Reporte: Práctica de Laboratorio 7

| Victoria Rodríguez de León  *Ingeniería en Robótica y Sistemas Digitales*  *Instituto Tecnológico y de Estudios Superiores de Monterrey*  CDMX, México  [A01656328@tec.mx](mailto:A01661890@tec.mx) | Israel Macías Santana  *Ingeniería en Robótica y Sistemas Digitales*  *Instituto Tecnológico y de Estudios Superiores de Monterrey*  CDMX, México  [A01027029@tec.mx](mailto:A01661649@tec.mx) | Rodríguez Alanis Lisa Valeria  *Ingeniería en Robótica y Sistemas Digitales*  *Instituto Tecnológico y de Estudios Superiores de Monterrey*  CDMX, México  [A01656306@tec.mx](mailto:A01656306@tec.mx) |
| --- | --- | --- |

***Abstract — El protocolo CAN (Controller Area Network) es crucial en sistemas embebidos debido a su capacidad para permitir una comunicación robusta y eficiente entre múltiples microcontroladores en entornos complejos, como los encontrados en el área automotriz. El presente estudio se centra en la utilización de componentes de hardware para la implementación efectiva de un sistema CAN considerando la configuración de parámetros en STM32 además de la implementación de filtros y la definición de funciones para la transmisión y recepción de datos, con un enfoque en el manejo de errores y señalización LED basada en encabezados de mensajes. En la etapa final se desarrolla un tablero de control integrando los sistemas previamente desarrollados. Los resultados fueron validados con el funcionamiento adecuado del experimento. A través de este estudio se puede obtener un panorama general de conceptos que establecen una base sólida para la solución de futuros desafíos.***

***Keywords — CAN, ESP32, MCP2515, SN65HVD230, NUCLEOH74Interrupción, Dashboard ESP32, Arduino Cloud***

I. Introducción

El desarrollo y la integración de sistemas embebidos en diversas industrias han sido un pilar fundamental en la era de la automatización y la eficiencia operativa. Dentro de este contexto, el protocolo Controller Area Network (CAN) se ha establecido como un componente esencial, facilitando la comunicación y coordinación efectiva entre múltiples microcontroladores y dispositivos. Este protocolo es especialmente relevante en aplicaciones que requieren la gestión confiable y en tiempo real de una gran cantidad de señales y actuaciones. El presente estudio se enfoca en la exploración y aplicación del protocolo CAN en sistemas embebidos, destacando su papel en la eficacia de las comunicaciones en redes de dispositivos interconectados. A través de un enfoque práctico, esta investigación incorpora una serie de experimentos y aplicaciones prácticas utilizando hardware como NUCLEO H745ZI-Q, ESP32, Arduino Mega y otros componentes como MCP2515 y SN65HVD230. La metodología incluye pasos detallados para verificar el funcionamiento de cada componente, desde la configuración básica hasta la implementación de comunicaciones CAN más complejas.

Los ejercicios se estructuran alrededor de la configuración y prueba de componentes individuales, seguidos por la integración y el desarrollo de un sistema de comunicación CAN, primero a través de métodos de sondeo y luego mediante interrupciones. Este enfoque gradual permite una comprensión profunda de los principios del protocolo CAN y su aplicación en sistemas embebidos. Además, se explora la interacción de estos sistemas con la tecnología IoT, mediante la creación de un tablero de control interactivo con ESP32 y Arduino Cloud, lo que subraya la versatilidad y la aplicabilidad del protocolo CAN en entornos modernos de automatización e IoT.

Finalmente, el estudio presenta una evaluación detallada de los sistemas desarrollados, enfatizando la importancia de la documentación y la trazabilidad en proyectos de sistemas embebidos.

II. Marco Teórico

***A. CAN***

Controller Area Network (CAN) es un protocolo de comunicación estándar utilizado para permitir la comunicación confiable entre varios dispositivos de hardware (como microcontroladores, sensores, actuadores) dentro de un vehículo o sistema industrial sin la necesidad de un host central. Fue desarrollado originalmente por Bosch para su uso en automóviles, pero desde entonces ha encontrado aplicaciones en diversas industrias [2].

