BEÁGYAZOTT RENDSZEREK (C)

6. forduló

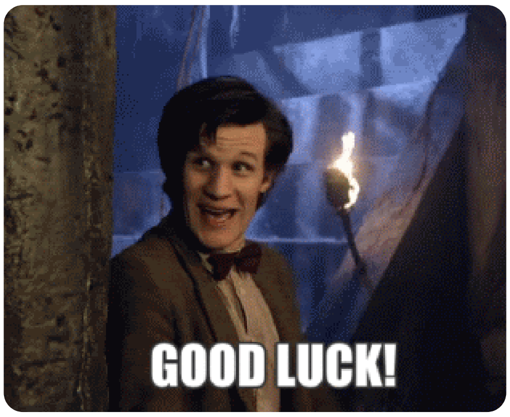
A kategória támogatója: Robert Bosch Kft.

Ismertető a feladatlaphoz

Kezdj neki minél hamarabb, mert a feladatot a forduló záró időpontjáig lehet beküldeni, nem addig lehet elkezdeni!

A feladatot ajánljuk NEM mobilon/tableten megoldani!

Sok sikert!



Mekk Mester szabadidejében is szeret programozni és kitalálta, hogy hobbiprojektnek csinál egy szélmérőt, amit BLE (Bluetooth Low Energy) kapcsolaton keresztül távolról is tud monitorozni. Ehhez a QS-FS01 szenzort és az ESP32 mikrovezérlőt választotta. A projekthez a platformio és az arduino framework-öt választotta.

Szélmérő adatlap: https://www.laskakit.cz/user/related_files/qs-fs-en.pdf

Modbus specifikáció: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

Arduino BLE library: https://www.arduino.cc/reference/en/libraries/arduinoble/

1. feladat 3 pont

Mekk Mester UART-RS485 konvertere még nem érkezett meg, de már nagyon szeretné kipróbálni a szenzort, ezért úgy döntött, hogy megpróbálja kiolvasni olyan módon, hogy az ESP32 UART RX-TX portját rövidre zárva emulálja az RS485 kommunikációt.

Hova kell kötnie a szenzor RS485 kivezetéseit, hogy ez így működjön?

Válasz

- ○ B kivezetést az RX-TX-re, A kivezetést GND-re
- ○ A kivezetést az RX-TX-re, B kivezetést GND-re

2. feladat 8 pont

Mekk Mester az UART2 perifériára kötötte a szenzort az előző feladatban részletezett módon.

Válaszd ki a kódot, amely megvalósítja helyesen a működést. (A CRC számításhoz a függvény az adatlapban található kóddal egy az egyben megegyezik, itt nem kerül külön bemutatásra).

