BMS Notes

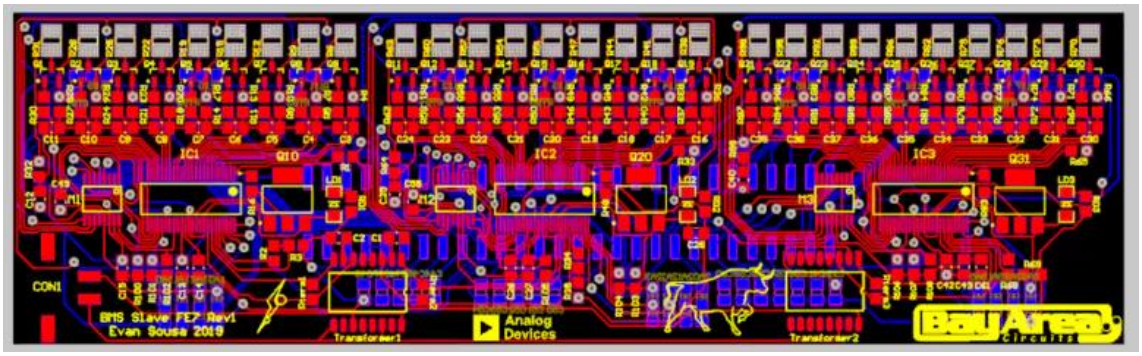


Figure A: Old BMS node board

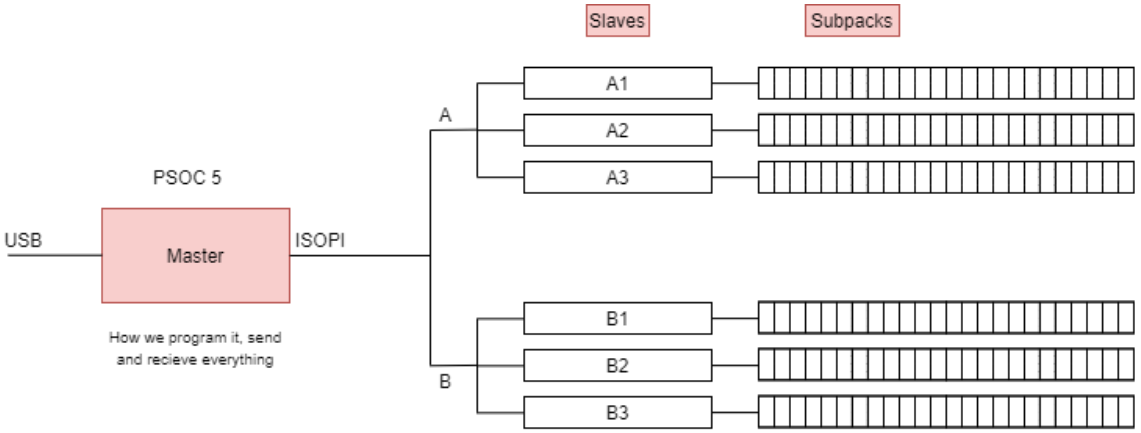


Figure B: Schematic of the old BMS

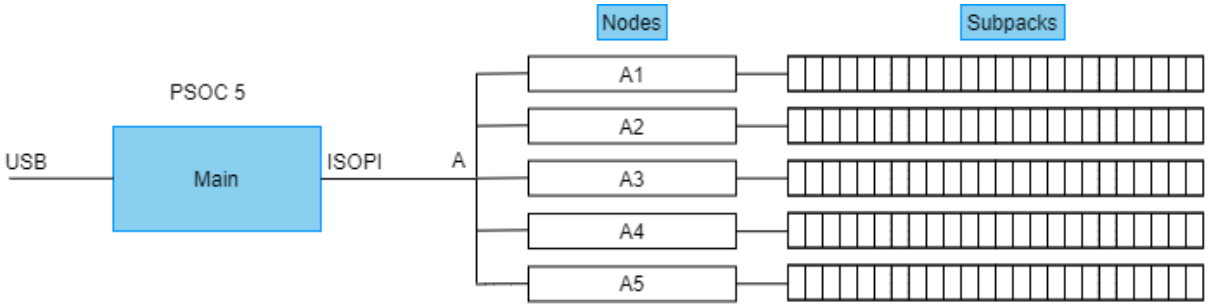


Figure C: Updated BMS schematic with 5 subpacks

Nodes

- Send commands here to get the voltages of the batteries
- Each has 28 voltages and 8 temperatures
- Each node has 3 LTC6811 chips that have ADCs which are responsible for sending and receiving voltages and temperatures
  - Each chip has a physical hardware address that is four bits in length
  - Voltages are monitored by the LTCs

