Prototype Description :

Battery Charge Display System for Bako Car

1. Problematic

The B1 model of the Bako vehicle uses a basic analog display that estimates the State of Charge (SoC) of the battery by reading the voltage level. While this method works for traditional batteries like lead-acid, it becomes highly inaccurate with lithium-based batteries, especially LiFePO4, because:

- Lithium batteries maintain a nearly constant voltage for most of their discharge cycle
- The voltage only drops sharply near the end, making it unreliable to estimate SoC from voltage
- The analog dial may still show "full" even when 60–70% of the capacity is already used

Objective

Develop a **functional prototype** that:

· Reads **battery charge data** via the **CAN Bus** from the Battery Management System (BMS)

· Displays the value in real time as a **precise numeric output**

· Uses an **Arduino Uno** as the central controller

· Powers all components safely using a **DC-DC step-down converter (6R1ML)**

· Visualizes the data using a **MAX7219-driven 7-segment display**

2. Components & Their Roles

	Component	Description	Function in System
	**Arduino Uno**	Microcontroller board	Processes CAN messages & controls display
	**MCP2515 CAN Module**	CAN Bus interface module	Reads data from vehicle's CAN network
	**DC-DC Converter (6R1ML)**	Step-down power supply (e.g., 12V→5V)	Regulates voltage from vehicle to safe 5V
	**MAX7219 7-Segment Display**	LED display driver	Shows battery percentage or voltage

3. Wiring

MCP2515 → Arduino Uno

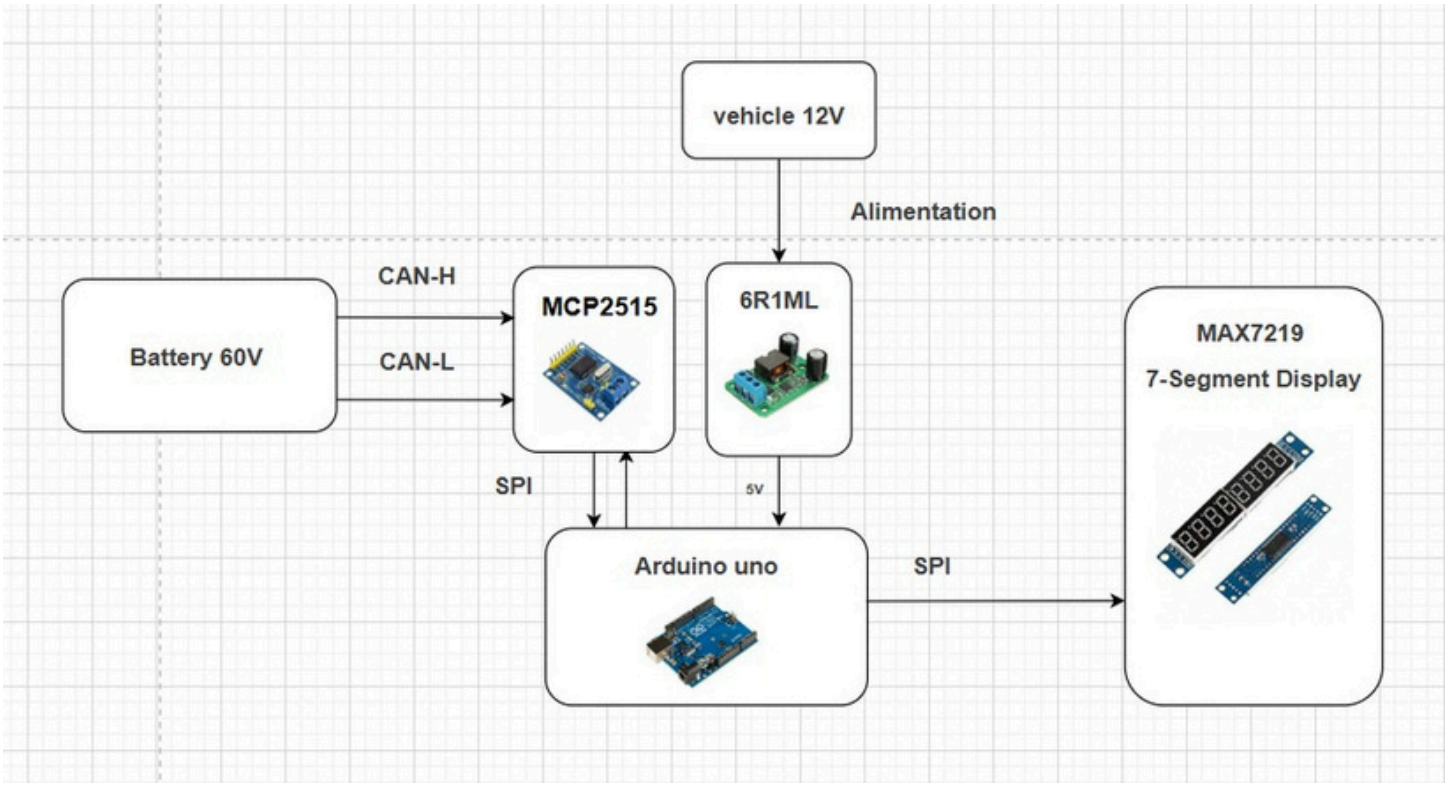
MCP2515 Pin	Arduino Pin
VCC	3.3V
GND	GND
CS	D10
SCK	D13
SO (MISO)	D12
SI (MOSI)	D11
INT	

