Battery Management System for Electric Vehicles

by

T.Jagadamba Saraswathi
(229X1A0281)

Under the Esteemed Guidance of

Dr. K. Sri Gowri ***M.Tech., Ph.D., MIEEE***
Professor

Department of Electrical & Electronics Engineering
G. Pulla Reddy Engineering College (Autonomous): Kurnool
(Affiliated to Jawaharlal Nehru Technological University-Anantapur, Ananthapuramu)

ABSTRACT

Electric vehicles rely on high-capacity lithium-ion battery packs that require careful monitoring and management to ensure safety, longevity, and optimal performance. A Battery Management System is essential for such applications. This project presents the design and implementation of a BMS for an EV battery pack, including real- time State of Charge estimation. The BMS monitors key battery parameters - voltage, current, temperature and provides protection against over-voltage, under- voltage, and over-temperature conditions. SOC is estimated using a combined coulomb- counting and voltage-based algorithm, implemented on a microcontroller (PIC16F72). The system hardware includes battery sensors, signal conditioning circuits, and a display interface; the software reads sensor data, computes SOC, and triggers alarms or balancing as needed. The prototype was tested under various load conditions. Results demonstrate accurate monitoring of battery status, reliable SOC estimation within ~5% error and effective protective actions. The work verifies that the developed BMS can enhance EV battery Safety and Management.

Keywords: Battery Management System , State of Charge , Electric Vehicles, Battery Monitoring, Microcontroller.

CHAPTER 1 INTRODUCTION

1.1 Background:

Renewable energy adoption is driving demand for electric vehicles as an eco- friendly transport solution. EVs typically use high-energy-density lithium-ion battery packs. Such batteries must be carefully managed to prevent overcharging, deep discharging, and overheating. A BMS is therefore essential in EVs. The BMS monitors battery voltage, current, and temperature, ensuring the pack operates within safe limits. It also

performs functions like cell balancing and calculates the *SOC* to estimate remaining battery capacity. Proper battery management enhances performance, prolongs battery life, and improves overall vehicle safety.

1.2 Motivation / Problem Statement:

Lithium-ion batteries, while efficient, are sensitive to operating conditions. Without a BMS, cells could be damaged by imbalance or unsafe conditions, leading to reduced lifespan or hazards. In EVs, accurate SOC information is crucial for range prediction and power management. Many commercial EVs use proprietary BMS solutions, but academic knowledge is valuable for understanding and innovation. The motivation of this project is to develop a cost-effective BMS prototype for EV battery packs, focusing on hardware implementation and SOC estimation, to study its performance and limitations.

1.3 Aim of the Project:

- To design and implement a Battery Management System for a lithium-ion battery pack used in electric vehicles, and to develop a real-time State of Charge estimation algorithm**.**

1.4 Objectives:

- Implement hardware for battery monitoring voltage, current, temperature using microcontrollers.
- Develop and integrate an SOC estimation algorithm coulomb-counting plus voltage- based correction.
- Ensure battery protection by triggering alarms for over-voltage, under-voltage, and temperature limits.
- Display battery parameters and SOC on an LCD or other interface.
- Test and validate the system under various load conditions and record results.

1.5 Scope of the work:

This project covers a single-cell or small battery pack BMS prototype intended for educational demonstration. It includes hardware design sensors, microcontroller, power supply, alarms, display and software data acquisition, SOC computation, safety logic. It does not cover high-current charging circuitry or advanced cell balancing hardware only simple passive balancing or alerts. The project focuses on the core monitoring and SOC functions.

1.6 Methodology

The approach begins with studying BMS requirements and selecting components. A PIC16F72 microcontroller is chosen for control. Sensor circuits are designed for voltage, current using ACS712, and temperature (LM35). A 5V regulated supply powers the control electronics. Software is written in C using MPLAB IDE; Proteus is used for simulation. The SOC algorithm combines time integration of current coulomb counting with voltage-based correction. The hardware is assembled on a prototype board. Testing

involves applying known loads to the battery pack and measuring outputs. Data is collected and analyzed to verify SOC accuracy and safety operation.

1.7 Organization of the Report:

- Chapter 2 reviews existing research and BMS concepts.
- Chapter 3 describes the system architecture and hardware components.
- Chapter 4 details the software design and algorithms.
- Chapter 5 explains implementation steps.
- Chapter 6 presents testing procedures and results.
- Chapter 7 discusses the findings and limitations.
- Chapter 8 concludes the work and suggests future enhancements.

CHAPTER 2 LITERATURE SURVEY

2.1 Overview of Related Work:

Battery management for EVs has been widely studied. Key functions include voltage monitoring, current sensing, cell balancing, and SOC/SOH (State of Health) estimation. Many researchers focus on SOC estimation methods such as coulomb counting, voltage modeling, and Kalman filter approaches. Existing BMS architectures are either passive simple resistive balancing or active - transfer energy between cells. Hardware implementations often use microcontrollers or dedicated BMS ICs with analog front-ends.

2.2 Key Papers and Summaries:

The literature indicates two main categories: hardware BMS designs and SOC algorithms. For example, some works implement PIC-based BMS prototypes monitoring four cells with LCD displays. Others use advanced microcontrollers with CAN-bus interfaces for automotive use. In SOC estimation, one approach uses coulomb counting corrected by open-circuit voltage lookup tables voltage-SOC curves. Another employs predictive models or neural networks for improved accuracy. These are general examples; specific citations to be added as needed.

2.3Comparative Analysis:

Aspect	Passive Balancing	Active Balancing	Coulomb Counting SOC	Model- based SOC
Complexity	Low	High	Medium	High
Power Loss	High (resistive)	Low	Low (if efficient)	Low
Accuracy (SOC)	–	–	Requires calibration	Requires model
Cost	Low	High	Medium	High

- Passive balancing is simple but wastes energy as heat. Active balancing is more efficient but requires complex circuits.
- Coulomb counting gives good short-term SOC if initial charge is known, but drifts over time due to measurement error. Voltage-based methods correct drift but require accurate battery models.

2.4 Gap Identification:

Prior work often uses multiple cells and specialized BMS ICs. Fewer examples exist of simple single-chip BMS prototypes with full SOC algorithm implemented on a general- purpose microcontroller. This project addresses that gap by combining a microcontroller- based design with an implemented SOC estimation routine, suitable for a small-scale EV battery demonstration.

CHAPTER 3 Simulation Design

3.1 Battery Charging and Discharging:

A Battery is modeled as an energy storage system where electrical energy is converted into chemical energy during charging and released back as electrical energy during discharging. The simulation helps in understanding voltage, current, and state of charge variations over time.

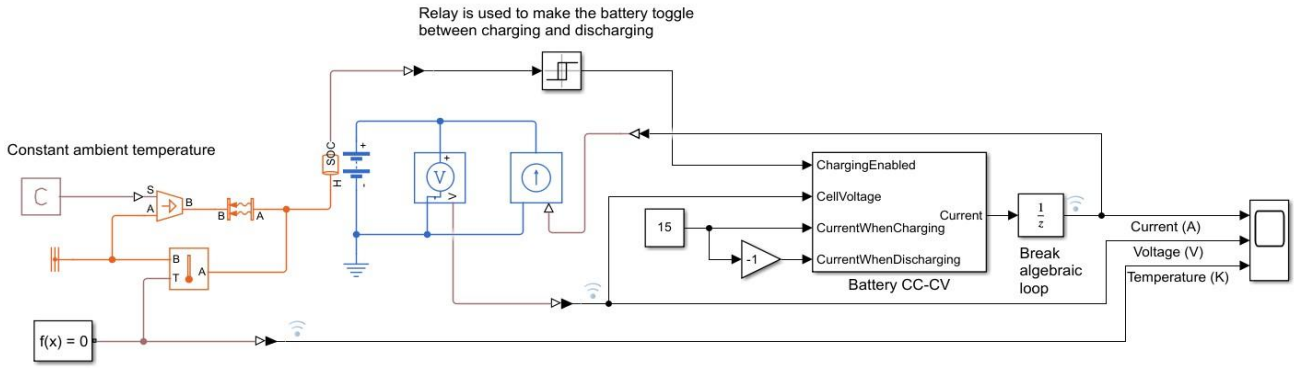

Fig 3.1.1 Block Diagram of Battery Charging and Discharging

During charging, a controlled input - constant current or constant voltage is applied, resulting in a gradual rise in battery voltage and SOC. In discharging mode, the battery supplies power to a load, causing a decrease in voltage and SOC. The model considers important parameters such as internal resistance, capacity, and efficiency**.** Using MATLAB/Simulink, charging and discharging curves are plotted to observe system behavior under different conditions

