BMS LV Documentation

Chiara Gentile

May 2022

1 GENERAL DESCRIPTION OF THE BOARD

This board is used to monitor the behaviour of the Low-Voltage Battery. In particular, the Battery Management System is the one that checks constantly the status of the battery in order to understand if it is working in the **Safe Operating Area (SOA)**.
This board includes two blocks:

1. **Master**: it is the one designed by us;
2. **Slave**: it is provided by a sponsor of ours (Podium) and it monitors the voltage and temperature of each cell of the low-voltage battery.

Both boards communicate according to the UART protocol and this communication follows three main steps:

1. **IDLE STATE**: wake-up procedure for powering on the slave device;
2. **INITIALIZATION STATE**: we set the slave so that it can acquire the data and we also set the Baud Rate;
3. **ACQUISITION STATE**: the slave has been set and now the master can send a message to it. Furthermore, the slave this time will be able to send information about voltages and temperatures of each cell.

2 Slave board

This board, as already said, is provided by Podium, one of our sponsors that provides us with slave devices and the high-voltage battery.
To make this board communicate with our BMS LV, we need to read carefully its data-sheet, because there all the steps for the interface are described, such as the commands and the registers to be set in order to instantiate correctly the instructions and the procedure. The useful data-sheets can be found in the drive, BMS LV folder, in the data-sheet section.

3 BMS LV Board description

Talking about the design of the board, we decided to keep the same configurations and circuits of last year. We also kept the same disposition of components, leaving in this way the board mounted on both surfaces.
However, we made some improvements, because we inserted new components and new filters, needed to protect in particular the micro-controller.
In the following lines, now, each block of the board is described.

3.1 POWER SUPPLY

This topic has been already discussed previously. Just make reference to its documentation, where it is possible to understand and to know what this circuit includes, such as the choices we made about the components are explained.

3.2 SLAVE CONNECTION SCHEME

We included in our project on Kicad a sheet dedicated to the slave board, where all the components and circuits we need to make it communicate with the BMS LV are represented. The schematic is shown in Figure 1.

This sheet only includes connectors, that are distinguished according to their function:

1. **J17**: this is a 2-pin Molex connector from where the BMS LV board takes the power signal GND from the car. From here, this signal moves along the car;

2. **J18**: this is connected directly to the negative side of the low-voltage battery (so it also corresponds to GND);

3. **J2**: this is the connector needed to send the information about the voltage readings of the cells from the slave to the master board, so we can simply say that it is important in order to know which is the voltage of each cell;

4. **J15, J2**: this is just used for continuity. When the two inputs are shorted, then a path has been created and the voltage we read is the same as the one at the input of this component;

5. **vsenses**: they are described in the general data-sheet of the slave board. However, these vsenses are its input channels that measure the voltages of individual cells in the range of 1 V to 4.95 V. Channels are used from lowest to highest, with VSENSE0 connected to the (–) terminal of the bottom cell;

6. **U1**: this is used to read the temperatures, that are always read by the slave and then sent to the BMS LV board. The information about the

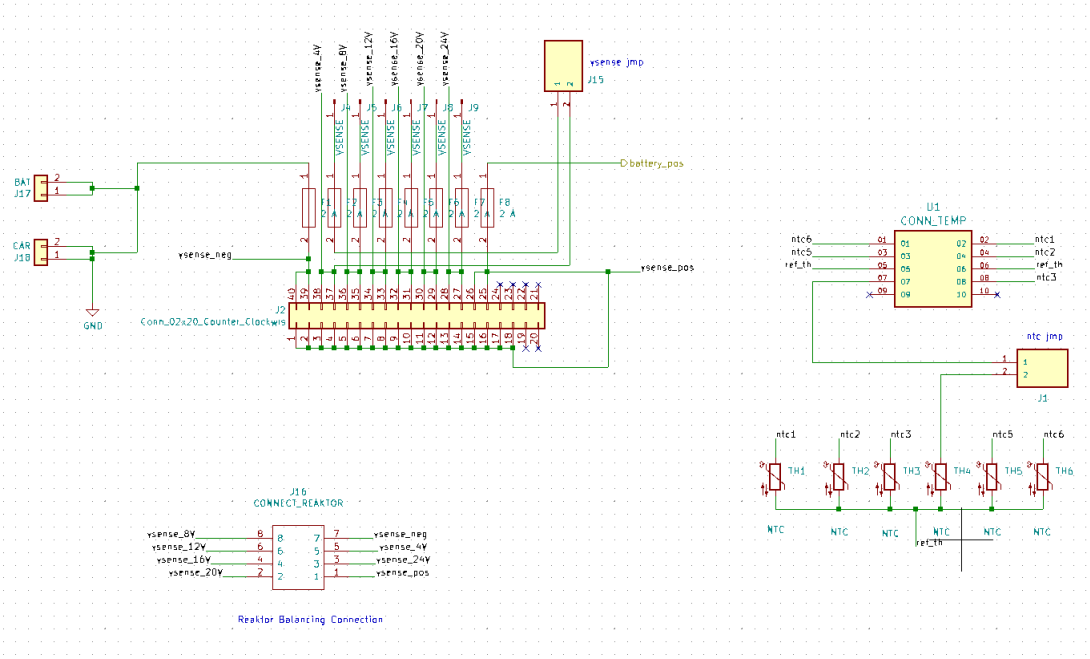


Figure 1: Slave connection scheme

temperature of the cells arrives from the NTCs, sensors that are connected between the low-voltage battery and the BMS LV board, by means of some traces. All this amount of data, collected by the slave, is finally sent to the master board through the appropriate connector J13;

