VMS CAN Bus Sensor Array

Alice Ma
Braeden Hamson
Jade Nguyen
Pushpesh Sharma
ECE 413 Senior Project Dev
Faculty Advisor: Roy Kravitz
Acknowledgement: Nate Ruble
Spring 2023

Table of Contents

Team Member Roles and Contributions

Team Member	Role(s)	Contributions
Alice Ma	Team Lead Software, Coding CAN	● Wrote the Data Translator code ● Planned project timeline ● Helped with Accumulator and Pedal control code ● Integrated CAN into existing chassis. ● Performed debugging and troubleshooting
Braeden Hamson	Team Manager Software, and Hardware	● Designed Data Translator Board, PDAC Board, and Pedal Controller Board ● Performed Assembly for all PCBs ● Assisted in coding CAN ● Integrated CAN into existing chassis ● Performed debugging and troubleshooting
Jade Nguyen	Software, Coding CAN	● Wrote the Accumulator Main Board code ● Integrated CAN into existing chassis ● Performed debugging and troubleshooting
Pushpesh Sharma	Hardware, PCB Design	● Designed Battery Segment Boards and Accumulator Main Board ● Performed Assembly for all PCBs ● Assisted in integrating CAN into existing chassis. ● Assisted in debugging and troubleshooting

Executive Summary

Viking Motorsports is a student-run engineering club at Portland State University who have been designing and building Formula SAE race cars for several years. This year they are pioneering their second EV (Electric Vehicle) build. The design and build process is divided into groups such as power train, chassis, suspension, driver aids, etc. Our capstone is based on a previously missing, but eventfully necessary, CAN Bus system.

The previous version of the EV was not capable of determining, and reporting, faults in the electrical systems of the vehicle. Thus, during the event of system failure, the person troubleshooting had to take the entire vehicle apart to determine the source of the error and fix it. A monitoring system was proposed to collect and send sensor data via CAN to a display screen for easy monitoring by the driver. The sensor array inputs come directly from the batteries, the charge systems, the pedal controllers, and steering modules. The data is logged via microcontrollers and converted into displayable forms.

As this is a purpose-built CAN bus system, our capstone deliverable is a complete wiring harness with PCBs and connectors: a plug and play system to the EV. Thus we would start with schematics of each of the boards placed in different locations on the car. Those schematics would then be converted to PCB layouts and manufactured boards. The schematics of each component would be accountable for following FSAE rules for component placement and any necessary galvanic isolation. The size and placement of the board would have to combine with the rest of the car thus several iterations of our designs were necessary before finalizing the PCBs.

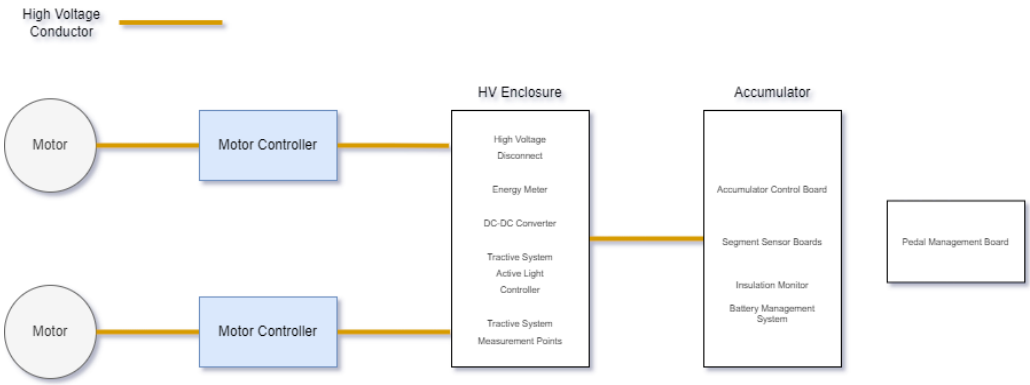
On the software side, this was also a complete design from the beginning. All the nodes on the CAN bus would be connected in parallel, meaning that each node is connected to all other nodes on the network. As there were three microcontrollers in the full harness, they both had to be able to communicate with each other via CAN to send data back and forth. To communicate with the CAN bus, we would use the MCP2551 CAN transceiver IC. The IC acts as an intermediate transmitter pair to connect the STM32 to the CAN bus. The 120Ω resister across the CAN_H and CAN_L would be required at each terminal node. The CAN controller would be initialized for 500 kbit/s (standard Baud rate). CubeIDE provided with the STM32 microcontrollers would be used to set up CAN communication. After configuring the clock, CAN and all of the needed GIOP pins should be enabled so we could generate the code to send and receive messages.

The overall approach for this project was simple. We split the project up by hardware and software. One team was working on the code and the other team was constructing the PCBs. Both teams would align together to ensure that all pin assignments were consistent and the final integration process was done together.

Background

The Viking Motorsports Team at PSU has been working on creating a fast, efficient, and innovative EV to participate in the FSAE competition. Our project involved monitoring several aspects of the EV via CAN and displaying the sensor outputs to the screen in the vehicle. Monitoring these aspects via CAN will allow for efficient diagnostics of the EV systems and make way for troubleshooting in the event of any failures.

The current electrical system is represented in the following diagram.



For simplicity, all low voltage connections have been omitted, as well as some minor systems. The previous versions of the car did not have any monitoring being done for any of the systems of the car. The individual components contained their own controllers that housed information on their status and faults. However, none of this information is accessible without connecting a laptop to the various systems directly.

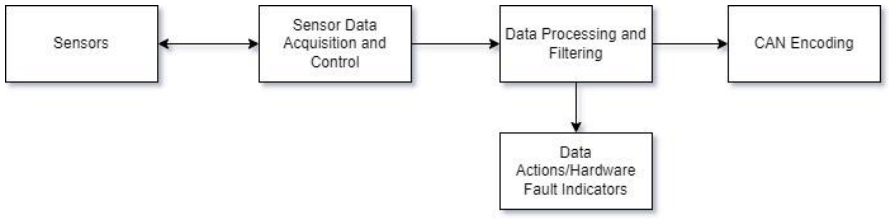
There are several different systems that need monitoring inside the EV. The HV (High Voltage) system requires monitoring of the temperature sensors for the compartments that hold the batteries. The temperature of the batteries affects the power output as well as the overall performance of the car. Other components of the HV systems include the accumulator isolation relays, fuse, current sensors, precharge and discharge system. The EV also requires monitoring of the LV (Low Voltage) systems as well as the status of the pedal control system.

The industry standards have several requirements for CAN protocols being utilized in road vehicles. Some of them include the following from ISO 11898:

- CAN specifies the high-speed physical media attachment (HS-PMA), including HS-PMAs without and with low-power mode capability and those with selective wake-up functionality.
- CAN specifies characteristics of setting up an interchange of digital information between electronic control units of road vehicles equipped with CAN.
- CAN specifies the characteristics of setting up an interchange of digital information between modules implementing the protocol's data link layer.

Being able to monitor, process, and display these systems statuses will be extremely beneficial for troubleshooting and overlay monitoring of the EV. The Formula SAE competition has very specific requirements regarding galvanic isolation of the HV and LV systems in the EV. These requirements need to be followed when developing our CAN system.

Data Flow Diagram



It is important to note that the VMS team has not utilized the CAN bus system for monitoring any sensor outputs in the past. However, even though the team has not implemented this system yet, there are several other off the shelf devices that utilize CAN to help monitor and also provide ability to alter the sensors and their calibration in order to get the best outcome from the overall system. Haltech offers a wide range of CAN Bus devices along with the sensors that are heavily used in the automotive industry, especially in DIY projects where factory engine management systems fail to deliver proper results. Research in companies like Haltech as well as other products like Vector used by Daimler can be beneficial for our project as well.

Requirements

As mentioned earlier, we want to adhere to the requirements of the EV overall when designing and constructing our CAN Bus systems. We want to follow any necessary restraints set by the FSAE competition guidelines so as to avoid any delays and provide smooth integration of our systems with the rest of the vehicle. Thus most of our requirements for this project are based on the number of sensors and components that we want to read and monitor for the management of the EV. Other requirements are based on what components we utilize in our project in order to stay on budget and provide smooth integration when the time comes.

- Must be powered from 12 VDC and/or 5 VDC and tolerate a +/- 2V fluctuation
- When a board interacts with HV (350 VDC) all HV components must be galvanically isolated from LV systems
- Must send data to a central data logging device connected to the CAN Bus
- Must use appropriate CAN Bus arbitration
- The team must cooperate with other members of the VMS team to ensure system compatibility
- Should follow accepted methods to protect devices on a CAN Bus
- Should use STM32 microcontrollers
- Should transmit data at a rate appropriate for the specific measurement

- Should monitor the following systems, in order of priority:
    1. AIR+, AIR-, and precharge signals
    2. APPS, BSPD, IMD, BMS OK, SDC status
    3. Tractive system voltage, tractive system current
    4. LV system voltages
    5. Orion BMS 2 data
        a. Battery state of charge
        b. Tractive system power output
        c. Battery voltage
        d. Battery current
    6. Cascadia Motion PM100DX data
        a. Individual motor power
        b. Inverter mode (running, fault, precharging etc.)
        c. Relay outputs (used to detect wiring faults)
- May monitor status of shutdown switches

Objectives and Deliverables

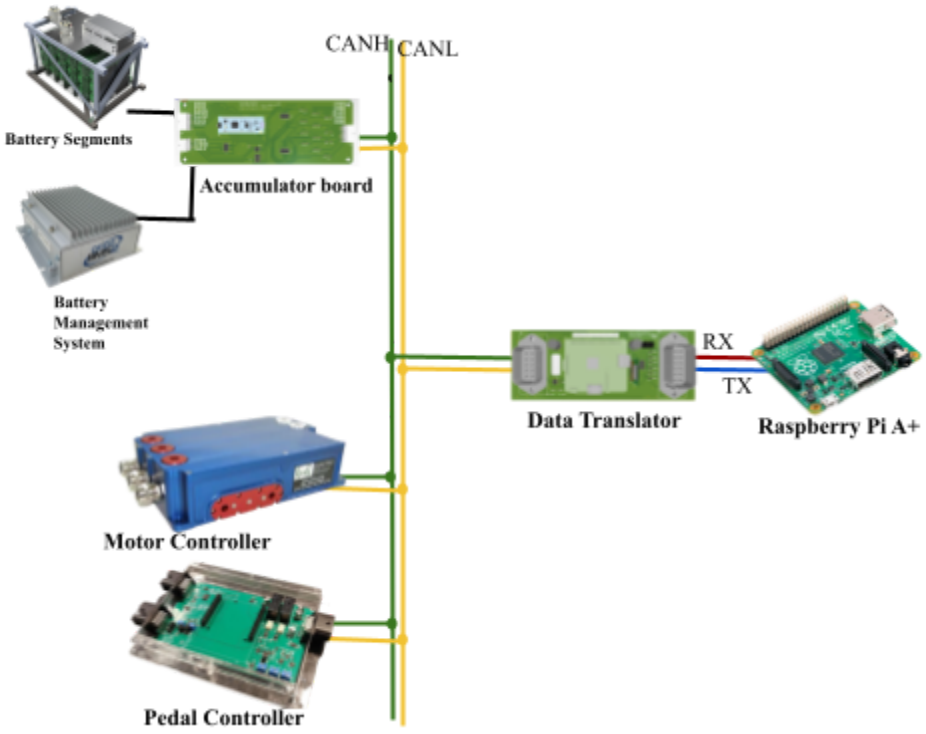
This section specs out what we were originally required to deliver to the industry sponsor by the end of the capstone. This initially included:

- Project proposal
- Weekly Progress Reports
- Final Report
- ECE Capstone Poster Session poster
- Travel with the VMS team to Michigan to attend the FSAE competition. It is held on June 14-17 (travel and other expenses are covered)
- Detailed design documentation, explaining what the design does, how the design works, and why the team made the decisions it did
- Simulations, including both the simulation inputs and outputs
- Electrical CAD: Schematics and board layouts, including output files (Gerbers, PDFs, etc.)
- Bill of materials and pricing
- Version control, including checked in previous revisions of your design (e.g., a Git repo).
- Individual PCBs with 3D printed housings for each of the boards.
- Functional code for STM32CubeIDE for the individual CAN hubs.
- A full harness that connected all PCBs and with all functional code compiled onto the controllers, ready to be plugged into the car.