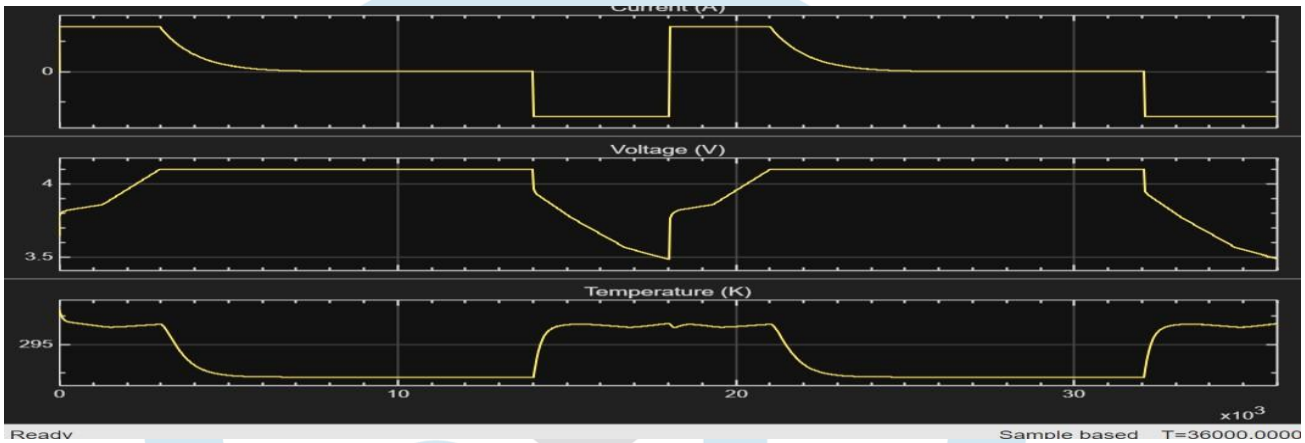

Fig 3.1.2 Output for Battery Charging and Discharging

3.2 Battery Monitoring:

Battery monitoring in this project is carried out using MATLAB to analyze key performance parameters such as voltage, current, and State of Charge. The simulation results obtained using MATLAB show the behavior of battery current with respect to defined charging and discharging limits. In the upper graph, the yellow waveform represents the battery current, which varies dynamically within the set charging limit - positive and discharging limit - negative boundaries.

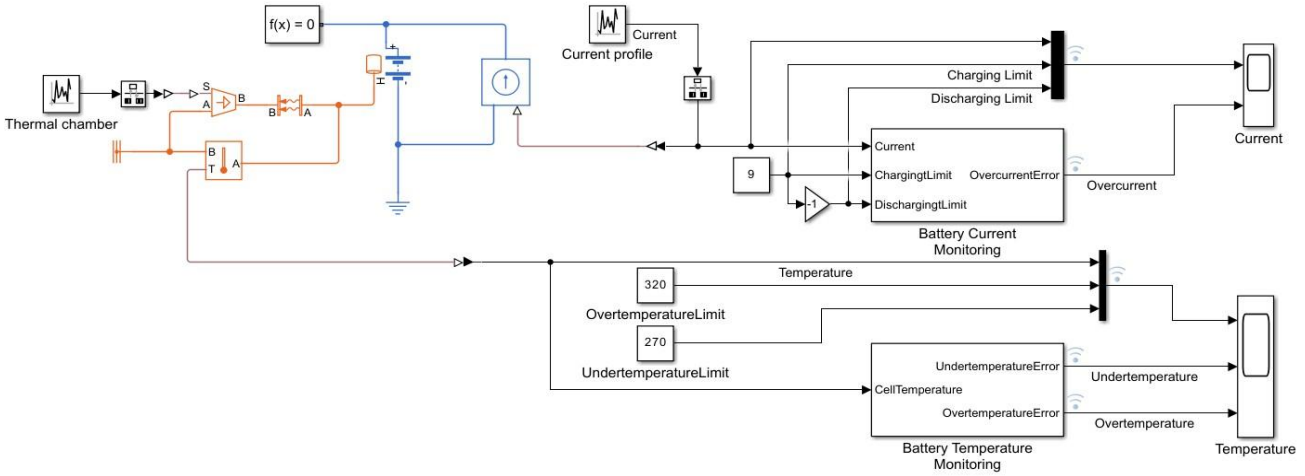

Fig 3.2.1 Block Diagram of Battery Monitoring

Whenever the current exceeds these predefined limits, an overcurrent condition is detected. This is clearly indicated in the lower graph, where sharp pulses (value = 1) represent instances of overcurrent. These spikes show that the monitoring system successfully identifies unsafe operating conditions.

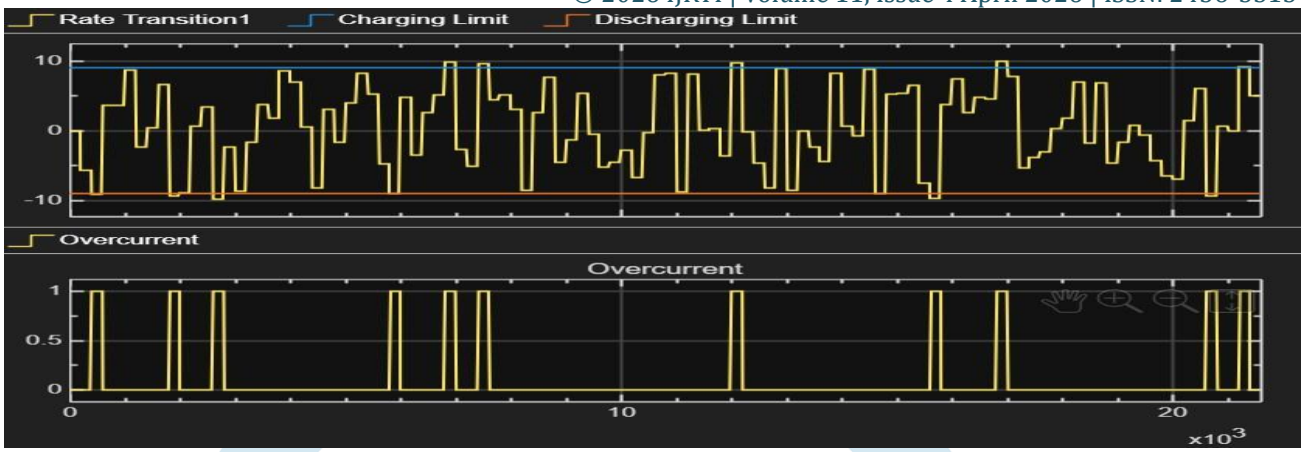

Fig 3.2.2 Output For Battery Monitoring

In the upper graph, the yellow waveform represents the battery temperature, while the upper and lower boundary lines indicate the over temperature limit and under temperature limit respectively. This is clearly shown in the lower graph, where pulses indicate over temperature and under temperature events. A value of 1 represents the occurrence of a fault, while 0 indicates normal operation. These results confirm that the monitoring system effectively detects abnormal thermal conditions.

