DIY guru™
INNOVATIONS IN FUTURE MOBILITY

Battery Monitoring and Safety Alert System (BMS-like Controller)

COURSE 5 – EMBEDDED SYSTEMS: EMBEDDED C, MICROCONTROLLER, ARDUINO & SIMULATIONS

BY NILABJA JANA, 1ST JULY,2025

02.11.2025

Index

Project Statement: Design and simulate a Battery Monitoring and Safety Alert System for a two-wheeler electric vehicle using Arduino UNO on Tinkercad Circuits.

Introduction

This project develops a simplified BMS-like controller to enhance safety in a two-wheeler electric vehicle. It continuously monitors battery voltage and temperature using analog inputs. When critical thresholds are detected (low battery or overheat), the system provides immediate visual alerts via multi-color LEDs and an audible alarm via a buzzer. System status is clearly displayed on an I2C LCD and detailed on the Serial Monitor. This simulation demonstrates essential EV safety features through cost-effective microcontroller logic.

Objective

To design and simulate a Battery Monitoring and Safety Alert System (BMS-like controller) for a two-wheeler electric vehicle using Arduino UNO on Tinkercad. This system will monitor battery voltage and temperature, trigger LED and buzzer alerts for critical conditions, and display status on an LCD and serial monitor.

📜 Component List

Sl. No.	Component ID	Quantity	Component Name / Description
1	U1	1	Arduino Uno R3
2	Rpot1	1	220 Ω Potentiometer
3	U2	1	Temperature Sensor (TMP36)
4	D1	1	Green LED
5	D2	1	Yellow LED
6	D3	1	Red LED
7	PIEZO1	1	Piezo
8	R1, R2, R3	3	100 Ω Resistor
9	R4	1	200 Ω Resistor
10	U3	1	PCF8574-based LCD 16x2 (I2C,

The **R4** pull-down resistor provides noise immunity and prevents erratic operation by keeping the associated pin at a logic LOW unless driven HIGH by another circuit element. This helps avoid spurious activation of alarms or false readings in sensor-driven logic.

⚙Circuit Design

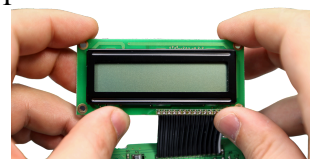
1. Display (I2C LCD - U3)

The 16x2 LCD screen shows all your information. It uses a simple four-wire I2C connection, which is very efficient.

VCC & GND: These pins connect directly to the Arduino's 5V and GND pins for power.

SDA (Data): This line connects to the Arduino's Analog Pin A4.

SCL (Clock): This line connects to the Arduino's Analog Pin A5.



2. Temperature Sensor (TMP36 - U2)

This sensor reads the ambient temperature. As an analog sensor, it provides a voltage that the Arduino reads and converts into a temperature value.

Power Pin: Connects to 5V.

Ground Pin: Connects to GND.

Vout (Signal): Connects to an analog input, Analog Pin A1.

3. Potentiometer

The 220 Ω knob allows for manual user input, like setting an alarm threshold or adjusting contrast.

Two Outer Pins: One connects to 5V, and the other connects to GND.

Middle Pin: This "wiper" pin sends the variable signal to Analog Pin A0.

4. Visual Alerts (LEDs)

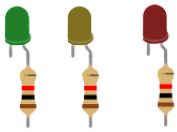
You have three colored LEDs (green, yellow, and red) to show different statuses. Each one must be connected with its own 100 Ω resistor (R1, R2, R3) to limit the current and prevent burnout.

Green LED (D1):

The long leg (+) connects to one side of its R1 resistor.

The other side of the resistor connects to Digital Pin 7.

The short leg (-) connects to GND.



Yellow LED (D2):

The long leg (+) connects to one side of its R2 resistor.

The other side of the resistor connects to Digital Pin 6.

The short leg (-) connects to GND.

Red LED (D3):

The long leg (+) connects to one side of its R3 resistor.

The other side of the resistor connects to Digital Pin 5.

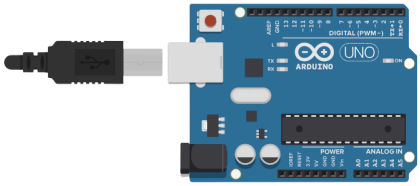
The short leg (-) connects to GND.

