SCIENTIFIC RESEARCH AND PRACTICE

48V Battery Management Systems for Formula Student Vehicle

Bramantio Yuwono

EPIC 2022-2023

Project Coordinator: Lucian Andrei Perişoară

June 2023

1. Introduction: Cell Balancing in Battery Management Systems (BMS)

Over the past few years, there has been a substantial rise in the need for dependable and high-performing energy storage systems. This has prompted the advancement of different battery technologies. Nonetheless, ensuring that voltage levels remain balanced within individual cells or modules poses a significant challenge in battery systems. Imbalanced batteries can result in decreased overall capacity, shortened lifespan, and potential safety risks.

In multi-cell battery systems, battery cell balancing is an essential procedure carried out by battery management systems to guarantee even charging and discharging of each cell. This balancing is critical due to slight discrepancies in characteristics like capacity, internal resistance, and voltage among the cells within a battery pack. Over time, these discrepancies can result in imbalances between the cells, which can have detrimental effects on the overall performance, capacity, and lifespan of the battery pack.

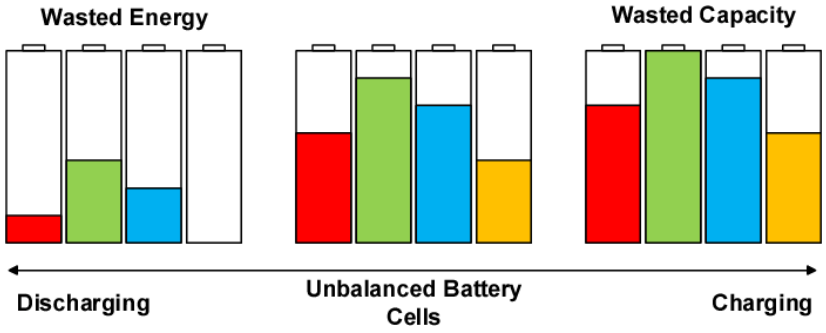

Figure 1. Energy loss and capacity loss under the unbalanced situation [1]

For example, in the case of a Lithium-ion battery bank, as shown in Figure 1, safety issues arise if overcharging or over-discharging takes place. During discharge, the battery cell with the lowest state of charge (SoC) or sometimes called the weak cell will reach the lower limit of the safe voltage range before the others, causing the battery management system to stop operating. Similarly, while charging, the cell with the highest SoC (the strong cells) will first reach the upper limit of the safe voltage range, leading the BMS to halt the charging process.

Battery management systems have a crucial role in overseeing and managing the cell balancing procedure. The BMS constantly monitors each cell's voltage, temperature, and other relevant parameters. Based on this information, it determines whether balancing is necessary and activates the appropriate balancing method.

Cell balancing is essential for maximizing the overall capacity and performance of the battery pack. Preventing overcharging in some cells and undercharging in others helps optimize the battery pack's capabilities. Additionally, cell balancing aids in prolonging the lifespan of the battery pack by reducing stress on individual cells, which can cause premature ageing.

It's important to highlight that cell balancing is an ongoing process that needs to be regularly conducted, particularly during charging and discharging cycles. This ensures that the battery pack operates at its peak performance level and maintains a consistent and balanced state across all cells.

1.1. Passive Balancing

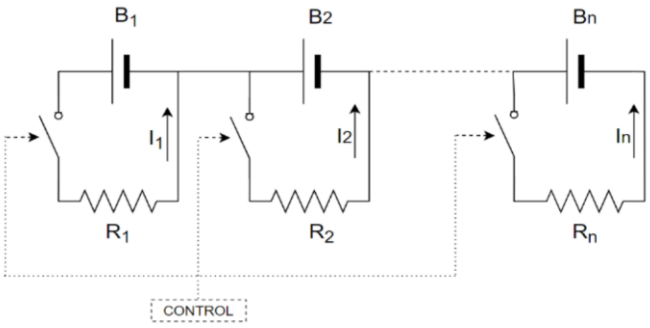

Figure 2. Example of passive balancing circuit with shunt-switched resistors [2]

In passive balancing, the excess energy from the strong cells is dissipated as heat through external resistors. This method is simple and cost-effective but not very efficient since it wastes energy as heat.

