RS485 Implementation - Developer Guide

Overview

This document provides a detailed explanation of the RS485 communication implementation within the PylonTech Battery Emulator project.
It describes how the Master-Slave detection works via CAN bus and how RS485 communication is integrated to interact with the Current Monitor.

1. Master-Slave Detection via CAN Bus

The system determines which device is the Master using `bms_id_manager.h`.
The Master is responsible for sending RS485 data, while all other devices act as Slaves and only listen.
If the Master fails, another device automatically takes over the role.

2. setupRS485() - RS485 Initialization

Initializes RS485 communication by setting up the GPIO pins.
The default state is configured to receive mode to avoid collisions on the RS485 bus.

3. setRS485Mode() - Switching Send/Receive Modes

Allows the system to switch between sending and receiving modes for RS485 communication.
Ensures that only the Master sends data, while Slaves remain in receive mode.

4. sendDataToCurrentMonitor() - Sending Data via RS485

Transmits Modbus RTU commands to the Current Monitor over RS485.
This function is only executed by the Master to prevent conflicts on the RS485 bus.

5. read_response() - Receiving Data via RS485

All devices (Master & Slaves) listen for responses from the Current Monitor via RS485.
The received data is then processed accordingly.

6. checkMasterStatus() - Synchronizing RS485 with CAN Bus

Ensures that only the device recognized as Master (via CAN bus) is allowed to send RS485 data.
Uses `bms_id_manager.h` to determine the current Master status.

7. loop() - Main Execution Cycle

Runs periodically to update system status, send RS485 data (if Master), and listen for responses. Also checks for changes in Master status and updates the device role dynamically.

Full RS485 Implementation Code

```
/*
 * RS485 Communication Implementation for PylonTech Battery Emulator
 *
 * This code integrates RS485 communication into the PylonTech Battery Emulator.
 * The RS485 connection is used to communicate with the Current Monitor.
 * Only the active master (determined via CAN bus) is allowed to send RS485 data, while all other devices remain in listening mode.
 *
 * Features:
 * - Master-Slave detection via CAN bus
 * - RS485 communication using UART2 (GPIO 16/17)
 * - Ensures only the master sends RS485 data, preventing bus conflicts
 * - Automatic switching between sending and receiving modes
 * - Reads responses from the Current Monitor
 */

#include <HardwareSerial.h>

// Define RS485 control pins
#define RS485_DE_PIN 16  // RS485 Data Enable (for sending)
#define RS485_RE_PIN 17  // RS485 Receive Enable (for receiving)

HardwareSerial rs485Serial(2); // Use UART2 for RS485 communication
bool isMaster = false;  // Variable to determine master status

/**
 * Initializes RS485 communication.
 * Sets up the necessary GPIO pins and configures the default state to receive mode.
 */
void setupRS485() {
    pinMode(RS485_DE_PIN, OUTPUT);
    pinMode(RS485_RE_PIN, OUTPUT);
    setRS485Mode(false); // Default to receive mode
}

/**
 * Switches between sending and receiving on the RS485 bus.
 * Ensures that only the master sends data while others remain in listening mode.
 * @param master: true = Sending mode, false = Receiving mode
 */
void setRS485Mode(bool master) {
    if (master) {
        digitalWrite(RS485_DE_PIN, HIGH); // Enable sending mode
        digitalWrite(RS485_RE_PIN, HIGH);
    } else {
        digitalWrite(RS485_DE_PIN, LOW); // Enable receiving mode
        digitalWrite(RS485_RE_PIN, LOW);
    }
}

/**
 * Checks the CAN bus to determine if this device is the master.
 * Reads incoming CAN messages and updates the isMaster variable accordingly.
 */
void checkCANBusForMaster() {
    if (CAN.parsePacket()) {
        int masterStatus = CAN.read();
        isMaster = (masterStatus == 1);
    }
}

/**
 * Sends a request to the Current Monitor via RS485.
 * Only the active master is allowed to send data.
 * @param data: Pointer to the data to be sent
 * @param length: Length of the data to be sent
 */
void sendDataToCurrentMonitor(const uint8_t* data, size_t length) {
    if (!isMaster) return; // Only the master can send!

    setRS485Mode(true);  // Set RS485 to sending mode
    delay(5);            // Short delay for stability

    rs485Serial.write(data, length);
    rs485Serial.flush(); // Ensure all data is sent

    delay(5);            // Wait time after sending
    setRS485Mode(false); // Switch back to receive mode
}

/**
 * Receives data from the Current Monitor via RS485.
 * All devices (Master & Slaves) listen.
 * Processes incoming data for further use.
 */
void read_response() {
    setRS485Mode(false); // Ensure the device is in receive mode

    if (rs485Serial.available()) {
        uint8_t responseData[32];  // Buffer for received data
        rs485Serial.readBytes(responseData, sizeof(responseData));

        // Processing of received data happens here
    }
}

/**
 * Setup function that initializes serial communication, RS485, and CAN bus.
 * Configures UART2 for RS485 and prepares the device for communication.
 */
void setup() {
    Serial.begin(115200);
    rs485Serial.begin(9600, SERIAL_8N1, 16, 17); // RS485 UART2 on GPIO 16/17
    setupRS485();

    CAN.begin(500E3);  // Initialize CAN bus at 500 kbps
}

/**
 * Main loop function that regularly checks the master status and manages RS485 communication.
 * - Checks if this device is the master using CAN bus communication.
 * - If master, sends a Modbus RTU request to the Current Monitor via RS485.
 * - Reads responses from the Current Monitor.
 */
void loop() {
    checkCANBusForMaster();  // Regularly check master status

    if (isMaster) {
        uint8_t requestData[] = {0x01, 0x03, 0x00, 0x00, 0x00, 0x06}; // Example Modbus request
        sendDataToCurrentMonitor(requestData, sizeof(requestData));
    }

    read_response(); // All BMS units listen to the response
    delay(1000);
}
```

Conclusion

This document explains the implementation of RS485 communication in combination with CAN bus Master-Slave detection.

The code ensures that only the Master transmits RS485 data while maintaining synchronized communication with the Current Monitor.

This approach prevents conflicts and ensures automatic Master handover if a failure occurs.