ChatGPT

Comprehensive Guide to CRC in CAN Communication (JBD BMS Project)

Overview

This document explains the concept and use of CRC (Cyclic Redundancy Check) in the context of CAN (Controller Area Network) communication with a JBD BMS (Battery Management System). It is designed for easy understanding and practical implementation on ESP32 using MCP2515.

---

1. What is CRC?

CRC stands for **Cyclic Redundancy Check**. It is a type of **error-detection technique** used in digital networks and storage devices to detect accidental changes to raw data.

Real-life Analogy:

Imagine you go grocery shopping and the shopkeeper gives you a bill:

- Item 1: Rice - Rs. 50
- Item 2: Oil - Rs. 120
- Item 3: Biscuits - Rs. 30
- **Total: Rs. 200**

At home, if you sum up and it comes to Rs. 200, the bill is valid. If not, it means there's an error. CRC works the same way. It ensures that data received is the same as the data sent.

---

2. Why Use CRC in CAN Communication?

In CAN communication:

- Data travels over physical wires.
- Noise, EMI (Electromagnetic Interference), or glitches may flip bits.
- CRC acts as a **checksum** to validate the integrity of data.

In our case, the **JBD BMS** adds a 2-byte CRC-16 to every response frame to verify correctness.

---

3. When is CRC Used?

- Whenever the BMS **responds to a remote frame**, it appends a **2-byte CRC** at the end.
- The **receiver (ESP32)** must recalculate CRC on received data and compare it with received CRC to ensure validity.

4. How CRC Works (Conceptual Flow)

Step-by-Step CRC Verification:

1. BMS sends 6-byte data + 2-byte CRC.
2. ESP32 receives all 8 bytes.
3. ESP32 runs CRC algorithm on first 6 bytes.
4. It compares calculated CRC with the received CRC.
5. If matched, data is valid. Otherwise, it's discarded.

5. Endianness Handling

- JBD BMS sends **normal data** (voltage, current, etc.) in **Big Endian** (high byte first).
- **CRC is sent in Little Endian** (low byte first).

Example:

If calculated CRC = 0x3F5A

- BMS sends: [0x5A, 0x3F]
- ESP32 must reconstruct it as:

```
received_crc = (data[7] << 8) | data[6];
```

6. CRC Calculation Code (From JBD BMS Document)

C Implementation:

```
#define CRC_16_POLYNOMIALS 0xA001

uint16_t Check_CRC16(uint8_t* pchMsg, uint8_t wDataLen) {
    uint8_t i, chChar;
    uint16_t wCRC = 0xFFFF;

    while (wDataLen--) {
        chChar = *pchMsg++;
        wCRC ^= (uint16_t) chChar;

        for (i = 0; i < 8; i++) {
            if (wCRC & 0x0001)
                wCRC = (wCRC >> 1) ^ CRC_16_POLYNOMIALS;
            else
                wCRC >>= 1;
        }
    }
    return wCRC;
}
```

---

7. Example Usage in ESP32 Code:

```
uint8_t data[8]; // Received CAN data

// Calculate CRC on first 6 bytes
uint16_t crc = Check_CRC16(data, 6);

// Read received CRC (Little Endian)
uint16_t received_crc = (data[7] << 8) | data[6];

if (crc == received_crc) {
    Serial.println("CRC OK  ");
} else {
    Serial.println("CRC Error  ");
}
```

---

8. Key Points Summary:

Concept	Detail
CRC Type	CRC-16 MODBUS
Polynomial	0xA001
Init Value	0xFFFF
Byte Order	Data = Big Endian, CRC = Little
Length	6 data bytes + 2 CRC bytes

---

9. Conclusion

Understanding CRC is critical when working with CAN-based BMS systems like JBD. It ensures your microcontroller accepts only **clean and accurate data**. Always verify CRC on every received message to ensure data integrity.

If implemented correctly, CRC makes your project **reliable and robust** against communication errors.

Tejas Chaudhari