Theo's MIT Motorsports Portfolio

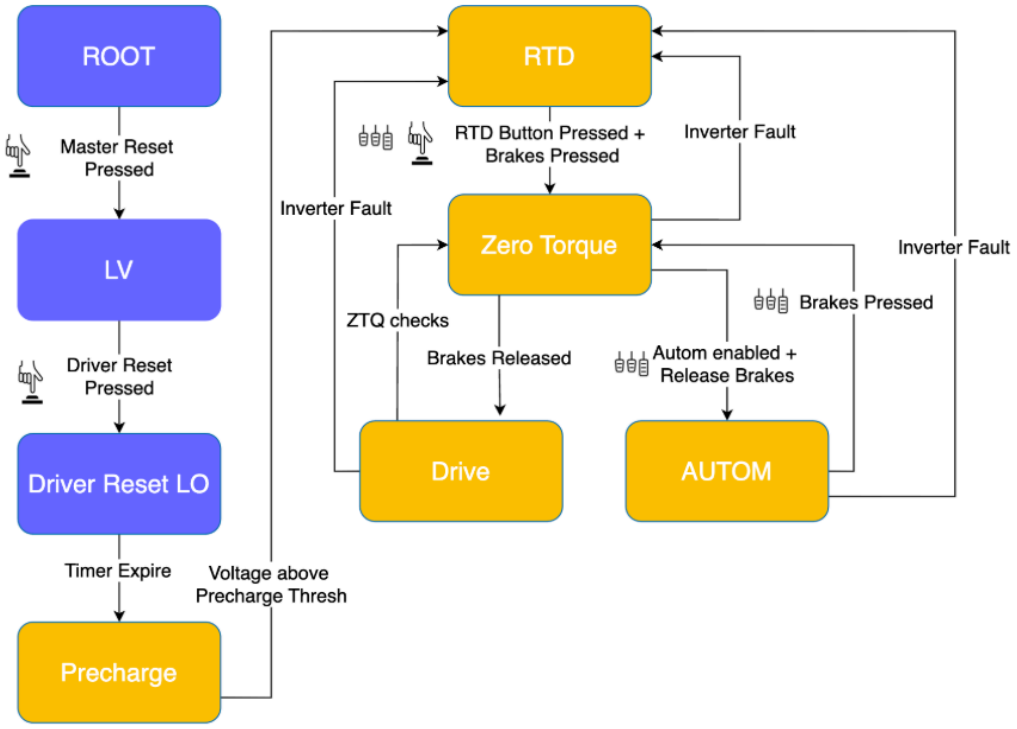
This is a portfolio of the firmware work I've done for MIT Motorsports. My other projects can be found on my GitHub (in particular, check out Radish, my custom programming language).

We use STM32 chips for all of our PCBs, using a combination of CAN, SPI, and I2C for communication. We use C++ with HAL libraries and CubeMX to program the chips, typically using packages such as FreeRTOS for threading and FATFS for external memory.

2025 Season

Vehicle Control Unit (VCU)

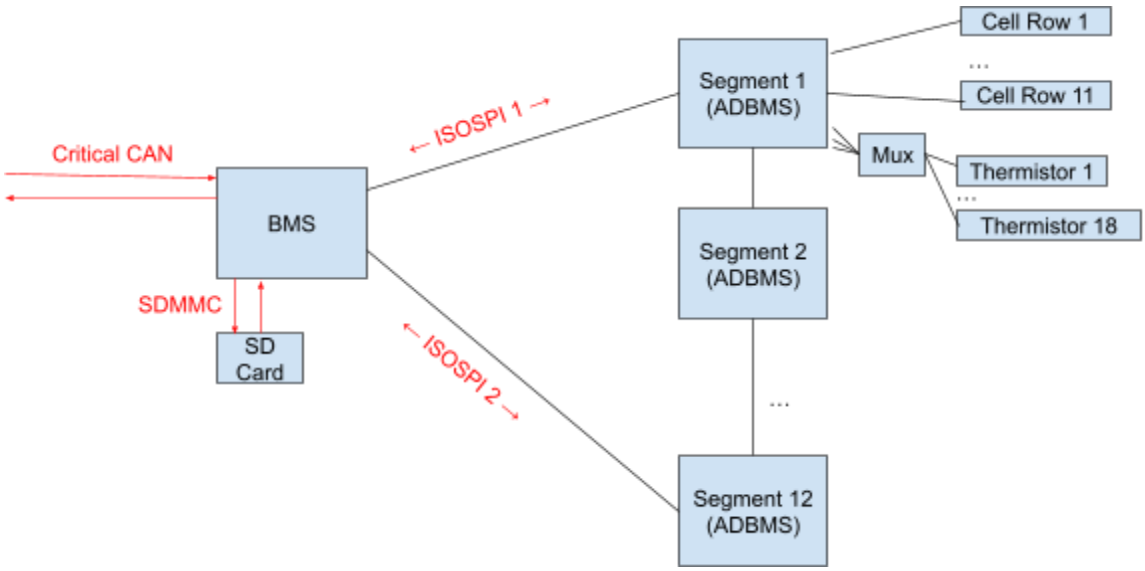
The VCU is the central PCB on the car, responsible for communicating with all the other boards and maintaining a state machine (shown below) to ensure that the car always behaves safely. It's also responsible for determining the torque output to the inverters, and a few other miscellaneous tasks. Although we already had code for the VCU from several years ago, it was getting to the point where it was outdated and hard for anyone to understand, so I was tasked with rewriting it. After doing so, I wrote a test suite to ensure that things would go as I expected and then began to test my code on a Nucleo development board. Once the VCU shipped, I tested my code on the real board and adjusted accordingly, and now we have a competition-ready VCU!