7. **J16**: this connector has been designed in the case we want to balance the cells. However, to do so, we need once again to make reference to the data-sheet of the slave, where it is explained how to do that via firmware.

4 HOW BOARDS ARE MOUNTED

As it can be seen in Figure 2, the slave board is connected on the BMS LV board, that is connected in turn to the LV battery. On the right, those green wires are connected to J14, that is the connector that takes the following signals:

1. CAN H, CAN L: they go to dSpace;
2. CAN H, CAN L: they go to the Steering Wheel;
3. 24 V: it goes to the GLVMS;

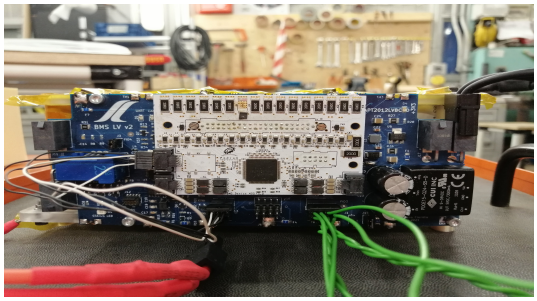


Figure 2: Front view

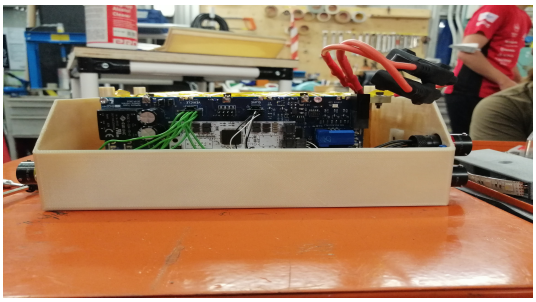


Figure 3: Board inside the box with the low voltage battery pack

4. Relay CMD (LV CMD): signal that goes to the firewall, exits and goes to the Relay Board. This is the negative CMD, sent from the BMS LV and going to the coil of that board, that may correspond to GND or high impedance.

These signals go out from the connector on the left of the box, that is shown in Figure 4, while the power signals connector takes 24 V and GND to the firewall, where they are split and sent to the *relay board* and to the *main hoop*. In Figure 5, these connectors are shown: the one on the top takes 3 GND wires, while the one on the bottom takes 3 24 V wires.
It is important to say that the BMS LV board receives the power supply from the low voltage master switch. Its route, in fact is the following: the power signal 24 V goes out from the power signal connector on the right of the box. It goes to the relay board and to the GLVMS. From there, there is a red wire taking 24 V to the BMS LV and it enters into the signal connector on the left of the box, so that connector both receives inputs and sends outputs.

5 COMMUNICATION WITH THE SLAVE BOARD

This board can monitor and balance from 6 to 16 cells per device. Thanks to this capability, we set 8 channels for the temperature readings and 8 channels for the voltage readings. The board monitors and detects different fault conditions, including over-voltage, under-voltage, over-temperature and communication faults. It works with a 14-bit Successive Approximation Register ADC, that can be set up to take single samples or multiple samples; however it outputs a single 16 bit average measurement, where the 2 additional bits are created by the averaging process.
To read the voltage at which each cell is working, that should not go far from

