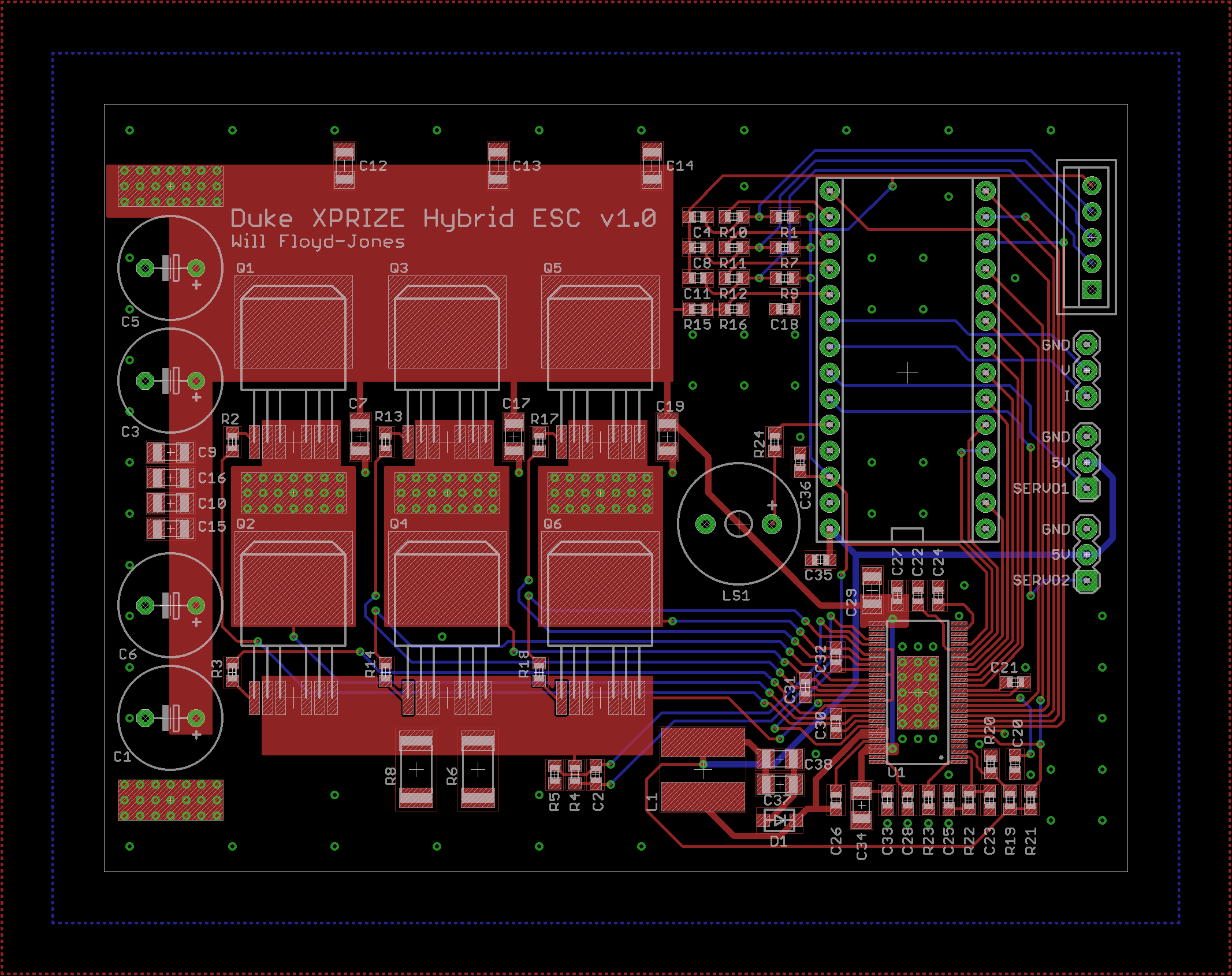
**Hybrid Gas-Electric Electronic Speed Controller**

Will Floyd-Jones



**Overview**

This report documents progress I have made on the electronic speed controller (ESC) for the hybrid gas-electric power system. I have designed a printed circuit board for the ESC which contains a microcontroller, gate driver IC, hall effect sensor inputs, a buzzer, and connectors to interface with other components in the hybrid system. The board will be used to spin a brushless DC motor to start the gas engine and control the servo motors and to regulate engine speed and battery voltage.

