Resistance Racing: Cornell Kiwi Kruiser

Battery Management System

Spring 2018 Technical Reflection

Logan Hernandez Horowitz I
5/18/2018

Table of Contents

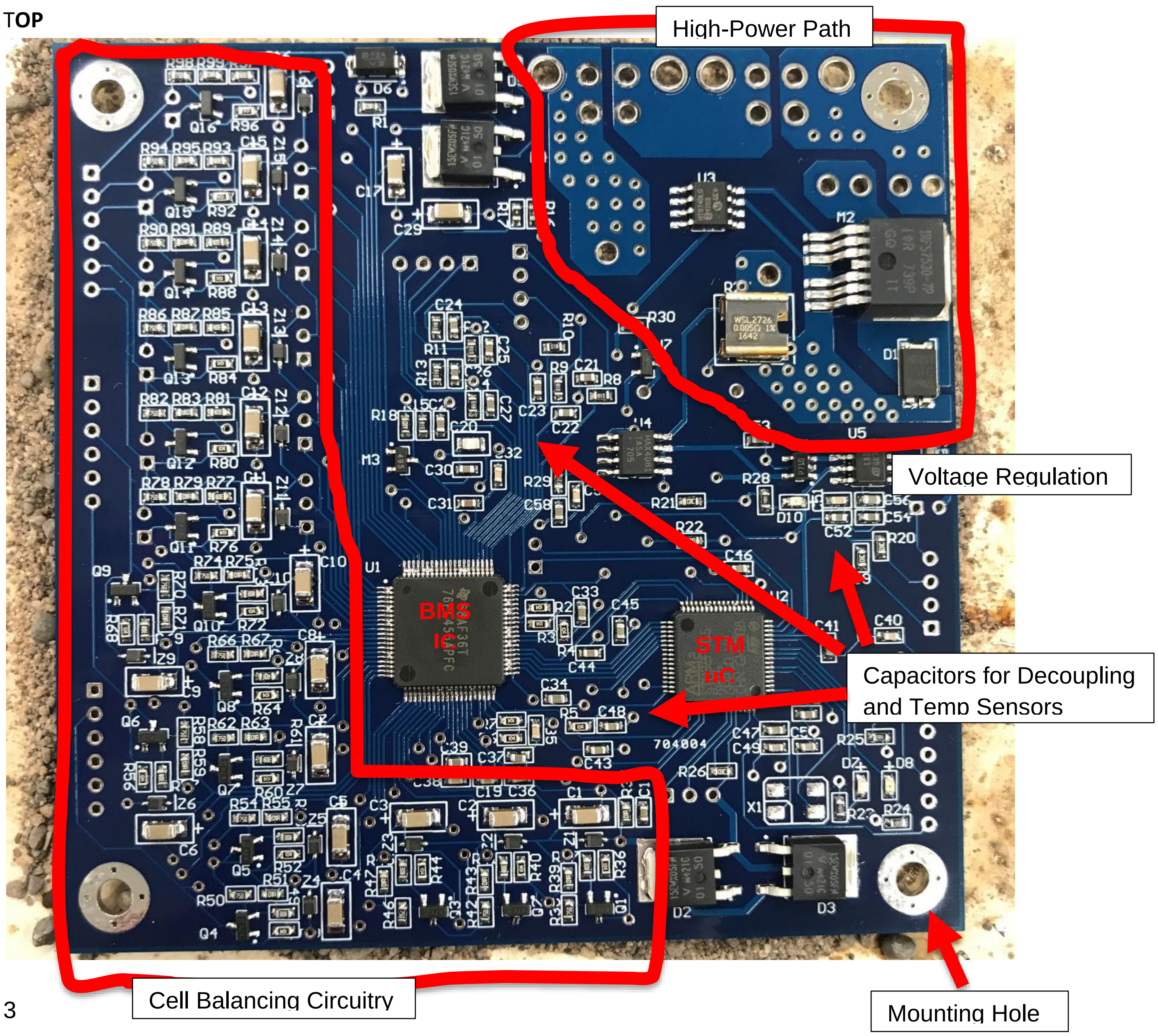
Summary

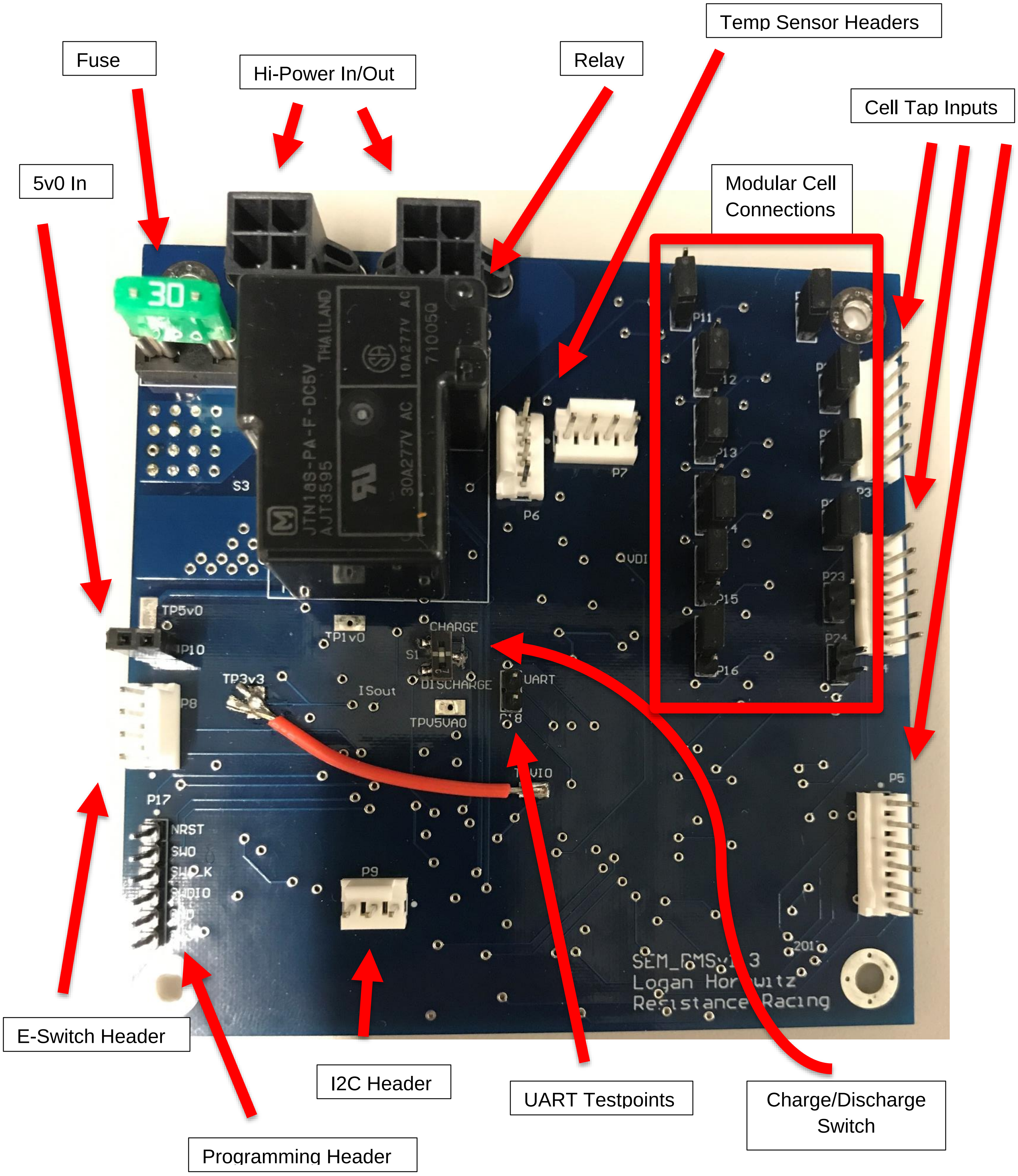
System Description

The battery management system acts as the overall safety monitor for the vehicle. It collects voltage, current, temperature, and fault data and then analyzes it to determine if the vehicle is operating in an unsafe condition. If so, the BMS communicates the information to the data acquisition system and cuts electrical power to the rest of the vehicle. It must ensure that all electrical power is isolated from the front of the vehicle during a fault. If the vehicle is operating safely, it simply sends the data to the DAQ and then checks the vehicle again. The board was intended to monitor current, voltage, temperature, faults from the BMS_IC, and the emergency switches. The final product we used at competition, however, was only able to monitor current and the e-switches.

