Beau Rogers

2017-2018 Supermileage Electric Motor Hand-off Document

Table of Contents:

BLDC Principle of Operation

The principle behind the electric motor to obtain maximum efficiency is to use higher operating speeds to give higher efficiencies. This is obtained by looking at two relationships in the electric motor. They are seen in the equation below.

$$e = k\omega \quad \& \quad T = kI$$

K is some constant in the electric motor that cannot be changed. ω is the speed of the motor, T is the torque, e is the voltage across the motor and I is the current. This means that the faster the motor spins, the more voltage (or back EMF) is created across the motor. This can be seen in figure 1 of the simplified circuit of an electric motor. The resistance R and inductance L is the internal resistance and inductance in the motor that cannot be changed

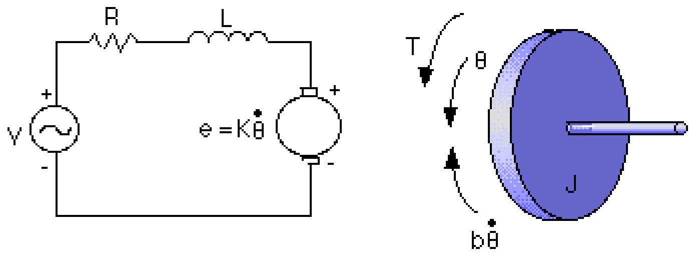


Figure 1: Basic Motor Circuit

As the speed is increased, the voltage across the motor will increase, thus making the voltage difference across the resistor smaller, decreasing the amount of heat loss and increasing the efficiency. The inductor in the circuit can be ignored as we are working with a DC motor and it has negligible effect. This, however, is not a perfect relationship. When the speed increases close to the maximum speed of the motor, very little current is being spent across through the motor. This means that very little torque is being produced and all the torque that you are producing is going to the mechanical losses. This reduces the efficiency very quickly. As a result, you get graphs that look as figure 2. Here, it can be seen how the red efficiency curve can go as high as 85% in this particular motor and then quickly drops off as the speed increases. In the competition, gear ratios and simulations can be used to target the required vehicle speed to match with the highest efficiency point of the motor.

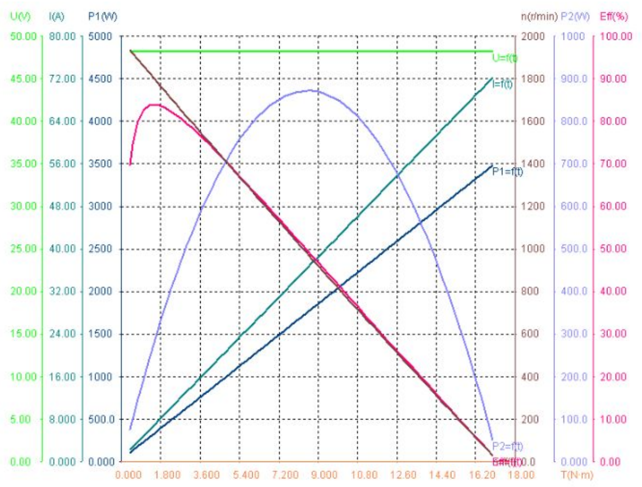


Figure 2: Example of Speed-Torque Curves of BLDC Motor

The type of motor being used in this case is a BrushLess Direct Current Motor (BLDC Motor). [A video explanation of how these work can be found in this youtube video if desired: https://www.youtube.com/watch?v=bCEiOnuODac]. The way this works is to have 3-phases (phases A-C) in Y configuration seen in figure 3 below. Mosfets, which essentially act as switches, are then toggled sequentially to allow current to flow through the desired phases. In the case of figure 3 below, the current is flowing through A-B (energized windings) with C having no current flowing through (de-energized winding). The choice of the which phase to send current through is dependent on the position of the windings in reference to magnets that rotates in response to the current flowing through each phase. The position of these rotors is determined by one of two methods known as 'sense' and 'senseless'. The 'sense' method uses stationary Hall Effect Sensors within the motor that detect the presence or absence of the magnets that can be used to determine which phases should be energized. The 'senseless' method is more complicated. The motion of the magnets passing by the windings can induce voltage, specifically in the non-energized winding. In the case of figure 3 below, this would be winding phase C. This induced voltage would then be registered by the motor controller and depending on which phases are energized would determine the next phase that needs to be energized.