Instructors: Dr. Martin Brooke

ECE 496 - Advanced Robotic System Design

*Duke University*

Table of Contents

**Background 2**

**Schematic 5**

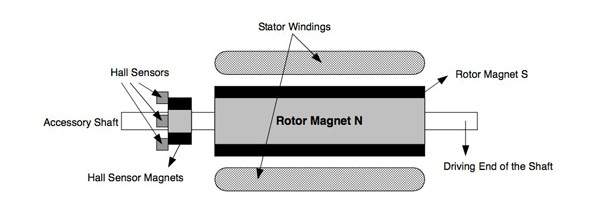
**Layout 8**

[**Code**](https://docs.google.com/document/d/1wcFQonSCTwOSvjcp4_RewytC23nD_s9eHOoJ1f-UhME/edit#heading=h.sfrdv3745265) **10**

**Next Steps 14**

**Background**

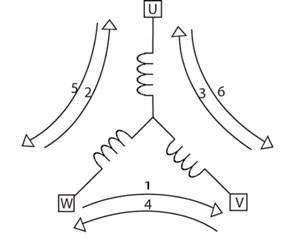
Brushless DC motors are gaining in popularity due to their efficiency advantage over their brushed counterparts. The brushes of a conventional motor are used to mechanically commutate the device- meaning that they apply current to the coil windings to swap the direction of the magnetic field. This makes turning a brushed DC motor very simple: apply DC current and the motor will spin. However, these brushes can wear out over time and efficiency losses occur due to arcing and suboptimal management of the magnetic field orientation.

Brushless motors require the system designer to electronically commutate the device- a microcontroller is used to apply currents to the coil windings based on the position of the rotor relative to the stator. The stator coil generates a rotating magnetic field which drags the rotor through each successive revolution. Since the relative position of the rotor must be known to determine which direction currents should travel through the stator windings, three hall-effect sensors are often embedded in the stator and used to indicate 6 unique rotor positions. As the rotor magnet passes over one of the hall-effect sensors, its output will switch from low to high. The figure below illustrates a brushless DC motor in a inrunner topology- that is, the rotor is on the inside and the stator is on the outside- with hall-effect sensors on the non-driving end.

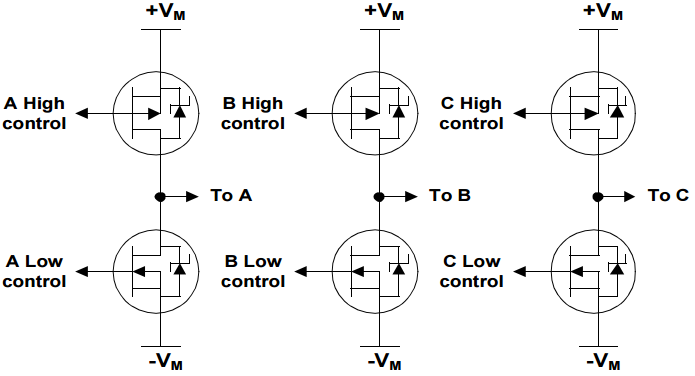
*Figure 1: Inrunner-style brushless DC motor*

There are more sophisticated ways to detect the rotor position- encoders which give far better resolution than the simple hall effect sensor topology, or a sensorless strategy which measures the back-EMF from the rotating motor’s phases to infer the rotor position. These complicate the system design but can be used to enhance reliability and efficiency. However, in the hybrid gas-electric system on the drone, the ESC only exists to start the motor. This means that it does not have to be very efficient since it is not on for very long and only drains the battery which is used only to start the motor. The sensorless strategy relies on back-EMF produced by a rotating motor- meaning that it cannot be used to start rotating the motor. Typically this is overcome by rotating the motor open-loop until it is going fast enough to produce sufficient back-EMF, but the motor torque will be low while it is being spun open-loop because the magnetic field will not be well-oriented. Again, the ESC in the hybrid system is for starting the motor- nothing else- so it would be foolish to employ a sensorless strategy where the motor would struggle to produce enough torque to drive the gas engine.

The stator of a brushless DC motor is comprised of three sets of coil phase windings, which are attached at a common point in the center as shown in Figure 2 below. The motor has three terminals- one for each stator phase- which connect to the ESC’s drive phases. These drive phases, shown in Figure 3 below, apply currents to the stator using three MOSFET pairs- which are used to drive one motor phase high and another low while the third is left floating. It is up to the system designer to ensure that the high side and the low side of a single drive phase are never on at the same time- this would cause a short between the positive and negative battery terminals through the drive phase’s MOSFETs. If hall-effect sensors are used for determining the rotor position, a simple look-up table can be used to select which MOSFETs to turn on based on the hall-effect sensor states.



