Battery Management System
For Electric Formula SAE Racecar
Winter Quarter Report

by

Drake Vogelpohl

TABLE OF CONTENTS

1 Introduction

This will focus on the hardware and firmware development of a custom Battery Management System (BMS) using the Analog Devices (AD) adbms68 chipset for a Formula Electric SAE vehicle. This will allow us to have higher confidence, better integration, and design of our high voltage battery (accumulator) that powers the vehicle. Projects like this that develop new innovative designs drive the learning and knowledge base on this team and will allow us to continually improve the competitiveness of the overall team.

NOTE: This report is being written at the end of WQ25 and thus will focus more on the designs and choices made throughout the quarter and less on background and upcoming plans. The idea is to come back next quarter and add the background, new designs, and testing that was done to validate the functionality. The goal for this report is for it to be referenced by any (NFR member) designing any BMS in future years, as well as documentation of work done this and next quarter.

1.1 Winter Quarter 2025 Scope

The work of this project is split into two different quarters. Winter quarter 2025 (WQ25) is focused on understanding the hardware, the initial PCB designs, and the initial firmware. The goal was to make an MVP that could be used in our current accumulator. Given the short length of each quarter, sacrifices had to be made in order to meet this deadline and we plan on addressing these sacrifices next quarter. Furthermore, this had to be designed as a complete “drop-in-replacement” for our current BMS, placing further restriction on the operation of the BMS, mainly on interactions with the rest of the car.

1.2 Spring Quarter 2025 Scope

Spring quarter 2025 (SQ25) will be focused on integrating with the rest of the car, validation of the BMS via testing with the accumulator both static and under load, and addressing the sacrifices made in the previous quarter.

2 Background

Northwestern Formula Racing (NFR) is a student-led organization that designs, builds, manufactures, and tests Formula style cars at Northwestern University. A 600V, 6.9kWh battery pack designed using 4.5Ah lithium-ion cells powers the vehicle. The battery pack needs a Battery Management System (BMS) to monitor individual cell voltage and to balance voltage differentials that arise due to cell tolerances. Commercial-Off-The-Shelf (COTS) BMSs exist, but they are expensive, large, and not optimized for individual battery packs. Developing a custom semi-distributed BMS offers several advantages, including lower costs, higher measurement accuracy, and simpler wiring achieved by placing the measuring device close to the monitored cells. This project will design, implement, and test a BMS specific to NFR's battery pack using Analog Devices' BMS chipset and an STM32 microcontroller.

The BMS lives in the High Voltage battery, or accumulator. The mainboards live in a section called the penthouse above the cells. The daughterboard lives on the individual segments, shown in Figure 1.

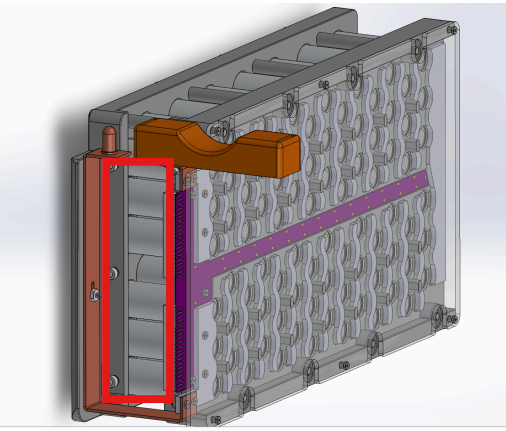

Figure 1: CAD model of a Segment Highlighting Where Custom BMS Would Sit in Red.

3 Hardware

The hardware is broken into two sections based on the two different PCBs that were designed. The “daughterboard” is the PCB that lives on each segment and does the actual voltage and temperature measurements as well as balancing cells when necessary. The “mainboard” lives in the penthouse (top section of the accumulator, above the segments) and communicates with the daughterboards, communicates with the rest of the car, controls the isolation relays, and measures the current flowing into/out of the accumulator.

3.1 Daughterboard

One daughterboard will live on each segment and contains two ADBMS6830 chips to perform 28 series voltage measurements and 16 temperature measurements. This board is constrained to the face of the segments (Figure 1) and has galvanic isolation on the communication in/out of the board. The daughterboard gets the signal lines through a sense PCB that is electrically connected and fused to each of the series cells. There is a connector on that PCB and a corresponding connector on the daughterboard that will be connected via a flex PCB acting as a ribbon cable. The daughterboard is designed according to best practices and the adbms6830 datasheet and functional block diagram (Figure 2) [1].

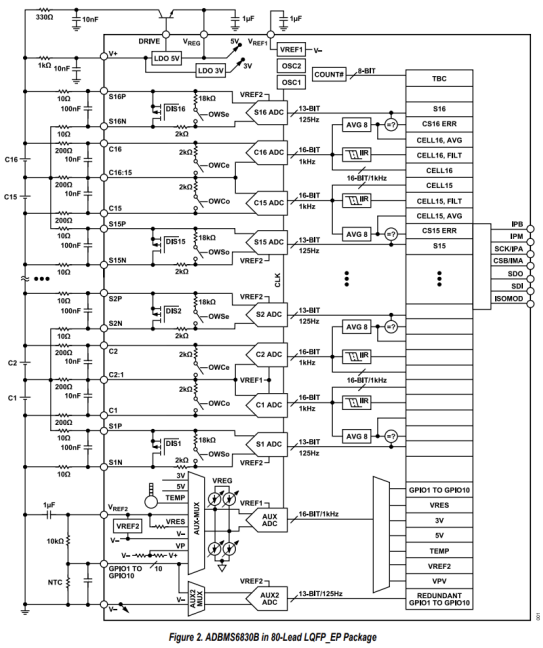

Figure 2: Functional Block Diagram for adbms6830B

Each daughterboard has the following sections of hardware: voltage sensing, temperature sensing, cell balancing, chip power, connectors, and communication lines.

Voltage Sensing is done by routing each series cell connection to the BMS. This is done by “sense” and “flex” PCBs described in more detail later. What is important to know is that the voltage sensing is differential on this chip. This means that if each of the 14 cells that a chip sees is at 4.2V there would be a voltage differential of 58.8V between the negative terminal of the bottom cell and the positive terminal of the top cell of this chip. However, each cell has its ***own*** channel and corresponding 16-bit Sigma-Delta ADC. It is interesting to note that other BMS chips have different ADC configurations. For example, Texas Instruments' BQ BMS chipset uses only one 24-bit Successive Approximation ADC that is muxed between each cell. The voltage sensing hardware is fairly straightforward, just a 200 Ohm resistor in series and a 10nF differential capacitor between sensing lines as shown in Figure 2, where Cellx is the voltage signal line input for cell x and Cx is to the corresponding pin on the chip.

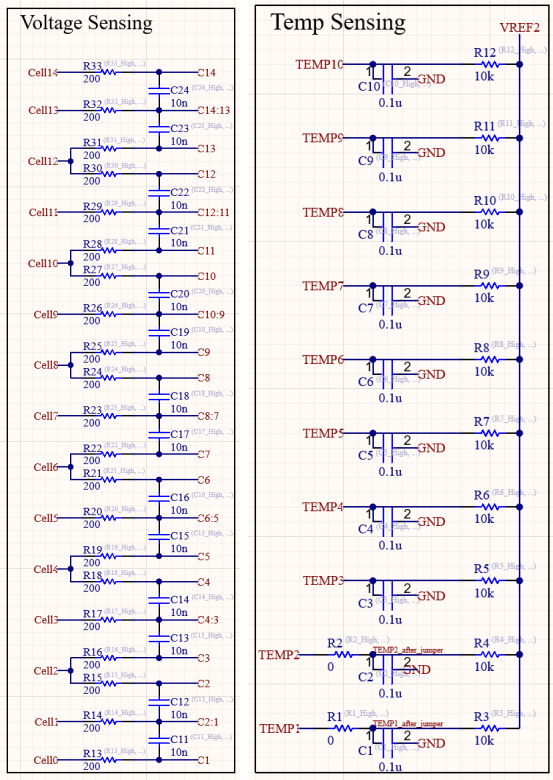

Figure 3: Schematic of Voltage and temperature sensing

Temperature sensing is done by a voltage divider between an on-board resistor and a thermistor on the segment. The chip provides a regulated reference voltage of 3V to be used for this purpose. Our negative temperature coefficient (NTC) thermistors are glued with thermally conductive, electrically insulative epoxy to the outside of cells evenly spaced throughout the pack. We decided to use 10k Ohm thermistors with long leads so that we can easily solder them from the cells to the sense PCBs. As shown in Figure 3 there are 10k Ohm resistors from Vref2 to the corresponding thermistors in the pack. A decoupling 100nF capacitor across the thermistor because the daughter board is mounted on the segment and sees high EMI. The chip then measures the voltage across the thermistor in reference to V-.

Voltage sensing is done on the C channel where each voltage has a dedicated pic of Cx. Cell balancing is done on a second, S, channel. There is a dedicated positive and negative pin for each cell labeled as SxP or SxN where x is the cell. This chip supports passive cell balancing where excess charge can be sloughed off via shorting a resistor across the cell. This can be done via the internal FETs that support up to 300mA of current, or external FETs that can be sized according to cell balancing needs and controlled via the internal FETs. We decided the internal FETs current would suffice. To get near this current we use 20 Ohm cell balancing resistors. Ideally two 10 Ohm resistors would be on either side of the internal switch to limit the heat dissipation each resistor sees. However, an LEDs was wanted across the cell balancing resistor to give a strong visual when cell balancing was occurring, and the forward voltage of the LED throughout the voltage range of the cell (2.5V-4.2V) would not be possible if two resistors were used.

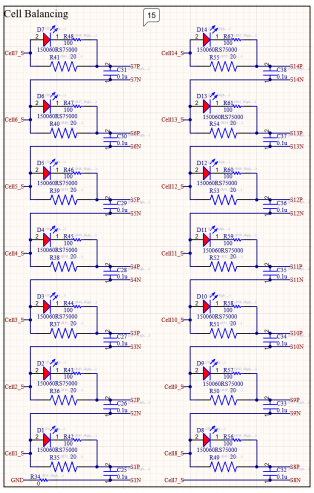

Figure 4: Cell Balancing Schematic

Due to reading highly sensitive analog voltage lines and needing to cell balance and power the chip on the same lines, kelvin connections are required. The sensing kelvin connections would ideally be broken off at the busbars, but due to needing thin wire bonds and not wanting to have twice as many potential points of failure, they are broken off at the sense PCB pads. This provides a path for current, drawn by powering the chip and cell balancing, that is separate from the path that sensing is done on. The purpose of this is while the resistance in the lines should be quite low and the current is also low, the signals are so sensitive that they still need to be separated. This is even more important when the sense lines are very long as is the case for our battery pack, causing there to be very non-ngelible resistance. The voltage drop across the sense PCB traces will be more important later.

The chip powers itself directly from the cells it is measuring. This also means it is always powered and thus can kill your battery. Luckily, the chip is only fully on and drawing the max current, <30mA, when actively being communicated with. When it is not, e.g. the accumulator is out of the car and the mainboard is not powered, the chips only draw a few micro amps of current, hardly enough to kill a large battery such as the accumulator unless left for thousands of days unattended. The chip powers itself through an internal LDO or low drop out linear regulator. It functions by changing the effective resistance of an external BJT to provide a constant voltage at the different current draws of the chip. Decoupling capacitors were placed as well as a ferrite bead to help maintain a constant voltage. Finally an LED was placed in series such that it will light up whenever the chip is not in sleep mode.

