SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

INSTITUTO TECNOLÓGICO DE MÉRIDA

TESIS

"DESARROLLO DE UN SISTEMA EDUCATIVO PARA MONITOREO E INTERACCIÓN CON SEÑALES EN EL BUS CAN"

PARA OPTAR AL GRADO DE:

MAESTRO EN INGENIERÍA

PRESENTA:

ING. RAMÓN ARIEL VELA XOOL

ASESOR:

DR. CARLOS ALBERTO LUJÁN RAMÍREZ

MÉRIDA, YUCATÁN, MÉXICO.

15 DE JUNIO DE 2015

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Mérida

"2015, Año del Generalísimo José María Morelos y Pavón"

DEPENDENCIA: DIV. DE EST. DE POSG. E INV.
No. DE OFICIO: X-092/2015

ASUNTO: AUTORIZACIÓN DE IMPRESIÓN

MERIDA, YUCATÁN A 02 DE JUNIO DE 2015

C. RAMÓN ARIEL VELA XOOL
PASANTE DE LA MAESTRÍA EN INGENIERÍA
PRESENTE

De acuerdo al fallo emitido por su asesor el **Dr. Carlos Alberto Luján Ramírez** y la comisión revisora integrada por el Dr. José Ramón Atoche Enseñat, M.C. José Agustín Hernández Benítez, y el M.C. Jorge Carlos Canto Esquivel, considerando que cubre los requisitos establecidos en el Reglamento de Titulación de los Institutos Tecnológicos le autorizamos la impresión de su trabajo profesional con la **TESIS:**

"DESARROLLO DE UN SISTEMA EDUCATIVO PARA MONITOREO E INTERACCIÓN CON SEÑALES EN EL BUS CAN"

A T E N T A M E N T E
IN HOC SIGNO VINCES

M.C. MIRIAM H. SÁNCHEZ MONROY
JEFA DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

S. E. P.
INSTITUTO TECNOLOGICO
DE MERIDA
DIVISION DE ESTUDIOS DE
POSGRADO E INVESTIGACION

C.p. Archivo
C.p. Titulación
MHSM/fja

DEDICATORIA

A Dios

Creador omnipotente, amoroso y bondadoso dedico esta tesis, por todo lo maravilloso que nos brinda, por haberme dado salud para lograr mis objetivos y otorgarme fuerzas para superar obstáculos y dificultades en el transcurso de toda mi vida.

A mi familia

A mis padres Benito y María, a mis hermanos Alfredo y Elsy quienes me apoyan incondicionalmente en todo momento.

Ramón Ariel Vela Xool

AGRADECIMIENTOS

A mi asesor, Dr. Carlos Alberto Luján Ramírez, por la orientación, ayuda y confianza que me brindó durante el desarrollo de esta tesis.

Al Dr. Jesús Sandoval Gío y al Dr. José Ramón Atoche Enseñat por su apoyo y gestiones desde la coordinación de la Maestría.

A mis sinodales quienes aprobaron mi tesis.

A todos los profesores de la Maestría por su dedicación y paciencia.

Al Lic. Christian Maldonado Lizarraga por su apoyo en las gestiones relacionadas con la beca y al proceso de titulación.

A mis compañeros de Maestría por su amistad y ayuda.

A mis amigos y a todas aquellas personas que directa o indirectamente contribuyeron en esta tesis.

A la CNBES por la beca de posgrado otorgada para facilitar la culminación de la maestría.

Por haber contado siempre con el apoyo de mi familia, amigos y maestros, a todos ellos muchas gracias y que Dios los bendiga.

*Ramón Ariel Vela Xoo*l

RESUMEN

El protocolo CAN (Controller Area Network) es ampliamente utilizado en la industria automotriz y por su relativo bajo costo, confiabilidad, fuerte inmunidad a la interferencia electromagnética, alta velocidad de transferencia y reducción del cableado necesario, también es una alternativa atractiva en la industria de la automatización. Considerando que el BUS CAN ofrece un protocolo de comunicación eficiente entre sensores, actuadores, controladores, y otros nodos en aplicaciones en tiempo real, se propone implementar una red CAN para la adquisición de datos utilizando la tarjeta electrónica Arduino basada en el microcontrolador Atmega328. Esta tarjeta es de fácil utilización ya que cuenta con la librería disponible necesaria para implementar el bus CAN y el software utilizado para su programación es de uso libre. El microcontrolador Atmega 328 no dispone de un controlador CAN integrado, por lo que es necesario colocarle una tarjeta de expansión (shield) que lo tenga.

En este trabajo se desarrolla un sistema para la enseñanza del CAN, con hardware de fácil adquisición e implementación. Con las tarjetas electrónicas construidas se puede implementar una red para control y adquisición de datos. Cada tarjeta (Nodo) tiene capacidad de hasta de 6 sensores analógicos y se pueden controlar hasta 7 entradas/salidas digitales. Para la programación del microcontrolador se utiliza el software para Arduino y la librería necesaria para la comunicación CAN, ambos disponibles de forma gratuita en internet. La interfaz gráfica para monitorear los datos se desarrolla en el entorno de programación de Visual Basic 2010. El código fuente del proyecto estará a disposición del usuario para que lo pueda adaptar a diferentes propósitos o aplicaciones con fines didácticos. Con la intención de facilitar la conexión de más nodos al bus del sistema, se construyeron unos conectores tipo “T”. El correcto funcionamiento de los nodos y de la red CAN implementada se verificaron utilizando el CAN bus analyzer, dispositivo comercializado por Microchip.

ABSTRACT

The CAN (Controller Area Network) protocol is widely used in the automotive industry and its relatively low cost, reliability, strong immunity to electromagnetic interference, high transfer rate and reduction of wiring required, it is also an attractive alternative in the automation industry. Considering that the BUS CAN provides a protocol for efficient communication among sensors, actuators, controllers, and other nodes in real-time applications it is proposed to implement a CAN network for data acquisition using the electronic card Arduino based in the microcontroller Atmega 328.This card is easy to use since it has the available library, needed to implement the bus CAN and the software for programming is free to use. The Atmega 328 microcontroller does not have an integrated CAN controller, so it is necessary to place an expansion card (shield) that does.

This work develops a system for the education of the CAN, with hardware of easy acquisition and implementation. With the built electronic cards, you can implement a network for control and acquisition of data. Each card (node) has a capacity of 6 analog sensors and it may control seven digital inputs/outputs. The programming of the microcontroller uses the software for Arduino and the necessary library for the CAN communication, both are for free in internet. The graphic interface to monitor the data, develops in the enviroment of programming of VISUAL BASIC 2010. The project's source code, will be available to any user who can adapt it to different purposes or applications oriented to didactic ends. Some "T" connectors were built with the purpose to facilitate the connection of more nodes to the bus of the system . The proper operation of the nodes and of the network CAN implemented, was verified using the CAN bus analyzer commercialized by Microchip.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. CAPÍTULO 1. DESCRIPCIÓN DEL PROYECTO

1.1. Introducción

CAN (*Controller Area Network*) es un protocolo de comunicación multidifusión de bajo costo y de implementación sencilla. El CAN fue desarrollado a medianos de la década de los ochenta por la compañía alemana Bosch GmbH para proporcionar una solución económica a las comunicaciones de bus en el automóvil. Actualmente es ampliamente usado también en la automatización de fábricas y plantas de control, en robótica, en dispositivos médicos así como también en sistemas para aeronáutica (Di Natale, Zeng, & Giusto, 2012).

El factor determinante en la industria automotriz para el desarrollo del protocolo de comunicación CAN y otros protocolos de bus fue la necesidad de investigar y desarrollar nuevas soluciones para el problema resultante del incremento de la complejidad en las comunicaciones entre los diferentes dispositivos del automóvil. La condición era reducir la cantidad y complejidad del cableado utilizado. La longitud del cableado era hasta de cientos de metros por lo que es comprensible que fueran muy difíciles de instalar y darle mantenimiento (Lawrenz, 2013).

CAN es un bus digital de difusión diseñado para operar a velocidades de 20 kbits/s a 1 Mbit/s, estandarizado como ISO 11898 para aplicaciones de alta velocidad (500 kbits/s). La velocidad de transmisión depende de la longitud del bus y de la velocidad del transceptor. CAN es una atractiva solución para sistemas de control embebido por su bajo costo, administración sencilla del protocolo y su capacidad de detección de error y retransmisión incluida en el circuito integrado. Los circuitos controladores de CAN soportan la comunicación CAN estándar, así como sensores y actuadores que son manufacturados para comunicarse por medio del protocolo CAN. Las redes CAN están exitosamente reemplazando las conexiones punto a punto en muchas aplicaciones (Di Natale, Zeng, & Giusto, 2012).

Esta tesis, que consiste en desarrollar un sistema educativo para el monitoreo e interacción con el bus CAN, está organizado en cuatro capítulos. El primer capítulo presenta el planteamiento del problema, los objetivos, la justificación, la metodología y el estado del

arte. En el capítulo dos se aborda el marco teórico que presenta los conceptos básicos que fundamentan el desarrollo del proyecto. En el capítulo 3 se describe el diseño y la implementación del sistema, el cual incluye hardware y software. Se explican las características principales del microcontrolador Atmega, del microcontrolador CAN y del tansceptor CAN. También se describe el software desarrollado durante la realización de este proyecto. En el capítulo 4 se exponen los resultados, las conclusiones y trabajos futuros. Al final también se incluye un anexo que proporciona un ejemplo de práctica laboratorio.

1.2. Planteamiento del problema.

En la actualidad, el bus CAN (Área de Red Controlada) tiene grandes aplicaciones, principalmente en la industria automotriz y la industria manufacturera. En este tipo de bus se insertan señales de control y monitoreo de datos provenientes de un gran número de sensores, actuadores y dispositivos de interfaz hombre máquina. El bus CAN permite una comunicación rápida y efectiva entre los diferentes dispositivos de un sistema de automatización.

En la actualidad se carecen de plataformas didácticas flexibles y de bajo costo, en la cual se pueda enseñar a los estudiantes de ingeniería o maestría, el funcionamiento del protocolo CAN de tal forma, que gasten menos tiempo en averiguar en ¿cómo es? o ¿cómo se hace?, sino que, inviertan más tiempo en aplicaciones que puedan tener impacto en la sociedad. Es por ello que se propone el desarrollo de una interfaz para el monitoreo e interacción con señales en el bus CAN, donde el estudiante pueda acceder al aprendizaje de una forma efectiva y sobre todo orientado al desarrollo de aplicaciones utilizando el protocolo CAN.

1.3. Objetivos.

1.3.1. Objetivo general.

Desarrollar un sistema didáctico para el monitoreo e interacción con las señales electrónicas que se encuentran en un bus CAN.

1.3.2. Objetivos específicos.

A. Diseñar los circuitos electrónicos de monitoreo e interfaz.
B. Construir tarjetas electrónicas de uso educativo para la elaboración de kits.
C. Desarrollar un software que permita la visualización y la interacción con la señales del bus.

1.4. Justificación.

El bus CAN es una red duradera y económica que permite a varios dispositivos comunicarse entre sí. Un beneficio es que permite a las unidades de control electrónico (ECUs) tener una sola interfaz CAN en los automóviles, en lugar de diferentes entradas analógicas y digitales para cada dispositivo del sistema (National Instruments, 2011). Las industrias automotrices han invertido mucho dinero en la implementación del bus en sus automóviles, y las industrias manufactureras en busca de modernizar sus instalaciones y mejorar aún más los tiempos de fabricación y calidad, invierten en cambiar a una automatización que utiliza el Bus CAN. Por lo antes mencionado, es evidente de que existe una gran potencial de mercado que se puede explotar o al menos un nicho de oportunidades donde los profesionistas en el ramo puedan ejercer su profesión con calidad.

Existen comercialmente tarjetas de interfaz del Bus CAN con otros protocolos de comunicación digital como I2C, SPI, RS232, RS485, Ethernet, entre otras. Sin embargo, los productos o equipos enfocados al quehacer educativo son escasos y costosos. El software comercial PCAN-Explorer 5, de Peak-System es un buen ejemplo de un programa de monitoreo e interacción con el CAN, sin embargo, el precio de una licencia para una sola PC

cuesta alrededor de €450.00 y la interfaz CAN para USB que permite la conexión a las redes CAN tiene un costo de €195.00 la según la página de internet del producto (Peak System, 2015), teniendo el kit un costo total de €645.00. Por otro lado, Microchip tiene una variedad de tarjetas y herramientas (CAN Development Tools) enfocadas al análisis del CAN, como por ejemplo, el CAN Bus Analyzer que es un monitor CAN que puede ser usado para desarrollar y depurar redes CAN. También, ofrece una línea de productos para aplicaciones embebidas usando el protocolo CAN. Incluye microcontroladores de 8, 16 y 32 bits y controladores de señales digitales (DSCs, *Digital Signal Controllers*) con CAN integrado (Microchip, 2015). Aunque de excelente calidad y orientadas al aprendizaje, encontramos que por sí mismas tienen la desventaja ser de muy limitada aplicación ya que solo incluyen ejemplos de comunicación básicos y el software ofrecido es bastante escueto y con nula flexibilidad para el crecimiento en su funcionalidad ya que no se encuentra disponible su código fuente. Por tales motivos se plantea el desarrollo de un sistema educativo para el monitoreo e interacción con el bus CAN de costo accesible y que se disponga del código fuente con la finalidad de que se pueda modificar y adaptar a diferentes proyectos futuros.

1.5. Metodología.

Para el desarrollo del proyecto es necesario conocer los fundamentos teóricos del protocolo de comunicación CAN por lo que primero se realiza una investigación documental consultando libros, artículos científicos publicados, manuales y hojas de datos de fabricantes, y páginas en internet que tratan del tema. Se definen los objetivos que se deben cumplir y se realiza una investigación del estado del arte para conocer la situación actual, cuál es la tendencia y de que se dispone en el mercado.

Se utilizará el método experimental mediante la elaboración y prueba de prototipos, tanto para el hardware como para el software. Experimentar con los prototipos permite detectar errores y deficiencias con la finalidad de mejorar el desempeño del sistema a desarrollar. Para comprobar el funcionamiento del hardware se utilizarán instrumentos de medición como el osciloscopio para observar las señales eléctricas y para determinar que los

datos enviados o recibidos son correctos se empleará un Can bus analyzer que fabrica y comercializa la empresa Microchip. Luego se procede a desarrollar la interfaz gráfica para la visualización de los datos presentes en el bus CAN.

1.5.1. Metas.

En cuanto a hardware:

A.1 Estudiar el funcionamiento y características del Bus CAN.
A.2 Comunicar dos nodos CAN utilizando tarjetas de expansión (shields) para arduino.
A.3 Observar las señales de los nodos anteriores con un analizador de CAN bus comercial.
A.4 Diseñar un circuito electrónico que permita la interacción y monitoreo del bus CAN desde una laptop.

B.1 Diseñar los circuitos de los nodos CAN.
B.2 Construir las tarjetas electrónicas de los nodos CAN.
B.3 Implementar una red CAN usando los nodos antes implementados.

En cuanto a software:

C.1. Determinar el software de programación a utilizar.
C.2. Desarrollar una interfaz que permita monitorear y enviar datos a través del bus CAN.
C.3 Realizar las pruebas pertinentes de hardware y software.

1.6. Estado del arte.

A inicios de la década de los ochentas, un grupo de ingenieros de la compañía alemana Robert Bosch GmbH estudiaron la posibilidad de aplicar sistemas de bus seriales dentro de los automóviles con la finalidad de satisfacer las demandas de la Sociedad de Ingenieros Automotrices (*SAE, Society of Automotive Engineers*) (Chamú Morales, 2005).

En febrero de 1986 Robert Bosch GmbH presentó de forma oficial el sistema de bus serial en el congreso de la SAE celebrado en la ciudad de Detroit, E.U.A. llamado CAN (Controller Area Network), fue el nacimiento de uno de los protocolos de red más exitosos. Actualmente casi cada nuevo automóvil de pasajeros manufacturado en Europa es equipado con al menos una red CAN. (CAN in Automation, 2015). La tabla 1 muestra la evolución del protocolo CAN.

Tabla 1.1 Evolución del protocolo CAN.

AÑO	SUCESOS
1983	Bosch inicia un proyecto interno para desarrollar una red de a bordo para automóviles
1986	Introducción oficial del protocolo CAN
1987	Se fabrican los primeros chips controladores CAN de Intel y Philips Semiconductors
1991	Bosch publica la especificación de CAN 2.0
1991	CAN Kingdom es introducido por Kvaser
1992	Se funda oficialmente CAN in Automation (CiA), grupo integrado por usuarios y fabricantes internacionales
1992	CiA publica el protocolo CAN capa de aplicación (CAL, CAN Aplication Layer).
1992	Los primeros coches de Mercedes-Benz utilizan red CAN
1993	Es publicado el estándar ISO 11898
1994	Primera conferencia CAN internacional (international CAN Conference iCC), organizado por CiA
1994	El protocolo DeviceNet es introduccido por Allen-Bradley
1995	Modificación ISO 11898 (formato de trama extendida) publicada
1995	Protocolo CANopen publicado por CiA
2000	Desarrollo del protocolo de comunicación accionado por tiempo basado en CAN (TTCAN, *Time Triggered CAN*)

El bus CAN es utilizado en el automóvil como bus de comunicaciones de lo que se denomina sistemas distribuidos embebidos, es decir para unir unidades de control electrónicas que están “embebidas” o “empotradas” en los sistemas que controlan. En un automóvil pueden existir varias redes CAN, aunque lo normal es que sean dos, una de alta velocidad de

transmisión y otra de baja velocidad. La red CAN de alta velocidad enlaza unidades de control en tiempo real como las de inyección de combustible, frenado ABS o unidad de encendido en motores de encendido provocado. La red CAN de baja velocidad enlaza dispositivos electrónicos tales como los elevalunas, el conjunto de luces y el sistema de climatización. Debido a su uso en la industria automotriz, un amplio número de controladores CAN están disponibles y muchos microcontroladores tienen controladores CAN integrados (Dressler & Fischer, 2007).

