Innovación y Desafíos en el Protocolo CAN: Análisis del Bus de Datos y Aplicaciones Modernas

Kendell Calderón Láscarez, Kaleb Granados Acuña, Lorenzo Sancho Fallas
kendelljr@estudiantec.cr kalebgranac13@estudiantec.cr Lorenzo7sancho@estudiantec.cr
Escuela de Ingeniería Mecatrónica
Instituto Tecnológico de Costa Rica

***Resumen*—Este artículo analiza la evolución, aplicaciones y desafíos del protocolo de comunicación Controller Area Network (CAN), un estándar ampliamente utilizado en la industria automotriz y otros sectores. A lo largo del documento, se discuten las principales versiones de CAN, incluyendo CAN 1.2 y CAN 2.0 (A y B), así como los diversos campos que componen los mensajes CAN. También se presentan las diferencias entre los tipos de tramas, como las de datos, de error y de sobrecarga. Además, se revisan las topologías y arquitecturas basadas en CAN, destacando las configuraciones más comunes y las mejoras propuestas por la comunidad industrial. Finalmente, se muestra un ejemplo práctico de comunicación CAN utilizando Arduino UNO R4 Minima, incluyendo los resultados de transmisiones en modo simplex y half-duplex, así como un análisis detallado del Data Frame.**

***Palabras clave*—Controller Area Network (CAN), Protocolo de comunicación, Trama de datos, Transceptores, Topología de bus**

I. Introducción

El protocolo de comunicación serial Controller Network Area nace para suplir la necesidad de complejidad en sistemas electrónicos en automóviles en los años 80. Con CAN se utilizan dos cables, uno de datos y el otro es tierra, por lo que disminuye la cantidad de cables, el peso y el costo que tenía el vehículo. Este sistema facilita el mantenimiento y ayuda a la ampliación de este. CAN se ha expandido a otros campos como la automatización de fábricas, robótica y dispositivos médicos debido a sus características, como el bajo costo, la detección de errores incorporada y su capacidad de resolución de contenciones de manera determinista [1].

El protocolo CAN se destaca por su priorización de mensajes, lo que garantiza que aquellos con mayor prioridad accedan primero al bus, y asegura tiempos de latencia predecibles, esenciales en sistemas de tiempo real. Además, es altamente flexible, permitiendo añadir o eliminar nodos sin reconfiguración. La red soporta recepción multicast con sincronización de tiempo, garantizando la consistencia de los datos a nivel de sistema.

El protocolo CAN es un sistema multimaster y también atómico, lo que significa que cualquier nodo puede iniciar la transmisión cuando el bus está libre, y además asegura que los mensajes se transmiten en su totalidad o no se transmiten, evitando la pérdida parcial de datos. Incluye robustos mecanismos de detección y corrección de errores, con retransmisión automática de mensajes corruptos. El protocolo diferencia entre fallos temporales y permanentes en los nodos, apagando automáticamente aquellos con fallos permanentes para mantener la integridad de la red [1].

II. Antecedentes

En lo que respecta a las normativas que regulan el protocolo, en 1993 se formalizó su estandarización mediante la norma ISO 11898, la cual estableció la capa de enlace de datos y los parámetros para la transmisión de alta velocidad. Posteriormente, en 1995, se incorporó un formato de trama extendido con un identificador de 29 bits.

Con el tiempo, ISO 11898 se dividió en partes, cubriendo la transmisión de alta velocidad (ISO 11898-2) y la transmisión tolerante a fallos y de bajo consumo (ISO 11898-3). Además, surgieron otras aplicaciones de CAN, como la comunicación entre camiones y remolques (ISO 11992) y los diagnósticos de vehículos (ISO 15765). Se han implementado variantes como TTCAN (ISO 11898-4), aunque no han tenido una adopción amplia en la industria [5].

La tecnología CAN ha evolucionado para ser utilizada no solo en automóviles, sino también en automatización industrial, agricultura, y vehículos ferroviarios, con perfiles de aplicación como CANopen.

III. Versiones de CAN

El protocolo tiene la peculiaridad de cambiar su arquitectura según la versión que se esté utilizando, esto debido a que con los años sus aplicaciones se han ampliado, por lo que su arquitectura de capas difiere con cada versión, pero se puede resaltar que las versiones más comunes son la CAN 1.2 y CAN 2.0. Luego se profundizará en todas las modificaciones que se les han realizado a estas arquitecturas.

III-A. CAN 1.2 - CAN 2.0A

Esta versión de CAN se puede tomar como una de las más tradicionales y al mismo tiempo robustas, ya que se trabajó de la misma manera que se conoce el CAN 2.0A (Se podría decir que son iguales). Este modelo permite un bitrate de hasta 1 Mbit/s para una red de 40 metros, lo cual también resalta su velocidad ante otros sistemas de comunicación serial, debido a que se confía que este protocolo funcione de manera confiable y rápida, ya que como se mencionó anteriormente, estos sistemas se implementan en automóviles en subsistemas como el “anti-lock braking system” el cual pone en riesgo la vida de las personas a bordo.

De la misma manera, la frecuencia de operación esta ligada a este bitrate, su rango va de **10kbit/s** a **1Mbit/s** como se mencionó. El controlador utiliza el clock interno para generar la señal de transmisión y recibir los bits correctamente. La frecuencia de operación se determina a partir del bit timing [1], que es una configuración del controlador CAN que asegura que la red pueda transmitir y recibir datos de manera fiable, ajustando parámetros como la sincronización y la longitud de los segmentos de tiempo [20].

Al hablar de la arquitectura por la que se rige esta versión, descompone la capa de enlace o "data link", del modelo OSI, en dos secciones llamadas: "CAN Object Layer" y "CAN Transfer Layer". Por lo que la versión de CAN 1.2 comprime las capas altas en solamente la capa de aplicación y las capas bajas en la capa física y las anteriormente mencionadas. En la figura 1, se puede observar la estructura de CAN 1.2.