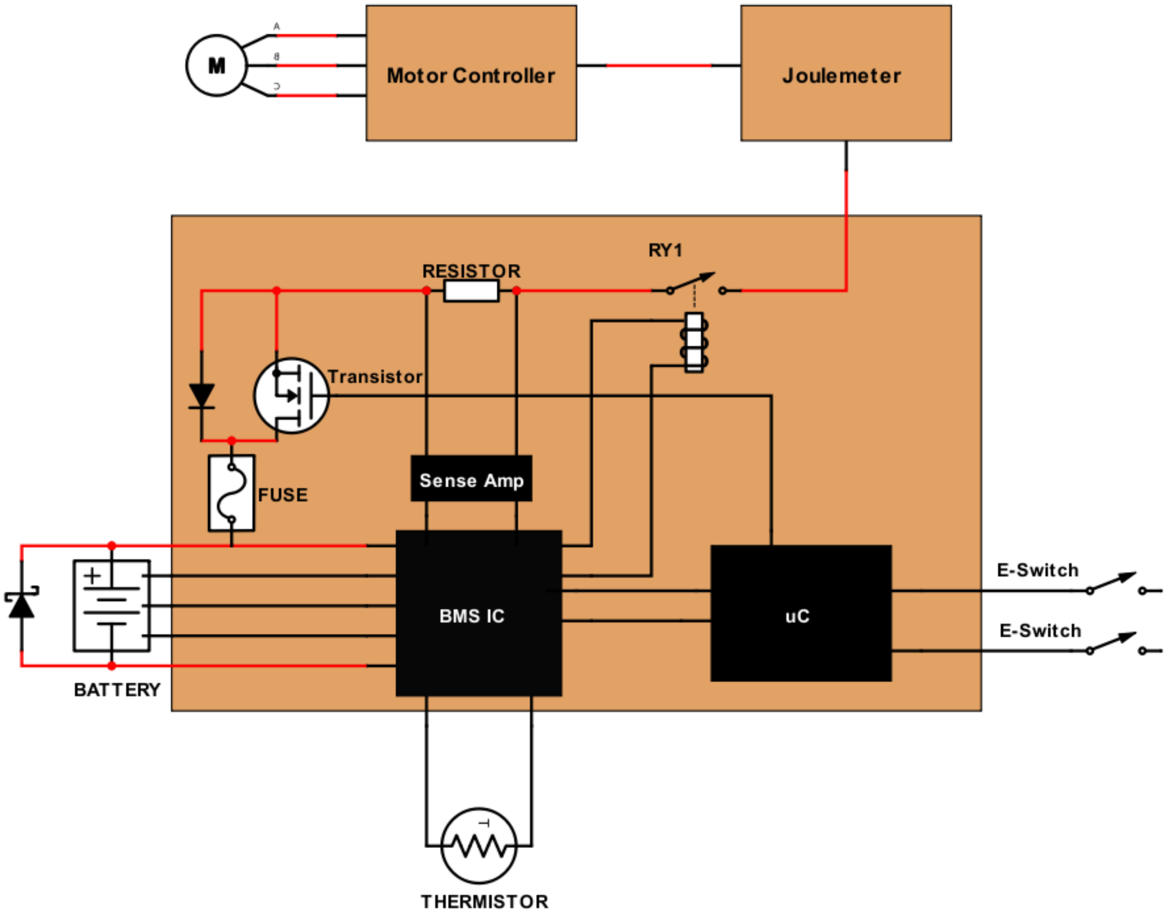
The pictures below shows a completed board with important aspects labelled.