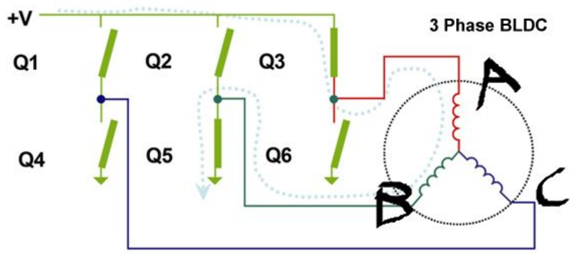
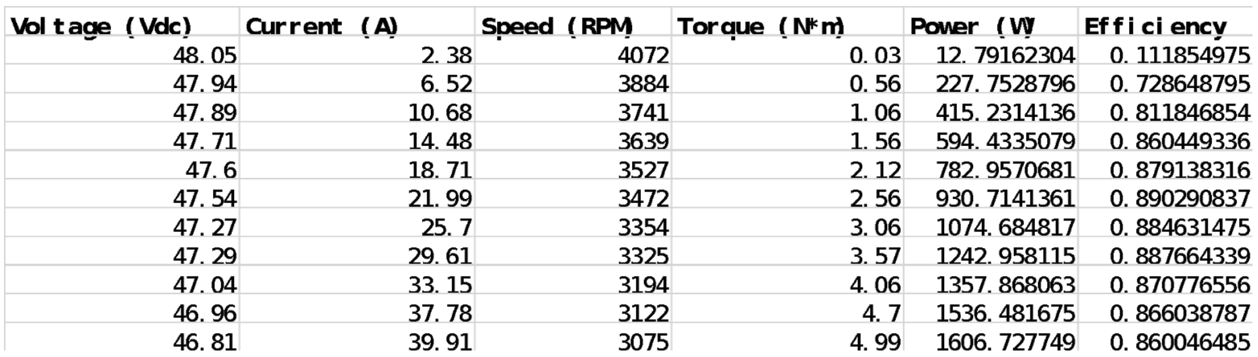


Figure 3:Basic BLDC motor diagram

Motor Choice and Simulations:

As mentioned in the section before, the desire was to have a motor whose expected operating speed would be falling the maximum efficiency of the motor. For deciding this, it was assumed that the vehicle would be wanting to operate as a speed of around 9m/s. The particular motor that was decided upon was a Golden Motor, 1500W BLDC motor, where the link can be found here (click BLDC tab -> ctrl+f 1500):
https://goldenmotor.com/
With this, the specs of the motors operating conditions were taken as seen below:



Voltage (Vdc)	Current (A)	Speed (RPM)	Torque (N*m)	Power (W)	Efficiency
48.05	2.38	4072	0.03	12.79162304	0.111854975
47.94	6.52	3884	0.56	227.7528796	0.728648795
47.89	10.68	3741	1.06	415.2314136	0.811846854
47.71	14.48	3639	1.56	594.4335079	0.860449336
47.6	18.71	3527	2.12	782.9570681	0.879138316
47.54	21.99	3472	2.56	930.7141361	0.890290837
47.27	25.7	3354	3.06	1074.684817	0.884631475
47.29	29.61	3325	3.57	1242.958115	0.887664339
47.04	33.15	3194	4.06	1357.868063	0.870776556
46.96	37.78	3122	4.7	1536.481675	0.866038787
46.81	39.91	3075	4.99	1606.727749	0.860046485

Figure 3: Efficiency Specification of the Golden Motor

When viewing these numbers, we can first see that the maximum efficiency occurs at 3472rpm. Next we want to have an equation that we can plug into a simulation. So using the speed-torque equations mentioned earlier, the voltage vs speed and current vs torque could be graphed and excel could spit out a linear equation as seen in figure 4 below.

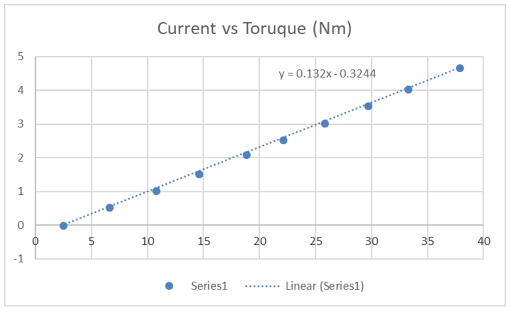


Figure 4: Linearize equation of the Current vs Torque

Once the equations were found, a simulation in Matlab could be produced that would graph out the speed, torque, power and motor velocity over time and compare it across different gear ratios. Another important factor in the electric motors is the limits imposed/suggested by the motor controller for the motor. To prevent the motor from overheating, current limitations are imposed on the motor such that that it has a maximum torque that can be produced. As a result, as the motor speeds up, the torque remains constant until the speed is fast enough such that the torque starts to reduce with speed. This can be seen in the bottom left graph in figure 5 below.