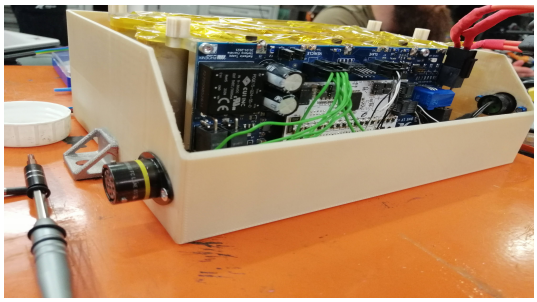


Figure 4: Signal connector

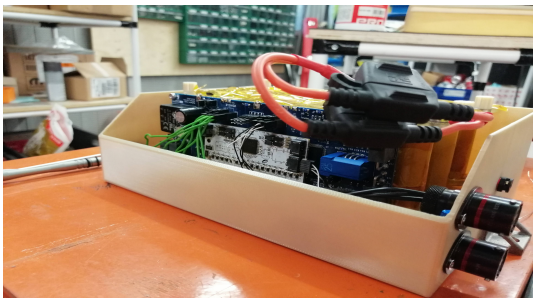


Figure 5: Power connectors: 24 V on the bottom and GND on the top

4.2 V (maximum possible value), we have to set the VSENSE input channels, that measure the voltages of individual cells in the range of 1V to 4.95 V.
To read the temperatures, instead, we set the AUXs Input Channels, that are used to measure external analog voltages from 0 V to 5 V. A typical use for these channels is to measure the temperature using NTCs.
The slave boards are in a stacked configuration, where the main micro-controller first communicates through one of the boards in UART and then the communication is relayed up the chain of connected slave devices using a differential communications protocol.
The differential VBUS uses an asynchronous byte-transfer protocol with **one start bit**, that is always zero, **eight data bit** and an optional **framing bit**. If detection of the start bit occurs, the receiver samples the input on the fourth clock edge to produce the bit and then it re-synchronises on every falling edge, that is expected at least once every 3 bits.
Via firmware, we also set some over-voltage and under-voltage thresholds, that are important to be set because far or below those values the cells cannot go, otherwise we would have some problems regarding the state of charge of the battery and their life cycle. Furthermore, these thresholds are essential because, when the battery pack is not being charged, its cells start self-discharging, decreasing in this way the voltage level at which they would be if they were undertaking the charge process. Going below the minimum allowable value is a problem.

To make the slave device communicate with our BMS LV, it should enter the active state. In fact, this device has three power states or modes:

1. SHUTDOWN/SLEEP: the lowest power state used for long periods of inactivity to extend battery life. The part must receive a *high signal on the WAKE-UP pin* or a WAKE-UP tone through the vertical communications bus to transition to the IDLE state.
   This wake-up pin is active low.

2. **IDLE**: the default condition when awake and ready to receive and execute

commands.
The device will wake up and enter the IDLE state when either of the following conditions occurs:

(a) WAKE-UP pin is high;

(b) COMML pins receive the WAKE tone, where COMML belongs to the differential communication circuits;

3. **ACTIVE**: the highest power state while communicating –> communication is active.

The WAKE-UP pin can bring the part from the SHUTDOWN state to the IDLE state. When it is in SHUTDOWN and the WAKE-UP pin becomes high, the slave device will transition from SHUTDOWN to IDLE. After applying the high signal, the WAKE-UP pin must de-assert and then return to the low state to allow the part to enter the SHUTDOWN again. Upon changing state, the device will transmit a differential wake-up tone on its COMM+ and COMM- pins to the next slave connected to the daisy chain. The bottom device in the stack sends the wake-up tone out of the COMMH pins in response to the WAKE-UP pin assertion. The next device receives the tone on its COMML pins. The tone then propagates up in the stack in this way to wake all devices in the stack.

Please, notice that the wake-up pin **must** be held in the low state to allow the device to enter and remain in the SHUTDOWN state (**setting the pin low does not place the device in shutdown**). Operations are unpredictable if this pin is left floating.
To set all the configurations needed to start the communication between BMS LV board and the slave device, make reference to the data-sheet of the device, where all the registers, values and commands are shown.

6 CURRENT SENSOR INTERFACE

In Figure 6, it is shown the implemented circuit for the current sensor. It is connected from one side - pins 1,2,3 - to the connector **J10**, that is connected in turn to the positive side of the low-voltage battery, and from the other side - pins 9, 10, 11 - to the rest of the car that takes the 24 V. So, if everything is working properly, what we have at the input pins are found at the output pins. The output signal, instead, is at pin 11, called "$CURR\,SENS$", where the voltage reading can be obtained. To understand which is the the correct voltage reading, we need to make reference to the data-sheet of the sensor LEM CAS 50 NP, look at the output characteristic and compute the output voltage V_{out} as it suggests: $V_{out} = 2.5\,V + \frac{\Delta_{VPN}}{\Delta_{IPN}} I_{IN}$, where V_{PN} and I_{PN} are written in the data-sheet, according to the chosen sensor, 2.5 V is the offset and I_{IN} is the