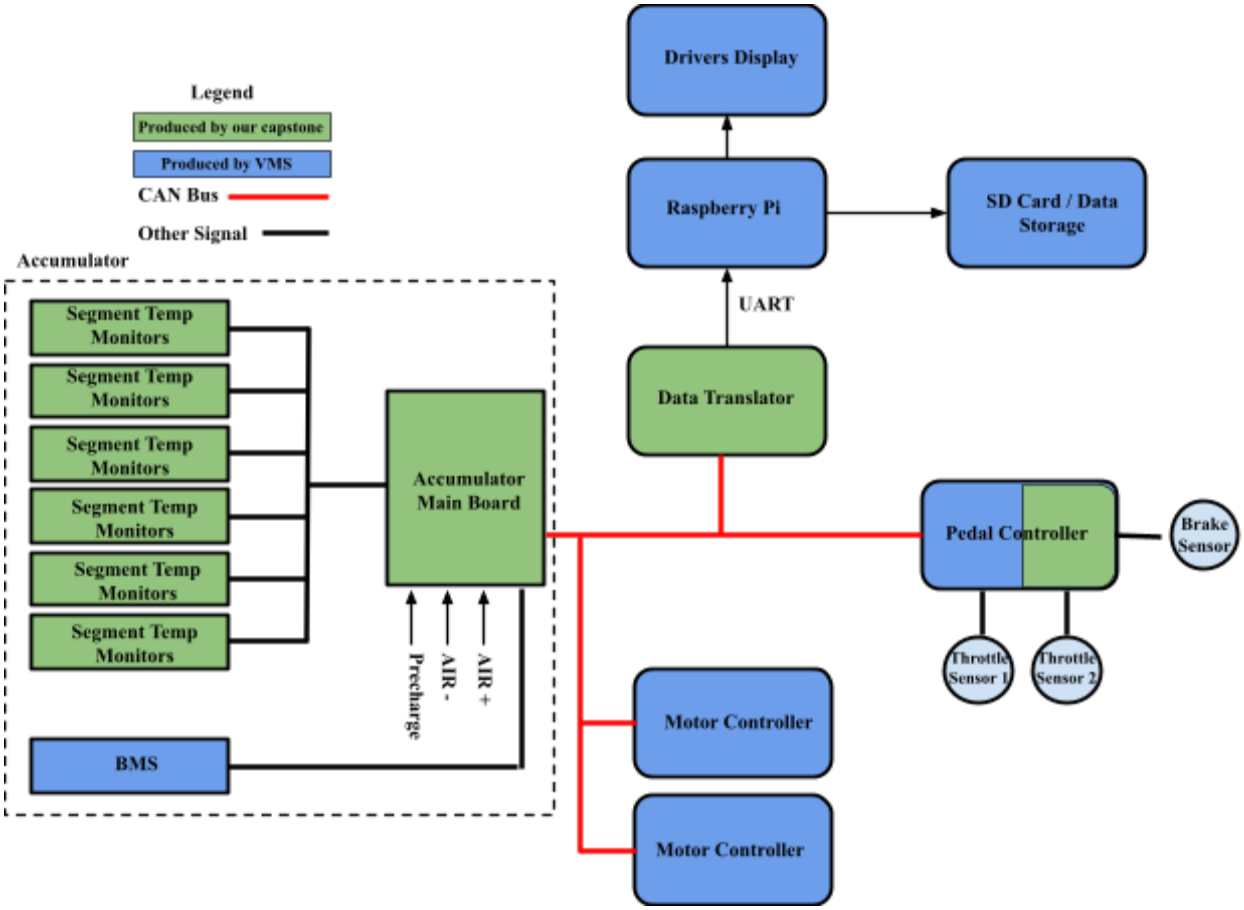
As the timeline of the project progressed, some of the requirements changed. Specifically, the rest of the EV was not finished in time thus the EV did not attend the FSAE competition in Michigan. Our deliverable for the full harness remained the same.

Design

We started the physical design with an iteration of what the sponsor had previously been working on. The diagram below shows the expected flow of information throughout the EV. This we designed the system based on nodes from where the information would start and nodes where it would end.



As mentioned before, the EV contains several other components that need to communicate together and our job was to send data from a lot of these components via CAN to the display. As seen in the diagram, the battery segments, battery management system, motor controller, and pedal controller are the main components that send data to the data translator via CANH (CAN high) and CANL (CAN low) signals. Also important to note that the battery segments also send voltage information to the battery management system but not via CAN, however, our design takes that into account when designing schematics and PCB layouts.

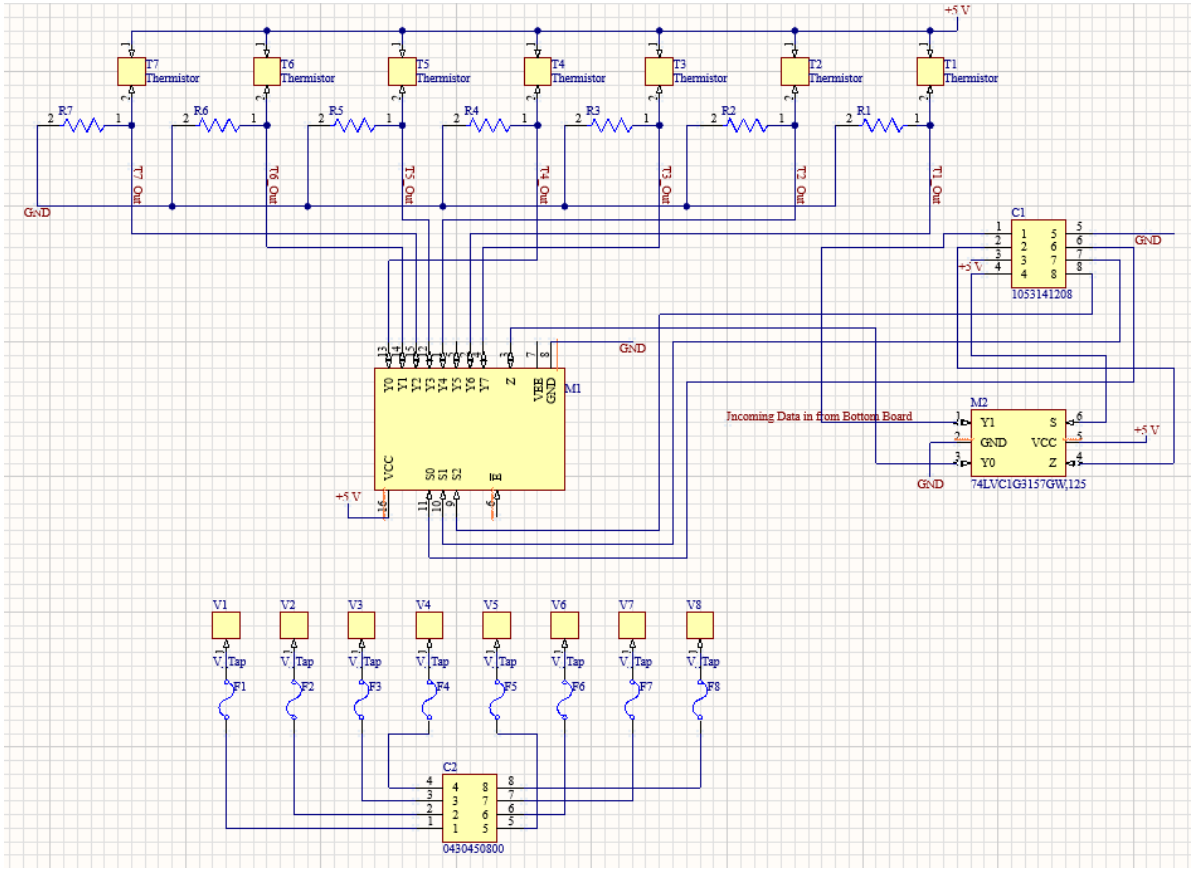


The figure above shows the complete layout of all components and their communications on the EV. The legend states that the green blocks denoting the modules are for our capstone specifically. Per the design specifications of the battery pack (also known as the accumulator), we have temperature data from the battery segments going to the main board. The main board shares that data with the Data Translator which conveys to the Raspberry Pi via UART protocols.

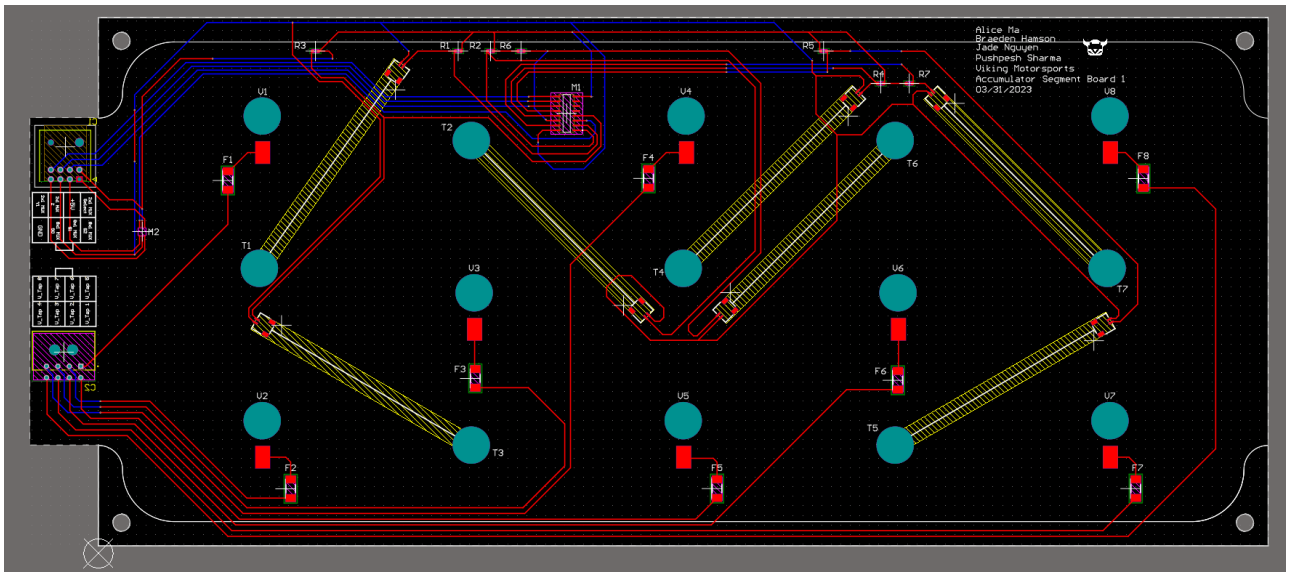
A new layout for the Accumulator was designed to hold six battery segments. The team also wanted to have one more spare segment, thus all components and peripheral parts were ordered with seven segments in mind. Voltage and temperature was measured and monitored using nickel taps and thermistors on the top and bottom of each segment. We started by speccing the thermistors from a vendor. From the sponsor's requirements of the segment, we wanted the thermistors to fit the following requirements:
Voltage Rating: 400 V, R = 10 kΩ, B = ~3950, Tolerance = 1 %, Total Length = 100 mm, Working Temp = -45 ^{0}C ~ 120 ^{0}C. See Project Resources section for the link to the vendor.