1.2. Active Balancing

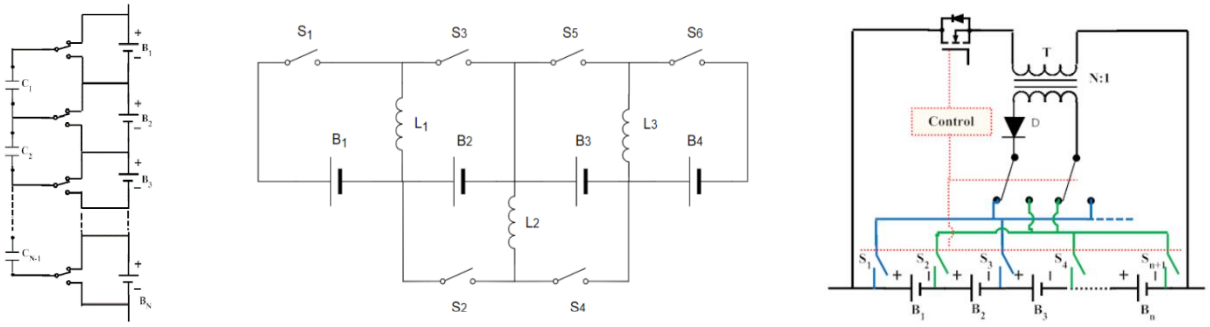

Figure 3. Examples of active balancing circuits with switched capacitors(left), single switched inductors(middle) and a single winding transformer(right) [2]

Active balancing redistributes the energy among the cells using active electronic circuits. These circuits monitor the individual cell voltages and transfer energy from the stronger cells to the weaker cells by discharging the stronger cells and moving the charge to the weaker cells using capacitors, inductors, transformers, and converters. This method is more efficient than passive balancing and allows for better control over the balancing process.

2. Battery Management Systems Topology

The proposed 48V battery management systems utilize the Master-Slave topology. In this configuration, each cell in the battery pack is paired with a slave module. These slave modules acquire essential data like voltage and temperature from their respective cells and forward it to a master module via the Controller Area Network (CAN) bus.

Once the master module receives this information, it analyzes the data and makes decisions accordingly. If any modules require cell balancing, the master module sends commands to the respective slave modules. Simultaneously, the master module also transmits the data to a PC using a USB interface. In the PC, data received from the master module is read and stored on Data Acquisition - Graphical User Interface (DAQ-GUI) implemented using LabView.

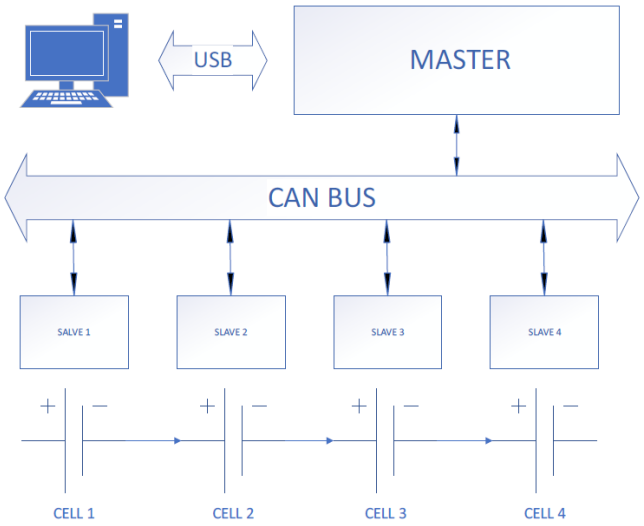

Figure 4 The block architecture of the proposed BMS [3]

The battery pack consists of four Lithium Iron Phosphate (LiFePO4) cells connected in series, forming a 4S1P configuration, with a nominal voltage of 12.8V. The proposed Battery Management System (BMS) incorporates the CALB CA180FA, which possesses the following specifications: a nominal voltage of 3.20V, a nominal capacity of 180Ah, an operating voltage range of 2.60V to 3.60V, a discharging minimum cut-off voltage of 2.50V, a charging maximum cut-off voltage of 3.65V, dimensions measuring 278x180x71 mm, and a weight of 5.8 kg.