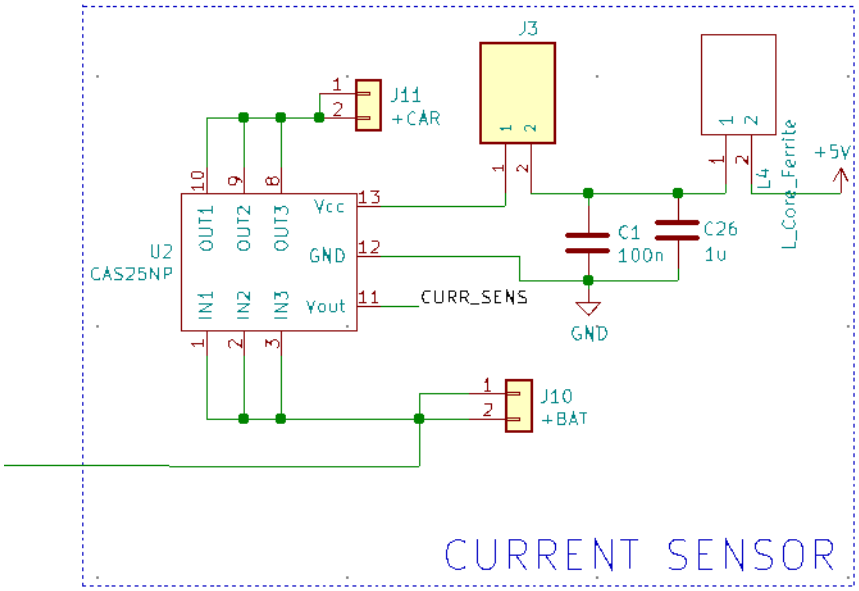


Figure 6: Current sensor scheme

current flowing at that moment into the board.
It is important to remark that the supply voltage of the BMS LV board and of the current sensor **must be** different! The sensor, in fact, takes the positive 24 V directly from the battery by means of J10 and it moves around the car exiting from J11. The board, instead, takes the supply voltage from the connector J14 shown in Figure 7.

To supply the sensor at 5V, we need to short-circuit the connector J3, that is used for the continuity like the others described in the previous sections.
In order to have a good signal, on the right, we put 2 capacitors and a ferrite, whose aim is simply to filter the disturbances in a better way.

7 MCU

To better understand this section, make reference to the relative documentation. However, spending some words about the micro-controller of our board, it is essential to say that, from it, is sent the GND signal that goes to the negative

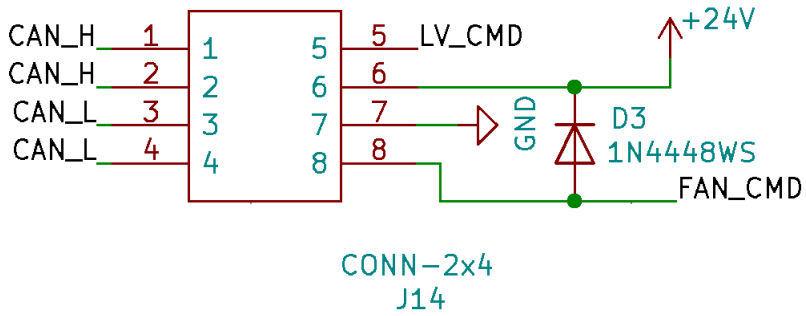


Figure 7: Supply connector of the board. Here, the board receives 24 V, that is different with respect to the connector from which, instead, the current sensor takes the power supply.

command of the relay board (connected to one pin of the coil of the relay). There are also some LEDs that represent:

1. *RED LED blinking*: no slave connection
2. *RED LED fixed*: fatal error
3. *RGB GREEN*: core running
4. *BLUE*: start UART communication
5. *ORANGE LED*: CAN communication

8 COMPLIANCE WITH THE RULES

In this section, the LV system is going to be commented according to the rules.

1. **T 11.1.1**: The Low Voltage System (LVS) is defined as every electrical part that is not part of the TS, see EV 1.1.1
   Comment: our battery does not include anything related to the high voltage side of the car. The maximum reachable value is 29.4 V, that is still considered low voltage, because this is lower than 60 V DC.
2. **T 11.1.4**: The maximum permitted voltage that may occur between any two electrical connections in the LVS is 60 V DC or 25 V AC RMS.
   Comment: this is linked to the previous comment.

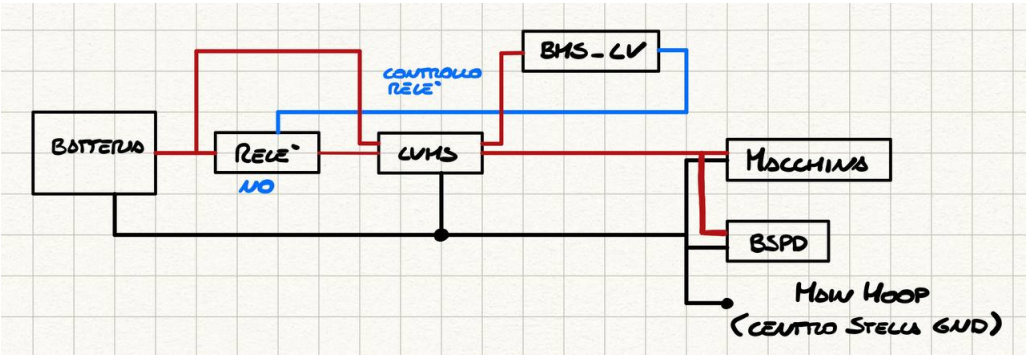


Figure 8: Low voltage scheme

3. **T 11.1.5**: [EV only] The LVS must be grounded to the chassis.
   Comment: the power signal GND goes to the Main Hoop and from there to the fusebox.