Aunque inicialmente el bus CAN surgió de la necesidad de comunicar las distintas unidades electrónicas existentes en el automóvil, tales como frenado, ABS, inyección de combustible entre otros, su utilización se ha ido expandiendo a otros campos de aplicación distinta de la automoción. Siendo usado en muchas aplicaciones de control industrial embebidos. Desde principios de la década de los noventas se han desarrollado buses de campo basados en CAN, los más populares en automatización industrial por ejemplo son DeviceNet y CANopen. DeviceNet, promovido por la ODVA (*Open DeviceNet Association*), asociación global fundada en 1995, es principalmente utilizado en la automatización de fábricas. Es un enlace de comunicación de bajo costo para conectar dispositivos industriales tales como interruptores de límite, sensores fotoeléctricos, sensores de procesos, electroválvulas, variadores de frecuencia para motores, a una red y eliminar el excesivo y costoso cableado. El otro protocolo de comunicaciones de alto nivel basado en el bus CAN y promovido por la organización “CAN in Automation” (CiA) es el CANopen, la red para sistemas de control embebido (Dressler & Fischer, 2007).

El bus CAN está muy bien adaptado para interconectar dispositivos de entrada-salida, sensores y actuadores inteligentes. Además esta comunicación la realiza con un buen comportamiento en tiempo real y de un modo muy seguro dada la sofisticada detección y confinamiento de errores que utiliza. Se puede afirmar por tanto que entre los buses de tiempo real aplicables a sistemas distribuidos en los que la relación prestaciones/coste es determinante, puede decirse que hasta la fecha ninguno ha alcanzado la combinación de

aceptación, disponibilidad de dispositivos de diversos fabricantes y robustez que ofrece el bus CAN. Todas estas consideraciones han hecho que el abanico de campos de aplicación donde se ha introducido el bus CAN sea cada vez más amplio. Desde 1994, se han estandarizado varios protocolos de alto nivel a partir de CAN, como CANopen y DeviceNet , y su uso se ha extendido a otras industrias (National Instruments, 2011). Por último en el año 2001 Bosch ha especificado el protocolo TTCAN (*Time Trigered CAN*) donde se intenta acercar esta nueva versión a las características necesarias que ha de tener un bus de comunicaciones en sistemas de tiempo real estricto, todas las acciones deben terminar dentro del plazo especificado (Di Natale, Zeng, & Giusto, 2012; CAN in Automation, 2015).

La industria aeronáutica está interesada en el protocolo CAN, ya que la capa de aplicación CANaerospace se utiliza para el control de los sistemas de las aeronaves. CANaerospace fue lanzado por primera vez en 1998 y desde entonces ha sido implementado en numerosas aplicaciones aeronáuticas y también es el principal estándar de interfaz para cabinas de simulador de vuelo (CANaerospace, 2015).

En 1997, 24 millones de interfaces CAN fueron producidos en un año, 2 años después esta cifra se triplicó (Lawrenz, 2013). Este éxito es debido a que dichos elementos presentan importantes ventajas como son un bajo coste, fiabilidad y capacidad de funcionar en entornos agresivos. Estas características, unidas a las de robustez, facilidad de uso y alto grado de capacidad de tiempo real, hicieron que el bus CAN pronto se utilizara en aplicaciones industriales, tanto en sistemas distribuidos embebidos (robots, máquinas textiles, máquinas empaquetadoras), como en sistemas abiertos para automatización, siendo soporte de varios buses de campo como CANOpen, DeviceNet presentado por Allen-Bradley , SDS (*Smart Distributed System*) de Honeywell y CAN-Kingdom introducido por Kvaser (Kvaser, 2015).

Otros campos de aplicación del bus CAN son la automatización naval (por ejemplo para el control distribuido de salas de máquinas desatendidas), como subred en edificios

inteligentes, controlando sistemas tales como los ascensores, sistemas de aire acondicionado, etc., en el mundo del ferrocarril (control de puertas en vagones) y en el mundo de la aviación para unir sensores de estado de vuelo y sistemas de navegación. Los fabricantes de equipo médico utilizan CAN como una red embebida en los dispositivos médicos. De hecho, algunos hospitales utilizan CAN para manejar cuartos de operación completos. Los hospitales controlan componentes operativos del cuarto como luces, máquinas de rayos X y camas de pacientes con sistemas basados en CAN. CANopen también es utilizado en aplicaciones no industriales como en equipo de laboratorio, cámaras deportivas, telescopios, puertas automáticas e incluso, máquinas de café (National Instruments, 2011). También hay que considerar la existencia de muchos interfaces hardware y paquetes software para facilitar el análisis del bus CAN de empresas como Vector, Peak-System National Instruments y Microchip se mencionan como ejemplos. Vector, que desarrolló el primer software para análisis del CAN bus, ofrece la herramienta de software CANalyzer 8.5 para análisis y simulación. Se usa para determinar si hay comunicación y que datos hay en el bus. También permite enviar y almacenar datos (Vector, 2015). El hardware requerido (interfaz) para conectar la computadora a la red CAN es comercializado por la misma empresa.

En la figura 1.1 se muestra el hardware y el software para monitoreo de datos en el CAN bus comercializado por la compañía Peak-System. PCAN explorer es una herramienta universal para el monitoreo del tráfico de datos en las redes CAN. La integración de VBScript (*Visual Basic Script Edition*) a esta aplicación permite la creación de macros para automatizar tareas complejas. Tiene integrado un data logger que permite visualizar, analizar y almacenar el tráfico de datos en el bus. El hardware necesario para la conexión con la red bus CAN es el adaptador PCAN-USB. Esta misma empresa ofrece al mercado diversas herramientas de software y hardware para el bus CAN (Peak System, 2015).

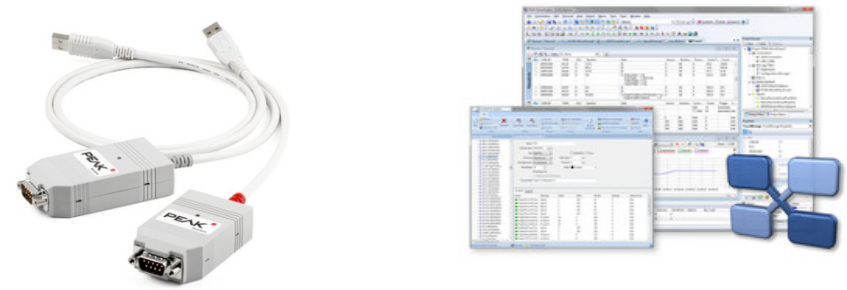


Figura 1.1 Interface CAN-USB y software PCAN-Explorer de Peak-System.

En la figura 1.2 se puede ver el CAN bus analyzer tool y una de las pantallas del software que lo acompaña. El CAN bus analyzer tool es una herramienta proporcionada por Microchip, sencilla de usar y de bajo costo para monitorear el bus CAN. El kit viene con el hardware y software requerido para conectar una red red CAN a una computadora. La interfaz gráfica permite observar el tráfico de datos en el bus (Microchip, 2015).

Figura 1.2. CAN bus Analyzer y una de las pantallas del software.

National Instruments proporciona una variedad de herramientas de hardware y software para el desarrollo de aplicaciones CAN. NI ofrece interfaces CAN para diferentes plataformas incluyendo USB, PCI, PXI, PCMCIA, y NI CompactRIO. Para PCI, PXI, y PCMCIA, se encuentran disponibles las capas físicas de alta velocidad, de baja velocidad/tolerante a fallas y de un solo cable. Para PCI y PXI, National Instruments

proporciona la primera interfaz industrial CAN seleccionable por software que contiene un transceptor de alta velocidad, de baja velocidad/tolerante a fallas y de un solo cable en la tarjeta, por cada puerto y en el mismo dispositivo. Esto quiere decir que simplemente es seleccionar por software el modo que desea utilizar. National Instruments incluye el software controlador NI-CAN con todas las interfaces CAN de National Instruments y lo ofrece como un software de descarga gratuita en ni.com. El software controlador NI-CAN proporciona funciones de alto nivel y fáciles de usar para ayudarle a desarrollar aplicaciones CAN

Los fabricantes de osciloscopios comienzan a incluir modos de disparo apropiados al bus CAN. Por ejemplo, los osciloscopios de la serie PicoScope 5000, para PC que comercializa la empresa Pico Technology, ver figura 1.3, posee entre sus múltiples funciones la decodificación en serie, ya que puede capturar miles de estructuras de datos ininterrumpidos. Los protocolos que se incluyen actualmente son I²C, SPI, RS232/ UART, CAN, LIN y FlexRay (Pico Technology, 2015). Es posible decodificar los datos del bus CAN con la función de decodificación en serie que se incluye en este osciloscopio para computadora. A diferencia de un analizador convencional de buses, PicoScope permite ver la forma de onda eléctrica de alta resolución al mismo tiempo que los datos. Los datos se incorporan a la vista de osciloscopio. El software para este equipo es de descarga libre en la página del fabricante.

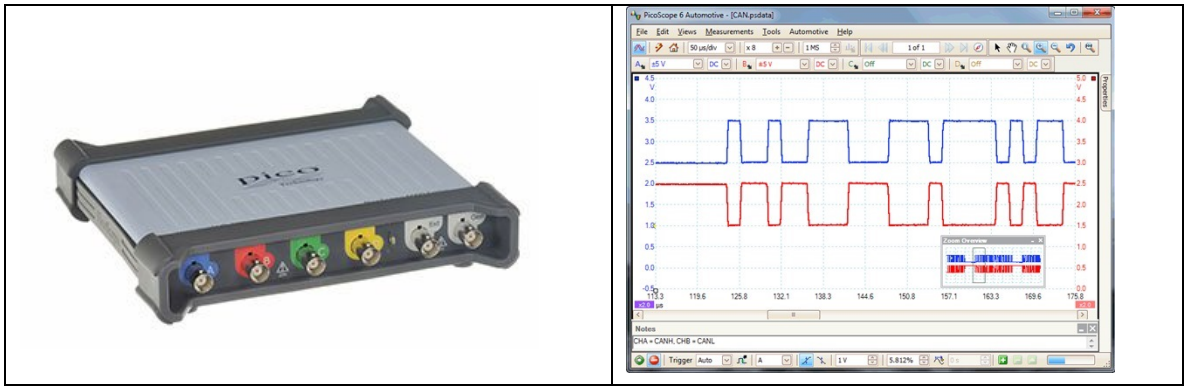


Figura 1.3. Osciloscopio PicoScope de la serie 5000 con decodificador CAN.

CAPÍTULO 2. MARCO TEÓRICO.

2.1. Red multiplexada CAN bus.

El CAN es un bus serial para el envío y recepción de mensajes pequeños para el control en tiempo real. El mensaje a enviar consiste de entre 1 a 8 bytes y ha sido diseñado para operar hasta velocidades de1 Mbit/sec (Bril, Lukkien, & Davis, 2006). Es un protocolo de comunicación altamente difundido por la asociación CiA (CAN in Automation), organización internacional que apoya y desarrolla al protocolo CAN y a protocolos de capa superior basados en CAN, la cual está constituida por reconocidos desarrolladores internacionales. El protocolo CAN implementa una comunicación basada en el mensaje, no en la dirección, teniendo independencia de direcciones de origen y destino (Vidal & Zúñiga, 2005). Esto significa que los mensajes no son transmitidos de un nodo a otro nodo basado en sus direcciones. Todos los nodos en el sistema reciben cada mensaje transmitido en el bus. Cada nodo del sistema decidirá si el mensaje recibido debe ser inmediatamente descartado o almacenado para su procesamiento (Pazul & Microchip, 1999).

CAN es un bus de transmisión dónde un número de procesadores son conectados al bus vía una interfaz. Una fuente de datos es transmitida como un mensaje, consistiendo de entre 1 a 8 bytes. Una fuente de datos puede ser transmitida periódicamente, esporádicamente o por demanda. Por ejemplo, la velocidad de un motor o de una rueda puede ser codificada como un mensaje de un byte y transmitida cada 100 milisegundos. A la fuente de datos se le asigna un identificador único, representado como un número de 11 bits (dando 2032 identificadores). El identificador sirve para dos propósitos: filtrar la información al receptor indicado y para asignar la prioridad del mensaje (Tindell, Burns, & Wellings, 1995). En el protocolo CAN un mensaje no tiene destinatario.

Otra característica útil del protocolo CAN es la habilidad de un nodo para solicitar información de otros nodos. Esto es llamado como RTR (*Remote Transmit Request*), en lugar de estar esperando información, este nodo solicita específicamente que le sea enviado datos por algún nodo. Un beneficio adicional de este protocolo basado en mensajes es que nodos adicionales pueden ser conectados a la red sin necesidad de reprogramar los otros nodos para

reconocer al nodo añadido. Este nuevo nodo comenzará a recibir mensajes de la red y, basado del identificador del mensaje, decidirá si procesa o descarta la información recibida (Pazul & Microchip, 1999).

El protocolo de red CAN ha sido definido para proporcionar una comunicación determinística en sistemas distribuidos complejos con las siguientes características y capacidades (Di Natale, Zeng, & Giusto, 2012) (Chamú Morales, 2005) (Mesa Montoya, 2008) :

- Asignación de prioridad por mensajes, es decir, es un protocolo basado en mensajes, no en direcciones.
- Un mensaje es diferenciado por un campo llamado identificador, que no indica el destino del mensaje, pero si describe el contenido del mismo.
- Recepción por multidifusión (multicast) con sincronización orientada a bit.
- Sistema robusto en cuanto a consistencia de datos.
- Es un sistema multimaestro. Cuando el bus está libre, cualquier nodo puede comenzar la transmisión de un mensaje, y el mensaje con mayor prioridad gana la arbitración del bus.
- Todos los nodos CAN son capaces de transmitir y recibir datos y varios pueden acceder al bus de datos simultáneamente.
- Un nodo emisor envía el mensaje a todos los nodos de la red, cada nodo, según el identificador del mensaje, lo filtra y decide si debe procesarlo inmediatamente o descartarlo. Como consecuencia el sistema se convierte en multicast en el cual un mensaje puede estar dirigido a varios nodos al mismo tiempo.
- Detección y señalización de errores.
- Retransmisión automática de tramas erróneas tan pronto como el bus esté libre.
- Detección de fallas permanentes en los nodos, y desconexión automática para aislar los nodos defectuosos.

2.2. CAN y el modelo OSI.

Muchos protocolos de red son descritos usando las siete capas del modelo OSI (*Open System Interconnection, Interconexión de sistemas abiertos*). El objetivo de OSI es permitir la comunicación entre sistemas distintos sin que sea necesario cambiar la lógica del hardware o el software subyacente. El modelo OSI, no es un protocolo, es un modelo para comprender y diseñar una arquitectura de red flexible, robusta e inter-operable. El modelo fue diseñado con la intención de proveer una estructura general para cualquier tipo de sistema complejo de comunicación, debido a esto no todos los sistemas pueden usar las siete capas del modelo (Defaz Andrango, 2007).

CAN está basado en las dos primeras capas del modelo OSI. Como se observa en la figura 2.1 (Richards & Microchip, 2002). Estas dos primeras capas, son la capa física y la capa de enlace de datos. ISO (*International Standards Organization, Organización Internacional de Normalización*) ha definido un estándar el cual incorpora las especificaciones CAN así como la capa física. El estándar ISO-11898, fue originalmente creado para las comunicaciones CAN de alta velocidad en vehículos. ISO-11898 especifica la capa física para asegurar la compatibilidad entre los transceptores CAN. Un controlador CAN típicamente implementa de forma completa la especificación CAN en hardware, como se muestra en la figura 2.1. El estándar ISO 11898-2 es la norma de capa física más importante en aplicaciones automotrices y CiA lo recomienda para aplicaciones industriales.