5. Audio Alert (Piezo Buzzer)

The piezo buzzer creates sounds, like an alarm.

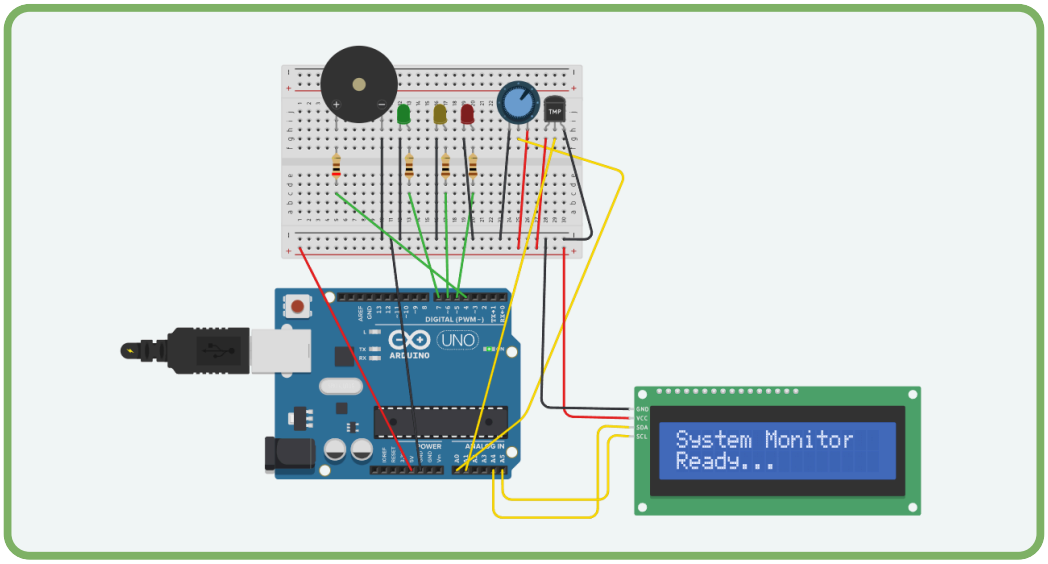
Positive Pin (+): Connects directly to Digital Pin 4.

Negative Pin (-): Connects to GND.

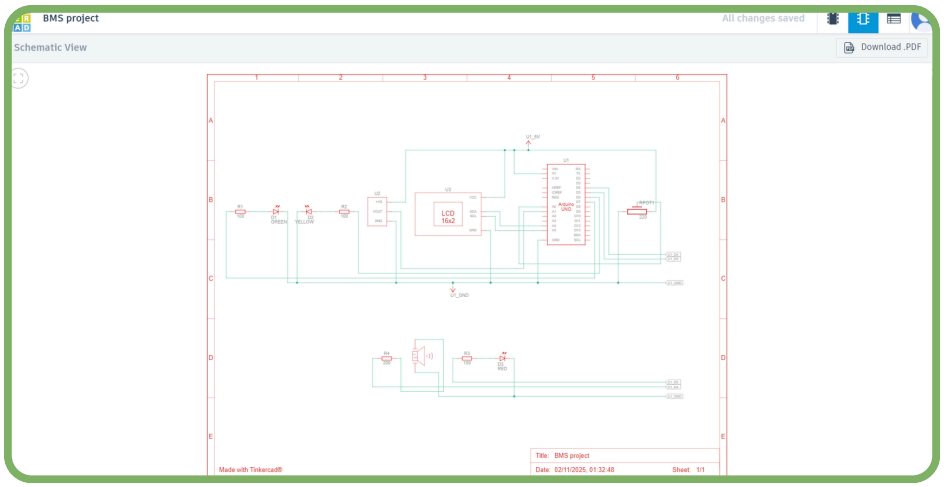


6. U1: Arduino Uno (The Controller)

This is the brain of your project. It will get power from the USB cable and distribute 5V (power) and GND (ground) to all the other components.