Battery Segment Board
A PCB design was created specifically for the top and bottom to mount the thermistors and voltage taps. The top and bottom boards contain a different number of thermistors as the layout of the bottom side is different from the top. However, both designs include voltage dividers to convert the resistance from the thermistors to a readable value for the microcontroller as well as connectors to send the data output to the Accumulator Shelf Board. Using Altium Designer

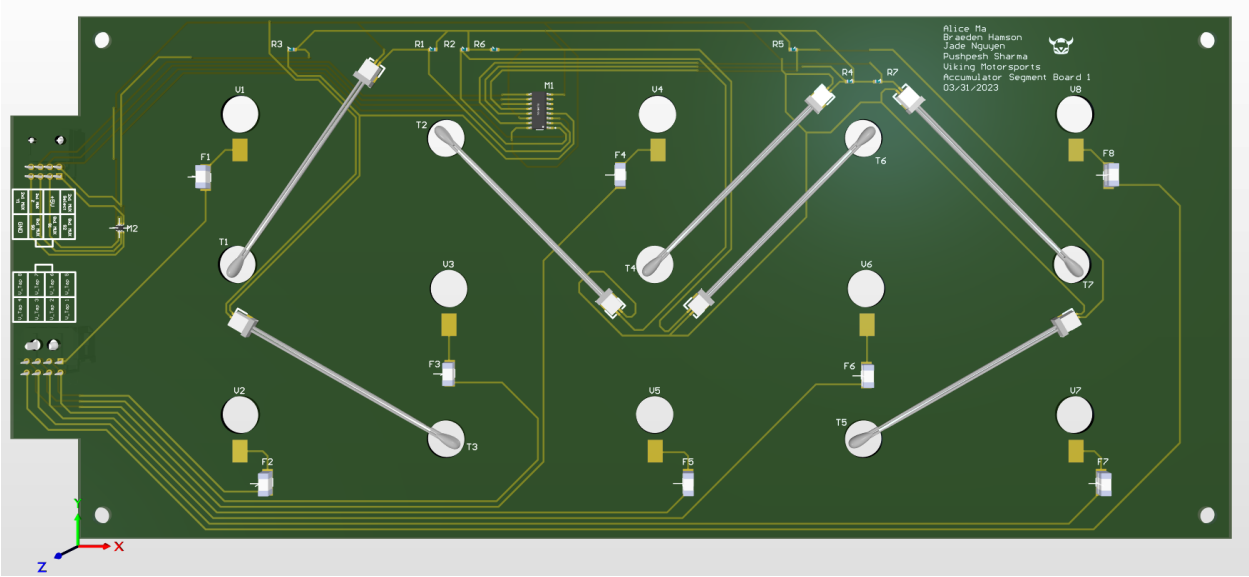
software, schematics of each of the boards were created, then PCB layouts formed, then manufacturing gerber files generated for boards to be printed by a vendor. Each board contained connectors for data pins being input or output to another board.



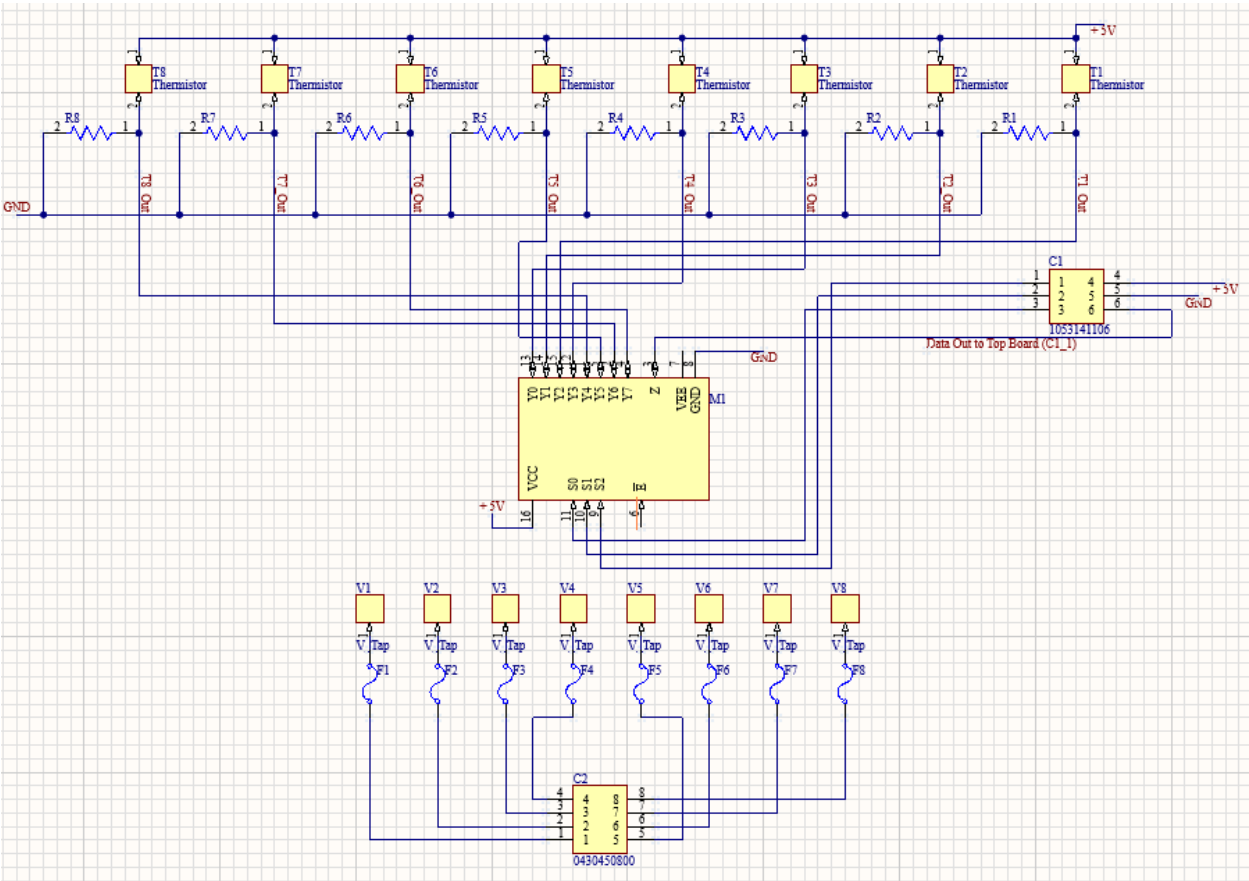
Top Segment Schematic



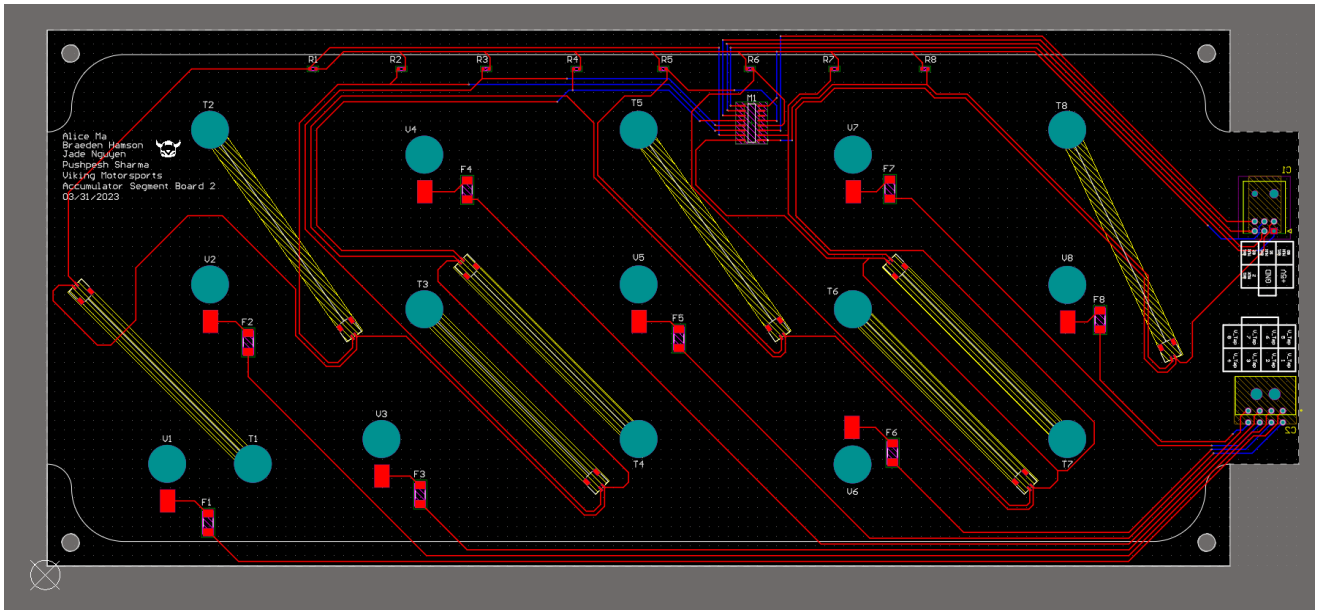
Top Segment PCB Layout



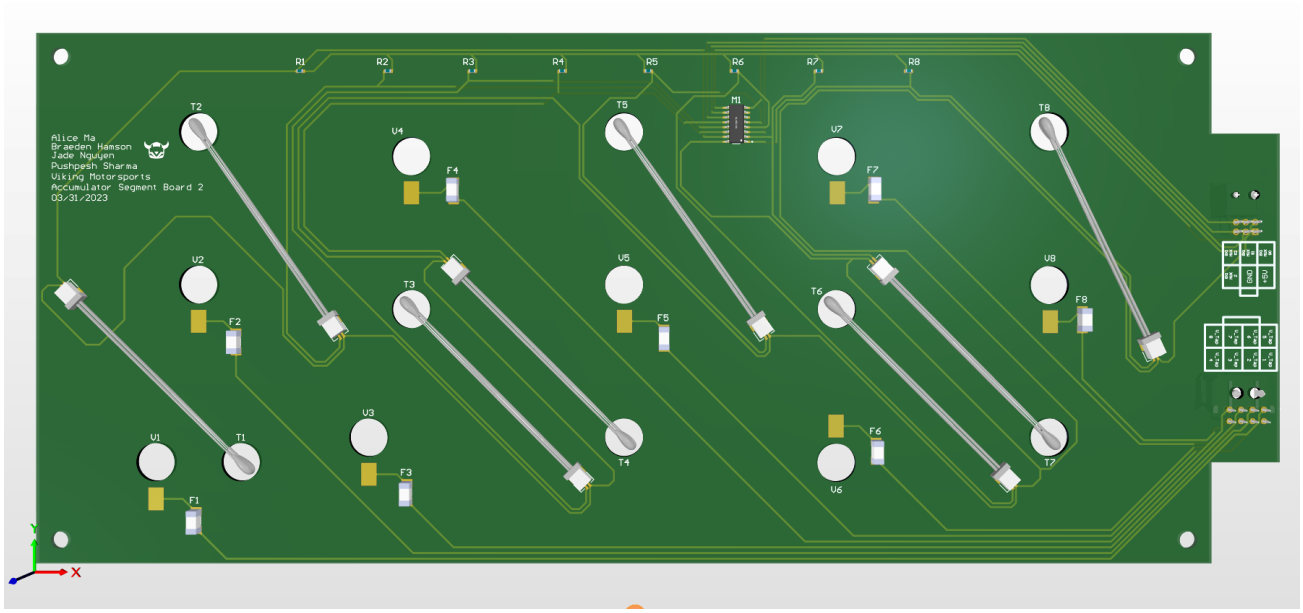
Top Segment PCB 3D View



Bottom Segment Schematic



Bottom Segment PCB Layout



Bottom Segment PCB 3D View

On the schematics for both the top and bottom segments, the thermistor inputs go into an 8 x 1 multiplexers that have select bits which are sent from the STM controller on the Accumulator Main Board. The Z inputs for the top and bottom board on a single battery segment are sent to a 2 x 1 multiplexer which has the select bit also sent from the STM. Each battery segment has 15 thermistors total, and each segment only sends a voltage of 1 thermistor each time but it will cycle through all 15 thermistors on the battery segment. Thus the multiplexers help reduce the number of wires and allow the STM to cycle through the thermistors at the designated sampling rate and collect the temperature data.

Both the top and bottom segment PCBs take into account the need to maintain galvanic isolation between HV and LV components. Both designs maintain at least a 4 mm gap between the LV and HV components as stated in the FSAE rule book. There are also no components on the bottom of the PCB to reduce the noise risk from the HV components to the data sent by the thermistors. The STM from the Accumulation board will read the voltage of the thermistor and convert it to temperature.

