TeensyESC: Brushless Motor Controller Design and Analysis

Aaron Becker

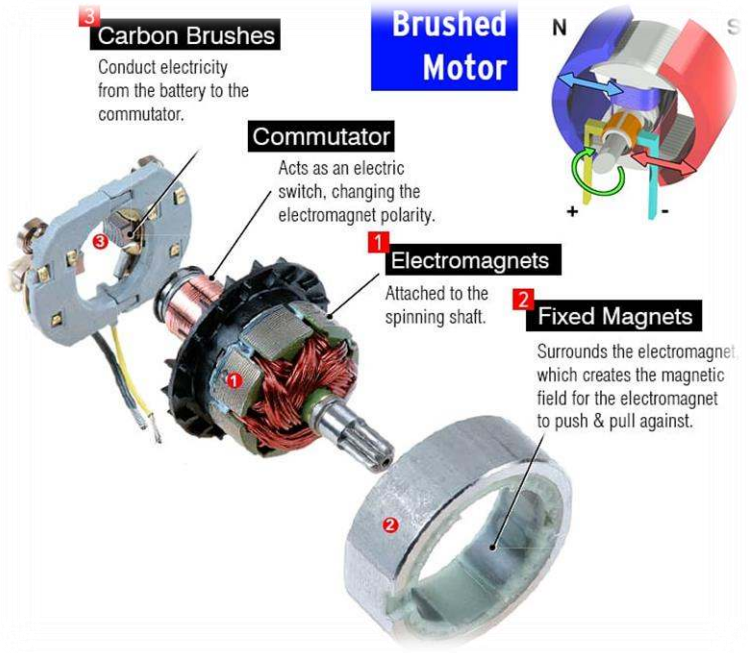
February 2023

1 Project Overview

1.1 Motors

Motors are ubiquitous in society today, having a significant impact on various industries. They are used for a huge range of applications, including driving manufacturing processes, powering transportation systems, and generating electricity. Often, the connection between electricity and the rotational motion generated is an afterthought; it's easy to make the assumption that electrical work generates mechanical work with perfect efficiency.

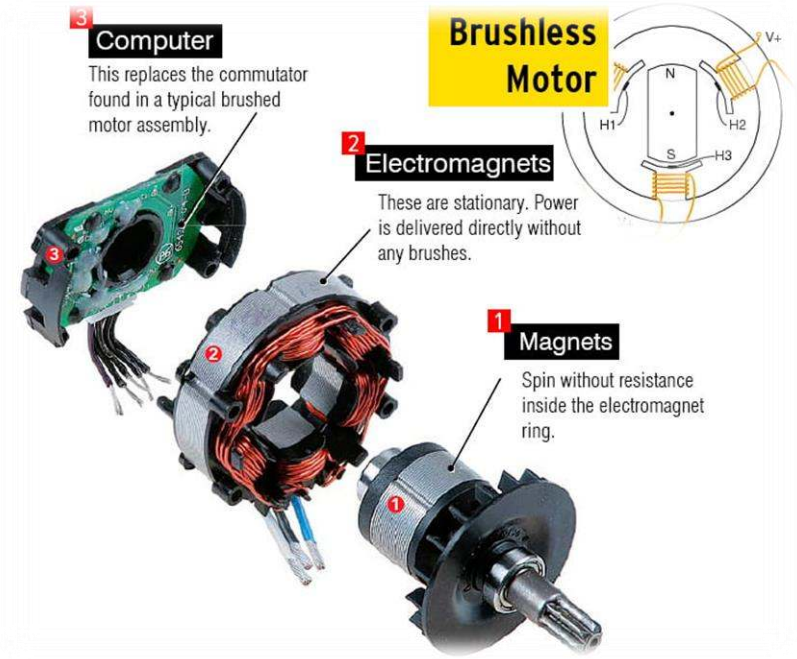
However, this is untrue: the link between electricity and mechanical work is not so simple. There are actually two kinds of motors, which perform this conversion in different ways: classical "brushed" motors and "brushless" motors. Brushed motors (see Fig. 1) are called as such because they use a mechanical brush to transfer electrical energy to the rotating part of the motor (also known as the rotor), which is simple and inexpensive. However, this method has trade-offs: brushes are subject to mechanical wear over time, which reduces the efficiency and lifespan of the motor over time.



Above, Fig. 1: A classical brushed motor and its internal parts, including carbon brushes

On the other hand, brushless motors (also known as BLDC motors, see Fig. 2) are made without any brushes; the internal coils are energized by a special circuit known colloquially as an ESC (or Electronic Speed Controller). They are now found in many different types of devices, large and small, from commercially available drones to electric cars.

Designing a motor this way has several advantages, including greater efficiency over a larger speed range. However, the tradeoff is that the driving circuit becomes much more complex.



Above, Fig. 2: A brushless motor and its internal parts, including rear PCB that includes encoder

1.2 Electronic Speed Controllers

Now that the fundamental difference between brushed and brushless motors is clear, the problem also becomes clear: how do we generate the correct signals at the brushless motor phases to replace the brushed commutation process?

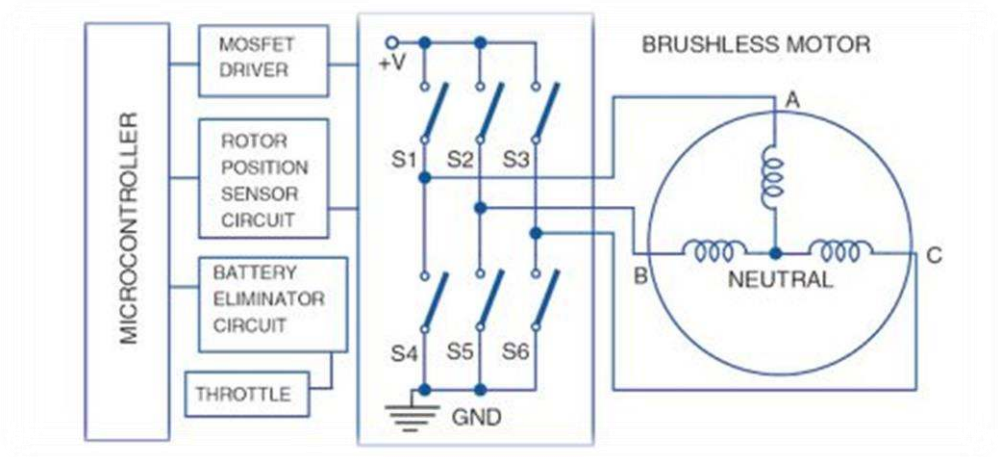
The answer is MOSFETs, or electrical switches that allow us to take a small voltage from a control computer, and switch large currents. The problem here is that motors are not particularly easy to deal with; there are three main design problems designing a controller we have to overcome. This overview serves to explain things at a high level, not delve into some of the deeper math and analysis required to truly understand what's going on.

- Inductive load, back-EMF: due to the magnetic field created when energizing the coils, motor windings "resist" changes in current.

- Heat dissipation: No component is perfect; energy is wasted in the system as heat, which must not exceed physical limits of components
- Signal integrity: Large magnetic fields can create EMI issues for sensitive sensor traces on PCBs.