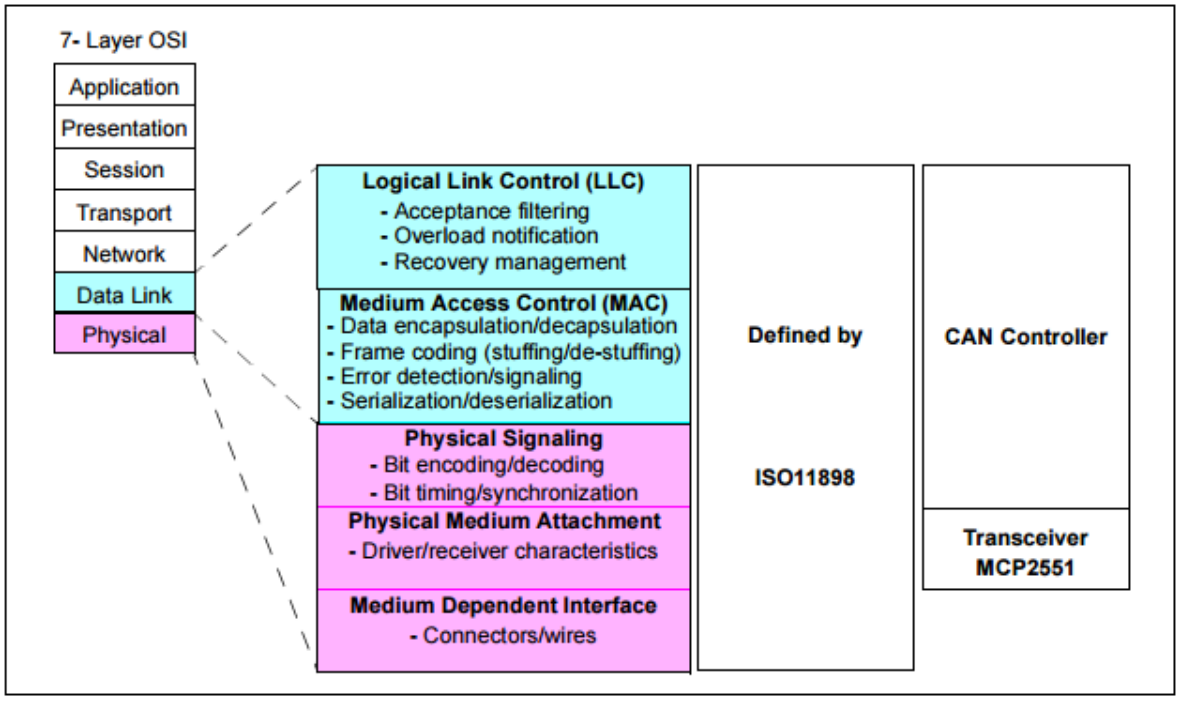


Figura 2.1 CAN y el modelo OSI.

Este modelo OSI de 7 capas propuesto en 1984 se mostró ineficiente para su utilización en redes industriales con requerimientos de baja latencia, debido a la sobrecarga que este modelo impone en cada capa. Para solventar este inconveniente, la mayoría de redes industriales utilizaron únicamente 3 de estos niveles, el nivel físico, el nivel de enlace y el nivel de aplicación, por lo que a continuación se repasan las particularidades de estos niveles en el entorno industrial (Sacón Chango & Villalva Taipe, 2013).

El nivel físico define el medio físico que se utiliza en la transmisión y las características físicas del mismo, como niveles de voltaje, sistema de codificación, entre otros. Estas características determinan la topología, la velocidad de transmisión y el número máximo de nodos en una red. El número máximo de nodos en una red no está limitado por el protocolo, depende de la capacidad de los circuitos integrados empleados.

El nivel de enlace define los formatos de trama, mecanismos de protección ante errores en la transmisión (CRC o código de redundancia cíclica). Controla el flujo de información entre los nodos de la red. Es decir, se encarga de la transmisión de los bits en tramas (*frames*) de información, se ocupa de que los mensajes lleguen a su destino sin errores, controla las secuencias de transmisión, los acuses de recibido y si en determinado caso no se recibió bien un mensaje se encarga de retransmitirlo (Pacheco Hernández, 2011). En la mayoría de las redes, incluyendo los buses de campo, en este nivel se ubica el subnivel de acceso al medio (MAC, *Medium Access Control*). La capacidad de satisfacer los requerimientos de tiempo real de las aplicaciones industriales dependerá en gran medida de si el mecanismo de acceso al medio tiene un comportamiento determinista o no determinista.

El nivel de aplicación define los interfaces entre el usuario y el sistema, y habitualmente incluye el nivel de usuario, denominado así dado que habitualmente es la forma en que el usuario ve el bus de campo, aislándole del manejo de los niveles inferiores. Los estándares proponen a este nivel objetos específicos para diferentes dominios de aplicación como la robótica, control numérico, control de procesos, entre otros (Yunta, 2010). Existen diferentes estándares que definen la capa de aplicación; algunos son muy específicos y están relacionados con sus campos de aplicación. Los más populares que ofrece el mercado son: CAN Open, Smart Distributed System, Device Net, OSEK, J1939 o CAN Kingdom. A continuación se detallan brevemente los rasgos principales de algunos de ellos (Mesa Montoya, 2008):

CAN Open: Fue originalmente diseñado para la industria de sistemas de control, pero las redes CAN Open también son usadas para aplicaciones de campo como transporte público, equipos médicos, etc. Las especificaciones cubren el nivel de aplicación, el perfil de la comunicación, el armazón de los aparatos programables de los nodos y recomendaciones de cables y conectores. Es uno de los HLP más utilizados en las aplicaciones basadas en el bus CAN.

Device Net: El protocolo de comunicación Device Net es abierto y aceptado como un estándar de industria en todo el mundo. Fue diseñado para comunicar dispositivos de control inteligentes así como sensores de campo, estaciones de pulsadores, arrancadores, interfaces de operador simples, y variadores de control de velocidad. Es un sistema de CAN barato. Un aspecto curioso es que este protocolo también puede integrarse en CAN Kingdom. Es rápido y llega a soportar 3 velocidades: 125Kbps, 250Kbps y 500Kbps. Además soporta 64 nodos activos. Device Net básicamente distingue entre mensajes de procesos de mayor prioridad (I/O Mensajes) y mensajes de menor prioridad (Mensajes Explícitos). Además utiliza una técnica orientada a objetos, en el que la información está estructurada en diferentes objetos.

SDS (Smart Distributed System): Es un sistema de bus para sensores y actuadores inteligentes. Es un modo eficiente de conectar pequeños aparatos a un controlador máster. Todos los nodos SDS tienen asignados un número del rango (de 0 a 125) llamado dirección del aparato y debe ser único para todos los nodos del sistema. Este número es usado como base para seleccionar un conjunto de identificadores de CAN que pueden ser usados por este módulo.

CAN Kingdom: Este protocolo de alto nivel está diseñado principalmente para realizar sistemas de control, móviles hidráulicos, etc. Algunos denominadores comunes de estos sistemas son el alto desarrollo en tiempo real, alta demanda de seguridad. El protocolo CAN Kingdom soporta entre otras cosas, cambios dinámicos de identificadores, identificadores estándar y extendidos, un reloj global y un hardware que limita el número de módulos del sistema. En este protocolo, el nodo por defecto en la red tiene toda la información necesaria para inicializarse en el sistema. Las especificaciones de los sistemas serán los métodos de acceso al bus, la gestión de la red, las listas de mensajes y los formatos de los datos. El diseñador de sistemas es responsable del software implementado en el nodo y él puede decidir en qué condiciones los nodos serán aceptados por el sistema. En el fondo el diseñador de CAN Kingdom tiene la máxima libertad para crear su propio sistema (Mesa Montoya, 2008).

2.3. Señal transmitida por una red CAN.

La capa física en CAN es responsable de la transferencia de bits entre los distintos nodos que componen la red. Define aspectos como, niveles de señal, codificación, sincronización y tiempos en que los bits se transfieren al bus. La transmisión de datos se realiza por impulsos eléctricos en forma de señal cuadrada a través de un bus de datos. Los buses utilizados en las redes CAN son bialámbricos, es decir, constan de dos cables entrelazados, llamados CAN High y CAN Low. Por cada cable del bus circula una señal cuadrada que varía entre dos valores, y siempre se cumplirá que ambas señales cuadradas son simétricas. De esta forma, la diferencia de tensión entre las dos líneas del bus solo puede tomar dos valores, representando cada uno de ellos un bit. El bit dominante o bit 0 representa la mayor diferencia de tensión entre señales, es nombrado dominante debido a que hace desaparecer los “1” lógicos de la red. El bit recesivo o bit 1 representa la menor diferencia de tensión entre señales. Al transmitirse la información en forma de diferencia de tensión entre los dos cables, cualquier interferencia externa a la red afectaría de manera a las dos señales, manteniéndose el valor de la diferencia de tensión intacta y, por lo tanto, el mensaje. Al mismo tiempo al ser simétricas las señales de los dos cables, se anulan los campos magnéticos creados por los cambios de tensión (Llanos López, 2011) .

Los valores de tensión suelen oscilar entre 1.5 y 2.5 volts en el cable CAN-L y entre 2.5 y 3.5 para el CAN-H, como se observa en la figura 2.2. Los bits recesivo y dominante quedarían de la siguiente manera (Kaschel & Pinto, 2002) (Richards & Microchip, 2002):

- Bit dominante: la tensión diferencial (CAN-H - CAN-L) es del orden de 2.0 V con CAN-H = 3.5 V y CAN-L = 1.5 V.

- Bit recesivo: la tensión diferencial (CAN-H - CAN-L) es del orden de 0 V con CAN-H = CAN-L = 2.5 V.

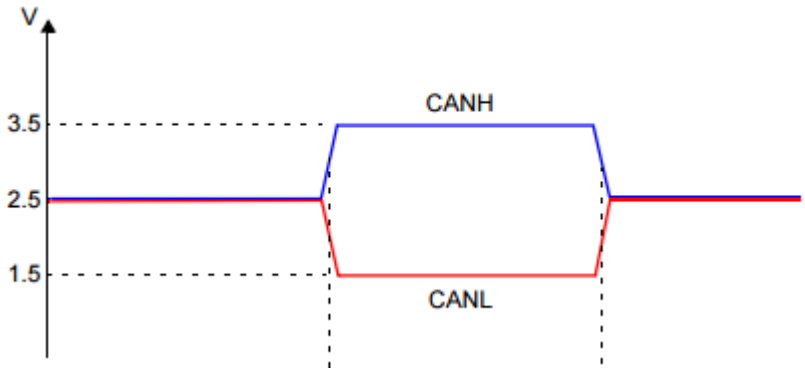
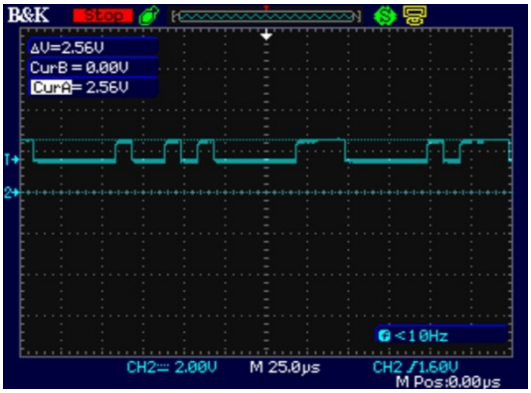
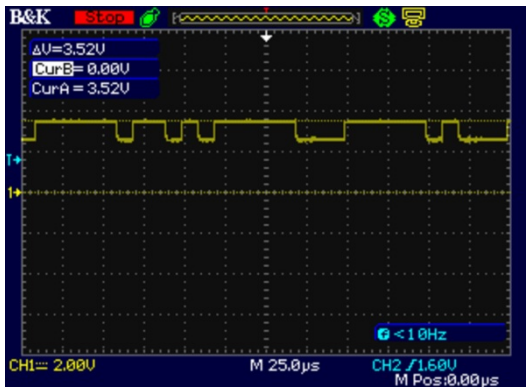


Figura 2.2 Niveles nominales ISO 11898 del bus CAN

En las figuras 2.3a y 2.3 b se muestran las señales medidas en las terminales de salida CAN-L y CAN-H con respecto a masa de una tarjeta CAN bus Shield de Sparkfun. Se observa que el voltaje de CAN-L es de 2.56 V y el de CAN-H es de 3.52 V. La figura 2.4a muestra que ambas señales son simétricas. Y la figura 2.4b indica el voltaje diferencial entre la señal CAN-H y CAN-L. El voltaje diferencial medido es de 2.08 V. El bus se encuentra transmitiendo a 100 kbps.

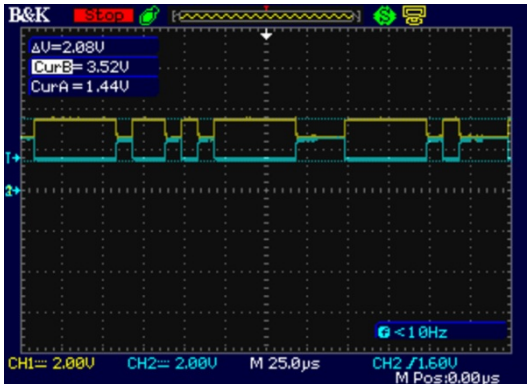


a) CAN-L

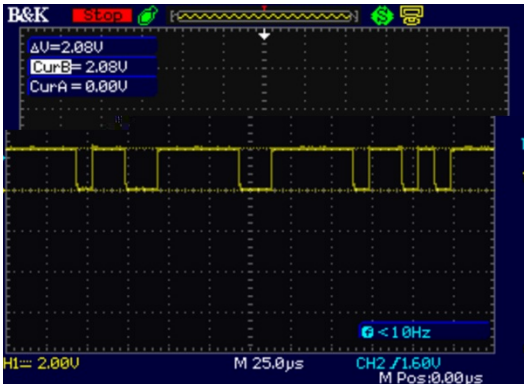


b) CAN-H.

Figura 2.3 Señales CAN-L y CAN-H



c) Señales simétricas de CAN-L y CAN H



d) Señal diferencial en el bus

Figura 2.4 Señal CAN simétrica y diferencial.

2.4. Velocidad de transmisión de datos.

La velocidad de transmisión de datos se mide en bits por segundo (bps) y nos indica la cantidad de bits que son enviados en un segundo, o lo que es lo mismo, la cantidad de información por segundo (Llanos López, 2011). En la tabla 2, consultada en (Kaschel & Pinto, 2002) y en (Di Natale, Zeng, & Giusto, 2012), muestra las longitudes máximas que puede tener el bus dependiendo de la velocidad de transmisión.

Tabla 2.1 Velocidades de transmisión típicas

Velocidad	Tiempo de bit	Longitud máxima
1 Mbps	1 µS	30 m
800 Kbps	1.25 µS	50 m
500 Kbps	2 µS	100 m
250 Kbps	4 µS	250 m
125 Kbps	8 µS	500 m
50 Kbps	20 µS	1000 m
20 Kbps	50 µS	2500 m
10 Kbps	100 µS	5,000 m

2.5. Protocolo de comunicación CAN bus.

CAN fue desarrollado, inicialmente para aplicaciones en los automóviles y por lo tanto la plataforma del protocolo es resultado de las necesidades existentes en el área de la automoción. La Organización Internacional para la Estandarización (ISO, International Organization for Standarization) define dos tipos de redes CAN: una red de alta velocidad (hasta 1 Mbps), bajo el estándar ISO 11898-2, destinada para controlar el motor e interconectar la unidades de control electrónico (ECU); y una red de baja velocidad tolerante a fallos (menor o igual a 125 Kbps), bajo el estándar ISO 11519-2/ISO 11898-3, dedicada a la comunicación de los dispositivos electrónicos internos de un automóvil como son control de puertas, techo corredizo, luces y asientos. CAN es un protocolo de comunicaciones serie que soporta control distribuido en tiempo real con un alto nivel de seguridad y multiplexación.

La comunicación entre nodos CAN se realiza, como en cualquier red de comunicación, siguiendo un protocolo. El protocolo es el “lenguaje” utilizado para transmitir el mensaje y debe ser conocido por el emisor y el receptor. Los mensajes o tramas de datos son series de bits en forma de ceros y unos que se agrupan en campos. Para poder leer estos mensajes hay que conocer los campos que los forman, que son los siguientes y se representan en la figura 2.5:

- Campo de inicio.
- Campo de estado o campo de identificación.
- Campo de control.
- Campo de datos.
- Campo de aseguramiento.
- Campo de confirmación.
- Campo de fin de la trama.
- Separador de tramas.

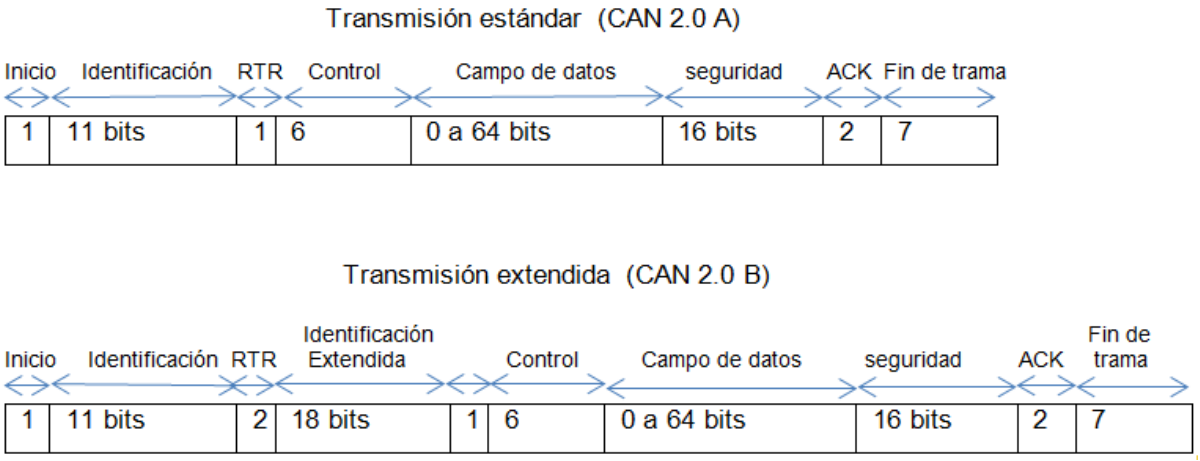


Figura 2.5 Formatos de Transmisión.

Campo de inicio (*Start of frame, SOF*): El campo de inicio de la trama está compuesto de un único bit dominante. Se trata de un bit de sintonización dominante que indica el inicio de la transmisión del mensaje. El flanco descendente de este bit es utilizado por los nodos receptores para sincronizarse entre sí.