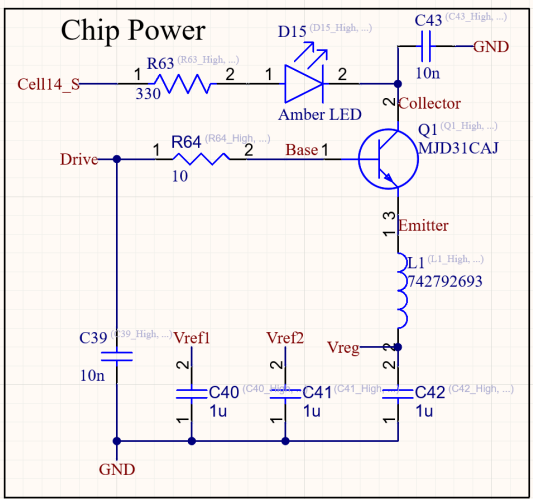

Figure 5: External LDO circuitry

The connectors for the daughterboard were chosen to meet the voltage requirement of each segment. The ones we chose have SMD pads and the header pins on the daughterboard and corresponding connector on the flex PCB to shield the sense lines from accidental shorts as much as possible. The problem with these high pin count connectors (ours are 40-pin) is they are hard to remove but don't have positive locking. They are difficult enough to remove such that they "shouldn't" come off while driving, but that has the added annoyance of being quite difficult to remove and sometimes rip the flex PCB when removing. A solution to look into for future years are zero insertion force, or ZIF, connectors with a positive locking feature. This connector has a couple of benefits. Mainly, as implied by the name it is very easy to install the flat flex cable and is locked such that it wouldn't be loosened by the vibrations in an automotive environment. Secondarily, we already use flex PCBs to route the signals from the sense PCBs to the daughterboards. This means that we would only have to buy one connector (the ZIF) instead of two (one connector and corresponding header pins), and that we would no longer have to solder to the flex PCB, cutting down on complexity and cost. The only fear with ZIF connectors is that it feels it is possible to misalign the pins when inserting due to the small pitch and no physical pins to align anything. If adjacent pins are misaligned or shorted together, this would cause no harm in a normal setting when everything is powered off when plugging in. However, in this use case the batteries are always energized so a misalignment or short would be detrimental causing a dead short on the cells and blowing the fusing. While the fusing is there for this exact reason, it is very difficult to replace and something that ideally would never be tested. For this reason, more testing via a PCB with this connector and a corresponding flex PCB would be required to prove that it will never short when inserting before it is used with cells.

Finally the communication lines. The comm lines (Figure 6) are the least straightforward aspect of this PCB. The datasheet does not give very many helpful references for how to design the circuitry for them. What is known is that there needs to be resistors tying the differential lines together, series resistance in the lines, decoupling capacitors to help with noise, TVS diodes to prevent voltage differentials killing the chip when plugging the connector in, and some form of galvanic isolation to isolate each chip in the pack from each other and the mainboard. To start, the datasheet provides a reference for 22nF of series capacitance if using capacitors for galvanic isolation, and the TVS diodes are just standard used for this purpose. Looking into similar

isolation protocols, online resources, and back of the napkin math was done to determine 47 Ohms of series and termination resistance. Finally based on the 2mbps speed of the communication protocol, 220 pF decoupling capacitors were chosen to properly filter out noise while leaving the signal intact. A common mode choke was also chosen based on the speed of the isolated protocol to provide better signal integrity. It works by using the common mode choke in differential mode where the flux in the core cancles to pass only differential-mode current. This helps if there is noise that affects only one of the lines, by filtering it away.

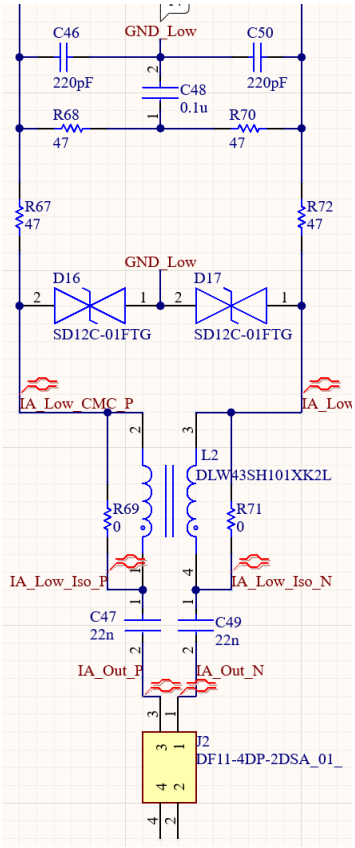

Figure 6: Communication Circuitry

The communication protocol itself is a proprietary isolated Serial Peripheral Interface or isoSPI communication protocol developed by Linear Technologies (since acquired by Analog Devices). As mentioned above, it runs at 2 mbps and operates in a daisy-chain configuration. This means that each chip has a port where it is the slave, port A, and a port where it is the master, port B. Each chip will listen on the slave port for any communication and then pass the signal up the line via the master port. This allows as many chips to be in the daisy-chain as needed.

3.1.A Cell Depopulation

Because each chip has the capability to measure 16 cells and we only want each chip to measure 14 cells due to the accumulator architecture, we have to depopulate the top channels by tying them to the top cell via a resistor as described in the datasheet and shown in Figure 7. Note that we are depopulating an even number of cells, and there is different hardware for depopulating an odd number, again outlined in the datasheet.

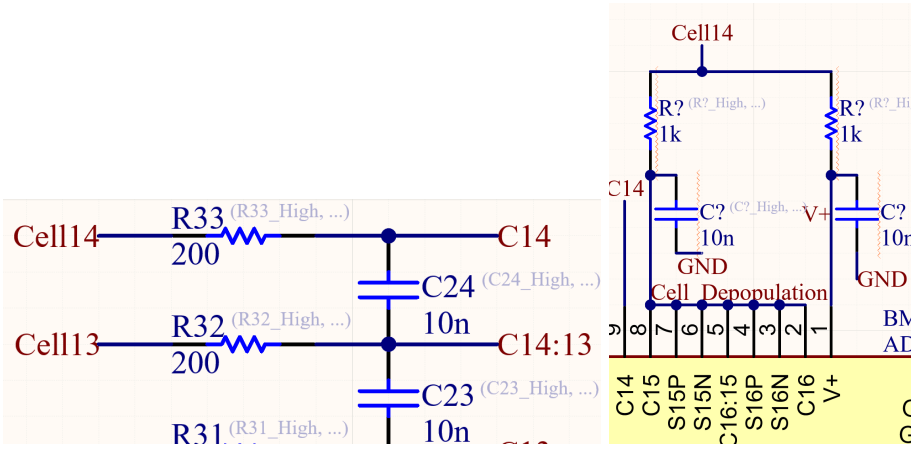

Figure 7: Cell Depopulation Schematics

3.1.B Sense PCBs, Flex PCBs, and Issues

While these were not designed by me (designed courtesy of Anton Walvoord), they are integral to the design of the BMS. We have sense PCBs that are glued to the face of each segment as shown by the purple PCB in Figure 1. The PCB has exposed pads near each busbar and are connected to the busbars via a ~1A fusing wire bond. Wire bonding for fusing is not required, but the accumulator requires cell wire bonding to do negative tab watercooling. As a result, we have a wire bonding sponsor, Hesse Mechatronics, who was willing to do the sense wire bonds at the same time as the cell wirebonds. At the top of the sense PCB is a header. To get from the sense PCB header to the BMS is the “flex” PCB. This is a flexible PCB with two headers that acts as a glorified ribbon cable. The top section sense PCBs is displayed in Figure 8 for reference.

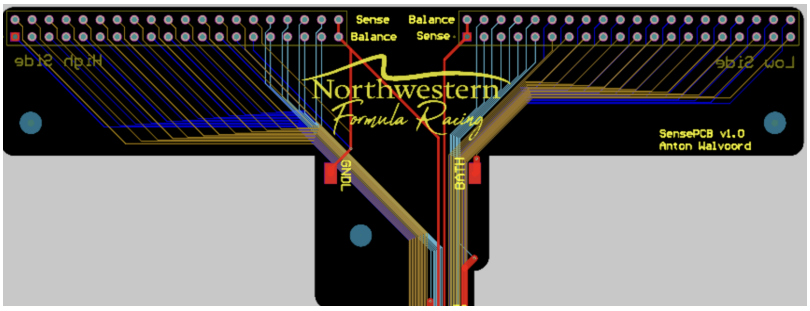

Figure 8: Top section of the Sense PCB

Keen readers might have noticed how thin the traces are in Figure 8. They are the JLCPCB minimum width of 0.1mm. Because these are quite long PCBs, the longest trace has a resistance of ~1.5 Ohms. While this doesn't sound significant because the sensing kelvin connection preserves the sensitive voltage signal, it is significant. When cell balancing with maximum current, there is too much of a voltage drop over the cell balancing lines such that the sense lines go below the minimum voltage in reference to the V- or chip ground where Cx > -0.3V in reference to V-. In other words the sense lines are still the at desired voltage of the busbars due to the kelvin connection, but because of the voltage drop of when balancing at maximum current, the V- is artificially greater than the busbar voltage by more than 0.3V. On top of this issue the pads that get wire bonded to were labeled backwards, i.e. cell1 was labeled as cell14 and vice versa. Unfortunately, due to not enough people checking over it, the sense PCBs were fabricated and glued to the segments before these issues were caught.

These issues were eventually caught and solutions were as follows: flipping the sense lines in the flex PCB to fix the backwards labeling issue. The voltage drop issue was solved by balancing at a lower current determined by estimating the resistance and the maximum voltage drop. Unfortunately the lower cell balancing current means charging will take longer, and the cell balancing LEDs had to be taken off.

3.1.C Layout

The size of each daughterboard is predetermined by the face of the segments it lives on as shown in Figure 1. After that the PCB is broken into two halves, one for each chip. With only the galvanic isolation caps crossing over this gap. On the top layer, (Figure 9) there are the voltage sensing lines (along with associated filter networks), LDO circuitry, and communication circuitry (both in and out) all leading to the chip in the center. The differential capacitors across the sensing lines are placed as close as possible to the chip, as well as the external capacitors that are required by the chip. The power (LDO) circuitry is placed around the outside so as to not interfere with the analog signals, along with cell balancing resistors and the temperature sensing as these lines are not as sensitive. Underneath the sensing lines is an uninterrupted ground pour

to provide good return paths for the signals. The component placement was done in such a way to make it easier to assemble by having many components in a line (apart from the differential caps that were placed as close as possible to the chip and the comms circuitry). The digital communication circuitry was placed such that it was away from the sensitive analog signals and allowed for the lines to route as close together as possible because they are a differential pair. The ground plane does not fully extend under the comms line to not be under the galvanic isolation and ideally a differential pair is its own return path. However, the ground pour could be done better such that there are no impedance discontinuities under the communication lines that could potentially cause ringing.