El CAN tiene las siguientes características principales [1]:

* Orientado a mensajes: La información que se va a intercambiar se descompone en mensajes a los que se les asigna un identificador y se encapsulan en tramas para su transmisión.
* Topología de bus: Los dispositivos se conectan a un mismo bus físico, lo que permite que cualquier dispositivo pueda transmitir o recibir datos en cualquier momento.
* Detección y corrección de errores: incluye mecanismos de detección y corrección de errores para garantizar la fiabilidad de la comunicación.
* Alta velocidad: se pueden transmitir datos a velocidades de hasta 1 Mbit/s.

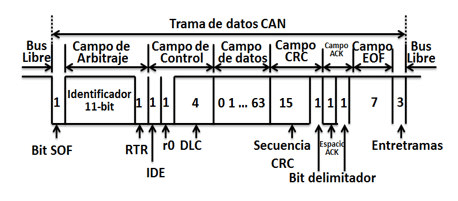
La arquitectura de un sistema CAN consta de los siguientes elementos [1]:

* Nodos: Son los dispositivos que se conectan al bus CAN. Cada nodo consta de un CPU, un controlador CAN y un transceptor
* Bus: Es el medio físico que conecta los nodos. Se basa en un cable de dos hilos, uno para la transmisión y otro para la recepción
* Controlador CAN: Se encarga de la transmisión y recepción de datos adecuada en el bus.
* Transceptor: Adapta los niveles de señal de los datos enviados y recibidos por el nodo.

***B. Trama de datos CAN***

Es el formato principal utilizado para la transmisión de datos entre los nodos de una red CAN. Sus componentes clave son los siguientes [2]:

* SOF(State of Frame): marca el inicio de la trama de datos CAN.
* Identificador: define la prioridad del mensaje. En CAN estándar, tiene 11 bits, mientras que en CAN extendido es de 29 bits. Cuanto más bajo es el valor numérico del identificador, mayor es la prioridad del mensaje.
* RTR (Remote Transmission Request): Un bit que indica si se trata de una trama de datos (dominante) o una solicitud de trama remota (recesiva).
* IDE (Identifier Extension Bit): Un bit que muestra si el identificador es estándar (dominante) o extendido (recesivo).
* Bits reservados para uso futuro
* DLC (Data Length Code): Es un campo de 4 bits que indica el número de bytes de datos que la trama está transportando (0 a 8 bytes).
* Data Field: Se basa en los datos reales que se transmiten, que pueden ser de hasta 8 bytes.
* CRC (Cyclic Redundancy Check): Contiene un valor de suma de comprobación para la detección de errores.
* Delimitador CRC: Un bit de espacio (recesivo) que sigue al campo CRC.
* ACK: El transmisor envía un bit recesivo y el receptor lo sobrescribe con un bit dominante si ha recibido correctamente la trama.
* Delimitador ACK: bit de espacio (recesivo) que sigue al bit de ACK.
* EOF (End of Frame): Un campo de 7 bits de espacio (recesivo) que indica el final de la trama.
* Intermisión:Tres bits de espacio donde no se permite el inicio de la siguiente trama. Esto garantiza un tiempo mínimo entre tramas sucesivas.
* FCS (Frame Check Sequence): Secuencia utilizada para verificar la integridad de la trama transmitida.

******

***Fig. 1. Trama de datos CAN [4].***

Una de las maneras en las que se puede configurar CAN es por interrupción, lo que corresponde auna técnica avanzada para gestionar la conexión entre dispositivos cuando se hace uso del protocolo. Este enfoque mejora la eficiencia y capacidad de respuesta. Para esto se debe de hacer la configuración en el microcontrolador de manera que al recibir un mensaje CAN, se genere una interrupción y el microcontrolador ejecute una función de servicio de interrupción (ISR) para manejar ese mensaje.

Es crucial configurar los filtros CAN y los búferes para asegurar que sólo se procesen los mensajes relevantes. Los filtros se configuran para permitir solo ciertos identificadores de mensaje y tipos de trama, lo que es esencial para la operación correcta en una red CAN con múltiples nodos.

***C. ESP32***

Es un módulo de microcontrolador de 32 bits con un procesador dual-core de hasta 240 MHz, Wi-Fi de doble banda y Bluetooth de baja energía utilizado para aplicaciones de IoT y robótica. Tiene ventajas como lo es el soporte para I2C, SPI, UART, ADC además de que ofrece una amplia gama de pines de Entrada/Salida general (GPIO). Para el caso de CAN, el ESP32 se emplea para establecer una red de comunicación conectándose con un módulo MCP2515 (un controlador CAN independiente) a través de su puerto SPI. También permite su uso para la integración del sistema CAN a aplicaciones de IoT y desarrollo de interfaz de usuario.

