
IEEE
CAIRO UNIVERSITY STUDENT BRANCH
IEEE CUSB

ieee**cusb**

# Embedded AVR

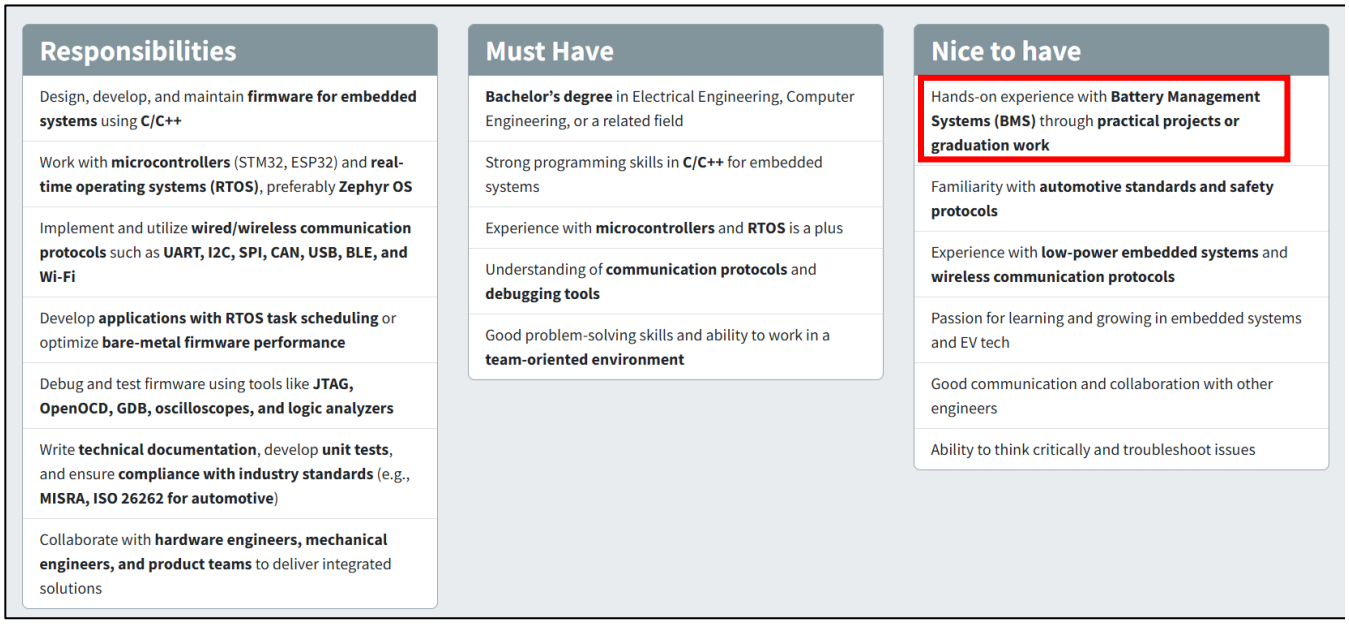
## *C Mini-Project*

**2024-2025**

IEEE CAIRO UNIVERSITY STUDENT BRANCH

# Battery Management Systems

A **Battery Management System (BMS)** is an electronic system that manages and monitors rechargeable batteries, ensuring their safe, efficient, and reliable operation. It plays a critical role in modern applications such as electric vehicles (EVs), renewable energy storage systems, portable electronics, and industrial equipment. The primary purpose of a BMS is to protect the battery from damage, extend its lifespan, and maintain optimal performance.



| Responsibilities | Must Have | Nice to have |
|---|---|---|
| Design, develop, and maintain **firmware for embedded systems** using **C/C++** | **Bachelor's degree** in Electrical Engineering, Computer Engineering, or a related field | Hands-on experience with **Battery Management Systems (BMS)** through **practical projects or graduation work** |
| Work with **microcontrollers** (STM32, ESP32) and **real-time operating systems (RTOS)**, preferably **Zephyr OS** | Strong programming skills in **C/C++** for embedded systems | Familiarity with **automotive standards and safety protocols** |
| Implement and utilize **wired/wireless communication protocols** such as **UART, I2C, SPI, CAN, USB, BLE, and Wi-Fi** | Experience with **microcontrollers** and **RTOS** is a plus | Experience with **low-power embedded systems** and **wireless communication protocols** |
| Develop **applications with RTOS task scheduling** or optimize **bare-metal firmware performance** | Understanding of **communication protocols** and **debugging tools** | Passion for learning and growing in embedded systems and EV tech |
| Debug and test firmware using tools like **JTAG, OpenOCD, GDB, oscilloscopes, and logic analyzers** | Good problem-solving skills and ability to work in a **team-oriented environment** | Good communication and collaboration with other engineers |
| Write **technical documentation**, develop **unit tests**, and ensure **compliance with industry standards** (e.g., **MISRA, ISO 26262 for automotive**) | | Ability to think critically and troubleshoot issues |
| Collaborate with **hardware engineers, mechanical engineers, and product teams** to deliver integrated solutions | | |

