



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Escola Tècnica
Superior d'Enginyeria
Informàtica



etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

16 de mayo de 2025

TRABAJO ACADÉMICO

RS485

REDES INDUSTRIALES | GRUPO A1

Grado en Informática Industrial y Robótica

Autor/a:

Lourdes Francés Llimerá | lfralli@epsa.upv.es

Ángela Espert Cornejo | aespcor@etsinf.upv.es

Tutor/a:

Lenin Guillermo Lemus Zúñiga | lemus@upvnet.upv.es

TABLA DE CONTENIDOS

LISTADO DE TABLAS	3
LISTADO DE FIGURAS	5
LISTADO DE CÓDIGO	7
1. INTRODUCCIÓN	9
1.1. INTRODUCCIÓN.....	9
1.2. OBJETIVO	9
2. MONTAJE PROTOCOLO RS485	11
2.1. DEFINICIÓN	11
2.2. CÓDIGO.....	11
2.3. MONTAJE	15
3. BIBLIOGRAFÍA.....	17
3.1. INFORMACIÓN SOBRE PROTOCOLOS Y SUS MONTAJES	17
3.2. ELEMENTOS EMPLEADOS PARA LOS MONTAJES	17





LISTADO DE TABLAS

No se encuentran elementos de tabla de ilustraciones.





LISTADO DE FIGURAS

Figura 1: Funcionamiento código RS485.....	15
Figura 2: Conexiones RS485.	16





LISTADO DE CÓDIGO

Código 1: ESP32-S3 Emisora.....	13
Código 2: ESP32-S3 Receptora.....	14





1. INTRODUCCIÓN

1.1. INTRODUCCIÓN

En las prácticas de Redes Industriales, se ha trabajado con la ESP32-S3. Durante el desarrollo de las clases, se ha solicitado que los alumnos realicen tres montajes, uno por cada tipo de comunicación dado en teoría, siendo estos RS485, UART y CAN.

1.2. OBJETIVO

El objetivo de este documento es recopilar y organizar de forma estructurada los códigos, montajes y conocimientos adquiridos a lo largo del desarrollo del montaje relacionado con el protocolo RS485, presentando un resumen claro y coherente de los contenidos.



2. MONTAJE PROTOCOLO RS485

2.1. DEFINICIÓN

RS485 es un estándar de comunicación serial que permite la conexión de múltiples dispositivos en una red, utilizando una topología bus. Este protocolo es especialmente útil en aplicaciones industriales y de automatización de edificios, donde se requiere la interconexión de varios dispositivos en una misma red. RS485 es conocido por su capacidad para soportar largas distancias de comunicación y su resistencia a interferencias electromagnéticas.

Para que dicho protocolo funcione, un dispositivo maestro inicia la comunicación y los dispositivos esclavos responden. La señalización diferencial utilizada en RS485 permite una comunicación robusta y resistente a interferencias. Cada dispositivo en la red debe ser configurado para transmitir y recibir datos en el momento adecuado, lo que se controla mediante la señal DE.

CARACTERÍSTICAS

A continuación, se enumeran una serie de características que conforman el protocolo.

- **Comunicación multidispositivo:** RS485 permite la conexión de hasta 32 dispositivos en un bus.
- **Larga distancia:** Este protocolo soporta distancias de comunicación de hasta 1.200 metros, lo que lo hace ideal para aplicaciones en grandes instalaciones.
- **Resistencia a interferencias:** Utiliza señalización diferencial, que proporciona una mayor inmunidad a las interferencias electromagnéticas, lo que es crucial en entornos industriales con altos niveles de ruido electromagnético.
- **Configuración compleja:** Requiere una configuración más compleja que UART, especialmente en términos de control de la señal DE.
- **Problemas de terminación de línea:** En redes extensas, puede ser necesario implementar terminaciones de línea para evitar reflejos de señal y problemas de comunicación.

2.2. CÓDIGO

Para implementar RS485 en Arduino, se utiliza un transceptor RS485. Los pines TX y RX se conectan a los pines correspondientes del transceptor; y, dependiendo del transceptor empleado, se utiliza un pin adicional para controlar la señal DE (*Data Enable*). Esta señal DE es crucial para controlar si el dispositivo está transmitiendo o recibiendo datos.