*Figure 2: Y-Configuration of stator coils*



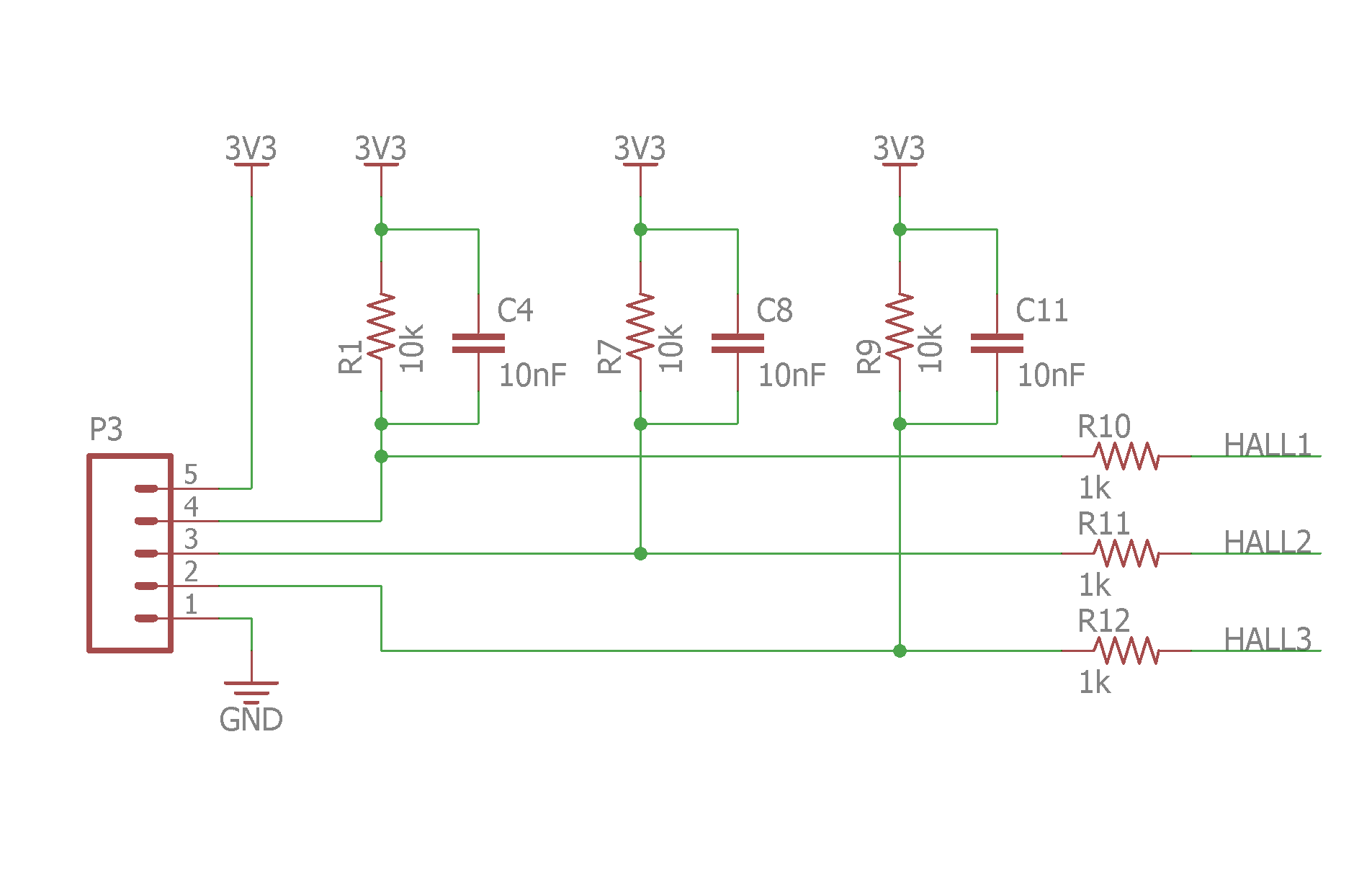
*Figure 3: 3- Half H-bridge MOSFET drive phase schematic*