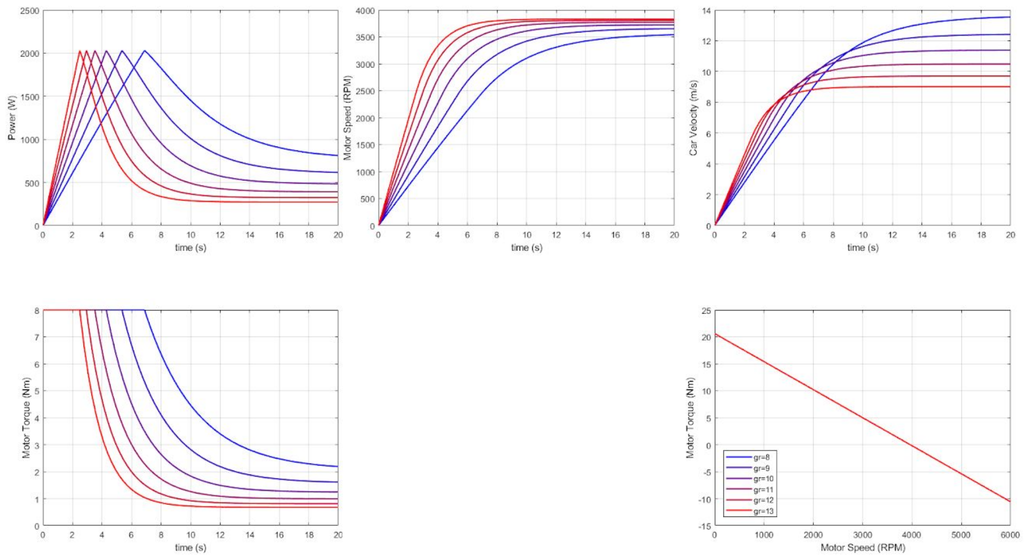


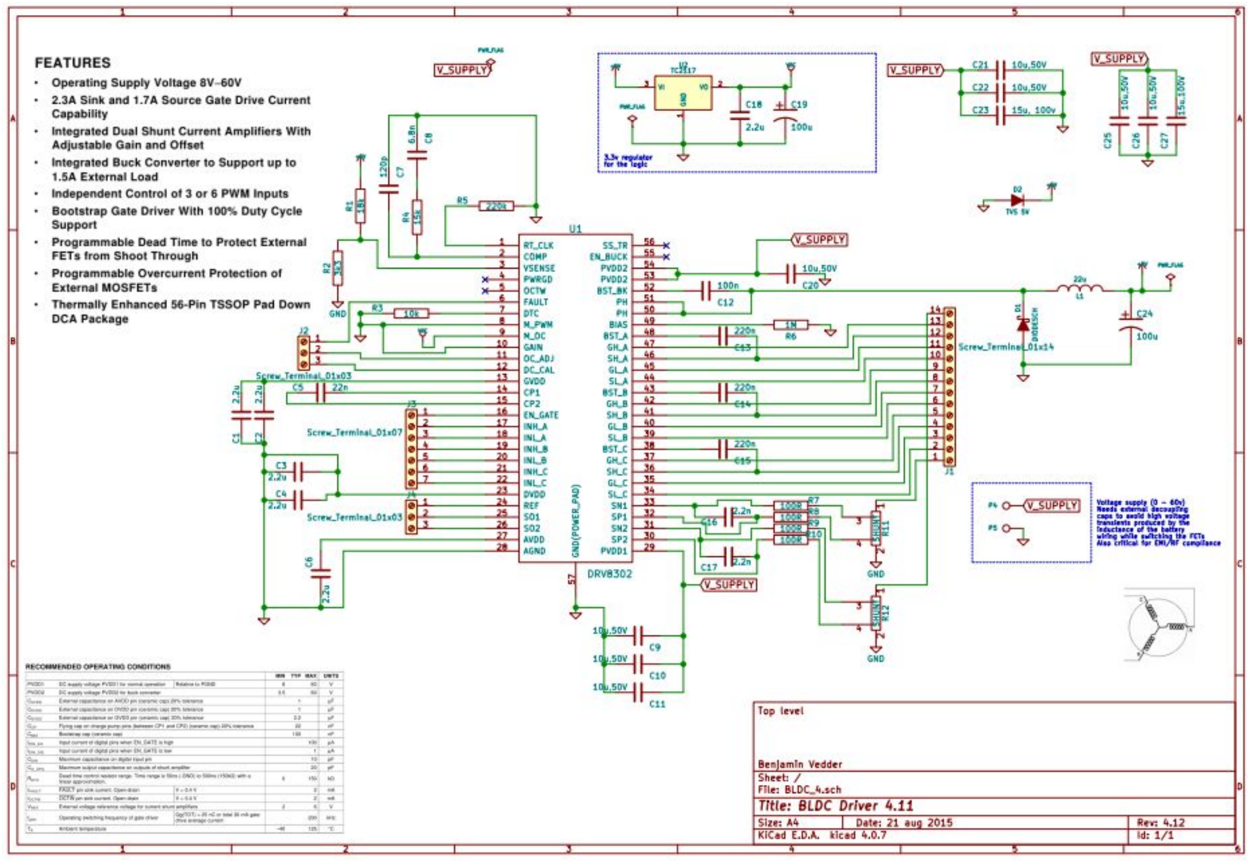
Figure 5: Motor Performance Simulation Results