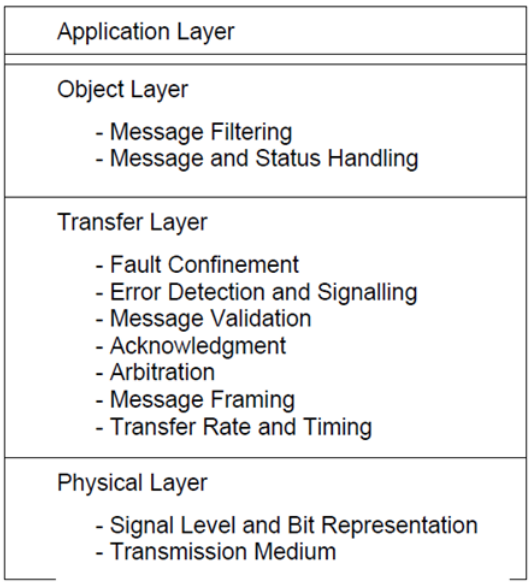

Figura 1. Estructura de capas de un nodo CAN 1.2. [1]

Para desglosar lo demostrado en la figura 1, se puede decir que la Capa Física define cómo se transmiten las señales a través del medio, permitiendo flexibilidad en la implementación según las necesidades de la aplicación. La Capa de Transferencia es el núcleo del protocolo, gestionando la transmisión y recepción de mensajes, la temporización, el arbitraje, la detección de errores y el confinamiento de fallos, asegurando una comunicación eficiente y confiable. Por último, la Capa de Objetos filtra los mensajes recibidos y maneja su estado, determinando qué datos deben procesarse en cada nodo [1].

Algo que destaca de las diferentes versiones de CAN es la estructura de sus mensajes, en el caso de CAN 1.2 tiene diversos tipos de tramas que ayudan al manejo de datos del protocolo, se tienen:

- **Data Frame**: Es un mensaje estándar de CAN. La trama de datos está estructurada de la manera en que se muestra en la figura 2:

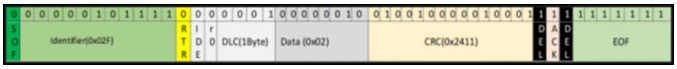

Figura 2. Estructura de capas de un nodo CAN 1.2. [1]

  - **Start of Frame (SOF)**: Es un único bit dominante que marca el comienzo del frame. Indica a todas las estaciones que sincronizarse para comenzar la transmisión.
  - **Arbitration Field**:
    - **Identifier**: Son 11 bits que contienen el identificador del mensaje. Estos bits determinan la prioridad del mensaje en la red. Los mensajes con identificadores más bajos tienen mayor prioridad.
    - **RTR Bit (Remote Transmission Request)**: Es un bit que indica si el frame es una solicitud remota o un frame de datos. Para los Data Frames, este bit es dominante (0).
  - **Control Field**:
    - **IDE (Identifier Extension)**: Indica si el frame es en formato estándar (11 bits) o extendido (29 bits). En el formato estándar de CAN 2.0A, este bit es dominante.
    - **Reserved bits**: Son bits reservados para futuros usos, que deben ser transmitidos como dominantes.
    - **DLC (Data Length Code)**: Son 4 bits que especifican la cantidad de bytes de datos que contiene el frame (entre 0 y 8 bytes).
  - **Data Field**: Puede contener de 0 a 8 bytes de datos. La longitud de este campo es determinada por el DLC.
  - **CRC Field**: Este campo contiene un código de redundancia cíclica (CRC) de 15 bits, seguido de un delimitador CRC de 1 bit. Es utilizado para la detección de errores durante la transmisión.
  - **ACK Field**:
    - **ACK Slot**: Es un bit dominante que indica que al menos una estación receptora ha recibido correctamente el mensaje.
    - **ACK Delimiter**: Es un bit recesivo que separa el ACK Slot del siguiente campo.
  - **End of Frame (EOF)**: Consiste en 7 bits recesivos que marcan el final del Data Frame.
- **Remote Frame**: Se utiliza para la solicitud de transmisión de datos desde otro nodo en la red. Su principal diferencia es que la trama no contiene un campo de datos. En este caso, se utiliza un bit RTR que indica que el nodo transmisor está solicitando datos de otro nodo. Es una herramienta funcional para que un nodo sepa información o estado de otro nodo sin transmisión de datos.

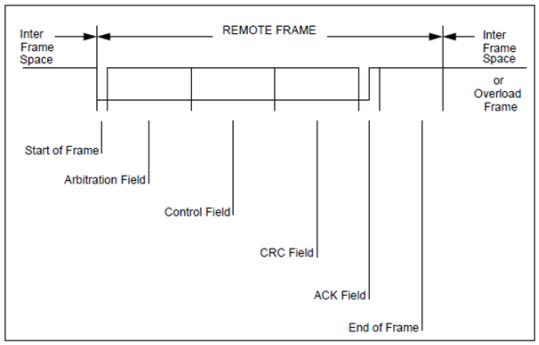

Figura 3. Trama remota de CAN Estándar. [1]

- **Error Frame**: Cuando se detecta un error en la transmisión, cualquier nodo puede transmitir esta trama. Su propósito es interrumpir la transmisión actual para alertar a todos los nodos sobre la presencia del error. Consta del "Error Flag", el cual se compone de seis bits dominantes que violan la regla de bit stuffing, causando que todos los nodos de la red indiquen error flags. También, se conforma de su respectivo delimitador de 8 bits recesivos, los cuales deshabilitan el nodo en caso de que los errores persistan [4].

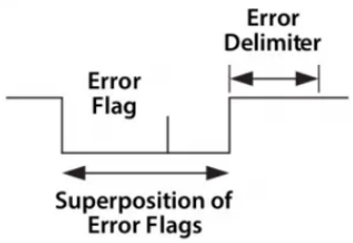

Figura 4. Trama de error en CAN estándar. [4]

- **Overload Frame**: Es utilizado en necesidad de más tiempo de procesado de los mensajes por parte de un nodo. Esto lo realiza mediante un retraso entre tramas de datos o tramas remotas. Cuenta con seis bits dominantes que indican el "Overload" y un delimitador de 8 bits recesivos que evitan que se envíen mensajes nuevos hasta resolver lo sobrecargado.
- **Interframe Space**: Espacio entre trama de datos o tramas remotas, los cuales son 3 bits recesivos que reciben el nombre de Intermission e incluyen el tiempo de bus ocioso. Estos evitan que los nodos transmitan mediante el proceso de intermisión. Caso que se detecte la transmisión de uno, se genera una bandera de sobrecarga hasta que el bus ocioso continúa hasta que ningún nodo tenga algo que transmitir. En este momento, se da la próxima transmisión.