The proposed BMS utilizes a decentralized architecture in the form of a Master-Slave configuration, wherein each cell is paired with a slave module. When the master module initiates a balancing operation, the designated slave module discharges the battery through two power resistors (passive balancing). The proposed BMS ensures a high balancing current of up to 1.3 A and precise balancing with a voltage imbalance tolerance of up to 2 mV. Due to the Master-Slave BMS topology, the balancing operation can work in parallel so more than one cell can be balanced at the same time.

Other than balancing operation, the proposed BMS also provides voltage and temperature monitoring features that will be used as a deciding factor for overcharging and deep discharging protection.

2.1 The Master Module

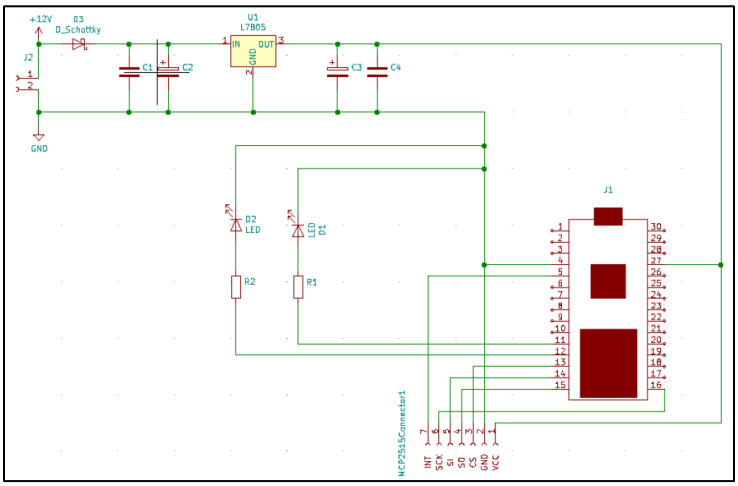

Figure 5 Electronic schematics of the master module [3]

The Master module is constructed using the following primary components: an Arduino Nano development board, a CAN board featuring the MCP2515 CAN Controller and TJA1050 CAN transceiver, and an L7805CV linear voltage regulator. Two LEDs are utilized as follows: LED D1 is connected to digital pin D8 of the microcontroller, indicating when a cell reaches its maximum voltage value. The other LED is connected to pin D9 of the Arduino and indicates when a cell reaches its minimum voltage value.

2.2 The Slave Module

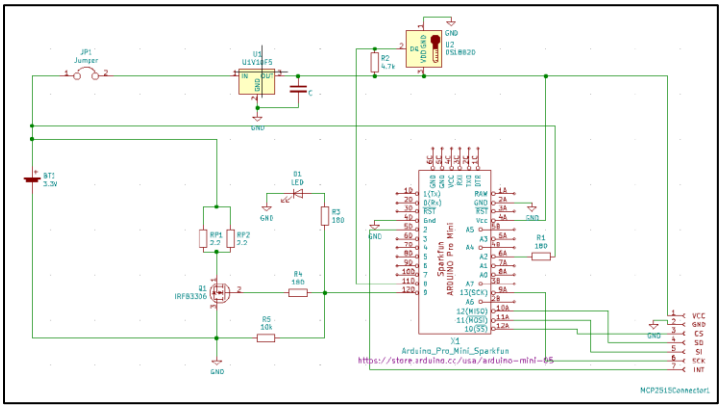

Figure 6 Electronic schematics of the slave module [3]

The Slave module consists of the following key components: a DC-DC converter labelled as U1, a temperature sensor designated as U2, a MOSFET transistor referred to as Q1, two power resistors identified as RP1 and RP2, an Arduino Pro Mini development board featuring the ATMega328P microcontroller, and a CAN board integrating the MCP2515 CAN controller and TJA1050 CAN transceiver.

The microcontroller measures the voltage of the cell by connecting the positive terminal of the cell to the analogue pin A2 of the Arduino board. To safeguard the input of the analogue-to-digital converter, a 180-ohm resistance is employed.

The DS18B20 temperature sensor is used to measure the cell's temperature. Its DQ pin is connected to digital pin 8 of the Arduino board. To ensure continuous operation of the communication line when one-wire communication is not in use, a 4.7-kilohm resistor is employed.

The passive balancing circuit comprises the MOSFET transistor IRF83306 and two power resistors, RP1 and RP2, with values of 2.2 ohms and 5 W, connected in parallel. When balancing becomes necessary, the microcontroller controls the transistor by applying a voltage to its grid terminal (pin 9) through the resistor R4. This connection enables the two power resistors to be connected to the cell, resulting in the consumption and dissipation of energy from the cell as heat.