- Temperatures are monitored by an assortment of MUXs

Subpacks

- The batteries, each subpack consists of 28 cells that each have a voltage that must be monitored

With changes to the motor controller we are using, some modifications had to be made to the BMS which can be seen in the table below:

Old Version	New Version
• 28 voltages sensors per subpack • 6 subpacks • 15 thermistors per subpack • 2 SPI buses, A and B	• 28 voltages series connections per subpack • 5 subpacks • 15 thermistors per subpack

Along with these changes, there is the possibility that we decrease the number of SPI buses to one due to the architecture of the boards. Since each node has three LTC chips with four-bit addresses, that means with the new modifications we would only have 15 addresses to account for instead of the previous 18. Therefore, we would not need to make another bus in order to accommodate the BMS nodes, refer to Figure C for a diagram of what it would look like. If the SPI is not too busy, this is a possibility to pursue. The code uploaded to GitHub changes it to a one bus system but you can refer to the previous BMS code to compare the changes made.

The following section goes over some of the changes made to the BMS code if any specifications change in the future.

BMS code

cell_interface.h

```
45  #define N_OF_CELL (168u)
46  #define N_OF_TEMP (90u) // Monitoring the CELLS
47  #define N_OF_TEMP_BOARD (54u) // Number of thermistors monitoring the BOARD
48
49  #define N_OF_SUBPACK (6u)
50  #define N_OF_BUSSES (2u)
```

- Change these unsigned number to the new amount you currently have
  - N_OF_CELL = # of subpacks * # voltage sensors per subpack
  - N_OF_TEMP = # of subpacks * # thermistors per subpack
  - N_OF_TEMP_BOARD = # of board thermistors per node * # of nodes in total
    - total number of thermistors on the BMS node board. There are 3 thermistors per LTC-6811's set of discharge resistors; 3 LTCs per NODE; 6 Nodes total. 9 board thermistors * 6 = 54
  - N_OF_SUBPACKS = # of subpacks

- N_OF_BUSSES = # of busses

```
52  // Stores cell data
53  uint16_t cell_codes[IC_PER_BUS * N_OF_BUSSES][12];
54  uint16_t cell_codes_lower[IC_PER_BUS][12];
55  uint16_t cell_codes_higher[IC_PER_BUS][12];
```

- What does the 12 refer to?
  - 12 is the highest number we can put in a SPI which only accepts 7-bit addresses
  - Communication protocol, a language for microprocessors
  - Most likely won't change
  - Reference (via Conrad): https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

```
85  #define NUM_LTS (18u) // TO
86  #define TEMPS_ON_BOARD (8u)
87  #define LT_PER_PACK (3u)
```

- NUM_LTS = # of LTC chips per node * # of nodes
- TEMPS_ON_BOARD = # of temperatures per node
  - Can also be found in line 389
- LT_PER_PACK = # of LTC chips per node

```
308 //sort temperatures and return median?
309 float32 get_median_temp(float32 temps[6][24]);
```

- 24 refers to the total number of temperature readings per BMS node board

cell_interface.c

```
167 uint8_t get_cell_volt(){
168
169     LTC68_ClearFIFO(); //spi fifo clearing is good practice
170     int error1 = 0;
171     int error2 = 0;
172
173     CyDelay(1);
174     Select6820_Write(0); // allow conversions to happen on bus 0
175     CyDelay(1);
176     wakeup_sleep(0);
177     CyDelay(1); // Waited more
178     //for (int i = 0; i < 100; i++) {
179
180         LTC6804_adcv();
181         LTC6804_adcv();
```

- Previously this function did some initialization for the two buses, since they are declared with the function *Select6820_Write();* where the argument was the number of the bus we wanted to communicate with (either 0 or 1)
- As a result any part of the code that mentions *Select6820_Write(1);* or *wakeup_sleep(1);* is commented out along with any of the operations happen between them