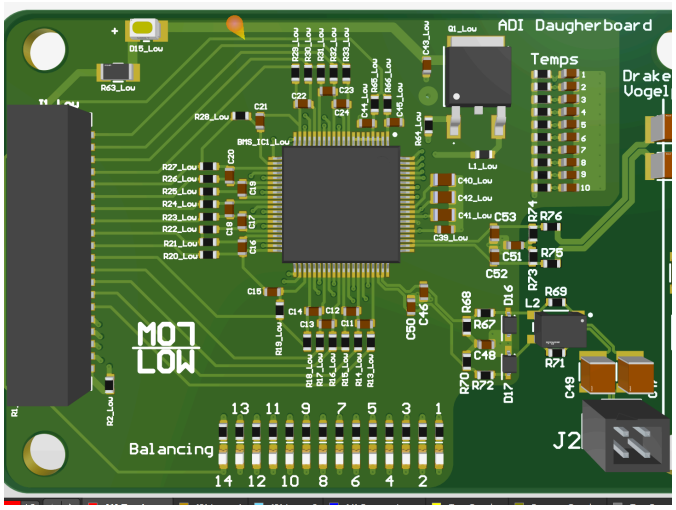

Figure 9: Top 3D view of Daughterboard Layout

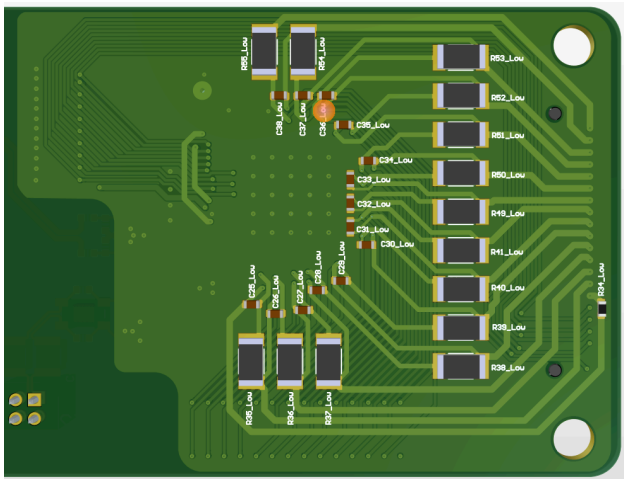

Figure 10: Bottom 3D view of Daughterboard Layout

On the bottom layer is just the cell balancing resistors and their associated differential caps (placed right under their corresponding pads). The purpose of them being on the bottom layer is to make the top layer with the sense lines less crowded. Also cell balancing is switching on and off with noisy edges that can potentially affect the sense lines. For this reason they are placed on the other side of the board with many ground layers of separation. These resistors also can generate a decent amount of heat. When fully balancing with 20 Ohms resistors, there can be up to 0.882W of power per resistor. This heat is not ideal to have right next to the chip and sensing resistors whose value will slightly shift due to temperature.

After learning more about PCB layout and looking at this design more there are some adjustments I would like to make. First, due to oversight the flex PCBs currently cover the high side communication connector as shown in Figure 11. The sensing lines are also spaced decently apart from each other, as to not have them couple with each other, but that doesn't make sense as they are all differential to each other and thus should be placed close-ish together. Furthermore, I would like to have this board be a 6-layer board. Currently the third plane is not just a ground pour, but also has the cell balancing LEDs and thermistors on it. This is not ideal as it compromises the return path for both these signals and the cell balancing signals. The decision to make this a 4-layer board was actively made to save cost with the knowledge that it would lead to decreased integrity as mentioned above. The idea is these signals are not very important and it wasn't worth it to move to a 6-layer board, but it would be interesting to see the design as a 6-layer board.

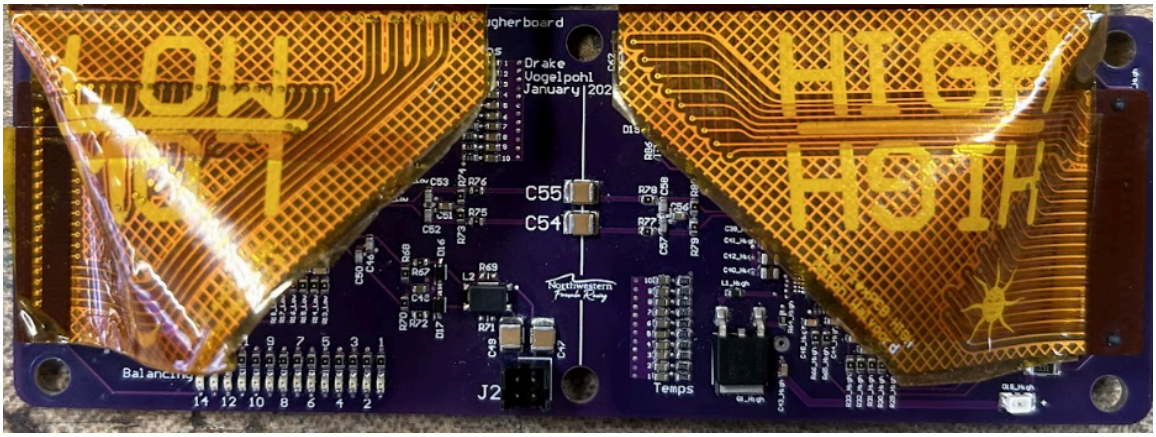

Figure 11: Daughterboard with Flex PCB Attached

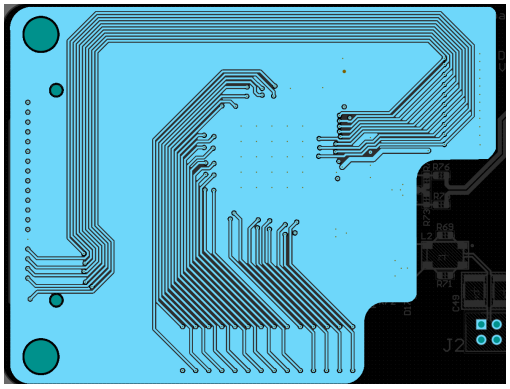

Figure 12: Daughterboard layer 3

3.1.D List of improvements

- ZIF connector as described above
- Moving the high side communication line connector so that it is not blocked by the flex PCB
- Make sensing traces closer together
- Make the ground pour extend completely under isoSpi leading up to the galvanic caps
- Make it a 6 layer (or more board) so that:
    - No traces have to be routed on plane layers
    - The communication lines can be shielded on the top and bottom by ground layers
    - Star grounding could be used to isolate the sensitive voltage sensing signals ground from the rest of the board and connect it only on lower layers
        - I.e. Split the ground pours to protect the analog signals and connect on lower layers at single points with vias
- Signal integrity simulations
    - This would be nice to learn how to do and to validate the current designs susceptibility to external interference

3.2 Mainboard

The mainboard sits in the wiring compartment (Figure ?) and contains an STM32F4 microcontroller and the ADBMS6822 communication chip. This board also contains an isolated amplifier to measure pack current via the pack shunt resistor and circuitry to drive the isolation relays. The mainboard talks with the daughterboards to gather cell data, evaluate fault states, and communicate this with the rest of the car over a Controller Area Network (CAN) bus. The firmware lives on the STM32F4 and will execute the functionality described above.

The mainboard is a different complexity compared to the daughterboard. The daughterboard has fairly straightforward circuitry dictated by the datasheet. That being said there is a lot to talk about a simple capacitor network for example for the daughterboard due to describing how it interacts with the ADBMS6830B chip. The layout is also much more interesting as well for the daughterboards as a large portion is sensitive analog lines. The mainboard has more complex circuitry but it is “less interesting” as there is no external interaction with an ASIC.

On the surface the mainboard has the following tasks: regulating its own voltage, the STM32 corresponding circuitry, ADBMS communications via the ADBMS6822 chip and corresponding circuitry, current sensing, isolation relay control, and sending its data over CAN.

Starting with voltage regulation, the vehicle has a power and ground line that runs throughout the harness. These come from the low voltage battery, made from the same cells as the accumulator, connected in a 6 series 3 parallel configuration resulting in a max voltage of 25.2V and a min of 15V. Due to changing in voltage and the chips on this board taking both 3.3V and 5V, we have to regulate this voltage. Rather than regulating once at the battery and sending that regulated voltage throughout the car we choose to run the unregulated voltage because it is a higher voltage so we have less current flows and we get lower transmission line losses. Also most boards use either 3.3V or 5V (or both) and as a result twice as many wires would be needed in the harness. Regulation is done by switching converters. On this team we have standardized this component and use the Traco TSR switching converters. Specifically, the mainboard uses the TSR-1-2433 and the TSR-1-2450 regulating to 3.3V and 5V respectively. These come with built in inductors and capacitors, making them very easy to use. To better decouple the regulator, a 4.7uF bulk and a 100nF decoupling capacitor were added to V_in on each regulator.

In case the unregulated line was plugged in backwards, reverse polarity protection was added. This was done with a PMOS with drain tied to the unregulated input, source tied to the input to the regulators, and gate tied to the drain (Figure 13). The PMOS has a Vds of 30V but a Vgs of only 25V. Because the LV battery (unregulated line) can go up to 25.2V, a voltage divider to the gate was placed to lower the Vgs in the event of a reverse connection. Also like most signal lines on this board, a decoupling cap was placed from the gate to dain. A 0 Ohm DNP resistor was placed across the PMSO in the event that this circuitry was incorrect so the PMOS could be bypassed without needing to respin the board.

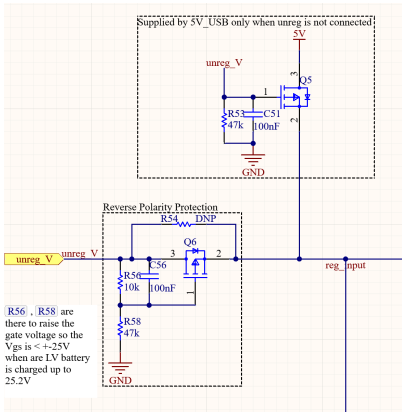

Figure 13: Reverse Polarity and 5V supply schematic

In order to not constantly need a power supply or battery to supply unregulated when testing, a similar PMOS was placed from 5V to the regulator input with the gate tied to the regulator input (Figure 13). The PMOS turns off when unregulated is not present as to not short 5V to unregulated. However, when unregulated is not present, the PMOS turns on and allows 5V to supply voltage to the regulators. Normally a Schottky diode would be used, but as this line is already 5V so the voltage drop across the Schottky would be undesirable. 5V is high enough above the dropout for the 3.3V regulator such that it can still work, and the supplied 5V through the micro usb will directly power the chips needing 5V.