The communication between the MCP2515 CAN controller and the Arduino board occurs via the SPI interface. Therefore, the INT pin of the CAN module is connected to digital pin 2 of the Arduino, the SCK pin to digital pin 13, the SI pin to digital pin 11, the SO pin to digital pin 12, and the CS pin to digital pin 10.

2.3 The Communication Protocol between Master and Slave Modules over the CAN bus

The message transmitted by the Master module has a CAN ID of 0x100, with a data field containing 2 bytes. The first byte, called SlaveID, stores the identification of the Slave module for which the balancing command is intended. The second byte indicates the balancing state decided by the Master module after processing the information. It specifies whether to initiate balancing (0x01) or stop it (0x00).

For the slave modules, the messages contain a CAN ID ranging from 0x101 to 0x104, representing the address of each slave module. This is followed by a 4-byte data field structured as follows:

- The first byte denotes the current balancing state of the cell.
- The subsequent two bytes (High Byte, Low Byte) are utilized to transmit the cell's voltage.
- The final byte is allocated for transmitting the cell's temperature.

CAN ID	Length (bytes)	Data							
		0	1	2	3	4	5	6	7
0x100 (Master)	2	SlaveID	Balancing Command	N/A	N/A	N/A	N/A	N/A	N/A
0x101 (Slave1)	4	Balancing State	Voltage [HB]	Voltage [LB]	Temperature	N/A	N/A	N/A	N/A
0x102 (Slave2)	4	Balancing State	Voltage [HB]	Voltage [LB]	Temperature	N/A	N/A	N/A	N/A
0x103 (Slave3)	4	Balancing State	Voltage [HB]	Voltage [LB]	Temperature	N/A	N/A	N/A	N/A
0x104 (Slave4)	4	Balancing State	Voltage [HB]	Voltage [LB]	Temperature	N/A	N/A	N/A	N/A

