Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

16 de mayo de 2025

TRABAJO ACADÉMICO

CAN

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 CAN **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*

PÁGINA EN BLANCO

LISTADO DE TABLAS

No se encuentran elementos de tabla de ilustraciones.

PÁGINA EN BLANCO

LISTADO DE FIGURAS

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

PÁGINA EN BLANCO

LISTADO DE CÓDIGO

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

PÁGINA EN BLANCO

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 CAN, presentando un resumen claro y coherente de los contenidos.

PÁGINA EN BLANCO

2. MONTAJE PROTOCOLO CAN

2.1. DEFINICIÓN

CAN (*Controller Area Network*) es un protocolo de comunicación diseñado para entornos de alta confiabilidad y baja latencia, ampliamente utilizado en la industria automotriz y la automatización industrial. Este protocolo es especialmente útil en aplicaciones que requieren una comunicación rápida y segura entre múltiples dispositivos en una misma red. CAN es conocido por su capacidad para priorizar mensajes y garantizar la transmisión de datos críticos sin retrasos.

En una red CAN, todos los dispositivos escuchan el bus y priorizan los mensajes según sus identificadores. Cuando un dispositivo envía un mensaje, todos los demás dispositivos en la red lo reciben y lo procesan según su prioridad. Esta arquitectura descentralizada y la capacidad de priorización de mensajes garantizan una comunicación eficiente y segura.

CARACTERÍSTICAS

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

- **Comunicación *peer-to-peer*:** En una red CAN, todos los dispositivos pueden enviar y recibir mensajes de manera directa, sin la necesidad de un controlador centralizado. Esto facilita la implementación de redes descentralizadas y robustas.
- **Priorización de mensajes:** CAN utiliza identificadores de mensajes para determinar la prioridad de transmisión. Los mensajes con identificadores más bajos tienen mayor prioridad, lo que garantiza que los datos críticos se envíen sin retrasos.
- **Alta velocidad:** Este protocolo soporta velocidades de transmisión de hasta 1 Mbps, lo que permite una comunicación rápida y eficiente en entornos donde la alta velocidad es crucial.
- **Hardware adicional:** Requiere hardware adicional, lo que puede aumentar los costos de implementación.
- **Configuración compleja:** La implementación de CAN puede ser más compleja que UART y RS485, especialmente en términos de configuración de la red y protocolos de capa superior.
- **Requisitos de protocolos de capa superior:** En algunas aplicaciones, puede ser necesario implementar protocolos de capa superior para gestionar la estructura y el flujo de los mensajes.

2.2. CÓDIGO

Para implementar CAN en Arduino, se utiliza un transceptor CAN. Los pines TX y RX se conectan a los pines correspondientes del transceptor.

Para la realización del código se ha empleado la biblioteca *ESP32-TWAI-CAN.hpp*. Además, se han empleado dos ESP32-S3 para realizar dicha comunicación simulando una conversación emisor y

receptor, donde el receptor solamente recibe mensajes de una identificación deseada. Se adjunta a continuación el código.

ESP32-S3 EMISORA

```
#include <ESP32-TWAI-CAN.hpp>

// Pines de transmisión y recepción CAN
#define CAN_TX 17
#define CAN_RX 18

void sendFrame()
{
    CanFrame frame           = {0};
    frame.identifier         = 0x7DF;  // ID estándar del mensaje
    frame.extd               = 0;
    frame.data_length_code   = 8;

    // Datos enviados en el mensaje CAN
    frame.data[0]            = 0x48; // 'H'
    frame.data[1]            = 0x6f; // 'o'
    frame.data[2]            = 0x6c; // 'l'
    frame.data[3]            = 0x61; // 'a'
    frame.data[4]            = 0xAA;
    frame.data[5]            = 0xAA;
    frame.data[6]            = 0xAA;
    frame.data[7]            = 0xAA;

    ESP32Can.writeFrame(frame);
}

void setup()
{
    Serial.begin(115200);
    ESP32Can.setPins(CAN_TX, CAN_RX);
    ESP32Can.setRxQueueSize(5);
    ESP32Can.setTxQueueSize(5);
    ESP32Can.setSpeed(ESP32Can.convertSpeed(500));

    if (ESP32Can.begin())
    {
        Serial.println("CAN bus iniciado correctamente.");
    } else
    {
        Serial.println("Error al iniciar CAN bus.");
    }
}
```

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

    if (currentStamp - lastStamp > 1000)
    {
        lastStamp = currentStamp;
        sendFrame();  // Envía el mensaje cada segundo
    }
}
```

Código 1: ESP32-S3 Emisora.

ESP32-S3 RECEPTORA

```
#include <ESP32-TWAI-CAN.hpp>

// Pines CAN
#define CAN_TX 17
#define CAN_RX 18

CanFrame frame; // Estructura para almacenar el frame recibido

void setup()
{
    Serial.begin(115200);
    ESP32Can.setPins(CAN_TX, CAN_RX);
    // Configuración de buffers
    ESP32Can.setRxQueueSize(5);
    ESP32Can.setTxQueueSize(5);
    ESP32Can.setSpeed(ESP32Can.convertSpeed(500));

    if (ESP32Can.begin())
    {
        Serial.println("CAN bus iniciado correctamente.");
    } else
    {
        Serial.println("Error al iniciar CAN bus.");
    }
}

void loop()
{
    // Lee un frame CAN con un timeout de 1000 ms
    if (ESP32Can.readFrame(frame, 1000))
    {
```