Válasz

○

```
#include <Arduino.h>
#include "HardwareSerial.h"

typedef enum { SENDING, RECEIVING } RS485_STATE;

typedef enum { WAIT_FOR_STATION_NUMBER, WAIT_FOR_COMMAND, WAIT_FOR_DATA_LENGTH

#define rs485Serial Serial2
#define loggerSerial Serial

const uint8_t STATION_ID = 0x02;
const uint8_t READ_WIND_COMMAND = 0x03;
const uint8_t READ_COMMAND[8] = {STATION_ID, READ_WIND_COMMAND, 0x00, 0x00, 0x

uint8_t readBuffer[7];
uint8_t* readDataPtr = 0;
uint8_t receivedBytes = 0;
uint8_t responsePayloadLength = 0;
uint8_t crcLow = 0;
uint8_t crcHigh = 0;
RS485_STATE rs485tate;
RS485_RECEIVING_STATE rs485ReceivingState;

int16_t calculateCRC(uint8_t* data, size_t size);
```

```
void resetRS485Receiver();

void setup() {
    loggerSerial.begin(115200);
    rs485Serial.begin(9600);

    resetRS485Receiver();
}

void loop() {
    rs485Serial.write(READ_COMMAND, sizeof(READ_COMMAND));
    delay(100);

    while (rs485Serial.available() > 0) {
        uint8_t readedByte = (uint8_t)rs485Serial.read();

        if (rs485tate == RECEIVING) {
            switch (rs485ReceivingState) {
                case WAIT_FOR_STATION_NUMBER:
                    if (readedByte == STATION_ID) {
                        readDataPtr = readBuffer;
                        *(readDataPtr++) = readedByte;
                        rs485ReceivingState = WAIT_FOR_COMMAND;
                    } else {
                        resetRS485Receiver();
                    }
                    break;
                case WAIT_FOR_COMMAND:
                    if (readedByte == READ_WIND_COMMAND) {
                        *(readDataPtr++) = readedByte;
                        rs485ReceivingState = WAIT_FOR_DATA_LENGTH;
                    } else {
                        resetRS485Receiver();
                    }
                    break;
                case WAIT_FOR_DATA_LENGTH:
                    responsePayloadLength = readedByte;
                    *(readDataPtr++) = readedByte;
                    if (responsePayloadLength == 2) {
                        receivedBytes = 0;
                        rs485ReceivingState = WAIT_FOR_DATA;
                    } else {
                        resetRS485Receiver();
                    }
                    break;
                case WAIT_FOR_DATA:
                    *(readDataPtr++) = readedByte;
                    receivedBytes++;
                    if (receivedBytes == responsePayloadLength) {
                        rs485ReceivingState = WAIT_FOR_CRC_LOW;
                    }
                    break;
                case WAIT_FOR_CRC_LOW:
                    crcLow = readedByte;
                    rs485ReceivingState = WAIT_FOR_CRC_HIGH;
                    break;
                case WAIT_FOR_CRC_HIGH:
                    crcHigh = readedByte;
                    int16_t receivedCRC = (((int16_t)crcHigh << 8) | crcLow);
                    int16_t calcCRC = calculateCRC(readBuffer, receivedBytes +
                    if (receivedCRC == calcCRC) {
                        uint16_t rawWindSpeed = (uint16_t)((readBuffer[3] << 8
                        loggerSerial.printf("Anemometer measured: %f \r\n", (r
                    }
                    resetRS485Receiver();
                    break;
            }
        } else {
            receivedBytes++;
            if (receivedBytes == sizeof(READ_COMMAND)) {
                rs485tate = RECEIVING;
            }
        }
    }

    delay(100);
}

void resetRS485Receiver() {
    while (rs485Serial.available() > 0) {
        rs485Serial.read();
    }
    receivedBytes = 0;
    rs485tate = SENDING;
    rs485ReceivingState = WAIT_FOR_STATION_NUMBER;