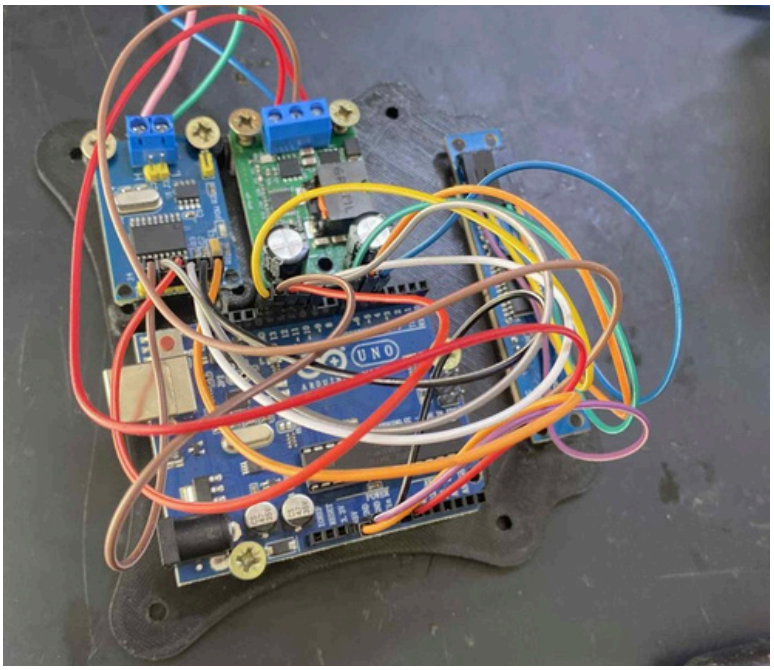
MAX7219 → Arduino Uno

MAX7219 Pin	Arduino Pin
VCC	5V
GND	GND
DIN	D6
CS/LOAD	D4
CLK	D5

DC-DC Converter (6R1ML)

DC-DC Pin	Connection
Input (Vin)	Vehicle Battery (12V)
Output (Vout)	Arduino 5V
GND	Common Ground