Because the microcontroller can only read analog signals below 3.3V, we have to use voltage dividers to convert them to resistance values. Then, we use that resistance to convert it back to the temperature (electrical resistance changes with temperature)

$$R = \frac{(5-V_{read})*3300}{V_{read}} \text{ (Ohm)}$$

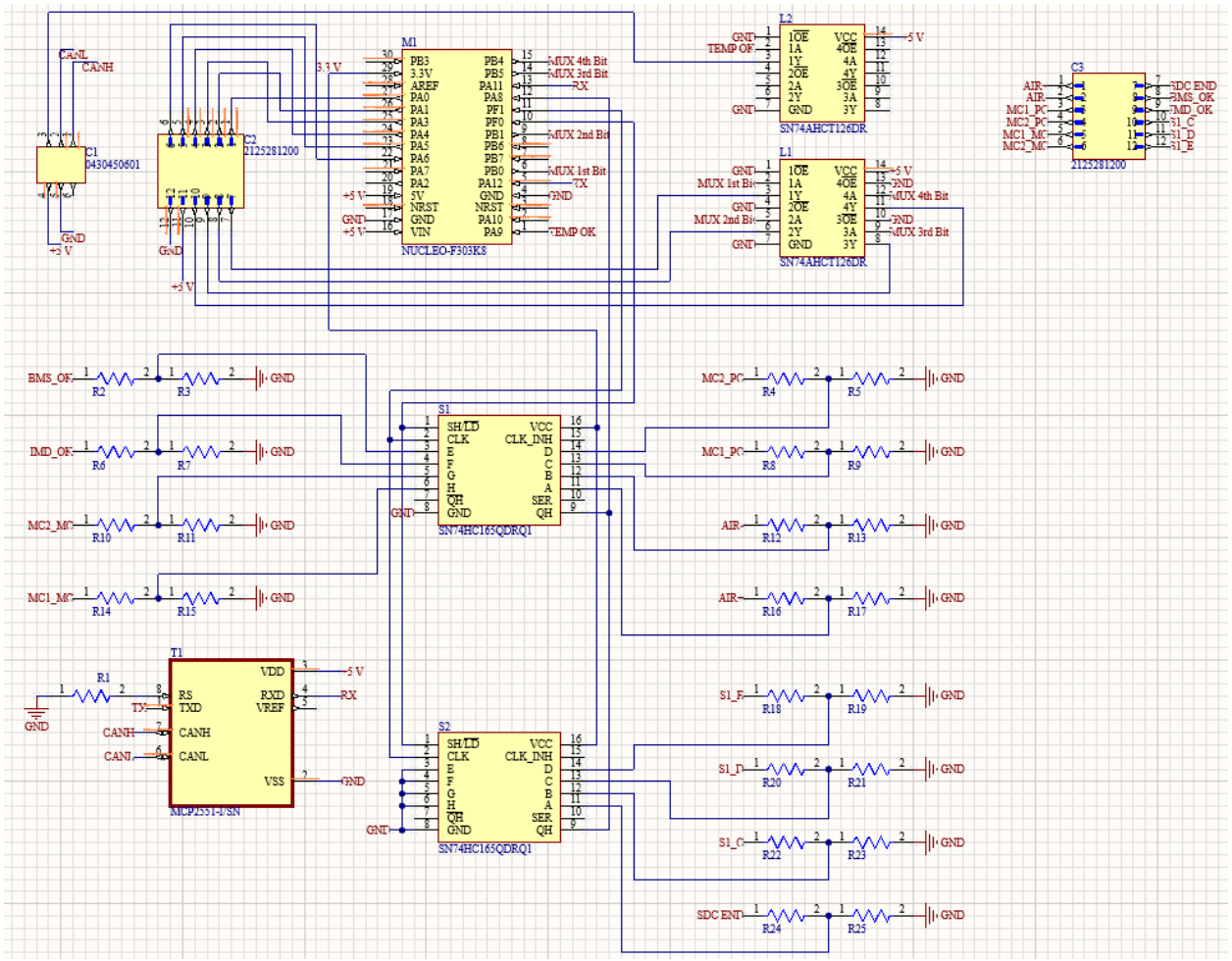
$$Temp = \{[\frac{1}{T_0} + \frac{1}{B} ln(\frac{R}{R_0})]^{-1} - 273.15\} \quad [^0C]$$

With T_0 = 298.15 Kelvin
B = 3250 Kelvin
R = the thermistors resistance at Ohm
R_0 = 10k Ω

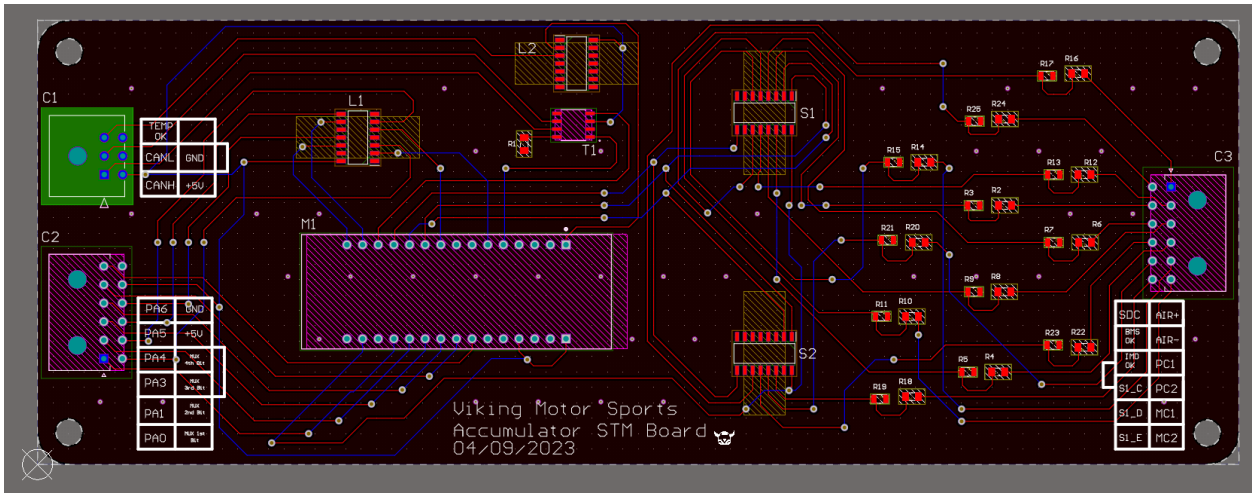
To be more accurate, instead of taking 1 temperature sample from each thermistor on the segment boards, we take 10 samples, calculate the average and eventually send them to the CAN bus. All of the logic and calculation will be coding in the microcontroller on the Accumulator main board.

Accumulator Main Board

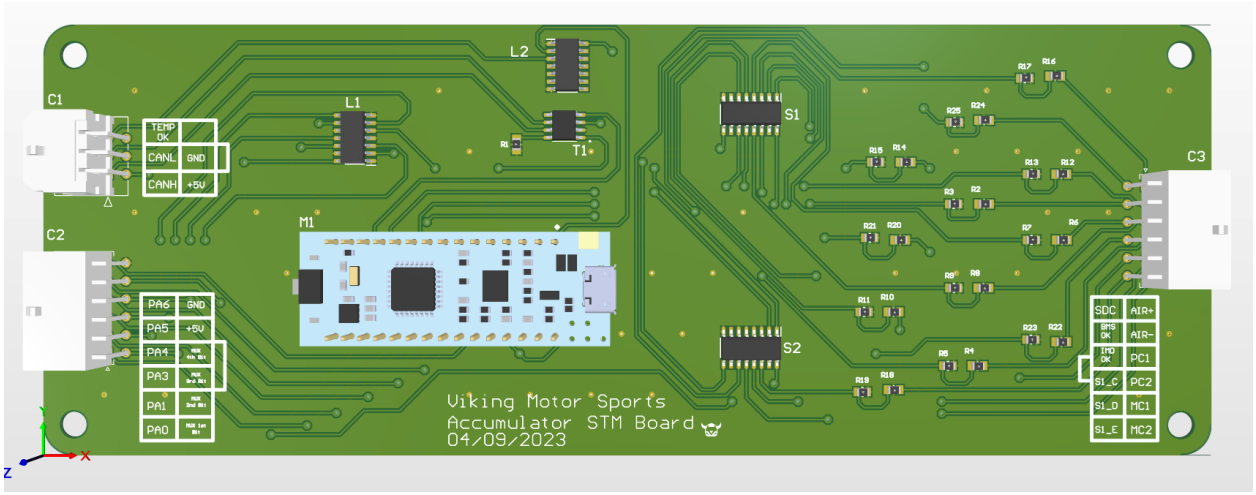
The Accumulator Main Board housed the first STM32 used in our CAN system. The purpose of this board is to collect and modify the incoming information from the segments and battery management systems, and send it to CAN bus. In the microcontroller, 6 ADC (analog to digital converter) pins are enabled to receive data from all 6 battery segments at the same time. The STM also sends out select bits for the 8x1 MUXs and 2x1 MUXs on the segment boards to choose which the thermistor on the segment will send out the value. All of these select bits will need to go through a buffer gate that converts the 3V signal to 5V in order to be read by the multiplexers. The code allows the STM to cycle through each of the thermistors on the segment boards. Including the top and bottom side of the battery segment there are a total of 15 thermistors on the battery segment. Another buffer gate (level shifter) is used to send out a TEMP_OK signal to the battery management system. Besides that, the other inputs from the battery management systems are all 12V signals which are converted through a voltage divider in order to be read by the STM. The STM reads these signals through shift registers that allow it to cycle through parallel to serial inputs. The microcontroller then creates RX and TX signals which are connected to a CAN transceiver on the shelf board. The CAN transceiver creates CAN_H and CAN_L signals that are sent to the next STM which is housed on the Data Translator Board. The complexity of this board is present but still allows us to keep the PCB to a single layer design and since all the components are on the top of the board, it can be mounted easily inside the accumulator.



Accumulator Main Board Schematic



Accumulator Main Board PCB Layout



Accumulator Main Board PCB 3D View

For the coding, we have to enable the CAN protocol in the STM. We initialize it with operating mode is NORMAL mode and 500kBrau rate. We set up 6 ADC analog inputs to receive the voltage values from each battery segment, and store it to ADM (analog direct message) without going through the CPU which saves lots of time as we need to pull out those voltages, convert it to temperature values and send it to the CAN bus. We also configure 4 digital output pins for select bits and write the logic to cycle through all 15 thermistors on each segment, making sure that we constantly update the temperature of the battery to the Data Translator.
Below is the format of the messages that are sent to the CAN bus with the first 6 bytes are temperatures of 6 segments and the last byte is the chosen thermistor.

```
 Seg1 = 30C      Seg2 = 13C      Seg3 =3C
 Seg4 = 12C      Seg5 = 11C      Seg6 =13C
30 13 3 12 11 13 1
30 13 3 12 11 13 2
30 13 3 12 11 13 3
30 13 3 12 11 13 4
30 13 3 12 11 13 5
30 13 3 12 11 13 6
30 13 3 12 11 13 7
30 13 3 12 11 13 8
30 13 3 12 11 13 9
30 13 3 12 11 12 10
30 13 3 12 11 13 11
30 13 3 12 11 13 12
30 13 3 12 11 13 13
30 13 3 12 11 12 14
30 13 3 12 11 12 15
```