III-B. CAN 2.0B

En el caso del CAN 2.0B, a pesar de ser una extensión del CAN 2.0A, tienen algunas diferencias notorias. Primeramente, el CAN 2.0B extiende la longitud del identificador a 29 bits, aumentando la cantidad de identificadores posibles a un número mucho mayor. Además, la versión B es compatible con ambos formatos de identificador [1].

Cabe mencionar que, si existiera un mensaje que tenga un formato de 11 bits y otro de 29 bits, ambos teniendo el mismo identificador, el mensaje del formato estándar tiene la prioridad debido al proceso de arbitraje del bus. Este diseño del bit se le conoce como SRR o *Substitute Remote Request*. Como ventaja de esta característica, tiene una aplicación más útil para sistemas más grandes y complejos, con mayor cantidad de dispositivos conectados [24].

El CAN 2.0B tiene una arquitectura conformada por tres secciones como se puede ver en la figura 5, que se definen como:

- **Capa Física**: Consiste en la definición de cómo se transmiten los bits a través del bus. No se limita a un tipo específico de medio, permitiendo flexibilidad en diferentes implementaciones y aplicaciones.
- **Sublayer de Acceso al Medio (MAC)**: Encargado de enmarcar los mensajes y arbitrar cuando varios nodos intentan transmitir simultáneamente, además de manejar las señales de reconocimiento y detección de errores.
- **Sublayer de Control de Enlace Lógico (LLC)**: Realiza el filtrado de mensajes, notificación de sobrecarga y la gestión de recuperación.

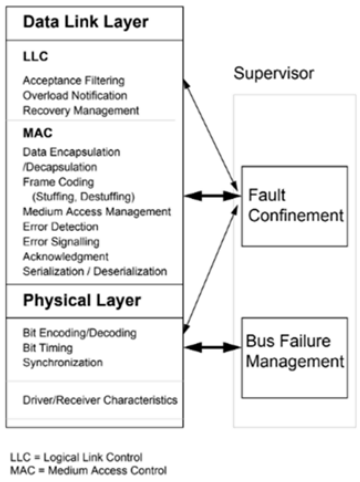

Figura 5. Arquitectura de CAN 2.0B [1]

El formato de la trama de mensaje en el CAN 2.0B es similar al CAN 2.0A, con la principal diferencia en la posibilidad de una trama extendida o estándar. A continuación, se detallan las principales diferencias que hay entre la trama de datos de CAN 2.0A y CAN 2.0B, estas se pueden observar en las figuras 2 y 6 respectivamente:

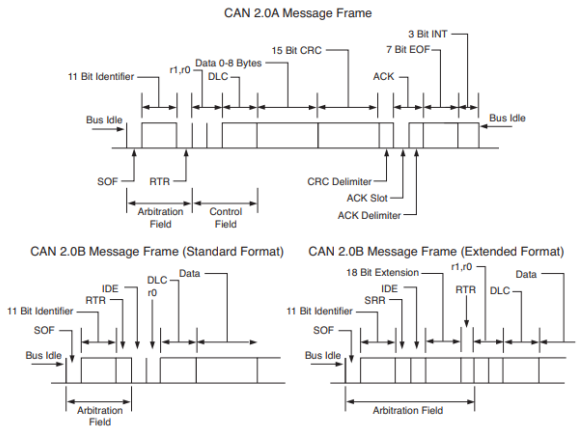

Figura 6. Trama de datos de CAN2.0B . [24]

- **Arbitration Field:**
  - **Identifier:** En CAN 2.0B, el identificador puede ser de **29 bits** (11 bits del formato estándar más 18 bits adicionales del formato extendido). Esto contrasta con CAN 2.0A, que solo utiliza un identificador de **11 bits**.
  - **SRR Bit (Substitute Remote Request)**: Este bit es exclusivo de CAN 2.0B cuando se utiliza el formato extendido. Es un bit **recesivo** que sustituye al bit RTR en las tramas extendidas, permitiendo que las tramas estándar tengan prioridad sobre las tramas extendidas durante el arbitraje [25].
- **Control Field:**
  - **Bit IDE (Identifier Extension)**: En CAN 2.0B, el bit IDE es **recesivo** si el frame utiliza el identificador extendido de 29 bits. En CAN 2.0A, este bit es **dominante**, ya que solo admite el identificador de 11 bits [25].
  - **Reserved Bits (R1 y R0)**: En **CAN 2.0B**, se utilizan dos bits reservados (R1 y R0), que deben transmitirse como dominantes. En CAN 2.0A solo había un bit reservado.

En CAN 2.0A, el Remote Frame utiliza únicamente un identificador de 11 bits para solicitar un Data Frame de otro nodo sin enviar datos. Sin embargo, en CAN 2.0B, se introduce la posibilidad de usar un identificador de 29 bits (formato extendido), lo que permite solicitudes de tramas con identificadores más largos y una mayor flexibilidad en el manejo de la red.

Por otro lado, la Error Frame no cambia en cuanto a su estructura entre CAN 2.0A y CAN 2.0B, aunque la presencia del identificador extendido en CAN 2.0B podría influir en las condiciones de arbitraje y detección de errores, especialmente cuando se trabaja con tramas de 29 bits. Aun así, la forma en que se genera y maneja la Error Frame sigue siendo la misma en ambas versiones.

Finalmente, la Overload Frame no sufre ningún cambio estructural entre CAN 2.0A y CAN 2.0B, ya que su función de solicitar pausas debido a sobrecargas en los nodos receptores es consistente en ambas versiones.

Al haber aclarado las principales diferencias entre las dos versiones de CAN, se puede describir un poco más del protocolo

IV. Manejo de Errores

El manejo de errores en el protocolo CAN está diseñado para asegurar la fiabilidad de la comunicación al detectar, notificar y corregir fallos en los mensajes transmitidos por el bus. Cuando un error es detectado, el nodo afectado genera una señal de error (ya sea activa o pasiva, dependiendo del estado del nodo) y transmite un **ERROR FLAG**. Si el error se puede corregir, se retransmite el mensaje. Si un nodo acumula demasiados errores, pasa a un estado pasivo o se desconecta para evitar interferir en la comunicación de otros nodos [1].

