the TJA1050T transceiver along with a logic level shifter. The RFID system operates at a 13.56MHz frequency, and for debugging purposes, we utilize RS485.

3.2.4 Firmware Development

We selected the STM32F103C8T6 [datasheet] as the MCU for our BMS board because it offers all the required peripherals CAN, UART, and I2C,while remaining cost-effective. It also has strong library support, making firmware development easier. Since each battery pack requires a separate BMS, minimizing the cost per unit is essential. Compared to other STM32 variants, this MCU provides a good balance of performance, features, and affordability.

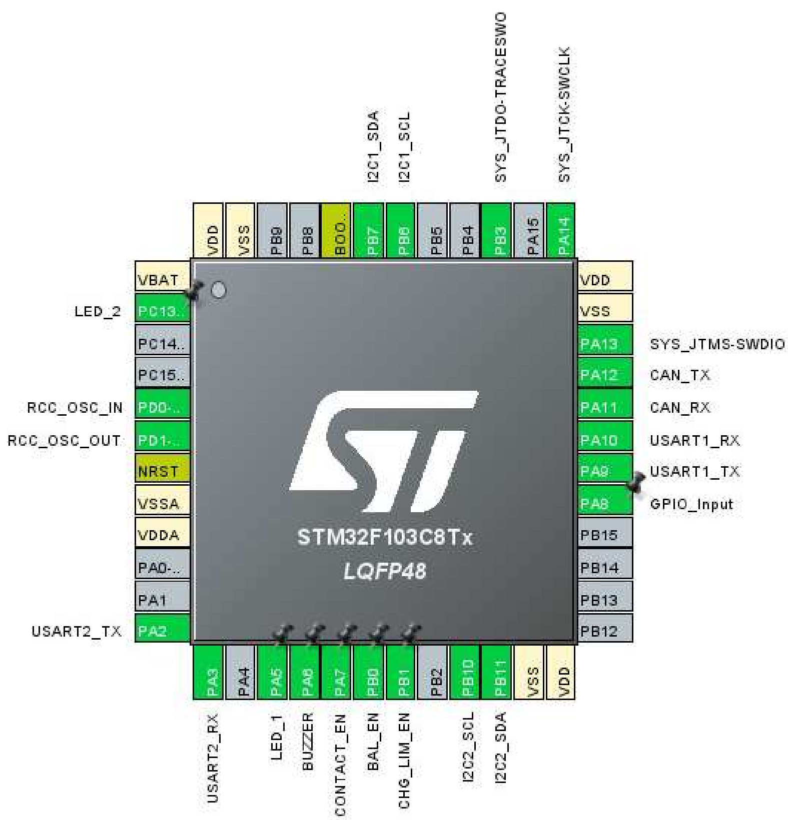

Figure 3.34: STM32F103 Pinout and Connections setup for BMS using STM32CubeIDE

Table 3.1: STM32F103C8T6 Peripheral Configuration

Peripheral	Configuration	Remarks
HCLK	72 MHz (max)	System clock
CAN	Baud Rate: 500,000 bit/s	Communication with Central Unit
USART1	Baud Rate: 1,000,000 bit/s Word Length: 8 bits (incl. parity) Parity: None Stop Bits: 1 RX: Pull-up enabled	Communication with Monitoring IC
USART2	Baud Rate: 115,200 bit/s Word Length: 8 bits (incl. parity) Parity: None Stop Bits: 1	Debugging and UI Communication
I2C1	Standard Mode Clock Speed: 100 kHz	EEPROM communication
I2C2	Standard Mode Clock Speed: 100 kHz	Fuel Gauge communication

Table 3.2: STM32F103C8T6 GPIO Configuration

GPIO Pin	Default State	Function
Active Buzzer	Pull-down	Audio alert
Contactor Enable	Pull-down	Main power relay control
Active Balancer Enable	Pull-up	Balance circuit control
Charge Limiter Enable	Pull-down	Controls charging circuit
2x LEDs	Output	Status indication
SWD (Trace Async)	Debug interface	Programming and debugging

The BMS firmware was developed with a primary focus on authentication, protection, and data communication with the central unit. In real-world deployment, troubleshooting is often handled by technicians who require a simple and effective way to monitor and configure BMS parameters. It was design as 4 Main States State Machine and Controlling others using Interrupts. To support this, a Windows-based user interface application was developed along with the firmware, providing an intuitive interface for diagnostics, configuration, and visualization of real-time data.