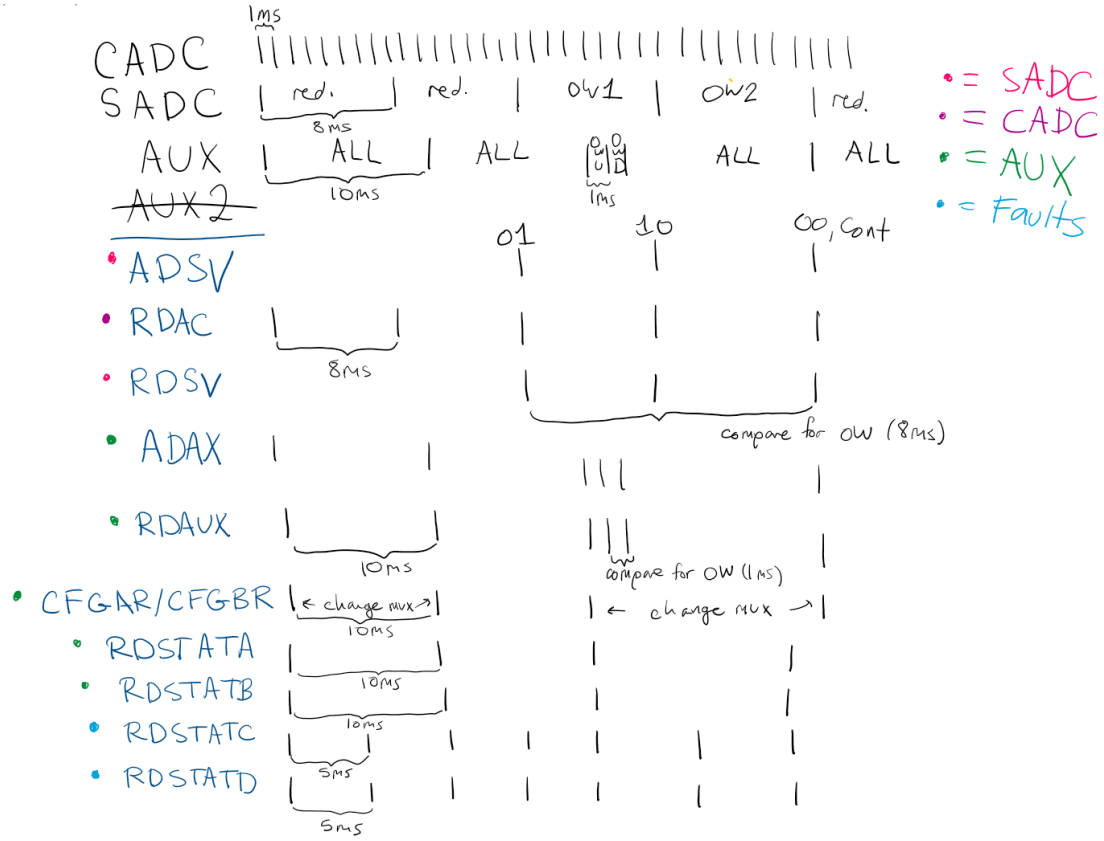
A simplified diagram of the VCU state machine.