Los 5 tipos de errores en CAN son:

1. **Error de Bit**: Ocurre cuando un nodo transmite un bit en el bus y lee de vuelta un valor diferente del que envió.
2. **Error de Bit Stuffing**: Se produce cuando no se sigue la regla de ”bit stuffing”, que requiere insertar un bit opuesto después de 5 bits consecutivos del mismo valor lógico.
3. **Error de CRC**: Aparece cuando el valor de la secuencia CRC calculado por el receptor no coincide con el transmitido en el mensaje.
4. **Error de Formato (Form Error)**: Se detecta cuando un campo del mensaje no cumple con los valores lógicos esperados, como el bit de inicio o los delimitadores.
5. **Error de Acknowledgment (ACK Error)**: Ocurre cuando un transmisor no detecta un bit dominante en el campo de ACK enviado por los receptores para confirmar la recepción del mensaje [6].

A pesar de describir como se hace el manejo de error en el protocolo bajo ciertas condiciones de error, los mecanismos nativos no siempre garantizan la transmisión atómica. Esto significa que, en escenarios específicos, pueden ocurrir duplicados o pérdidas de mensajes, lo que genera inconsistencias en la recepción. Estos fallos en la entrega de mensajes son críticos en sistemas distribuidos que dependen de una comunicación confiable, como los sistemas embebidos en tiempo real utilizados en sectores como el automotriz o la automatización industrial.

La probabilidad de ocurrencia de estos errores está directamente influenciada por la tasa de error de bit (BER) en el canal de transmisión. En entornos agresivos, como fábricas con maquinaria pesada o equipos que generan interferencias electromagnéticas, la BER puede aumentar significativamente, lo que compromete la confiabilidad del protocolo CAN. La mayor tasa de errores en estos entornos sugiere que las aplicaciones en dichos escenarios deben tener en cuenta las condiciones ambientales para garantizar una comunicación segura y efectiva [7].

Además, aunque en condiciones normales la BER de CAN suele ser lo suficientemente baja como para cumplir con los estándares de seguridad en aplicaciones críticas, como los sistemas de control de vehículos o dispositivos médicos, los entornos industriales más agresivos pueden generar una probabilidad de error mayor. Esto exige la implementación de

mecanismos de tolerancia a fallos o redundancia en sistemas basados en CAN para asegurar un rendimiento fiable en aplicaciones donde la seguridad y la precisión son fundamentales [7].

Aunque el protocolo CAN puede ser susceptible a errores en entornos con interferencias electromagnéticas (EMI) o tasas de error de bit elevadas, también es conocido por su robustez en diversas condiciones adversas. Los mecanismos intrínsecos de detección y corrección de errores que posee, como los errores de CRC, errores de forma y de confirmación, le permiten recuperarse rápidamente de los fallos sin comprometer significativamente la comunicación [8]. En muchas situaciones, CAN detecta un error, genera una retransmisión casi instantánea y garantiza que los datos lleguen a su destino de manera correcta y ordenada.

Incluso bajo condiciones severas de interferencia electromagnética, como se mostró en pruebas con vehículos CAN [8], los sistemas logran mantener la integridad de la comunicación con mínimos retrasos. Por ejemplo, en situaciones donde se inyectaron interferencias electromagnéticas en un vehículo equipado con CAN, se registraron errores ocasionales, pero el bus pudo recuperarse rápidamente con tiempos de retransmisión en el rango de micro-segundos, garantizando que no hubiera pérdidas significativas de datos o fallos en el rendimiento del sistema.

Esta capacidad de recuperación bajo entornos difíciles hace que CAN sea un protocolo confiable, especialmente en aplicaciones críticas como el control automotriz y otros sistemas embebidos distribuidos.

V. Puertos Físicos Utilizados

Para poder transmitir datos con el protocolo CAN, realmente no se necesita un puerto especifico ya que cualquier puerto que se utilice para comunicación serial suele funcionar, como lo es el caso del mini-DIN 8 pins, DB15, DB9 , etc. Como se muestra en la figura 7 respectivamente.

Figura 7. Puertos que se pueden utilizar para CAN.

Pero, esto conlleva un error, al dificultar la forma en que se le da mantenimiento a sistemas que tengan integrados el protocolo, por lo que la organización CAN in Automation (CiA) responsable de la promoción y estandarización de las tecnologías basadas en el protocolo CAN [9]. Se fundó para coordinar el desarrollo de protocolos y perfiles de aplicación para garantizar la interoperabilidad entre dispositivos de diferentes fabricantes. Ante esta necesidad de estandarizar el protocolo crearon CANopen. CANopen proporciona un protocolo de alto nivel que gestione las complejidades de la comunicación en redes basadas en CAN. Su objetivo es facilitar la integración de dispositivos en aplicaciones industriales, automotrices y otras áreas mediante la estandarización de la configuración, gestión y transferencia de datos en redes embebidas [10]. Con este protocolo derivado de CAN, se llegó a la estandarización del puerto d-sub de 9 pines como se muestra en la figura 8 y la tabla I correspondiente:

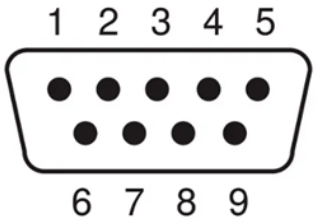

Figura 8. D-sub 9 pines.

Cuadro I
Conveniencia de pines de CiA

Pin Nº	Descripción
Pin 1	No conectado (Reservado)
Pin 2	CAN_L (línea baja del bus CAN)
Pin 3	CAN_GND (Tierra del bus CAN)
Pin 4	No conectado (Reservado)
Pin 5	CAN_SHLD (Pantalla del bus, opcional)
Pin 6	GND (Tierra del dispositivo)
Pin 7	CAN_H (línea alta del bus CAN)
Pin 8	No conectado (Reservado)
Pin 9	No conectado (Reservado)

En la tabla I, se puede observar que hay CAN HIGH y CAN LOW, los cuales no se han detallado, ya que se especificó cómo CAN permitía comunicarse con solo un cable de datos y tierra, pero en la aplicación del protocolo se utiliza una comunicación diferencial para transmitir datos entre nodos en un sistema. A diferencia de otros protocolos de comunicación como SPI o I2C, CAN emplea dos cables dedicados para transmitir señales simultáneamente: CAN High (CAN_H) y CAN Low (CAN_L) [11].