Se adjunta a continuación el código de ambas ESP32-S3

ESP32-S3 EMISORA

```
#include <SoftwareSerial.h>
```

```

// Define RS485 pins
#define RS485_TX 17
#define RS485_RX 16

#define HEAD 0xAA
#define TAIL 0xFE

#define SLAVE_TARGET 0x23 // Esclavo al que se envia el comando

// Initialize SoftwareSerial for RS485
SoftwareSerial rs485Serial(RS485_RX, RS485_TX);

void setFrame()
{
    // Define the frame structure
    byte frame[8];
    frame[0] = HEAD;
    frame[1] = SLAVE_TARGET;
    frame[2] = 0x48; // 'H'
    frame[3] = 0x6f; // 'o'
    frame[4] = 0x6c; // 'l'
    frame[5] = 0x61; // 'a'
    frame[6] = TAIL;
    frame[7] = 0xAA; // Padding

    rs485Serial.write(frame, 8);
    rs485Serial.flush(); // Ensure all data is sent
    delay(1000); // Small delay to ensure data is sent

    Serial.print("Mensaje enviado\n");
}

void setup()
{
    Serial.begin(115200); // Debugging serial
    rs485Serial.begin(9600); // RS485 serial
}

void loop()
{
    static uint32_t lastStamp = 0;
    uint32_t currentStamp = millis();

    if (currentStamp - lastStamp > 1000)
    {
        lastStamp = currentStamp;
    }
}

```

```

        sendFrame(); // Send the frame every second
    }
}

```

Código 1: ESP32-S3 Emisora.

ESP32-S3 RECEPTORA

```

#include <SoftwareSerial.h>

// Define RS485 pins
#define RS485_TX 17 // TX pin for RS485
#define RS485_RX 16 // RX pin for RS485

// Initialize SoftwareSerial for RS485
SoftwareSerial rs485Serial(RS485_RX, RS485_TX);

void setup()
{
    Serial.begin(115200); // Debugging serial
    rs485Serial.begin(9600); // RS485 serial
}

void loop()
{
    if (rs485Serial.available() > 0)
    {
        // Read incoming data
        byte incomingByte;
        byte frame[8];
        int idx = 0;

        while (rs485Serial.available() > 0 && idx < 8)
        {
            incomingByte = rs485Serial.read();
            frame[idx++] = incomingByte;
            delay(3); // Small delay to ensure all data is read
        }

        // Check if the received frame is valid
        if (frame[0] == 0xAA && frame[6] == 0xFE)
        {
            Serial.println("Received valid frame:");

            // Print the received frame in hexadecimal
            for (int i = 0; i < 8; i++)
            {
                Serial.print(frame[i], HEX);
            }
        }
    }
}

```

```

        Serial.print(" ");
    }
    Serial.println();

    // If the message is the expected one, reconstruct the text
    if (frame[1] == 0x23)
    {
        Serial.print("Mensaje recibido: ");

        char receivedMessage[5] = {0}; // Buffer for text
        for (int i = 0; i < 4; i++)
        {
            receivedMessage[i] = static_cast<char>(frame[i + 2]);
        }

        Serial.println(receivedMessage); // Display msg as text string
    }
}
}
}

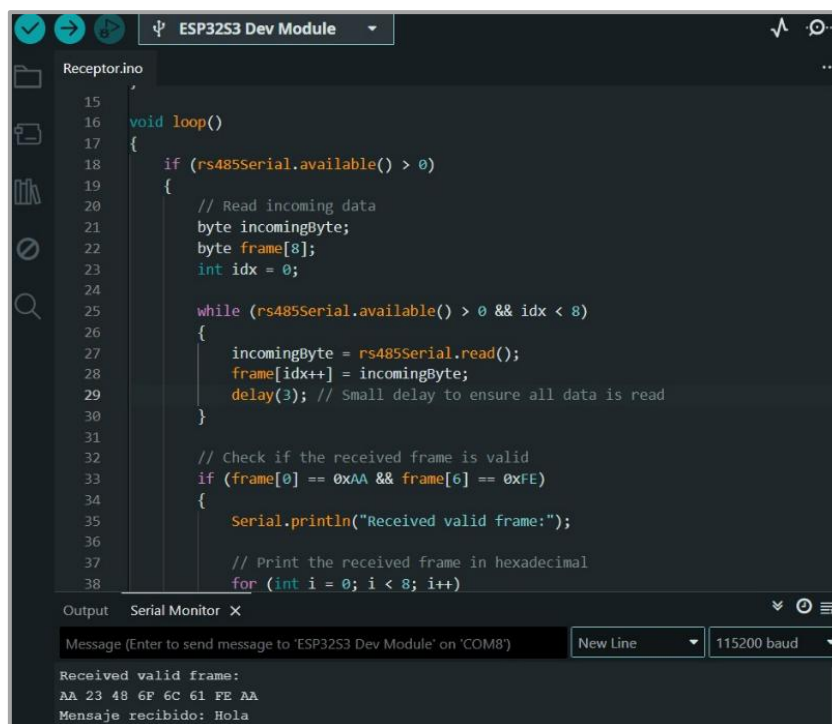
```