*Figure 1 Junior Embedded Software Engineer Job Opening at EVRaid*

# Project Objectives

The objectives of this project are:

1. **Monitor** the voltage, temperature, and state of charge (SoC) of each cell in a battery pack.
2. **Balance** the cell voltages to ensure uniform performance.
3. **Protect** the battery pack from overvoltage, undervoltage, and overtemperature conditions, through an alarming action.
4. **Display** the status of each cell in real-time.
5. **Ensure** the system is modular and scalable for different battery configurations.
6. **Practice** C programming language through a real-life automotive application.
7. **Learn** the basic functions of the battery management systems.

# *Project Structure*

The project consists of the following components:

- **Header File (BMS.h)**: Contains constants, type definitions, and function declarations.
- **Source File (BMS.c)**: Implements the functions for monitoring, balancing, and displaying cell status.
- **Main File (main.c)**: Contains the main function to demonstrate the BMS functionality.

# Functions Description

1. **initCells**
   - **Purpose**: Initialize the battery cells with user input.
   - **Input**: Array of Cell structures and the number of cells.
   - **Process**:
     - Prompt the user to enter voltage, temperature, and state of charge (SoC) for each cell, simulating sensor input.
     - Set the initial status of each cell to OK.

2. **monitorCells**
   - **Purpose**: Monitor and update the status of each cell.
   - **Input**: Array of Cell structures and the number of cells.
   - **Process**:
     - Check if the voltage is within the safe range (MIN_VOLTAGE to MAX_VOLTAGE).
     - Check if the temperature is within the safe range (MIN_TEMPERATURE to MAX_TEMPERATURE).
     - Update the status of each cell based on the checks.

3. **balanceCells**
   - **Purpose**: Balance the cell voltages to ensure uniformity.
   - **Input**: Array of Cell structures and the number of cells.
   - **Process**:
     - Calculate the average voltage of all cells.
     - Adjust the voltage of each cell to bring it closer to the average.
     - Ensure the voltage does not exceed MAX_VOLTAGE or fall below MIN_VOLTAGE.

4. **isBalanced**
   - **Purpose**: Check if the cells are balanced within a threshold.
   - **Input**: Array of Cell structures and the number of cells.
   - **Process**:
     - Calculate the average voltage of all cells.
     - Check if the voltage of each cell is within the BALANCE_THRESHOLD of the average.
     - Return 1 if balanced, 0 otherwise.

5. **displayCellStatus**
   - **Purpose**: Display the status of each cell.
   - **Input**: Pointer to a Cell structure and the cell index.
   - **Process**:
     - Print the voltage, temperature, SoC, and status of the cell.

# Notes

- Ensure that the project adheres to the specified configurations and utilizes the defined data types consistently throughout the implementation. This includes maintaining the constants, structures, and enumerations as shown

```c
#define NUM_CELLS 2
#define MAX_VOLTAGE 4.2
#define MIN_VOLTAGE 3.0
#define MAX_TEMPERATURE 60.0
#define MIN_TEMPERATURE 0.0
#define BALANCE_THRESHOLD 0.05 // Voltage difference threshold for balancing
#define NOT_BALANCED 0
#define BALANCED 1

typedef enum {
    OK,
    OVER_VOLTAGE,
    UNDER_VOLTAGE,
    OVER_TEMPERATURE,
    UNDER_TEMPERATURE
} CellStatus;

typedef struct {
    float voltage;
    float temperature;
    float stateOfCharge;
    CellStatus status;
} Cell;
```

**Hint for isBalanced Function**

- Use the **fabs function** from the **math.h library** to calculate the absolute difference between each cell's voltage and the average voltage.