Looking at the graph, the attention can be drawn to the top right and top middle graphs. Here, we can examine the theoretical acceleration time and maximum speed and compare it to the rpm. So observing the top middle graph, with a gear ratio of 12, we can see that we can get to 3500rpm in about 5.5 seconds. Now if we look at the top right graph, we can see that at around 5.5 seconds we would expect to be driving around 9m/s. So driving at speeds a little lower than this provide can provide the desired speeds with efficiencies in the mid 80% range. Or if more speed is needed, the gear ratio can be reduced to 11 and the same procedure followed.

Motor Controller-Mosfet Driver

The motor controller portion is largely inspired by an open source motor controller http://vedder.se/2015/01/vesc-open-source-esc/. Here, the firmware, schematics, PCB, Bill of Materials and help with the motor controller can be found here. However, with the spirit of the competition being to create our own motor controller, this cannot be copied verbatim. As a result, the schematic of the motor controller is stripped down to meet our needs.

The primary needs is to drive the electric motor, with some form of circuit protection and the possibly to have the option to implement some sort of regenerative braking system. To do this, external sensors on the system were stripped away, the micro-controller on the board was replaced with an Arduino mega and the schematic of the mosfet driver was reduced down to the circuit seen below. Please refer to the original schematic found on the website for better understanding.



FEATURES
Operating Supply Voltage 8V–60V
2.3A Sink and 1.7A Source Gate Drive Current Capability
Integrated Dual Shunt Current Amplifiers With Adjustable Gain and Offset
Integrated Buck Converter to Support up to 1.5A External Load
Independent Control of 3 or 6 PWM Inputs
Bootstrap Gate Driver With 100% Duty Cycle Support
Programmable Dead Time to Protect External FETs from Shoot Through
Programmable Overcurrent Protection of External MOSFETs
Thermally Enhanced 56-Pin TSSOP Pad Down DCA Package
V_SUPPLY
3.3v regulator for the logic
U1
DRV8302
Screw_Terminal_01x03
Screw_Terminal_01x07
Screw_Terminal_01x14
RECOMMENDED OPERATING CONDITIONS
Top level
Benjamin Vedder
File: BLDC_4.sch
Title: BLDC Driver 4.11
Size: A4
Date: 21 aug 2015
KiCad E.D.A. kicad 4.0.7
Rev: 4.12
Id: 1/1

Figure 6: Modified Vedder Schematic of DRV8302

The mosfet driver chip is meant to act as a buffer between the microcontroller and the actual electric motor. The intention of the is to be on a separate circuit board such that the mosfets and motor controller can be plugged into this board externally. The particular chip is a DRV8302 Texas instrument mosfet driver. For ease of implementation, this circuit is build on a breadboard with a surface mount to through hole converter to allow for the specific chip to be put onto a breadboard. Terminal blocks are used to connect the mosfets to the mosfet driver as well as connect microcontroller to the mosfet driver.