# 

**Schematic**

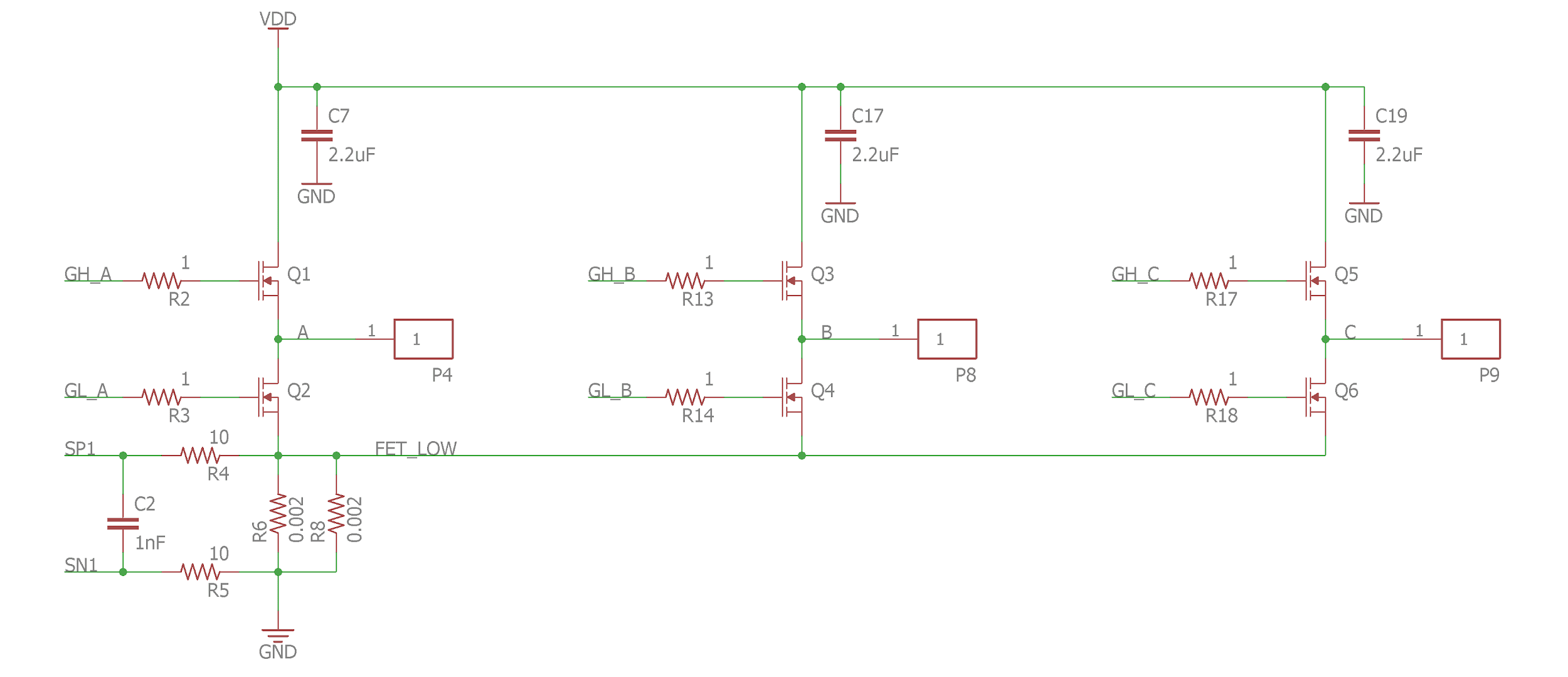
The schematic capture and PCB layout for this project were done in EAGLE. I selected EAGLE over more capable tools such as Altium Designer because this PCB is not too complex and EAGLE is still the standard among hobbyists and should be the easiest to use if this design is re-used or modified in the future. The schematic contains sections for decoupling capacitors, the hall-effect sensor inputs, connectors, the gate driver IC, the MOSFET drive stages, and the Teensy microcontroller.

The hall-effect sensor portion of the schematic consists of a 5-pin polarized JST connector for power, ground, and the three inputs. The inputs are pulled up to 3.3V (the microcontroller digital IO operating voltage) by a 10k resistor because the hall effect sensors are open-collector. They are filtered with a simple RC low-pass filter with a cutoff frequency of 16 Khz and a 10% to 90% rise time of 22 µs. These values can easily be changed later as necessary by replacing either the resistor or capacitor (or both) from the board.

