SOFTWARE ANALIZADOR DE CAN
PARA AVT-852

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN

SISTEMAS INTELIGENTES MULTIMEDIA

PRESENTA

L.I. J. GUADALUPE ESCOTO HERNANDEZ

GUADALAJARA, JALISCO, ABRIL 2017

SOFTWARE ANALIZADOR DE CAN PARA AVT-852

Carta del jurado

Dedicatorias

A mis padres Carmen y Jesús

Por estar siempre a mi lado con su amor y cariño, por hacer de mí una persona de provecho para la sociedad y su ayuda incondicional en cualquier situación de la vida.

A mi esposa Gemma

Por apoyarme en todo este proceso, por animarme en los momentos que perdía la esperanza y ayudarme con su cariño y comprensión cuando más lo necesitaba.

A mi hija Estefanía

Por ser mi inspiración para ser mejor persona y salir delante de cualquier contratiempo de la vida, por su sonrisa y por su mirada que me hace sentir fuerte y feliz.

A mis hermanos Ramón, Manuel, Martha, Carmen, Teresa, Jaime, Jorge y Francisco

Por enseñarme cada uno a su modo a querer a mi familia y a ser mejor persona, por estar junto a mí en todas las etapas de mi vida.

Agradecimientos

A mi escuela CIATEQ

Por darme la oportunidad de tomar la maestría y crecer de manera profesional y personal.

A mis compañeros de clase

Por darme su amistad y su apoyo en los momentos más difíciles de este proceso.

A mis maestros

Por dejarme adquirir algo de su conocimiento y ayudarme a tener nuevas herramientas para luchar en la vida y ser mejor.

A la señora Claudia Del Toro

Por su forma de ser y su apoyo en todo tipo de situaciones a lo largo de toda la maestría. Por sus palabras y por su apoyo sincero con todos y cada uno de nosotros.

Índice general

Índice de figuras

Resumen

El protocolo de comunicación CAN se encuentra presente en varias industrias, desde la automotriz, hasta la aeroespacial. Es un protocolo que se utiliza para la interconexión de microcontroladores que forman una red tolerante a fallos para el envío y recepción de información a través del bus por parte de los nodos. Los sistemas para el análisis del bus son herramientas de vital importancia para verificar el correcto funcionamiento de los dispositivos que se encuentran en la red, así como para analizar que los mensajes o paquetes que viajan en el medio, sean los indicados en base a las especificaciones de dicha red.

Específicamente, en este trabajo se desarrolló una aplicación que permite analizar el bus de CAN por medio de la interfaz AVT-852, la cual, tiene una conexión USB y permite el envío y recepción de mensajes desde y hacia los nodos. Dicha herramienta permite visualizar el registro de los mensajes que pasan por el bus, ya sea de manera filtrada por identificador o en su totalidad. También nos da la capacidad de interactuar como un nodo más de la red al poder enviar mensajes por medio de los canales seleccionados. Otras de las características que contiene la aplicación son la visualización de señales específicas en forma de gráficos. Esto se logra utilizando el número de bit de inicio y el offset necesario para dicha señal. Por ejemplo, se pueden visualizar los mensajes individuales por medio del identificador o cualquiera de los valores de los bytes provenientes del bus.

Esta aplicación es una herramienta que se puede utilizar sin conocimientos técnicos de programación ni de configuración del dispositivo. Además, esta aplicación permite interactuar con los nodos de la red con solo seleccionar los valores y arrancar el registro del bus.

Abstract

CAN communication protocol is present in various industries from automotive to aerospace, it is a protocol used to interconnect microcontrollers forming a fault tolerant network for sending and receiving information through bus by nodes. Systems for analyzing the bus are tools of vital importance to verify the correct functioning of the devices on the network and to analyze the messages or packets traveling in the medium are the indicated based on the specifications of that network.

Specifically, in this assignment an application was developed that is able to analyze the CAN bus via the AVT-852 interface which has a USB connection and allows sending and receiving messages to and from the nodes. This tool lets you visualize the recording of messages passing through the bus either way filtered by identifier or in complete package; it also allows us the interaction as a network node to be able to send messages through selected channels. Other features that will contain the application are displaying specific signals in graphical form using the start bit number and offset required for the signal and display individual messages by ID or any of the values bytes from the bus.