```
        Serial.printf("Received frame: %03X \r\n", frame.identifier);

        // Si el mensaje es el esperado, se reconstruye el texto
        if (frame.identifier == 0x7DF)
        {
            Serial.print("Mensaje recibido: ");

            char receivedMessage[5] = {0}; // Buffer para texto
            for (int i = 0; i < 4; i++)
            {
                receivedMessage[i] = static_cast<char>(frame.data[i]);
            }

            Serial.println(receivedMessage); // Muestra el mensaje como cadena
        }
    }
}
```

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.

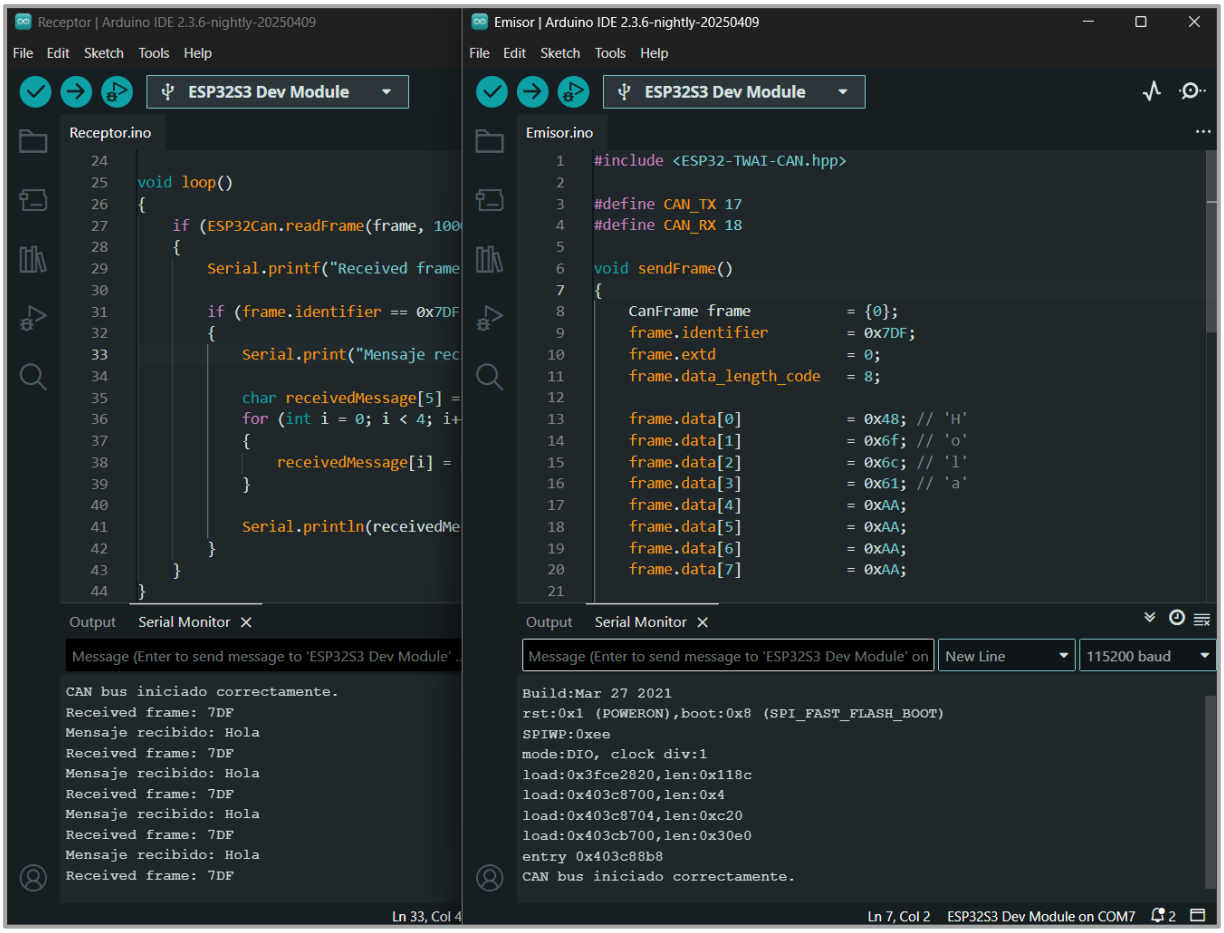

Figura 1: Funcionamiento código CAN.

2.3. MONTAJE

MATERIAL NECESARIO

- ESP32-S3-WROOM-1.
- ESP32-S3 GPIO Extension Board.
- WCMCU-230 (transceptor CAN)
- 10 cables, para realizar las conexiones pertinentes (en el montaje real, se emplean 6 cables macho-macho y 4 cables macho-hembra).

CONEXIONES