Mosfet control pins:

The chip can be powered off a 48V supply and take the 5V data signals coming off the Arduino mega. The chip also has a buck converter to allow for 3.3V devices to be powered off it. However, that is a feature that was not needed and removed from the above schematic (all pin reference for this section will be referring to pins seen in figure 6). When looking at pins 16-22, these are the input signals from the microcontroller to the gate controllers for Mosfet H-bridge. These take a PWM signal at a specific frequency (this will be discussed further in the microcontroller section) with the duty cycle of the waveform controlling the speed at which the motor can run. Pin 16 is the En_gate pin, this has to be enabled high to drive any of the other pins. This can be useful as an easy pin to manipulate in order to shutdown the motor in that case of a fault being found. Pins 17-19 (inclusive) are the 'high side' mosfets while pins 20-22 are the 'low side' mosfets. This resembles the mosfet configuration seen below in figure 7. This configuration allows for the microcontroller to control the direction the current flows through each of the 3-phases as well as controlling the speed.

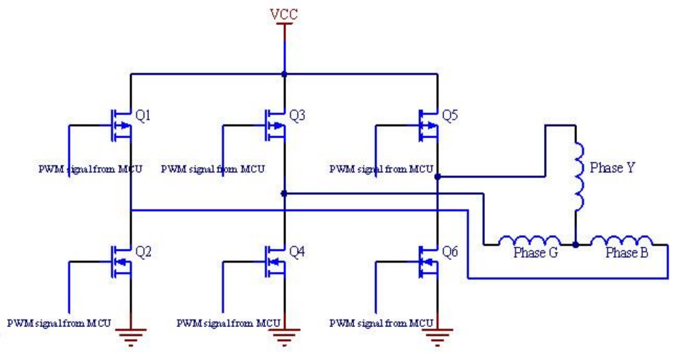


Figure 7: BLDC Mosfet Bridge

Current Sensing pins:

Pins 25 & 26 are information about the current flow through the shunt resistors R11 and R12. The purpose of these is to allow to read the current that is flowing through the motor and have the microcontroller determine if there is too much current running through the motor. This is done with an analog voltage between 0V and the reference voltage input into pin 24. Since we are using an arduino mega, the maximum recommended voltage to input to the microcontroller is 5V. Therefore, pin 24 is meant to be biased with 5V. The expected analog voltages are based off the equation seen below.

$$V_0 = (V_{ref}/2) - G(V_{SN} - V_{SP})$$

Where V_ref is the 5V reference voltage, (V_sn-V_sp) is the voltage across the shunt resistor R11 or R12 and G is the gain of the amplifier. G is determined by pin 10 and is either 10V/V, when G=0V, or 40V/V when G=5V. **Please note** the absolute maximum voltage on this pin is 7V, so do not exceed this voltage on any logic pin of the mosfet driver.

Faults:

One of the benefits of the mosfet driver is its ability to detect faults. These are controlled/detected using pins 6, 11 and 12. **A mistake in the schematic** is seen below on the left side of figure 8. The intersection circled in blue, by capacitors C1 and C2, is not meant to be an intersection. This shorts DVDD (3.3V buck converter output) straight to ground. **This can destroy the chip if hooked up to a power supply that is not current limited!!!!!!!** Instead, the wire should be routed as found in page 5 of the Vedder schematic, seen on the right side of figure 8. In particular, pin 12 (OC_ADJ) is best left to be high (3.3V). This is because the chip will automatically current limit the motor if the current passes a threshold determine by this pin. However, if this is desired in the future, equations can be found in the datasheet for configuring this pin.