Given all of this, the high-level diagram for most ESCs is below (*Fig. 3*). Here, we can see the six MOSFETs that make up the main switching part of the speed controller. These allow connecting each phase to either ground or the positive supply voltage (but not both at once!).

In addition, there are other parts of the circuit that are important as well. The rotor position sensor, in particular, is critical in order to generate the right pulses for the motor at the right time.



Above, Fig. 3: *A simple brushless motor driver circuit, at a high level*

With all of this in mind, I proceeded to the design phase, which is described in the next section.

---

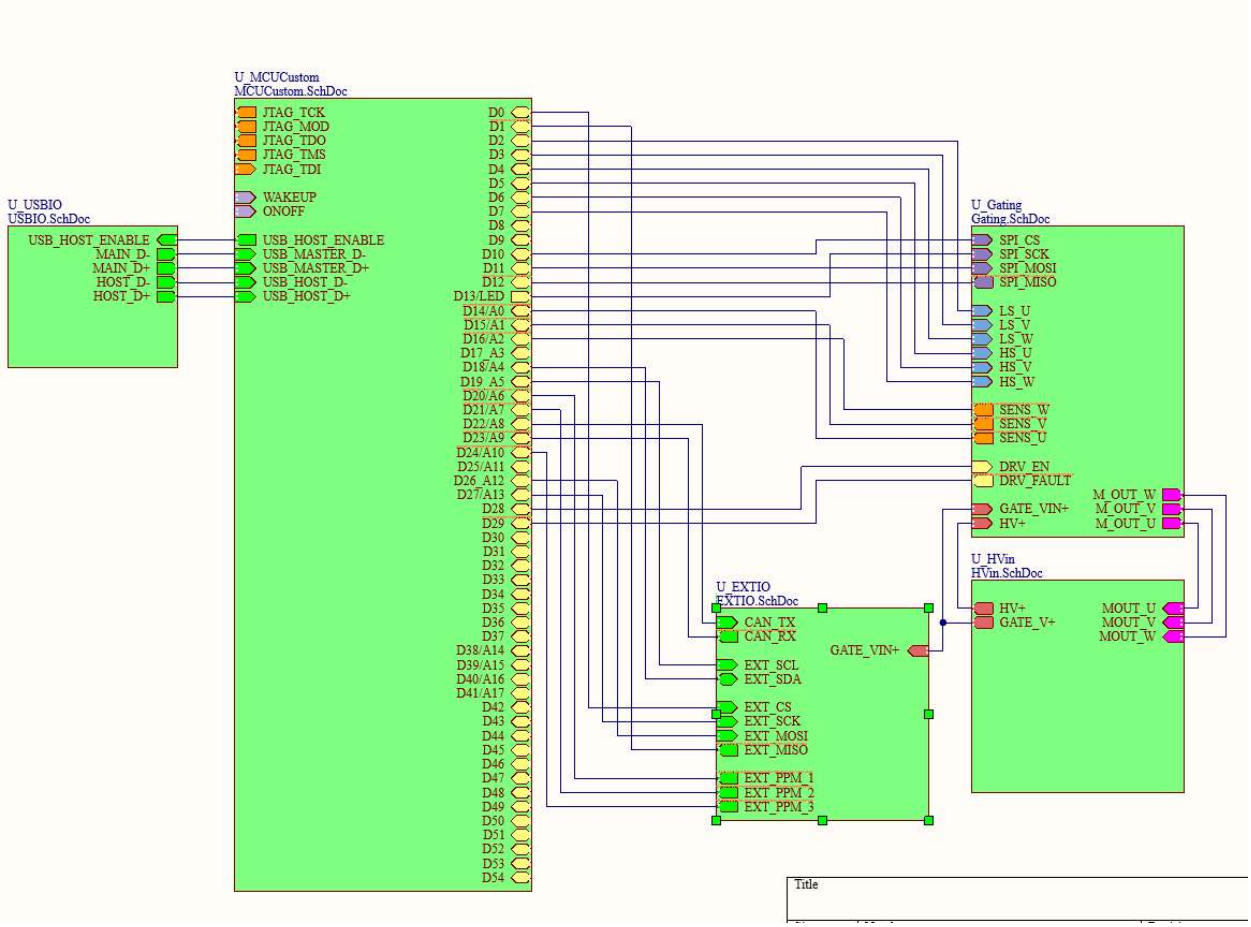
2 Schematic Design

2.1 High Level Schematic

I used Altium Designer as my program of choice, in accordance with the expectations of the class.

One of its features is hierarchical schematics, where individual "sheets" are linked together logically instead of one massive sheet for complex designs. This lets me organize high level features, like block-to-block interconnects, relatively easily without the mess of making connections in one sheet alone.

Below is the finalized top sheet for my design (*Fig. 5*):



Above, Fig. 4: Top sheet of schematic, showing all of the lower level sheets (green) linked, as well as connections between sheets

The parts of the entire design consist of the following:

1. **HVin**, which takes in a high drive voltage of up to 48V DC and steps it down to 12V and 5V DC. Also contains the motor output terminals.

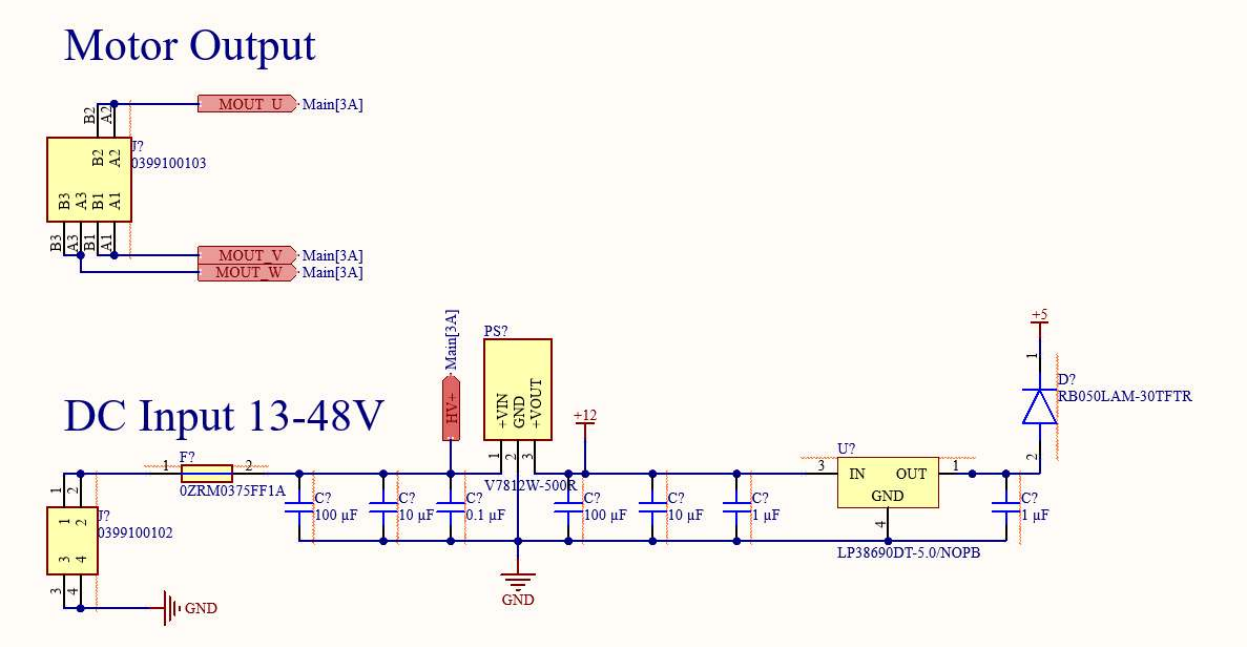
2. **EXTIO**, which contains several different types of connectors to interface with external peripherals and sensors. These include I2C, SPI, CAN, and PPM control, to allow the design to be used in a wide range of scenarios.
3. **Gating**, which contains the MOSFET gate drivers, switching circuits, current sense resistors, and motor interfacing
4. **MCUCustom**, which contains the microcontroller of choice, an IMXRT1062. This is the same microcontroller that the Teensy 4.1 uses, and is based on the ARM Cortex-M7 architecture. It runs at a whopping 600MHz, incredibly fast, compact, and energy efficient.
5. **USBIO**, which contains the two USB ports (one for uploading code, and one USB 2.0 host for connecting peripherals such as keyboards, flash drives, etc.)