4. **T 11.3.1**: An LVMS according to T 11.2 must completely disable power to the LVS.
   Comment: this is possible when we turn off the master switches. These switches, in fact, are used to provide the car with the low voltage needed to start the ready-to-drive sequence. If anything happens or if we just want to disable the low voltage in the car, we can do that turning off these LVMS. Looking at Figure 8, it is necessary to notice that the LVMS have two "inputs" that are the positive pole got directly from the battery and the positive pole received instead from the relay. As it will better explained in the Relay board documentation, if the coil is not closed, the connection relay–>LVMS is interrupted, hence the LVMS only receives the supply voltage from the battery and then it goes to the BMS LV board, but, since the connection to the relay is now broken, the LVMS won't be able to supply the vehicle with the low voltage since there is no more a link between them (while the BMS LV board will still be on).
   The BMS LV board also knows if the car is receiving power because it decides the state of the coil by means of the LV_{CMD} signal.

5. **T 11.7.1**: LV batteries are all batteries connected to the LVS.
   Comment: we have 7 series and 5 parallels and the cells are all connected to the low voltage system.

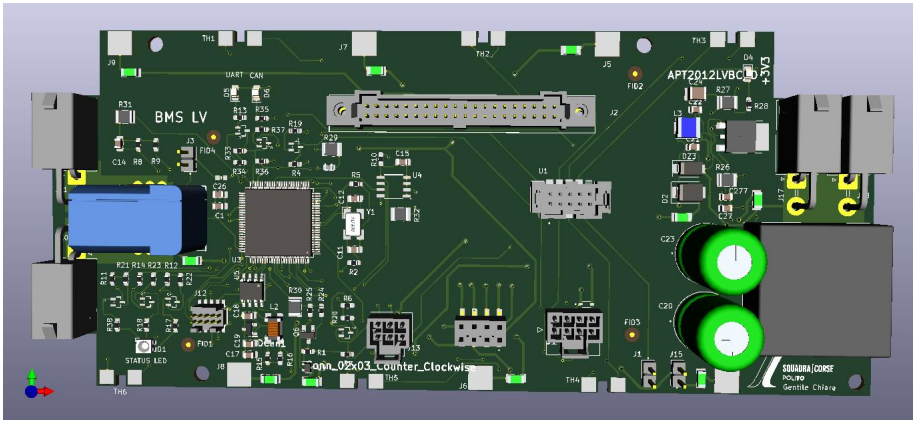


Figure 9: Front view

9 DESIGN

As it can be seen from Figure 9, we realized this board using 4 layers: 2 of signal, that are represented by the red and green traces, one layer for 3.3 V and the remaining layer for GND. Furthermore, since into this board will flow a huge amount of current, approximately 50 A, we drew some huge pads for heat dissipation. This is important to have near the current sensor and near the connectors used to take the power from the battery and send it to the car. (In the picture, they are those dotted pads on the right and on the left).

10 SC21: PROBLEMS

Last year, the main problems were the absence of functional filters protecting the micro-controller and the board in general, but also the under-sizing of fuses, whose rating was very low.
Because of these, this year, we have designed more efficient filters, described in the appropriate section, and we increased the fuses values, that are fire-retardant and useful to protect the circuit against current peaks.
Something very important introduced this year is a component that protects the MCU inputs, preventing it from being damaged in case a higher voltage is present at its inputs.

11 Pictures

:

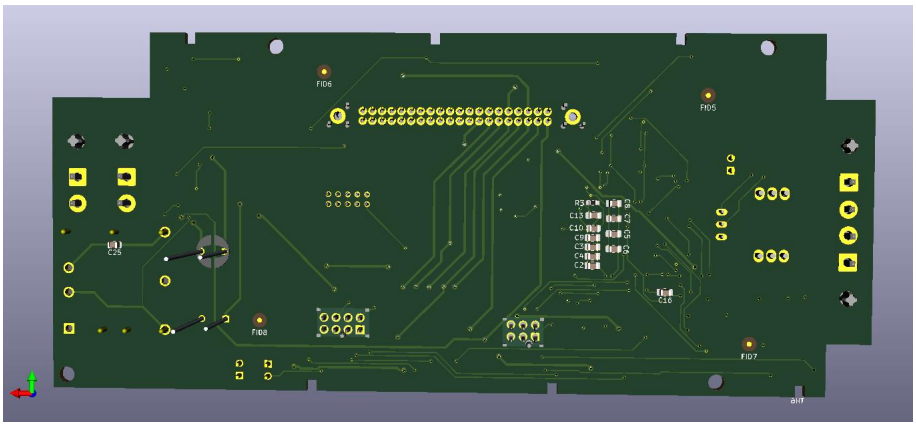
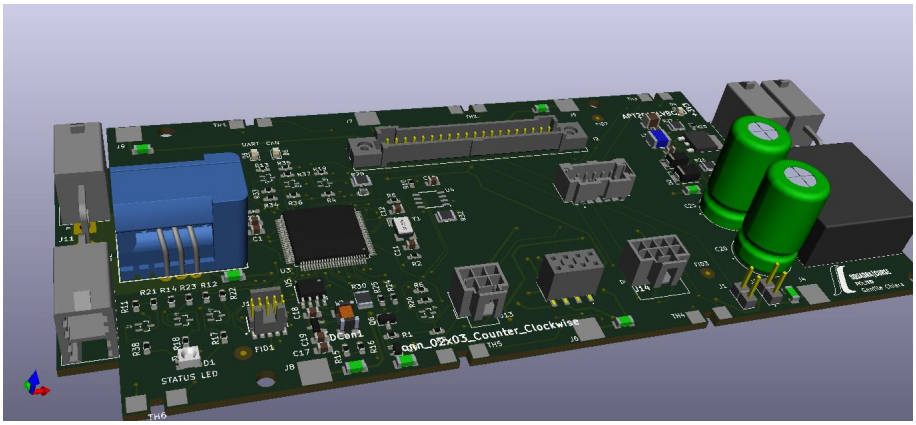
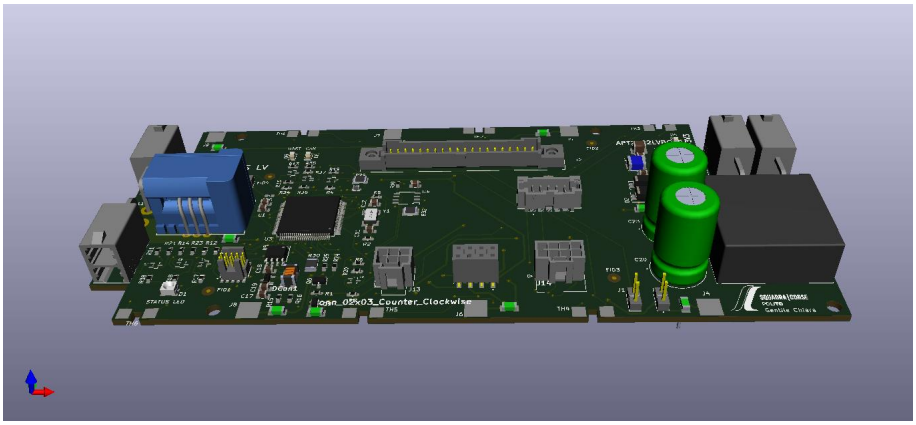
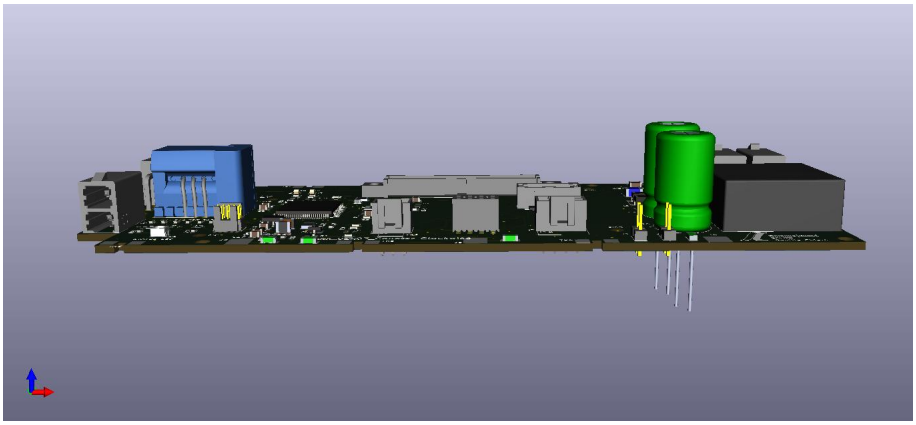