Figure 7 The structure of CAN frames

3. Design of the Master-Slave BMS Software

3.1 The Master Module

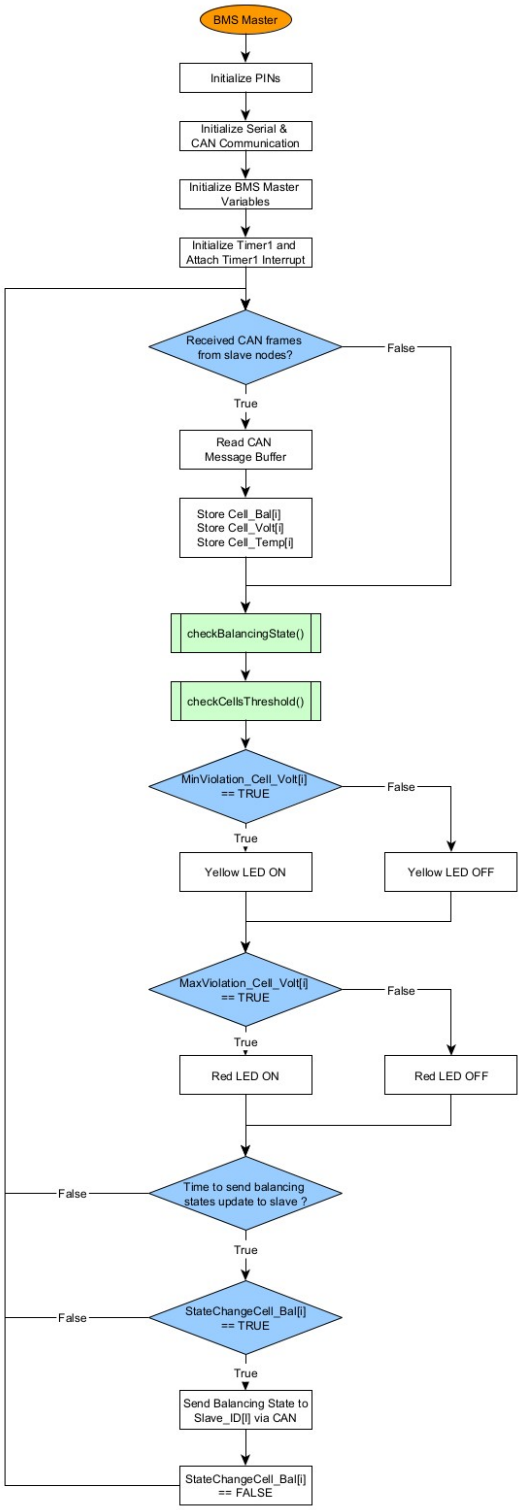

Figure 8 Flowchart of BMS Master Software

In the setup() function of the Arduino code, several initializations are performed. This includes setting up the necessary PINs, configuring serial and CAN communication, initializing BMS Master variables, and setting up Timer1. To enable CAN communication on Arduino, the libraries <SPI.h> and <mcp_can.h> are utilized. Timer1 is used to synchronize the communication between the Master BMS and the DAQ GUI on the PC and it is implemented with <TimerInterrupt.h> library. The time interval for sending data to the DAQ GUI can be adjusted by setting the value of DAQ_INTERVAL_MS. The initialized BMS Master variables are as follows:

- Cell_Bal[4]: integer array indicating the balancing state for each of the four cells.
- Cell_Volt[4]: integer array representing the voltage values of the four cells.
- Cell_Temp[4]: integer array storing the temperature values of the four cells.
- MaxViolation_Cell_Volt[4]: boolean array indicating whether the maximum voltage threshold has been exceeded for each cell.
- MinViolation_Cell_Volt[4]: boolean array indicating whether the minimum voltage threshold has been violated for each cell.
- MaxViolation_Cell_Temp[4]: boolean array indicating whether the maximum temperature threshold has been exceeded for each cell.
- StateChange_Cell_Bal[4]: boolean array indicating whether there is a need to change the balancing state for each cell.
- buff_Cell_Bal[4]: buffer integer array storing the balancing states to be sent via CAN.

In the loop() function, the code first checks if the Master BMS has received a CAN frame from the slave nodes. If a CAN frame is received (CAN Interrupt pin is equal to 0x00), the Master BMS updates the balancing state, voltage, and temperature values the respect to the received Slave_ID. The checkBalancingState() function is then called to determine if any cells require balancing, and the checkCellThreshold() function is used to check if the voltage and temperature thresholds have been violated. If the minimum or maximum voltage thresholds are violated, the Yellow LED or Red LED, respectively, are activated. If the period for sending balancing state updates has elapsed, the updated balancing state will be sent to the Slave_ID that has a flagged StateChangeCell_Bal. The time interval for sending balancing state updates to the slave modules can be adjusted by setting the value of the CAN_Timeout variable.

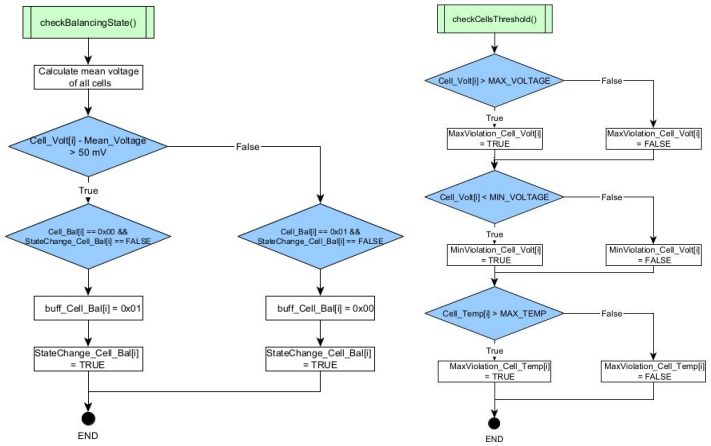

Figure 9 Flowchart of function checkBalancingStates() and checkCellsThreshold()

The checkBalancingState() function is responsible for updating the balancing state of all cells. It begins by calculating the mean voltage of all the cells. The update of the balancing state is only executed if the StateChange_Cell_Bal is not flagged, indicating that a previous balancing state update is not required. If the voltage of a cell exceeds the balance tolerance value (50 mV), the balancing state is set to 0x01, indicating that cell balancing is necessary. Conversely, if the voltage of a cell is below the balance tolerance value, the balancing state is set to 0x00, indicating that cell balancing is not required. This process is carried out for each cell.