}
```

```
#include <Arduino.h>
#include "HardwareSerial.h"

typedef enum { SENDING, RECEIVING } RS485_STATE;

typedef enum { WAIT_FOR_STATION_NUMBER, WAIT_FOR_COMMAND, WAIT_FOR_DATA_LENGTH
```

```
#define rs485Serial Serial2
#define loggerSerial Serial

const uint8_t STATION_ID = 0x02;
const uint8_t READ_WIND_COMMAND = 0x03;
const uint8_t READ_COMMAND[8] = {STATION_ID, READ_WIND_COMMAND, 0x00, 0x00, 0x

uint8_t readBuffer[7];
uint8_t* readDataPtr = 0;
uint8_t receivedBytes = 0;
uint8_t responsePayloadLength = 0;
uint8_t crcLow = 0;
uint8_t crcHigh = 0;
RS485_STATE rs485tate;
RS485_RECEIVING_STATE rs485ReceivingState;

int16_t calculateCRC(uint8_t* data, size_t size);
void resetRS485Receiver();

void setup() {
    loggerSerial.begin(115200);
    rs485Serial.begin(9600);

    resetRS485Receiver();
}

void loop() {
    rs485Serial.write(READ_COMMAND, sizeof(READ_COMMAND));
    delay(100);

    while (rs485Serial.available() > 0) {
        uint8_t readedByte = (uint8_t)rs485Serial.read();

        if (rs485tate == RECEIVING) {
            switch (rs485ReceivingState) {
                case WAIT_FOR_STATION_NUMBER:
                    if (readedByte == STATION_ID) {
                        readDataPtr = readBuffer;
                        *(readDataPtr++) = readedByte;
                        rs485ReceivingState = WAIT_FOR_COMMAND;
                    } else {
                        resetRS485Receiver();
                    }
                    break;
                case WAIT_FOR_COMMAND:
                    if (readedByte == READ_WIND_COMMAND) {
                        *(readDataPtr++) = readedByte;
                        rs485ReceivingState = WAIT_FOR_DATA_LENGTH;
                    } else {
                        resetRS485Receiver();
                    }
                    break;
                case WAIT_FOR_DATA_LENGTH:
                    responsePayloadLength = readedByte;
                    *(readDataPtr++) = readedByte;
                    if (responsePayloadLength == 2) {
                        receivedBytes = 0;
                        rs485ReceivingState = WAIT_FOR_DATA;
                    } else {
                        resetRS485Receiver();
                    }
                    break;
                case WAIT_FOR_DATA:
                    *(readDataPtr++) = readedByte;
                    receivedBytes++;
                    if (receivedBytes == responsePayloadLength) {
                        rs485ReceivingState = WAIT_FOR_CRC_HIGH;
                    }
                    break;
                case WAIT_FOR_CRC_HIGH:
                    crcHigh = readedByte;
                    rs485ReceivingState = WAIT_FOR_CRC_LOW;
                    break;
                case WAIT_FOR_CRC_LOW:
                    crcLow = readedByte;
                    int16_t receivedCRC = (((int16_t)crcHigh << 8) | crcLow);
                    int16_t calcCRC = calculateCRC(readBuffer, receivedBytes +
                    if (receivedCRC == calcCRC) {
                        uint16_t rawWindSpeed = (uint16_t)((readBuffer[3] << 8
                        loggerSerial.printf("Anemometer measured: %f \r\n", (r
                    }
                    resetRS485Receiver();
                    break;
            }
        } else {
            receivedBytes++;
            if (receivedBytes == sizeof(READ_COMMAND)) {
                rs485tate = RECEIVING;
            }
        }
    }

    delay(100);
}

void resetRS485Receiver() {
    while (rs485Serial.available() > 0) {
        rs485Serial.read();
    }
    receivedBytes = 0;
    rs485tate = SENDING;
    rs485ReceivingState = WAIT_FOR_STATION_NUMBER;
}
```