In the following sections, each of these will be explored in depth.

2.2 HV In

The motor controller is designed to run at up to 48V DC, as that is the maximum voltage of the TMC6200 chip. However, 48V is much too high to run any sort of processor or sensor. The purpose of this part of the schematic, then, is to safely regulate this high voltage down to lower voltages that are useful across other parts of the board.

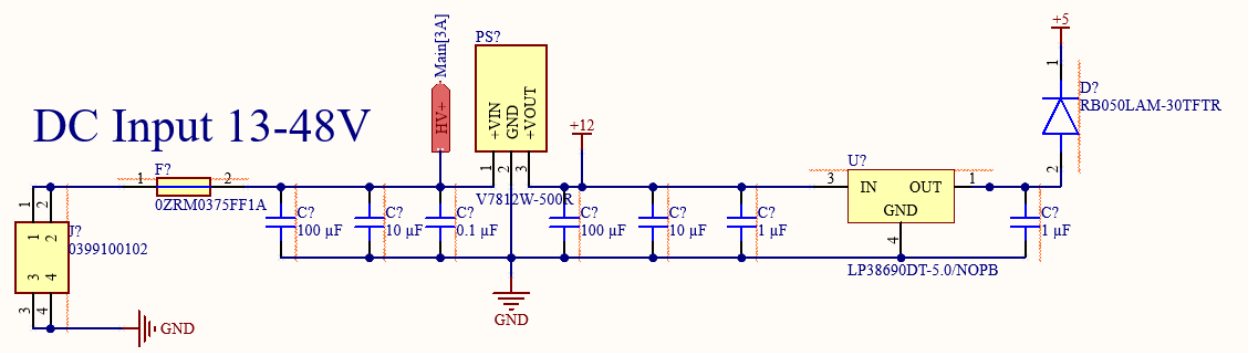
See the finalized schematic below:



Above, Fig. 5: HV in schematic, showing motor outputs and HV DC input.

One part of the schematic is relatively simple: the motor output. This simply takes the three phase voltages generated by the gate driver and switching circuit and outputs it to a high-current rated screw terminal.

Next, there is also the DC Input section. This is slightly more complex, as there are some other design considerations to keep in mind. See below in more detail:



Above, Fig. 6: DC Input section only

Here, there is a PPTC (Polymeric Positive Temperature Coefficient Device) on the positive terminal of the input to the left. This device has a known resistance, and as such, with increased current dissipates more heat. If the current gets sufficiently high over a long enough time period, it will heat to a critical temperature where its resistance increases dramatically, essentially cutting off current flow.

Moving to the right, the middle device is an off-the-shelf DC-DC buck converter that is through-hole, which steps the 48V down to 12VDC. This simplifies the design (while increasing cost), since buck converters are very sensitive to electrical noise and layout considerations. The 12VDC generated is used to drive the MOSFET gates.

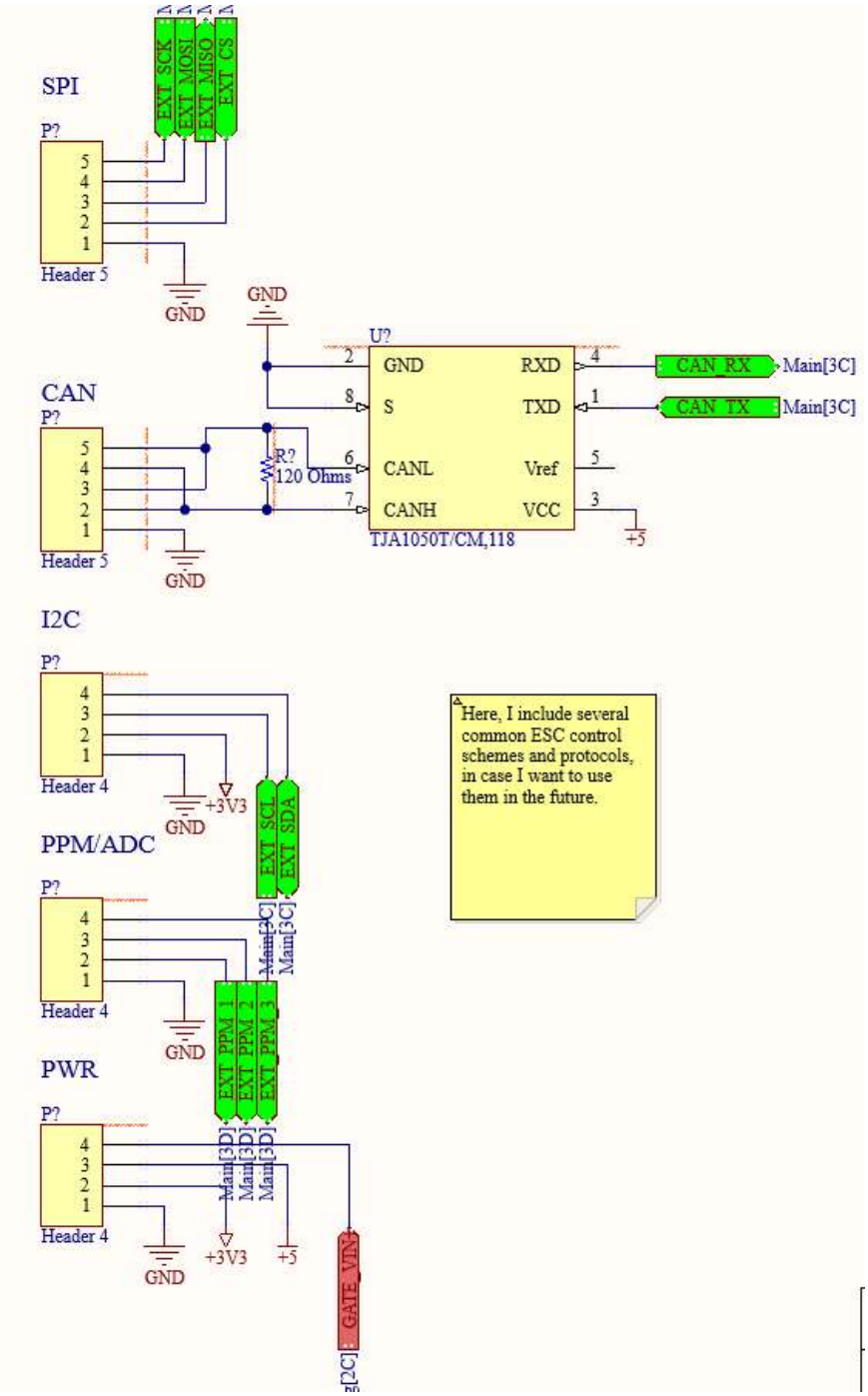
Last, there is a LDO (Linear Dropout Regulator) which produces 5VDC from the stepped-down 12V. This is used as an input to the gate driver's charge pump as well as the 3.3V regulator for the main microcontroller. There is also a Schottky diode, which blocks reverse current flow into the voltage regulator, as the 5V bus is also connected to the USB ports.