Battery Management System (BMS)

The BMS is responsible for monitoring our high-voltage battery. To do so, it's attached to a circular chain of proprietary ADBMS chips (12 of them), each of which is connected to up to 16 battery cells and 18 thermistors (the ADBMS only has 10 GPIO ports for auxiliary input, so we use one for a 2-way mux which allows us to read 18 inputs from the 9 remaining ports). We communicate with the ADBMS over 2 ISOSPI buses which go in opposite directions around the chain, so that if one link breaks we can still communicate with all chips. The BMS controls the ADBMS chips by periodically triggering ADC reads to measure the voltage across each cell and the temperature of each thermistor, as well as periodic register reads to access the results of these measurements and status information. It stores measurement results and status information to determine the state of about 1000 different faults and sends general information over CAN for the VCU to read. The BMS also has an SD card, to which measurements and fault information are written periodically. I wrote the BMS code over winter break (once I was finished testing the VCU) and then tested it in January, and – after many difficulties communicating with the ADBMS – now have a robust, working version that can successfully monitor our battery!



A sketch of BMS communication with peripherals.



A list of the different commands the BMS usually sends to the ADBMS stack, at specified rates.

Firmware Team Management

In addition to working on my own projects, I'm responsible for overseeing the rest of the firmware team and making sure that projects get done on time. To this end, I often test our other PCBs as well - in particular, I've spent a fair amount of time testing:

- Our telemetry board, which is responsible for relaying CAN messages over WiFi (using the LWIP stack for STM32) so that we can view data off the car in real time, and storing said data onto an SD card and an EMMC chip
- Our low voltage BMS, which is responsible for communicating with ADCs and thermometers over I2C to measure the state of the low voltage battery (this code runs on the VCU chip)

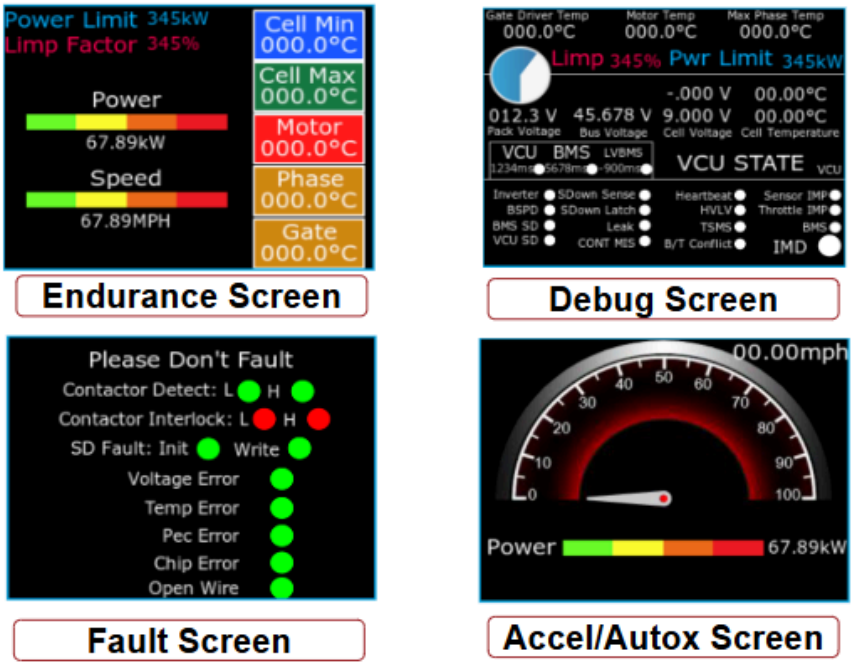
2024 Season

Custom Inverters

My first major project was writing code for custom motor inverters, which we plan to use in the 2026 season (we transitioned to 4WD in the 2025 season, which was enough of an architecture change that the team decided it best to stick with ready-made inverters). The custom inverters will receive torque commands from the VCU, control the motor correspondingly, and send feedback values back to the VCU. The custom inverters must operate at a high frequency to accurately control the motor, and communicate with gate drivers and resolvers for feedback. As these inverters don't exist yet, I don't have much more to say about the project, but I'm excited to test it next year!

Dashboard

I spent my second semester writing code for the dashboard, which controls an LCD screen and a set of buttons for the driver to use. I used TouchGFX (a low-level graphics library) to design and program six screens for different drive scenarios, some of which are shown below. The dashboard receives data such as battery voltage and fault information from our main CAN bus and displays it on the screen, and sends information about which buttons are pressed back to the VCU. After programming the dash and testing it as much as I could before putting it on the car, I went on testing trips to see how it behaves in a realistic drive scenario and get direct feedback from drivers. It worked pretty well!



Some screens from the MY24 dashboard.