Campo de identificación o arbitraje: El protocolo CAN requiere que todos los mensajes transmitidos tengan un identificador. Este identificador determina la prioridad del mensaje. Cuanto más bajo sea el valor del identificador más alta es la prioridad, y por lo tanto determina el orden en que van a ser introducidos los mensajes en la línea. Los bits de identificador se transmiten en orden de más significativo a menos significativo. El campo de arbitraje (identificación) consiste de 12 bits, está formado 11 bits de identificación y 1 de RTR (Remote Transmit Request) ó 32 bits, 29 bits de identificación, 1 bit para definir que el mensaje es una trama de datos extendido, 1 bit sin usar y 1 bit RTR, esto debido a que el protocolo CAN soporta dos formatos, el CAN estándar (versión 2.0 A) y el CAN extendido (versión 2.0 B).

Campo de control: Formado por 6 bits que informan de la cantidad de información que contendrá el campo de datos que viene a continuación. De esta forma el receptor podrá así

saber si ha recibido el mensaje completo. El primer bit indica si la trama es estándar, con un campo de estado de 11 bits (CAN 2.0 A), o extendida con un campo de estado de 29 bits (CAN 2.0 B). Un bit dominante (0) indica una trama estándar y un bit recesivo (1) una trama extendida. El segundo bit es una reserva para futuras ampliaciones del mensaje, y los cuatro bits restantes indican la longitud en bytes del mensaje. Si estos cuatro bits son 0000 significa que no es una trama de datos.

Campo de datos: Consta de un máximo de 64 bits (8 bytes) que contienen la información. Es decir, el mensaje puede contener de cero a 8 bytes de datos. Por ejemplo, si la información contiene cuatro bytes se envían únicamente 4 bytes. Esta longitud de datos es suficiente para los requerimientos de comunicación de datos en vehículos y niveles de entrada/salida en sistemas embebidos y automatización. En estos campos de aplicación, la transmisión de tramas de datos más largas no es tan importante como el soporte de altas velocidades de transmisión para mensajes cortos. La transmisión de tramas de datos más largas por medio de mensajes CAN es requerida solo en casos excepcionales, como por ejemplo en la configuración de parámetros del nodo, o carga de un programa. En este caso, protocolos de capas superiores basados en CAN, tales como CANopen o DeviceNet proveen los servicios adecuados para una transmisión fragmentada, limitando el tamaño del mensaje a pocos bytes, por otro lado, esta fragmentación garantiza el tiempo de latencia más corto posible al bus por parte de los mensajes de más alta prioridad. El tamaño más corto posible del mensaje es particularmente importante cuando la transmisión tiene lugar en un ambiente hostil porque la probabilidad de ocurrencia de daño que pueda sufrir una trama se incrementa proporcionalmente a su tamaño. De esta forma, el tamaño corto de los mensajes del protocolo CAN permite transmisiones de datos aún en un ambiente electromagnético ruidoso.

Campo de aseguramiento (*Cyclic Redundancy Check, CRC*): Formado por 16 bits para detectar errores en la transmisión. Este campo es el resultado de una serie de cálculos realizados a partir de los campos anteriores. El nodo receptor calcula el campo de aseguramiento y comprueba que coincida con el campo de aseguramiento enviado por el nodo

emisor. Para ello se utilizan los 15 primeros bits, mientras el último siempre es un bit recesivo que delimita el campo CRC. Si detecta un error envía una trama de error compuesta por la señal de error y un limitador, el cual está formado siempre por 8 bits recesivos.

Campo de confirmación o reconocimiento (*ACK*): Formado por dos bits y en él los receptores indican al emisor si les ha llegado el mensaje completo o solicitan que lo envíe de nuevo. Estos bits son siempre enviados como recesivos, pero las receptoras que tras calcular el campo de aseguramiento concluyen que han recibido bien el mensaje cambian el primero de estos bits por uno dominante.

Campo de fin de la trama (End Of Frame, EOF): Cierra la trama, consiste en siete bits recesivos sucesivos.

Separador de tramas (Inter Frame Spacing, IFS): También se le conoce con el nombre de campo de intermisión o ínter-trama. El espacio entre tramas separa una trama de la siguiente trama de datos o interrogación remota y consta de 3 bits recesivos. Estos bits van a continuación del campo de fin de la trama. Después de éstos tres bits que separan las tramas sigue un tiempo de bus en reposo. El tiempo de bus en reposo no es fijo, sino que varía según la situación del bus.

2.6 Detección y señalización de errores.

En el protocolo CAN se ha realizado un importante sistema de manejo de errores. Este sistema permite detectar errores en los mensajes que se transmiten para así en el caso que sea necesario se retransmitan los mensajes erróneos. Si un controlador del bus detecta un error, enviará un flag de error para avisar del mismo y así destruir el tráfico del bus. Los otros nodos posteriormente, detectarán un error debido a la violación de la regla del bit stuffing (relleno) en el flag de error y enviarán otros flags de error. El protocolo CAN define cinco tipos de detección de errores, dos de estos modos son al nivel de bit y el resto al nivel de trama:

Error de bit: Cuando un nodo está transmitiendo, éste monitoriza el nivel del bus y si el bit que lee en el bus no coincide con el que ha transmitido, es que se ha producido un error, así que se señala un “Error de bit”. En el siguiente tiempo de bit el nodo transmisor envía una trama de error, entonces, la trama fallida será reenviada después del espacio de intertramas.

Error de Stuff: Como ya se pudo ver, este error detecta si dentro del área codificada por el método del bit stuffing, existen seis bits consecutivos del mismo nivel. El nodo que detecte esto, enviará una trama de error justo en el tiempo de bit siguiente al de detectar el sexto bit del mismo nivel. La trama fallida será retransmitida después del espacio de intermisión.

CRC: Tanto este método para detectar errores como los dos siguientes (ACK y Forma) se realizan mediante el chequeo de la misma trama de datos a través de los campos correspondientes. El CRC es un campo de las tramas que contiene un código de redundancia cíclica el cual comprueba si hubo errores en la recepción del mensaje. Con el CRC podemos detectar errores aleatorios en hasta 5 bits o una secuencia seguida de 15 bits corruptos. Si el CRC calcula en el nodo receptor no coincide con el CRC enviado (el que contiene la trama), entonces el receptor descarta el mensaje recibido y envía una trama de error, pidiendo una retransmisión de la trama.

Error ACK: Otro campo que se encuentra en las tramas. En esta ocasión se trata de dos bits que indican si el mensaje fue recibido satisfactoriamente. El nodo transmisor manda el ACK en recesivo esperando a que el nodo receptor lo sobrescriba en dominante, de lo contrario se le considera una trama corrupta y lo retransmitirá. Así, si el transmisor detecta un ACK positivo, es decir, un bit dominante durante el campo ACK, sabrá que al menos un nodo ha recibido correctamente el mensaje.

Error de forma: Algunas partes de los mensajes en CAN tienen formas fijas. Estas zonas son: el delimitador de CRC, el delimitador de ACK, EOF y el espacio de intermisión

(IFS). Los bits de estas zonas deben ser recesivos. Si algún controlador de CAN detecta que alguno de estos bits es dominante, genera una Trama de Error porque se ha producido un "Error de forma", esta trama se enviará en el tiempo de bit siguiente al bit erróneo. Por tanto, como se puede ver la detección de un error por parte de un nodo se hace público al resto de nodos mediante la transmisión de tramas de error. Entonces la transmisión de ese mensaje fallido es abortada y será retransmitida tan pronto como el arbitraje de la red se lo permita al nodo correspondiente. Por otra banda, cada nodo contiene dos contadores de error: el contador de Transmisiones de Error (TEC) y el contador de Recepciones de Error (REC). Hay varias reglas que gobiernan el modo en que estos contadores deben incrementarse y/o decrementarse. En esencia, si un nodo transmite una Trama de Error su contador de Transmisiones de Error se incrementa en 8 y los nodos que reciben una Trama de Error incrementan en 1 el valor de su contador de Recepción de Error. Si un nodo transmite una trama correctamente, el TEC decrementa y si un nodo recibe una trama correctamente, su REC decrementa. Inicialmente los nodos están en estado de Error Activo. Cuando uno de sus dos contadores llega a superar el valor 127, el nodo pasa a estar en estado de Error Pasivo (Mesa Montoya, 2008).

2.6. Arbitraje del bus

El método de acceso al medio utilizado es el de Acceso Múltiple por Detección de Portadora, con Detección de Colisiones y Arbitraje por Prioridad de Mensaje (CSMA/CD+AMP, Carrier Sense Multiple Access with Collision Detection and Arbitration Message Priority). De acuerdo con este método, los nodos en la red que necesitan transmitir información deben esperar a que el bus esté libre (detección de portadora); cuando se cumple esta condición, dichos nodos transmiten un bit de inicio (acceso múltiple). Cada nodo lee el bus bit a bit durante la transmisión de la trama y comparan el valor transmitido con el valor recibido; mientras los valores sean idénticos, el nodo continúa con la transmisión; si se detecta una diferencia en los valores de los bits, se lleva a cabo el mecanismo de arbitraje (Mesa Montoya, 2008).

Cuando varios nodos empiezan simultáneamente una transmisión, el conflicto en el bus es resuelto por un proceso de arbitraje no destructivo basado en contención sobre el campo de arbitraje de la trama CAN. El campo de arbitraje está compuesto por el identificador de trama y por el bit de solicitud remota de transmisión (RTR, Remote Transmission Request) el cual es usado para diferenciar entre una trama de datos y una trama de solicitud de datos. En el formato básico de la trama, el identificador de trama tiene 11 bits, en el formato extendido tiene 29 bits. El bit más significativo del identificador es transmitido primero (el del extremo izquierdo).

El arbitraje no destructivo está basado en la posibilidad de dos niveles físicos en el bus, uno dominante y uno recesivo. Estos niveles pueden representados fácilmente, por ejemplo, por un circuito trasmisor en colector abierto. Para conseguir un 1 lógico en el bus es necesario que todos los nodos transmitan un 1, mientras que para tener un 0 lógico es suficiente que un solo nodo transmita un 0. Por tanto un nivel 0 es llamado “dominante”, y un nivel 1 “recesivo”.

El bus estará en nivel recesivo mientras se encuentre desocupado. Un nodo señala el inicio de la transmisión de una trama transmitiendo un bit dominante SOF (Start Of Frame, inicio de trama). Durante la fase de arbitraje cada nodo monitorea el nivel del bus y lo compara con el nivel transmitido. Cada nodo que ha transmitido un bit recesivo y ha monitoreado un bit dominante detiene su transmisión inmediatamente y pasa a ser receptor de la trama transmitida por otro nodo. La figura 2.6 ilustra la secuencia de la fase de arbitraje.

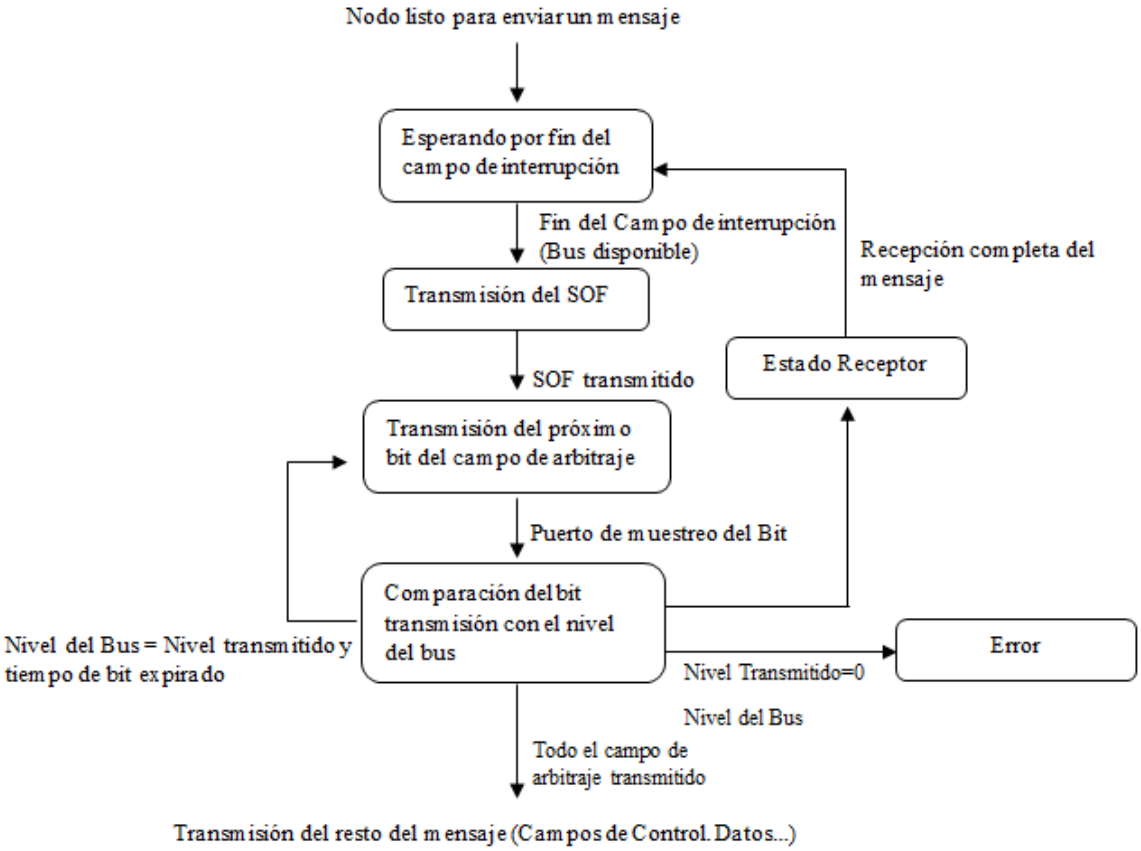


Figura 2.6 Secuencia de la fase de arbitraje.

La figura 2.7 tomada de (IXXAT, 2004) muestra la secuencia de un proceso de arbitraje entre tres nodos. Los nodos 1, 2 y 3 inician simultáneamente el proceso de arbitraje. En el punto 2, el nodo 2 detecta que en lugar de bit recesivo el bus tiene un bit dominante y por lo tanto pierde su acceso al bus. Lo mismo ocurre con el nodo 1 en el punto 3. Al final de la fase de arbitraje solo el nodo 3 continúa transmitiendo su mensaje.

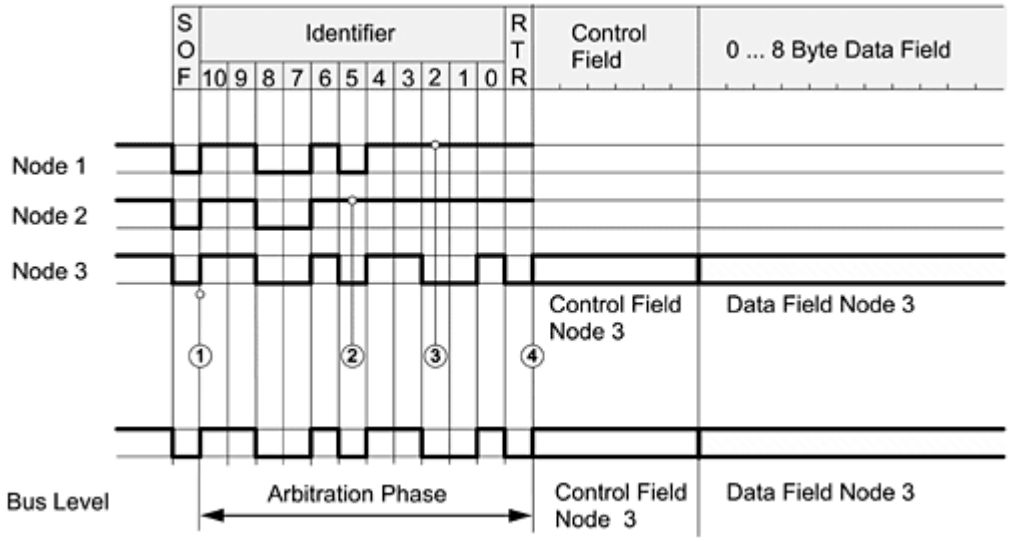


Figura 2.7 Principio del proceso de arbitraje.

2.7. Componentes físicos de una red CAN.

Una red CAN está formada por un bus datos, un número variable de unidades de control electrónico, sensores y actuadores, y resistencias de terminación. Las unidades de control, a su vez, están compuestas de un transceptor y un controlador, encargados de convertir la señal eléctrica del bus en una señal digital y viceversa, y de gestionar la información. Los componentes físicos de una red CAN son, por lo tanto, los siguientes (Llanos López, 2011):

- Unidades de control electrónico conteniendo un controlador y un transceptor CAN.
- Buses de datos.
- Resistores de terminación.

El controlador y el transceptor CAN están alojados en las unidades de control. El controlador CAN recibe del microprocesador, en la unidad de control, los datos que han de ser transmitidos. Los acondiciona y los pasa al transceptor CAN. Asimismo recibe los datos procedentes del transceptor, los acondiciona y los pasa al microprocesador. El transceptor

CAN es un transmisor y receptor. Transforma los datos del controlador CAN en señales eléctricas adecuadas y transmite éstas sobre los cables del CAN bus. También recibe los datos y los transforma para el microcontrolador (Mesa Montoya, 2008).