2.3 EXT IO

In addition to USB functionality, I also wanted to include a number of different common ports used in all manner of different applications to enable me to test this with many different kinds of devices. The kinds of IO I chose include:

- **PPM** (or **PWM**) modulation: a digital or analog signal that conveys target speed
- **I2C**, **SPI**: Common sensor protocols
- **CAN**: Common in automotive applications
- **USB**: Useful for keyboard and mouse, other HID devices
- **PWR** header: Breaks out the different voltage levels available to connect to external devices

Below is the schematic:



Above, Fig. 7: EXT_IO schematic, showing the different ports

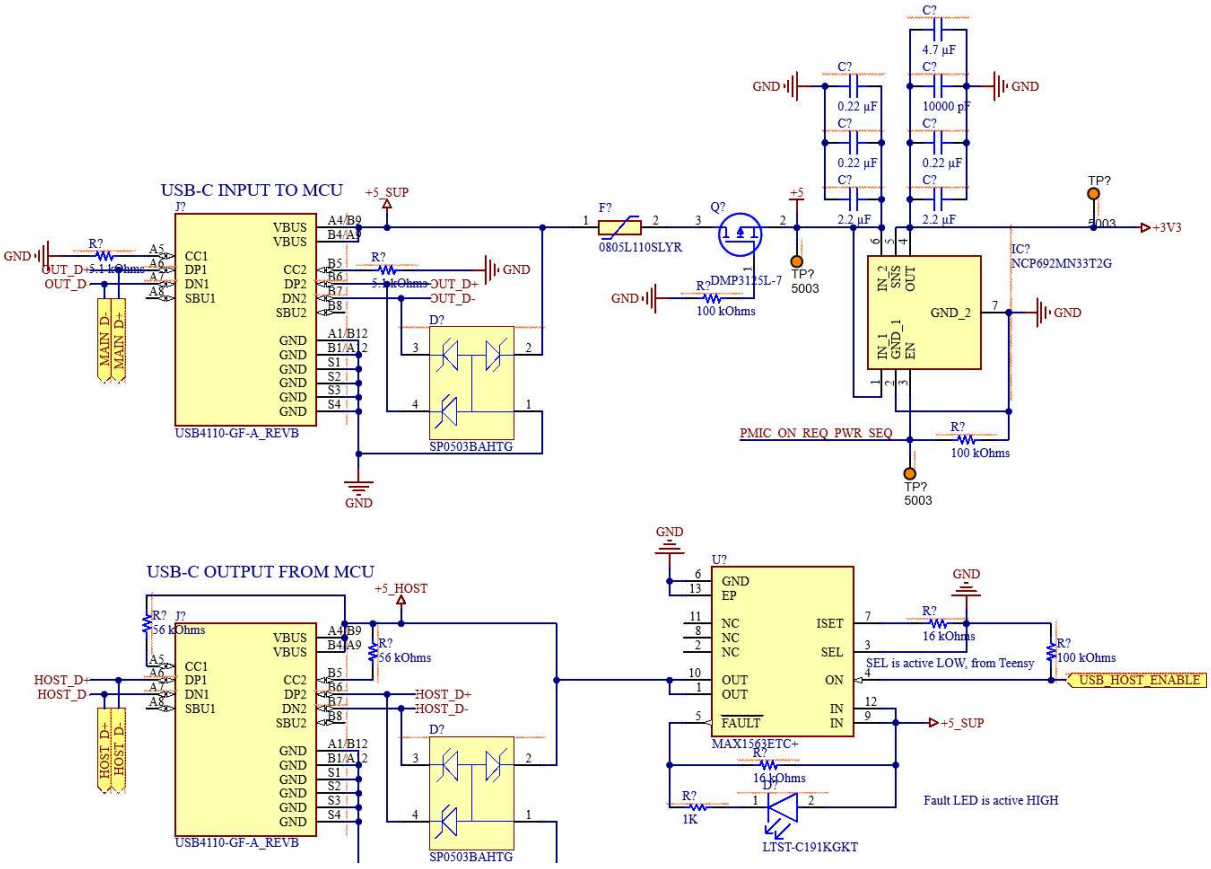
Although the IMXRT1062 (the microcontroller) includes a CAN protocol controller, it does not actually include the CAN transceiver required to generate CAN traffic. For that, I selected the TJA1050 part, which takes the logic-level signals from the microcontroller and generates the required differential pair voltages.

The other protocols are also hardware interfaces, as the IMXRT1062 has many different options for ports. The PPM channels are just mapped to standard ADC inputs, so are capable of reading either a digital PWM signal or an analog voltage.

2.4 USB IO

As mentioned previously, there are actually two USB Type-C ports on the board. One is a USB Full Speed capable input port, which is connected to the microcontroller's input port, and is used for uploading code. The second port is a USB host port, and supports Type-C handshaking with a MAX1636 chip, which supports up to 1A of current delivery in this configuration. The intention of the host port is to allow connection of useful peripherals, say a keyboard for example, to control the ESC.

Below is the schematic:



Above, Fig. 8: USBIO schematic, showing both USB Type-C ports and supporting circuitry

Looking deeper at the source port, there are standard TVS diodes to provide some static protection, as well as another PPTC device (sized for 5V) to protect your computer's port in case too much current is being drawn.

Next to this, there is a P-Channel MOSFET that provides reverse-polarity protection. In the case that the USB VBUS and ground are reversed, the gate will be pulled logic high, which turns the MOSFET off and blocks current flow. In fact, I also did some simulations to confirm that this circuit works, which are described in the Simulation Appendix at the end of this document.

There is also a NCP692-series 3.3V linear regulator, which is actually controlled by the chip itself via the PMIC_ON_REQ_PWR_SEQ pin, seen just below it. The purpose of this net is that the chip goes through a complex startup sequence when voltage is applied, and ensures that the input voltage is stable before turning on the regulator. The exact details are outside the scope of this paper, but if you are interested, PJRC (the designer of the Teensy 4.1) has written an excellent document here, under the header "Power Up Sequence".

On the source side, I'm using the MAX1563 USB current switch, which implements short protection, current negotiation, and current limiting to ensure that the host device operates within a safe voltage and current range. It also has a Fault LED for easy indication of any issue.