1.)Main State Machine

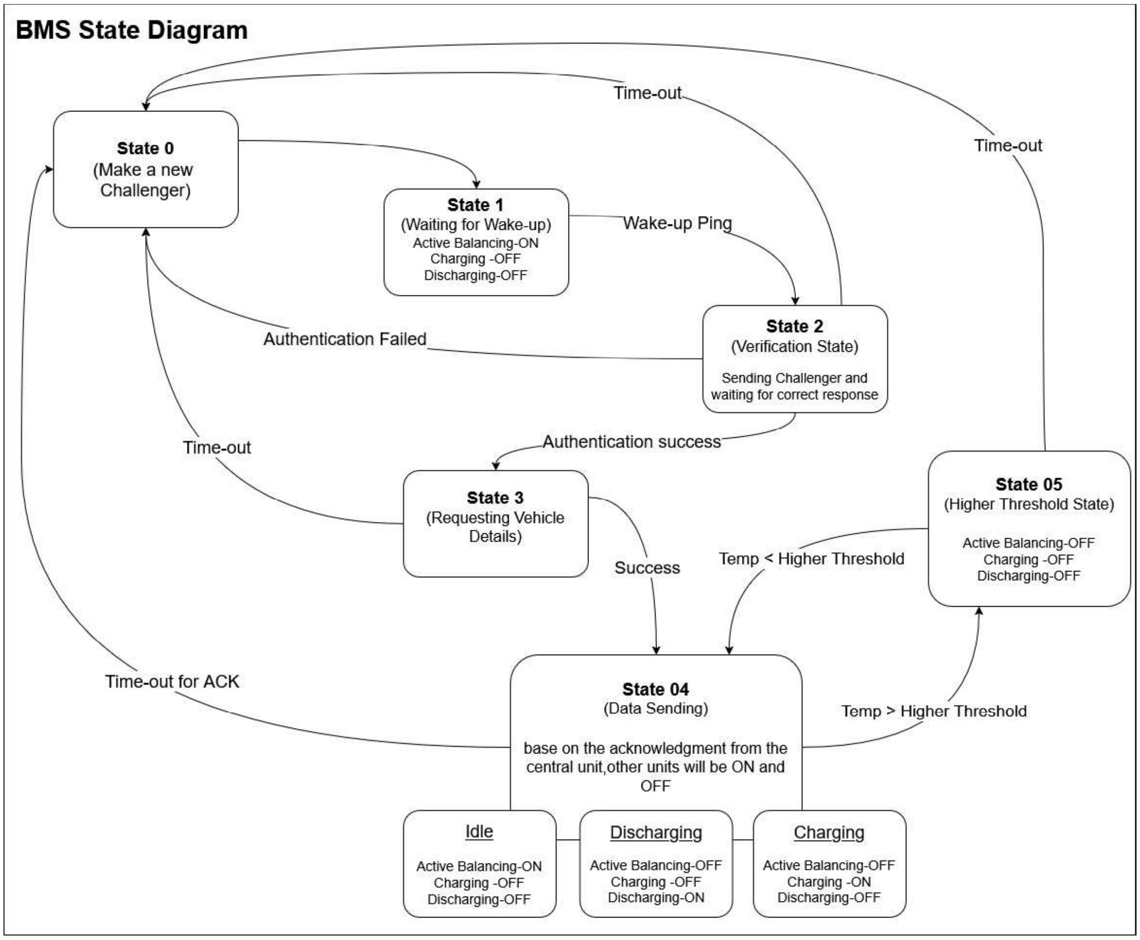

Figure 3.35: State Machine

2.) Custom Communication protocols

A custom communication protocol with structured frames was developed to enable efficient data exchange and scalable system integration.

CAN communication

Communication between the BMS and the central unit is handled using Standard CAN identifiers, which use 11-bit IDs, allowing for a total of 2048 unique CAN IDs. To transmit different types of data, each message type is assigned a dedicated CAN ID, and each BMS is also given a unique CAN ID. Approximately 10 CAN IDs are reserved for special or system-level messages, leaving around 2038 IDs available for BMS devices. While two BMS units may technically share the same CAN ID without functional issues, it becomes problematic when both are present in the same vehicle, especially when displaying data on the vehicle dashboard. To support vehicle-level targeting, each BMS also stores a separate unique identifier in EEPROM.