Cualquiera de las dos ESP32-S3 puede actuar como emisora o receptora, siendo la conexión de ambas idéntica.

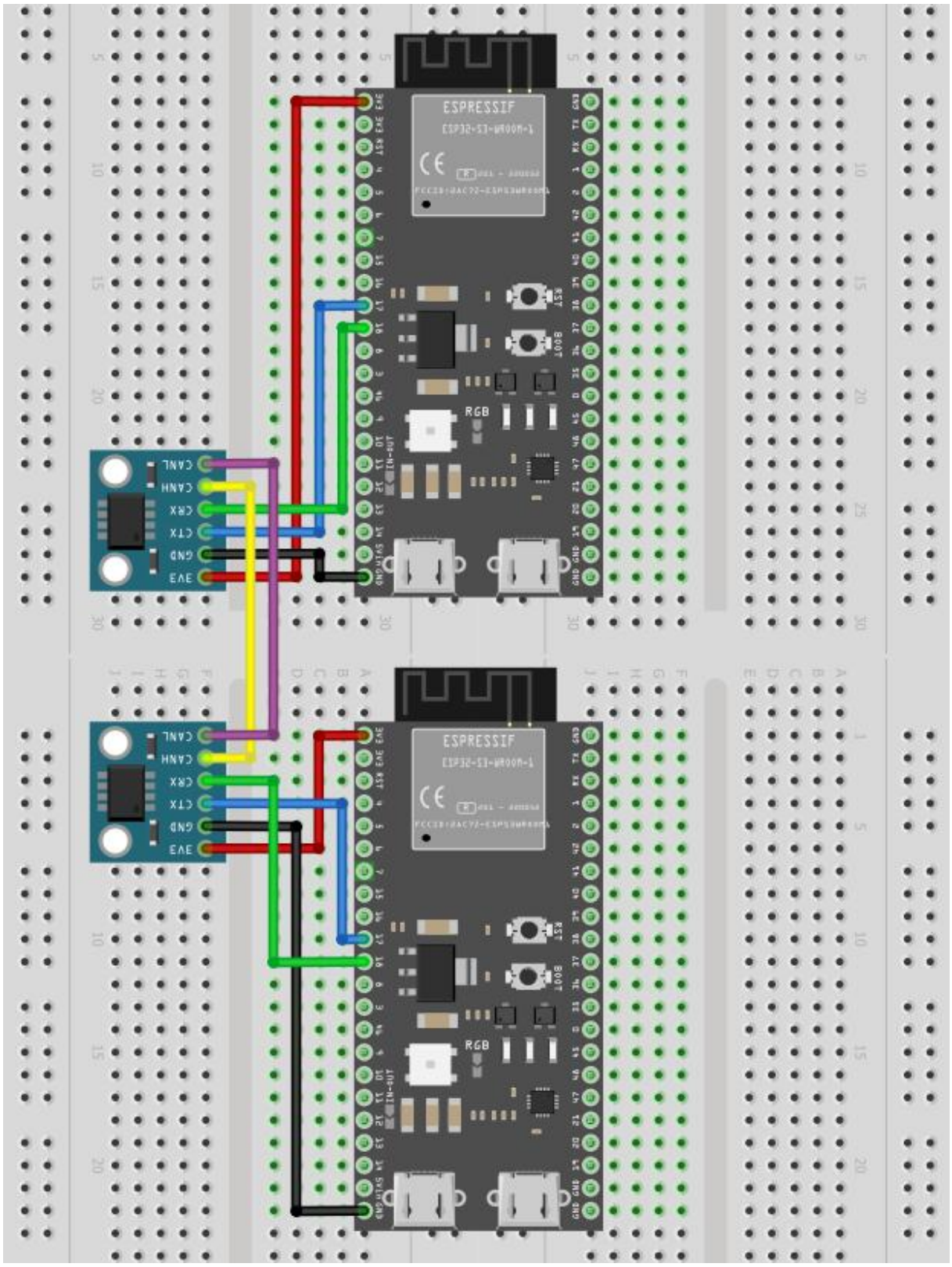

Figura 2: Conexiones CAN.

3. BIBLIOGRAFÍA

3.1. INFORMACIÓN SOBRE PROTOCOLOS Y SUS MONTAJES

- Arduino. (n.d.). MKR CAN Communication. Arduino Documentation. Recuperado de https://docs.arduino.cc/tutorials/mkr-can-shield/mkr-can-communication/
- Circuit Digest. (n.d.). Arduino CAN Tutorial – Interfacing MCP2515 CAN Bus Module with Arduino. Recuperado de https://circuitdigest.com/microcontroller-projects/arduino-can-tutorial-interfacing-mcp2515-can-bus-module-with-arduino

3.2. ELEMENTOS EMPLEADOS PARA LOS MONTAJES

- Fritzing Forum. (2024). *ESP32-S3 DevKit with USB-C*. Recuperado de https://forum.fritzing.org/t/esp32-s3-devkit-with-usb-c/24673
- Fritzing Forum. (2022). SN65HVD230 CAN Bus Transceiver. Recuperado de https://forum.fritzing.org/t/sn65hvd230-can-bus-transciever/15315