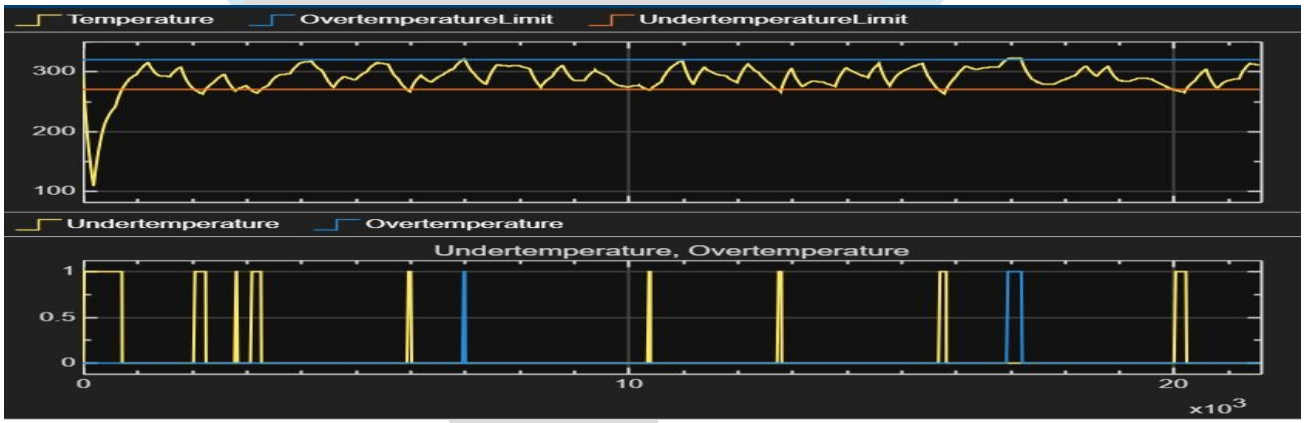

Fig 3.2.3 Output for Under Temperature and Over Temperature

From the results, it is observed that the battery temperature mostly operates within the safe range but occasionally crosses the set limits due to changing operating conditions. When the temperature exceeds the upper limit or falls below the lower limit, fault conditions are generated.

3.3 Battery Passive Cell Balancing:

Battery passive cell balancing is implemented to maintain equal voltage levels among individual cells in a battery pack. Due to manufacturing variations and unequal operating conditions, some cells may charge faster and reach higher voltages than others, leading to imbalance and reduced overall performance.

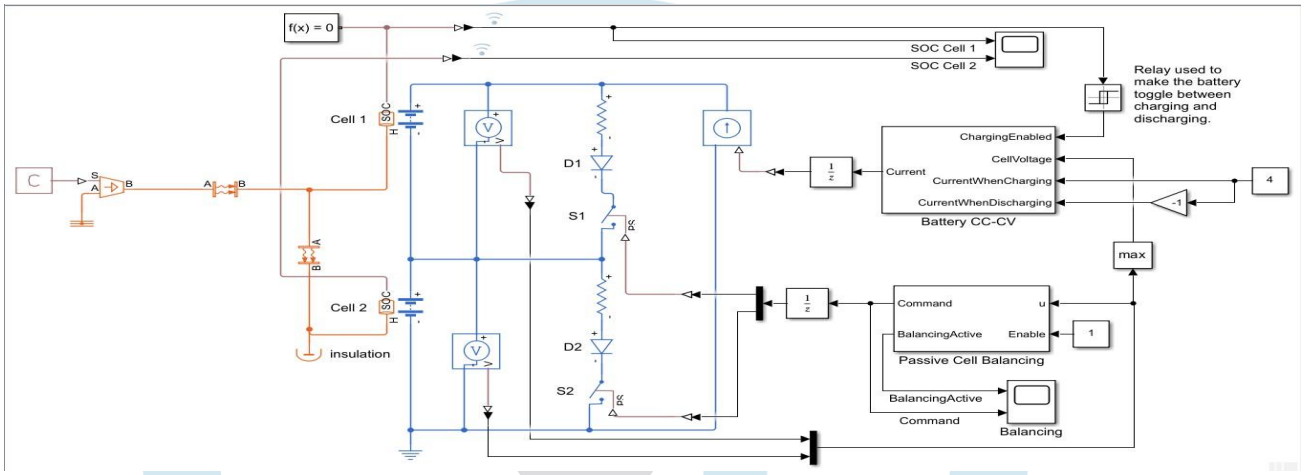

Fig 3.3.1 Block Diagram of Passive Cell Balancing

The graph 3.3.2 shows the SOC of two cells, where an initial imbalance is observed, with Cell 1 having a slightly higher SOC than Cell 2. During the charging process, both cells approach a high SOC level (~0.9), but the difference between them persists due to inherent cell variations.

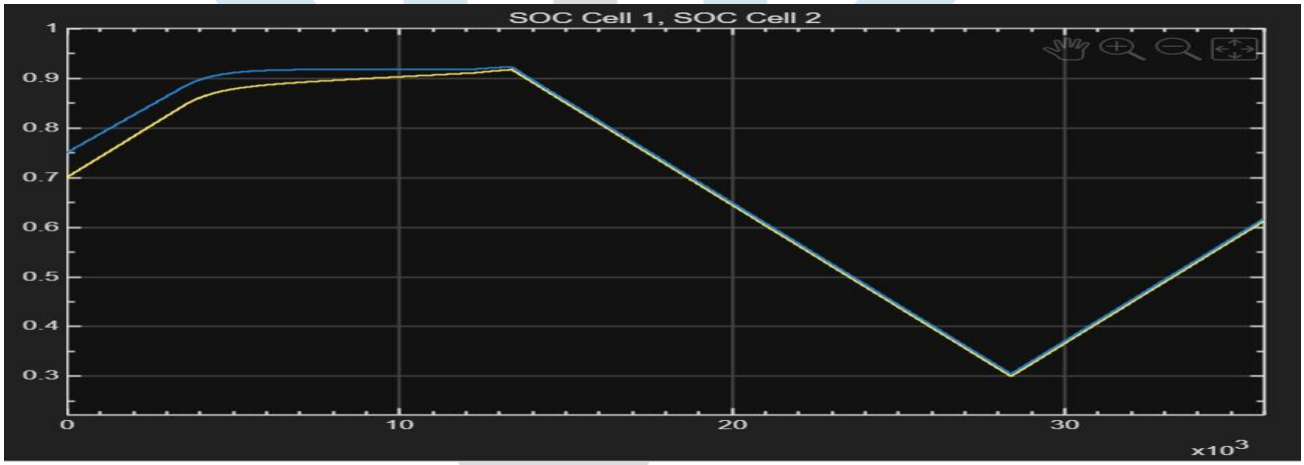

Fig 3.3.2 Waveforms of SOC of Two Cells

As the system transitions into the discharging phase, the SOC of both cells decreases almost linearly, maintaining a close profile. However, when the SOC difference exceeds a predefined threshold, the BMS activates the cell balancing mechanism. This is clearly observed in the "BalancingActive" waveform, where pulses indicate the time intervals during which balancing is enabled.