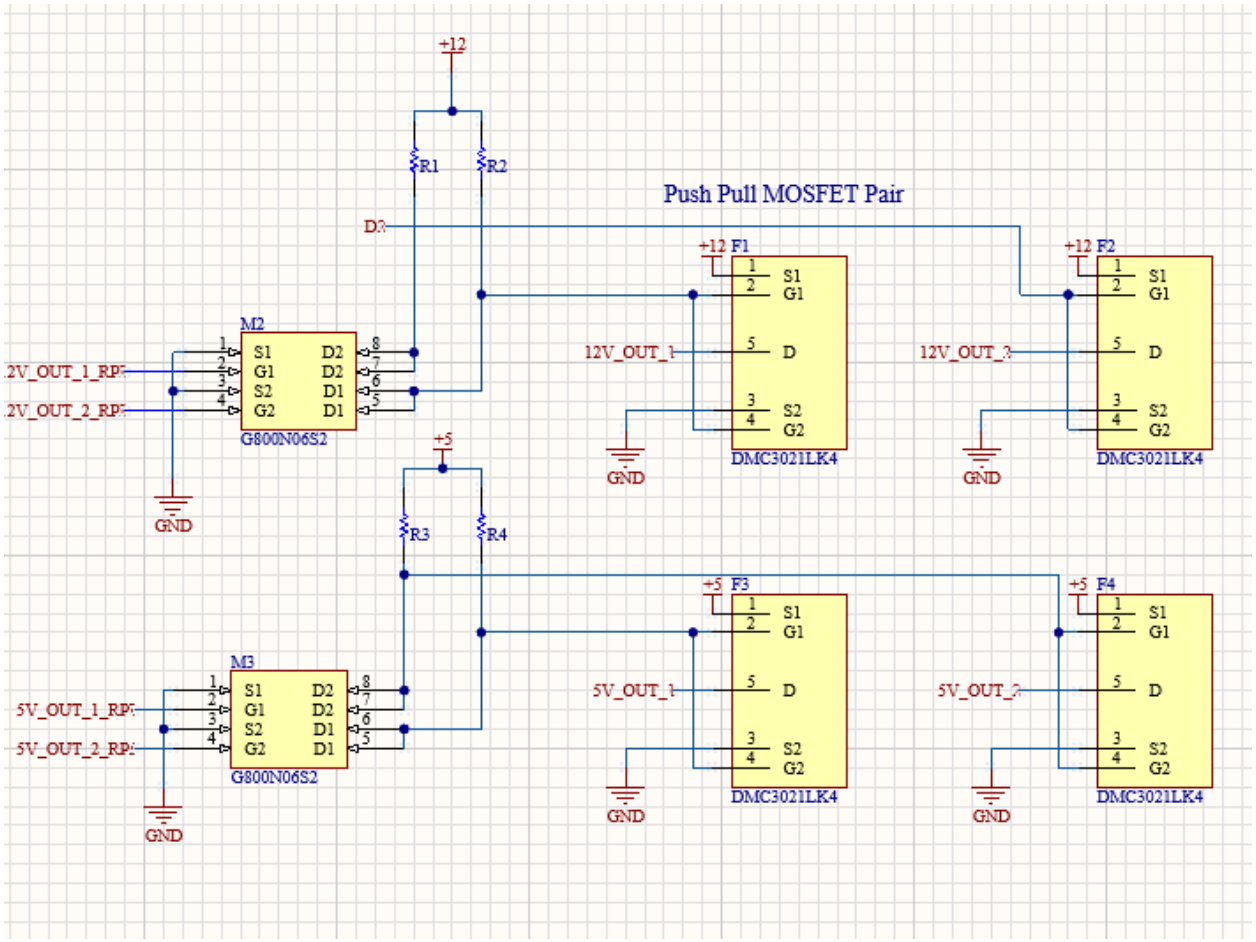
Serial Output from Accumulator Main Board Code

During that time, we write the code to check if the temperature of any cell in the battery package is over 60^0C, the TEMP_OK message will be set to 1 and sent to the BMS immediately to open the contactors and stop delivering power to the car.

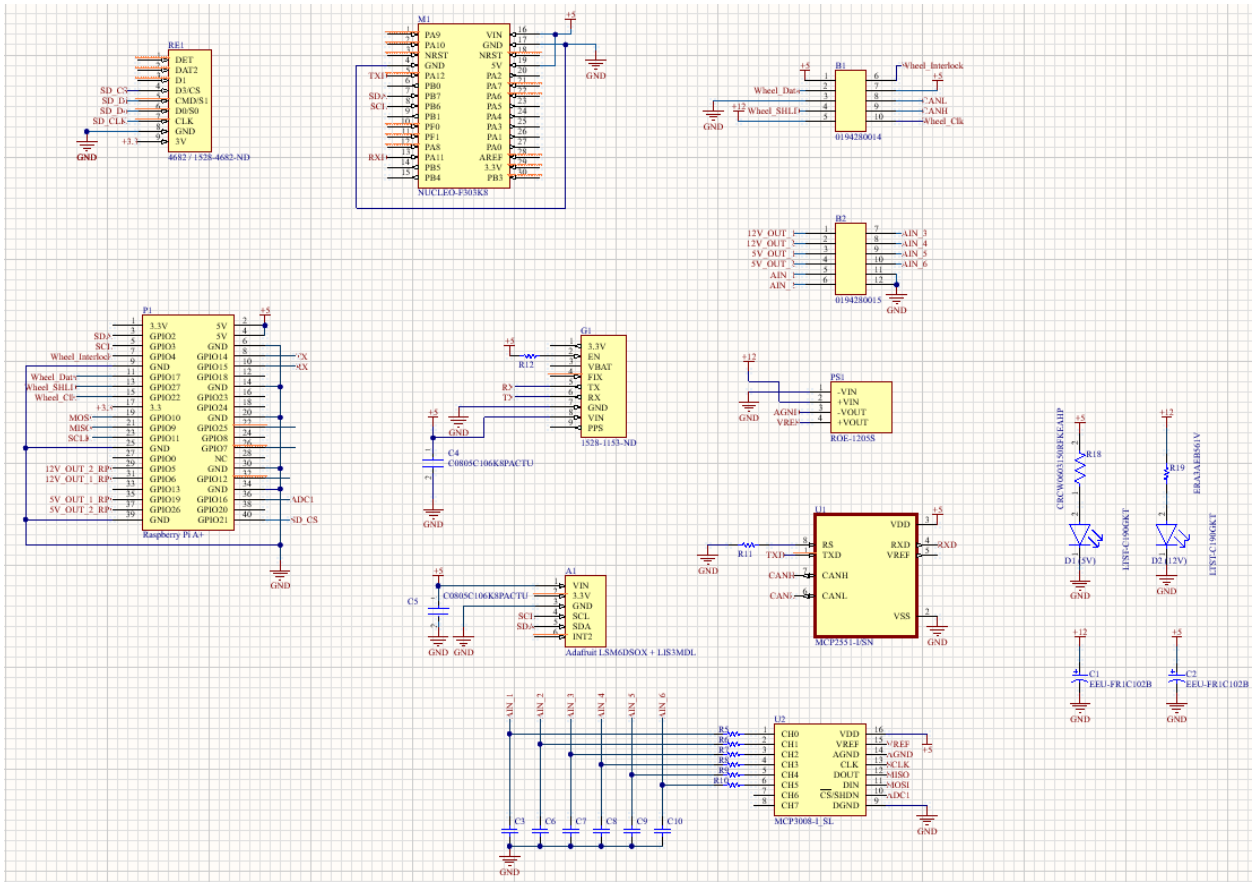
The STM32 also needs to receive 12 messages from the BMS (battery management system) and send it to the Data translator so the driver could check the status of the EV. However, the STM32 does not have enough GPIO pins, so we then just use the shift register and configure 3 pins on the STM to control the CLOCK and receive all of the messages.

Data Translator Board

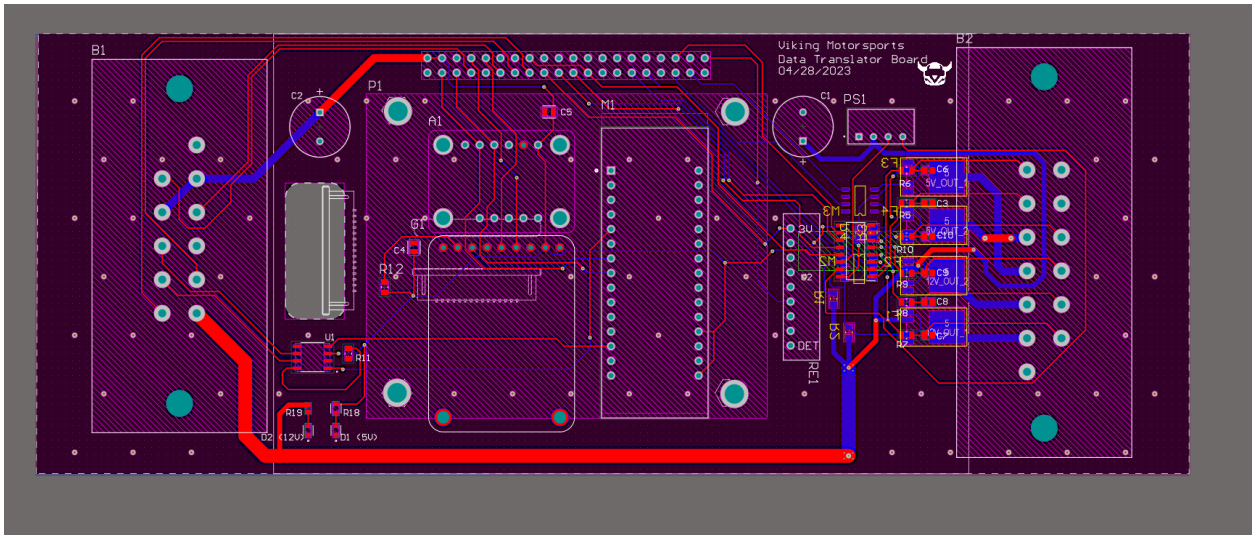
The Data Translator board is the heart of our capstone. It collects information from the Accumulator shelf board and microcontroller uses UART protocols to send the data to be displayed on the screen via RaspberryPi. The translator also collects information from the suspension, pedal controller, and GPS. Although these components are worked on by other teams, our Data Translator PCB takes those inputs into account and the code works together in sharing information and translating to the Raspberry Pi.



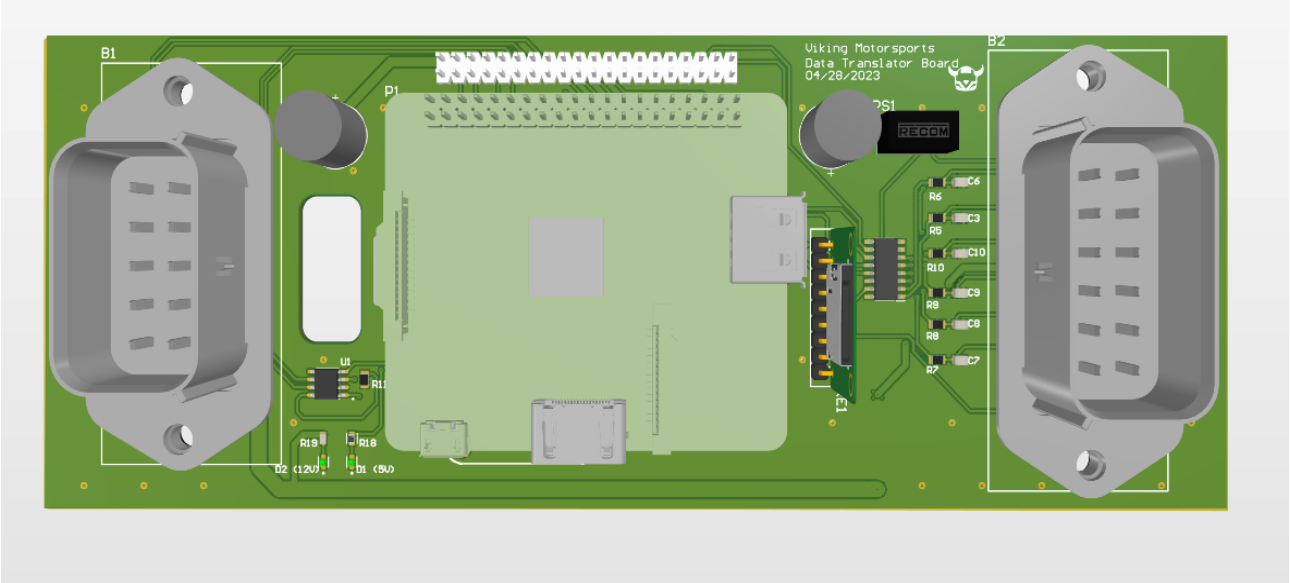
Data Translator Board Schematic Part 1



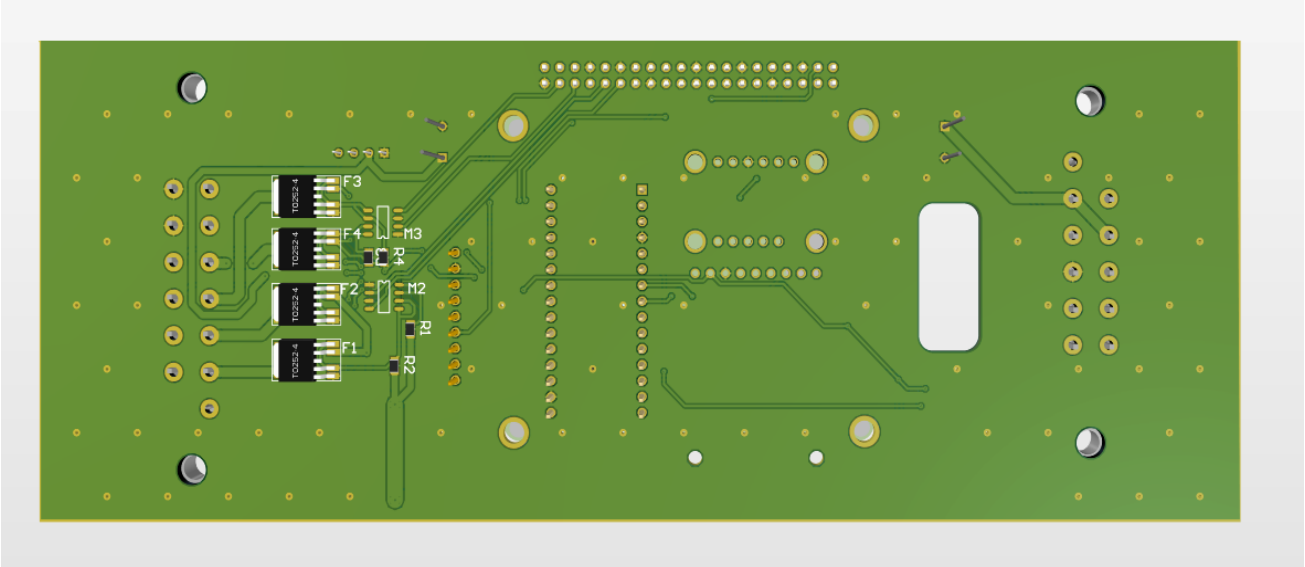
Data Translator Board Schematic Part 2



Data Translator Board PCB Layout (See Altium Files in Github for full view of top and bottom side)