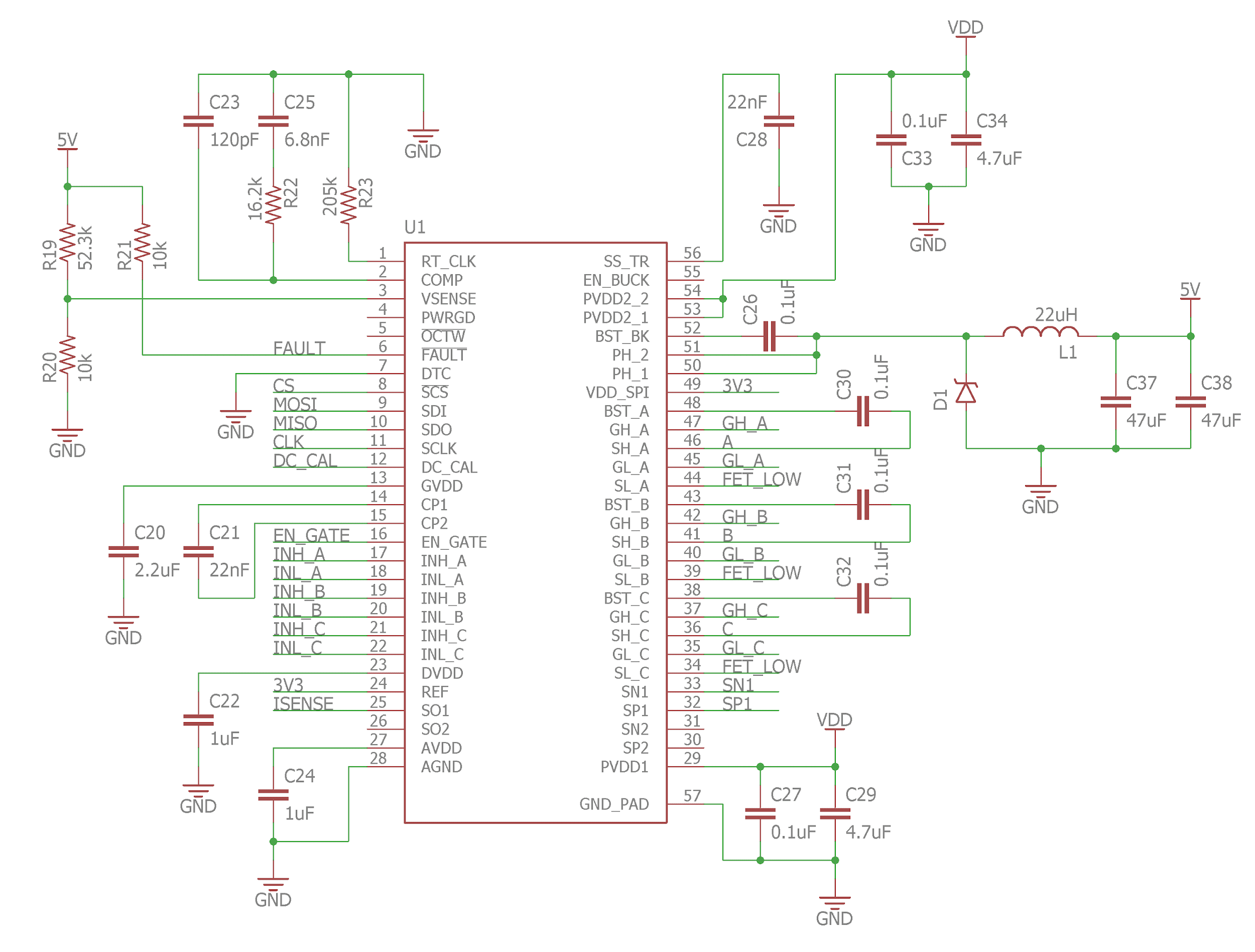


*Figure 4: Hall-effect sensor input schematic*

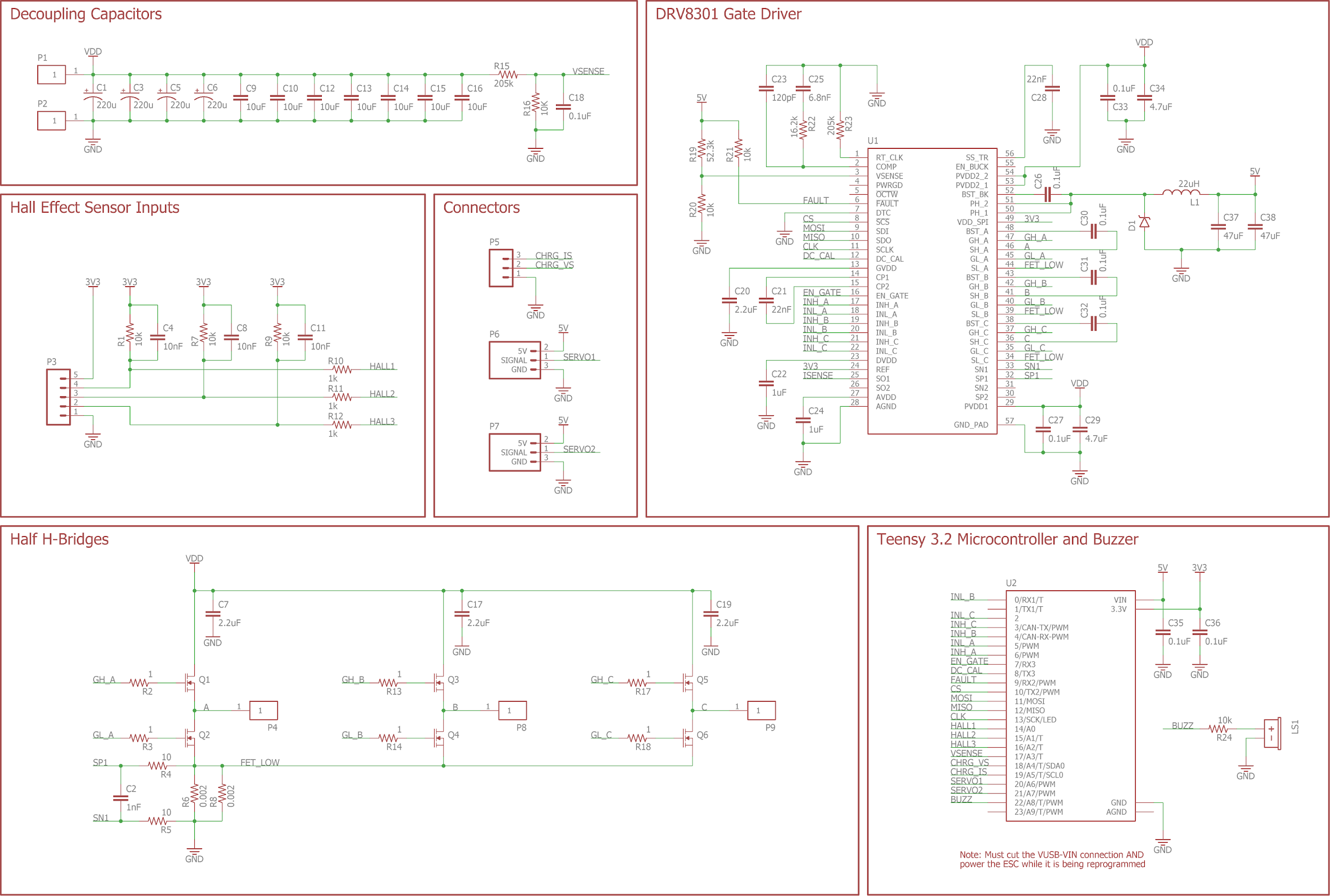
The drive stages portion of the schematic consists of the canonical configuration of three half-H bridges used to pull each of the motor terminals to V+ or ground. The MOSFETs used can conduct up to 240A. At the bottom, the node labeled “FET\_LOW” connects the source terminals of the low side MOSFETs to ground through a pair of shunt resistors in parallel, which are used to measure the sum of the currents through the motor.