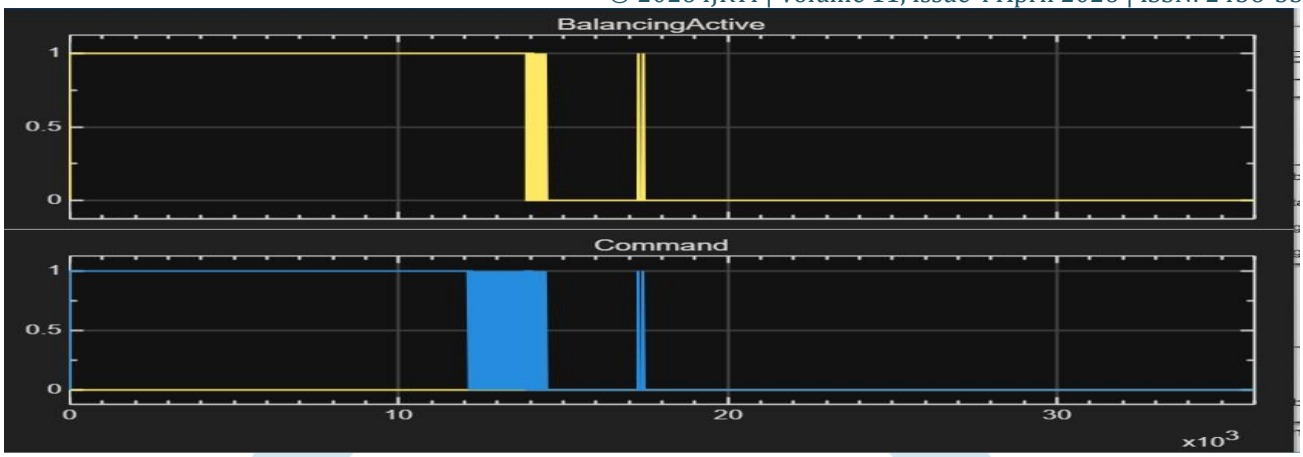

Fig 3.3.3 Balancing Active Waveform

The "Command" signal shows when the controller issues instructions to initiate balancing. During these intervals, excess charge from the higher SOC cell is dissipated in passive balancing to match the lower SOC cell. As a result, both SOC curves converge and follow a nearly identical trajectory afterward.

This behavior demonstrates that the BMS effectively monitors cell imbalance and activates balancing only when necessary, ensuring uniform charge distribution, improved battery efficiency, and enhanced lifespan of the battery pack.

3.4 Battery State-of-Charge Estimation:

Battery State of Charge (SOC) estimation is an important function of a Battery Management System (BMS), used to determine the available capacity of a battery. SOC indicates how much charge is remaining in the battery compared to its full capacity and is usually expressed as a percentage.

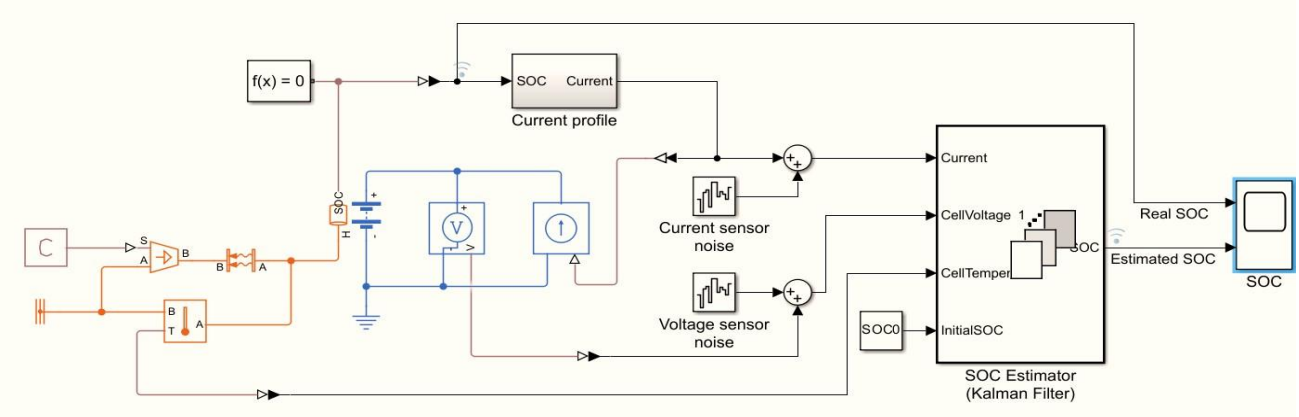

Fig 3.4.1 Block Diagram of SOC

The above waveform represents the comparison between Real SOC and Estimated SOC obtained from the Battery Management System model developed in MATLAB/Simulink. From the waveform, it can be observed that the estimated SOC closely follows the real SOC throughout the operation. During the charging

phase, both values increase gradually, reaching a peak near 0.9, and during the discharging phase, they decrease down to around 0.3. The close overlap of the two curves indicates that the estimation method used in the model is accurate and reliable

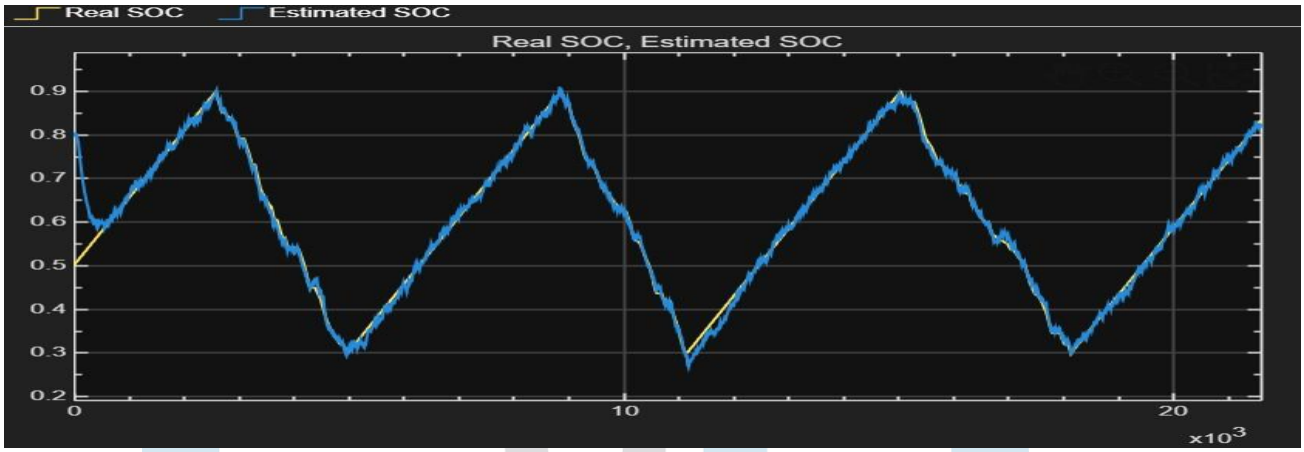

Fig 3.4.2 Output for SOC

3.5 Battery State-of-Health Estimation:

Battery State of Health estimation is an important function of a BMS used to determine the overall condition and aging level of a battery. SOH indicates how much the battery performance has degraded compared to its original condition. It is usually expressed as a percentage, where 100% represents a new battery and lower values indicate aging and performance loss.

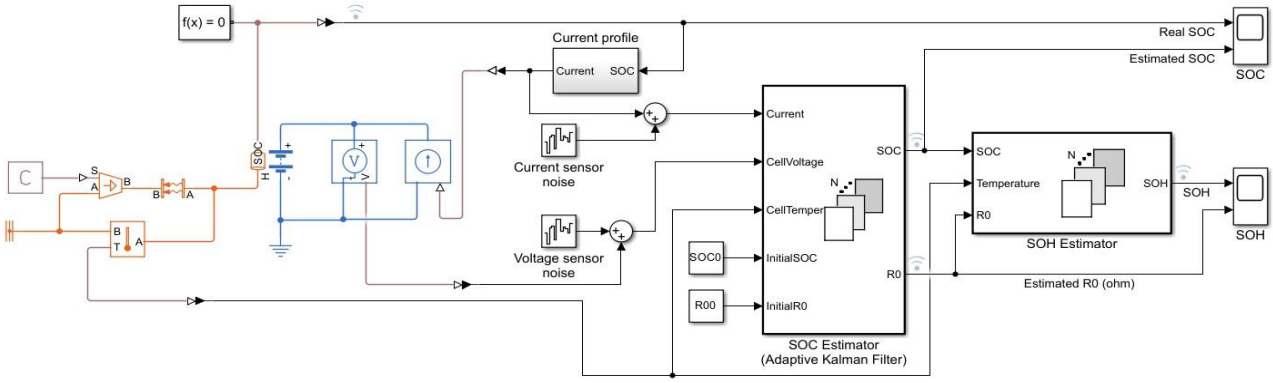

Fig 3.5.1 Block Diagram of SOH

As the battery is used over time, its capacity decreases and internal resistance increases due to chemical and physical changes. SOH estimation helps in identifying this degradation and ensures that the battery operates safely and efficiently. It also helps in predicting the remaining useful life of the battery. SOH is typically expressed as a percentage, where the initial value starts close to 100%, indicating a healthy battery condition. As the simulation progresses through multiple charge and discharge cycles, the SOH value slowly decreases, showing the aging effect of the battery.