Para la transmisión de la señal eléctrica el protocolo CAN requiere un cable de dos hilos entrelazados (bus de datos) denominados CAN High y CAN Low cuyas señales eléctricas son leídas como la diferencia de tensión. Este bus es terminado en ambos extremos por resistores de 120 ohms. El trenzado de los cables se realiza para crear un efecto de apantallamiento y así anular los campos magnéticos inducidos. Los resistores cierran el circuito eléctrico y evitan perturbaciones indeseadas en los datos transmitidos debido a fenómenos de reflexión; impiden que el mensaje rebote al llegar del bus. Si múltiples dispositivos electrónicos (nodos) son conectados a lo largo del bus, sólo los dispositivos en los extremos del cable necesitan resistores de terminación como se ilustra en la figura 2.8. A los resistores de terminación también se les conoce como elementos finales (Llanos López, 2011) (Di Natale, Zeng, & Giusto, 2012) (Richards & Microchip, 2002).

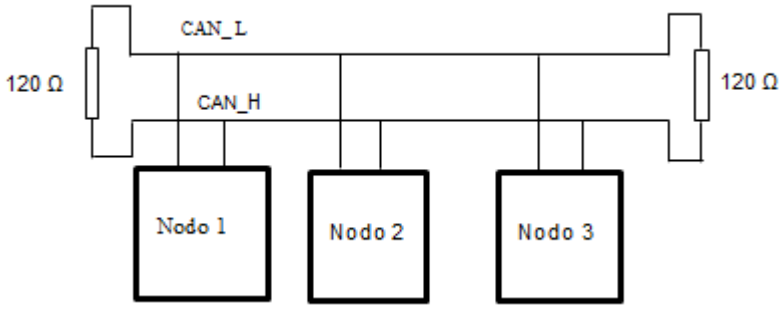


Figura 2.8 Resistencias de terminación en los extremos del bus de una red CAN.

CAPÍTULO 3. DISEÑO E IMPLEMENTACIÓN DEL SISTEMA BUS CAN.

3.1. Hardware.

En este capítulo se presenta el desarrollo del sistema educativo para el monitoreo e interacción con señales en el bus CAN. Generalmente el desarrollo de un sistema embebido involucra dos componentes esenciales, que trabajan en conjunto, el hardware y el software. Por esta razón este capítulo cual se divide en dos secciones, la primera tratará sobre el hardware empleado y su implementación y la segunda explica el software desarrollado.

3.1.1. Microcontrolador Atmega 328P.

El microcontrolador seleccionado para este proyecto, debido a su facilidad de programación y disponibilidad de librería para implementar el CAN, es el Atmega 328P-PU de la compañía Atmel. La plataforma del sistema Arduino gira alrededor de la familia de microcontroladores Atmel. Arduino es una plataforma electrónica y de programación en arquitectura abierta. Arduino se ha posicionado de manera vertiginosa como una herramienta tecnológica en la industria, universidades y centros de investigación, debido a la filosofía de arquitectura abierta, que lo hace un sistema adecuado para realizar automatización de procesos físicos a la medida. Puede ser usado para monitorear una gran cantidad de sensores analógicos y digitales, así como para controlar servomotores de corriente directa y alterna, a pasos, además de llevar a cabo aplicaciones con robots (Reyes Cortés & Cid Monjaraz, 2015).

Un microcontrolador es un chip programable, parecido a un microprocesador como el usado en los ordenadores personales, pero de mucha menor potencia. A cambio incorpora en un único chip todos los elementos necesarios para funcionar, tales como memoria, reloj y algunos periféricos. El Atmega 328P-PU es un microcontrolador basado en una arquitectura Harvard modificada de 8 bits con tecnología RISC (Reduced Instruction Set Computing), es decir con memorias separadas para programa y datos, y un conjunto de instrucciones simples para realizar la programación.

El microcontrolador combina una memoria flash interna de 32 kB para almacenar el programa, una memoria EEPROM de 1 kB para los datos no volátiles y 2 kB de memoria SRAM para datos volátiles. Tiene 23 pines de entrada/salida digital, 3 timers/contadores, interrupciones internas y externas, soporta comunicación tipo USART, I^2C y SPI, tiene 6 canales de entrada a un convertidor analógico/digital de 10 bits, tiene un timer watchdog con oscilador interno y 5 modos de ahorro de energía. Puede funcionar en un rango de tensión de 1.8 a 5.5 Volts (Lajara Vizcaíno & Pelegrí Sebastiá, 2014). Este microcontrolador utiliza un resonador o cristal de cuarzo de 16 MHz. De la hoja de datos del fabricante se pueden consultar las características más importantes de este microcontrolador (ATMEL, 2009). Los pines del microcontrolador se ilustran en la figura 3.1.

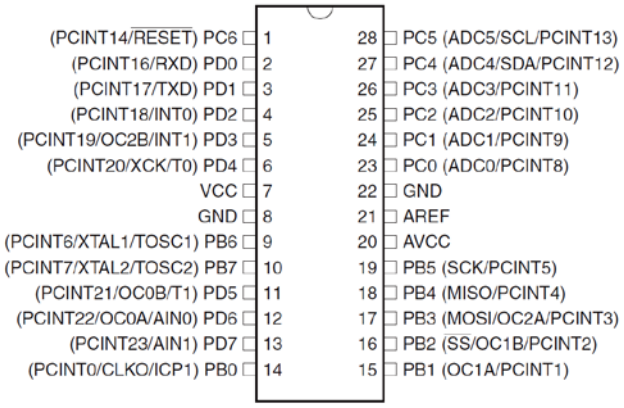


Figura 3.1 Pines del microcontrolador Atmega 328P.

3.1.2. Controlador CAN MCP2515.

En general, los controladores CAN no pueden ser directamente conectados al bus CAN, por lo que son requeridos los transceptores CAN para la conexión (Lawrenz, 2013). Como controlador CAN se seleccionó el circuito integrado MCP2515 el cual implementa la especificación CAN, versión 2.0B. Sus pines de conexión se muestran en la figura 3.2. Se

comunica con los microcontroladores vía SPI (Serial Periferial Interface) y su velocidad de transmisión puede ser hasta de 1 Mbps (Microchip Technology Inc, 2007).

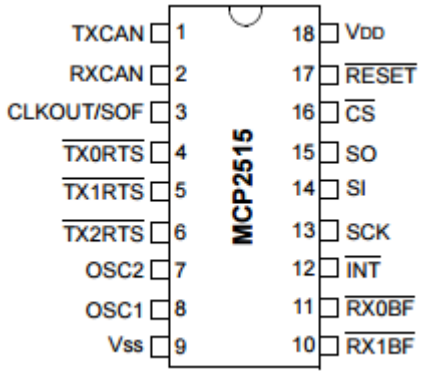


Figura 3.2 Pines del controlador CAN MCP2515.

3.1.3. Transceptor CAN MCP2551.

El transceptor CAN MCP2551, ver figura 3.3, es un dispositivo CAN que sirve como interfaz entre el controlador de protocolo y las líneas físicas del bus. Proporciona la capacidad de tener una transmisión y recepción diferencial y es totalmente compatible con el estándar ISO-11898. Puede operar a velocidades de transmisión de hasta 1 Mbps. Cada nodo en un sistema CAN debe tener un dispositivo para convertir las señales digitales generadas por el controlador CAN a señales adecuadas para su transmisión sobre el bus cableado. También provee un buffer entre el controlador CAN y los voltajes que pueden presentarse en el bus por causas externas.

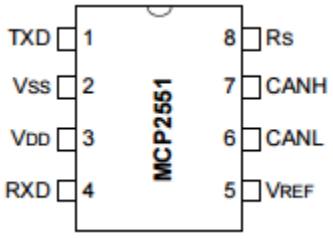


Figura 3.3 Pines del transceptor MCP2551.

Las salidas del transceptor CAN MCP2551 pueden manejar una carga mínima de 45 Ω, permitiendo un máximo de 112 nodos conectados al bus, esto es, considerando una entrada diferencial mínima de 20 kΩ y resistores de terminación de 120 Ω (Microchip Technology Inc, 2010).

3.1.4. Software EAGLE

Para el desarrollo del diseño del PCB se utilizó el software EAGLE 5.11.0, EAGLE es un programa de diseño de diagramas y PCBs con autoenrutador, este contiene un editor de diagramas electrónicos en el cual los componentes pueden ser colocados en el diagrama con un solo click y fácilmente enrutables con otros componentes a base de "cables" o etiquetas de igual manera contiene un editor de PCBs con un auteoenrutador bastante eficiente, el editor es capaz de producir archivos GERBER y demás, que son utilizados en el momento de la producción.

Eagle trae incluidas bibliotecas de componentes, sencillas de hacer y disponibles por parte de empresas, tales como SparkFun, o aficionados que las distribuyen en la red de forma gratuita.

3.1.5. Shield CAN bus.

Lo primero que se realizó en hardware fue la elaboración de un shield CAN. La figura 3 muestra el prototipo CAN para que funcione en conjunto con la tarjeta arduino. Esta tarjeta CAN tiene un relevador de salida que puede ser activado desde otro nodo CAN por lo que se pueden controlar cargas relativamente grandes como por ejemplo lámparas, bombas de riego, electroválvulas para automatización, para entre otros.

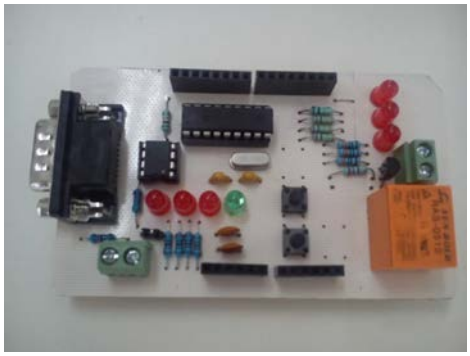
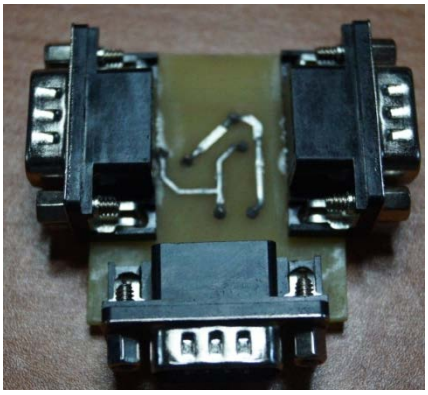


Figura 3.4 Prototipo shield CAN.

3.1.6. Bus físico.

Con la intención de facilitar la conexión de más nodos al bus del sistema, se construyeron unos conectores tipo “T”, uno de los cuales se muestra en la figura 3.5. La figura 3.6 ilustra la forma de cómo se formaría una red CAN utilizando los conectores “T”.



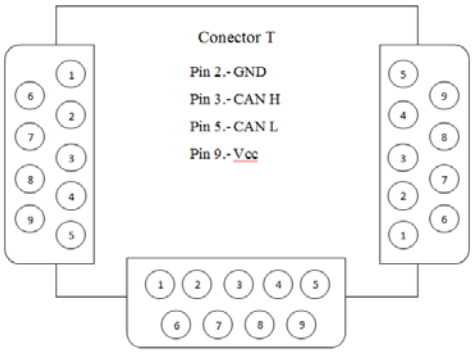


Figura 3.5 Conector "T".

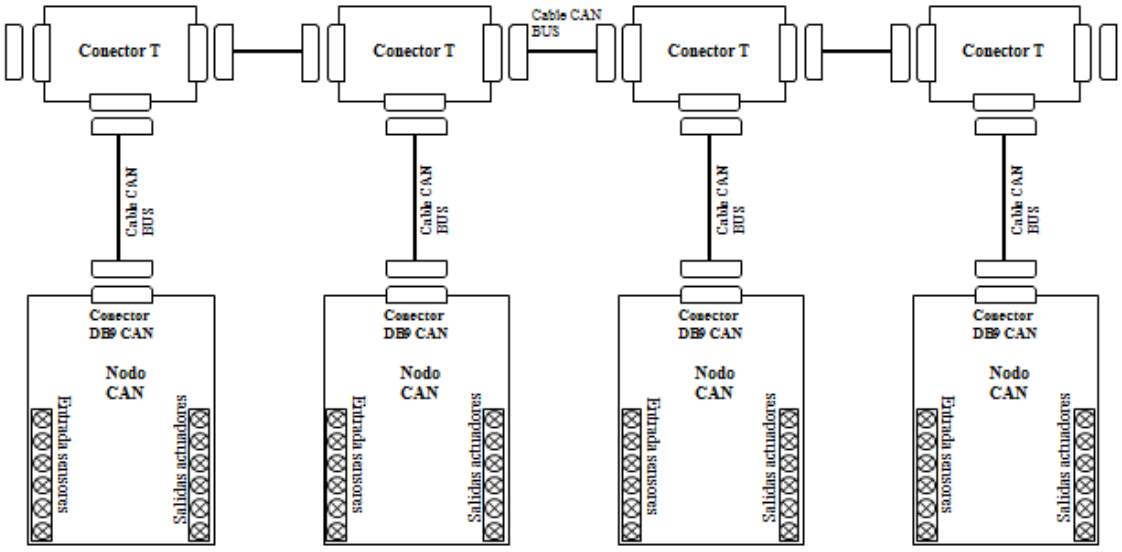


Figura 3.6 Ejemplo de conexión para red CAN.

En cada uno de los dos extremos del bus de la red CAN es necesario colocar una resistencia de terminación de 120 Ω. Esto con la finalidad de evitar perturbaciones indeseadas en los datos a transmitir. Las resistencias de terminación implementadas se muestran en la figura 3.7 y se conectan en uno de los tres extremos de los conectores “T”.

Figura 3.7 Resistencia de terminación.

3.1.7. Nodos CAN.

En las primeras implementaciones del protocolo CAN-BUS se desarrollaron pruebas con con shields comerciales de las compañías Sparkfun y Seeedstudio para Arduino. Para realizar las pruebas fue necesario descargar la librería que hace referencia la página de Seeedstudio, la biblioteca bajada de esta página de internet también es útil para el shield de Sparkfun. En la figura 3.8 se muestra el PCB del shield CAN bus que comercializa la compañía Seeedstudio.

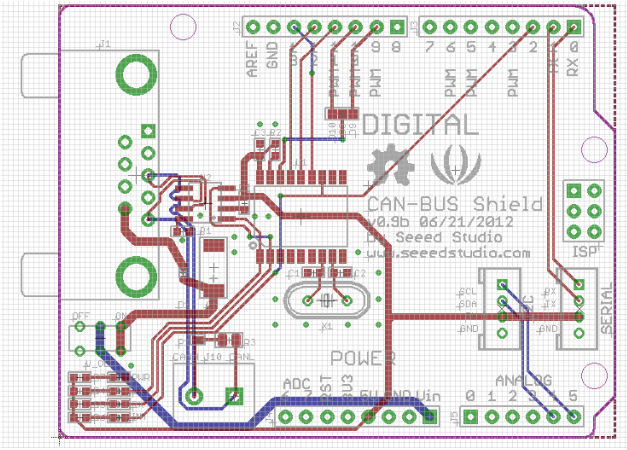


Figura 3.8 PCB del shield CAN bus para Arduino.

Después de realizar las pruebas y verificar que los nodos y el software implementado en Arduino funcionan correctamente se llevó a cabo el desarrollo de una tarjeta propia. En la figura 3.9 se aprecia el diseño de la primera tarjeta PCB CAN-BUS.

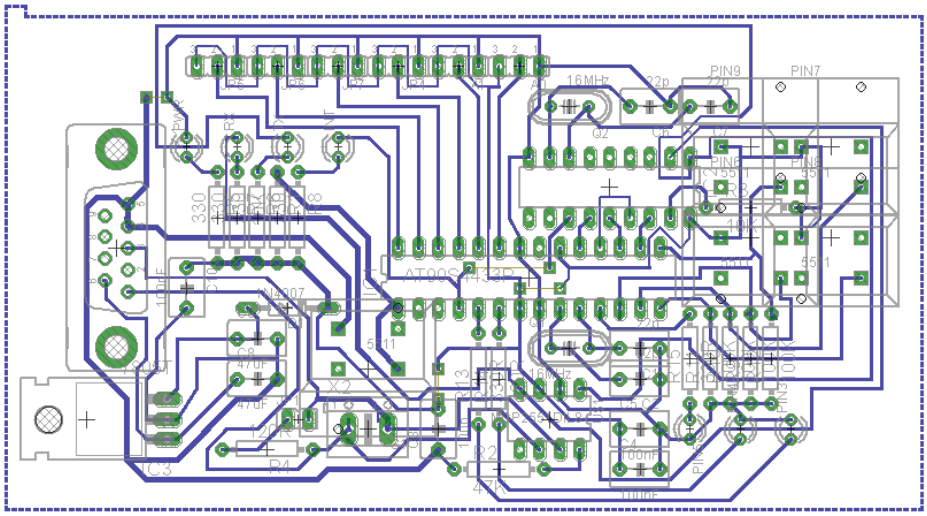


Figura 3.9. Primer diseño de placa CAN bus.

Al probar la placa mostrada en la figura 3.9 se percató que hacía falta hacerle algunas modificaciones como por ejemplo incluir la alimentación desde el conector DB9 y en la salidas y entradas se requerían conectores para una fácil conexión con los actuadores y sensores externos. Por lo que se decidió adicionar unos bornes con tornillos.

Para el desarrollo del diagrama esquemático esta segunda placa se utiliza una de las heramientas indispensables de EAGLE, el enrutado a base de etiquetas facilitando su armado y entendimiento. Se desarrollaron 4 partes el POWER, INPUT_OUTPUT, MICROCONTROLADOR, CAN_SPI como se observa en la figura 3.10.