*Figure 5: Drive Stages schematic*

The DRV8301 gate driver portion of the schematic is for the most part the schematic recommended in the Texas Instruments application note. The ICs integrated buck converter is used to produce the 5V rail for the rest of the chip.

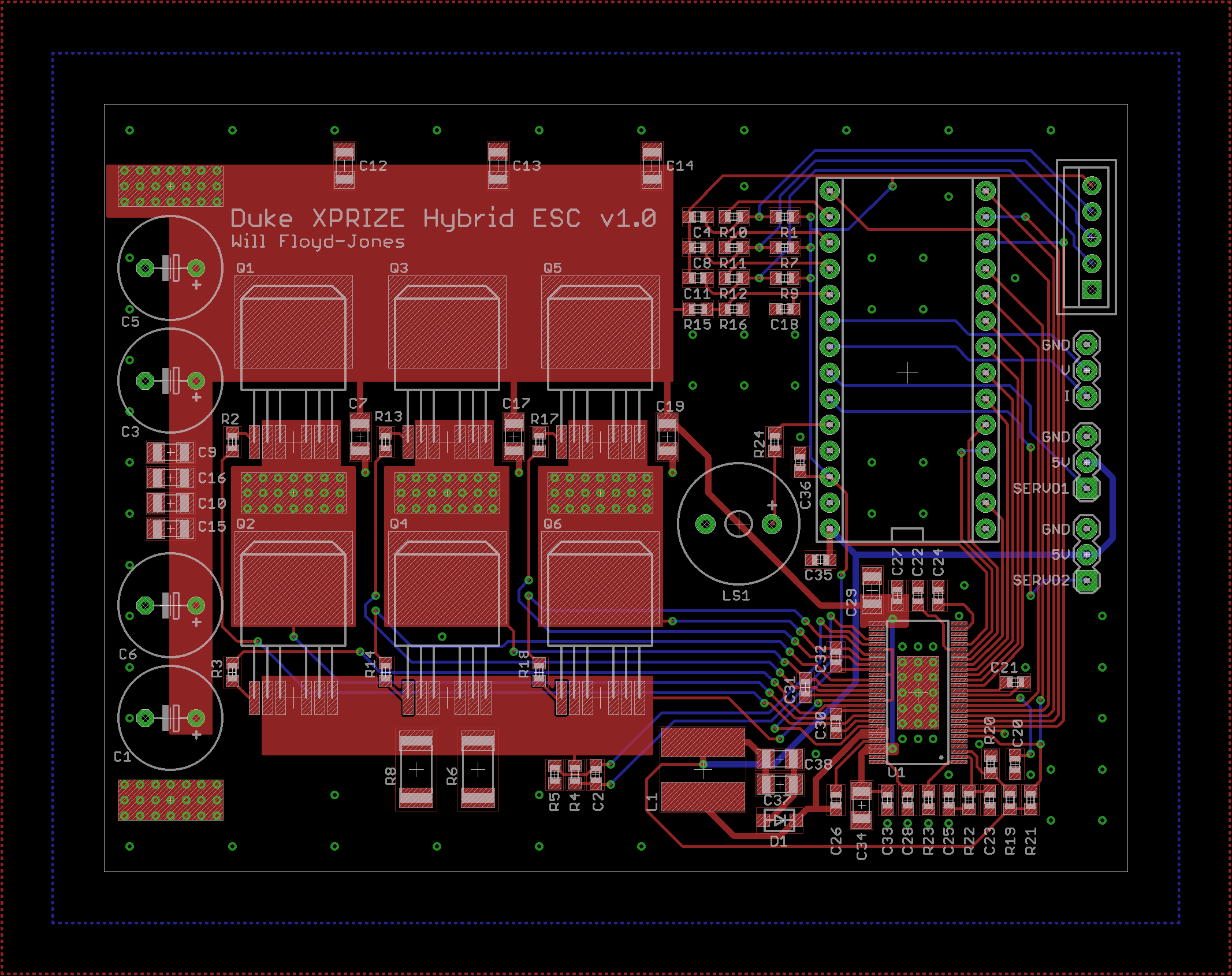


*Figure 6: Gate Driver IC Schematic*

****

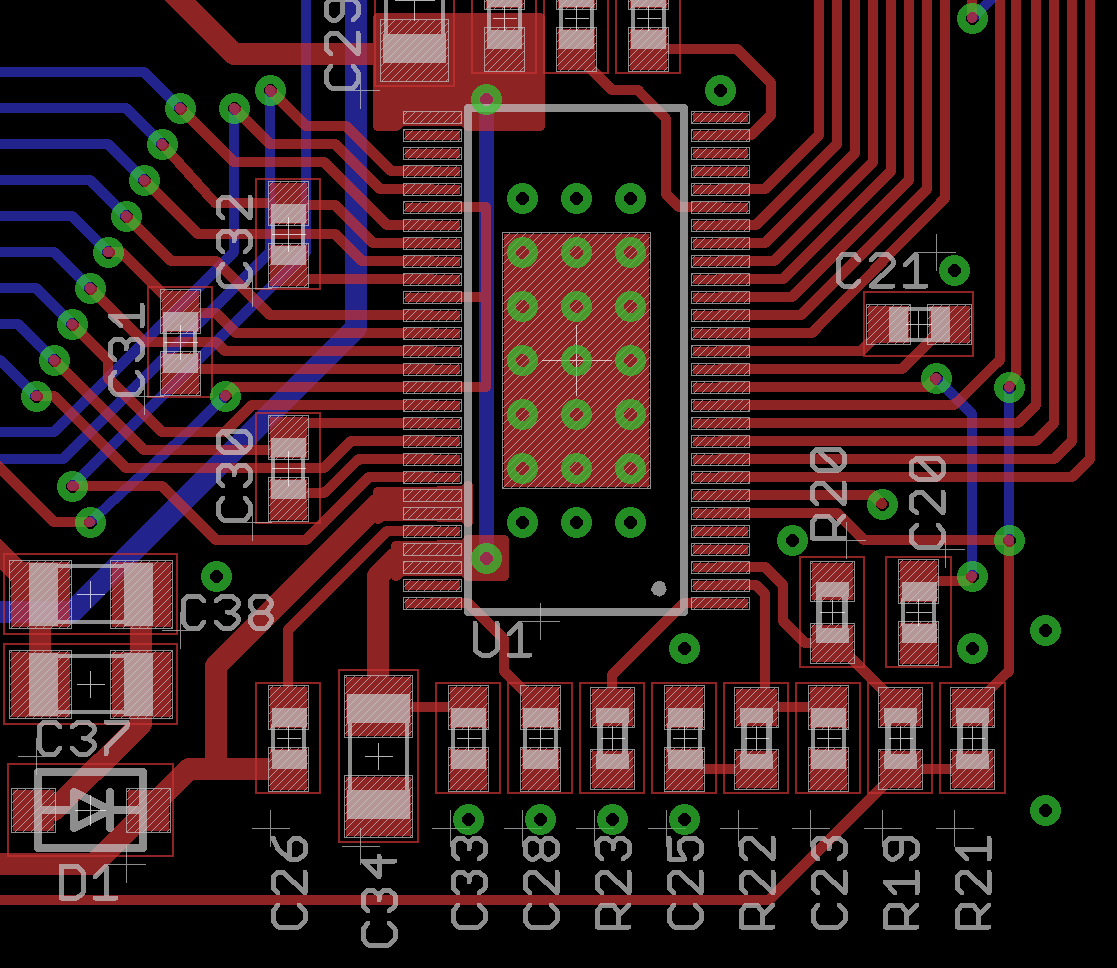
**Layout**

The PCB is laid-out with the drive stages on the left side and the microcontroller and gate driver IC on the right side. The drive stages are positioned such that enough current can flow between the motor terminal and the MOSFET gates; the traces connecting the high and low side MOSFETs to the motor terminal are each about 12 mm wide. Given that the circuit board will likely be fabricated with 1 oz/ft2 copper and most of the trace is actually covered with solder, each trace should easily conduct more than 30A. Under the drive stages is the low-side current sensor system, which shunt current through a pair of 2 mΩ resistors of size 2512 in parallel and amplify the voltage using the gate driver IC. This copper pour should be large enough to conduct the current necessary to start the gas motor. The left side of the board also contains the decoupling capacitors on the input and pads to connect to the silicon wires of the battery connector and the motor phase wires. These pads are wide enough to accommodate 10 AWG wire and have vias through them to help prevent mechanical stress from ripping the pad off the PCB.



*Figure 7: PCB Layout*

On the right side of the board, the gate driver is fanned out such that the ground pour on the bottom side of the board is continuous and can reach the IC. There are a large number of stitching ground vias on and around the IC pad to provide thermal relief minimize ground impedance. Pins on the gate driver IC which conduct relatively higher currents are surrounded with polygon pours. The right side of the board also contains connectors for the Teensy microcontroller footprint- a pair of 14-pin 0.1” pitch female headers. There are no components placed underneath the Teensy footprint in case it is desired to solder the microcontroller directly (without female headers), as this would diminish the likelihood that there should be a reason to remove the microcontroller to make board modifications. The connectors to the pair of servo motors, to the hall effect sensor cable, and to the buffer battery’s voltage and current sensors are positioned on the far right side of the board. This way the five power leads will come off the left side of the board, and the four communication cables will come off the right side of the board.