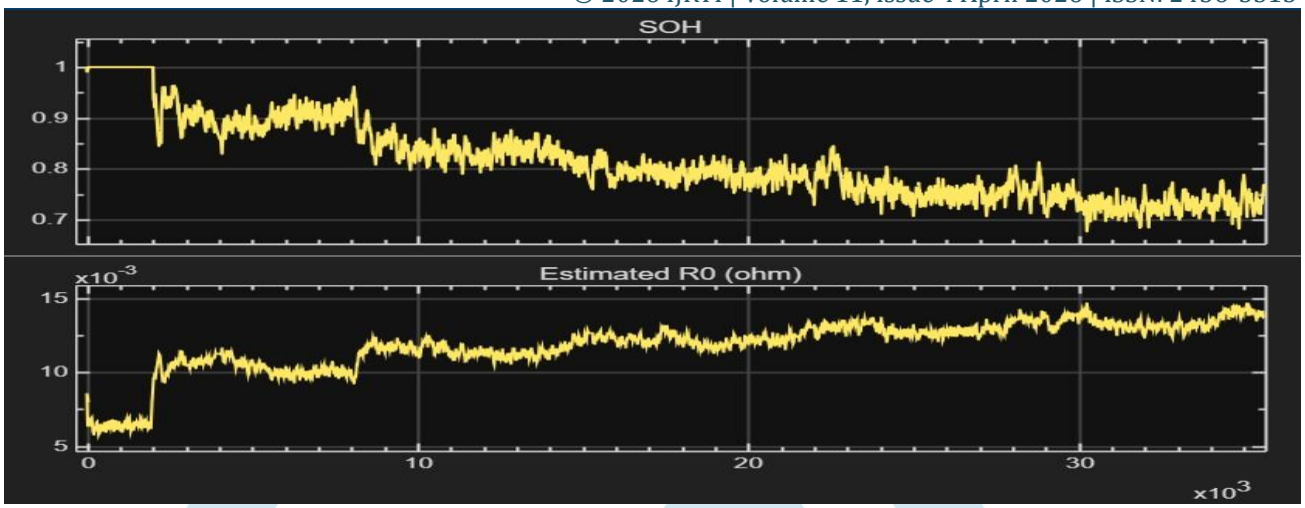

Fig 3.5.2 Output for SOH

CHAPTER 4 HARDWARE DESIGN

4.1 Block Diagram:

The BMS consists of the battery pack input, sensor circuits, a microcontroller, and output devices. A block diagram -Figure 3.1 shows: battery → voltage divider + current sensor + temperature sensor → PIC16F72 microcontroller → LCD display and alarm (LED/buzzer). The microcontroller reads the analog inputs, computes SOC, and controls outputs such as charging/discharging relays or alarms.

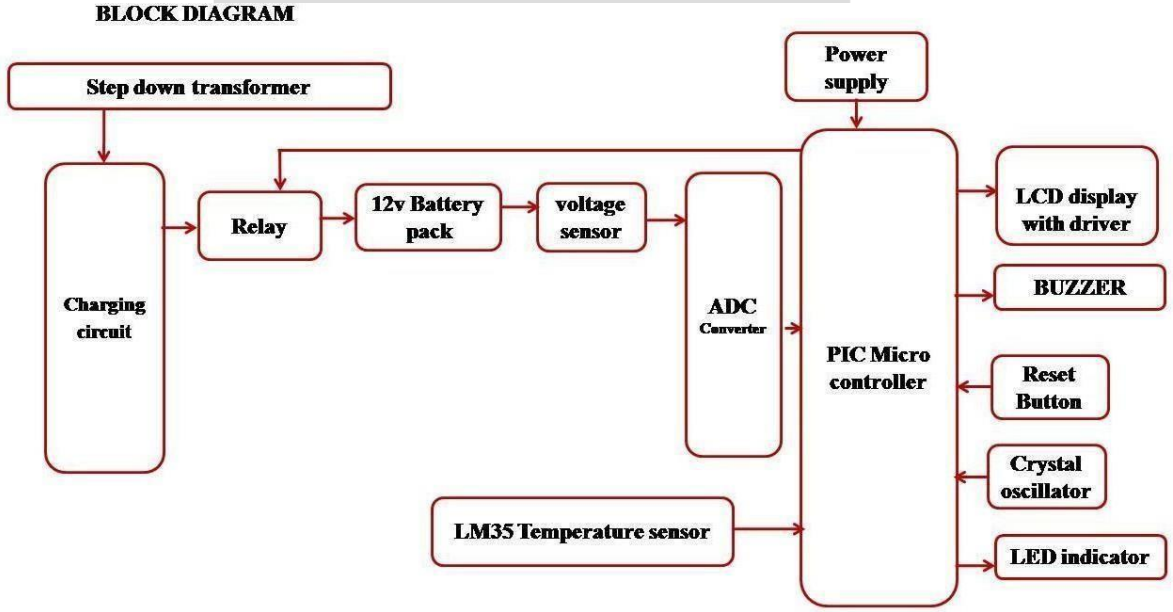

Fig 4.1 Block Diagram

4.2 Component List:

Table: Major Components

Component Specification/Model Quantity			Notes
Microcontroller PIC16F72 (8-bit MCU)		1	4-channel 10-bit ADC
Voltage Sensor Resistor Divider Network	–	Scales battery voltage to <5V ADC range	
Current Sensor ACS712 (30A range)	1	66mV/A sensitivity	

Component	Specification/Model	Quantity	Notes
Temperature Sensor	LM35 (Analog 0–100°C)	1	10 mV/°C linear output
LCD Display	16×2 Character LCD	1	Interface via 4-bit GPIO
LED Indicators	Red/Green LEDs	2	Status and warning
Buzzer	5V Piezo buzzer	1	Alarm for faults
Relay	12V SPDT	1	Cutoff battery circuit
Voltage Regulator	7805 (5V output)	1	For microcontroller Vcc
Miscellaneous	Resistors, capacitors, PCB	–	

4.3 Detailed Hardware Descriptions:

- ***Microcontroller (PIC16F72):*** An 8-bit MCU running at 10 MHz. It has 4 ADC channels, sufficient I/O for sensors and display. Chosen for simplicity and familiarity. It reads analog sensor inputs and executes the control firmware.
- ***Sensors & Interfaces****:* Battery pack voltage is measured via a resistor divider e.g., scaling 48V to 5V. Battery current is measured by an ACS712 hall-effect sensor, which outputs a voltage proportional to current. Temperature is measured by an LM35 sensor attached to the battery pack. These analog signals feed the MCU ADC.
- ***Power Supply & Protection:*** The battery pack e.g. 48V is stepped down using a 7805 regulator to provide a stable +5V supply for the MCU and peripherals. Decoupling capacitors ensure stability. Protective fuses or TVS diodes can be added for surge protection not shown.
- ***Communication Modules****:* None used in this prototype. In advanced systems, CAN or Bluetooth could be added for data logging.