TOP




Fuse
Hi-Power In/Out
Relay
Temp Sensor Headers
Cell Tap Inputs
5v0 In
Modular Cell Connections
E-Switch Header
I2C Header
UART Testpoints
Charge/Discharge Switch
Programming Header
SEM_BMSv1.3
Logan Horowitz
Resistance Racing

High-Level


M
A
B
C
Motor Controller
Joulemeter
RY1
RESISTOR
Transistor
FUSE
Sense Amp
BMS IC
uC
E-Switch
E-Switch
BATTERY
THERMISTOR

Jargon

IC: Integrated Circuit. A device designed for mounting on a PCB or in a breadboard, usually in a small black package, which performs some specific functionality (opamp, microcontroller)

PCB: Printed Circuit Board. A board of copper and fiberglass used for wiring small IC's and passives in a compact manner.

Trace: A connection of copper on a PCB; basically an integrated wire

Cell/Battery: These are sometimes interchangeable; I generally use battery to refer to the battery pack in the vehicle which is actually a series connection of 12 different batteries. I refer to each of these individual batteries as a cell.

Passive Device: Something with a linear IV relationship either directly or by first derivative. (resistor, cap, inductor)

Joulemeter: Device which measures current and voltage to determine the amount of instantaneous power consumption.

MC: Motor Controller. Delivers signals to the motor to control when and how fast it accelerates the vehicle. We are using a brushless DC motor and therefore a BLDC motor controller.

BMS: Battery Management System

DAQ: Data Acquisition System. Collects data and will provide visualization and analysis.

E-Switch: Big red latching emergency switch. We have two mounted on the vehicle per Shell's regulations. Power to the vehicle must be cut when either of these switches is pushed.

FET: Field Effect Transistor. Google stuff or take 3150 to understand what this is.

BJT: Bipolar Junction Transistor. Google stuff or take 3150&5790 to understand what this is.

Major Components

TI BQ76PL455A (BMS IC)

This TI chip is a dedicated battery monitor IC, which means that its job is to read the voltage of all of the cells it is connected to (12 in series in our case) as well as the value of 8 analog inputs (which we connected temperature sensors to). We need* to have this special IC do this because the series voltage of our pack was 44V which would blow up a normal IC. It is wired to all of the cells through a network of passives and transistors so that it can perform passive cell balancing during charging. It is actually not required by Shell, though, so we can choose whether or not to implement this feature next year. It communicates (or was supposed to communicate) with the microcontroller via UART.

Cell Balancing Circuitry

Our car was powered by lithium polymer batteries. This is a dangerous chemistry which is prone to explosive reactions or leaking if the cells are mistreated. For this reason, we need a battery management system to ensure that the voltages are kept within a safe range and the current draws will not cause excessive internal heating and thermal runaway. We also need to charge the cells in such a way that all of the cells become equally charged. This can be done in one of two ways: 1) If a cell's voltage is higher than others, divert the charging current that would flow through it into a resistor and burn that energy as heat, 2) If a cell's voltage is higher than others, divert the charging current from it to a cell which has a lower than average voltage. The former is called passive balancing while the latter is active balancing. We implement passive balancing on this BMS using a FET as a switch to divert the current into a resistor. Half of the board displayed above is covered by this network of resistances and FETs which is controlled by the BMS IC.

STM32F407 (STM uC)

This microcontroller is the brains of the board. We program it to configure functionality and it dictates all outputs. We probably could have gotten away with a cheaper microcontroller because we were not doing anything memory or computationally expensive but it is a good platform to learn about programming ARM in C. If we integrate DAQ and BMS in the future it will become more essential that the microcontroller is high-end. Another benefit to this choice is the plentiful amount of GPIO, ADC, and communication buses.

Relay & Power FET

The BMS must isolate the rest of the vehicle if aforementioned conditions are unsafe. It does this by means of a power MOSFET in series with a relay. The use of both is redundant and simply increases safety. The relay is just a voltage-controlled switch while the power MOSFET is basically a beefier version of ordinary 3150 MOSFETs but with the addition of a body diode which must be accounted for.

Sense Resistor & Sense Amplifier

Monitoring current without dissipating power is a nontrivial task. A common way is to pass that current through a very small resistor and measure the voltage across it. By V=IR, we can compute the current if the resistance is known (not always accurate if the resistor heats substantially, another reason to make it small). Because the resistance is so small, usually on the order of milliohms, we add an amplifier to boost that voltage to something we can read in reliably to an ADC.