Código 2: ESP32-S3 Receptora.

RESULTADOS OBTENIDOS

Los códigos se han probado siguiendo el siguiente orden:

1. Montaje de la ESP32-S3 emisora.
2. Subida del código de la ESP32-S3 emisora a la misma.
3. Montaje de la ESP32-S3 receptora.
4. Subida del código de la ESP32-S3 receptora a la misma y lectura por terminal de los mensajes recibidos.



The screenshot shows the Arduino IDE interface. The top toolbar includes icons for checking, running, and uploading code, along with a dropdown menu set to 'ESP32S3 Dev Module'. The main editor displays the 'Receptor.ino' file with the following code:

```
15
16 void loop()
17 {
18   if (rs485Serial.available() > 0)
19   {
20     // Read incoming data
21     byte incomingByte;
22     byte frame[8];
23     int idx = 0;
24
25     while (rs485Serial.available() > 0 && idx < 8)
26     {
27       incomingByte = rs485Serial.read();
28       frame[idx++] = incomingByte;
29       delay(3); // Small delay to ensure all data is read
30     }
31
32     // Check if the received frame is valid
33     if (frame[0] == 0xAA && frame[6] == 0xFE)
34     {
35       Serial.println("Received valid frame:");
36
37       // Print the received frame in hexadecimal
38       for (int i = 0; i < 8; i++)
```

Below the code editor is the 'Serial Monitor' window, which is open. It shows the output: 'Received valid frame:' followed by the hexadecimal data 'AA 23 4B 6F 6C 61 FE AA' and the text 'Mensaje recibido: Hola'. The baud rate is set to 115200.

Figura 1: Funcionamiento código RS485.

2.3. MONTAJE

MATERIAL NECESARIO

- ESP32-S3-WROOM-1.
- ESP32-S3 GPIO Extension Board.
- HW-519 (transceptor RS485).
- 11 cables, para realizar las conexiones pertinentes (en el montaje real, se emplean 8 cables macho-hembra y 3 cables hembra-hembra).

CONEXIONES

Debido al modelo de RS485 empleado, cualquiera de las dos ESP32-S3 puede actuar como emisora o receptora, siendo la conexión de ambas idéntica.