🔌 **Circuit Connections**



Schematic View

Public Tinkercad Simulation link:

LINK
https://www.tinkercad.com/things/6pUaQw9kpZN-bms-project/editel?sharecode=au5tPv7M0s8wzEG5rw04Uknoe_HfhYM2MRVkDLc1RuA

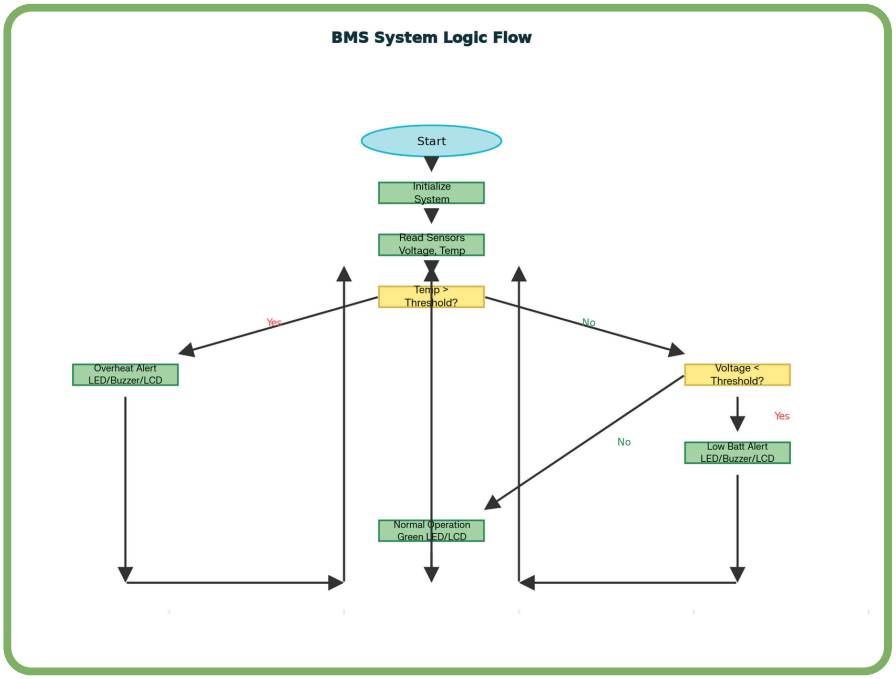
SCAN ME!