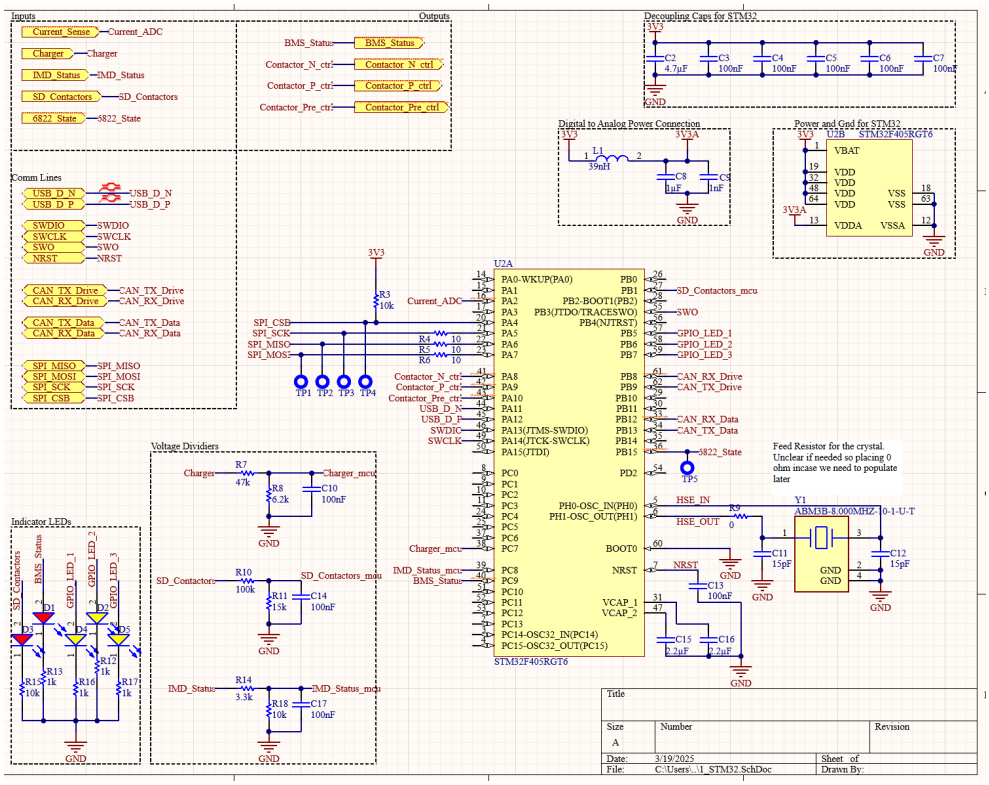

Figure 14: Schematic for STM32

The schematic for the STM32 corresponding circuitry is shown in Figure 14. Decoupling capacitors were placed where the SMT32 requires them, and on every power pin. An inductor was added to tie the 3.3V power to STM32 to the STM32 analog power. A quartz crystal was added to help reinforce the timing of the chip due to having both SPI and CAN communication. The 0 Ohm resistors in series with the crystal output is a feed resistor that is sometimes required to limit the current flowing into the crystal. Lastly there are 3 voltage dividers. The charger, shutdown contactors and IMD status. These are needed so the SMT32 can read the digital status of these lines (high or low).The charger line is at 24V regulated, the shutdown contactors is the same voltage as the LV battery, and the IMD status is at 5V. The STM32 has a max GPIO voltage of 3.3V, resulting in the need to divide this voltage down to be in an appropriate range for the SMT32. The BMS reads the IMD status so it can send it to the rest of the car (namely ECU). It has to do this because IMD status is determined by the HV board and the HV board is strictly hardware.

Status LEDs were placed throughout the board to visibly show specific digital line values and are described in Figure 15.

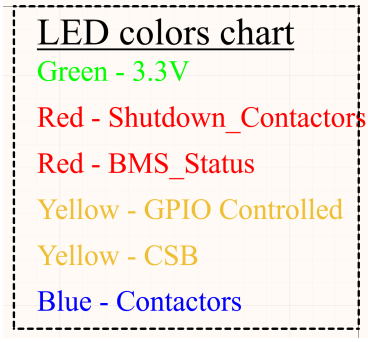

Figure 15: LED color code chart

The BMS status out line that originates from the STM32 and gets set to other boards on the car needs to be at 5V. As a result, a 3.3V to 5V level shifter was added (Figure 16).

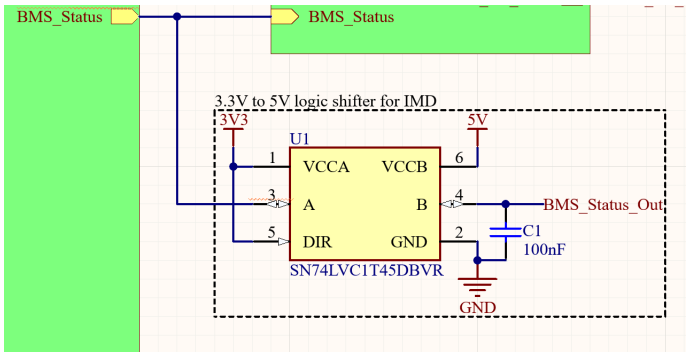

Figure 16: BMS Status 3.3V to 5V Level Shifter

A micro usb connector was added to be able to program the chip with USB. 22 Ohm feed resistors and a TVS diode made for USB was added (Figure 17) and USB was routed as a differential pair.

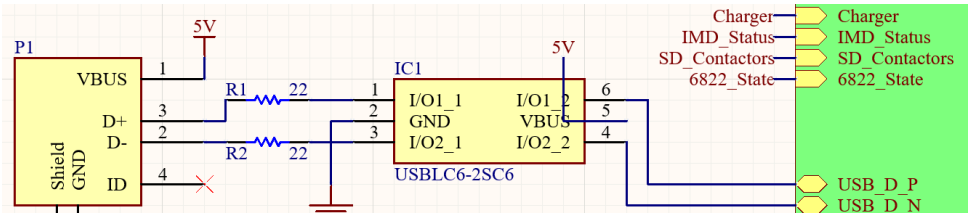

Figure 17: USB Circuitry

The SMT32 has SPI, but that needs to be translated to isoSPI to talk to the ADBMS6830B chips on the daughterboards. To do this is the ADBMS6822 SPI to isoSPI transceiver. The transceiver is fairly straightforward and SPI is connected to the corresponding SPI pins. Because the SPI is from the STM32 it is at 3.3V so the VDDS pin is connected to 3.3V. The isoSPI operates at 5V so the VDD pin is tied to 5V. An LED was also placed on the chip select bar (CSB) line such that it turns on whenever the line is pulled low to be a visual indication of when the mainboard is talking to the daughterboards. The isoSPI circuitry is identical to the daughterboards with one major change. A 1-to-1 transformer is used instead of capacitors to provide galvanic isolation. While this costs more and thus is not used on the daughterboards it provides safer isolation and is thus used on the mainboard.

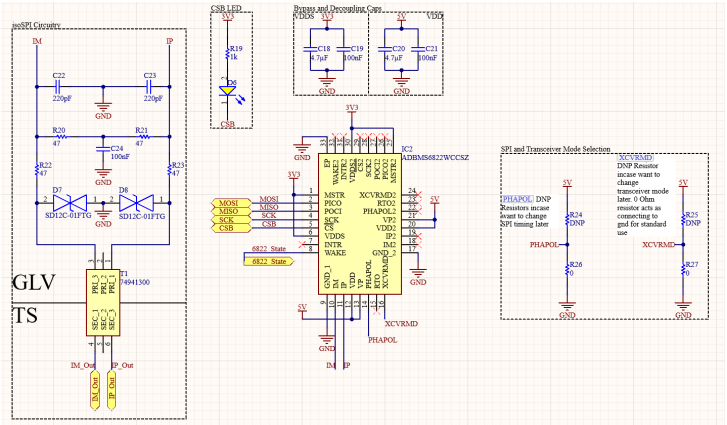

Figure 18: ADBMS6822 Circuitry

Two CAN transceivers were added on this board so the BMS can talk on both of the CAN lines we have in the vehicle. The CAN transceivers are standardized across the team. Data can is the CAN line that the CAN line that the sensors and other data acquisition boards talk over. On this line, the BMS spits out every voltage and temperature that it measures. Drive can is the CAN line that drivetrain electronics talk over. This is where the BMS sends out the BMS status (and what fault is present, if any), IMD status, max cell voltage, min cell voltage, max cell temperature, min cell temperature, battery voltage, and measured current. This can be thought of

as the important data that the rest of the car needs in order to drive. The drive CAN line is also where the BMS receives the ECU, Inverter, and Charger CAN messages.

The BMS controls 3 isolation relays (also called contactors or IRs) that control if tractive system (TS) voltage is present outside of the accumulator. There are two high current relays, one for TS+ and one for TS-. There is also a precharge relay. This relay along with a 6k Ohm resistor in series is connected across one of the high current relays to provide a current limited way to charge up the TS outside of the accumulator. This is necessary because the inverter has a large DC-link capacitor that has to be charged to the tractive system voltage before both high current relays are close to prevent a short. There is a circuit that runs throughout the vehicle called the shutdown circuit. This is a latching circuit i.e. if something is high it stays high until reset. The shutdown circuit is what directly powers the contactors. The BMS gets the shutdown circuit in and needs to switch it once based on if there is an internal BMS fault. The shutdown line then goes out of the BMS. The BMS then gets the shutdown circuit back at the end of the circuit and uses this to control the isolation relays. As a result, the BMS needs 4 switches to control the circuit. Once the first time and one switch per isolation relay.