Figure 10: Rear view







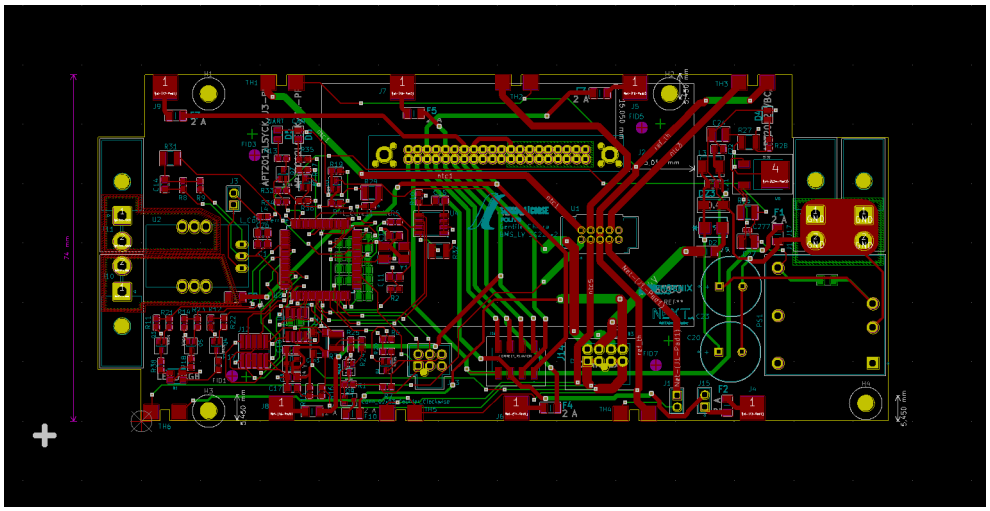


Figure 11: PCB

12 Firmware

There is a particular procedure to follow in order to begin the communication BMS LV - Slave. All these instructions are written in the data-sheet of the slave board. The user configuration is the following:

1. **Set an address for each device in the stack, using GPIO pins**
   (a) Make sure all devices are awake and ready to receive the Auto-Address Enable command;
   (b) Turn on the communications drivers in the chain;
   (c) Place all devices in Auto-Address learn mode;
   (d) Send out new addresses to all possible slave devices in incrementing order;
   (e) Read back the value stored in the Device Address register. The last device connected to the chain is the one to respond successfully;
   (f) Turn off the high-side communications receiver on the last device in the chain;
   (g) Turn off the single-ended transmitter on all the devices except for the last one;
   (h) Starting at the top of the stack of devices, clear all existing faults.
2. **Configure the AFE (Channel Selections and Fault Thresholds)** This is important to do before reading voltages from the daisy-chain. They have to be able to scan the desired channels at the desired time.
   We start from the highest device and going down to the last one of the chain.
   (a) Configure GPIO pins as required by our design;
   (b) Configure the initial sampling delay;
   (c) Configure voltage and internal sample periods;
   (d) Configure the oversampling rate;
   (e) Select the number of cells and desired channels to sample;
   (f) Set over-voltage and under-voltage thresholds;
   (g) Check and clear faults
3. **Reading Voltages from the chain** Each device monitors a section of a stack of cells. However, we only have a slave device and it has to be capable of capturing the voltages of each sub-stack in the LV battery.
   The device first samples; once the sampling is complete, the device will respond.

To better understand how the communication works, you have to keep in mind that:

1. The BMS "turns the slave on" and starts the communication is everything is fine and is working well;
2. The BMS LV board sends all the commands to the slave in order to tell it which values must be set as minimum/maximum, which channels activate or faults clear;
3. The slave must send an acknowledge (that we don't read by the way);
4. Once everything is set and ready, the slave reads the voltages from the Vsense inputs and the temperatures from the die analog temperature channels and send the information to the BMS LV;
5. If the slave detects a voltage or a temperature or something strange with respect to the set values by the BMS LV, it sends a message of error to the BMS and it will react according to the type of error.

12.1 Functions and Variables

In this subsection, variables and functions are going to be described.

1. **uint8t Frame[4]**: this is a CAN frame. CAN Data Frame is the most common message and it contains the following parts:
   (a) **CAN IDENTIFIER:** it determines the message priority when 2 or more nodes send messages through the same bus;
   (b) **DATA FIELD:** it contains from 0 to 8 data bytes and it corresponds to where we can send data;
   (c) **DATA LENGTH CODE (DLC):** it specifies the number of bytes in the data field;
   (d) **CRC FIELD (CYCLING REDUNDANCY CHECK):** it contains a checksum that is useful to identify errors;
   (e) **Acknowledgement Slot:** each CAN able to receive a message correctly, sends an acknowledgement bit, otherwise it sends the message again until the acknowledgment is sent.
2. **uint8t TxData[8], RxData[8]:** these are two arrays of 8 elements (that are 8 bytes) on 8 bits each and they are used to receive and transmit data;
3. **uint8t uart tx cmd[10]:** it is the UART variable responsible of sending commands to the slave. It contains 10 elements, that are actually 10 bytes (in fact, each byte is defined on 8 bit, so we are coherent with the assignation). However, we only need 8 bytes, that is the number of bytes needed for the command frame, but we consider two additional elements;

4. **uint8t uart rx cmd[40]:** this is the UART variable responsible of receiving commands and it includes 40 elements (that are still bytes) because we need 2 bytes*16 channels (8 channels for the temperature readings and 8 channels for the voltage readings) + 2 CRC bytes + 1 bytes for the initialization.

5. **uint16t cell voltages[8], cell temperature[8]:** these are the variables used to store the information about the

6. **cell rx uv[4], cell rx [4]:** these are used for the over-voltage and under-voltage detection.

7. **unsigned short crc16 table [256]:** this is a table computed on $http://www.sunshine2k.de/coding/javascript/crc/crc_js.html$ for the computation of the 2 CRC bytes
   CRC bytes are some control parameters between slave and BMS LV that verify whether the sent messages are correct or not and so they are a kind of check on the sent message. When the two boards (slave and master) don't read the same message, an error is generated. However, we don't check the correspondence between the CRC sent by the slave and the one sent by the BMS LV, but, if they are different, we read the error message.