or scan this to get direct access of this simulation link

📝 Fully commented .ino Arduino project code

```
#include <Wire.h>               // Library for I2C communication
#include <LiquidCrystal_I2C.h>   // Library for I2C-based LCD

// ---------- Pin Configuration ----------
const int voltageSensorPin = A0;  // Analog pin for voltage (via potentiometer)
const int tempSensorPin = A1;     // Analog pin for temperature sensor (TMP36)
const int greenLedPin = 7;        // Green LED - Battery OK indicator
const int redLedPin = 5;          // Red LED - Low battery indicator
const int yellowLedPin = 6;       // Yellow LED - Overheat indicator
const int buzzerPin = 4;          // Buzzer pin

// ---------- LCD Setup (I2C Address 0x27, 16x2) ----------
LiquidCrystal_I2C lcd(0x27, 16, 2);

// ---------- Constants for Calculations ----------
const float VOLTAGE_SCALING_FACTOR = 4.0; // Simulate up to 20V actual battery with 0-5V ADC using potentiometer
const float ARDUINO_REF_VOLTAGE_MV = 5000.0;  // Reference voltage in millivolts
const float ADC_RESOLUTION = 1024.0;          // 10-bit ADC (0–1023 values)
const float TMP36_MV_PER_DEGREE = 10.0;       // TMP36: 10mV per °C
const float TMP36_OFFSET_MV = 500.0;          // TMP36 outputs 500mV at 0°C

// ---------- Threshold Values ----------
const float LOW_BATTERY_THRESHOLD = 9.3; // 3.1V per cel// Voltage below this = low battery
const float OVERHEAT_TEMP_THRESHOLD = 50.0; // (optional: 45–60°C range)
  // Temperature above this = overheat

// ---------- Setup Function ----------
void setup() {
 Serial.begin(9600);        // Initialize serial monitor (for debugging)
 Wire.begin();              // Start I2C communication
 lcd.init();                // Initialize LCD
 delay(100);                // Small delay for LCD stability
 lcd.backlight();           // Turn on LCD backlight

 // Display startup message
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("System Monitor");
 lcd.setCursor(0, 1);
 lcd.print("Initializing...");
 delay(2000);
 lcd.clear();

 // Configure pins as outputs
 pinMode(greenLedPin, OUTPUT);
 pinMode(redLedPin, OUTPUT);
 pinMode(yellowLedPin, OUTPUT);
 pinMode(buzzerPin, OUTPUT);

 // Turn off all outputs initially
 digitalWrite(greenLedPin, LOW);
 digitalWrite(redLedPin, LOW);
 digitalWrite(yellowLedPin, LOW);
 digitalWrite(buzzerPin, LOW);
}

// ---------- Main Loop ----------
void loop() {
 // --- Read and Convert Voltage ---
 int rawVoltage = analogRead(voltageSensorPin);   // Read raw analog value (0–1023)
 float measuredVoltage = (rawVoltage / ADC_RESOLUTION) * (ARDUINO_REF_VOLTAGE_MV / 1000.0) * VOLTAGE_SCALING_FACTOR;

 // --- Read and Convert Temperature ---
 int rawTemp = analogRead(tempSensorPin);        // Read raw value from TMP36
 float voltageAtTempPinMv = rawTemp * (ARDUINO_REF_VOLTAGE_MV / ADC_RESOLUTION);
 float temperatureC = (voltageAtTempPinMv - TMP36_OFFSET_MV) / TMP36_MV_PER_DEGREE;

 // --- Prepare Display Messages ---
 String statusMessage1 = ""; // Line 1 of LCD
 String statusMessage2 = ""; // Line 2 of LCD

 // --- Reset LEDs and Buzzer before decision ---
 digitalWrite(greenLedPin, LOW);
 digitalWrite(redLedPin, LOW);
 digitalWrite(yellowLedPin, LOW);
 noTone(buzzerPin); // Stop any buzzer tone

 // --- Decision Logic ---

 //        OVERHEAT Condition
 if (temperatureC >= OVERHEAT_TEMP_THRESHOLD) {
   statusMessage1 = "OVERHEAT!";
   statusMessage2 = "Temp: " + String(temperatureC, 1) + "C";
   digitalWrite(yellowLedPin, HIGH);
   tone(buzzerPin, 1000); // 1kHz buzzer tone
 }
 //        LOW BATTERY Condition
 else if (measuredVoltage <= LOW_BATTERY_THRESHOLD) {
   statusMessage1 = "LOW BATTERY!";
   statusMessage2 = "Volt: " + String(measuredVoltage, 1) + "V";
   digitalWrite(redLedPin, HIGH);
   tone(buzzerPin, 500); // 500Hz buzzer tone
 }
 //        NORMAL Condition
 else {
   statusMessage1 = "BATTERY OK";
   statusMessage2 = "Temp: " + String(temperatureC, 1) + "C Volt: " + String(measuredVoltage, 1) + "V";
   digitalWrite(greenLedPin, HIGH);
 }

 // --- Update LCD Display ---
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print(statusMessage1);
 lcd.setCursor(0, 1);
 lcd.print(statusMessage2);

 // --- Display Data on Serial Monitor (for observation) ---
 Serial.print("Raw Voltage: ");
 Serial.print(rawVoltage);
 Serial.print(" -> Measured Voltage: ");
 Serial.print(measuredVoltage, 2);
 Serial.println(" V");

 Serial.print("Raw Temp: ");
 Serial.print(rawTemp);
 Serial.print(" -> Temperature: ");
 Serial.print(temperatureC, 1);
 Serial.println(" C");

 Serial.println("Status: " + statusMessage1);
 Serial.println("--------------------");

 delay(1000); // Update every 1 second
}
```