3. feladat 12 pont

Mekk Mester szeretne egy saját BLE szervizt készíteni a szélmérőhöz. Azt szeretné, hogy az utolsó mért adatot le tudja olvasni és a friss mérési adatokat az eszköz automatikusan tudja küldeni.

Válaszd ki a helyes megoldásokat!

Válaszok

☐

```
#include <BLE2902.h>
#include <BLEServer.h>

class AnemometerService {
  private:
    inline static constexpr char SERVICE_UUID[] = "2CA703E0-375D-11E9-B56E-080
    inline static constexpr char CHARACTERISTIC_UUID[] = "2CA703E1-375D-11E9-B

  public:
    AnemometerService(BLEServer* pServer)
        : mpServer(pServer)
        , mpService(nullptr)
        , mCharacteristic(CHARACTERISTIC_UUID, BLECharacteristic::PROPERTY_REA

    void begin() {
        mpService = mpServer->createService(SERVICE_UUID);
        mpService->addCharacteristic(&mCharacteristic);
        mCharacteristic.addDescriptor(new BLE2902());
        mCharacteristic.setValue(mWindSpeed);
        mpService->start();
    }

    void updateSpeed(float windSpeed) {
        mWindSpeed = windSpeed;
        mCharacteristic.setValue(mWindSpeed);
        mCharacteristic.indicate();
    }

  private:
    BLEServer* mpServer;
    BLEService* mpService;
    BLECharacteristic mCharacteristic;
    float mWindSpeed = 0.0f;
};
```

```
#include <BLE2902.h>
#include <BLEServer.h>

class AnemometerService {
  private:
    inline static constexpr char SERVICE_UUID[] = "2CA703E0-375D-11E9-B56E-080
    inline static constexpr char CHARACTERISTIC_UUID[] = "2CA703E1-375D-11E9-E

  public:
    AnemometerService(BLEServer* pServer)
        : mpServer(pServer)
        , mpService(nullptr)
        , mCharacteristic(CHARACTERISTIC_UUID, BLECharacteristic::PROPERTY_REA

    void begin() {
        mpService = mpServer->createService(SERVICE_UUID);
        mpService->addCharacteristic(&mCharacteristic);
        mCharacteristic.addDescriptor(new BLE2902());
        mCharacteristic.setValue(mWindSpeed);
        mpService->start();
    }

    void updateSpeed(float windSpeed) {
        mWindSpeed = windSpeed;
        mCharacteristic.setValue(mWindSpeed);
        mCharacteristic.notify();
    }

  private:
    BLEServer* mpServer;
    BLEService* mpService;
    BLECharacteristic mCharacteristic;
    float mWindSpeed = 0.0f;
};
```

```
#include <BLE2902.h>
#include <BLEServer.h>
class AnemometerService {
  private:
    inline static constexpr char SERVICE_UUID[] = "2CA703E0-375D-11E9-B56E-080
    inline static constexpr char CHARACTERISTIC_UUID[] = "2CA703E1-375D-11E9-E

  public:
    AnemometerService(BLEServer* pServer)
        : mpServer(pServer)
        , mpService(nullptr)
        , mCharacteristic(CHARACTERISTIC_UUID, BLECharacteristic::PROPERTY_REA

    void begin() {
        mpService = mpServer->createService(SERVICE_UUID);
        mpService->addCharacteristic(&mCharacteristic);
        mCharacteristic.addDescriptor(new BLE2902());
        mCharacteristic.setValue(mWindSpeed);
        mpService->start();
    }

    void updateSpeed(float windSpeed) {
        mWindSpeed = windSpeed;
        mCharacteristic.setValue(mWindSpeed);
        mCharacteristic.notify();
    }

  private:
    BLEServer* mpServer;
    BLEService* mpService;
    BLECharacteristic mCharacteristic;
    float mWindSpeed = 0.0f;
};
```

```
#include <BLE2902.h>
#include <BLEServer.h>

class AnemometerService {
  private:
    inline static constexpr char SERVICE_UUID[] = "2CA703E0-375D-11E9-B56E-080
    inline static constexpr char CHARACTERISTIC_UUID[] = "2CA703E1-375D-11E9-E

  public:
    AnemometerService(BLEServer* pServer)
        : mpServer(pServer)
        , mpService(nullptr)
        , mCharacteristic(CHARACTERISTIC_UUID, BLECharacteristic::PROPERTY_NOT

    void begin() {
        mpService = mpServer->createService(SERVICE_UUID);
        mpService->addCharacteristic(&mCharacteristic);
        mCharacteristic.addDescriptor(new BLE2902());
        mCharacteristic.setValue(mWindSpeed);
        mpService->start();
    }

    void updateSpeed(float windSpeed) {
        mWindSpeed = windSpeed;
        mCharacteristic.setValue(mWindSpeed);
        mCharacteristic.notify();
    }

  private:
    BLEServer* mpServer;
    BLEService* mpService;
    BLECharacteristic mCharacteristic;
    float mWindSpeed = 0.0f;
};
```

Megoldások beküldése