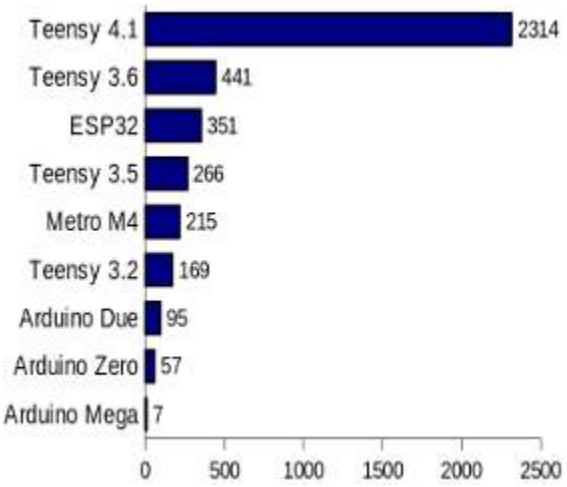
Both ports communicate at full USB 2.0 speeds, which is extremely fast for a microcontroller at around 480 Mb/sec. This means that code uploads extremely quickly, and one advantage of picking a more advanced chip like this.

2.5 Custom MCU: IMXRT1062

One extremely popular microcontroller series on the market today is the Teensy line by PJRC, which is much more powerful then standard Arduino boards. The most recent board released, the Teensy 4.1, is based around NXP's IMXRT1062 chip, which is extremely powerful, fast, and relatively inexpensive. The chip is based on a modern ARM Cortex-M7 processor, as opposed to Arduino's ancient AVR architecture.

Above, Fig. 9: Teensy 4.1, from PJRC's website. This board is tiny, about 1/3 the size of a standard Arduino Uno.

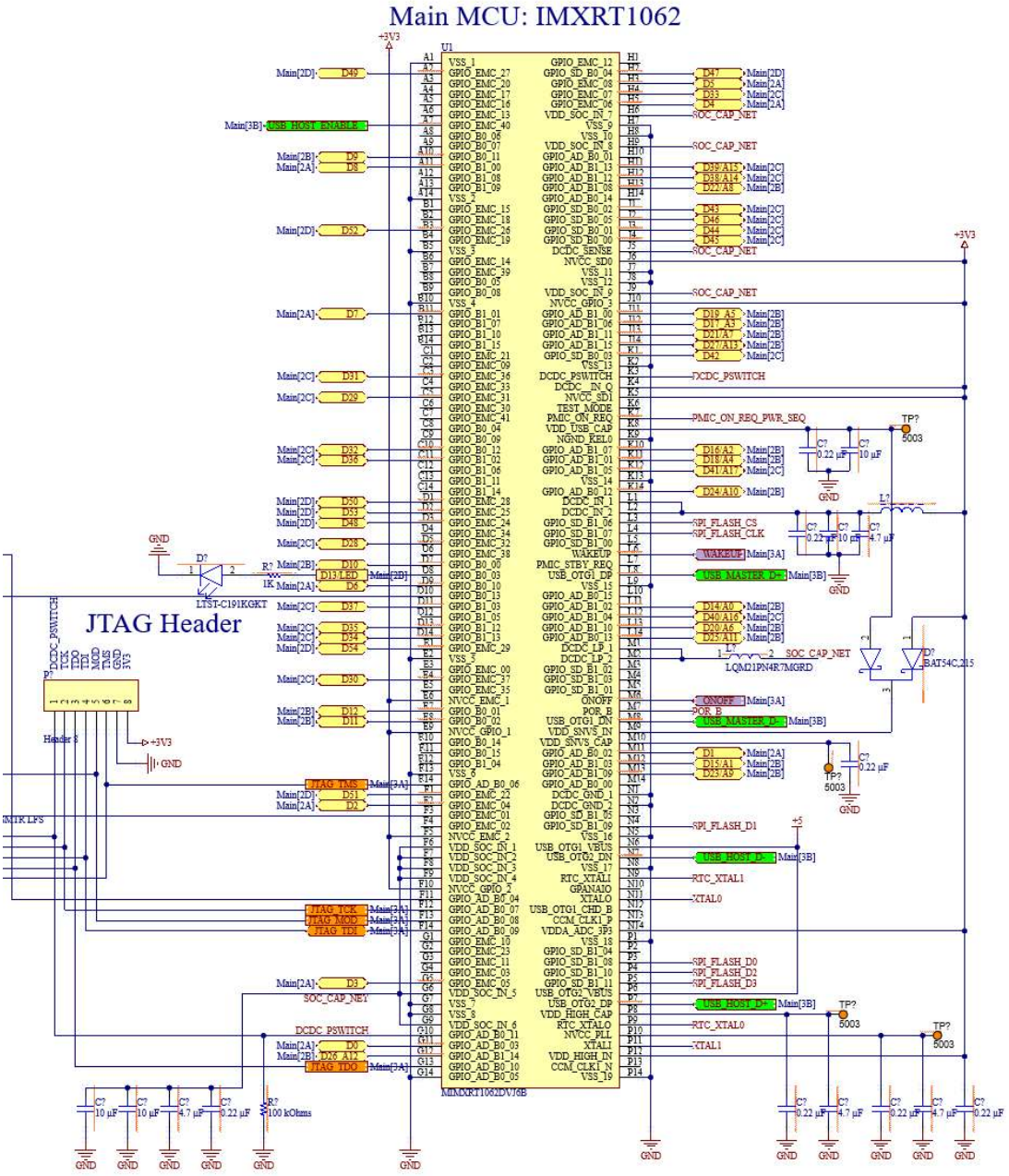
For a point of reference when talking about the sheer speed of this chip, it runs at 600MHz (you read that right), features built-in CAN bus and QSPI flash compatibility, and supports USB device and host at 480Mbit/sec speeds. It so far outclasses the Arduino Uno that the Uno is not even listed on the below comparison chart of various microcontroller boards.



Above, Fig. 10: PJRC CoreMark benchmark of Teensy 4.1 versus multiple other industry-standard alternatives

The only disadvantage is its complexity, both because it only comes in a BGA (Ball Grid Array) package rather than an easier to route and solder LQFP or TQFP package, and because of its sheer pin count, in a 14x14 array numbering 196 pins. I decided to take on the challenge for this project both because it will allow me to use this board in a control system running at a higher frequency, and because I intend to use this design as a stepping stone for future designs based around this chip.

To that end, it did increase project complexity and time, but I think it paid off, as the result is that I now can use this chip in custom projects.