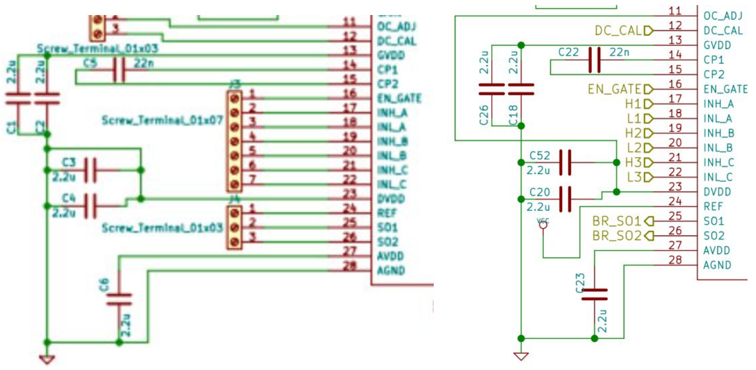


Figure 8: Mistake in schematic

Pin 6, is the output that indicates a fault condition to the microcontroller. This must be biased with a resistor to 5V. This pin will be in tri-state (ie. whatever voltage is at its input, in this case 5 volts) and be sunk to ground in the event of a fault determined by the input voltage to pin 11. The purpose of this is to measure the amount of current flowing through any of the transistors. This is done by measuring the voltage drop between the source and drain of the transistors in the mosfet bridge. The voltage across this exceeds the voltage input to pin 11, pin 6 will drop to ground. The desired voltage is equal to the equation below:

$$V_{DS} = I_{Fault} * R_{DS}$$

Where V_DS is the voltage across the transistor, I_Fault is the maximum expected current and R_DS is the on-resistance of the transistor. This R_DS value can be found in the datasheet of the transistor that is used. R_DS can be around 1mΩ, so if a fault current of 150A is wanted, pin 6 would need to be biased to 0.15V.

-suggested operating and test conditions
-tests results and current problem seen

Motor Controller-Microcontroller

To meet the requirements of interacting with the DRV8302 as well as allowing for speed control of the motor, the microcontroller required to have: (x1) PWM signals, (x3) ADC pins and (x5) pin change interrupts. The PWM signals are required to control the speed of the motor using the duty cycle of a consistent input frequency. The (x3) ADC pins are required to measure the current across the two shunt resistors R11 and R12 while the third is required to measure the desired speed input by the driver (this is done using a potentiometer). The (x5) pin change interrupts are required to detect the (x3) hall effect sensors on the motor while the other two are meant to detect the Fault and Over Current/Over Temperature pins. With these pin requirements and the desire to allow for a greater number of team members to be able to be able to program the microcontroller, the arduino mega was chosen. However, it may be recommended to look into ARM or ATMEL chips to meet the requirements as they may be better. Additionally, those brands are used in industry and look much better on a resume.

Mosfet Control:

The learnings from this section were found by reading the 'Motor Driver Theory' pdf found on the G-drive in the Electric Motor folder of the 2017-2018 supermileage year. The code for this microcontroller can be found in this same section. This include the main file as well as the master header file defining the names of the pins and variables. The mosfet control uses the PWM signal of the microcontroller to output signals onto PH_x_OUT_H and PH_x_OUT_L (where x=A-C). The specific pins that are output depend on the state of the hall effect sensors. This is done by using digital logic where the state of the transistors is determined by the digital logic of the hall effect sensors. A Kanaught map of the transistors can be seen in figure 9 below. Where Sx is Sensor A-C and xH/xL is transistor A-C.

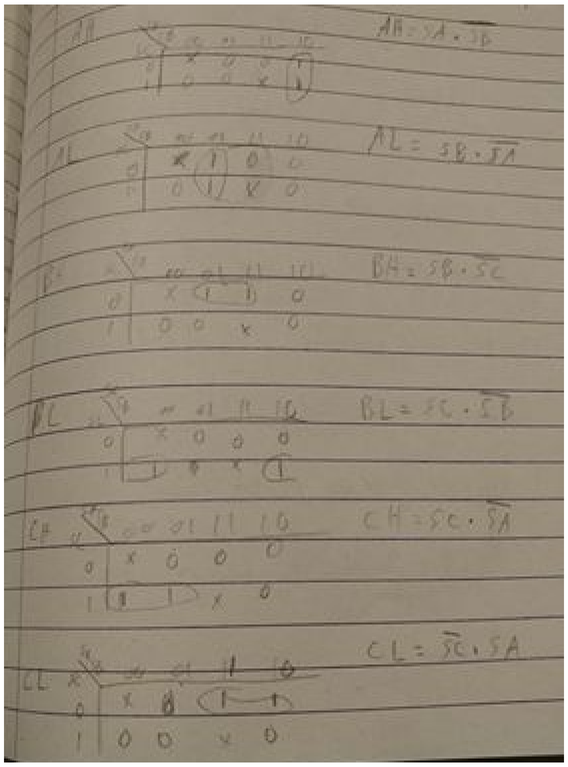

AH = SA · SB
AL = SB · $\overline{SA}$
BH = SB · $\overline{SC}$
BL = SC · $\overline{SB}$
CH = SC · $\overline{SA}$
CL = $\overline{SC}$ · SA