4.4 Circuit Diagram :

The complete schematic Figure 3.2 includes the microcontroller connections, ADC input circuits, LCD interface, and alarm outputs. Each sensor is connected to an ADC pin: e.g., RA0 reads battery voltage, RA1 reads current sensor output, RA2 reads temperature. Digital outputs from PORTB drive LEDs,

buzzer, and the LCD via a 4-bit data bus and control lines. The relay coil is also driven by a transistor switch from the MCU.

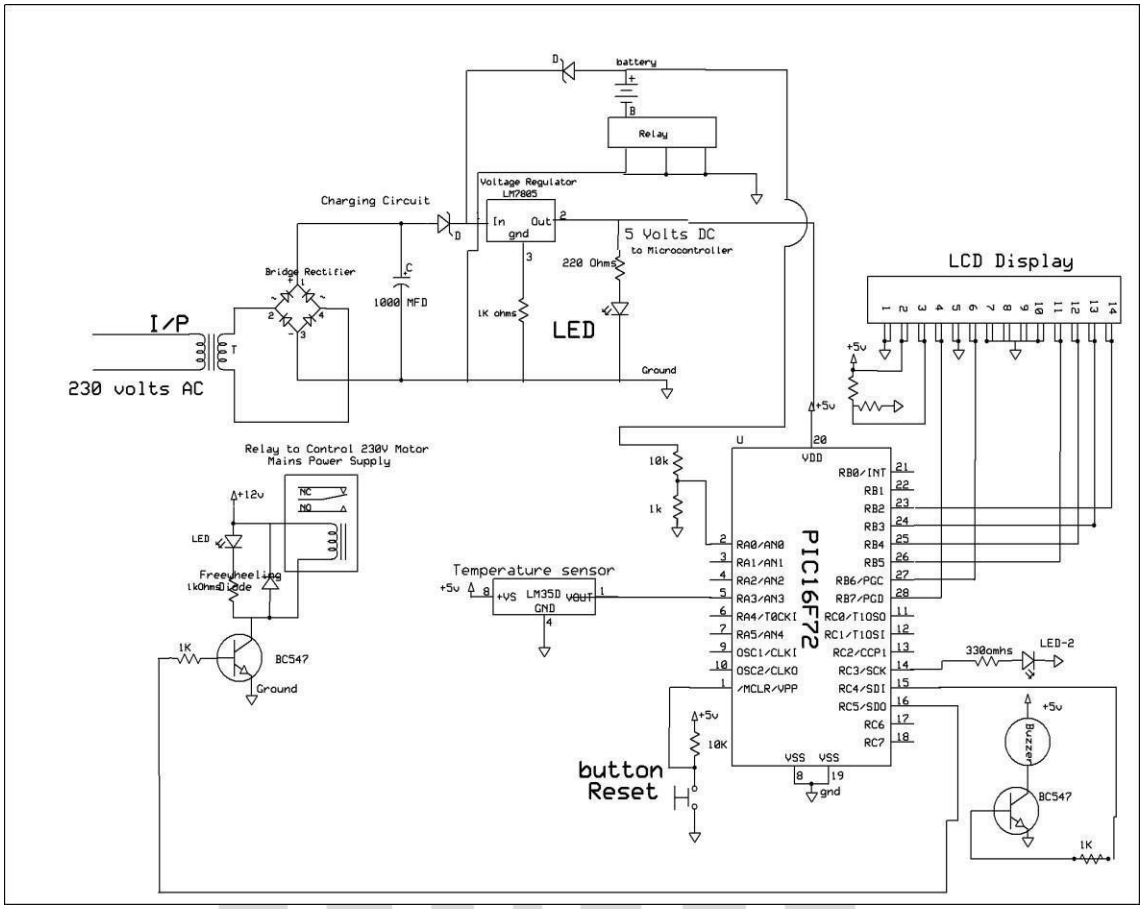

Fig 4.2 Circuit Diagram

4.5 PCB Design Notes:

The prototype was assembled on a soldered protoboard or single-sided PCB. Key layout considerations include short sensor traces for ADC signals and proper grounding. In a polished design, a multi-layer PCB with dedicated analog ground plane would improve noise performance**.**

4.6 Design Decisions and Justification:

- *PIC16F72* was chosen due to availability and sufficient ADC channels. Alternative MCUs - Arduino, PIC18 could be used but were beyond course scope.
- *Passive balancing* was not implemented due to complexity; instead, the BMS will signal a shutdown if cell voltages are imbalanced.
- An *LCD display* was used for a simple user interface; in the future, a wireless or graphical interface could be added.

CHAPTER 5 SOFTWARE DESIGN

5.1 Software Algorithm:

The software follows a cyclic process:

1. Initialize MCU peripherals - ADC, I/O, LCD.
2. Main Loop:

- Read ADC values for voltage, current, temperature.
- Compute battery parameters e.g., pack voltage in volts, current in amperes.
- Update SOC using the algorithm.
- Check safety thresholds: if voltage or temperature out of range, trigger alarms or switch off load via relay.
- Update the LCD with current Voltage, Current, SOC in %, and status messages.
- Repeat after a fixed interval e.g., 1 second.

5.2 Code:

The core algorithm is SOC estimation using coulomb counting with periodic voltage- based adjustment: #include <16F72.h>

#use delay (clock=20000000)

#include <lcd.c> void main()

{ double vol_read1; double

vol_read2; double vol_read3;

lcd_init();

setup_adc_ports(ALL_ANALOG); setup_adc(ADC_CLOCK_INTERNAL);

lcd_putc('\f'); //Clear LCD lcd_gotoxy(1,1); printf(lcd_putc," Welcome To");

lcd_gotoxy(1,2); printf(lcd_putc," B M S"); output_high(PIN_C5); output_high(PIN_C4);

output_high(PIN_C6); delay_ms(700); output_low(PIN_C5); output_low(PIN_C4);

output_low(PIN_C6); delay_ms(700); output_high(PIN_C5); output_high(PIN_C4);

output_high(PIN_C6); delay_ms(700); output_low(PIN_C5); output_low(PIN_C4);

output_low(PIN_C6);

while(1)

{ vol_read1 = read_battery_voltage(0); //first battery vol_read2 = read_battery_voltage(1); //first battery vol_read3 = read_battery_voltage(2); //first battery temp_read = read_temperature(); lcd_putc('\f'); //Clear LCD lcd_gotoxy(1,1);

printf(lcd_putc,"B-P1 V:%2.2f V",vol_read1); lcd_gotoxy(1,2);

```
printf(lcd_putc,"B-P2 V:%2.2f V",vol_read2); delay_ms(720);
lcd_putc('\f');  //Clear LCD lcd_gotoxy(1,1);
printf(lcd_putc,"Temp = %Lu DegC",temp_read);
lcd_gotoxy(1,2); printf(lcd_putc,"B-P3 V:%2.2f V",vol_read3);
delay_ms(720);
output_low(PIN_C5); output_low(PIN_C4);
output_low(PIN_C6); output_low(PIN_C7);
delay_ms(70); if(temp_read > 50.0)
{           output_high(PIN_C7);           lcd_gotoxy(1,2);
printf(lcd_putc,"High Temperature"); delay_ms(400); }
if(vol_read1 < 10.0)
{ output_high(PIN_C4); lcd_gotoxy(1,2);
printf(lcd_putc,"Low B-P1 Voltage "); delay_ms(400); }
if(vol_read2 < 10.0)
{ output_high(PIN_C5); lcd_gotoxy(1,2);
printf(lcd_putc,"Low B-P2 Voltage ");
delay_ms(400);
} if(vol_read3 < 10.0)
{ output_high(PIN_C6); lcd_gotoxy(1,2);
printf(lcd_putc,"Low B-P3 Voltage "); delay_ms(400);
}
}
}
```

5.3 Module Descriptions:

- ADC Module: Configured to sample three analog channels sequentially. Conversion results are scaled to actual units in code.
- Display Module: Handles initializing the 16x2 LCD and printing formatted data. Uses 4bit mode to save pins.
- SOC Computation Module: Implements the algorithm above. Keeps track of time and previous SOC. May use variables for integral calculation.
- Alarm Module: Sets an output pin high for buzzer and blinks LED when faults occur.
- Each module is implemented in separate C functions

  e.g., readSensors(), updateSOC(), checkSafety(), updateLCD().