Table 3.3: Custom CAN Headers Used in BMS Communication

CAN Header	Purpose
AUTH_CAN_HEADER	Transmits BMS authentication Challenger
DATA_CAN_HEADER	Transmits Real time data
ACK_CAN_HEADER	Receiving Acknowledges commands
CENTRAL_DETAILS_CAN_HEADER	Requesting Vehicle or Docking Station Details
WARNINGS_CAN_HEADER	Sends warning or fault messages

Unique BMS CAN ID is used in these communication scenarios:

- Receiving the digest of the authentication challenge
- Receiving central unit details (e.g., Vehicle ID or Docking Station ID)
- Tagging real-time data frames
- Tagging warning message frames

Custom CAN Data Frame Send by BMS

The CAN protocol allows a maximum of **8 bytes** of data per message. We designed a custom data frame format to efficiently transmit key BMS parameters as floating-point values.

Table 3.4: Structure of 8-Byte Custom CAN Data Frame

Bytes	Field	Size	Description
0–1	Data Type Identifier	2 bytes	Specifies the type of data
2–3	BMS ID	2 bytes	Unique identifier for the BMS unit
4–7	Data Payload	4 bytes	Data value in IEEE 754 float format

This structure allows for consistent interpretation and scaling across multiple BMS units.

Data Type Identifiers:

Table 3.5: Custom Data Type Identifiers for CAN Messages

Hex Code	Data Type
0x01	Total Voltage
0x02	Total Current
0x03	Average Temperature
0x04	State of Charge (SOC)
0x05	State of Health (SOH)

Custom CAN Warning message Frame Send by BMS

This frame is used to send warning messages from the BMS to the Central Unit.The 8 Bytes of CAN message is divided as above.

Table 3.6: Structure of Custom CAN Warning Message Frame

Byte(s)	**Description**
First 2 bytes	"WN" identifier (Warning Message)
Second 2 bytes	BMS ID
Last 4 bytes	Warning Message Code

Warning Message Codes:

- WHVT – Total voltage exceeds the upper voltage threshold
- WLVT – Total voltage below the lower voltage threshold
- WHVD – Voltage difference exceeds the warning threshold
- WHTD – Temperature exceeds disconnection threshold
- WHTW – Temperature exceeds warning threshold
- WLSC – SOC below the minimum threshold
- WLSH – SOH below the minimum threshold

3.) Authentication

The BMS is restricted to operate in either charging or discharging mode only while it is connected to the central unit. This design ensures that batteries cannot be discharged by unauthorized users or charged using third-party chargers instead of the official docking station.

To enforce authentication, an HMAC-SHA256 based challenge response mechanism is used. A secret HMAC key is securely stored in each BMS and central unit. When the BMSs connects to the central unit via the CAN bus, it initiates authentication by sending a randomly generated 4-byte challenge number.Upon receiving the challenge, the central unit computes a digest using the shared HMAC key and responds via the BMS's unique **bms ID**.To protect against replay attacks, a new random challenge is generated for each authentication session.

SHA-256 is a computationally intensive algorithm that involves complex mathematical operations. Implementing such an algorithm directly on an embedded system can be resource-demanding. To address this, we use the TinyCrypt Cryptographic library, which is specifically designed to provide cryptographic functions optimized for low-resource embedded systems. This allows us to perform SHA-256 hashing efficiently within the constraints of the BMS microcontroller.

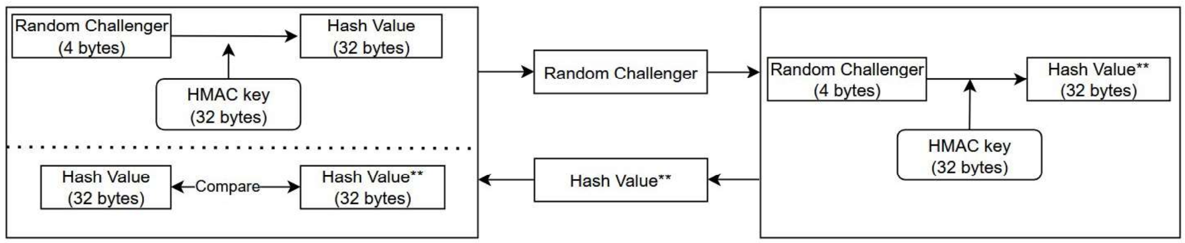

Figure 3.36: Authentication using SHA-256

4.) Protections

Protecting the battery cells is one of the primary responsibilities of the BMS microcontroller. Although the monitoring IC provides overvoltage (OV) and overtemperature (OT) flag mechanisms, the MCU is used to relay these flags to the central unit, giving us greater control and flexibility in handling fault conditions.