Relay Driver and FET Driver

GPIO pins on a microcontroller are designed to function as inputs or outputs, but internally they are connected to 3150-type FETs which have very limited current and voltage limits. Therefore, if one needs to drive a large current, they must use the supply rather than a GPIO pin. In order to still maintain control over that current, a driver stage is used. For example, the coil of our relay requires 161mA which approximately 10x what I would feel comfortable driving out of GPIO (datasheet will give specific tolerances for uC) so instead the GPIO output is connected to a relay driver (BJT) with current gain >100 so that the microcontroller does not need to source that much current but can still control when the relay is latched.

Capacitors Everywhere

Capacitors are used to decouple voltage signals and thereby make them less noisy. One way to think about it is from the standard capacitor equation $Q = CV$. This is equivalent to $V = \frac{1}{C}\int I$. Because the voltage across the capacitor is a function of the integral of current, it cannot change instantaneously. Furthermore, taking the derivative yields $\frac{dV}{dt} = \frac{I}{C}$ assuming constant current. So we see that the larger the capacitor, the more slowly the voltage across it will change. "A capacitor likes to hold the voltage across it" (Similarly "An inductor likes to maintain constant current", which actually causes problems when we want to stop the current flow). If we want the voltage across two things, like power rails, to be steady, we can therefore place a capacitor across them. Another way to think about it is $Z_C = \frac{1}{jwC}$ so any high-frequency fluctuation will be shunted by the capacitor. In practice, they are always used on voltage rails and usually placed on analog inputs.

* I don't plan on having one next year. 44V is relative to ground. If you only read the voltage with respect to the cell below it, you would see 3.7V but then you have to find a way to make that 3.7V referenced to ground. We will figure out the level-shifting and isolation ourselves so everything is self-made and understood completely

Initial Implementation

Individual System

There are essentially two distinct portions of the board which operate off of different power supplies.

<u>BMS IC</u>

The BMS IC section consisted of the TI BQ76PL455A battery monitor IC along with the supporting cell-balancing circuitry, and analog circuitry to filter the temperature sensor inputs. The chip runs a lot of internal power levels…

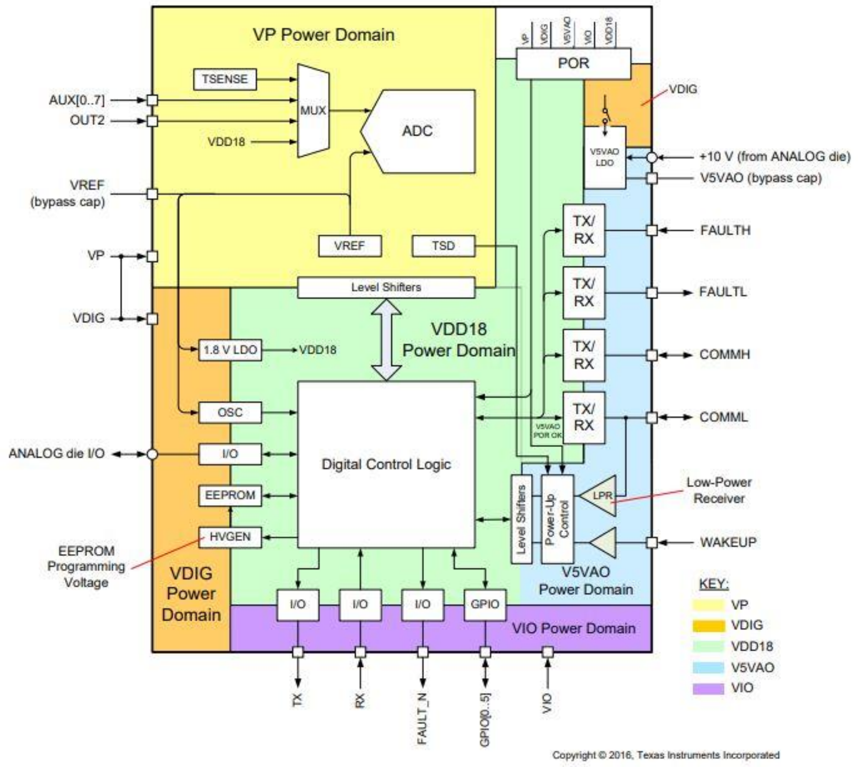


Figure 14. Digital Die Power Domains

…but all of them except for the VIO power domain are derived directly from the battery cells it is connected to. Because the entire chip is powered by the main stack of batteries, the only way to power and test it is by connecting it to the entire pack; this delayed our testing for a while as we had to wire all of the cells. The VIO portion is for communication and it is powered externally.