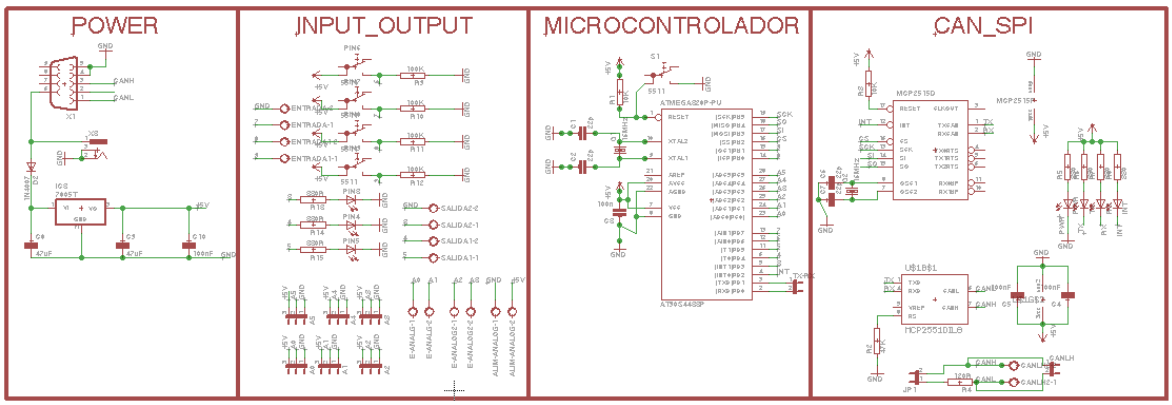


Figura 3.10. Diagrama esquemático de la tarjeta CAN.

Para el trazado de las pistas de cobre fue necesario utilizar una placa fenólica de doble cara. En las figuras 3.11 y 3.12 se muestran las vistas inferior y superior respectivamente de la placa CAN. En la figura 3.13 se muestran las entradas y salidas.

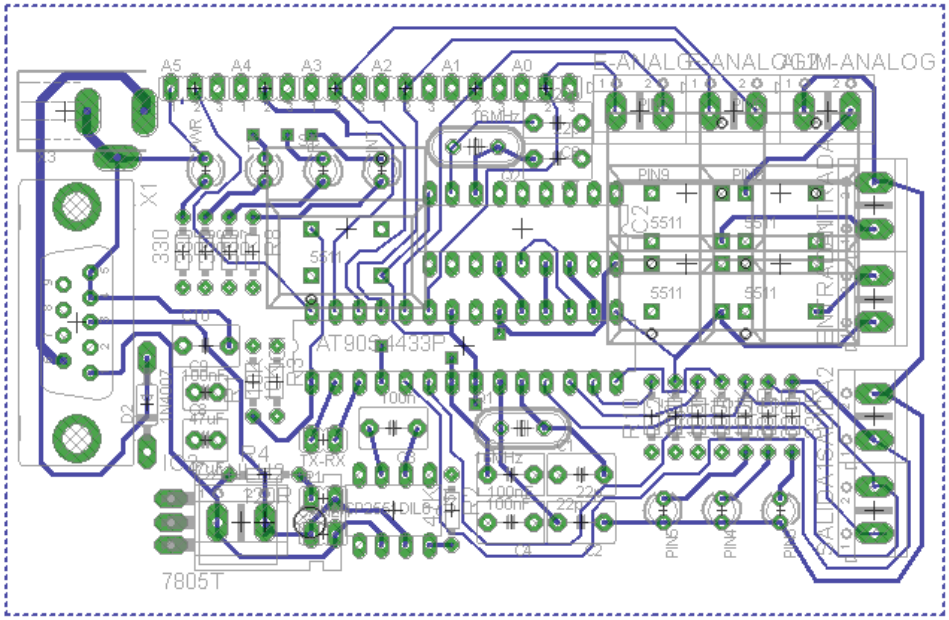


Figura 3.11. Pistas en la vista inferior de la placa CAN.

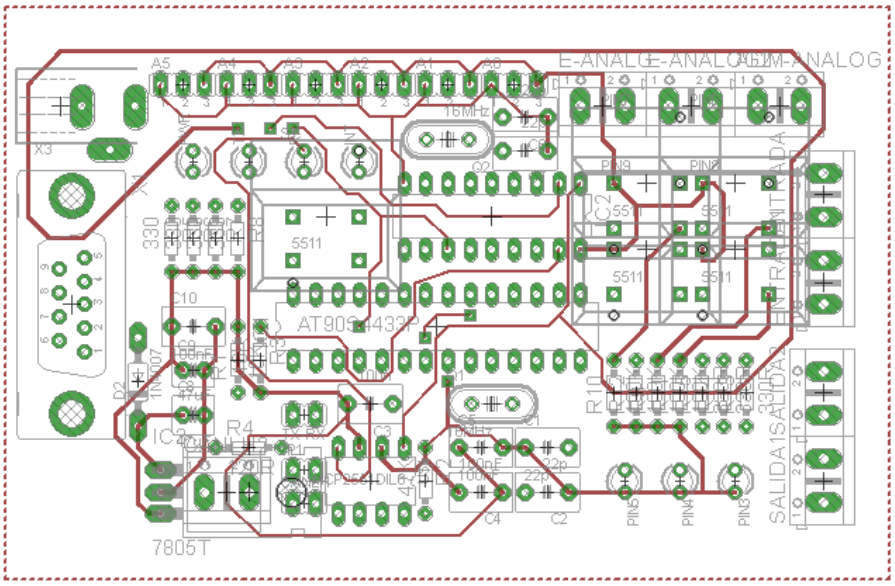


Figura 3.12. Pistas en la vista superior de la placa CAN.

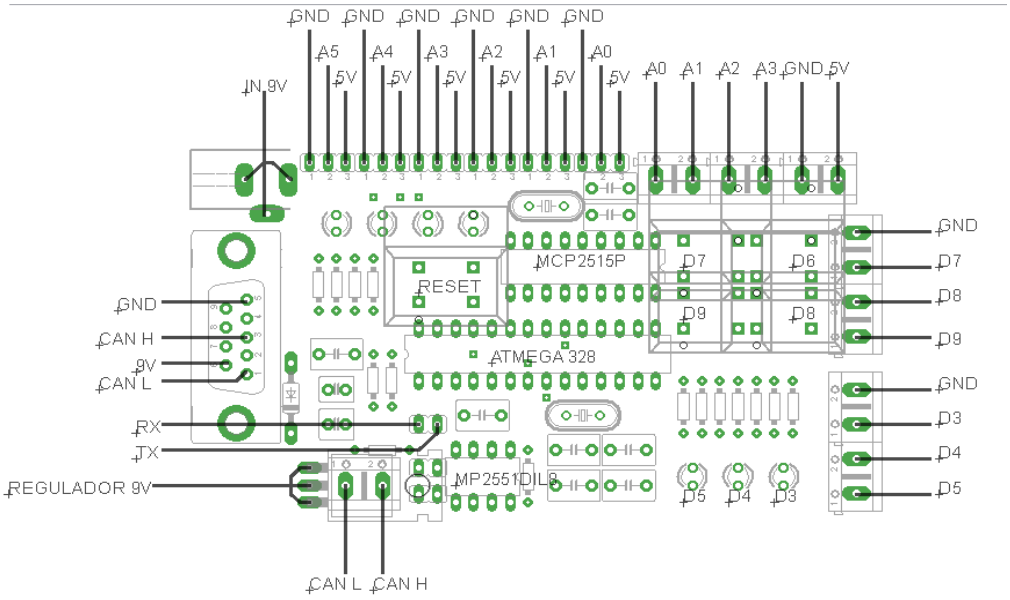


Figura 3.13. Entradas/Salidas de la tarjeta CAN.

En la figura 3.14 se puede ver la placa fabricada, cuyas dimensiones son de 10 x 6.5 cm. Cada placa contiene un microcontrolador ATMEGA 328P, un controlador CAN MCP2515 y un transceptor CAN MCP2551. Esta placa por si misma será un nodo CAN y únicamente se comunica a través del bus CAN, no se puede comunicar a través de USB a una computadora. Para visualizar los datos en una laptop o computadora es necesario de un nodo CAN con comunicación USB. El nodo CAN con Comunicación USB se puede implementar usando una tarjeta Arduino UNO en conjunto con un shield CAN, ya sea comercial, o la shield CAN elaborada en este mismo proyecto, mostrada con anterioridad en la figura 3.4.

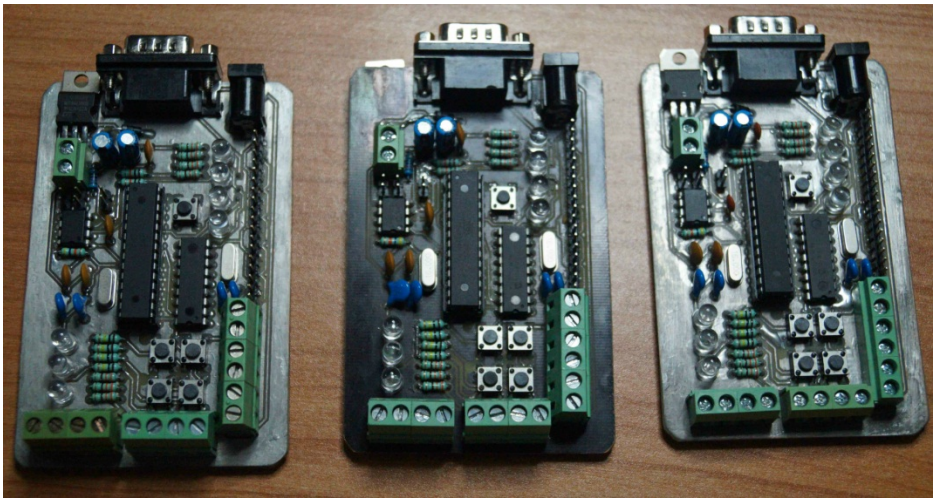


Figura 3.14 Nodos CAN elaborados.

3.2. Software

El trabajo en software para este proyecto se puede considerar, que a su vez, está dividido en dos partes. La primera es para programar al microcontrolador, para lo cual usaremos el software libre para programar Arduino. El software del sistema Arduino se puede descargar del sitio www.arduino.cc. La segunda es para generar la interfaz gráfica. Este segundo programa permite visualizar el tráfico de datos en el bus CAN, e indica el contenido de los mensajes, es decir, es la interfaz gráfica del usuario.

3.2.1. Software para el microcontrolador

El software para programar al microcontrolador ATMEGA328P presenta la gran ventaja de ser un sistema abierto y con muchos desarrollos existentes. Para la comunicación entre nodos se implementaron tarjetas electrónicas basadas en el microcontrolador antes mencionado. La disponibilidad librerías es la razón por la que se seleccionó este microcontrolador como "cerebro" de los nodos. La comunicación entre las tarjetas (nodos) se logró empleando las siguientes librerías:

- #include <mcp_can.h>
- #include <SPI.h>

El siguiente sketch muestra el código para que un nodo pueda recibir datos y desplegarlos en el monitor serial.

```
Serial.begin(115200);
  CAN0.begin(CAN_500KBPS);        // Puesta a 500kbps la velocidad de comunicación

CAN0.readMsgBuf(&len, rxBuf);    // Lectura de datos: len = longitud de datos, buf = data byte(s)
    rxId = CAN0.getCanId();                 // Obtiene el ID del mensaje
    Serial.print("ID: ");
    Serial.print(rxId, HEX);
    Serial.print("  Data: ");
    for(int i = 0; i<len; i++)             // Imprime cada byte de la trama recibida
    {
     if(rxBuf[i] < 0x10)                  // Pone un cero para formar cadenas uniformes en longitud
     {
      Serial.print("0");
     }
     Serial.print(rxBuf[i], HEX);  //Imprime la trama recibida con sus datos en hexadecimal
     Serial.print(" ");
```

Para el envío de tramas, se codifica de la siguiente forma:

```
Serial.begin(115200);
```

```
 // Inicializa el baudrate del CAN BUS a: 500kbps
 if(CAN0.begin(CAN_500KBPS) == CAN_OK) Serial.print("Inicialización correcta!!\r\n");
 else Serial.print("Inicializacón Fallida!!\r\n");
unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};
void loop()
{
 // send data:  id = 0x00, standrad flame, data len = 8, stmp: data buf
 CAN0.sendMsgBuf(0x00, 0, 8, stmp);
 delay(100);                  // Delay de 100 milis
}
```

Para realizar la interfaz gráfica se emplea el lenguaje de programación Visual Basic por lo que a continuación se encuentra una descripción histórica del mismo.

3.2.2. Introducción a Visual Basic

Lenguaje BASIC. El lenguaje BASIC (Beginner's all-purpose Symbolic Instruction Code) fué inicialmente un lenguaje de programación de propósito general para principiantes, el cual ha sufrido cambios radicales desde su creación a la fecha.

Visual Basic. La aparición de Windows a mediados de los años ochenta, supuso una gran revolución en el mundo del PC. Los usuarios de esta plataforma, disponían ahora de un entorno gráfico de trabajo, que facilitaba en gran medida su labor y dejaba atrás paulatinamente la aridez del trabajo en el modo comando (la interfaz MS-DOS). La aplicación tomaba el control del sistema operativo, el cuál esperaba las instrucciones del programa para ir ejecutándolo; sólo podíamos tener en ejecución una aplicación en cada momento; el modo gráfico era proporcionado por librerías específicas del lenguaje que estuviéramos utilizando.

La nueva arquitectura de programación de Windows cambiaba todos los esquemas que pudiera conocer el programador: programación basada en eventos y orientada a objetos; modo gráfico proporcionado y gestionado por el sistema y no por el lenguaje; múltiples aplicaciones funcionando simultáneamente; y lo más novedoso, y también más traumático para los programadores, el hecho de que el sistema enviaba información mediante mensajes a nuestra aplicación, a los que debíamos dar una adecuada respuesta, lo que suponía que a partir de ese momento, era el sistema el que controlaba a la aplicación, con lo que se acabaron los tiempos en los que nuestro programa tomaba el control absoluto del sistema operativo.

En estos primeros tiempos de la programación para Windows, sólo los llamados gurús de C y Windows, que conocían perfectamente los trucos y la arquitectura del nuevo entorno operativo de Microsoft, eran capaces de desarrollar las nuevas aplicaciones, para el asombro de los más modestos programadores de a pie (Blanco, 2002).

Uno de los grandes problemas para el programador, consistía en que debía centrarse excesivamente en el desarrollo de la parte del interfaz de la aplicación, controlando hasta el más mínimo detalle de lo que el usuario pudiera hacer con una ventana: captura y envío de mensajes desde y hacia las ventanas de la aplicación, gestión de manipuladores de ventanas y contextos de dispositivos para el dibujo de todos los elementos de la aplicación, escritura de los procedimientos de ventana, etc.; el más simple programa que mostrara un mensaje tenía un gran número de líneas de código.

En un escenario como este, en la mayor parte de los casos, se desviaba la atención de lo verdaderamente importante en la aplicación: la funcionalidad que necesitábamos dar al usuario. Programar una simple entrada de datos para almacenar en un fichero era toda una odisea. Por añadidura, tampoco existían herramientas de desarrollo que facilitaran la labor del programador, todo consistía en un puñado de aplicaciones independientes que funcionaban en modo comando: Compilador, enlazador, editor de código, etc., lo que hacía

que un programador no pudiera alcanzar el mismo nivel de productividad que tenía desarrollando las aplicaciones MS-DOS de aquel entonces.

Conscientes del problema que entrañaba el que los desarrolladores no migraran de forma masiva a la creación de programas para Windows, Microsoft puso en marcha un proyecto con el nombre clave Thunder (Trueno), encaminado a crear una herramienta de desarrollo que facilitara la escritura de programas para Windows. En 1991, este proyecto dio como fruto la primera versión de Visual Basic.

De 1991 a la fecha han salido a la luz distintas versiones del visual Basic, que sugieren mejoras a las necesidades de los desarrolladores: La versión, VB 3.0, aportó dos novedades importantes: nos liberó de los limitados controles VBX, hacia el más robusto y flexible modelo de controles OLE, también conocidos como controles OCX; y proporcionó soporte para manejar bases de datos a través de ODBC. La versión 4.0 disponía a su vez de versiones para crear aplicaciones que se ejecutaran para 16 o 32 bits, de forma que ya podíamos crear aplicaciones para el nuevo sistema operativo. Permitía la programación orientada a objetos. La versión 5.0 permitía la compilación de las aplicaciones a código nativo, superando la más lenta de versiones anteriores, basada en pseudo-código; como resultado, nuestros programas podían ejecutarse casi tan velozmente como los de C++. Ésta introdujo la posibilidad de crear controles Actives, con lo que ya no era necesario recurrir a C++ para crear nuestros propios controles, superando una nueva limitación. Respecto al manejo de bases de datos, se incluía una nueva jerarquía de objetos para datos: DAO (Data Access Objects). VB 6 incluía un nuevo modelo de acceso a datos mejorado: ADO (ActiveX Data Objects), cuya finalidad era la de reemplazar a los medios existentes hasta ese momento: RDO y DAO, por una única jerarquía de objetos de acceso a datos de cualquier tipo y en cualquier situación: bases de datos locales, cliente/servidor, acceso a datos a través de Internet, etc. Este modelo de objetos para datos, si bien se conserva en .NET, ha sido profundamente renovado para atender a las exigencias de las aplicaciones actuales.