***D. MCP2515***

Es un controlador independiente para CAN ampliamente utilizado, especialmente en aplicaciones donde el controlador principal no tiene un controlador CAN integrado. Se comunica con microcontrolador por SPI e incluye capacidades de filtrado de mensajes y gestión de errores para mantener la integridad de la comunicación [3].

***E. SN65HVD230***

Es un transceptor que actúa como un puente entre un controlador CAN y el bús físico. Cumple la funcionalidad de convertir las señales transmitidas por el controlador CAN en señales adecuadas para el bus y viceversa. Esto incluye la conversión de los niveles de voltaje de los datos digitales en señales que pueden ser transmitidas a través del bus CAN. Tiene un bajo consumo de energía y es capaz de manejar las velocidades de comunicación típicas de una red CAN [5].

***F. Dashboard***

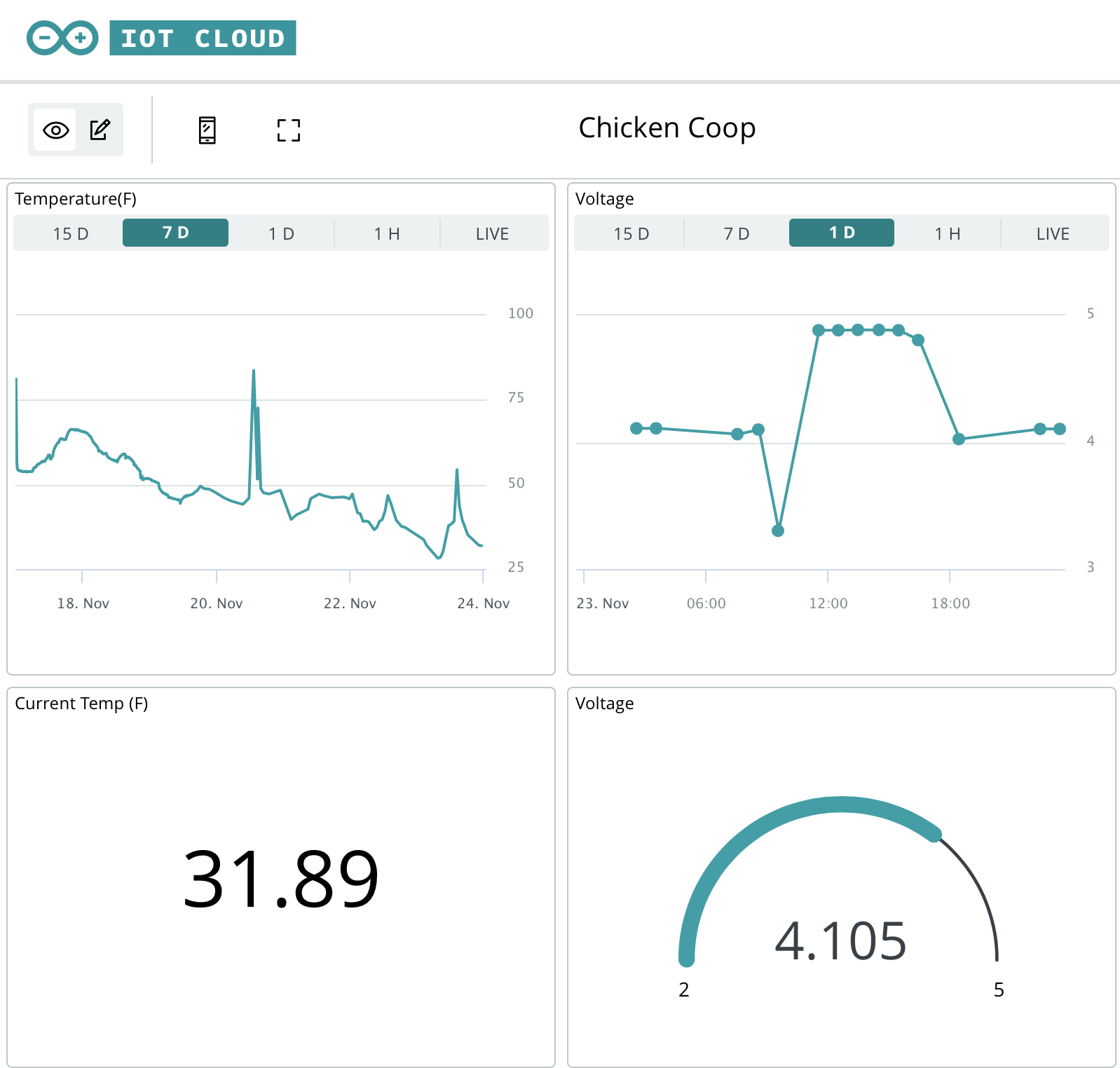
Es una plataforma que muestra información en tiempo real y proporciona control interactivo sobre los dispositivos y sistemas conectados. Para esta aplicación se usa el ESP32. Estos son algunos aspectos clave:

* Visualización de datos: recopilados por el ESP32
* Control remoto: realizado por el usuario
* Conectividad: aprovechando la capacidad del ESP32 para conectarse a redes Wi-Fi o Bluetooth para transmitir datos
* Interfaz de Usuario: presentación del dashboard, muestra información y responde a las entradas del usuario.

Un dashboard ESP32 podría usarse para mostrar datos recopilados a través del sistema CAN y para controlar dispositivos o actuadores conectados a la red CAN. También podría interactuar con el ESP32 para enviar comandos basados en la entrada del usuario desde el dashboard, integrando así la funcionalidad de IoT en el sistema [6].

***G. Arduino IoT Cloud***

Arduino Cloud es una plataforma en línea desarrollada por Arduino que permite a los usuarios crear, conectar y controlar sus proyectos de Arduino desde cualquier lugar del mundo a través de Internet. Esta plataforma ofrece una variedad de herramientas y servicios que facilitan la integración de proyectos basados en Arduino con la Internet de las Cosas (IoT). Algunas funciones principales son la capacidad de programación remota, dashboard personalizable, conectividad con diversos dispositivos, integración con servicios de terceros, entre otros [7].



***Fig. 2. Arduino Cloud [7]***

III. Metodología

***A. Ejercicio I: CAN con MCP2515***

El objetivo de este ejercicio se basó en comprobar la conexión por protocolo CAN entre las tarjetas de desarrollo ESP32 Wroom y Arduino Mega utilizando los controladores CAN MCP2515. Para que la conexión pueda tener éxito se necesita comunicar el MCP2515 mediante el protocolo SPI (Serial Peripheral Interface) a cada una de las tarjetas. Para que la comunicación CAN exista, las terminales de H y L (*high* y *low*) de cada chip tienen que estar conectadas entre sí por medio de un par de cables trenzados de 14 mm, pues esto permitirá que la inductancia en los cables sea mejor, reduciendo el ruido que puede haber en el ambiente (véase fig. 1 para conocer el diagrama de conexiones).

Para llevar a cabo el ejercicio de comunicación, se instaló la biblioteca de CAN ‘Seeed\_Arduino\_CAN’ y se cargaron los programas ‘CANsend’ en la ESP32 y ‘CANmonitor’ en el Arduino Mega. El programa 'CANsend', ejecutado en la ESP32, prepara dos buffers de caracteres con distintos datos y asigna un identificador único a cada uno, conocidos como headers. Estos datos son enviados a través del protocolo CAN. Por otro lado, el programa 'CANmonitor' en el Arduino Mega está diseñado para recibir estas tramas de información. Verifica el identificador de cada header que llega y muestra tanto la trama recibida como su identificador correspondiente en el monitor serial. Para completar el ejercicio, se abren dos monitores seriales, uno para cada dispositivo, permitiendo observar la comunicación entre la ESP32 y el Arduino Mega. Este procedimiento no solo verifica la funcionalidad de los MCP2515 y la comunicación CAN, sino que también demuestra la capacidad de enviar y recibir información estructurada entre dos microcontroladores diferentes.