VI. Funcionamiento de CAN High y CAN Low

Los cables CAN_H y CAN_L transportan voltajes que se interpretan de forma diferencial, como se mencionó anteriormente. Dependiendo de los valores de voltaje, se determina si la señal es “Dominante.º “Recesiva”, esto se puede observar en la figura 9.

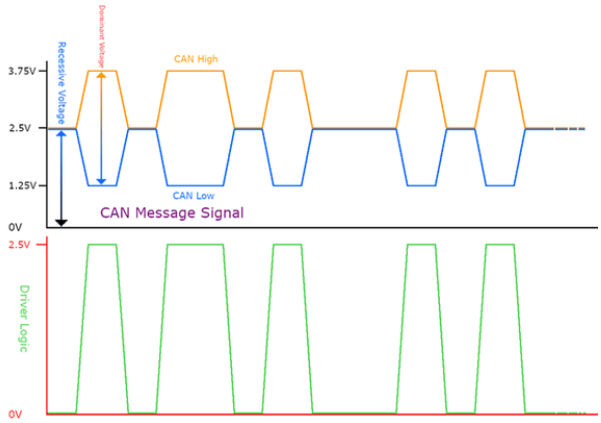

Figura 9. Señales CAN_H y CAN_L en un mensaje. [11]

- **Señal Recesiva:** Ambos cables, CAN_H y CAN_L, tienen un valor de 2.5V. Esta condición indica un valor binario de **1**.
- **Señal Dominante:** Cuando CAN_H sube a 3.75V y CAN_L baja a 1.25V, el sistema interpreta un valor binario de **0**.

Aunque la lógica parece inversa esto se debe a cómo CAN prioriza la detección de la señal dominante para garantizar la correcta sincronización y manejo de colisiones en el bus. Esta inversión en la lógica garantiza que los ”0”dominantes prevalezcan sobre los ”1recesivos en caso de conflicto de datos.

A la hora de implementar una red con sus respectivos nodos, se debe colocar una resistencia en ambos extremos de los nodos para proporcionar una correcta diferencia de voltajes entre CAN_H y CAN_L y también aseguran que el sistema tenga una impedancia coincidente, lo cual es crítico para evitar reflexiones de señal y garantizar una transmisión estable a altas frecuencias. Normalmente, esta resistencia es de 120 ohms. Esta resistencia es conocida como “terminating resistor” (Rt) [12].

VII. Topologías en CAN

Como se ha mencionado, CAN tiene la característica de ser un protocolo de comunicación serial de bus, por lo que no se diferencia tanto de algunos protocolos ya muy conocidos como I2C y SPI. Sus topologías nacen normalmente de CANopen, ya que permite la flexibilidad suficiente para poder hacer aplicaciones reales como: automatización de fábricas, automatización de edificios, automatización del hogar, sistemas de control, sistemas de monitoreo y automatización de vehículos [13].

La topología más común es conocida como “The General Architecture of CANOpen Network”, que consta de: CAN_High, CAN_Low y CAN_Ground. Lo que distingue a este protocolo es el requisito de un resistor de terminación.

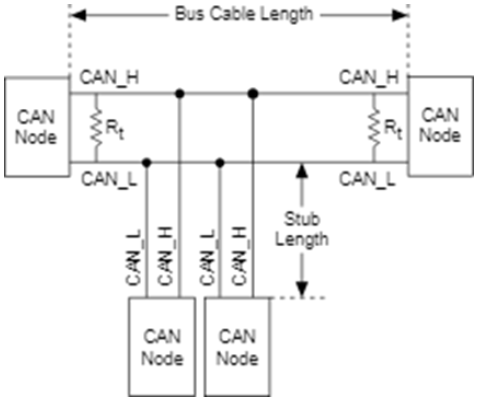

Figura 10. Topología general de CANopen [23]

La capacidad de CAN de soportar diversas topologías es una de sus principales fortalezas. Esto permite su implementación en una variedad de escenarios, desde pequeños sistemas embebidos hasta redes industriales complejas. Las topologías como el bus lineal, la estrella [14], y aquellas que utilizan repetidores permiten que el protocolo sea escalable y flexible en términos de distancias [12], topología de puente [12], etc.

VIII. Arquitecturas Basadas en CAN

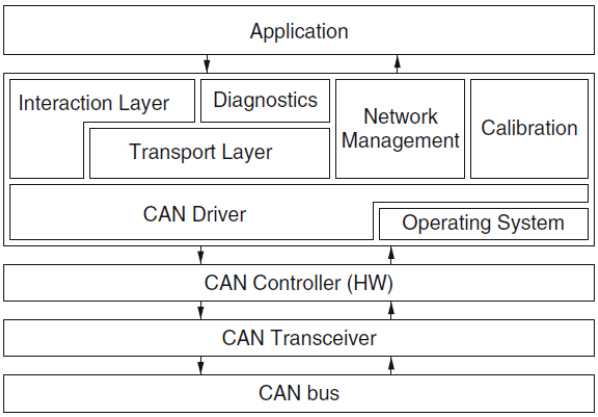

Figura 11. Arquitectura típica de un sistema basado en CAN [3]

El protocolo ha brindado la información suficiente para replicar su arquitectura y mejorar ciertas deficiencias como los que se hablaron en la sección de errores. Es muy común que CiA publique estos avances en la tecnología debido a lo ágil que es. En la figura 11, se puede observar una arquitectura base que recomienda para crear una arquitectura basada en CAN [3], la cual ha desencadenó que sus usuarios implementaran sus propias mejoras como:

- **CANcentrate:** Propone una arquitectura basada en una topología de estrella activa, donde un concentrador central aísla errores y mejora la confiabilidad en redes CAN, especialmente en aplicaciones donde la seguridad y la robustez son cruciales [14].
- **FlexCAN:** Es una arquitectura que extiende CAN para aplicaciones críticas de seguridad, añadiendo mecanismos como gestión de redundancia, canales CAN replicados, y manejo avanzado de errores. Está diseñada para

aumentar la confiabilidad y seguridad en aplicaciones embebidas como automóviles y vehículos industriales [15].