VB no ha sido ajeno al el desarrollo de aplicaciones para Internet, y en la versión 6.0, se incluían elementos que intentaban proporcionar al programador, capacidades de acceso a Internet para evitar su cambio a otras herramientas o lenguajes más específicos para la Red. (Blanco, 2002). Por otro lado, un punto fuerte de la programación web, en el que VB sí ha tenido éxito, ha sido el desarrollo de componentes, que encapsulan reglas de negocio, y pueden ser llamados desde páginas ASP. Estos componentes, compilados en formato de DLL, se ejecutan en la capa intermedia del esquema de funcionamiento en tres capas de una aplicación en Internet. A pesar de los intentos de dotarle de capacidades para el desarrollo de aplicaciones web, VB adolecía de algunos aspectos que han influido en que no haya podido entrar en este sector de la programación.

Algunas de estas características son la falta de un pleno soporte para la programación orientada a objetos, en concreto, la falta de herencia; la creación y manipulación multihebra; poca interacción con otros lenguajes como C++; una pobre gestión de errores, etc. (Blanco, 2002)

Como se puede ver Visual Basic 2010 es un lenguaje que puede ser utilizado para construir aplicaciones de forma sencilla y conveniente, pero esto no es realmente el problema, lo que debe tenerse en cuenta es que Visual Studio 2010 es un entorno integrado para la construcción, prueba, depuración, y el despliegue de una gran variedad de aplicaciones: Aplicaciones de Windows, Aplicaciones Web, clases y optimización de controles e incluso aplicaciones de consola. Ofrece numerosas herramientas para automatizar el proceso de desarrollo, herramientas visuales para realizar muchas de las tareas de diseño y programación común, y otras características más (Petroutsos, 2010).

.NET. Es toda una nueva arquitectura tecnológica, desarrollada por Microsoft para la creación y distribución del software como un servicio. Esto quiere decir, que mediante las herramientas de desarrollo proporcionadas por esta nueva tecnología, los programadores podrán crear aplicaciones basadas en servicios para la web. La plataforma .NET

Framework, proporciona la infraestructura para crear aplicaciones y el entorno de ejecución para las mismas (Blanco, 2002, pág. 32).

Visual Basic .NET. VB.NET aporta un buen número de características que muchos programadores de VB han demandado desde hace largo tiempo. En cierto modo, algunas de estas incorporaciones hemos de agradecérselas a la plataforma .NET, ya que al integrar VB dentro del conjunto de lenguajes de .NET Framework, dichos cambios han sido necesarios, no ya porque los necesitara VB, sino porque eran requisitos derivados de la propia arquitectura de .NET. Entre las novedades aportadas por VB.NET tenemos plenas capacidades de orientación a objetos (Full-OOP), incluyendo por fin, herencia; Windows Forms o la nueva generación de formularios para aplicaciones Windows; soporte nativo de XML; gestión de errores estructurada; un modelo de objetos para acceso a datos más potente con ADO.NET; posibilidad de crear aplicaciones de consola (ventana MS-DOS); programación para Internet mediante Web Forms; un entorno de desarrollo común a todas las herramientas de .NET (Blanco, 2002).

3.2.2. Interfaz gráfica.

El software empleado para concentrar la información de todos los nodos está enteramente escrito en Visual basic 2010. La razón de esta elección es por la simpleza de codificación, poco peso del programa resultante, facilidad de depuración y modificación, ya que la mayoría del software de entorno corre en algo similar a una Consola Virtual, tal como en lenguajes de programación de alto nivel como por ejemplo, Java, cuyo código corre en una máquina virtual. En contraparte, las desventajas de esta elección son, entre otras, lentitud de ejecución del código, necesidad de tener instalado cuando menos .NET Framework 3.5, Visual Studio 2010 Express, Visual Studio 2008, ejecución de código únicamente en plataforma Windows, por citar las más importantes.

La figura 3.15 presenta la interfaz gráfica, creada con Visual Basic 2010, desde la cual el usuario podrá visualizar las tramas de datos presentes en el bus. Los datos recibidos los

puede almacenar en un archivo .txt para un posterior análisis. También permite enviar datos en el bus para que otros nodos lo puedan recibir. Los datos se pueden visualizar en formato decimal o en hexadecimal, según convenga al usuario.

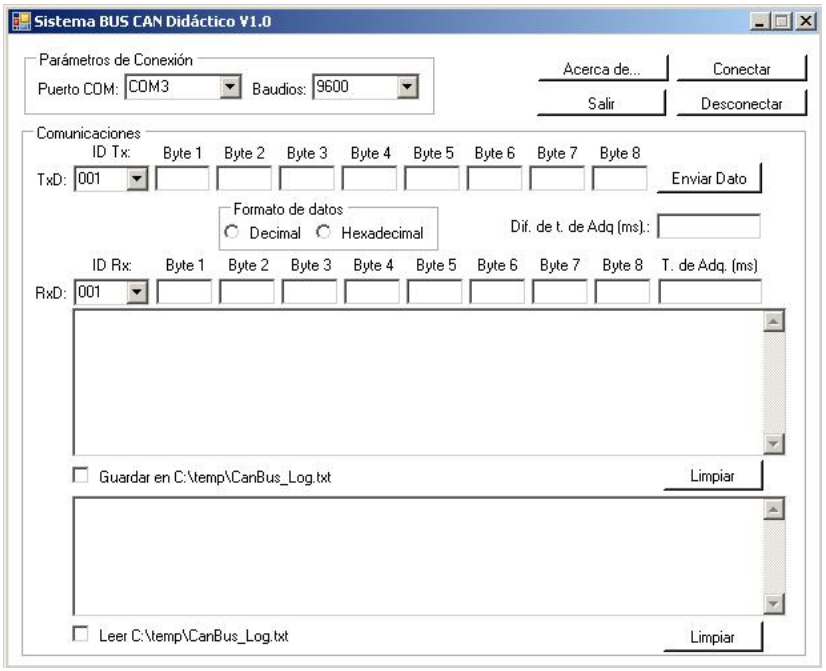


Figura 3.15 Interfaz gráfica desarrollada en visual Basic 2010

CAPÍTULO 4. RESULTADOS Y CONCLUSIONES

4.1. Envío y recepción de datos.

Para probar el sistema de interconectaron 3 nodos CAN, uno se configuró como nodo CAN cliente para que le mande los datos necesarios a la interfaz gráfica y se puedan ver en la pantalla de una computadora. Los otros dos se configuraron como nodos CAN clientes, cada uno de estos pueden enviar al bus los datos de seis sensores analógicos. Los sensores se pudieron simular con potenciómetros. Para la interconexión de los nodos CAN se emplearon conectores “T”. Lo anterior se ilustra en la figura 4.1.

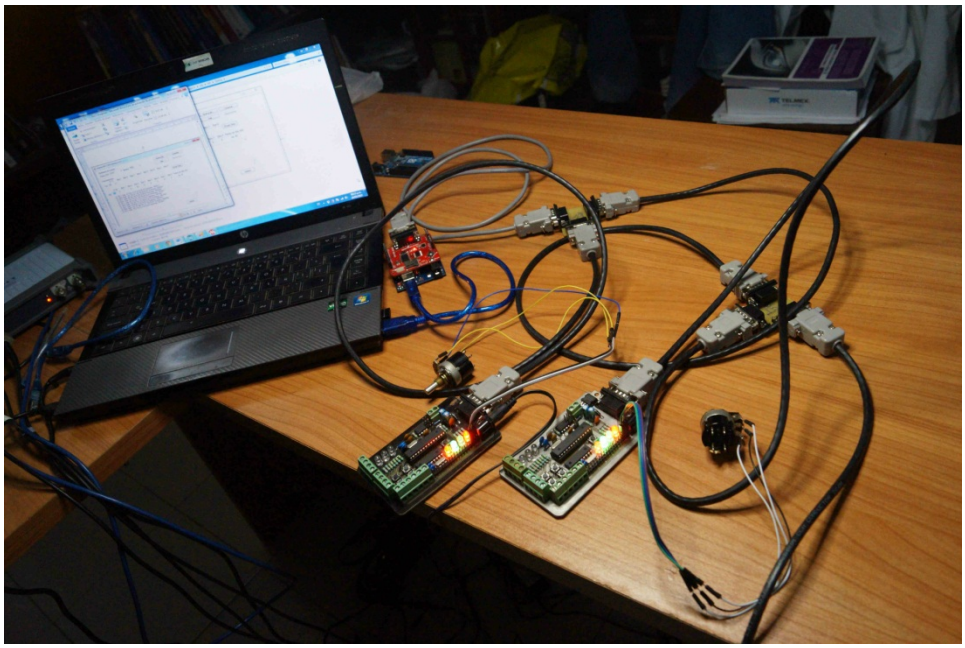


Figura 4.1 Red CAN implementada.

La interfaz gráfica realizada en Visual Basic 2010 permite monitorear, enviar y almacenar los datos de los nodos CAN. La interfaz gráfica leyendo los datos de dos nodos CAN se muestra en la figura 4.2.

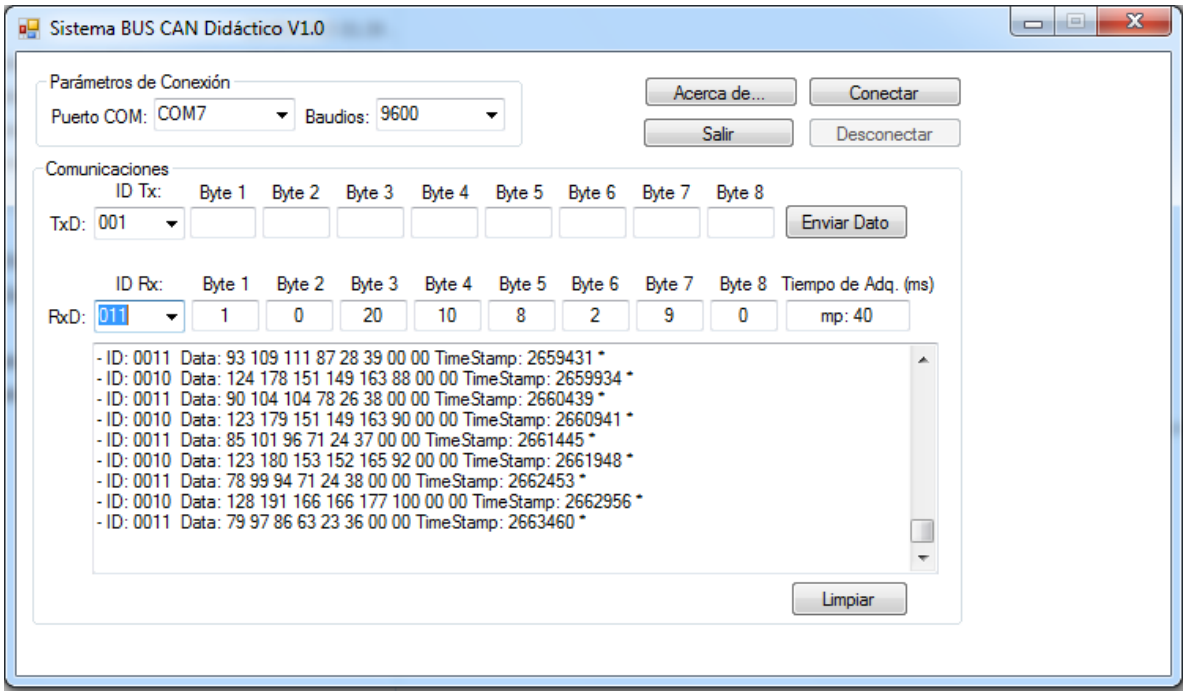


Figura 4.2 Interfaz gráfica desplegando los datos de bus CAN.

La velocidad de transmisión en el bus CAN fue establecido a 1 Mbit/s sin posibilidad de ser cambiada desde la interfaz gráfica, en caso de necesitar cambiar esta velocidad se requiere entrar el entorno de programación de Arduino. Para establecer comunicación entre la computadora y la red CAN se debe seleccionar en la interfaz gráfica el puerto COM utilizado, el cual está configurado para una velocidad de 9600 baudios. En la sección de datos de sensores se despliegan los valores enviados por el nodo seleccionado en CAN ID. En la sección “Datos en el CAN bus” se muestran todos los datos del Can bus sin importar el valor de ID. Al seleccionar “Logueo en C/Temp/ los datos se guardan en un archivo de texto en la dirección indicada.

Con la finalidad de verificar que los datos mostrados en la interfaz gráfica implementada se conectó a esta red CAN un dispositivo llamado CAN bus analyzer comercializado por la compañía Microchip, lo cual se muestra en la figura 4.3. El CAN bus analyzer y la interfaz gráfica mostraron coincidencia en los datos.

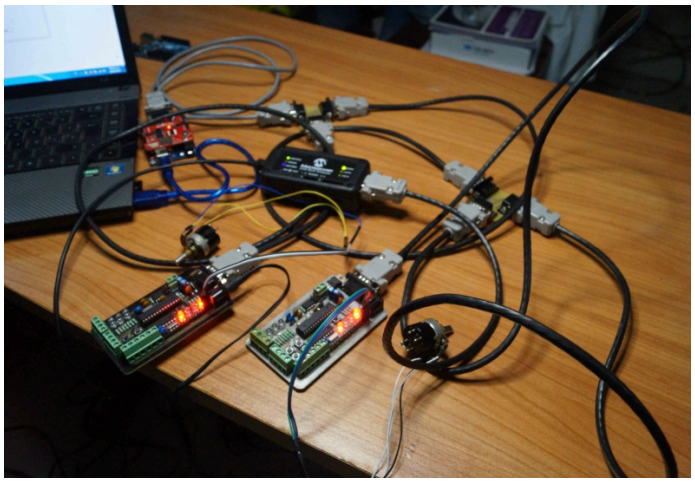


Figura 4.3 CAN analyzer conectado al bus CAN.

La figura 4.4 muestra la interfaz gráfica utilizada para verificar que los datos desplegados en la interfaz desarrollada en esta tesis corresponden con los que realmente están en el bus. Los datos comparados coincidieron, lo que indica que la interfaz y los nodos trabajan satisfactoriamente y son confiables.

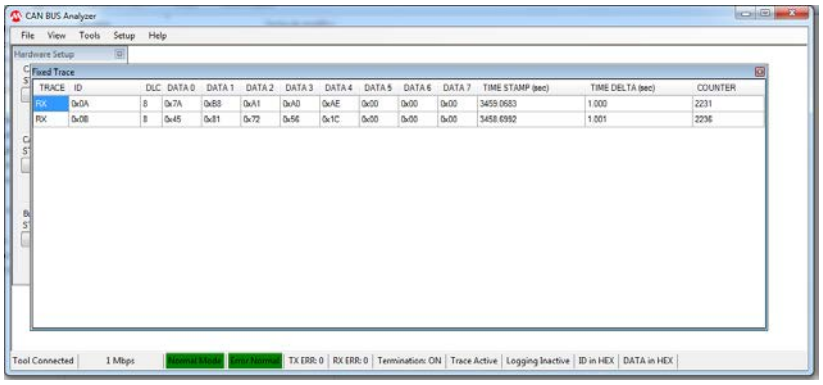


Figura 4.4 Interfaz gráfica del CAN bus Analyzer.

4.2. Conclusiones y trabajo futuro.

El protocolo CAN en topología de bus sólo necesita dos cables para realizar la comunicación. El bus CAN permite la reducción de cableado y ofrece un protocolo de comunicación eficiente entre sensores, actuadores, controladores y otros nodos para aplicaciones en tiempo real.

Al disponer del hardware y de una interfaz (software) de diseño propios permite tener un sistema flexible que se puede adaptar a las necesidades de diferentes proyectos. Esto da solución al problema planteado de no disponer de plataformas didácticas flexibles y de bajo costo, en la cual se pueda enseñar a los estudiantes.

El sistema desarrollado permite dar solución a la interconectividad para sensores que están a cierta distancia uno de los otros. Se pueden monitorear varios parámetros de un sistema dado, como por ejemplo, temperatura, presión, posición, velocidad y cualquier otra variable de interés. En este trabajo los valores enviados fueron visualizados en una interfaz gráfica elaborada en Visual Basic la cual se podría adaptar según las necesidades de diferentes proyectos.

Para poder llevar a cabo la implementación, se tuvo la idea de enlazar la comunicación serial entre Arduino y Visual Basic. La razón de esto, radica en el hecho de que Arduino envía sus datos al puerto serial en forma de cadenas de caracteres, es decir, puede tratarse como simple texto. La función primordial de Visual Basic en este sistema, es el de filtrado de cadenas de texto: se efectúa el rastreo de una cadena recibida, y con la función trim() se extrae el dato requerido y se coloca en el TextBox deseado.

Como trabajo futuro se contempla adicionarle a la tarjeta electrónica de cada nodo CAN la capacidad de comunicación usando USB. Otra mejora contemplada es la miniaturización de la tarjeta electrónica y la utilización de algún microcontrolador con CAN integrado, como por ejemplo, el PIC18F2680 de la compañía Microchip. También sería

conveniente adicionarle un módulo que permita observar las señales eléctricas presentes en el bus, como si se tratase de un osciloscopio o de un analizador lógico. Con la finalidad de mejorar el sistema en futuras versiones, se recomienda tomar en consideración las opiniones de los usuarios.

ANEXOS

A1. Manual de instalación y uso

Introducción

El presente manual tiene como finalidad la familiarización del usuario con el sistema educativo para el monitoreo e interacción con señales en el bus CAN. El software desarrollado es llamado de aquí en adelante CAN Bus Edu. Dicho manual muestra los aspectos más importantes del programa, tales como su utilización, alcances y limitaciones. Posteriormente se explica de manera directa el apartado técnico referente a su creación, interfaces, protocolos y lo concerniente al proceso de ingeniería relativo a su diseño.