In the checkCellsThreshold() function, the cell voltages and temperatures are compared to their respective threshold values. If the cell voltages exceed the maximum threshold value, the MaxViolation_Cell_Volt is flagged. Similarly, if the cell voltages fall below the minimum threshold value, the MinViolation_Cell_Volt is flagged. Moreover, if the temperature of a cell exceeds the maximum temperature threshold value, the MaxViolation_Cell_Temp is flagged.

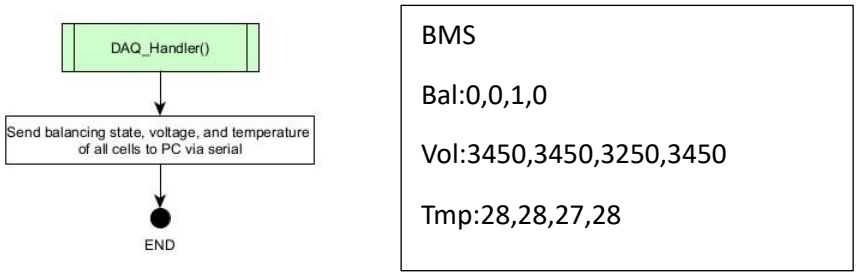

Figure 10 Flowchart of DAQ Interrupt Handler and example of data format sent to PC via serial

DAQ_Handler() is attached to the Timer1 interrupt, meaning that it will be triggered when the counter surpasses the predetermined value. Once triggered, DAQ_Handler() is executed. Within this function, the balancing state, voltage, and temperature of all cells are transmitted to the DAQ GUI on the PC through serial communication. It is crucial to adhere to the format depicted in Figure 10 to ensure proper synchronization of the transmitted data from the Master BMS to the DAQ GUI.

It is important to notice as well the verification of temperature will not yield an action to cut the power connection because the relay to disconnect the load to the battery pack is not yet implemented on this BMS.