Above, Fig 11: IMXRT1062JV6B schematic

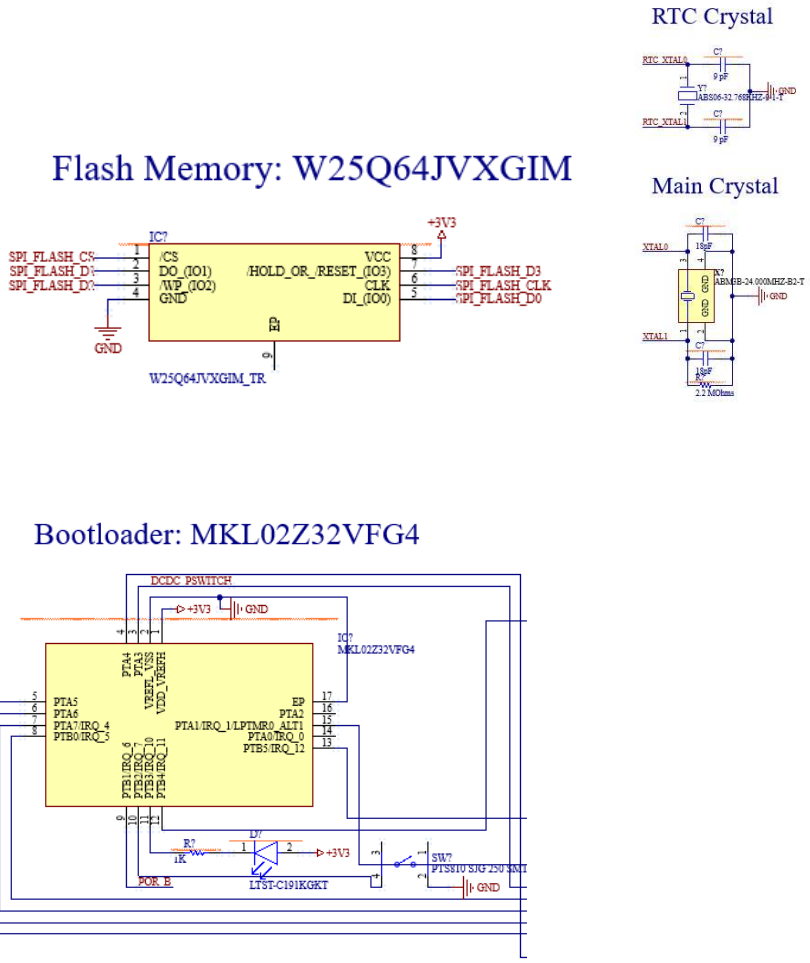
Here, instead of breaking out just the pins I'd need for this project, I decided to break out all of the pins used in the Teensy 4.1 design, so that I'd be able to use this schematic design in other projects with different needs.

The chip itself takes 3.3V supply voltage and also supports a RTC (Real-Time Clock) to keep accurate measurement of time. There are two external crystals, one running at 24MHz for the main clock and one running at 32.768 KHz for the RTC. This is a great video by Steve Mould on why

exactly the RTC crystal is 32.768 KHz if you are interested:
https://www.youtube.com/watch?v=_2By2ane2I4

In addition, there are a number of test points broken out so that if there are issues with the design, I can debug them.

There also are a few other supporting components, including a flash memory chip, the aforementioned crystal oscillators, and a bootloader chip supplied by PJRC that configures the IMXRT1062 chip (which comes blank from the factory) with proprietary firmware that enables all of the features mentioned above. All of the various components are below:

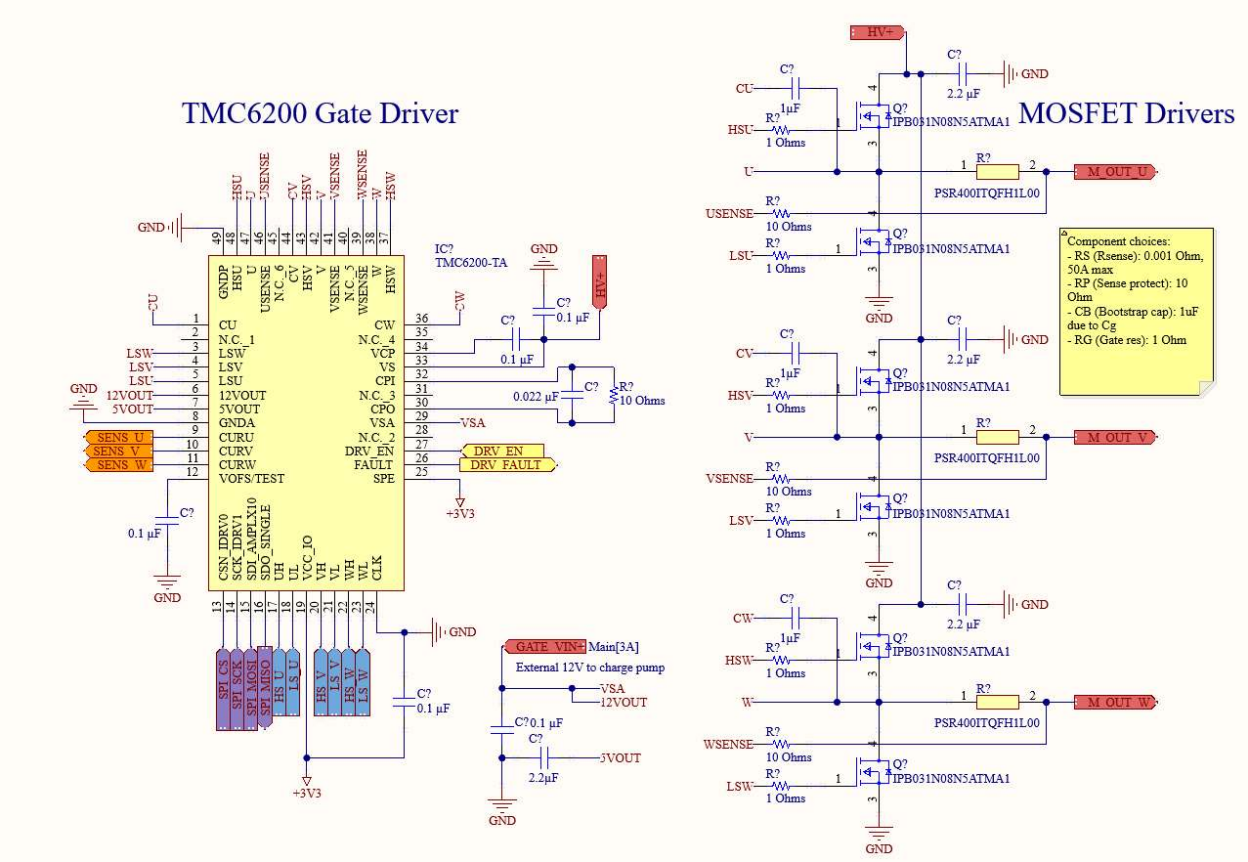


Above, Fig 12-15: Flash memory, bootloader chip, and RTC/main crystals which support the IMXRT1062 chip. Note that the bootloader communicates over JTAG, which is also broken out to test pins for debugging

2.6 Gate Driver and MOSFETs: TMC6200

All of this circuitry is helpful, but it does not actually yet accomplish the task of energizing the motor phases to cause rotation. That task falls to the next part of the circuit, which consists of the TMC6200 gate driver chip which takes in an input signal from the microcontroller and produces pulses sufficient to drive the six IPB031N08N5ATMA1 (from here on referred to as IPB031) MOSFETs which connect the motor phases alternately to ground and the supply voltage.

The full schematic is below:



Above, Fig 16: Gate driver and MOSFETs

There are two voltage inputs to this sheet; the HV+ that drives the motor, and GATE_VIN+, which drives the MOSFET gates. These are denoted in the red color at the top and left side of the schematic. On the right, the three motor outputs are connected to a pair of MOSFETs, one for the high and one for the low side. There is also a 1 milliohm current shunt resistor per phase on the low side of each MOSFET pair, which when combined with the built-in current amplification hardware in the TMC6200 produces a measurable voltage at the microcontroller's ADC pins that will eventually be part of the FOC feedback loop.

The TMC6200 was chosen for this project as an alternative to the traditional DRV8302, due to availability concerns with that chip since it is quite popular. I also wanted to explore the feasibility of using this chip in future designs, since it is also inexpensive and pairs with other exciting chips from Trinamic, which can even do built-in FOC in hardware.