Warning messages are sent to the central unit under the following conditions:

- Over Voltage
- Under Voltage
- Over Current
- High Cell Voltage Imbalance
- Low State of Charge (SOC)
- Low State of Health (SOH)

Temperature protection is handled with two thresholds:

- If the temperature exceeds the first threshold, a warning message is sent.
- If it exceeds the second threshold, the BMS stops charging or discharging to ensure battery safety.

The over voltage warning helps to terminate charging, while the under voltage alert notifies the user to replace or recharge the battery. The threshold values can be configured using the Windows application, allowing the BMS to be easily adapted for different types of battery cells.

Discharging from the BMS occurs through a contactor, which is controlled by the BMS MCU. The contactor is only enabled when the BMS is connected to the central unit. This design ensures that the user does not have access to the output voltage while transporting the battery pack between the docking station and the vehicle, thereby enhancing safety.

5.) Non-Volatile Memory

We used a 256KB EEPROM to store essential configuration data and user logs. The table below illustrates how memory pages and addresses are allocated.

Page 01 – Device Information (Start Address: `0x0000`)

Variable	Address	Size (Bytes)
BatteryID	0x0000	20
FirmwareVersion	0x0014	4
CellType	0x0018	6

Page 02 – Configuration Parameters (Start Address: `0x0040`)

Variable	Address	Size (Bytes)
MaxVoltageDiff	0x0040	4
MaxVoltage	0x0044	4
MinVoltage	0x0048	4
WarningTemperature	0x004C	4
DisconnectTemperature	0x0050	4
MaxCurrent	0x0054	4
MinSOH	0x0058	4
MinSOC	0x005C	4

Page 03 – Reserved (Start Address: `0x0060`)
Skipped 96 bytes for future use

Page 04 – Program Variables (Start Address: `0x00A0`)

Variable	Address	Size (Bytes)
Log_Index	0x00A0	1
Latest User Log	0x00A2	8

Page 05 – User Logs (Start Address: `0x00E0`)

Variable	Address	Size (Bytes)
USER_LOG[0]	0x00E0	8
USER_LOG[1]	0x00E8	8
USER_LOG[2]	0x00F0	8
USER_LOG[3]	0x00F8	8
USER_LOG[4]	0x0100	8
USER_LOG[5]	0x0108	8
USER_LOG[6]	0x0110	8
USER_LOG[7]	0x0118	8

In this firmware version, only the most recent 8 user logs are stored. However, the system can be expanded to store additional logs if required.

We chose STM32F446RE [datasheet] as the microcontroller for the Central Unit, primarily due to its support for two independent CAN peripherals, which are essential for simultaneous communication with both the BMS and the vehicle system. Compared to other STM32 MCUs, the STM32F446RE offers higher performance with a 180 MHz Cortex-M4 core, floating-point unit (FPU) support, and larger memory options—making it well-suited for more demanding control and communication tasks.

ESP32 microcontroller choose for both the adapter board and the battery interface unit at the docking station, primarily due to its built-in Wi-Fi and Bluetooth capabilities. This allows seamless internet connectivity for real-time access to the cloud database, enabling updates of user information and State of Charge (SoC) levels for each battery. The ESP32 also offers sufficient processing power and peripheral support for handling communication and control tasks efficiently.

3.2.5 Windows Application for BMS Interface

To simplify diagnostics, configuration, and real-time monitoring, we developed a Windows application for the BMS. Manually modifying firmware parameters for different battery types or troubleshooting scenarios is time-consuming and error-prone; this application addresses that challenge by providing a user-friendly interface.