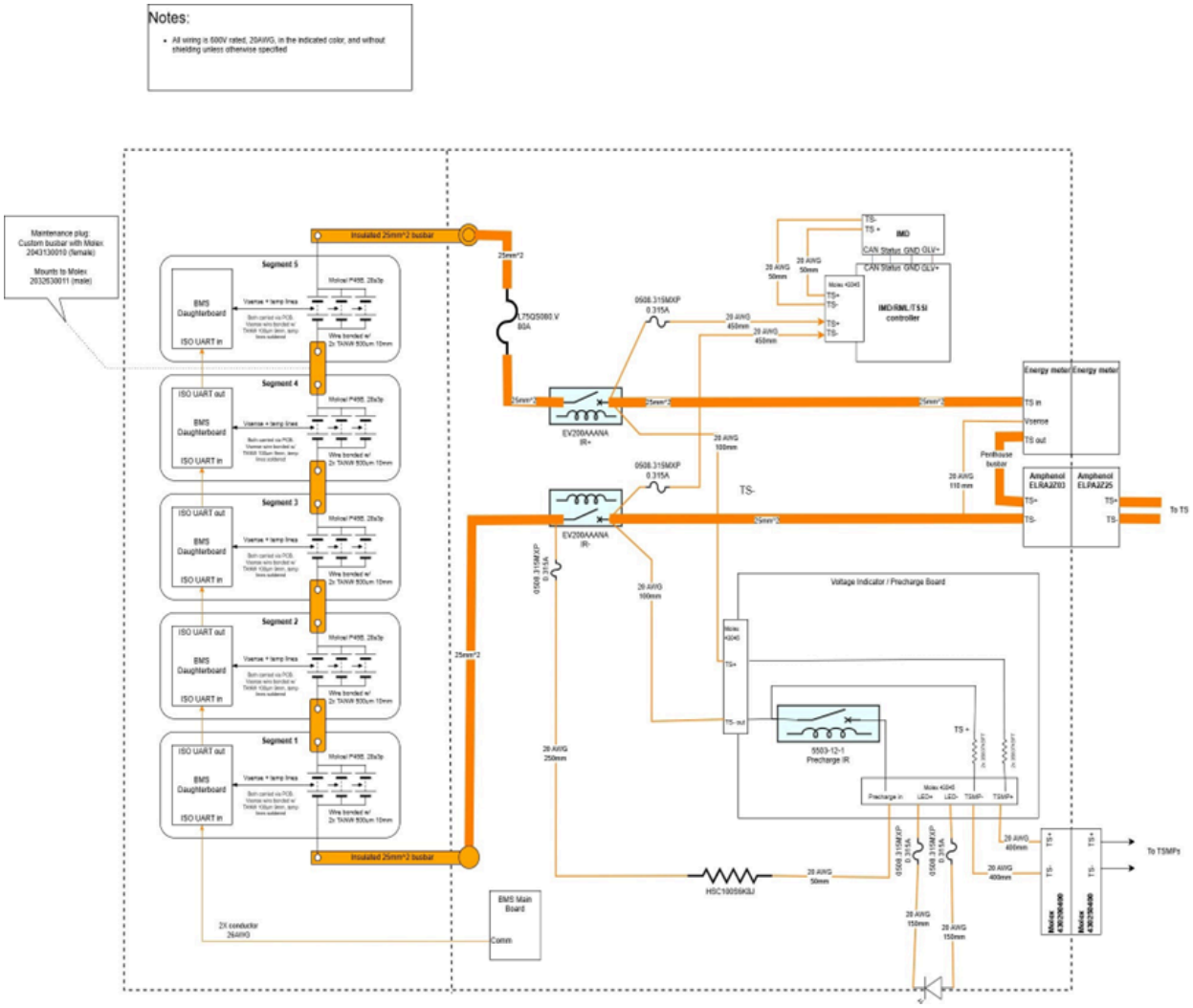

Figure 19: Accumulator TS Schematic

The switch is controlled by the STM32. The signal lines go into the gate of an NMOS that is used to invert the signal and drive a PMOS that is in series with the shutdown line. The gates of both FETs are pulled to the drains so that in a case with no signal from the SMT32 it defaults to off. Much like the reverse polarity protection PMOS, this PMOS has a voltage divider to drive the gate due to the Vgs of 25V and max shutdown voltage of 25.2V. Again like most signal lines, there are decoupling caps placed on the gates of the transistors. Lastly, there is a flyback diode placed across the isolation relays to limit a large voltage being generated by the inductor in the isolation relays. Figure 20 displays the schematic for one of the 4 shutdown switches. The rest are more or less identical.

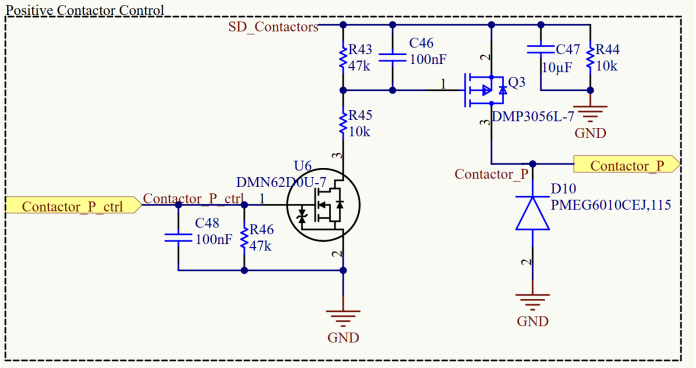

Figure 20: Isolation Relay Control Schematic

Finally the mainboard current sensing. This is arguably the most interesting aspect of the mainboard circuitry. At a high level, there is a 100uF shunt resistor connected in series with the high current path. Because of this a small voltage is developed across the resistor dependent on the current. The mainboard takes this small differential voltage, isolates and amplifies it and turns it into a single analog signal that the STM32 can read. It does this through an isolated amplifier that galvanically isolates the tractive system. Specifically the AMC3302DWE isolated amplifier that has a fixed gain of 41. This chip has an integrated isolated power supply and has dedicated pins to the connection across the shunt resistor to maintain the kelvin connection. The isolated amplifier spits out an isolated and amplified differential pair. The differential signal is then fed into a differential amplifier which is low pass filtered and fed into an ADC on the STM32 according to the schematic in Figure 21.

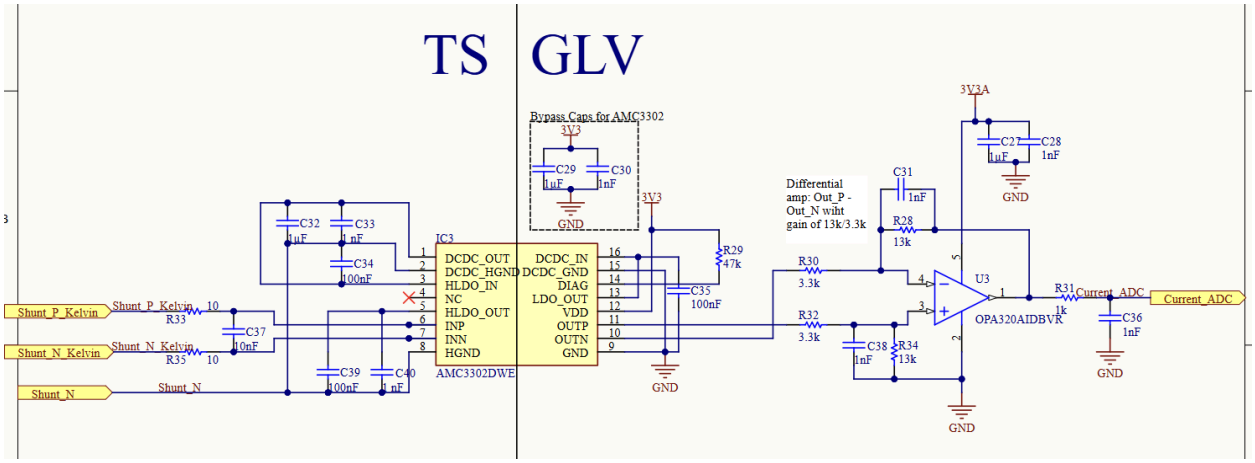

Figure 21: Current Sensing Schematic

The shunt resistor has a resistance of 100u Ohms. For the max continuous current of 135A this results in 13.5mV * Fixed gain of 41 = 0.55V differential output of the isolated amp. Budgeting for worst case 200A, the isolated amp output voltage is 0.82V. In order to maximize the precision

a gain was chosen to use the full ADC range of 0 to 3.3V. Having a gain of 4 results in a max voltage of 3.28V at 200A. The STM32 ADCs are 12 bits resulting in ~0.05A/LSB.

3.2.A Layout

The layout is not as notable as the daughterboard. Care was taken to minimize the area while still maintaining signal integrity, including separating the analog current sensing circuitry from the rest of the circuitry. Other best practices were followed such as placing decoupling capacitors as close to the pins as possible.

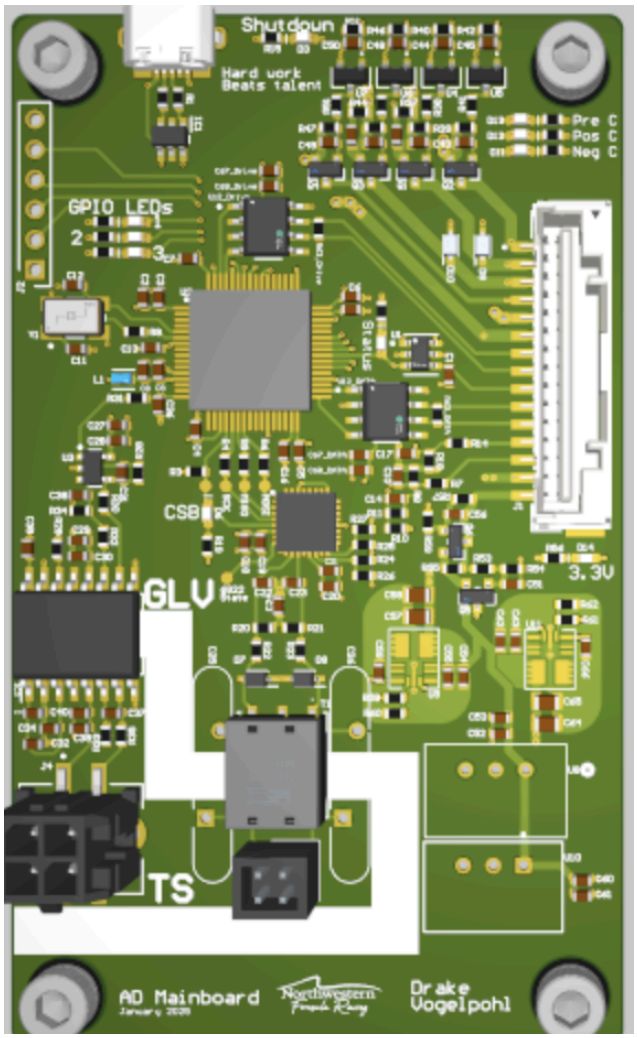

Figure 22: 3D view of mainboard layout

The only non-self explanatory aspect of the layout is the TS keepout zone. Because this board has both tractive system and grounded low voltage (every electrical part not part of the tractive system) present, a keepout zone is required due to the high voltage differential and by FSAE rules shown in Figure 23.

EV.6.5.7 If Tractive System and GLV are on the same circuit board:

a. They must be on separate, clearly defined and clearly marked areas of the board

b. Required spacing related to the spacing between traces / board areas are as follows:

Voltage	Over Surface	Thru Air (cut in board)	Under Conformal Coating
0-50 V DC	1.6 mm	1.6 mm	1 mm
50-150 V DC	6.4 mm	3.2 mm	2 mm
150-300 V DC	9.5 mm	6.4 mm	3 mm
300-600 V DC	12.7 mm	9.5 mm	4 mm

Figure 23: FSAE Rules for TS and GLV on the same board

The thick stripe is 4mm wide and denotes where the keepout zone is that separates the tractive system from the rest of the board with only the galvanic isolation crossing over the keepout.

3.2.B List of improvements

- Due to charging and regenerative braking while driving, current not only flows out of but also into the accumulator. Currently the current sensing only reads the discharging current and will saturate to 0V when current is negative (charging). To address this there needs to be an offset on the differential amplifier and change the gain. I.e. R34 connected to an offset of 1.65V (3.3V/2) rather than ground.
    - Setting the offset to 1.65V and a gain of 2 results in using 0.1V-3.29V of the ADC and ~0.1A/LSB
- Due to the high voltage present in the accumulator any small measurement error in current results in large error in charged/discharged power. MCUs like the STM32 do not have very high precision ADCs because it is not necessary, and if high precision is needed external ADCs are used. To solve this adding a more sensitive external ADC would be a large improvement.
- Copper was poured under the TS keepout and needs to be removed
- Start grounding for the analog current circuitry would be interesting to look into to better decrease the noise present in the current measurement

4 Firmware

The firmware is broken into two sections. The “low level” section interacts with the ADBMS chips to get the voltages, temperatures, and status register values and perform open wire checks and cell balance when appropriate. The “high level” section focuses on the functionality of using the custom BMS as the brains of the accumulator and parsing through the data collected by the low level section.

4.1 AD BMS (Low Level)

The low level code is mainly interacting with the driver files Analog Devices (AD) supplied to Northwestern Formula Racing after signing an NDA. These drivers are used to talk to the ADBM6830B chips that live on the daughterboards to do the following actions: configure the chips, get the status registers, get raw voltage values, get raw temperature values, perform open wire check, and cell balance when necessary. After using these drivers to get raw values, they are processed to usable values in the calculate values step. These processed values are then used to update internal faults in the AD internal faults step. That is where the low level stops. Every variable the low level updates are called AD variables/values (any variable or value that is updated with data from ADBMS630B chips). For complete understanding, the high level section calls the low level functions (update, calculate and process faults), reads current, reads external values, and updates the finite state machine context in the update values steps. The high level then evaluates the finite state machine and sends/receives CAN messages with the values updated in the previous step.

