signal for this communication.

8.3 IEEE 1679.1-2017

IEEE Guide for the Characterization and Evaluation of Lithium-Based Batteries in Stationary Applications.

8.4 IEEE Std 1185-1994

IEEE Guide for Installation Methods for Generating Station Cables. Corrections issued August 1, 2000.

9. Acknowledgements

The team would like to recognize Joe Justice and WikiSpeed for their contribution to this project and our educational achievements. The team next would like to recognize Dr. Randy Kong for his insight and expertise. Finally, the team would like to recognize Mark Soper for his contribution to assemble components, insight, and knowledge to this project.

The team would like to acknowledge the following professors for their contributions towards the project:

Dr. Hrair Aintablian, Ph.D.
Dr. Arnold Berger, Ph.D.
Dr. Walter Charzenko, Ph.D.
Dr. Kaibao Nie, Ph.D.

10.REFERENCES

Adafruit. “Using a Photocell" July, 2012. Web.

<https://learn.adafruit.com/photocells/using-a-photocell>.

Adafruit. “Using a Temp Sensor" July, 2012. Web.

<https://learn.adafruit.com/tmp36-temperature-sensor/using-a-temp-sensor>.

Arduino. “Serial" Spring, 2014. Web.

<https://www.arduino.cc/en/pmwiki.php?n=Reference/Serial>.

Circuit Basics. "HOW TO SET UP THE DHT11 HUMIDITY SENSOR ON AN ARDUINO" Oct, 2015. Web.

<http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/>.

CordlessDrillZone. "Brushless Vs Brushed Motor: Why You Should Know The Difference." July, 2018. Web.

<https://cordlessdrillzone.com/drill-wars/brushless-vs-brushed-motor/>.

Digikey. "Fundamentals of Current Measurement: Part 2 – Current Sense Amplifiers." Nov, 2018. Web.

<https://www.digikey.com/en/articles/techzone/2018/nov/fundamentals-of-current-measurement-part-2-current-sense-amplifiers>.

Digikey. "How to Power and Control Brushless DC Motors." July, 2011. Web.

<https://www.digikey.com/en/articles/techzone/2016/dec/how-to-power-and-control-brushless-dc-motors>.

NK Technoglogies. "Current Sensing Theory" Fall, 2017. Web.

<https://www.nktechnologies.com/engineering-resources/current-sensing-theory/>.

Sommer, Chris. "3-Phase H-Bridge Design and MOSFET Selection." Nov, 2013. Web.

<http://www.egr.msu.edu/classes/ece480/capstone/fall13/group09/Documents/AppNotes/Chris.doc.>.

Sparkfun. Analog to Digital Conversion" Fall, 2015. Web.

<https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all>.