Data Translator Board PCB 3D View (Top Side)



Data Translator Board PCB 3D View (Bottom Side)

Since the Data Translator is in charge of receiving a large amount of messages from the CAN bus, we used the CAN filters in the STM32 to filter only the messages we need. Instead of directly receiving the motor controller voltage, current, and RPM message from the motor controller, we are receiving these three messages from the pedal controller instead all in one message. We did this because the motor controller messages were coming in at such a high speed that the STM32 couldn't send out the UART transmitting. The pedal controller slowed down the transmission from the motor controller to about 100 milliseconds for the data controller to receive. The Data Translator receives the CAN data in a receiving callback loop that is triggered in from an interrupt when the STM32 receives messages that are passed through the filters. We decode all the messages coming from the motor controllers using the formulas:

Tx Header: *0x0A5* (Motor position message)

RBM = $(byte\ 3\ *\ 256)\ +\ byte\ 2$

Tx Header: *0x0A6* (Current message)
Current = $\frac{(byte7\ *\ 256)\ +\ byte\ 6}{10}$

Tx Header: *0x0A7* (voltage message)
Voltage = $\frac{(byte1\ *\ 256)\ +\ byte\ 0}{10}$

The rest of the messages are received without needing a decode and we directly copy them in a 125 byte unsigned integer array to be sent to the Raspberry Pi. The array sent has a data layout that we assigned and the Raspberry Pi knows exactly what data is in which element. The biggest part of the array is the thermistor temperatures since there are 6 segments and 15 thermistors each. The 125 byte unsigned integer UART array is sent in the while loop of the code twice every second.

Electrically this board is very straight forward. Premade STM and Raspberry Pi boards were socketed into the main PCB to simplify construction. As well as facilitate programming. By not using discrete chips soldered directly to the board we avoided the issues of developing and debugging methods of soldering chips to the board and programming them in that configuration.

The other notable feature of this board is the 12 pin connector which is connected to the R Pi's GPIO pins through MOSFET pairs. As well as analog inputs for measuring signals up to 12 V through an ADC. This allows for future expansion and gives the ability to control and measure various things should the need arise.

Pedal Controller Board
The pedal controller board was designed and constructed by the VMS team, however, we collaborated with the other CS capstone team on the code. We receive the CAN messages from the Motor Controller, and decode it to understandable data since they would use it as the information for their Torque-Vectoring calculation.

The microcontroller that is used is STM32F446. It has the same hardware set up with a CAN transceiver that creates CAN_H and CAN_L signals. We use it to receive data from the Motor Controller and send out information to the Data Translator Board.
In the code, we use filters to eliminate all of the unnecessary information and only receive messages from different ID headers that we want from the Motor controller. We decode them, save them in particular local variables for later use.
Decode formulas:

Tx Header: *0x0A5* (Motor position message)
RBM = $(byte\ 3\ *\ 256)\ +\ byte\ 2$

Tx Header: *0x0A6* (Current message)
Current = $\frac{(byte7 * 256) + byte\ 6}{10}$

Tx Header: *0x0A7* (voltage message)
Voltage = $\frac{(byte1 * 256) + byte\ 0}{10}$
Power = current * voltage

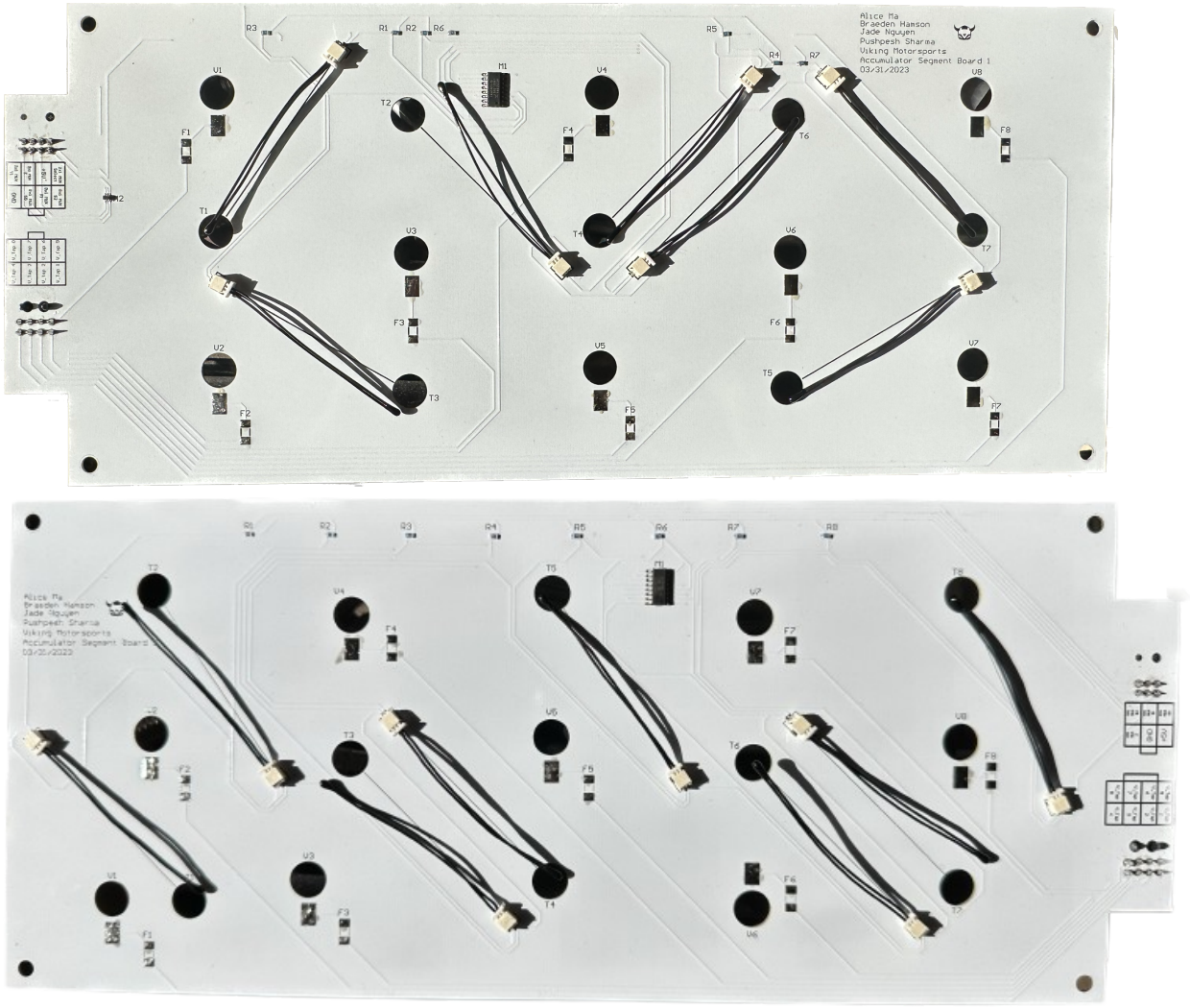
Beside that, we also need to check the APPS_OK_MCU signal and send it to the Data Translator via CAN bus. To check that signal, we combine our code with the Torque Capstone group and change a few features to ensure it works properly. In the last minutes, as Data Translator could not receive too many messages from the Motor controller and send it to UART with a high speed, we finally decided to to let the Pedal controller reduce the pressure on Data translator by make it select and compress all the useful data into one message then send it to Data translator with a slower speed.

In order to integrate the CAN into the rest of the EV, we would have needed the updated versions for other components on the EV such as the new accumulator, the chassis and suspension layout as well any other changes being made that would get in the way for our wiring harness. Our designs for the boards were made in collaboration with the rest of the VMS team as they designed the accumulator and the chassis layouts. However, the new chassis and accumulator were not constructed in time for us to test our CAN system. Thus the sponsor, and our capstone team made the pivoting decision to use the existing chassis to test our CAN system in the meantime. The other necessary components were parts of different capstone projects which would also get more freedom and continuation with this decision.

The existing chassis was from a previous iteration of an EV, thus it included the battery management system, the motor controller, the original battery segments and the peripheral wiring to connect these nodes. The previous chassis also contained a CAN Bus twisted pair wiring to connect these nodes together which we could tap into to send and receive our messages as well. Thus, the decision to use this chassis for testing our prototype was a no brainer.

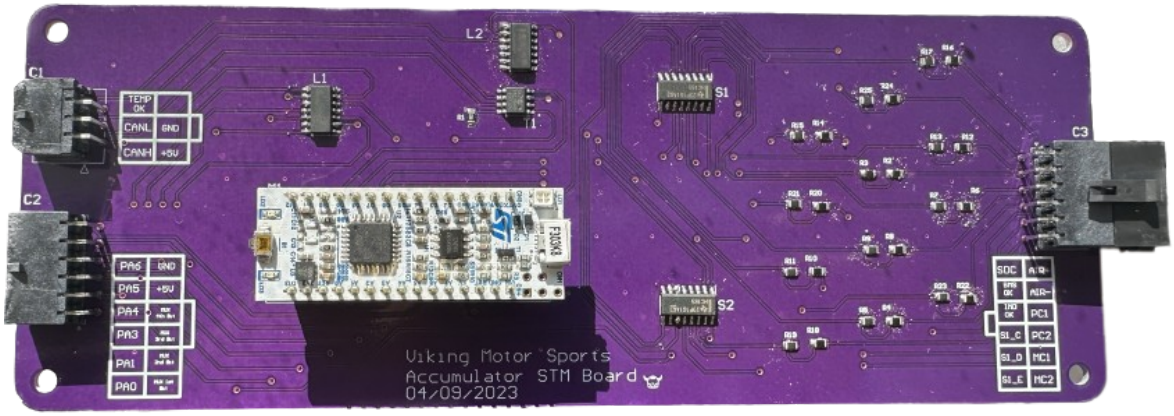
We started by assembling our PCBs. See Appendices for BOMs related to each specific board we assembled. We ordered the components and connectors for each PCB from DigiKey and they had arrived prior to us getting the boards back from manufacturing. Each of the boards we designed contained pinout diagrams for each of the connectors on the board. The layout of the diagrams were in the shape of the connector to tell us which pin on the connector did what. This aided in the construction of the wiring harness later on. We first assembled the battery segment boards using the provided stencils from JLC PCB. The stencils allowed us to use solder paste to easily connect the components onto their designated footprints. The battery segments for our capstone contained the thermistors that were connected to connector mates which were soldered onto the boards and the thermistors themselves were fixed into place via kapton tape to hold their position above the holes in the board. The board contains microfit connectors for the voltage taps (See Evaluation section for further discussion on the voltage taps) and nanofit connectors for the thermistors. The wiring harness is created based on the Altium schematic and the pinout diagram on the boards. For our prototyping and testing, we only constructed one full battery segment with

the top and bottom segment boards. This was sufficient for our testing purposes. Wiring harness created connections between the top and bottom of the segments and the pinouts for the select bits were sent to the Accumulator Main board.