The use of this application is intended as a tool that can be used without programming knowledge or configuration device to use, allowing interaction with network nodes by just selecting the values and start the bus recording.

Glosario

AVT. - Advanced Vehicle Technologies.

CAN. - Controller Area Network (Controlador de Area Local).

LIN. - Local Interconnect Network (Red De Interconexión Local).

USB. - Universal Serial Bus (Bus Universal En Serie).

RS232. - Recommended Standard 232 (Estándar Recomendado 232).

.NET. - Framework de Microsoft para el desarrollo de aplicaciones.

DLL. - Dynamic-Link library (Biblioteca de enlace dinámico)

FRAMEWORK. - Conjunto estandarizado de conceptos, prácticas y criterios.

OSI. - Open System Interconnection (Modelo de Interconexión de sistemas abiertos)

MOST. - Media Oriented Systems Transport (Transporte De Sistemas Orientados a Medios)

TCP-IP. - Protocolo de red para interconexión de sistemas computacionales.

CSV. - Comma Separated Values (Archivo de valores separados por comas)

CiA. - CAN in Automation.

CSMA/CD + AMP. – Acceso múltiple con escucha de portadora y detección de colisiones, y resolución de colisiones por arbitraje a nivel de bits.

TOKEN. - Paso de testigo.

NRZI. - No retorno a cero invertido.

EndPoint. – Punto terminal.

Pipe. – Tubería.

1 Introducción

El protocolo de comunicación CAN es uno de los más importantes en la industria automotriz. Este protocolo permite la interconexión de microcontroladores a lo largo de toda la carrocería del vehículo. De esta forma todos los componentes se encuentran interconectados para transferir información ya sea del estado de algún dispositivo o de alguna medición. Los equipos de prueba cuentan con, además de la circuitería para estimular entradas y leer salidas, interfaces de comunicación CAN para llevar a cabo el diagnóstico y el intercambio de mensajes con la unidad bajo prueba. Existen varias herramientas que le ayudan al ingeniero a analizar el tráfico de mensajes de CAN desde y hacia la unidad bajo prueba o para simular el comportamiento de algún dispositivo en la red. Entre estos equipos se encuentra la AVT-852 que es un dispositivo que permite enviar y recibir datos desde una red CAN, es desarrollada por la empresa Advanced Vehicle Technologies, Ésta es una herramienta muy potente, pero utilizada solo para el intercambio de mensajes no para el análisis del bus, muchas veces por el costo de desarrollo de un software analizador o por la complejidad de dicho equipo. Es por eso que desarrollara un software que permita utilizar las AVT-852 que se encuentran en el equipo de prueba para que no sea necesario instalar hardware adicional.

1.1 Antecedentes

En la actualidad los sistemas automotrices contienen un número elevado de microcontroladores interconectados para el envío de información. Esta interconexión se realiza por medio del protocolo CAN el cual funciona de manera similar a una red de computadoras. En el desarrollo de dichos equipos se necesita analizar el bus de CAN para verificar que cada uno de los elementos de la red funcione de manera correcta enviando los mensajes que les corresponden en el tiempo indicado. Ninguno de los equipos de prueba desarrollados por la empresa tienen un analizador de CAN integrados lo que hace que el ingeniero necesite transportar equipo adicional. Estos factores afectan el tiempo de prueba ya que dichas herramientas necesitan un conocimiento previo para su uso y puesta en marcha. Es por esto que la empresa decidió desarrollar un software analizador de CAN que pueda ser utilizado en los equipos existentes y en los nuevos sin la necesidad de dispositivos adicionales ya que se utilizarán los mismos equipos con los que cuentan los probadores actuales.

1.2 Definición del problema

Los ingenieros necesitan llevar a cabo análisis del bus de CAN en sitio, para esto es necesario contar con un equipo portátil, el cual debe de tener instalado un software analizador además del convertidor el cual necesita ser adquirido para dicha actividad.