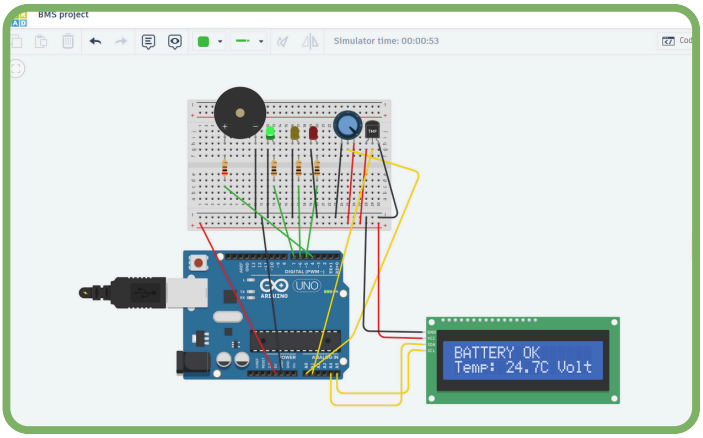
"//The voltage read on analog pin A0 (0–5V) is adjusted by a scaling factor of 4.0 to simulate a battery voltage range of 0–20V. This models the behavior of a higher-voltage EV pack using a low-voltage potentiometer in the simulation/hardware. In actual EVs, a resistor divider network would safely bring down battery voltage to safe ADC input range.//



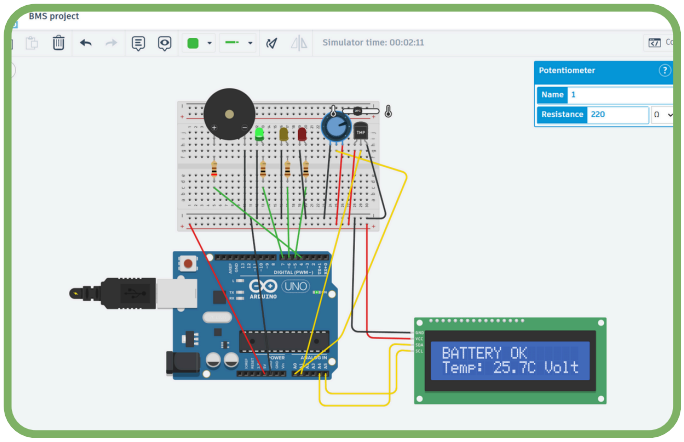
System Flowchart: Battery Monitoring and Safety Alert System

RESULTS & OUTPUTS

Project Simulation States



Screenshot (1)

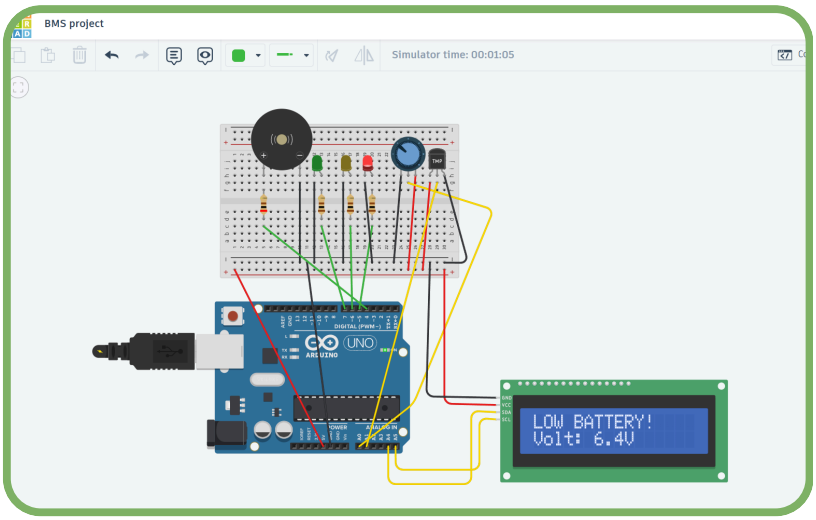


Screenshot (2)

1. Safe/Normal Status (Battery OK)

Screenshots: Screenshot (1).jpg, Screenshot (2).jpg

Analysis: These images show the system in its ideal state. The display reads "BATTERY OK," indicating that both the temperature (approx. 25°C) and voltage levels are within the safe, normal range. This condition would be visually indicated by the Green LED ( ) being on.