4. Code

```
#include <SPI.h>
#include <mcp2515.h>
#include "LedControl.h"

// ..... CAN Variables ..... //
struct can_frame canMsgTx; // Variable pour l'envoi CAN
struct can_frame canMsgRx; // Variable pour la réception CAN

MCP2515 mcp2515(10); // MCP2515 connecté sur la broche 10 (CS)
LedControl ecran = LedControl(5, 6, 7, 0); // Affichage MAX7219

const int pwmPin = 9;       //Pin PWM pour le contrôle du MOSFET
bool socRead = false;       //Drapeau pour vérifier si SoC a été lu
float lastSoC = 50;         //Dernière valeur valide du SoC

// ..... Fonction d'envoi CAN ..... //
uint8_t CAN_Tx(uint32_t id, uint8_t dlc, uint8_t *pData) {
 canMsgTx.can_id = id | CAN_EFF_FLAG;
 canMsgTx.can_dlc = dlc;
 memcpy(canMsgTx.data, pData, dlc);
 return (mcp2515.sendMessage(&canMsgTx) == MCP2515::ERROR_OK);
}
```

```
// ..... Fonction de lecture SoC ..... //
float readSoC() {
 float soc = -1;

  if(mcp2515.readMessage(&canMsgRx) == MCP2515::ERROR_OK) {
    uint32_t canId = canMsgRx.can_id & 0x1FFFFFFF;

    if(canId == 0x18FF28F4) {
    soc = canMsgRx.data[1];
    socRead = true;
    Serial.print("State of Charge (SoC): ");
    Serial.print(soc, 2);
    Serial.println(" %");
    }
  }

  if(soc >= 0) {
    lastSoC = soc;
  }

 return soc;
}

// ..... Affichage sur MAX7219 ..... //
void displaySoC(float soc) {
 ecran.clearDisplay(0);
```

```
 // Affichage statique ou symbole
ecran.setChar(0,  7,  '5',  false);
ecran.setChar(0,  6,  '0',  false);
ecran.setRow(0,  5,  B01001110);
ecran.setRow(0, 4, B00001001);

 if(soc < 0) {
   soc = lastSoC;
 }

 int socInt = (int)soc;
 int digit1 = (socInt / 100) % 10;
 int digit2 = (socInt / 10) % 10;
 int digit3 = socInt % 10;

 if (socInt >= 100) {
   ecran.setDigit(0, 4, digit1, false);
   ecran.setDigit(0, 3, digit2, false);
   ecran.setDigit(0,  2,  digit3,  true);
   ecran.setDigit(0, 1, 0, false);
 } else {
   ecran.setDigit(0, 3, digit2, false);
   ecran.setDigit(0, 2, digit3, true);
   ecran.setDigit(0, 1, digit1, false);
   ecran.setDigit(0, 0, 0, false);
 }
}
```

```
// ..... Setup ..... //
void setup() {
 Serial.begin(9600);
 pinMode(pwmPin, OUTPUT);

 ecran.shutdown(0, false);
 ecran.setIntensity(0, 3);
 ecran.clearDisplay(0);

 mcp2515.reset();
 mcp2515.setBitrate(CAN_250KBPS, MCP_8MHZ);
 mcp2515.setNormalMode();

 analogWrite(pwmPin, 255);

 uint32_t canBaseId = 0x18CD28F4;
 uint8_t canTxBytes[8] = {0x00};
 CAN_Tx(canBaseId, 8, canTxBytes);
}

// ..... Boucle principale ..... //
void loop() {
 float soc = readSoC();
 displaySoC(soc);
 delay(100);
}
```

5. System Operation

1. **MCP2515** interfaces with the **vehicle's CAN Bus** and listens for messages from the BMS.
2. **Arduino Uno** reads incoming CAN messages and extracts the relevant data (e.g., charge).
3. The extracted numeric value is sent to the **MAX7219 driver**.
4. The **7-segment display** shows the live battery status in real time (e.g., 84 %).

6. Expected Result

When powered and connected to the CAN bus:

- The system **reads real-time battery charge data**
- The 7-segment display **shows the exact v alue (e.g., 87%)**
- Drivers get **precise feedback**, not vague level bars
- The setup is **portable, reliable, and expandable**

8. Implementation Steps

1. Wire all components as per the schematic
2. Write and upload the Arduino code
3. Build a protective enclosure for the prototype (e.g., plastic or wood casing)
4. Test using CAN message simulation or connect directly to the Bako's CAN bus

Conclusion

This prototype offers a practical and affordable way to display **accurate battery charge values** in real time. It addresses a real user need for **clear, precise, and numeric battery monitoring** in electric and solar vehicles like the Bako.