3.2 The Slave Module

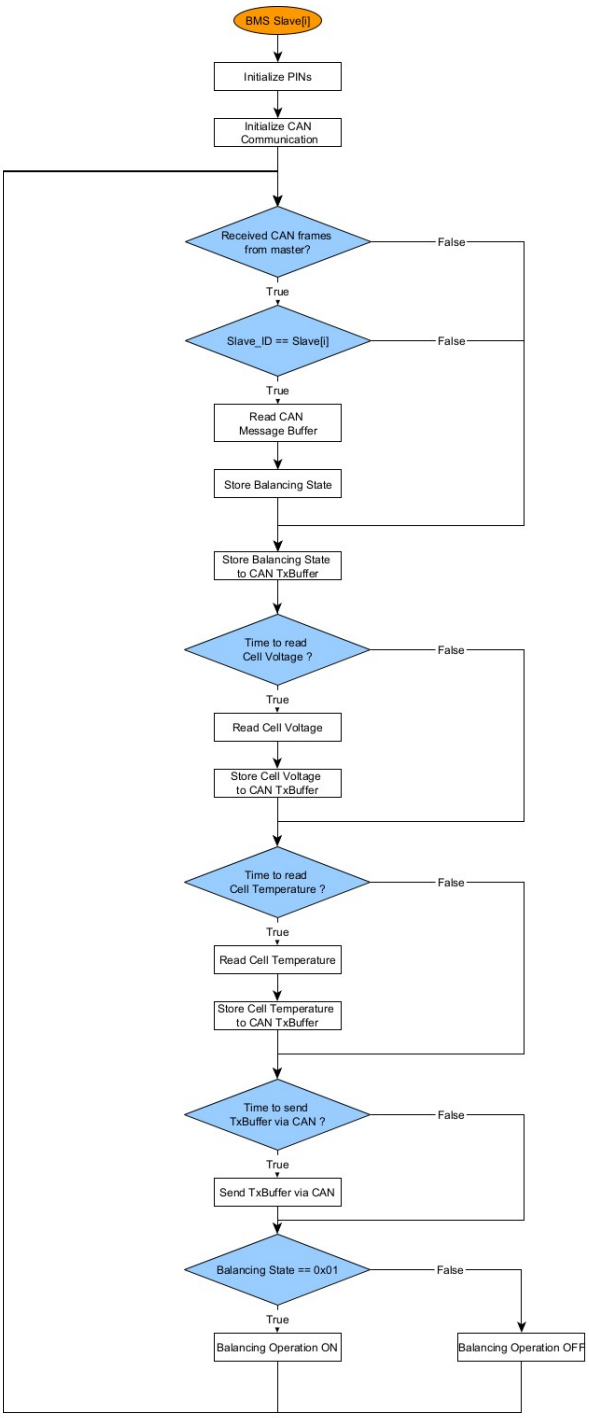

Figure 11 Flowchart of BMS Slave Software

In the setup() function, the initialization process begins by setting up the necessary PINs for measuring cell voltage, CAN interrupt, and performing balancing operations. It is followed by initializing the CAN communication. The millis() function is executed, which helps determine when voltage and temperature readings should be performed and when the CAN frame should be transmitted to the Master BMS. The frequency of voltage and temperature readings can be adjusted by modifying the Temp_Timeout value, while the interval for sending CAN frames can be configured by changing the CAN_Timeout value. To read temperature from the DS18B20 sensor, the <OneWire.h> and <DallasTemperature> libraries are utilized.

In the loop() function, the code first checks if there is a CAN frame received from the Master BMS. If a frame is received, it verifies if the frame is intended for the corresponding Slave_ID. If it matches, the transmitted cell balancing state is stored. The code then proceeds to check the timing to determine if voltage and temperature readings should be performed. If it is necessary, the cell voltage and temperature readings are conducted. The obtained balancing state, cell voltage, and temperature values are stored in the CAN TxBuffer. If it is time to transmit the TxBuffer to the Master BMS via CAN, the code executes the CAN transmitting function. At last, if the Master BMS requests the balancing process, the slave module activates the D9 pin, and vice versa.

4. Design of the Data Acquisition Graphical User Interface

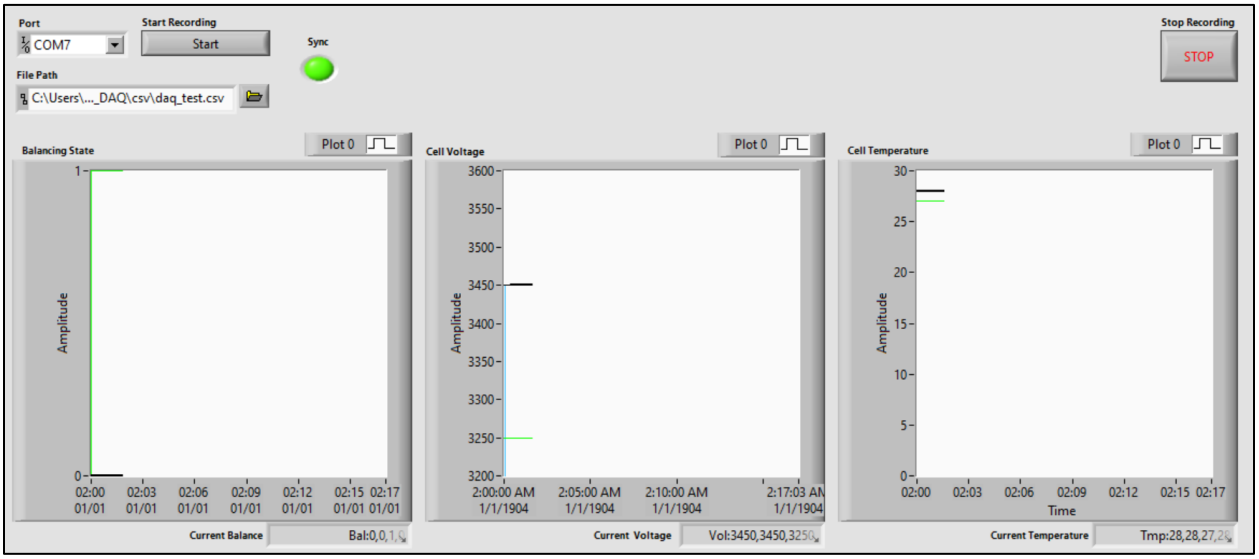

Figure 12 DAQ GUI on Labview

The DAQ GUI display is depicted in Figure 12. Before commencing the recording process, the user must select the serial port to which the Master BMS is connected on the PC. Additionally, the file path needs to be specified, including the desired file name for storing the recorded data. It is important to note that both the Port dropdown and the File Path textbox must be filled to proceed. Once the Start button is clicked, the recording process can commence.