- **Flexible Time-Triggered communication on Controller Area Network (FTT-CAN):** Combina comunicaciones time-triggered y event-triggered en una única arquitectura, lo que permite gestionar tráfico con diferentes requisitos temporales, ofreciendo flexibilidad y garantizando la transmisión en tiempo real en sistemas distribuidos [16].
- **N-Server CAN (Hierarchical Scheduling):** Introduce una arquitectura de programación jerárquica, utilizando servidores N para gestionar el acceso al bus CAN de manera eficiente, permitiendo separar diferentes flujos de mensajes y mejorar la flexibilidad del sistema en aplicaciones de tiempo real [17].
- **Clock Synchronization in CAN:** No es tanto una arquitectura completa, se acerca más a una solución para implementar la sincronización de relojes en sistemas embebidos distribuidos basados en CAN, algo que el protocolo CAN estándar no incluye de forma nativa [18].
- **CAN-RT-TOP:** Es una arquitectura orientada a tareas que permite a las aplicaciones seleccionar dinámicamente las prioridades de sus mensajes, independientemente del nodo de destino. Esto mejora la flexibilidad en sistemas donde un nodo ejecuta múltiples tareas con diferentes requisitos temporales, optimizando la transmisión de mensajes según su urgencia [19]. Está diseñada para sistemas distribuidos de tiempo real, donde la comunicación de mensajes pequeños es frecuente, como el control de sensores y actuadores.

Estas son solo algunas de las recopilaciones de las modificaciones y arquitecturas que se pueden implementar en CAN, estas no se unen en una sola debido a las aplicaciones que los usuarios han tenido con el protocolo, algunos prefieren el CAN2.0A base y sin modificaciones debido a su fiabilidad y robustez. Como se mencionó anteriormente, no hay errores significativos en el protocolo para recurrir a implementar de manera inmediata alguno de estos cambios a la arquitectura.

IX. Ejemplo de comunicación

Se implemento un ejemplo de CAN utilizando dos microcontroladores con el protocolo incorporado, en este caso, fueron Arduino R4 minima [21]. Estos fueron lanzados en el año 2023, por lo que se puede resaltar que la tecnología en este microcontrolador es muy reciente. A pesar de contar con un bus de CAN, se requiere un transceptor y necesario porque, aunque estos microcontroladores puedan tener un controlador CAN integrado, este solo maneja la lógica de protocolo y comunicación a nivel de bits. Sin embargo, no tiene la capacidad de generar o interpretar las señales eléctricas diferenciales necesarias para transmitir los datos a través del bus físico utilizando los cables CAN_H y CAN_L [22].

El software fue desarrollado utilizando el entorno Arduino IDE 2.0, donde dos scripts diferentes (CANWrite.ino y CANRead.ino) fueron programados en los microcontroladores para manejar la transmisión y recepción de mensajes CAN.

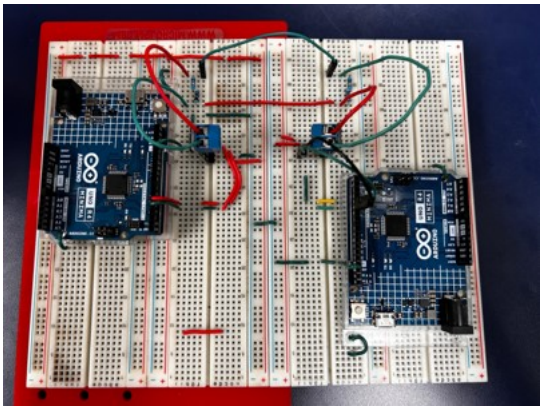

Figura 12. Montaje del sistema con Arduino UNO R4 Minima y transceptores CAN.

IX-A. *Resultados de la Transmisión Simplex y Half-Duplex*

En las imágenes 13 y 14, se muestran los resultados de las transmisiones en modos simplex y half-duplex, respectivamente. En el caso de la transmisión simplex, se observó una secuencia de mensajes CAN transmitidos exitosamente de un nodo emisor a un nodo receptor, tal como se muestra en los monitores seriales de ambas computadoras. Los mensajes contienen identificadores estándar y datos en formato hexadecimal. Por otro lado, la imagen de la transmisión half-duplex muestra la recepción y transmisión de mensajes entre dos nodos que comparten el bus, alternando el rol de emisor y receptor.

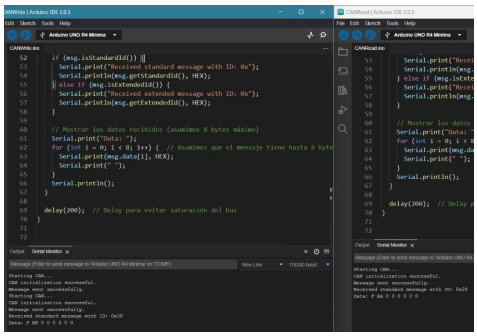

Figura 13. Transmisión en modo simplex entre nodos CAN.

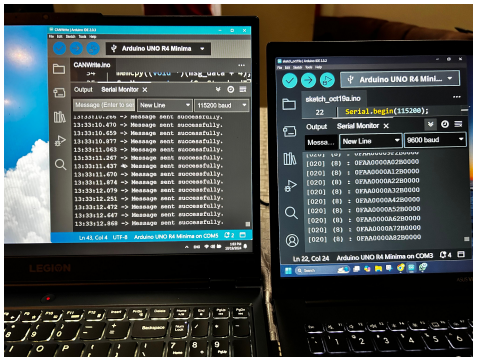

Figura 14. Transmisión en modo half-duplex en el bus CAN.

IX-B. Análisis del Data Frame

Las imágenes 15 a 18 muestran las formas de onda correspondientes a diferentes campos del *Data Frame* CAN, obtenidas mediante un osciloscopio digital. A continuación, se describe cada uno de los campos:

IX-B1. Arbitration Field: La imagen 15 muestra la forma de onda del *Arbitration Field*. Este campo es utilizado para la resolución de colisiones en el bus, permitiendo que el nodo con la menor identificación (ID) tenga prioridad en la transmisión. En este caso, se observa un identificador estándar de 11 bits.