1.3 Justificación

Los equipos de prueba que se desarrollan actualmente en nuestra empresa, utilizan las AVT-852 para llevar a cabo las tareas de envío y recepción de mensajes por el bus de CAN. Con el desarrollo de esta aplicación los equipos actuales tendrán un software analizador de CAN integrado. Esto sin la necesidad de agregar equipo adicional ni externo, ya que se podrá utilizar cualquiera de las AVT conectadas al equipo para registrar la información de bus.

1.4 Objetivos

1.4.1 Objetivo general:

Desarrollar un software para el análisis del bus de CAN utilizando la AVT-852 la cual utiliza una conexión USB por medio de la cual se alimenta la energía necesaria para su funcionamiento. Esta aplicación permitirá analizar en tiempo real el tráfico del bus de CAN, así como generar registros de dicha información, además permitirá al usuario interactuar con el bus por medio del envío de mensajes, comparación de paquetes y la posibilidad de graficar las señales que se encuentran en los bytes de los paquetes que provienen de la red.

1.4.2 Objetivos específicos:

- Desarrollar un software en vb.net que permita el análisis del Bus de CAN de manera fácil en intuitiva.
- Desarrollar una librería o dll para controlar la AVT desde el framework de net.
- Integrar el software analizador de CAN a los equipos de prueba ya existentes, reutilizando las AVT con las que cuentan dichos equipos.

1.5 Hipótesis

Se puede utilizar la AVT-852 como herramienta para análisis del bus de CAN y se puede agregar a los equipos de prueba nuevos o existentes sin la necesidad de agregar hardware adicional. Solo será necesario la instalación del software y la configuración adecuada.

2 Marco teórico

Los analizadores de bus son un conjunto de herramientas que nos ayudan a analizar los datos que transitan desde un dispositivo hacia otro. Las herramientas físicas se conectan en el medio de comunicación como si fueran un nodo más. Esto permite recibir los datos que viajan a través del medio físico de comunicación. Este medio puede ser un cable o el aire en caso de comunicaciones inalámbricas. Estos datos son transferidos desde el hardware de análisis hasta un software analizador instalado en un equipo de cómputo. Este software convierte los datos en información relevante que sirve a los usuarios para el estudio de la comunicación entre nodos.

Existen varios tipos de analizadores, dependiendo del medio de comunicación que se quiera verificar. Las interfaces más comunes en las computadoras de escritorio son el USB para la transferencia de información de alta velocidad; el RS232, el cual sirve para la instrumentación por medio de comandos; el Ethernet, el cual permite la interconexión de varias computadoras, por medio de una red. Siendo este último el protocolo que nos facilita el envío y recepción de la información y recursos entre todos los nodos conectados a la red.

2.1 CAN

El protocolo CAN fue desarrollado por la empresa BOSCH. Es un sistema de transmisión de mensajes multi maestro, el cual especifica una velocidad máxima de trasmisión de 1 megabit por segundo. A diferencia de las redes tradicionales como la de USB y Ethernet, CAN no envía grandes bloques de datos de punto a punto desde un nodo A hacia un nodo B bajo la supervisión del maestro central. En una red de CAN, se transmiten pequeños mensajes como la temperatura o las revoluciones por minuto a toda la red, lo que nos permite contar con una consistencia de datos en cada uno de los nodos del sistema.

El protocolo CAN está especificado en un estándar internacional ISO, el cual lo define como una comunicación serial. Originalmente desarrollado para la industria automotriz y así suplir el cableado complejo de los arneses internos por un bus de dos hilos. La especificación indica que el protocolo tiene una alta inmunidad a la interferencia eléctrica o ruido y la habilidad de autodiagnóstico y reparación de errores en los datos. Esas características han aumentado la popularidad del protocolo CAN en otras áreas

como son: la automatización de edificios, el área médica, así como el área de manufactura.

El estándar internacional ISO-11898: 2003 (CORRIGAN, 2008) sobre el protocolo de comunicación CAN, describe como se pasa la información entre dispositivos de la red y conforma el modelo de interconexión de sistemas abiertos (OSI) el cual está desarrollado en términos de capas. Las comunicaciones entre dispositivos conectados por el medio físico están definidas en la capa física del modelo. La arquitectura del ISO11898 define las dos capas más bajas de las siete del modelo OSI/ISO como capa de enlace de datos y la capa física.