Figure 9: Kanaught Map

Using the digital logic that results from the Kanaught maps, the pins can be written to in a single clock cycle. Additionally, we can look at the state of the PWM signal to determine if the pin should be high or low. The resulting code for each pin looks as:

```
digitalWrite(PH_A_OUT_H, (In_A && !In_B)&& PWM_State);
```

In this case, PH_A_OUT_H will go high if the hall effect sensor A is high, sensor B is low and the PWM state if high. It does not matter if sensor C is high or low. The PWM signal is determined through… a bit of a work around. A PWM signal is output through a pin called "PWM_No_Connect" and the state is read through "PWM_Work_Around" and stored in the boolean variable "PWM_State." To improve this, this should be done using an Output Compare Interrupt routine to create a timer based interrupt that can toggle the pins high and low. However, this interrupt would for some reason not work. The frequency of the PWM signal is determined by the pre-scalars of the microcontroller clock. Through testing, it was found that the output frequency that gives a clean PWM signal is 4kHz. This is done by configuring a prescaler of 8 through TCCR2B = _BV(CS21). It is belived this can be improved if the Output Compare Interrupt is used as opposed to using the analogout() function in the arduino library.

Sensor Reading:

For sensor readings, there are four things that need to be read. The first is the fault pin 5. This has not been implemented in the code yet, but will be a simple pin change interrupt routine. This should prevent all the PWM signals from outputting when the interrupt detects a falling edge (ie. The pin goes from high to low).
The next one is the ADC speed control. This is implemented by using the circuit seen in figure 10. Here, a 10kΩ potentionmeter is used to vary the voltage at the microcontroller ADC input between 0.97V and 0.025V (the potentiometer does not drop to a resistance of 0Ω). The reason these voltages are chosen is because the *digitalread()* function will return a value between 0-1023, for voltages between 0-5V. Voltages of 0.025-0.97V results in the *digitalread()* function to read a raw value between 243 and 6. This is valuable because this raw value can be put directly into the *analogwrite()* function that takes a value between 0-255 to spit out a duty cycle. By doing this, the microcontroller does not need to do mathematical operations that use up clock cycles and can lead to unexpected behaviour/errors when performing high speed operations.

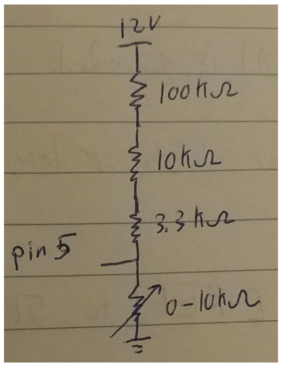


Figure 10: Speed control circuit

The last is the ADC for the current sensing. This will be the voltage output from the Mosfet driver in which the voltage will relate to the current as described earlier in *Motor Controller-Mosfet Driver* section. This has not been implemented in the code. It is recommended to use the same raw value technique as described above. Ie. decide how the microcontroller will respond to certain currents and set the corresponding raw ADC values for which the microcontroller can respond. Multiplication and division cycles can take tens to hundreds of clock cycles and can potentially cause problems.

Test Results/Current State:

The current state of the motor controller is testing phase to confirm the function of the Mostfet driver. This was done by connecting LEDs and resistors to be the 'motor' and have an extra arduino uno drive signals into the arudino mega. These signals are the expected signals that would be output by the motor. By baising the supply voltage with 24V, it was found that the system behaved as expected. However as the voltage was risen up to 36V, a flickering pattern was observed in the LEDs, indicating there would be a flickering pattern that would result when running the actual motor at 48V.

There are still more aspects that need to be written in the code that were mentioned before. This includes:

- Creating routines to respond to faults
- Detecting the current from the Mosfet driver
- creating an output compare routine to replace the current 'work around routine' to make the program more efficient.
- creating and understanding how to program the micocontroller to respond to 'senseless' motor control. Better understanding of this can be found in the 'Motor Controller Theory' pdf mentioned earlier

Once the motor can be run with the motor controller, the next step will be to figure out the required current capacity of the battery to drive two rounds. This will first require to make a dino for the electric motor, then in the supermileage shop, there are energy meters that were purchase late last year. These can be put at the output of the battery when running the motor to find out the amount of energy used in a given amount of time/speed, giving the required amp-hours needed for the battery during competition. Since the battery is an expensive purchase, this will need to be determined through tests.