To reiterate the low level steps are: configure the daughterboard chips on startup, update raw AD values in the AD update step, process the AD values in the calculate values step, and update internal faults in the update internal faults step. These steps are done in two files adbms_update_values.c/h and adbms_owc.c/h. This section will be going over these two files and explaining some of the less clear decisions that were made and problems and the solutions to those problems that came up when trying to communicate with daughterboards.

To start, the chips are configured with the following settings: turning on the reference voltage, turning every internal GPIO pull down off so they can be used for temperature sensing, setting the internal overvoltage and undervoltage in the hardware of the ADBMS6830B chips, and turning off every cell internal balancing switch (Figure 24). After setting the chip configurations, cell sensing on the c channel is turned on with continuous measurement. This means the c channels will update their values continuously until turned off or the chips go into sleep mode. Sleep mode is wherever the chips go 2 seconds without being communicated with and the reason for the wakeup command. The c channel voltage sensing is set with the redundant measurements turned on, discharge off, and open wire off. Next the auxiliary channels, the channels temperature sensing is measured on, are turned on. This is done with auxiliary open wire switch off, pull up turned off, and the channels muxed to ADC functionality. As a side note, the auxiliary channels also support other behaviors than just analog to digital and have internal pull ups/downs.

```
for (uint8_t cic = 0; cic < NUM_CHIPS; cic++)
{
    /* Init config A */
    ADBMS_ICs[cic].tx_cfga.refon = PWR_UP;
    ADBMS_ICs[cic].tx_cfga.gpo = 0X3FF; /* All GPIO pull down off */
    ADBMS_ICs[cic].tx_cfgb.vov = SetOverVoltageThreshold(OVERVOLTAGE);
    ADBMS_ICs[cic].tx_cfgb.vuv = SetUnderVoltageThreshold(UNDERVOLTAGE);
    ADBMS_ICs[cic].tx_cfgb.dcc = 0;
}
adBmsWakeupIc(NUM_CHIPS);
adBmsWriteData(NUM_CHIPS, &ADBMS_ICs[0], WRCFGA, Config, A);      You, 4 weeks ago • Ma
adBmsWriteData(NUM_CHIPS, &ADBMS_ICs[0], WRCFGB, Config, B);

// turn on cell sensing
adBms6830_Adcv(RD_ON, CONTINUOUS, DCP_OFF, RSTF_OFF, OW_OFF_ALL_CH);
Delay_ms(1); // ADCs are updated at their conversion rate is 1ms

adBms6830_Adax(AUX_OW_OFF, PUP_DOWN, AUX_CH_TO_CONVERT); // turn on cell aux conversion,
Delay_ms(8);
```

Figure 24: AD chip configuration

The adBmsWakeupIc, adBmsWriteData, adBms6830_Adcv, and adBms6830_Adax commands are all functions part of the driver library provided by Analog Devices. ADBMS_ICs is an array of type cell_asic, again defined in their library. All of their functions that talk to the BMS chips take this array and use it to write to the chips and/or populate the corresponding values in the array after a read. The ADBMS_ICs variable needs to be a global variable. This took quite a lot of debugging to figure out, but if it is malloc-ed and part of a different array the values are not updated correctly. Their functions that update the values do some pretty convoluted (probably unnecessarily complicated) logic to populate the values it receives from the daughterboard chips

including collacs. As a result, if this is not a global variable the values that get updated are shifted over so some are twice the expected value and some are 0. This is thought to happen due to how the STM32F4 handles lazy page allocation because this error goes away if there is a delay >5 seconds placed before getting the values. This error plus the unnecessarily complicated logic in their drivers is one of the major reasons it would be worth writing our own driver file, but more on this later.

After setting the correct chip configuration the AD values are needed to be updated every loop. This is done by the ADBMS_UpdateValues function in the adbms_update_values.c/h file (Figure 25).

```
void ADBMS_UpdateValues()
{
    // get all status registers
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDSTATA, Status, A);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDSTATB, Status, B);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDSTATC, Status, C);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDSTATD, Status, D);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDSTATE, Status, E);
    //printStatus(NUM_CHIPS, &ADBMS_ICs[0], Status, ALL_GRP);

    // get voltages from ADBMS
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDCVA, Cell, A);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDCVB, Cell, B);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDCVC, Cell, C);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDCVD, Cell, D);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDCVE, Cell, E);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDCVF, Cell, F);

    // get temps from ADBMS
    adBms6830_Adax(AUX_OW_OFF, PUP_DOWN, AUX_CH_TO_CONVERT); // turn on cell aux conversion,
    Delay_ms(1); // ADCs are updated at their conversion rate is 1ms
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDAUXA, Aux, A);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDAUXB, Aux, B);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDAUXC, Aux, C);
    adBmsReadData(NUM_CHIPS, &ADBMS_ICs[0], RDAUXD, Aux, D);
```

Figure 25: ADBMS_UpdateValues function

This makes heavy use of the adBmsReadData function from the AD provided library. Every group of values i.e. status, c channel measurements, aux, ect. is broken down into 8-bit registers and then grouped as A, B, C, … and each call to read data only gets one register group and places its value into the corresponding location in the ADBMS_ICs array. It is worth noting that NUM_CHIPS tells this function how many chips are present in the daisychain and thus how many values need to be populated in the array. The other thing to note is the auxiliary channels need to be set every time before reading because there is no continuous update for them.

Once the AD data is updated in the ADBMS_ICs array it needs to be processed from raw values to voltages. Raw values of type int received from the daughterboards are converted to a voltage of type float according to the following mapping that comes from the 16-bit ADCs having a LSB of 150 uV.

$$Voltage = (raw\ value + 10000) * 0.000150$$

In the calculate values step (done by the ADBMS_CalculateValues function) these raw values are turned to voltages and temperatures and are placed in the high level struct along with the max, min and average values. The voltages are done according to the equation above. The temperatures are a little more complicated. The aux measures the voltage across the thermistors. Knowing the voltage across the thermistor, the voltage divider reference voltage, and the value of the other resistor of 10k Ohms, the resistance of the thermistor can be calculated according to:

$$R_{thermistor} = \frac{V_{thermistor} R_{series}}{V_{ref} - V_{thermistor}}$$

This resistance can then be mapped to temperature according to the thermistor datasheet. In our case, the thermistors are NTC thermistors with a B value of 3435, T_0 of 298.15 kelvin, and a resistance of 10k Ohms at T_0 and can be mapped according to the standard NTC thermistor equation (Figure 26).

$$T = \frac{B}{ln\left(\frac{R_{thermistor}}{R_0 \times e^{\frac{-B}{T_0}}}\right)}$$

Figure 26: NTC resistance to temperature equation

Conveniently, open wire detection for the thermistors can be done at the calculate AD values step. Open wire detection will be described in more detail later, but is a check to see if a signal to the board is disconnected via fusing or otherwise. In this case if a thermistor is disconnected there is no voltage divider, the 10k Ohm on-board resistor just becomes a pull-up, pulling the ADC to the reference voltage. Exploiting this, if the measured temperature voltage equal to the

reference voltage (within a threshold of 100mV to be safe) the temperature open wire flag can be set.

When talking to ADBMS6830B chips, if the connection is broken (e.g. disconnecting the comms line) or the chip is otherwise not responding, the ADBMS_UpdateValues() will not hang or otherwise show any indication that it is not making a connection. What will happen is it will read all 0s. Luckily all 0s maps the voltage that is read to 1.49V. This results in an undervoltage error and puts the BMS in an internal fault state. While this isn't the most ideal behaviour, it is still viable to be used in the vehicle as a communication error will result in a fault. However, there is a packet error check (PEC) that is just a cyclic redundancy check (CRC) at the end of every message. When communication is broken or otherwise incorrect the PEC flag is raised with an error value of 1. Currently this is not being utilized due to all 0s results in a fault and keeping with the MVP theme, but this could be used to re-request packets if they fail and raise a fault flag only if enough fail in a row. Doing this would provide better reliability and resiliency especially when operating in a high EMI environment such as the daughterboards sitting on top of a segment where individual messages could easily fail. It is definitely worth coming back to how communication errors are handled.

After updating and calculating the AD voltage and temperature values internal faults need to be set. The internal faults are overvoltage fault, undervoltage fault, over temperature fault, and open wire fault. Note that the temperature open wire is already handled as described above and under temperature can be neglected as it should be impossible to achieve. This is due to the cells tolerating up to -40 degrees C and we would not drive in conditions anywhere near that.

To do open wire detection or OWC for the cells, the internal OWC switches need to be set. Because of differential capacitor networks on the c channel and s channel voltages (Figure 3 & 4), a capacitor divider is formed when a signal line is disconnected resulting in the ADC still reading a reasonable value even if no cell is connected. The purpose of the open wire check is to check if the cell is actually connected to the daughterboard. An individual cell could become disconnected from the mainboard due to a number of reasons, a bad crimp, poor soldering, ect. The most important reason, though, is the cells are connected to the sense PCB via a wire bond

rated to fuse at 1A. The open wire check is really checking to see if this bond is ever damaged or fused. In order to do this check, internal switches with series resistance are close across the differential signal (Figure 27). If one of the cells is not present, this switch will discharge the differential capacitor and the ADC will read 0V. This check will create a voltage divider with the filter resistors and thus the voltages read during this check can be thought of as binary with 0 meaning open wire and anything but 0 meaning no open wire. These switches are grouped into odd and even groups and one group is switched and measured then the other. This is because if adjacent switches are switched, then they will just create a voltage divider similar to the capacitors and the ADCs will read a plausible value. Because the values collected by OWC are thrown out (assuming no faults), OWC was opted to be done on the s channels as to leave the c channels untouched and still reading voltages. When cell balancing is turned on, the internal balancing FET shorts the open wire switch. This results in cell balancing being turned off when open wire detection is being checked. While not the most desirable behaviour, cell balancing only occurs when charging, and thus can take slightly longer with no real downside. Also the cells we use have a fairly tight tolerance and should need less cell balancing as a result.