To establish synchronization between the DAQ GUI and the Master BMS, the GUI needs to wait for a few instances of data transmission from the Master BMS. The synchronization status is indicated by the Sync green LED located adjacent to the Start button. Once synchronization is achieved, the graph will begin displaying the balancing state, voltage, and temperature for all cells. The raw serial data received can also be observed at the bottom of the graph. Throughout the recording process, the DAQ GUI will continuously update the selected CSV file specified in the File Path textbox. Pressing the Stop button will halt the recording process.

Timestamp	Balance_	Balance_	Balance_	Balance_	Voltage_(	Voltage_(	Voltage_(	Voltage_(	Temperat	Temperat	Temperat	Temperat
6/17/2023 22:17												
6/17/2023 22:17												
6/17/2023 22:17	0											
6/17/2023 22:17	0	0										
6/17/2023 22:17	0	0										
6/17/2023 22:17	0	0	0									
6/17/2023 22:17	0	0	1									
6/17/2023 22:17	0	0	1	0								
6/17/2023 22:17	0	0	1	0								
6/17/2023 22:17	0	0	1	0								
6/17/2023 22:17	0	0	1	0	3450	3450						
6/17/2023 22:17	0	0	1	0	3450	3450	0					
6/17/2023 22:17	0	0	1	0	3450	3450	3					
6/17/2023 22:17	0	0	1	0	3450	3450	32					
6/17/2023 22:17	0	0	1	0	3450	3450	325					
6/17/2023 22:17	0	0	1	0	3450	3450	3250					
6/17/2023 22:17	0	0	1	0	3450	3450	3250	0				
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3				
6/17/2023 22:17	0	0	1	0	3450	3450	3250	34				
6/17/2023 22:17	0	0	1	0	3450	3450	3250	345				
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450				
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450				
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28
6/17/2023 22:17	0	0	1	0	3450	3450	3250	3450	28	28	27	28

Figure 13 Recorded CSV Files from DAQ GUI

Figure 13 shows the recorded CSV files obtained from DAQ GUI. It can be seen that DAQ GUI needs to wait for some data transmission before it can record the data from Master BMS completely. It is important to notice that the stored data is raw data acquired from serial communication with Master BMS. Further conversion to the actual voltage and temperature values has not been implemented yet on the DAQ GUI.

5. Conclusion

From this work, we can understand some key points:

- The software of Master-Slave BMS has been implemented on Arduino. The Master BMS will collect data from all cells and then decide which cells are required to be balanced. The slave BMS will receive the balancing request from the Master BMS and will dissipate power through power resistors if the balancing is required.
- The voltage and temperature monitoring is also performed by Master-Slave BMS. If the maximum and minimum voltage operating range is violated, the corresponding indicator will be activated by the BMS. In the case of temperature threshold violation, no action has been implemented yet, because the relay to disconnect the load from the battery pack has not been installed on the BMS.
- To record the balancing state, voltage, and temperature of all cells, DAQ GUI has been implemented on LabView. The recorded data can be stored in CSV files to be analyzed afterwards. It is important to note that the recorded data is raw data acquired from serial communication with Master BMS.
- To ensure synchronization between Master BMS and PC, Timer1 interrupt is utilized. However, the DAQ GUI requires a couple of instances of data transmission to record the data properly.
- All software implementation in this work has not been tested on the actual Master-Slave BMS. Testing and improvement of the Arduino software and DAQ GUI are advised before incorporating the BMS into the actual formula student car.

REFERENCE

[1] J. Qi and D. Dah-Chuan Lu, “Review of battery cell balancing techniques,” in *2014 Australasian Universities Power Engineering Conference (AUPEC)*, Sep. 2014, pp. 1–6. doi: 10.1109/AUPEC.2014.6966514.

[2] M. Daowd, N. Omar, P. Van den Bossche, and J. Van Mierlo, “A Review of Passive and Active Battery Balancing based on MATLAB/Simulink,” Journal of International Review of Electrical Engineering (IREE), vol. 6, pp. 2974–2989, Nov. 2011.

[3] Lucian Andrei Perişoară, “Laboratory 4. Master-Slave BMS for 4S1P LiFePO4 battery pack”, Battery Management Systems and Battery Life Cycle Course, May 2023.