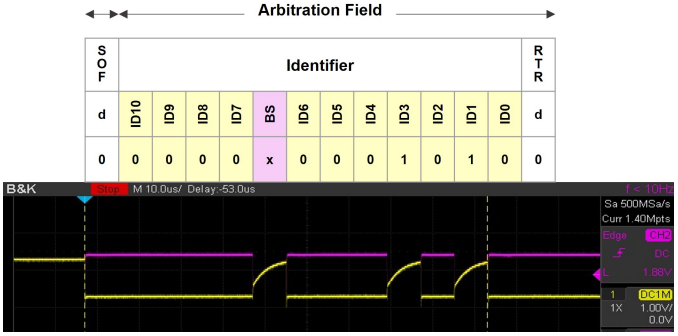

Figura 15. Forma de onda correspondiente al Arbitration Field.

IX-B2. Control Field: En la imagen 16 se puede observar la forma de onda correspondiente al *Control Field*. Este campo incluye información sobre el tipo de mensaje que se está enviando (datos o solicitud de transmisión) y el número de bytes de datos presentes en el mensaje. Se aprecia la configuración para un mensaje con un tamaño de datos de 8 bytes.

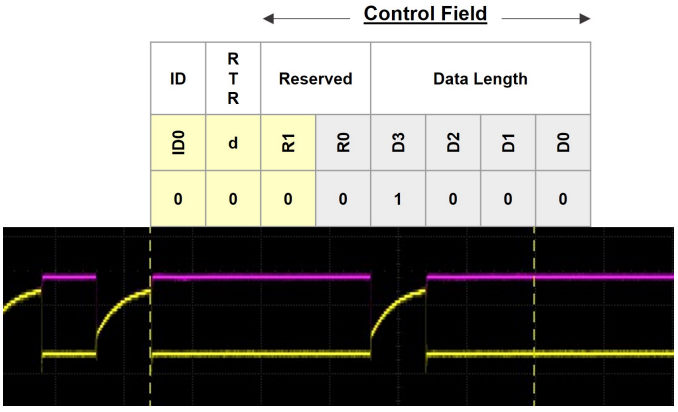

Figura 16. Forma de onda correspondiente al Control Field.

IX-B3. Data Field: La imagen 17 presenta el *Data Field*, el cual contiene los datos útiles que son transmitidos en el mensaje CAN. En este caso, se muestra una secuencia de datos en formato hexadecimal, representados en los bits de la forma de onda.

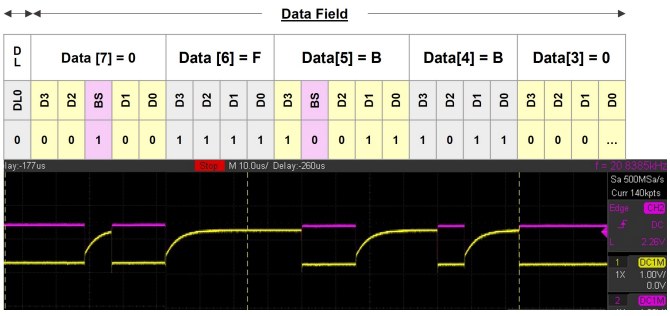

Figura 17. Forma de onda correspondiente al Data Field.

IX-B4. CRC, ACK y EOF: Finalmente, en la imagen 18, se muestra la forma de onda del campo *CRC* (Cyclic Redundancy Check), que contiene el código de verificación de redundancia cíclica calculado para detectar errores en la transmisión. También se observa el campo *ACK* (Acknowledgment), que es utilizado por el nodo receptor para indicar que el mensaje fue recibido correctamente, y el campo *EOF* (End of Frame), que marca el final del frame CAN.

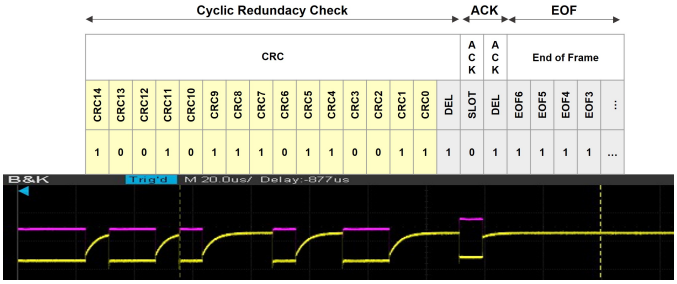

Figura 18. Forma de onda correspondiente a CRC, ACK y EOF.

Toda este ejemplo funcional de la comunicación con CAN se puede observar en el siguiente enlace:

https://youtu.be/eSdyY7zbx0Y

Además, las presentaciones utilizadas para mostrar este contenido del protocolo CAN de manera didactica y rápida se puede encontrar en este enlace a Google Drive:

https://drive.google.com/drive/folders/1-xVXl_Q7qVINi7D-kf1aKoCrQvkzIusI?usp=sharing

X. Conclusiones

El protocolo de comunicación serial Controller Area Network (CAN) ha demostrado ser una solución eficaz y confiable para la comunicación en sistemas embebidos y distribuidos, especialmente en entornos industriales y automotrices. A lo largo de los años, CAN ha evolucionado desde su concepción inicial para simplificar el cableado y los costos en vehículos, hasta convertirse en una herramienta versátil aplicada también en automatización industrial, médica y robótica. Sus características de priorización de mensajes, multimaster y detección de errores robusta han sido fundamentales para

su adopción en sistemas críticos que requieren tiempos de respuesta predecibles y alta fiabilidad.

Además de su capacidad de adaptación, el protocolo CAN soporta una variedad de topologías de red, lo que permite escalar la red según las necesidades del sistema. Su flexibilidad ha llevado a la creación de variantes como CANopen, que optimizan el uso en sistemas distribuidos y brindan soporte para aplicaciones complejas, mejorando la interoperabilidad y simplificando la integración de dispositivos. Asimismo, los avances en las diversas arquitecturas que se pueden elaborar a partir de CAN.

CAN sigue siendo una pieza clave en la industria de sistemas embebidos, gracias a su capacidad de adaptación, seguridad y fiabilidad. A pesar de la aparición de nuevas tecnologías, su evolución constante asegura que CAN continuará siendo una solución preferida en muchas aplicaciones críticas, lo que subraya su relevancia en la ingeniería moderna.