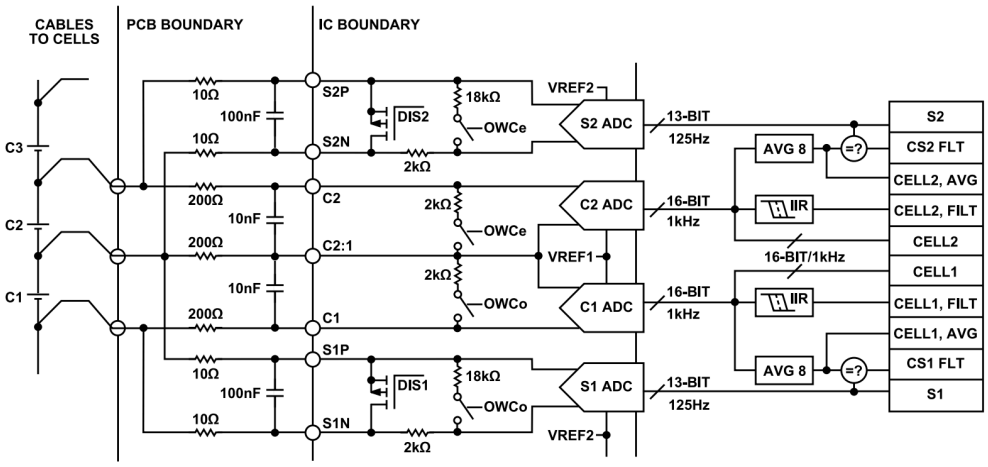

Figure 27: Open Wire Detection Block Diagram

As described above, open wire check from a software perspective should be very straightforward: close the even OWC switches and set a fault if a 0 is read, then open them and close the odd OWC switches and set a fault if a 0 is read, then open them. The chip provides 1 bit in memory that can be set to close the odd switches and one bit for the even switches. Their library drivers mimic this functionality by setting the OWC parameter in the adBms6830_Adcv

or adBms6830_Adsv functions to be 1 for even and 2 or odd (0 means no OWC). As mentioned previously, the idea was to do this on the s channel and thus could leave the c channel alone. When experimenting with this, it was required to set the c channel at the beginning of every check to have redundant measurement off. Then we could set the s channel open wire switches and checks as described above. Finally, at the end of the check it was required to re-configure the configuration of the chip i.e. WRCFGA register and then turn on the c channel redundant measurement. Furthermore, after every configuration a delay of 8ms is required or the chip freezes and will only provide incorrect values until reset by a poweroff. This is quite strange behaviour and worth looking into why this happens and if there is a way to not have the chip config and c channel config writes. More will be said in the High level section, but the purpose of the code written this quarter was to develop a MVP as quickly as possible, then come back and polish areas where sacrifices were made. This is one of those areas. Definitely worth coming back to. It is also worth noting that OWC does not have to be checked as frequently as updating the rest of the AD values. The voltages and temperatures are constantly needed to be measured for the safety and state of charge estimation of the pack. The open wire detection does not need to be updated as frequently because if there is an open wire the car will immediately turn off. Also an open wire is not as dynamic as the voltages and temperatures and thus requires a lower frequency. Where a lower frequency might mean once every 10 seconds (if not longer). How OWC is performed and when it happens warrants further analysis to be done next quarter.

The last function of the low level code is cell balancing. Unlike the rest of the low level functionality, this only happens when changing and thus will never be turned on while driving. Due to manufacturing tolerances and chemistries in cells, groups of series cells will naturally unbalance when charging or discharging. To fix this they need to be balanced hence the name cell balancing. This can be done in one of two ways, active and passive. Active is when energy from higher charged cells are used to charge lower cells i.e. redistributing charge from higher charged cells to lower ones. Passive is done by dissipating charge from higher charged cells to match the charge of lower level cells through a resistor. This effectively bleeds the charge away as heat. While active balancing has obvious benefits, in our case, the complexity far outweighs the benefits. Especially due to the fact that the cells we use have a fairly tight tolerance and thus will not become very unbalanced after just one discharge. Balancing is only done when charging

because we do passive balancing. As passive balancing essentially wastes the energy as heat it is harmful to do when we are trying to squeeze the most out of our pack while racing. On a similar note, balancing is necessary to do while changing because charging has to conclude when the first cell reaches its max voltage. This means if the cells are not balanced we are leaving potential charge on the table. As shown in Figure 27, on the s channels are dissipation resistors and an internal PMOS that can be closed to enable cell balancing.

Cell balancing can be enabled in two ways on the ADBMS6830B chips: continuous or PWM. Exactly as it sounds continuous, continuously closes the PMOS while PWM will close the PMOS according to the PWM duty cycle. For our use case, continuous works perfectly fine and is much simpler to enable, simply writing the specific bit high will turn on the switch. This chip also supports discharging in sleep mode when in PWM with a discharge timeout set. While this is a super cool feature, it is quite complex to implement, as we only balance when charging this would not be useful, and it could potentially get us in trouble by draining the pack if enabled incorrectly. There are more harms to balancing in sleep mode (extended balancing state in the datasheet) but not worth getting into here. While charging, balancing will be enabled and every cell above a threshold above the minimum cell will be turned on (Figure 28), and it is then turned off after charging is concluded or when open wire check is performed due to reasons mentioned above. Note the dcc is a 16-bit int that corresponds to the 16 different s channel cell balancing switches. To turn on a specific channel that bit is set to 1, thus to turn off cell balancing dcc is set to 0.

```
void cellBalanceOn(float min_voltage)
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
    for (int i = 0; i < NUM_CHIPS; i++)
    {
        uint16_t dcc = 0;
        for (int j = 0; j < NUM_VOLTAGES_CHIP; j++)
        {
            if (ADBMS_getVoltage(ADBMS_ICs[i].cell.c_codes[j]) - min_voltage > CB_THRESHOLD)
            {
                dcc |= 1 << j;
            }
        }
        ADBMS_ICs[i].tx_cfgb.dcc = dcc;
        adBmsWriteData(NUM_CHIPS, ADBMS_ICs, WRCFGB, Config, B);
    }
}       jasonlin1222, 2 weeks ago • cell balance
```

Figure 28: Cell Balancing Function

When we received the sponsorship from Analog Devices for this chipset they gave us the dev boards for the ADBMS6830B and ADBMS6822. The ADBMS6822 was designed in such a way that it can plug into the top of a Nucleo. In theory this is very beneficial for debugging the designed mainboard and daughterboards as their functionality can be compared to the dev boards. It also is helpful for writing software as there is professionally designed hardware that can be used to validate the functionality of certain firmware. However, the ADBMS6822 and the Nucleo were not making a proper connection. It is unclear if the issue is with the Nucleo or the ADBMS6822 dev board, but they would communicate properly only when pressure was put on them or they were flipped upside down. This was very difficult to debug as the dev boards working and our code having the error was assumed until painfully proven otherwise after hooking the dev boards up to a logic analyzer. Fortunately, the daughterboard and mainboard were designed correctly on the first try and they were able to be used to test and validate the firmware as it was being written.

As mentioned throughout this section, the library provided by Analog Devices was used to communicate with the ADBMS6830B chips. It would be an interesting exercise to write the driver files for this chip ourselves. At a high level, the driver files are just reading and writing to specific addresses in memory of the ADBMS6830B chips. AD uses the HAL SPI driver from STM32, so the only thing their files are doing is designating which registers/addresses to write to and read from. The addresses are described in the datasheet, and is no different from writing any other driver for a peripheral other than just the sheer amount being accessed. Their library, as mentioned above, is also quite convoluted and more complicated than it has to be resulting in undesirable behaviour when not using global variables. Because of this, it would be both a good exercise and worthwhile to write our own driver files for this chip.

4.1.A List of Improvements

- Changing both the implementation of open wire check and how often it gets called as described above
- As cell balancing has to be turned off when performing open wire check, turning it on immediately after finishing OWC would increase the uptime.

- Using the PEC to resend failed packets and only raising a communication internal fault after a certain amount fails in a row, as described above.
- S channels can be used to perform a voltage measurement when not cell balancing or performing open wire check. While these channels provide more noisy values, they can be compared against their c channel counterparts as a redundant measurement. The datasheet further describes this.
- This is not mentioned, but there is an configurable internal IIR or FIR hardware filter present on the c channels. It is worth looking into how to enable this and validating if filtering provides more accurate values.
- In a similar notion as the hardware filter present on the ADBMS6830B chips, it would be worth looking into having a software filter for the AD values.

4.2 NFR BMS (High Level)

The high level code focuses on using the AD values collected by the low level section and collecting its own values (both through CAN and GPIOs) to act as the brains of the accumulator including communication with the rest of the car, defining the accumulator states, and controlling the isolation relays.

Before diving into how the high level section functions, it is worth mentioning two things. First, a lot of it is going to be changed next quarter as we move to an RTOS, more on this in section 4.2.A. Second, This section of the code should not exist. At least at the same complexity it currently does. This point will be elaborated on in section 4.2.C.

Instead of describing the direct implementation as the previous section, this will go over the logic present and a verbal description of how it was implemented. This is because the logic is what is important here and the implementation directly follows from the logic. Also the implementation is going to be changed quite drastically as we move to implementing an RTOS next quarter.

The functionality of the BMS can be broken down into initialization and 3 timer groups. A timer group is just a way of calling a function only after the designated time has passed based on the internal clock of the STM32. The reason for doing this is that without a timer group the STM would continually process these loops quicker than is required and having a timer group allows functions to be called at different rates. The 3 timer groups are the mainboard loop, drive CAN loop, and data CAN loop.

Initialization purely consists of hardware initializations of the GPIOs, ADCs, SPI, and CAN buses and setting the configurations of the daughterboards via writing to the ADBMS6830B chips as described in section 4.1.

The mainboard loop can be thought of as the loop that performs the functionality of the BMS. It calls the low level functions (AD update, calculate and process faults), reads current, reads external values, and updates the finite state machine context in the update values steps (Figure 29). It then evaluates faults in the check faults step. Finally, it then evaluates the finite state machine.

```
void UpdateValues()
{
    // ADBMS values
    ADBMS_UpdateValues(&mainboard.adbms);
    ADBMS_CalculateValues(&mainboard.adbms);
    UpdateADInternalFault(&mainboard.adbms);
    //ADBMS_Print_Vals(&mainboard.adbms);

    // update STM32 Pin values
    // reads: shutdown_contactors, IMD_Status, 6822_State
    mainboard.shutdown_present = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1);      // shutdown status
    mainboard.imd_status = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8);            // IMD_Status
    mainboard.comms_6822_state = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15);     // 6822_State
    mainboard.charger_pin = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_7);           // Charger_Pin

    // update fsm context
    ((fsm_context_t *)mainboard.fsm->context)->total_pack_voltage = mainboard.adbms.total_v;
    ((fsm_context_t *)mainboard.fsm->context)->charger_pin = mainboard.charger_pin;
    ((fsm_context_t *)mainboard.fsm->context)->min_voltage = mainboard.adbms.min_v;
    ((fsm_context_t *)mainboard.fsm->context)->sd_contactors_present = mainboard.shutdown_present;

    // get current
    mainboard.current = getCurrent(mainboard.hadc) - mainboard.current_offset;
    mainboard.overcurrent_fault = mainboard.current > OVERCURRENT;

    // get fsm state
    mainboard.fsm_state = getCurrentState(mainboard.fsm);
}
```

Figure 29: Update Values function

The current is gathered by reading the ADC voltage, and mapping that to current based on the hardware described in section 3.2. It takes the raw 12-bit value read from the ADC and maps it to voltage by dividing by 2^12 and multiplying by 3.3V as it is a 3.3V ADC. That voltage is then mapped to current by dividing by the gain and shunt resistance.

The check faults functions check for both internal and external faults. An internal fault is defined as a BMS fault (overvoltage, undervoltage, over temperature, open wire, and overcurrent) and has to be latching (i.e. if ever pulled high, it has to stay high) due to FSAE rules. The internal faults are what drive the BMS state, where no faults present have a healthy state of 1 and if there are any faults the state is 0. External faults are any external condition that would stop the BMS from closing the isolation relays and are no shutdown circuit present (the line that drives the relays) or a timeout from either the ECU (electric control unit). A timeout is defined as going over a set time without receiving a CAN message from a particular controller. This is quite easy to check for as the time of the most recent message can be stored and if the time between then and the current time is ever greater than a threshold, a timeout is set. The idea is if any of these controllers stop communicating the isolation relays should be open for safety.