<u>STM uC</u>

The STM microcontroller is the brain of the board and it is powered off of a 3.3V supply regulated down from the 5v0 input to the board. It is connected to the BMS IC by UART.

As I had mistakenly forgotten to connect VIO to the 3.3V on the board, the BMS IC was entirely isolated from the STM portion of the board as long as I did not use those UART pins. Therefore, I began all my testing with the STM part. It worked perfectly on first try. I was able to program the microcontroller, light LEDs, and even verify that the FET and relay drivers worked perfectly with their corresponding power MOSFET and power relay by latching that shut and reading 0ohm across them.

The BMS IC portion did not work, however. I populated 3 boards and they exhibited the following characteristics:

1. Would overheat uncontrollably if the cells were connected. The test points which should have been ~5v0 were ~10V.
2. Would overheat uncontrollably if all 12 cells were connected, but not if 10 were connected.
3. Mysterious solder short between GND and 3v3

Both General Yao and I scoured the BMS IC datasheet and my schematics but could not find any error which would cause this although it most certainly was caused by our attempt to short down the top cells to however many we were using (BMS IC supports 16 and we used 12).

Full System Integration

We could not establish I2C communication between the BMS and the RasPi because we were attempting to use a confusing, bloated library rather than writing the code ourselves. In the future, all communication code will be handwritten in C utilizing the registers explained in the datasheet and debugged by the logic analyzer. We were able to hack together a communication system, however, by bitbanging the RasPi with single-ended UART data.

Integrating with the batteries' cell connections was problematic because they were not long enough to reach the board and soldering onto those wires was sketchy and eventually explosive. In the future, the board must be designed to plug immediately into the batteries.

Charging was also an issue because LIPO chargers only go up to 6 cells, while we had 12 in series. In the future, we should design our BMS so that it can remain connected during charging.

Notable Failures

The BMS IC never functioned. As a result, we had very little usable data and very little indication of whether or not the vehicle was operating safely. There were also a number of design errors:

- VIO not connected
- Crystal footprint incorrect
- High-Power relay pads not large enough, holes too big
- Temp sensors floating
- DC-DC board was in the way of programming header so it had to be removed every time we reprogrammed

Areas of Improvement

System Architecture

Three different projects' progress were halted this semester by an IC which did not respond the way its datasheet said it would. Eric's Joulemeter never worked as a result, my BMS got us through tech and that's about all it did, and we never got the seven-segment display working. We could eliminate the possibility of a single malfunctioning IC ruining both our BMS and DAQ functionality if we designed everything ourselves. Specifically, I think we could remove the BMS IC entirely and design our own BMS which handles everything off of the microcontroller. It would be a significant EE design challenge, but a rewarding one which is debuggable rather than a frustrating dead end.

Conclusion

I have a few recommendations which I believe should be implemented this year. The most notable ones are stated above:

1) Design the entire BMS without a dedicated battery monitor IC

2) Write all BMS code by hand in C

Both of these design decisions follow the principle of doing everything from the ground-up ourselves. This is tedious at first but ultimately a much better learning experience which saves tons of time fruitlessly debugging systems designed by other people.

Additionally, in terms of specific BMS design decisions I would:

- Attempt to keep passive cell balancing, but again with our own simplified circuit which we model and verify in spice
- Keep both a relay and a power mosfet as power control mechanisms. It is safer to have both and one can always be removed/shorted if necessary
- Add a low-power fuse (All circuitry should be fused)
- Keep using both the Molex and the Amp connectors
- Keep the switch to choose between charging vs. discharging (but make it bigger)
- Add a snubber circuit around the power relay
- Use the same sense resistor, sense amplifier combo for current measurement
- Design cell connections to be battery-compatible or modifiable
- Design cell connections to be charger compatible