The IPB031 MOSFETs were selected due to their low Rds_ON, which minimizes active losses. Supply availability is a major concern, though; more popular options were impossible to find in the current market. The tradeoff made here is both cost (they are expensive), and that their gate capacitance Qg is quite a bit larger then the reference design, necessitating changes to both the boothstrap capacitors, denoted CB in the schematic, and the gate protection resistors. They can also handle up to 120A of theoretical drain current each if sufficient cooled, although the shunt

measurement circuitry is only designed to measure up to 50A with safe headroom. See the table below for an example of the tradeoffs taken into account when picking components as part of the gate driver circuit:

CHOICE OF R_{SENSE} AND AMPLIFICATION DEPENDING ON MAX. COIL CURRENT				
R_{SENSE} [mΩ]	*Amplification factor*	*Current range [A]*	*RMS motor current limit [A]*	*Max. power dissipation of* R_{SENSE} [W]
150	10	0.7	0.5	0.05
150	5	1.3	1	0.15
100	5	2	1.5	0.23
75	5	2.6	2	0.3
33	10	3	2.2	0.16
25	10	4	3	0.23
50	5	4	3	0.45
33	5	6	4.5	0.67
15	10	6.5	5	0.38
25	5	8	6	0.9
10	10	10	7.5	0.56
5	10	20	15	1.1
2.5	20	20	15	0.56
2.5	10	40	30	2.3
1	20	50 (40@1.65V ofs.)	37	1.4

Above, Fig 17: Table provided by Trinamic showing different current shunt options; this is only one of several different components selected by careful examination of the datasheet. Look at the Altium source file for notes on exact values of each component.

Although untested, I believe that the above schematic is a good starting point for testing a motor controller of this magnitude (around 2kW peak power). Once built, further investigation into switching characteristics and losses will be necessary to ensure the controller is operating efficiently. Physical layout, which has not been completed at this time, also makes a huge difference in efficiency and losses in the final result, and needs to investigated further. I also thought about simulating a single motor phase in LTSpice, but decided that routing considerations made such a large difference in the results that the results would not be particularly conclusive.

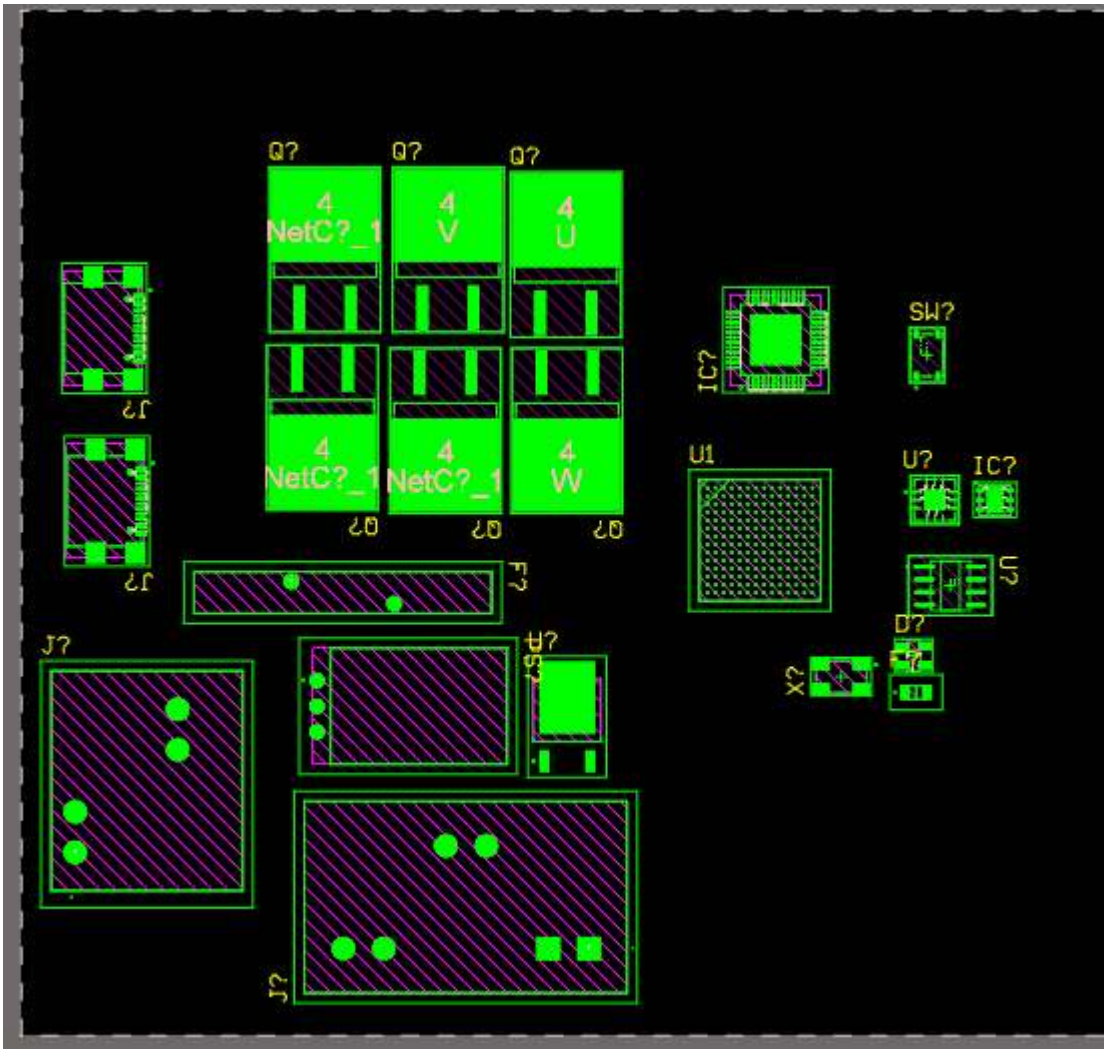
---

3 Layout

3.1 Early Prototype Layout

While the schematic capture of the project I feel is mostly finalized, the layout is still unfinished an active area of development going forward, as I fully intend to finish this project.

Below is an extremely early prototype layout that shows the main microcontroller, inputs, and all six MOSFETs. There are no passives imported yet in this version of the design; I usually like to do a iterative process of placing large components while simplifying connections.



Above, Fig 18: Early prototype layout, no passives or airwires visible.

Without having finished it yet, I intend to make this a 4-layer PCB, since I need that many layers to reach some of the inner BGA pins, even with ideal fanout. I also plan to have two different sides of

the board, one that is LV (logic level, analog only) and one that is HV to minimize the effects of EMI or capacitive coupling on the sensitive analog electronics like the current shunts.