Por su parte, la demostración técnica ha demostrado ser una solución viable y eficiente para la comunicación entre dispositivos en un entorno de red industrial. A lo largo de esta demostración, hemos construido un sistema capaz de transmitir y recibir mensajes CAN, visualizar las formas de onda correspondientes a los distintos campos del *Data Frame* y validar el correcto funcionamiento del protocolo a través de las transmisiones simplex y half-duplex.

El análisis de las formas de onda capturadas en el osciloscopio permitió estudiar en detalle cómo se organiza y transmite la información en el bus CAN. Las observaciones sobre los campos de arbitraje, control, datos y CRC reflejan con claridad cómo el protocolo resuelve las colisiones de mensajes, valida la integridad de los datos y asegura una comunicación confiable mediante la retroalimentación de los nodos receptores.

REFERENCIAS

[1] R. Bosch GmbH, *CAN Specification Version 2.0*, Robert Bosch GmbH, Stuttgart, Germany, Sep. 1991.

[2] M. Numeir, “CAN Protocol: Types of CAN Frames,”*Medium*, Jun. 3, 2023. [Online]. Available: https://medium.com/@mohammednumeir13/can-protocol-types-of-can-frames-51c8444176bb

[3] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. New York: Springer, 2012. DOI: 10.1007/978-1-4614-0314-2.

[4] Kvaser, “Controller Area Network (CAN BUS) Protocol,”Sep. 24, 2024. [Online]. Available: https://kvaser.com/can-protocol-tutorial/

[5] “A short history of standardization and CAN”, *Control Engineering*, Feb. 23, 2021. [Online]. Available: https://www.controleng.com/articles/a-short-history-of-standardization-and-can/

[6] CSS Electronics, “CAN Bus Errors Explained - a Simple Intro [2022]”, CSS Electronics, May 12, 2022. [Online]. Available: https://www.csselectronics.com/pages/can-bus-errors-intro-tutorial

[7] J. Ferreira, A. Oliveira, P. Fonseca, and J. Fonseca, “An Experiment to Assess Bit Error Rate in CAN”, in *Proceedings of RTN 2004, 3rd Int. Workshop on Real-Time Networks*, University of Catania, Italy, Jun. 29, 2004.

[8] R. T. McLaughlin, “EMC Susceptibility Testing of a CAN Car”, SAE Technical Paper 932866, *Worldwide Passenger Car Conference and Exposition*, Dearborn, Michigan, Oct. 25-27, 1993.

[9] CAN in Automation (CiA), “About Us”, 2024. [Online]. Available: https://can-cia.org/about-us

[10] CAN in Automation (CiA), “CANopen”, *CAN Knowledge*, 2024. [Online]. Available: https://www.can-cia.org/can-knowledge/canopen

[11] Digi-Key Electronics, “Overview of the CAN Bus Protocol”, Digi-Key Forum, 2024. [Online]. Available: https://forum.digikey.com/t/overview-of-the-can-bus-protocol/21170/1

[12] “CANopen Network Topology,”*Instrumentation Tools*, 2024. [Online]. Available: https://instrumentationtools.com/canopen-network-topology/

[13] “I-7565-H1”, A2S.pl, *Converters - CAN*, 2024. [Online]. Available: http://www.a2s.pl/en/i-7565-h1-p-5437.html

[14] M. Barranco, G. Rodriguez-Navas, L. Almeida, and J. Proenza, “CAN-centrate: An Active Star Topology for CAN Networks”, in *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*, Aveiro, Portugal, Sep. 22, 2004.

[15] J. R. Pimentel and J. A. Fonseca, “FlexCAN: A Flexible Architecture for Highly Dependable Embedded Applications”, in *Proceedings of RTN 2004, 3rd Int. Workshop on Real-Time Networks*, University of Catania, Italy, Jun. 29, 2004.

[16] L. Almeida, P. Pedreiras, and J. A. G. Fonseca, “The FTT-CAN Protocol: Why and How”, *IEEE Transactions on Industrial Electronics*, vol. 49, no. 6, pp. 1189-1201, Dec. 2002.

[17] T. Nolte, M. Nolin, and H. Hansson, “Hierarchical Scheduling of CAN using Server-Based Techniques”, in *Proceedings of RTN 2004, 3rd Int. Workshop on Real-Time Networks*, University of Catania, Italy, Jun. 29, 2004.

[18] G. Rodriguez-Navas and J. Proenza, “Clock Synchronization in CAN Distributed Embedded Systems, ”, in *Proceedings of RTN 2004, 3rd Int. Workshop on Real-Time Networks*, University of Catania, Italy, Jun. 29, 2004.

[19] J. López Campos, J. J. Gutiérrez, and M. González Harbour, “CAN-RT-TOP: Real-Time Task-Oriented Protocol over CAN for Analyzable Distributed Applications”, in *Proceedings of RTN 2004, 3rd Int. Workshop on Real-Time Networks*, University of Catania, Italy, Jun. 29, 2004.

[20] F. Hartwich and A. Bassemir, “The Configuration of the CAN Bit Timing”, in *6th International CAN Conference*, Turin, Italy, Nov. 2-4, 1999.

[21] Arduino, “Arduino UNO R4 Minima Product Reference Manual”, 2024. [Online]. Available: https://docs.arduino.cc/resources/datasheets/ABX00080-datasheet.pdf

[22] S. Broyles, “A System Evaluation of CAN Transceivers”, *Texas Instruments Application Report*, Rev. A, Mar. 2002, revised Jul. 2018.

[23] National Instruments, ÇAN Bus Topology and Termination,”2024. [Online]. Available: https://www.ni.com/docs/en-US/bundle/sbrio-9627-feature/page/can-bus-topology-termination.html

[24] J. A. Cook and J. S. Freudenberg, Çontroller Area Network (CAN),”*EECS 461, Fall 2008*, University of Michigan, 2008.

[25] A. Daga, Çontroller Area Network (CAN) and Extended CAN-FD, “Wayne State University, 2024. [Online]. Available:https://engweb.eng.wayne.edu/~ad5781/ECECourses/ECE5620/Notes/CAN_Extended_CAN-FD.pdf