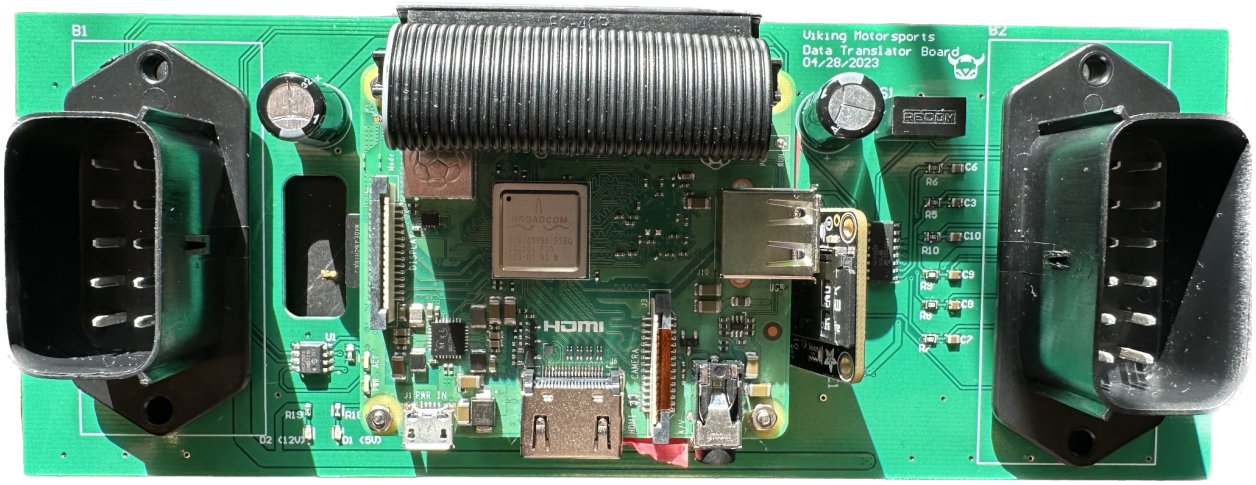
Assembled Battery Segment Boards

The assembly process was similar for the Accumulator Main board, however, this board was not ordered from JLC PCB. Instead due to time constraints, we ordered this board to be manufactured by Oshpark which does not provide stencils for their PCBs. The board thus had to be constructed manually by carefully applying solder paste and placing the components. This required some strategic placement, steadiness of the hand and obvious clean up of excess solder using brake cleaner in spray. The Accumulator Main board receives its wiring from our segment boards onto C1 for this capstone prototype but is ready to receive further data from the BMS and motor controller also.



Assembled Accumulator Main Board

The Data Translator Board was the most intricate one out of all of them for our capstone. It contained the most amount of components in a small construction and the design allotted for mounting the Raspberry Pi on top of the STM which was surface mounted on the PCB. The VMS team kindly provided an 3D enclosure which required the PCB layout to be very specific as the enclosure was mounted behind the dashboard of the car. The Data Translator is the central hub for this CAN system as it connects to the Raspberry Pi via UART and thus connects to a display mounted on the front of the dash via a ribbon connector. The assembly, once completed using the stencil, gets wiring using deutsch style connectors from the pedal controller.



Assembled Data Translator Board

The CAN Bus was the twisted pair wiring that connects all nodes together. The previously existing chassis already contained the CAN Bus that sent information from the BMS and VSM modules to the pedal controller. Thus the code was updated to take battery information from the BMS instead of the segment that we created. We also viewed the outputs of the Accumulator Main board via the serial output of the CubeIDE. See Results section for all our test outputs from this capstone.

Test Plan

When testing our system, we wanted to ensure the following factors:
Whether the data gets where it's needed and is it used correctly?
Can the system recognize when data is missing?
Are internal system faults survivable, and car system faults recognized and reported?
Thus we must conduct these tests for each and every signal that we want to receive and translate.

Test 1: Thermistors
Is the Accumulator Main Board STM able to receive data from the thermistors?
Result:. The STM could read exact data from the thermistor by embedded advanced ADCs

Test 2: Thermistors
Is the Accumulator Main Board STM able to cycle through all the thermistors for each battery segment?
Result: We were able to read through 15 thermistors in one battery segment and constantly sent the information every 2 second

Test 3: Accumulator Main Board
Is the STM1 able to log the thermistor values using the equations and convert the resistance data to temperature values?
Result: The calculated temperatures from the measured voltage was quite accurate as we need to round the results to ones so we can send it to CAN. However, it was good enough for the driver to keep track of the temperature of the battery.

Test 4: Pedal Control Board
Is the STM32F446 able to receive data from the Motor Controllers and send it to the Data translator?
Result: We successfully combined our code with another group and got the right signal that we wanted. We received all the messages from the Motor Controller, decoded it and sent it to the Data Translator with a slower speed.

Test 5: Data Translator
Is the DT board able to receive all signals from the Accumulator Main Board via CAN?
Result: Yes, the DT successfully received all the messages from the Accumulator Main Board correctly with all the 15 thermistor information.

Test 6: Data Translator
Is the DT board able to receive signals from the pedal controller?
Result: Yes, the DT successfully received all the messages from the pedal controller correctly, both the two signals from the pedal controller and the motor controller messages.

Test 7: Data Translator
Is the DT board able to convert these signals into UART to be sent to the Raspberry Pi?
Results: Yes, the DT was able to copy the data received from the different messages into the UART array respectively.

Test 8: Data Translator
Is the DT board able to send the data to the Raspberry Pi?
Results: Yes, the DT was able to send out the 125 byte unsigned integer array through UART to the Raspberry Pi correctly and the Raspberry Pi was about to receive the data and display it correctly on the screen.

Results

Our prototype for this capstone project works. It completes almost all of the initial requirements that were given to us by the industry sponsor. Lets recap on what those requirements were:

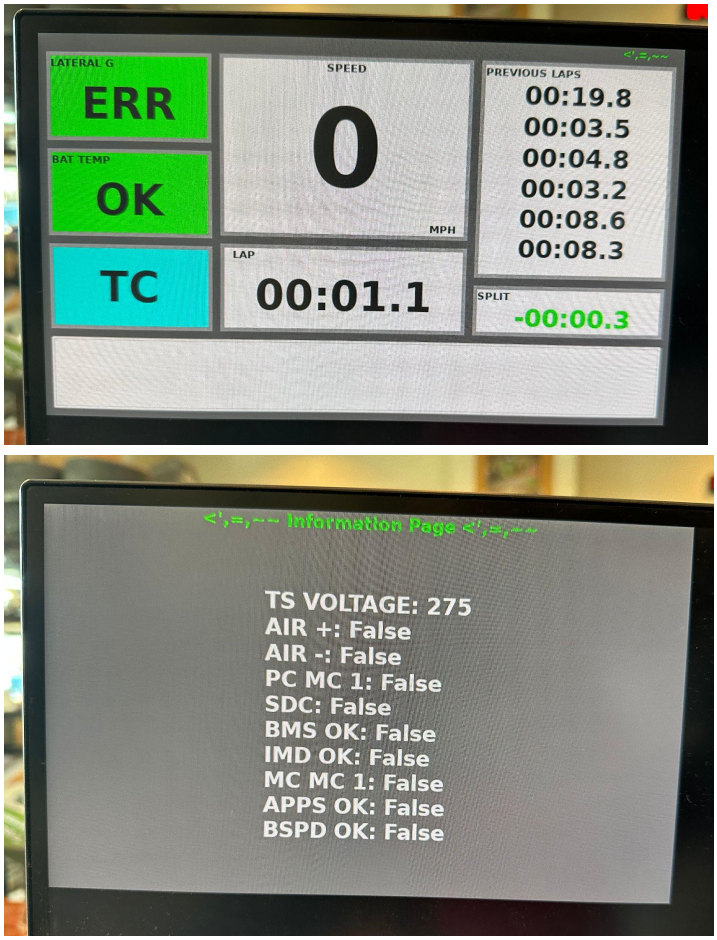
- Must be powered from 12 VDC and/or 5 VDC and tolerate a +/- 2V fluctuation
- When a board interacts with HV (350 VDC) all HV components must be galvanically isolated from LV systems
- Must send data to a central data logging device connected to the CAN Bus
- Must use appropriate CAN Bus arbitration
- The team must cooperate with other members of the VMS team to ensure system compatibility
- Should follow accepted methods to protect devices on a CAN Bus
- Should use STM32 microcontrollers
- Should transmit data at a rate appropriate for the specific measurement
- Should monitor the following systems, in order of priority:
    7. AIR+, AIR-, and precharge signals
    8. APPS, BSPD, IMD, BMS OK, SDC status
    9. Tractive system voltage, tractive system current
    10. LV system voltages
    11. Orion BMS 2 data
        a. Battery state of charge
        b. Tractive system power output
        c. Battery voltage
        d. Battery current
    12. Cascadia Motion PM100DX data
        a. Individual motor power
        b. Inverter mode (running, fault, precharging etc.)
        c. Relay outputs (used to detect wiring faults)
- May monitor status of shutdown switches

At the end of the capstone, we were able to complete all of the requirements except for the ones marked in red. Due to our designs of the battery segment boards, the placement of those components was more than 4mm thus there was no need for galvanic isolation between HV and LV components. Thus that requirement was purely negligible. We were, however, unable to

monitor the status of the LV system voltages. Since we used the previously existing chassis which already had pre existing wiring, we were unable to create a harness to monitor the LV system voltages. This was discussed with the sponsor and listed as something to be completed in the next revision of this prototype.

The capstone was successful. We were able to successfully develop and construct a prototype that followed closely inline with the rest of the capstone projects entailing the other components of the car. We successfully collaborated with an entire EV team and constructed a network system that communicates with 3 of the major controllers inside the EV then filters and translates the data to be displayed on an LCD screen.

The data flow is exactly how we planned it initially in our design proposal. The information starts from the Battery Segments which are designed to collect both voltage and temperature data from the top and bottom side of the each segment inside the EV, then goes to the Accumulator which inclusively collects data from the Battery Management Systems, and Motor Controller. The data is then sent to the Data Translator which acts as a central hub for all information. The Data Translator filters and converts the CAN information to UART protocol and it is sent to the Raspberry Pi. We collaborated and want to acknowledge Nate Ruble for setting up the Raspberry Pi creating the code to display the UART data onto the LCD display.



LCD Screen Outputs from Raspberry Pi

Above are the screen captures from what the LCD display contains. We are successfully able to transmit Battery Temp (the data is shown as a range and displays Max Temp amount when the batteries are running HOT). We also display state of charge as well as the information from the Battery Management system on the information screen.
This is a significant feature to be added onto the previous version of the car. The prototype meets the stated requirements and allows for easy access to diagnostics information which can be seen by the user on the screen or can be logged on a laptop to be tracked later on.

We tested the system by cycling through the LV and HV states of the batteries. Once the HV system is turned on, we can see real time system states for all 10 of the items on the information page. The system shows the value for TS voltage, Accumulator Isolation Relay states, Pre charge signals, Shutdown circuit signals and Main Contacter signals. It also shows the states of the Accelerator and Brake pedal sensors. In the event of system/component failure, the accuracy of these states is important in quickly determining the source of the problem and resolving the issue.

Evaluation

- *What worked?*

The project as mentioned above was successful. Our initial research gave us great insight on how CAN works and how we can use it for our purposes on this capstone project. We understood that the best way to tackle the integration process would be to create separate nodes that can be created with respect to the size and placement constraints from the rest of the components of the EV.

Our initial plan to split up the overall project into Hardware and Software worked great as well. It allowed the team members to utilize some of their previous knowledge and experience and also gave them the opportunity to build on new skills that they can use in the future as well. Those good at hardware got to build the PCBs and learn how to integrate the device. Those good at coding got to set up the CAN communication and learned about assembling PCBs and constructing wiring harnesses. Then the entire team came together for final integration, testing and debugging. The team overall worked great.

Our PCB manufacturer was great. They manufactured the boards perfectly based on the gerber files we gave them and fulfilled the delivery in a reasonable time given that they were coming from China. JLC PCB also provided stencils for the PCBs which significantly aided in the assembly process of the intricate designs.

The STM32 microcontroller worked well for our current purposes. The setup for the CubeIDE for programming the microcontroller worked well. We were able to utilize a lot of resources available as well as the previous knowledge passed down by the sponsor to successfully use the STMs for our uses. The availability of CAN, I2C, and UART protocols was necessary for this project and the microcontroller sufficiently provided that. We discussed what we did not like about STM32s later.