*Figure 8: Gate driver portion of PCB layout*

The board is 75 mm wide and is intended to be housed in the extruded aluminum Hammond Manufacturing enclosure shown below in Figure X. It will be inserted into the slots of the enclosure and secured by the endcaps. I will likely 3D print the endcaps so that the enclosure can be securely fastened to the drone frame, the PCB (which is about 20 mm shorter than the enclosure) doesn’t move around, and all the the cables can exit the enclosure cleanly.

The PCB should be able to fit into the 16.31 mm space between the top of the bottom slot and the top of the enclosure, as the tallest component is the aluminum decoupling capacitors at 13.5 mm from the surface of the board.

**Code**

The board has not yet arrived and has yet to be assembled, but I have written some code which should serve as a good baseline once I am ready to begin testing. The functions I have written and their descriptions are below.

1. **hall\_ISR()**: Called as a pin-change interrupt every time one of the three hall-effect sensors changes state. Determines the state of the hall sensors by calling get\_hall\_state(), uses a hard-coded look up table to figure out how to switch the MOSFETs, changes the states of the MOSFETs accordingly by calling write\_state().
2. **get\_hall\_state()**: Called by hall\_ISR(). Reads each of the three hall effect sensor pins and bit-shifts the results to assemble them into an unsigned integer.
3. **write\_state()**: Called by hall\_ISR(). Takes in a position and uses a switch statement to write the appropriate state to the high-side MOSFETs and the low-side MOSFETs.
4. **write\_low()**: Called by write\_state(). Writes the given value to each of the three low-side MOSFET pins on the gate driver IC.
5. **write\_high()**: Called by write\_state(). Writes the given value to each of the three high-side MOSFET pins of the gate driver IC. There are three cases which apply current to the stator- in each case two of the MOSFETs are off while the thirds is PWMed according to the “throttle” value. The throttle will likely be a global variable that is some function of time. It will ramp up from 0 to 255 such that the gas motor can start.

void hall\_ISR()

{

uint8\_t hall = get\_hall\_state();

uint8\_t pos = hallOrder[hall];

if (pos > 6)

{

write\_state(255);//error

return;

}

pos = (pos + HALL\_SHIFT) % 6;

write\_state(pos);

}

uint8\_t get\_hall\_state()

{

uint8\_t hall = (digitalReadFast(HALL3) << 2) | (digitalReadFast(HALL2) << 1) | (digitalReadFast(HALL1) << 0);

return hall & 0x07;

}

void write\_state(uint8\_t pos)

{

write\_high(0);

write\_low(0);

switch(pos){

case 0://LOW A, HIGH B

write\_low(0b001);

write\_high(0b010);

break;

case 1://LOW A, HIGH C

write\_low(0b001);

write\_high(0b100);

break;

case 2://LOW B, HIGH C

write\_low(0b010);

write\_high(0b100);

break;

case 3://LOW B, HIGH A

write\_low(0b010);

write\_high(0b001);

break;

case 4://LOW C, HIGH A

write\_low(0b100);

write\_high(0b001);

break;

case 5://LOW C, HIGH B

write\_low(0b100);

write\_high(0b010);

break;

}

}

void write\_low(uint8\_t phase){

digitalWrite(INLA, (phase & (1<<0)));

digitalWrite(INLB, (phase & (1<<1)));

digitalWrite(INLC, (phase & (1<<2)));

}

void write\_high(uint8\_t phase){

switch(phase){

case 0b001: // Phase A

analogWrite(INHB, 0);

analogWrite(INHC, 0);

analogWrite(INHA, throttle);

break;

case 0b010: // Phase B

analogWrite(INHA, 0);

analogWrite(INHC, 0);

analogWrite(INHB, throttle);

break;

case 0b100:// Phase C

analogWrite(INHA, 0);

analogWrite(INHB, 0);

analogWrite(INHC, throttle);

break;

default://ALL OFF

analogWrite(INHA, 0);

analogWrite(INHB, 0);

analogWrite(INHC, 0);

}

}

**Next Steps**

Once I get the boards, I will complete the following steps:

1. **Assembly**: Assembling 1 board using solder paste stencils and testing the power rails and gate driver IC will take about 4 hours.
2. **Getting a motor to spin**: I anticipate this will take about a week. Let’s hope I don’t blow too many things up. This step will involve writing a decent amount of code to interface the Teensy microcontroller with the gate driver IC via SPI and GPIOs. It also requires that I figure out how to properly construct the look-up table to translate hall-effect sensor states to drive stage states, which involves some trial and error.
3. **Integration**: Getting *our* motor to spin and integrating the ESC to the rest of the system. This should take about a week as well. There is already existing Arduino code to turn the servos to properly start the engine.