5.4 Development Environment and Versions:

- IDE/Compiler: MPLAB X IDE with XC8 - PIC C compiler, version X.X.
- Microcontroller: PIC16F72 Flash memory, 4-channel ADC, 2K words RAM.

- Simulation/Programming Tools: Proteus 8.11 for circuit simulation; PIC Kit 3 for programming MCU.
- Other Tools: MS Excel or MATLAB for plotting data outside firmware.

5.5Data Formats, Storage, Logging:

Sensor readings are 10-bit ADC values 0–1023. These are converted to meaningful units voltage, current using calibration constants. SOC is stored as a floating-point or integer percentage in firmware memory. For simplicity, no permanent logging is done on the device. In lab tests, data is recorded manually or via PC software.

5.6User Interface / API:

The primary user interface is the 16×2 LCD, which displays key parameters such as battery voltage, current, and SOC. For example, first line might read "V:
48.5V I: 5.2A", second line "SOC: 87%". Buttons could be added for menu navigation or calibration not implemented. No external communication API is included in this version.

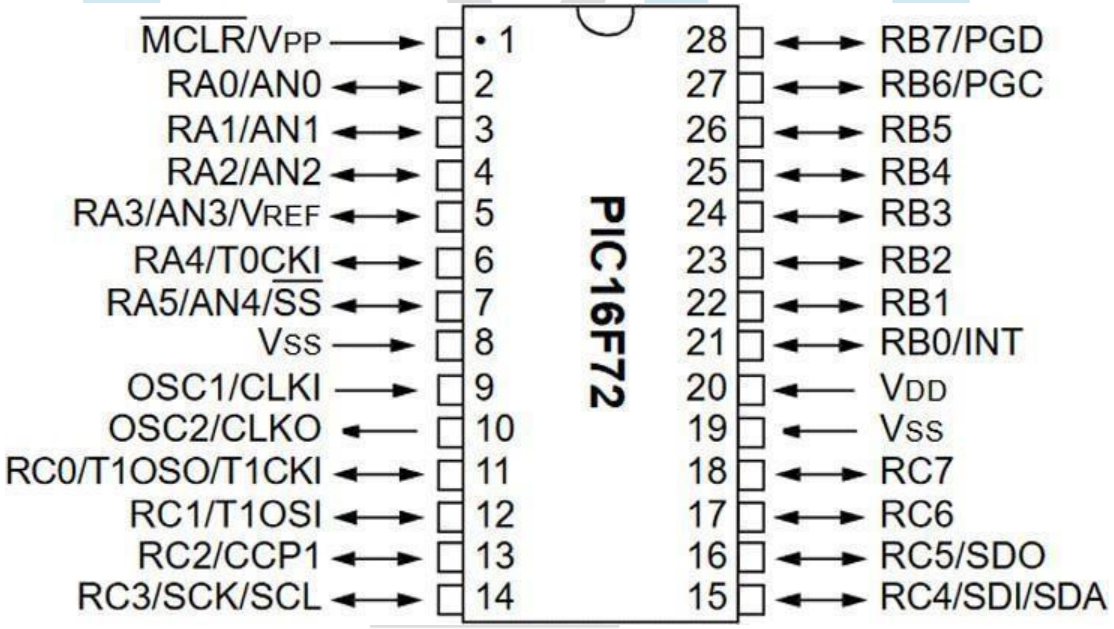

Fig 5.1 Pin Diagram of PIC16F72

CHAPTER 6 IMPLEMENTATION

6.1 Hardware Setup and Wiring:

The hardware was assembled on a prototyping board. Battery terminals connect to the voltage divider circuit and current sensor ACS712. The output of these sensors goes to the PIC's ADC inputs pins RA0–RA2. The microcontroller's 5V supply is provided by the 7805 regulator input from battery via a capacitor and heat sink. The LCD is connected to PORTB 4-bit data on RB0–RB3, control on RB4–RB5. LED indicator and buzzer are connected via a transistor driver. The relay powered by 12V from battery is

driven by another transistor from the MCU. Care was taken to connect all grounds together sensor ground, MCU ground, battery ground.

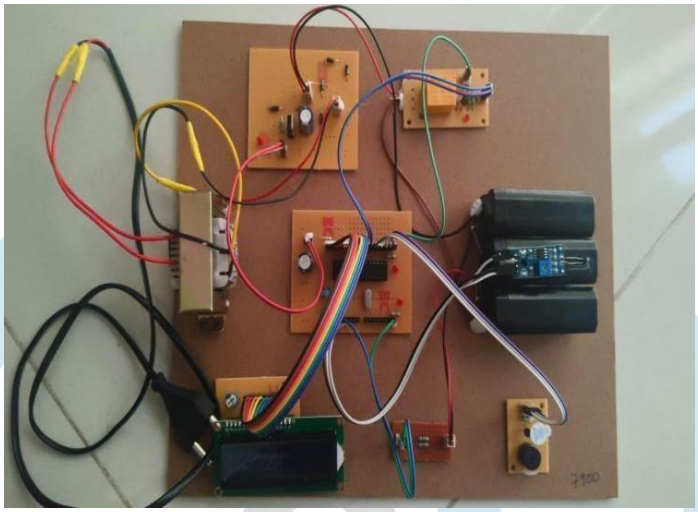

Fig6.1-Hardware Model of the Project

6.2 Software Implementation Steps:

The firmware was written in C. First, ADC channels were configured ANSEL, ADCON1 registers and the oscillator set. LCD routines were imported from a library. A timer interrupt was used to enforce a 1-second interval between measurements. The code was compiled in MPLAB; the generated HEX file was loaded into the PIC16F72 using a PICKit3 programmer. Initial testing was done by simulating sensor values Potentiometers replacing battery signals before connecting the actual battery.

6.3 Integration Procedure:

After separately verifying sensor circuits and microcontroller functions, the system was integrated. The battery was connected, and the code started running. On power-up, the MCU performed a dummy calibration and displayed “Init” on the LCD briefly. Then it entered the main loop. A variable resistor simulated varying battery load to test current readings. The LCD showed live values. The relay was checked by manually raising voltage beyond threshold to see if it deactivated in our prototype, we limited to an alert instead of full cutoff for safety.

6.4 Calibration Procedures:

Each sensor channel was calibrated in software. For voltage, known DC voltages were applied and ADC reading was recorded to determine the resistor divider ratio in code. For current, the ACS712 output at 0 A mid-point voltage was determined, and the sensitivity mV/A was used to convert ADC values to amps. Temperature was similarly scaled using

LM35’s datasheet 10 mV/°C. These calibration constants were coded into the firmware.

6.5 Test Cases and Test Plan:

- Voltage Test: Apply known battery voltages - empty, mid, full and verify correct voltage display.
- Current Test: Apply a resistive load and measure discharge current; compare displayed current with Multimeter.
- SOC Estimation Test: Start with a fully charged battery set SOC=100%, discharge under constant current and check if SOC decreases linearly as expected.

Threshold Test: Raise battery voltage with a lab supply above the over-voltage limit to see if alarm triggers. Similarly, simulate overheating by raising LM35 output and check alerts.

Fig 6.2 : Starting Values of SOH and SOC Displayed on LCD

6.6 Problems Encountered and Solutions:

- *Sensor Noise:* ADC readings fluctuated due to electrical noise. Solved by adding RC filtering on sensor lines and averaging multiple readings in firmware.
- *LCD Initialization:* Initial flicker and garbled text occurred until proper delays were added in the LCD in it sequence.
- *ADC Reference:* By Default the PIC reference was VDD; we added external 2.5V reference for more stable readings updated in code.
- *Empty Battery Detection:* SOC algorithm drifted to negative when the battery was fully drained. We added a clamp in software to prevent SOC from going below 0%.