The application is organized into **three main tabs**:

- **Real-Time Data:** Displays live values from the BMS, including individual cell voltages, cell temperatures, current, State of Charge (SOC), and State of Health (SOH).
- **Device Info:** Shows essential device information such as the BMS ID, firmware version, cell type, and logs of the last users.
- **Configuration:** Allows users to set threshold levels for warnings (e.g., voltage limits, temperature thresholds, SOC, and SOH).

To use the application, connect the BMS to a PC via a USB-to-UART converter, wired to the USART2 port of the STM32. After selecting the correct COM port and pressing the `Connect` button in the application, communication is established. All data exchanges that are both reading and writing, are performed using a custom communication protocol.

UI → BMS: Read Requests

Command	Description	Format
DINFO	Request Device Information	ASCII string
RDATASTART	Start real-time data transmission	ASCII string
RDATASTOP	Stop real-time data transmission	ASCII string
GCONFIG	Request Configuration Data	ASCII string

Table 3.7: Read Requests from UI to BMS

BMS → UI: Read Responses

Field	Example	Notes
FMVERSION	FMVERSION:1.0.0	Firmware version
BMSID	BMSID:LFP16-2025/04/21-010	BMS version
CELLTYPE	CELLTYPE:LFP	Lithium type (e.g., LFP, NMC)
USERLOGS	USERLOGS:U1=Log1;...;U10=Log10	User log entries
BALANCESTATE	BALANCESTATE:ON	Balancing state (ON/OFF)
CHARGESTATE	CHARGESTATE:CHARGING	Charging status
DISCHARGESTATE	DISCHARGESTATE:ENABLED	Discharge status

Table 3.8: Device Info Fields

Device Information Format **BMS ID Structure:**

- First segment (e.g., `LFP16`): Battery chemistry and number of cells
- Second segment: Manufacturer data
- Third segment: Unique identifier similar to CAN header for the BMS

Field	Example	Description
TV	TV:48.76	Total Voltage (V)
TC	TC:-12.34	Total Current (A)
BTS	BTS:T1=100;...;T6=200	Battery temperature sensors
CV1	CV1:C1=2.45;...;C8=1.00	Cell voltages (Cells 1–8)
CV2	CV2:C9=2.45;...;C16=1.00	Cell voltages (Cells 9–16)

Table 3.9: Real-Time Data Fields

Real-Time Data Format **Note:** All responses are newline-terminated (`\n`). Values are in float format unless otherwise specified.

Configuration Data

Read Requests

- TWT: `temp_warning_threshold`
- TDT: `temp_disconnecting_threshold`
- VHT: `voltage_higher_threshold`
- VLT: `voltage_lower_threshold`
- CHT: `current_higher_threshold`
- SLT: `soc_lower_threshold`
- VDT: `voltage_different_threshold`

Write Requests

- `STWT:{value:.2f}` – Set temperature warning threshold
- `STDT:{value:.2f}` – Set temperature disconnecting threshold
- `SVHT:{value:.2f}` – Set voltage higher threshold
- `SVLT:{value:.2f}` – Set voltage lower threshold
- `SCHT:{value:.2f}` – Set current higher threshold
- `SSLT:{value:.2f}` – Set SOC lower threshold
- `SVDT:{value:.2f}` – Set voltage difference threshold
- `SHLT:{value:.2f}` – Set SOH lower threshold (not yet implemented)

3.3 Techniques and Tools

3.3.1 Draw.io

For system architecture and block diagram design, we use draw.io, a versatile and user-friendly tool for creating detailed schematics and flowcharts. It offers a wide range of shapes, templates, and collaboration features, making it ideal for documenting system designs clearly and efficiently.

3.3.2 Altium Designer

For PCB design, we use Altium Designer which provides advanced tools for schematic capture, layout, and simulation. Altium 365 cloud-based collaboration and version control features allow multiple team members to work seamlessly on designs, ensuring efficient workflow management and real-time updates.

3.3.3 LTSpice

LTspice is a powerful and widely used circuit simulation software developed by Analog Devices. It is a SPICE-based simulator that allows engineers to model, test, and optimize analog and power electronics circuits before hardware implementation. LTspice provides an extensive library of semiconductor models, passive components, and switching regulators, making it ideal for simulating complex power electronics systems. In our project, we used LTspice to simulate the boost and flyback converters in our DC-DC converter-based active cell balancer, using controller ICs from Analog Devices. Additionally, we simulated a buck converter for our active charging limiting circuit, ensuring its efficiency and performance before prototyping.