`lt_addr`

```
262     Select6820_Write(0);
263     wakeup_sleep(0);
264     CyDelay(10); // Waited more
265     for (int i = 0; i < 3; i++) {
266         LTC6804_adow(pup);
267     }
```

- Kept this loop at 3 since I'm assuming there are still 3 LTCs per node

```
363  float32 get_median_temp(float32 temps[6][24]) //not used anywhere?
364  {
365      int num_temps = (TEMPS_ON_BOARD + 16) * 6;
366      float32 local_temps[24 * 6];
367      for(int i = 0; i < 6; i++) {
368          memcpy(local_temps + 24*i, temps[i], 24 * sizeof(float32));
369      }
370
371      float32 start;
372      int j;
373
374      //insertion sort
375      for(int i = 1; i < num_temps; i++) {
376          start = local_temps[i];
377          j = i - 1;
378          while(start < local_temps[j] && j >= 0) {
379              local_temps[j + 1] = local_temps[j];
380              j--;;
381          }
382          local_temps[j + 1] = start;
383      }
384
385      return local_temps[num_temps/2];
386  }
```

- Need to investigate if we're actually using it, if we aren't no point in making the loops generalized
- Supposed to be for debugging purposes

```
462     // the number of lt chips.
463     uint8_t num_lts = 15;
464     uint8_t orig_cfga_data[9][5];
465
466     uint8_t j = 9;
467     uint8_t lt[9];
468
469     for (uint8_t i = 0; i < IC_PER_BUS; i++){ //Bus 0
470         lt[i] = i;
471     }
472     get_cfga_on_init(lt, orig_cfga_data);
473     get_lt_temps(lt, orig_cfga_data);
474
475     for (uint8_t i = 0; i < IC_PER_BUS; i++){ //Bus 1
476         lt[i] = j;
477         j++;
478     }
479     get_cfga_on_init(lt, orig_cfga_data);
480     get_lt_temps(lt, orig_cfga_data);
481
482     // Bad thermistor in pack
483     bat_pack.subpacks[5]->temps[0]->temp_c = (double) 20;
484     check_temp();
485
486
487     return 0;
```

- The number of LTCs will have to be changed if the number of nodes changes, calculate it the same way done earlier
- Also commented the code relating to Bus 1 since it no longer exists

```
1218    uint8_t temp_cfg[IC_PER_BUS * N_OF_BUSSES][6];
```

- 6 here is referring to the number of nodes, make generalized

Data.h

Please refer to the file since it's very short

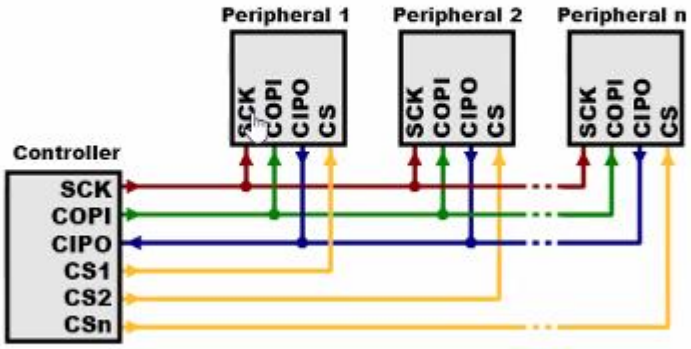
- Going on assume these values remain the same since I believe these values refer to the specifications of the LTC chips

Supplemental Notes:

Multiple peripherals

There are two ways of connecting multiple peripherals to an SPI bus:

1. In general, each peripheral will need a separate CS line. To talk to a particular peripheral, you'll make that peripheral's CS line low and keep the rest of them high (you don't want two peripherals activated at the same time, or they may both try to talk on the same CIPO line resulting in garbled data). Lots of peripherals will require lots of CS lines; if you're running low on outputs, there are binary decoder chips that can multiply your CS outputs.



2. On the other hand, some parts prefer to be daisy-chained together, with the CIPO (output) of one going to the COPI (input) of the next. In this case, a single CS line goes to *all* the peripherals. Once all the data is sent, the CS line is raised, which causes all the chips to be activated simultaneously. This is often used for daisy-chained shift registers and addressable LED drivers.