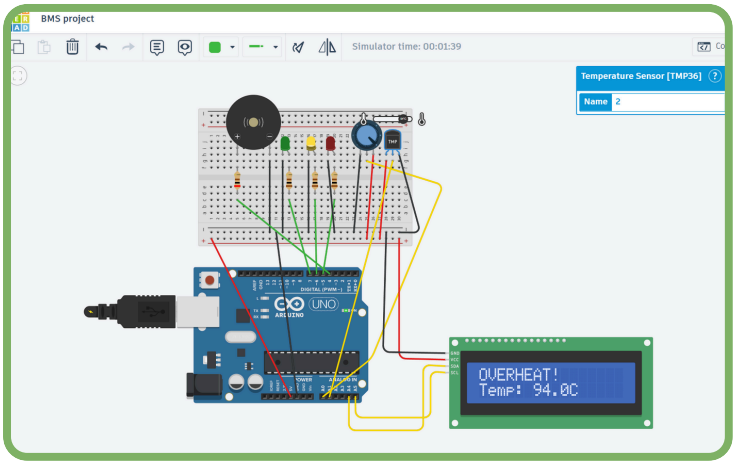


Screenshot (3)

2. Low Battery Warning

Screenshot: Screenshot (3).jpg

Analysis: This output demonstrates the low voltage warning. The system has sensed that the battery's charge is too low (down to 6.4V) and is displaying a "LOW BATTERY!" message. This alert would be paired with the Red LED ( ) to notify the user.



Screenshot (4)

3. Critical Overheat Alert

Screenshot: Screenshot (4).jpg

Analysis: This output shows a critical safety alert. The temperature has exceeded the safe limit, reaching 94.0°C. The system displays an "OVERHEAT!" warning. This emergency state is designed to trigger the Yellow LED ( ) and activate the piezo buzzer as an audible alarm.

BMS Project: Conditions & Outputs Chart

System State	Detected Condition	LCD Display Message	Green LED	Yellow LED	Red LED	Buzzer
Normal	Temp & Voltage are in the safe range	BATTERY OK Temp: 25.0C Volt: 12.5V	ON	OFF	OFF	OFF
Warning	Voltage drops below the safe limit	LOW BATTERY! Volt: 6.4V	OFF	OFF	ON	ON
Critical	Temperature rises above the safe limit	OVERHEAT! Temp: 94.0C	OFF	ON	OFF	ON

Chapter 5 : Challenges, Lessons Learned. & Conclusion

Challenges:

- Sensor Calibration: Converting the sensor's raw analog voltage into an accurate Celsius temperature.
- System Integration: Managing all components (LCD, sensors, LEDs, buzzer) at the same time without code conflicts.
- Defining Thresholds: Deciding the exact voltage and temperature limits for the "Warning" and "Critical" alerts.

Lessons Learned:

- Analog Sensing: How to read and map analog sensor inputs.
- I2C Protocol: Using the Wire library to control the LCD screen efficiently.
- State-Based Code: Writing logic for the system to switch between "Normal," "Warning," and "Critical" modes.
- Multi-Alert System: Combining visual (LEDs) and audible (buzzer) feedback to alert the user.

Conclusion:

This project is a successful prototype of a Battery Management System. It proves the concept of monitoring voltage and temperature in real-time. The system effectively triggers the correct visual and audible alarms based on pre-set safety rules, serving as a strong foundation for more advanced future versions.

Acknowledgment

I would like to sincerely thank Ashutosh Dehury and Sourabh Kumar for their invaluable guidance and support throughout this project, which was completed as part of the Internshala Electric Vehicle Course. My gratitude also extends to DIYguru Education and Research Pvt. Ltd. for providing the platform and resources for this learning experience. This report is based on my own simulations and analysis, and any external information used has been properly cited.

References

- Arduino Official Website. (2024). Arduino Language Reference. Retrieved from: https://www.arduino.cc/reference/en/
- Analog Devices. (2024). TMP36 Datasheet: Low Voltage Temperature Sensors. Retrieved from: https://www.analog.com/en/products/tmp36.html
- De Brabander, F. (2021). LiquidCrystal_I2C Library. GitHub Repository. Retrieved from: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library
- Arduino. (2024). Arduino Uno R3: Official Documentation. Retrieved from: https://store.arduino.cc/products/arduino-uno-rev3
- Tinkercad Official Website. (2024). Circuits - Electronics Simulation. Retrieved from: https://www.tinkercad.com/
- Internshala – EV Recorded Sessions, Live Classes.
- DIYguru – Official YouTube Channel. Video link: https://youtube.com/shorts/uBIAZ8DXEOw?si=feQYi62EPuW7pA3O