Evaluating the finite state machine (FSM) is remarkably straightforward due to the lightweight FSM library by Evan Bertis-Sample. Instead of evaluating the stats of the FSM through a case-switch this library breaks it down into: on enters called when entering a state, on update call when staying at a state, on exits called when exiting a state, and predicate functions that dictate when to move between states.

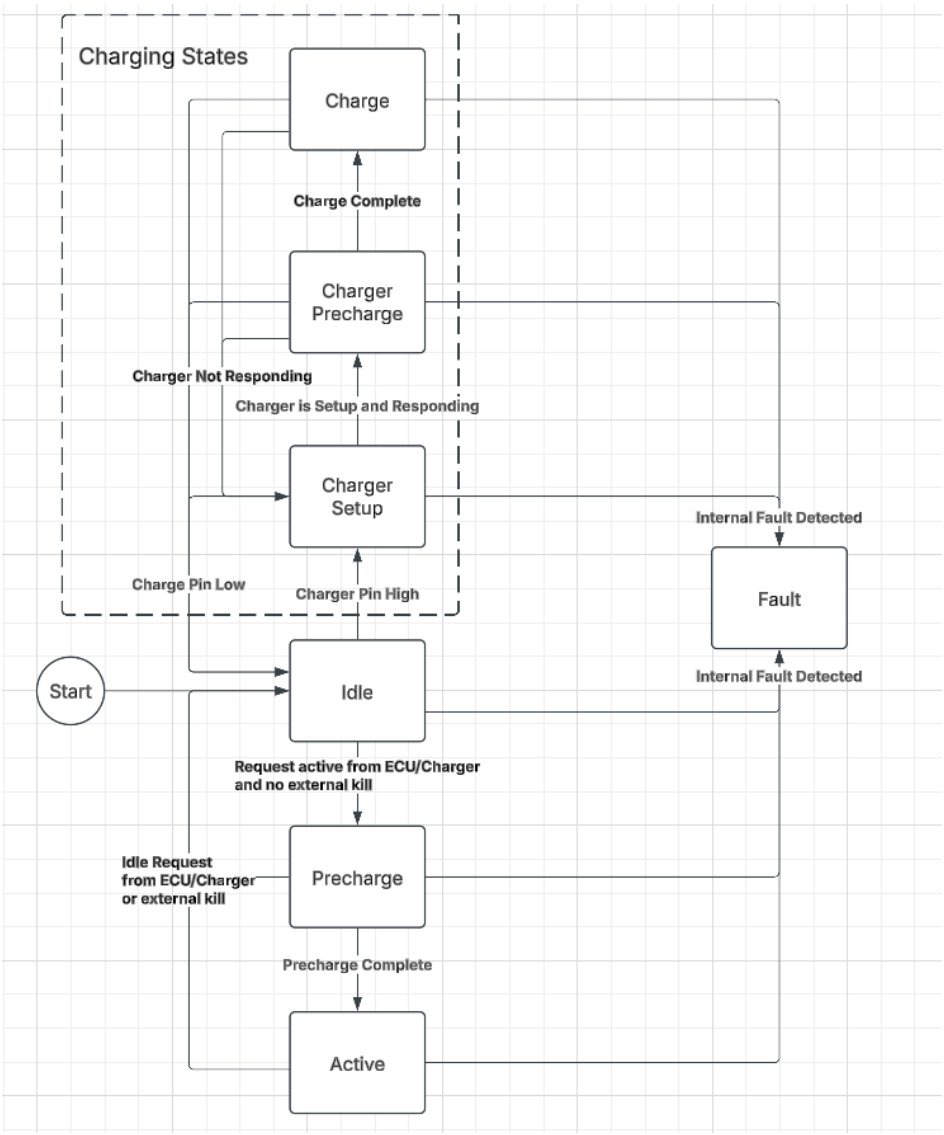

Figure 30: FSM Diagram

There are 7 states within the BMS: Idle, Precharge, Active, Fault, and the charging states of charger setup, charger precharge, and charge. Idle is when there are no internal faults, but the isolation relays are open. Idle is switched to Precharge when there are no external faults and there is a command to close contactors from the ECU. Note the ECU sends the command it expects BMS to be in (opened or closed relays) at a frequency of 10Hz and will continuously send the message not just on a transition. The reason for this is increased safety as the BMS and

ECU constantly talking ensures they are both in the state the other wants them to be in. In precharge, the Positive and Precharge relays are closed allowing the tractive system outside of the accumulator and the DC-link cap to be charged. Precharge is switched to Active once the DC-link cap is fully charged designated by the inverter voltage. Because there are equivalent series resistances between TS+ and TS-, there is a voltage divider between the 6k Ohm precharge resistor and the voltage present across the DC-link cap. For this reason the system cannot precharge up to the full voltage of the accumulator. Knowing this, FSAE rules dictate precharging only has to occur until the voltage is 90% of the accumulator voltage. So precharge is switched to active when the inverter dc voltage is 90% of the accumulator voltage. Active means closing the negative relay, no longer current limiting the tractive system. Precharge and active can be switched to the idle state if ECU ever says to open the contactors or an external fault is present.

The charging states operate on the same principles, but based on feedback from the charger rather than the ecu and inverter. Going into charging is done by pulling the charging pin high. In the vehicle harness this pin is left floating, but in the charger harness this pin is pulled high. When pulled high, the BMS starts sending a specific CAN message that the charger reads. This message contains the max charging voltage, max allowable charging current, and whether to enable charging or not. The max allowable charging voltage and current are system parameters based on the battery that is being changed and the mains supply being used to power it (as to not blow a mains fuse). The enable charging bit is a 1 to enable charging and a 0 to disable. In charger setup this is set to 0 to disable charging. In nominal operation, the charger sends a CAN message with the voltage present at its terminals, current, and fault status. The fault status will be a 1 (fault detected) until it receives a message from the BMS. Once that fault is cleared, charger setup can be moved into charger precharge (positive and precharge relay closed). Although there is no DC-link cap in the charger, the idea is to limit the tractive system current until the charger completes the handshake by responding with a voltage consistent with the measured tractive system voltage. Once this is checked, charger precharge can be moved to charging. This closes the negative relay removing the current limit and sets the charging bit to 1 to enable charging. If the charger ever stops responding or responds with a fault the state will move back into charger setup. On update while charging, cell balancing is turned on with the described functionality in section 4.1, and cell balancing is turned off when exiting the charging state.

Every state has a potential transition to the fault state if an internal fault is ever detected. Once in the fault state, it is not possible to exit without restarting, even if the fault is cleared implementing the latching functionality.

Outside of the mainboard loop, there are two CAN timers: the Drive CAN line and the Data CAN line. These are on their own timer because the mainboard loop runs at a higher frequency (100 Hz) compared to the CAN lines that run at 10 Hz and 1 Hz respectively. The Drive CAN

sends all the important variables that are needed to properly inform the rest of the car of the state of the accumulator. This includes: battery voltage, battery temperature, battery current, every fault status (both internal and external), the BMS state, the IMD state, max cell temperature, min cell temperature, max cell voltage, min cell voltage, and estimated battery state of charge. The Data CAN sends out every measured cell voltage and temperature.

4.2.A RTOS

Using the timer libraries works fine, but it is quite sloppy as there can be blocking functions and things could operate at even more specific times to squeeze better performance out of the BMS. In order to achieve this, the code will be refactored to run off of a Real Time Operating System (RTOS), FreeRTOS. This allows for specific tasks to be scheduled and happen at specific times. During this refactoring, the code architecture will be changed to reflect the addition of an RTOS, changing the implementation of the high level code. While the logic of how to step through states will mostly stay the same, how and when things get called will change. Furthermore, the charger was thrown in at the end after writing most of the code. Because of this, it is not integrated very well into the architecture, another change to come back to when refactoring.

A large benefit of moving to an RTOS is direct memory access, or DMA, for AD update values. Because a lot of time is just spent requesting and waiting for the raw voltage and temperature values from the daughterboards, this can be done in the background and whenever new values are received they can be directly placed into memory via DMA. This frees up a lot of time while also getting results as soon as they are available. A win-win.

4.2.B State of Charge Estimation

There are very interesting algorithms to estimate the state of charge (SOC). To keep a long story short there is no good way to measure the SOC because the voltage of the cells drop under load (and have hysteresis). The driver still needs to know how much of the battery is left to inform how hard to drive, so the SOC needs to be estimated. To start a rudimentary Coulomb Counting method will be implemented to track the current discharged from/charged into cells while driving to estimate the amount of charge left. There are a host of issues with this algorithm, and very interesting algorithms that are proposed to get more accurate results. For more information, look at this presentation on state of charge estimation algorithms.

4.2.C Ideal NRF BMS Behaviour

Northwestern Formula Racing has imposed this external functionality on our BMS to act as the “brains of the accumulator”. This is very unnecessary and actively harmful. There is already a “brain for the vehicle” being the ECU. Having the BMS and ECU both act as brains means their interactions have to be handshakes causing complicated interactions and confusion. Having this functionality imposed on the BMS also means that its main goal of measuring voltages and temperatures gets overshadowed by the FSM behavior and interactions with the rest of the car.

In an ideal world there would only be one brain, the ECU, and the BMS can serve purely as a slave. Simply getting the AD values, calculating faults and SOC, and sending important information to the ECU. In return, the ECU can deal with all the different states present in the car and simply tell the BMS what contactors to close at what times over CAN. This simplifies the BMS firmware as almost all of the high level code would go away, and this high level, low level split wouldn't need to exist. This would free up resources on the BMS and allow it to entirely focus on being simply a BMS. The best part is this wouldn't change the hardware at all, and adding the BMS states to the ECU is negligible complexity. Unfortunately it is too late in the design process to make this change and this year's AD BMS had to be a drop in replacement for the BQ BMS resulting in this functionality. In future years I hope that this functionality is separated from the BMS.

References

[1] Analog Devices, “ADBMS6822 Datasheet and Product Info | Analog Devices.” Accessed: Feb. 06, 2025. [Online]. Available: https://www.analog.com/en/products/adbms6822.html

[2] “ADBMS6830B Datasheet and Product Info | Analog Devices.” Accessed: Mar. 18, 2025. [Online]. Available: https://www.analog.com/en/products/adbms6830b.html

[3] SAE International, “Formula SAE Electric.” Accessed: Feb. 11, 2025. [Online]. Available: https://www.sae.org/site/attend/student-events/formula-sae-electric

[4] SAE International, “Formula SAE Rules 2025.” Aug. 31, 2024. [Online]. Available: https://www.fsaeonline.com/cdsweb/gen/DownloadDocument.aspx?DocumentID=4bafd174-62da-48b6-b016-589385ca5ae6

[5] Molicel, “INR-21700-P45B Datasheet.” [Online]. Available: https://www.molicel.com/wp-content/uploads/INR21700P45B_1.2_Product-Data-Sheet-of-INR-21700-P45B-80109.pdf

[6] Ewert Energy Systems, “Orion BMS 2 Specifications.” [Online]. Available: https://www.orionbms.com/downloads/documents/orionbms2_specifications.pdf

[7] R. Suganya, L. M. I. L. Joseph, and S. Kollem, “Understanding lithium-ion battery management systems in electric vehicles: Environmental and health impacts, comparative study, and future trends: A review,” Results in Engineering, vol. 24, p. 103047, Dec. 2024, doi: 10.1016/j.rineng.2024.103047.