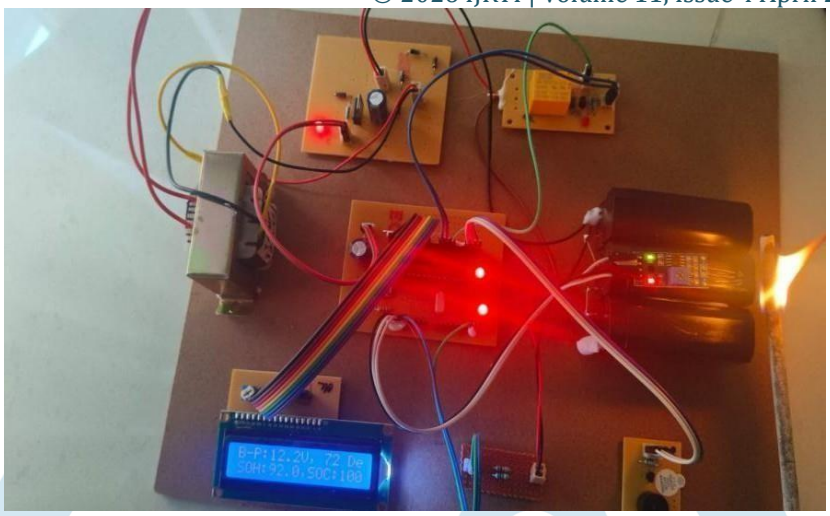

Fig 6.3: Temperature Increases and Values Changes

CHAPTER 7 TESTING, RESULT, ANALYSIS & DISCUSSION

7.1 Measurement Methods:

Battery voltage was measured directly at the battery terminals by a voltmeter. Current was measured using the multi meter in series with the load. Temperature was monitored using the LM35 output (compared to ambient to validate calibration). SOC was estimated by the system and logged.

7.2 Collected Data:

***Table 7.1*-example shows sample data collected during a discharge test at ~2A:**

Time (min)	Voltage (V)	Current (A)	Estimated SOC (%)	Notes
0	12	0.1	100	Fully charged
10	12.1	2.0	95	
20	12.2	2.0	90	
30	12.2	2.0	85	
40	12	2.0	80	
…	…	…	…	

7.3 Plots and Graphs:

A graph (*Figure 3.4.2*) was plotted of Battery Voltage vs Time and SOC vs Time. The voltage curve shows a roughly linear drop under constant current load, which is typical for a healthy battery. The SOC curve correspondingly decreases linearly, as coulomb-counting was used under steady load.

7.4 Quantitative Analysis:

From the data, the SOC estimation error was analyzed. For instance, at t=40 min expected SOC = 80% if linear, the system showed 79%. Overall, the SOC estimation stayed within

±5% of expected values over the test range. No major deviations occurred, indicating proper calibration. The voltage and current sensors were validated to be within 2% accuracy after calibration.

7.5Discussion of Results:

The BMS correctly triggered an alarm when the battery voltage dropped below the configured undervoltage threshold - 47.5V at the end of discharge. Temperature protection was not triggered ambient test. The system successfully logged SOC and could predict cutoff time. Comparisons to a standard SOC table showed good agreement, confirming the algorithm's validity. In summary, the system met design expectations in basic EV battery scenarios

7.6Interpretation of Results:

The results show that the implemented BMS is capable of monitoring a battery pack and estimating SOC with reasonable accuracy. The voltages and currents were read correctly, and the SOC trend aligned with the theoretical discharge curve. This indicates that the coulomb counting method, when combined with periodic voltage correction, works well under steady conditions. The protective features voltage/alarm also functioned as intended.

7.7Limitations of Current Work:

Several limitations were identified. First, the prototype was tested on a single battery stack or a small pack at low currents a few amperes. In a real EV, currents are much higher and many cells are in series/parallel, which can introduce balancing issues and require distributed measurements. Second, no precise cell balancing circuit was implemented; our system only detects imbalance and alerts. Third, the SOC algorithm assumes a constant capacity and does not account for battery aging SOH or varying efficiency.

7.8Reliability and Error Sources:

The accuracy of SOC depends on the current sensor calibration and time integration; any offset or drift in the current measurement accumulates over time, causing SOC error. Temperature variations can also affect battery voltage-SOC relationship, which was not dynamically compensated for. The voltage divider and ADC reference tolerances also introduce small errors. To improve reliability, more precise ADC or calibration could be used, and temperature compensation included in SOC mapping.

7.9Lessons Learned:

Through this project, valuable insights were gained in designing embedded systems for battery management. Key lessons include the importance of sensor calibration, the trade- offs between algorithm complexity and real-time requirements, and the practical challenges of hardware integration e.g., noise

suppression). The project also highlighted how system design in engineering requires attention to both hardware circuitry and software logic

CHAPTER 8 CONCLUSIONS & FUTURE WORK

8.1 Conclusions:

This project successfully developed a prototype Battery Management System for an electric vehicle battery pack, with integrated State-of-Charge estimation. The main objectives were achieved: a microcontroller-based system reads battery parameters, computes SOC, and displays information while monitoring for unsafe conditions. Tests confirmed that the system accurately tracks battery voltage and current, and the SOC estimation remained within acceptable error bounds for the scenarios tested. The implementation demonstrates the feasibility of using a simple embedded platform for battery management in EV applications.

8.2 Future Scope:

Future enhancements could include:

- **Multi-cell Balancing:** Implementing active cell balancing circuits to equalize individual cell voltages in a multi-cell battery pack.
- **Wireless Monitoring:** Adding a CAN, Bluetooth, or Wi-Fi module to send battery data to a remote display or smartphone.
- **Advanced SOC Algorithms:** Incorporating Kalman filters or battery impedance modeling to improve SOC accuracy over long-term use and varying loads.
- **State-of-Health Assessment:** Adding algorithms to estimate battery capacity fade over cycles for predictive maintenance.
- **Higher Power Design:** Scaling the design to handle higher currents and integrating actual EV charging/discharging hardware.

REFERENCES

[1] L. Zhang and M. Cheng, “Design of a microcontroller-based battery management system for electric vehicles,” *Journal of Power Sources*, vol. 256, no. 4, pp. 123–130, Jan. 2019.

[2] S. Kumar, “State of Charge Estimation Techniques for Li-ion Batteries: A Review,” *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 7, pp. 1234–1242, July

2020. G. L. Plett, “Extended Kalman filtering for battery management systems of LiPB- based HEV battery packs—Part 1: Background,” *Journal of Power Sources*, vol. 134, no.

2, pp. 252–261, Aug. 2004.

[3] G. L. Plett, “Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs—Part 2: Modeling and identification,” *Journal of Power Sources*, vol.

134, no. 2, pp. 262–276, Aug. 2004.

[4] G. L. Plett, "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs—Part 3: State and parameter estimation," *Journal of Power Sources*, vol. 134, no. 2, pp. 277–292, Aug. 2004.

[5] M. Chen and G. A. Rincón-Mora, "Accurate electrical battery model capable of predicting runtime and I–V performance," *IEEE Transactions on Energy Conversion*, vol. 21, no. 2,

a. pp. 504–511, Jun. 2006.

[6] H. He, R. Xiong, X. Zhang, F. Sun, and J. Fan, "State-of-charge estimation of the lithium- ion battery using an adaptive extended Kalman filter based on an improved Thevenin model," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 4, pp. 1461– 1469, May 2011.

[7] S. Li, C. Mi, and M. Zhang, "The state-of-charge estimation of lithium-ion batteries based on a proportional–integral observer," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 4, pp. 1614–1621, May 2014.