- *What was great?*

The team hands down worked great. It was a great combination of skills and personalities that worked together from day one without any issues. Early on we set up the roles for Team Lead and Team Manager to help guide us through the phases of this project. We went from Forming, Storming, Norming, and Performing together to reach an end prototype that we were all happy with. We even developed good friendships along the way. Building rapport is the only way a team works well together.

The Altium software for PCB designs works great. See Appendices for software details. Although expensive for everyday use, this tool was provided by the sponsor and was excellent even for someone fairly new at PCB designs. The software has every creature comfort imaginable from rule setup, component import, 3D view and auto routing and stitching capabilities, the software was a great resource for us throughout the project.

Our Advisor was great. We met with him every two weeks to discuss our progress, goals, and roadblocks and got feedback in coding formats and things to be mindful of when completing a project on a deadline.

- *What didn't work?*

For the thermistors: Since we chose an overseas vendor, the language barrier and time difference made this over a month long process before the thermistors could be invoiced and shipped to us.

We did not use shift register IC on the Accumulator Board as we finally figured out how to configure the BSM to send CAN messages directly to the Data translator. We could have made a smaller and simpler Main board if we knew how to do it earlier.

Some of the boards did not have complete functionality. When designing a network system from scratch, it is important to give emphasis to the level of involvement we have on the molecular details which can cause the largest mistakes in the long run. During our PCB design process, we selected a lot of different components from DigiKey. Most of them we selected as they had PCB footprints, layouts, and schematics readily available to be imported into Altium. However, we should have spent more time confirming that the .step or other import files for these components actually belonged to the said components and not a variant provided by the same vendor. For the Battery Segment Boards, the footprint for the Fuses used alongside the voltage taps did not match the component we actually wanted. The Fuse size was too big and did not fit on the board. This completely stalled the feature for collecting voltage information from the batteries. In other cases such as the Data Translator Board, the footprints we created ourselves for the large deutsch connectors contained vias that were not fully connected throughout. This caused shorting and loss of CAN communication once the board was constructed and already in the integration phase. These issues had to be resolved with bodged jumper wires which affected the overall look of the PCB and how it fit into some of the enclosures.

- *What would you do differently?*

We would not do a lot of things differently. The structure of our capstone project was excellent. The teamwork and collaboration was sufficient and up to the standards according to our faculty

advisor. The only thing we would change with this project would be the way the budget was handled. As the overall construction of the EV was managed by the sponsor, any and all parts and components ordered for our capstone actually belonged to the VMS team as a whole. Thus the parts were ordered through the VMS team and were delivered to the ME department. Some parts got lost, some did not get ordered, and some BOMs got mixed up due to clerical errors. We would probably like to determine a baseline budget early on and make a collective decision to order parts for our capstone ourselves and request reimbursement from the department later. This would relieve some of the work from the VMS team and allow us to control the deliveries for our parts.

We would have liked to get at least one revision for the prototype we constructed. This would mean developing and getting our PCBs fabricated at least a month sooner than we actually did, assemble and test them to determine the faults and correct them in the revision if the budget allowed. Since we were building the CAN system as the rest of the EV was being designed as well, we overestimated the success of our designs. With at least one revision, we believe that all of the issues would have been resolved and the integration process would have been much smoother.

- *What do you think the next team should do?*

The next team should use our prototype as the baseline to start the revision process. They should start by referring back to the requirements of the project, get confirmation from the sponsor on any updates and begin by retesting the prototype we created. The test results will give the next team a great idea into what has already been done and what needs to be done. They should then refer back to the specifics we would have liked to change in our revision and make those corrections. The next team should only fabricate PCBs for one battery segment, and only order board revisions with the minimum amount possible. This will save on costs and allow for further testing to ensure proper functionality.

Project Resources

- *Links to collaboration sites and repositories with a description of where to get all major parts of the projects (schematics, firmware, software, configurations, etc).*

For all documentation we used Google Docs, Google Slides and Microsoft Excel (specifically for BOMs).
Github Repository for our Capstone: https://github.com/VikingMotorsports/CAN-Bus-Capstone
CubeIDE for STM32: https://www.st.com/en/development-tools/stm32cubeide.html
Altium Designer for PCB designs: https://www.altium.com/altium-designer

- *List of tools we used*
  - Altium Designer 21.9.2 (versions update frequently)
  - STM32CubeIDE 1.12.1 Win

Helpful Links for CubeIDE
How to use CAN Protocol in STM32 » ControllersTech
How to solve debugger connection issues (st.com)
STM32 CAN Interface : 7 Steps - Instructables

Helpful Links for Altium Designer:
https://academy.fedevel.com/courses/online-switching-power-supply-design-course
https://www.youtube.com/watch?v=PqFtSpAXB9Q

- *Other resources*

Notion for major long term documentation and small notes along the way:
https://www.notion.so/VMS-CANbus-Capstone-0b1627731fa44dd9822c95b2836b963b?pvs=4
Discord server for VMS team: https://discord.gg/DF4htDu5x5
Link for Thermistor Vendor: Thermistors

- *Bills of materials (BOMs)*

Accumulator BOM (includes the Top and Bottom Segment Boards as well as the Accumulator Main board items)
Data Translator BOM

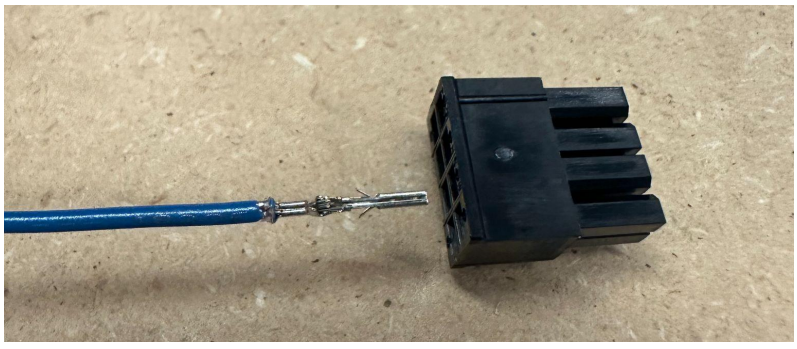
Appendices

- **User Manual**

Assembly: Once you have the boards manufactured, use the stencils to apply solder paste onto the footprints for the components. Then place the components onto the solder paste and use a heat gun to melt the solder paste and form a bond. The solder paste should turn shiny and dry solid.

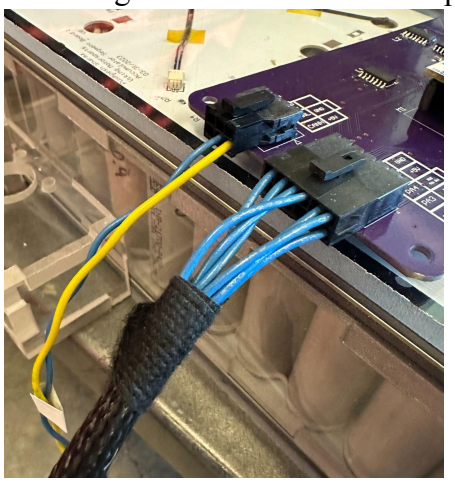
Test each of the boards for continuity between the traces and compare the connections to the initial schematics. Refer the following link for help with any bodging or PCB repair if the continuity tests fail or if there are missing connections on your boards.

Develop a wiring harness. At this time your boards should contain either the microfit or nanofit connectors. Your schematics should have the pinouts for each connector and the board layouts should also have the pinout diagrams to help you decide which pins perform what task.
Use either the Molex crimper (See Tooling section below) or any other crimper to crimp the wires. The crimps are different for microfit vs nanofit connectors so pay close attention.



The crimps should make a good connection with the wire and should collect some insulation for a strong hold as well. The crimp itself should be connected to the connector mate (See BOMs)

and not the actual connector mates (should make a “click” sound if connected properly). Once all the wires have been crimped, connect them to the connector mates and connect the mates to the soldered connectors on the boards. The size of the wire is dependent on availability in the VMS lab. For future projects, the wire spools might be something that is purchased as well. Once connected, perform another continuity test between the wires to ensure a solid connection. Wrap the wiring harness in insulation or plastic shield.

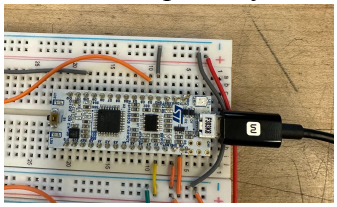


Repeat the process for all the boards and create 2 different nodes: the Accumulator Main board along with the Battery Segment, The Data Translator Board connected to the Raspberry Pi and the LCD display.

The battery segment itself was constructed by the VMS team and the top and bottom boards have specific placement inside them. For our prototype we did not glue the polycarbonate material used for the segments for testing and disassembly. Work with the VMS team to assemble the battery segment completely along with any of its peripheral parts.

Some other wiring and assembly was necessary for the Pedal Controller which was performed by the VMS team. There should be a common connection for the twisted pair CAN Bus throughout the wiring harness of the overall EV. Multiple nodes and controllers are programmed to communicate with each other via the CAN Bus. See Programming information below.

Programming: Once you have the code completed, programming the STM32s is simple. Just plug in the MicroUSB cable to the STM port and click Build on the CubeIDE to upload the code onto the microcontroller. You should have different code for both the Accumulator Main Board STM as well as the STM on the Data Translator Board. These should be programmed separately and tested separately as well.



Operation:
The system is intended to point users toward a fault. To use the system, turn on the vehicle and check the dash display's info section. Check that all signals are in the correct states.
The correct states are as follows:

Power on, high voltage off:
Main Contactor: False
Precharge: False
APPS OK: True
BSPD OK: True
BMS OK: True
TS Voltage: 0V

Power on, high voltage charging:
Main Contactor: False
Precharge: True
APPS: True
BSPD: True
BMS: True
TS Voltage: Increasing and within 230-300V

Power on, high voltage charged:
Main Contactor: True
Precharge: False
APPS: True
BSPD: True
BMS: True
TS Voltage: Stable at 230-300V

If APPS, BSPD, BMS are ever false this represents a fault. If the main contactor and precharge are ever true at the same time for more than a few seconds that is a fault.

Testing: See Test Plan section above.

Disassembly: The disassembly of our Capstone portion of the project can be tedious and can vary based on the extent of how much the wiring was integrated into the rest of the car. For some of the boards, the wiring harness allows for easy removal by just disconnecting the connector mates from the board. Any obvious wiring harnesses that are tied to the chassis of the car using zip ties will have to be removed prior to disconnecting the boards.

- **Tooling**

Molex Crimping Tool (provided by the sponsor)

Soldering Iron in VMS lab or in the Capstone Lab

Other wire cutters, crimpers, along with several other basic hand tools from VMS lab

Power generator and oscilloscope for testing.

Altium Design version 21.9.2 (version updates frequently) on Windows

- Go to the Altium website. https://www.altium.com/altium-designer
- Create an account using your PSU credentials and allow the sponsor to give you access to the software.
- Once you have access to the license, download the software from the website and install the software.
- Once installed, setup the software by creating PCB rules, via sizes, trace widths, stitching dimensions, etc. based on your requirements.

STM32CubeIDE version 1.12.1 on Windows

- Go to the STM32CubeIDE download page on the STMicroelectronics website. https://www.st.com/en/development-tools/stm32cubeide.html Look for version 1.12.1 and select the installer's download link for Windows. Keep the installer file saved.
- Accept the license agreement and choose the installation directory
- Select the components to install (it's recommended to keep them selected by default)
- Configure the debugging tools (choose your specific tools or leave the default options selected)
- Review the installation summary and launch STM32CubeIDE
- On the first launch, STM32CubeIDE then prompts you to configure some initial settings, like the workspace location. Follow the on-screen instructions and provide the necessary information

Faculty Advisor Approval

Advisor Name: Roy Kravitz
Signature: e-approved by Roy Kravitz (roy.kravitz@pdx.edu)
Date: 17-June-2023

Project Sponsor Approval

Viking Motorsports (VMS) is a student group focused on designing and building formula race cars from the ground up. Every year, VMS competes in Formula SAE, which is a worldwide collegiate competition created by SAE International.

Sponsor Name: Braeden Hamson
Signature:

Date: 6/16/2023