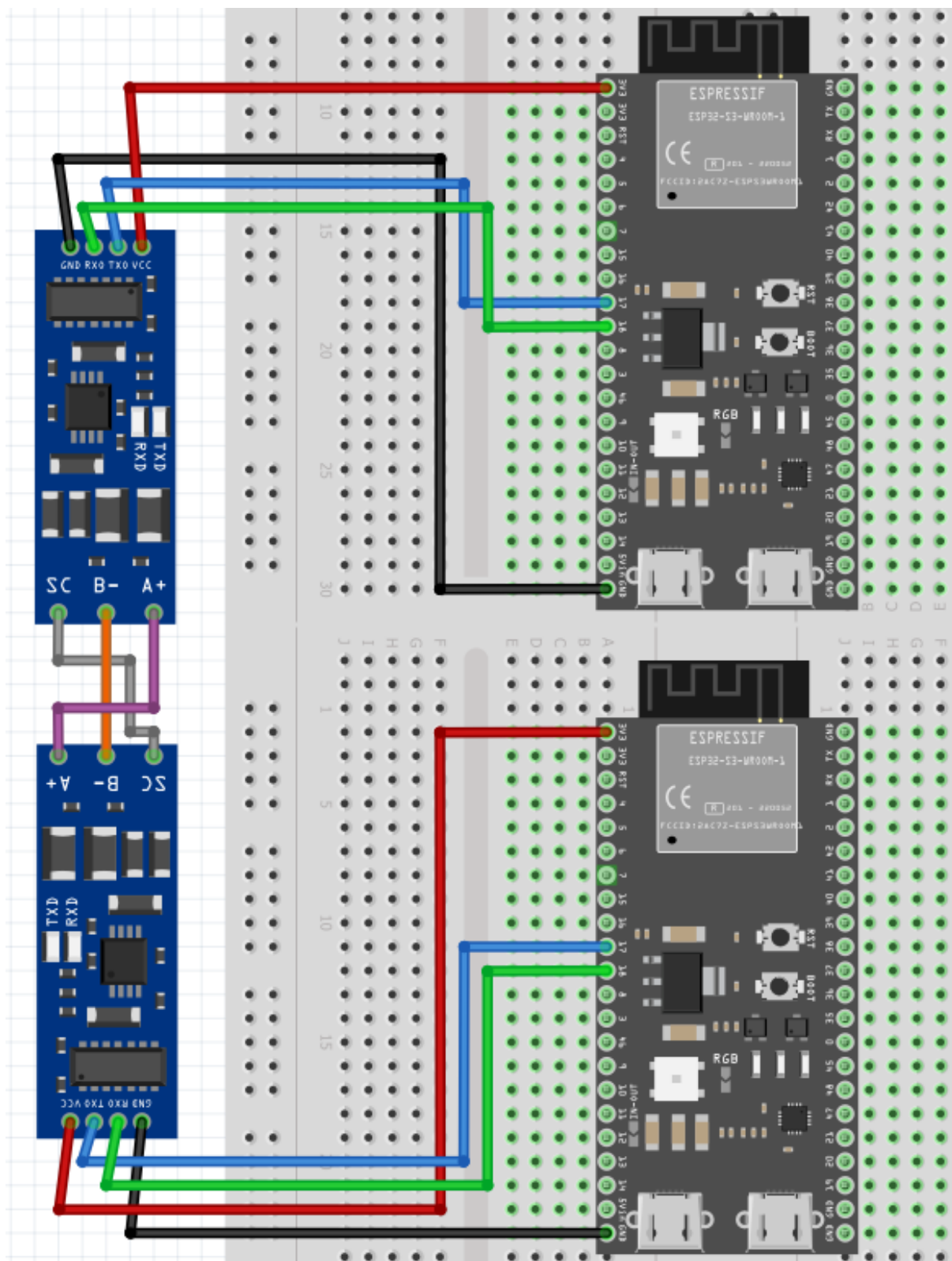


Figura 2: Conexiones RS485.

3. BIBLIOGRAFÍA

3.1. INFORMACIÓN SOBRE PROTOCOLOS Y SUS MONTAJES

- Naylamp Mechatronics. (n.d.). Comunicación RS485 con Arduino. Recuperado de https://naylampmechatronics.com/blog/37_comunicacion-rs485-con-arduino.html
- Llamas, L. (n.d.). Arduino RS485 (MAX485). Recuperado de <https://www.luisllamas.es/arduino-rs485-max485/>
- UsinaInfo. (n.d.). Conversor TTL para RS485 HW-519. Recuperado de <https://www.usinainfo.com.br/conversores-de-sinal/conversor-ttl-para-rs485-hw-519-8683.html>

3.2. ELEMENTOS EMPLEADOS PARA LOS MONTAJES

- Fritzing Forum. (2024). *Fritzing Part for MAX485 CD4069*. Recuperado de <https://forum.fritzing.org/t/fritzing-part-for-max485-cd4069/21343/6>
- Fritzing Forum. (2024). *ESP32-S3 DevKit with USB-C*. Recuperado de <https://forum.fritzing.org/t/esp32-s3-devkit-with-usb-c/24673>