***B. Ejercicio II: CAN por polling***

El objetivo de este ejercicio es comprobar la conexión por protocolo CAN por método de *polling* entre las tarjetas de desarrollo ESP32 Wroom, Arduino Mega y STM32 Núcleo H745ZiQ. Una vez que la conexión entre las tarjetas ESP32 y Arduino Mega haya sido exitosa, se tuvo que realizar la conexión adyacente de la STM32 para la comunicación CAN por medio de un *transceiver* SN65HV. En la tarjeta se configuró la interfaz de CAN1 para hacer la conexión con el *transceiver,* donde los pines de CANTx y CANRx se conectan entre ellos, y los pines de H y L, se conectan a las mismas terminales del chip MCP2515 que está conectado a la ESP32 Esto va a causar que los datos enviados por la Núcleo lleguen al chip MCP2515 conectado a la ESP32 y este envíe la trama al otro MCP conectado al Arduino Mega.

**Configuración de CAN en STM** La configuración de CAN para la tarjeta Núcleo tiene que tener los parámetros establecidos en la Tabla 1. La cual de forma general contiene la habilitación de auto retransmisión, pausa de transmisión, el modo de la interfaz, el tamaño de los buffers de transmisión y de recepción, así como la estructura con la que los datos se van a acomodar, entre otros. Además, se tiene que configurar un filtro para seleccionar y priorizar la transmisión de algunos mensajes. El filtro configurado para el programa está en formato clásico e identifica valores estándar. Véase *Tabla 1* para conocer los parámetros de configuración.

| Campo | Valor |
| --- | --- |
| Frame format | Classic mode |
| Mode | Normal mode |
| Auto retransmission | Enable |
| Transmit pause | Enable |
| Protocol exception | Enable |
| Nominal prescaler | 5 |
| Nominal sync jump width | 8 |
| Nominal time seg1 | 0x1F |
| Nominal time seg2 | 8 |
| Data prescaler | 1 |
| Data sync jump width | 1 |
| Data time seg1 | 1 |
| Data time seg2 | 1 |
| Message ram offset | 0 |
| Std filters nbr | 1 |
| Ext filters nbr | 0 |
| Rx fifo0 elms nbr | 1 |
| Rx fifo0 elmt size | 8 |
| Rx fifo0 elms nbr | 0 |
| Rx fifo0 elmt size | 8 |
| Rx buffer nbr | 0 |
| Rx buffer size | 8 |
| Tx events nbr | 0 |
| Tx buffer nbr | 0 |
| Tx fifo queue elmts nbr | 1 |
| Tx fifo queue mode | FIFO |
| Tx elmt size | 8 |

***C. Ejercicio III: CAN por interrupción***

***D. Ejercicio IV: CAN con IoT***

Para este ejercicio se implementó un dashboard con ESP32 y Arduino Cloud, en donde se utilizó un hotspot generado por un teléfono celular para realizar la conexión. Dentro del dashboard generado, se crearon 2 botones y 2 LEDs que se configuraron para que al presionar el botón 1, se envíe el mensaje 1 del inciso anterior (cuya funcionalidad es de CAN por interrupción) y al presionar el botón 2, se envíe el mensaje 2, correspondientemente. También se configuró la NUCLEO H745Z1-Q, habilitando en STM32CubeIDE los pines correspondientes a los LEDs color rojo y verde para que al recibir el primer mensaje, se encienda el LED 1 y lo mismo para el segundo mensaje, con el LED 2.

IV. Resultados

***A. Ejercicio I: CAN con MCP2515***

***B. Ejercicio II: CAN por polling***

***C. Ejercicio III: CAN por interrupción***

***D. Ejercicio IV: CAN con IoT***

V. Conclusiones Individuales

**Victoria Rodríguez de León (A01656328)**

En esta práctica se abordaron conceptos fundamentales relacionados con el diseño y la implementación de sistemas embebidos avanzados utilizando CAN. Se comenzó con la verificación y configuración del ESP32 y Arduino Mega con el controlador CAN MCP2515, seguido por la implementación de un sistema de comunicación CAN en la NUCLEO H745ZI-Q con el transceptor SN65HVD230. Esto incluyó la configuración detallada de los parámetros de la red, como el formato de trama, el modo de operación, y las configuraciones de retransmisión automática y pausa de transmisión, así como la implementación de filtros y buffers de mensajes. La práctica evolucionó hacia la creación de un sistema CAN por polling y luego por interrupción, lo que permitió una comprensión más profunda de las diferencias en la eficiencia y la capacidad de respuesta entre estos dos métodos. Se hizo hincapié en la programación cuidadosa para el manejo de errores y la interacción entre los dispositivos a través del sistema CAN, incluyendo el control de LEDs en respuesta a mensajes específicos. Finalmente, se integró un dashboard usando ESP32 y Arduino Cloud para proporcionar una interfaz de usuario para interactuar con el sistema. Esto no solo demostró la aplicación práctica de las redes CAN en sistemas embebidos, sino también su integración con tecnologías IoT, ampliando así el alcance y la funcionalidad del sistema.