La adquisición de datos es una tarea que se puede realizar por diversos medios y tecnologías actualmente disponibles, mismas que pueden facilitar en menor o mayor medida dicha tarea. Este sistema pretende ofrecer una forma más para lograr dicha labor, pero de una forma sencilla, sin muchos requisitos técnicos, de manera rápida y escalable. El problema que se pretende resolver parte de la idea de que la adquisición de datos puede realizarse en distintos ambientes, sobretodo en ambientes pesados donde la comunicación por medios tradicionales puede restar fiabilidad o bien, su implementación no es viable.

El software desarrollado, además de usar el hardware implementado en esta tesis, ofrece la facilidad de poder usar hardware comercialmente disponible y ampliamente usado y conocido como es el caso de la tarjeta Arduino. El Arduino no tiene integrado el hardware necesario para implementar los nodos CAN, pero se dispone en el mercado de las tarjetas CAN (CAN bus shield) fácilmente adaptables al Arduino, como por ejemplo las comercializadas por Sparkfun y Seeedstudio.

Para la puesta en marcha del sistema se requiere de un nodo servidor (nodo con conexión USB para la visualización y envío de datos desde una computadora portátil) y de 1 hasta 112 nodos clientes (nodos a los que se podrán conectar sensores y cuyos datos se

observarán en la computadora). Cada nodo CAN cliente puede aceptar hasta seis sensores analógicos.

Requerimientos de Hardware

- PC o Laptop compatible con Windows XP, 7, 8 u 8.1 en sus versiones de 32 o 64 bits. Al menos 1GB de RAM para XP, y 2GB para las demás versiones. Espacio libre de al menos 40MB en disco duro para el software y para los respectivos archivos de loggueo en modo texto.

- Arduino Uno/Mega con shield CAN conectado como Servidor a una PC o laptop, mediante un cable USB tipo B a un puerto USB 2.0, correctamente configurado y con conexión correcta al monitor serial en el respectivo puerto COM asignado por el sistema operativo.

- Al menos uno y hasta 112 nodos CAN compatibles con la codificación de sketches del Arduino Uno.

- Cable UTP con la longitud necesaria con conector DB9 en cada uno de sus extremos.

- Conectores “T”.

- 2 conectores con Resistencias de terminación de 120 Ω .

- Sensores y transductores según la necesidad, conectados a los nodos CAN.

- Una fuente de alimentación de 9 a 12 Volts.

Requerimientos de Software

- Como se mencionó anteriormente, se requiere tener instalado el sistema operativo Windows, mínimo la versión XP, soportando hasta la 8.1 en sus versiones de 32 y 64bits.

- Tener instalado el software de programación para Arduino y con los controladores ya configurados. Se necesita la librería mcp_can.h la cual se puede descargar de https://github.com/coryjfower/MCP_CAN_lib o de http://www.seeedstudio.com/wiki/CAN-BUS_Shield.

- Visual Studio 2010 o 2010 Express

- .NET Framework 3.5

- Visor de archivos .txt

Procedimiento de Carga de Software

Cargar el sketch Servidor_Recepcio.ino en el Arduino elegido como servidor. Este Arduino únicamente se encargará de concentrar las cadenas de datos con el estatus de los sensores y pines digitales de cada uno de los nodos CAN Clientes, así como gestionar el volcado de datos y las órdenes recibidas desde el Software Can Bus Logger corriendo en la PC o Laptop. El Arduino Servidor no llevará conectados sensores ni cargas a activar, únicamente las líneas CAN H y CAN L conectadas bus CAN.

Cargar el sketch Cliente_Envio.ino a cada uno de los nodos CAN clientes. Este sketch se encarga de tomar el status de cada Cliente, transformarlo en una cadena CAN válida y enviarla al Servidor para la gestión de la información. Después de la carga de los sketches, no es necesario mantener conectados los nodos Cliente a la PC, la red CAN dispone de alimentación. Recuerde mantener conectado el Arduino Servidor, que es el que se encargará de gestionar los datos entrantes y salientes junto con el software Logger 1.exe.

Ejecutar el software aplicativo CAN Bus Edu.exe (software desarrollado en esta tesis). Aparecerá una interfaz gráfica como se muestra en la figura 1.

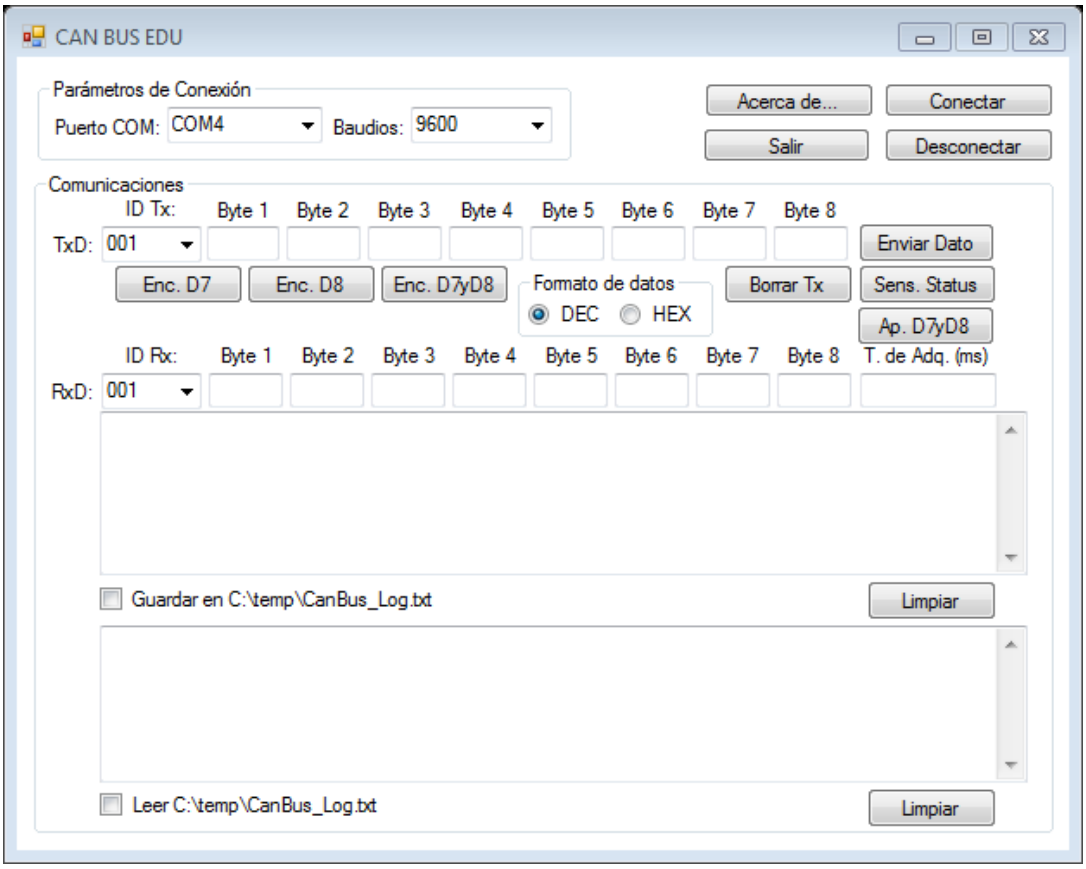


Figura A1.1. Aspecto general de CAN Bus Edu.

En caso de no haber hardware conectado, el programa enviará un aviso como se muestra en la figura 2.

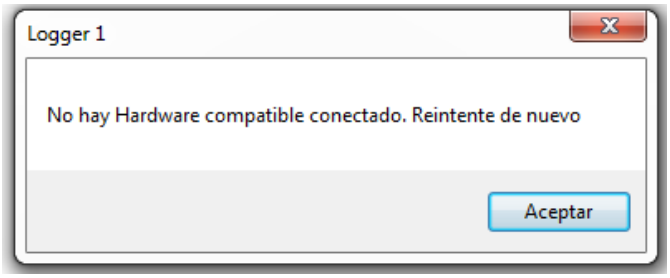


Figura A1.2. Mensaje de aviso de hardware no detectado.

Una vez conectado el Arduino Servidor con el CAN Bus Shield y su puerto COM asignado por el sistema operativo, seleccionar la velocidad de comunicación serial (esta es independiente de la velocidad de envío y recepción de las tramas, que por defecto está puesta en 1000kbps). Se recomienda una velocidad de 9600 baudios.

Seleccionar el puerto COM donde esté conectado el Arduino Servidor. Presionar en CONECTAR y la interfaz ya está habilitada para enviar y recibir datos.

El sistema está desarrollado de tal forma que la gestión de recibir y enviar datos se realiza desde el software CAN Bus Edu.exe, siempre y cuando estén cargados los sketches en los respectivos Arduinos. El monitor serial del nodo CAN servidor puede servir como visualizador de las tramas pero sin poder interactuar entre ellos y, esto siempre y cuando el software CAN Bus Edu.exe no esté activo.

En la figura 3 se observa el monitor serial del nodo CAN Servidor. Se muestra el estatus del único nodo CAN Cliente conectado a la red CAN. En caso de existir más nodos, se visualizaría en orden cada uno de ellos con sus respectivas cadenas de estatus de sensores.

Para un correcto funcionamiento del sistema, se recomienda que cada nodo CAN Cliente envíe y reciba tramas a la misma velocidad de ajuste contenida en el nodo CAN Servidor.

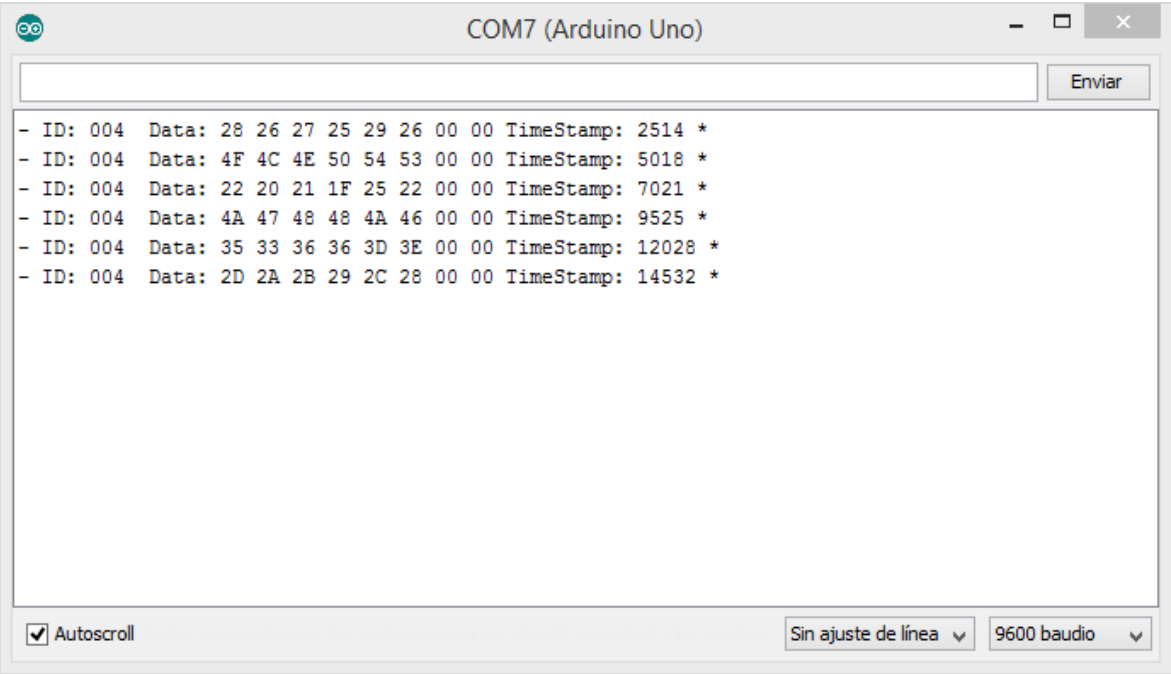


Figura A1.3. Monitor serial del nodo CAN Servidor recibiendo la trama de un único Nodo CAN Cliente conectado.

BIBLIOGRAFÍA.

ATMEL. (2009). *ATMEL*. Recuperado el 10 de Mayo de 2015, de http://www.atmel.com/images/8161s.pdf

Blanco, L. M. (2002). *Programación en Visual Basic.NET*. Madrid, España: Grupo Eidos.

Bril, R. J., Lukkien, J. J., & Davis, R. (2006). *Message response time analysis for ideal controller area network (CAN) refuted.* England: University of York.

CAN in Automation. (2015). *CAN in Automation*. Recuperado el mayo de 2015, de www.can-cia.org

CANaerospace. (2015). *CANaerospace, the airborne CAN interface standar*. Recuperado el 22 de mayo de 2015, de http://www.canaerospace.net/canaerospace.html

Chamú Morales, C. A. (2005). *Desarrollo de un sistema educativo para la enseñanza del protocolo de comunicaciones CAN*. Oaxaca, México: Universidad Tecnológica de la Mixteca.

Defaz Andrango, M. (2007). *Estudio del protocolo CAN (Controller Area Network) y su aplicación en redes de control.* Quito: Escuela Politécnica Nacional.

Di Natale, M., Zeng, H., & Giusto, P. (2012). *Understanding and Using the Controller Area Network Communication Protocol, Theory and Practice.* New York, USA: Springer.

Dressler, C., & Fischer, O. (2007). *Industrial Automation Asia*, 22-23.

IXXAT. (2004). *Controller Area Network (CAN) -Introduction.* Recuperado el 23 de mayo de 2015, de http://ixxat.com/can-controller-area-network-introduction_en.html

Kaschel, H., & Pinto, E. (2002). *Análisis protocolar del bus de campo CAN*. Universidad de Santiago de Chile.

Kvaser. (2015). *SDS, DeviceNet and CAN Kingdom*. Recuperado el 21 de mayo de 2015, de http://www.kvaser.com/sds-devicenet-can-kingdom/

Lajara Vizcaíno, J. R., & Pelegrí Sebastiá, J. (2014). *Sistemas integrados con Arduino.* Barcelona, España: Marcombo.

Lawrenz, W. (2013). *CAN System Engineering.* Germany: Springer.

Llanos López, M. J. (2011). *Circuitos eléctricos y auxiliares del vehículo.* Madrid, España: Paraninfo.

Mesa Montoya, C. A. (2008). *CAN bus, introducción a los sistemas de comunicación del vehículo*. Recuperado el 28 de mayo de 2015, de http://api.ning.com/files/MXZLZ1wc*WJrOi9Hb2Qbgl6M75c6Ftff39vufF8*Oo-21INeBl51XhYb*7kA0EyeXAVrET5HCl3Of6ZIbw9P1xLw55R76nsa/CANBUSIntroduccinalossistemasdecomunicacindelvehculo.pdf

Microchip. (2015). Recuperado el 18 de mayo de 2015, de http://www.microchip.com/pagehandler/en-us/technology/can/home.html

Microchip. (2015). *CAN bus Analyzer Tool*. Recuperado el 18 de mayo de 2015, de http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=APGDT002

Microchip Technology Inc. (2007). *Stand-Alone CAN controller with SPI interface.* Recuperado el 20 de abril de 2015, de http://ww1.microchip.com/downloads/en/DeviceDoc/21801e.pdf

Microchip Technology Inc. (2010). *High speed CAN transceiver MCP2551*. Recuperado el 20 de abril de 2015, de http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf

National Instruments. (2 de febrero de 2011). *Introducción al CAN*. Recuperado el 16 de mayo de 2015, de www.ni.com/white-paper/2732/es/

Pacheco Hernández, J. E. (2011). *Monitoreo de hábitos de manejo por medio de una red CAN.* Puebla, México: Universidad de las Américas Puebla.

Pazul, K., & Microchip. (1999). *Controller Area Network (CAN) Basics.* Recuperado el 10 de febrero de 2014, de http://ww1.microchip.com/downloads/en/AppNotes/00713a.pdf

Peak System. (2015). Recuperado el 18 de mayo de 2015, de http://www.peak-system.com/PCAN-Explorer-5.249.0.html?&L=1

Petroutsos, E. (2010). *Mastering Microsoft Visual Basic 2010.* Indianápolis, USA.: Wiley Publishing Inc.

Pico Technology. (2015). *Pico scope Serie 5000*. Recuperado el 14 de mayo de 2015, de https://www.picotech.com/download/datasheets/PicoScope5000Series-es.pdf

Reyes Cortés, F., & Cid Monjaraz, J. (2015). *Arduino, aplicaciones en robótica, mecatrónica e ingenierías.* México: Alfaomega.

Richards, P., & Microchip. (2002). *A CAN physical layer discussion.* Recuperado el 10 de Febrero de 2014, de http://ww1.microchip.com/downloads/en/AppNotes/00228a.pdf

Sacón Chango, G. J., & Villalva Taipe, D. F. (2013). *Diseño e implementación de un prototipo de red industrial basado en el estándar Asi (Actuador Sensor Interface) para el sistema de mezclado de líquidos.* Ecuador: Escuela Superior Politécnica de Chimborazo.

Tindell, K., Burns, A., & Wellings, A. (1995). *Calculating controller area network (CAN) message response times.* University of York.

Vector. (2015). *CANalyzer 8.5*. Recuperado el 20 de mayo de 2015, de http://vector.com/vi_canalyzer_en.html

Vidal, J., & Zúñiga, M. (2005). *Implementación de una red industrial CAN para un sistema SCADA.* Colombia: Universidad del Cauca.

Yunta, M. Á. (2010). *Implementación de las comunicaciones PC-Autómata-Robot mediante interfaz ethernet industrial.* Madrid: Universidad Carlos III.