8. **unsigned short crc16 (unsigned crc, unsigned char const *buffer, unsigned int len):** this function is suggested by the datasheet of the slave (pag 55). It is on 16 bits (= 2 CRC –> 2 CRC frames). This function returns the crc value that goes then to the UART Send CMD function, where the crc is set in the variable crc result , while *buffer from cmd[i] that is computed according to a variable list. Then, we check the transmitted data and we control if it coincides with the CRC of the slave that identifies the way to start the communication.

9. **HAL StatusTypeDef UartSendCMD (int timeout, int size, ...):** this function is used to send the commands in Uart to the slave device, so that we can fix some values that are needed to start our communication with that board, such as some voltage and temperature thresholds.
   This function returns type, that is 1 when the transmission has not been completed correctly.

10. **void SetUpBmsLV(void):** this function is used to set all the commands to send to the slave board so that it can wake up, activate itself and execute the instructions I tell him to do.
    To learn more about the way to send commands, make reference to the next section.

11. **uint32t ReturnTime100us (void):** this function is used to increment the counter (incremented every 100 us). To each interrupt, the counter is incremented because the ARR has an overflow.

12. **int delay fun(uint32t *delay100uslast, uin32t delay100us):** this is used to count the time in between two executions: if the new time is lower than the actual one, the function returns 0 and this means that we can't transmit another command because the time has not been completed yet, so the instruction has not been sent entirely. Once the new time we read is greater than the actual one, this implies the fact that we can send the new command and so the communication goes ahead.

13. **void GetVoltagesandTemperature(uint32t delay100us):** it is used to read voltages and temperatures fromt the slave device, that first samples and then it returns the values to the BMS LV. We also check the voltage values to be sure that we don't read any over-voltage or under-voltage value and, if this happens, we open the relay. The same holds for the temperatures.

14. **HAL TIM PeriodElapsedCallback(TIM HandleTypeDef):** it is the function that increments the counter variable when an interrupt occurs.

15. **CAN Msg Send(CAN HandleTypeDef ...):** this function is used for the CAN communication: we send the data to dSpace via CAN by means of this function. In particular, we check if there are free levels, that is essential to identify whether we can send the message or not in order to send the message. Then, there is an additional check in order to identify if we can add the transmitting message to the CAN path. In case of problems, we send an error message.

16. **void CAN Tx(uint32 t delay 100us):** this sends the messages and data in CAN to dSpace. In particular, we send the information about the voltage and temperature of the cells, such as the data collected with the current sensor of the BMS LV board. However, we send the data of the sensor in two steps, because the curr sens is defined on 16 bits, so we send the first 8 bits in TxData[0] and the remaining 8 bits in TxData[1].
Finally, all this piece of information is sent using the function described previously (CAN Msg Send).

17. **CoreBmsLV (uint32 t delay 100us):** this is the core function of our BMS LV, that contains the functions for reading the time delay, the data coming from the memory DMA, that contains the data collected by the current sensor. We also turn on the yellow LED identifying the LED of the slave. Then we read the voltages and temperatures, we set some parameters on the slave with the function Uart Send CMD and then we send the data with the CAN Tx function.

18. **void HAL ADC ConvCpltCallback(ADC HandleTypeDef *hadc):** We check whether the current of the LV is higher than 50 A or lower than -20 A. We set these limits by computing first the output voltage from the current sensor, to which a particular current is associated. In particular,

we have the formula $V_5 = 2.5V + 0.0125 * I_{in}$, where 0.0125 has been computed from the datasheet: $\frac{(3.125-1.875)}{2I_{PN}}$, that corresponds to $\frac{\Delta}{V}\Delta I_{PN}$. Then, we compute the voltage entering the uController with a voltage partition: $V_{3v3} = V_5 \frac{82}{82+42.2}$.

Finally, we compute the ADC value, that is given by $V_{3v3}\frac{2^12-1}{3.3}$, where N is the number of bits of our ADC converter (N = 12 bits).
I_{IN} has been set as maximum value (50 A) and minimum value (-20 A).

19. **void HAL CAN RxFifo0MsgPendingCallback(CAN HandleTypeDef *hcan):** it triggers when an interrupt occurs

20. **void LedBlinking(GPIO TypeDef *GPIOx, uint16 t GPIO Pin, uint32 t delay 100us):** for blinking some LEDs.

21. **int check fault ov uv(uint8 t fault reg):** we look for overvoltages and undervoltages and, in that case, we return the fault register first 8 bits.

22. **void Ping Mng(uint32 t delay 100us):** for the transmission of data via CAN.

23. **void Open Relay(uint8 t msg):** this opens the relay in case some particular errors occur, such as over-voltages, under-voltages or over-currents.