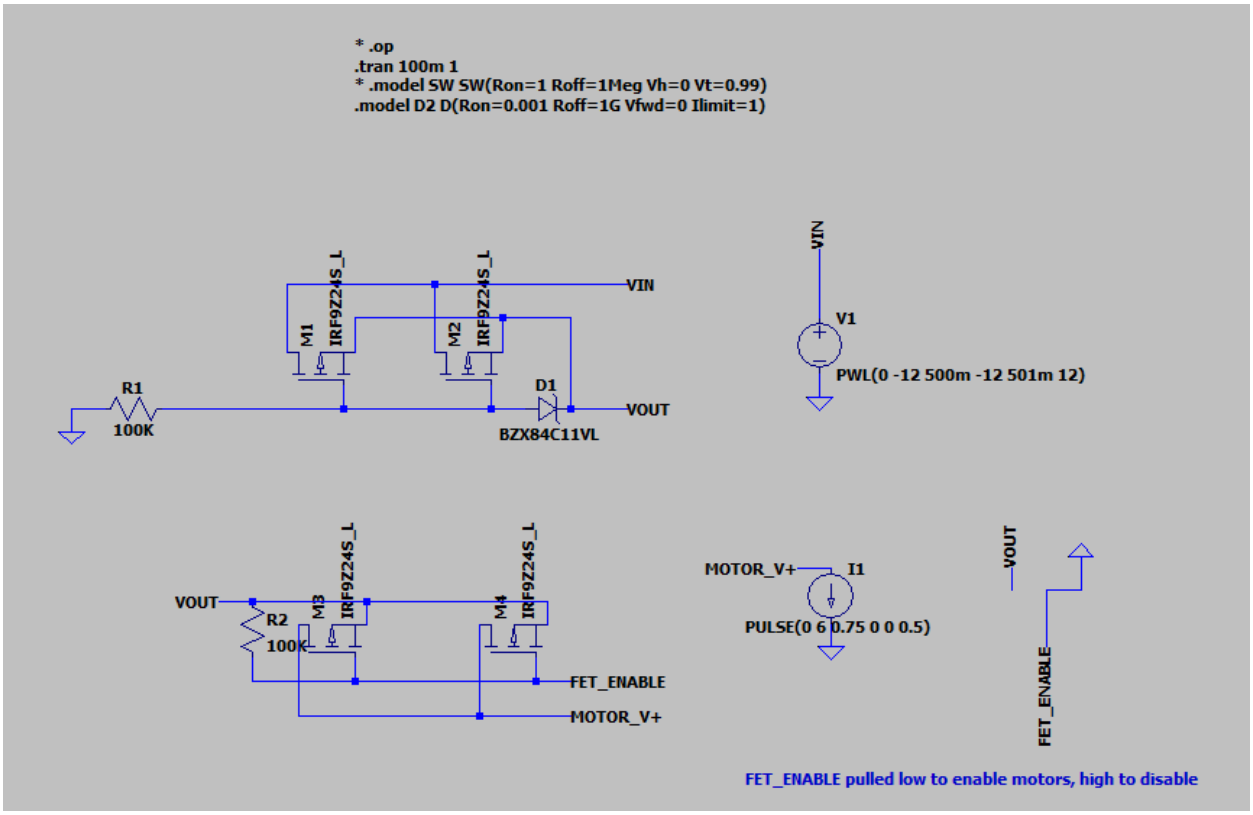
---

4 Appendix

4.1 Simulation: Polarity Protection in LTSpice

The polarity protection intended for the USB port in section 2.3 was additionally validated with LTspice, on a similar circuit.

I set up my simulation as follows:

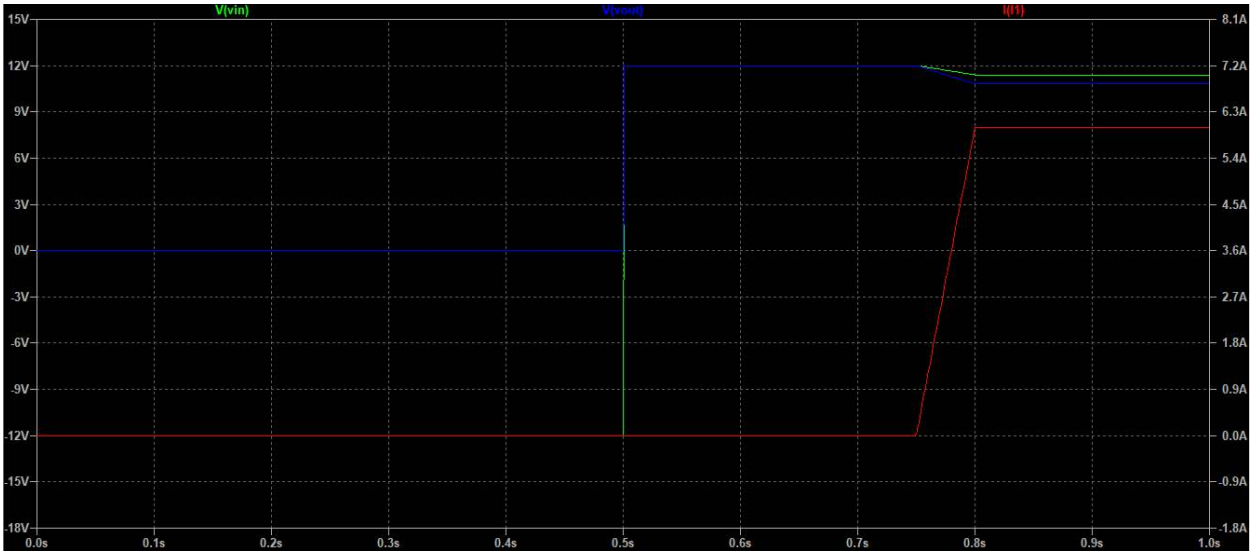


Above, Fig 19: LTSpice simulation setup

Here, I have a pulsed input voltage that starts at -12V and changes to +12V after a certain period of time. Then, a simulated motor turns on (a linear current sink) that makes the voltage drop over the MOSFETs visible.

The expected behavior is for the MOSFET to block all of the current while the voltage is reversed, and pass all of it (besides a small forward voltage drop, visible under load) when the voltage is correct.

Below are the results:



Above, Fig 20: LTSpice simulation results. Green: input voltage, blue: output voltage, red: current

The picture is a little bit hard to see, but the green line is the input voltage, which starts at -12VDC. The blue line is the output voltage after the protection circuit, and the red line is the current drawn by the "motor" current sink.

As evidenced by the results, when the green line is below zero (from 0 to 0.5 sec), the blue line representing output is zero volts. When the voltage reverses and becomes positive, the blue line matches it, only showing a slight difference caused by the voltage drop when the current becomes large. I'm happy with these results and I believe that they provide some justification for the circuit I ended up choosing, as I was a bit worried I was wiring up my MOSFET incorrectly.

4.2 Part Files and PDFs

If you are curious, I have fully open-sourced this design, and all files are available from my Github at https://github.com/aaroexxt/TeensyESC.

In particular, the full schematic is available to view at:
https://github.com/aaroexxt/TeensyESC/blob/main/TeensyESCV1.pdf

If you are interested in more of my previous projects, you can check out my main portfolio site at:

https://ambecker.com/

4.3 Conclusion

I want to give a huge thank you to Fischer and Adi for teaching such a great class, and being kind enough to be a little lenient with the deadline for this paper in light of other work and travel. I hope this paper has been enjoyable or informative as well! I fully intend to finish this design and this is just the beginning of testing and validating it. I'm really excited to see it work!

Also, deserved credits that I used while making this project:

- Teensy 4.1 Kicad Project
- PJRC Forum
- TMC6200 Datasheet (I stared at this for longer then you want to know)
- Robert Feranec on Youtube for his excellent Altium tutorial

And thank you, for making it this far :)