Referencias bibliográficas

[1] Corrigan, S. (2002). Introduction to the Controller Area Network (CAN) Application Report Introduction to the Controller Area Network (CAN). <https://www.ti.com/lit/an/sloa101b/sloa101b.pdf?ts=1699818320691&ref_url=https%253A%252F%252Fwww.google.com%252F>

[2] National Instruments. (2023). Controller Area Network (CAN) Overview & Specifications. Ni.com. <https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/controller-area-network--can--overview.html>

[3]Microchip. (2023). MCP2515. Microchip.com. <https://www.microchip.com/en-us/product/mcp2515>

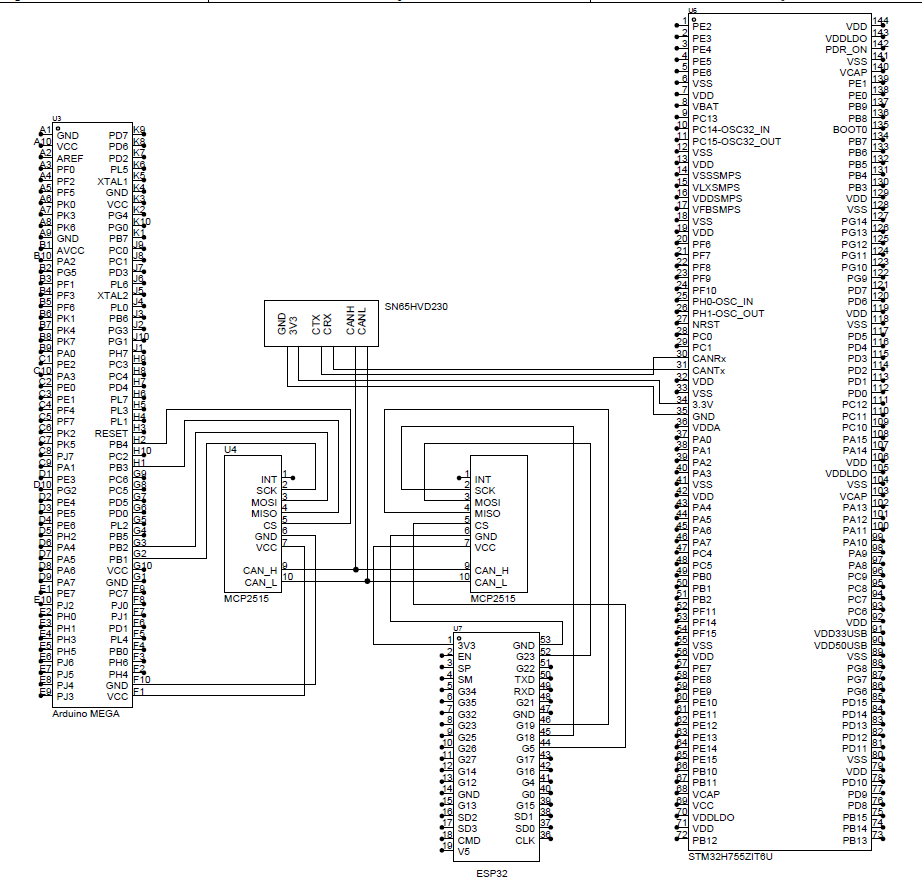
[4] Cook, J., & Freudenberg, J. (n.d.). Controller Area Network (CAN) EECS 461, Fall 2008 \*. https://www.eecs.umich.edu/courses/eecs461/doc/CAN\_notes.pdf

[5] Texas Instruments. (2018). SN65HVD230. Ti.com. <https://www.ti.com/product/SN65HVD230>

[6] Arduino. (2023). ESP-DASH - Arduino Reference. Arduino.cc. <https://www.arduino.cc/reference/en/libraries/esp-dash/>

[7] Arduino Cloud. (2023). Arduino.cc. https://cloud.arduino.cc/

Anexos



*Anexo 1. Diagrama de Conexiones.*