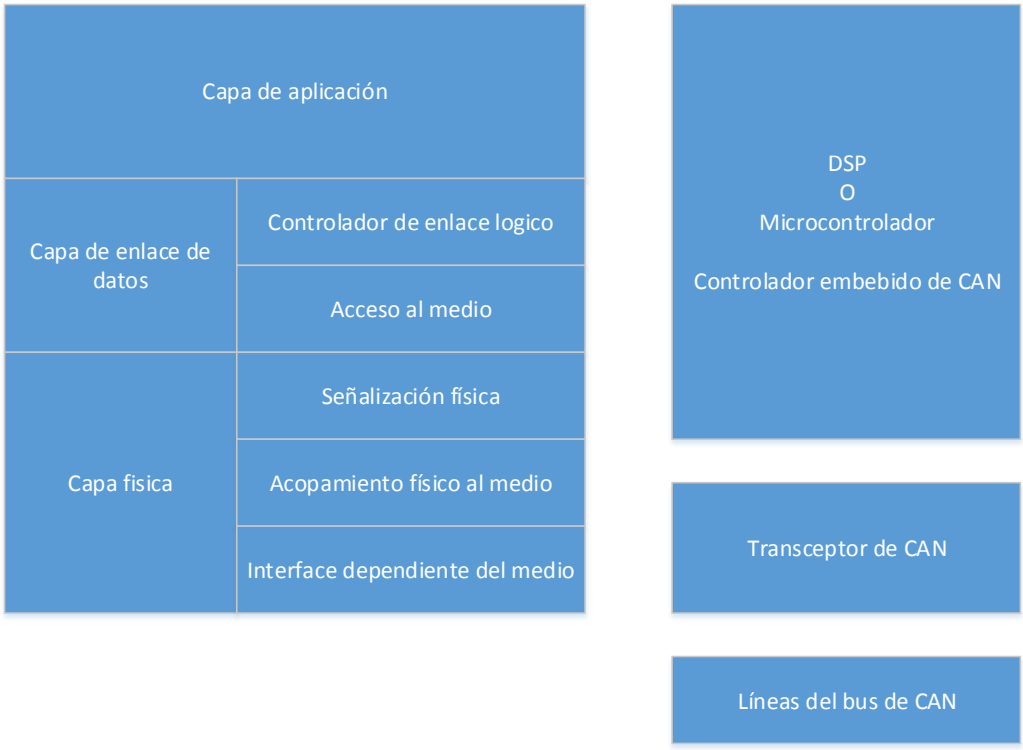


Figura1: La arquitectura en capas del estándar ISO 11898

Como se muestra en la figura, la capa de aplicación establece el enlace de la comunicación hacia un nivel de protocolo de aplicación independiente como puede ser CANopen. Este protocolo esta soportado por el grupo internacional de fabricantes y usuarios, Automatización en CAN (CiA). Información adicional de CAN se puede encontrar en la página web del CiA.

2.1.1 CAN estándar o extendido

CAN es un protocolo con acceso múltiple con escucha de portadora y detección de colisiones y arbitraje en la prioridad del mensaje (CSMA/CD + AMP). CSMA significa que cada nodo en el bus debe de esperar por un periodo de tiempo prescrito de inactividad antes de intentar enviar un mensaje. CD+AMP significa que las colisiones son resueltas por medio de un arbitraje a nivel de bits, basado en una prioridad pre programada de cada mensaje en el campo de identificación de un mensaje. El identificador con prioridad más alta siempre gana el acceso al bus. Esto es, el último valor lógico en alto en el identificador sigue transmitiendo porque tiene la prioridad más alta. Ya que cada nodo en la red forma parte de la escritura de cada bit, un nodo árbitro sabe si se envió el bit alto al bus.

El estándar ISO-11898:2003, sobre el identificador de 11 bits (ETSCHBERGER, 2008), nos indica que las velocidades de transmisión van desde los 125 kbps hasta 1Mbps. El estándar fue corregido con el identificador extendido de 29 bits. El campo del identificador estándar de 11 bits provee una combinación de 2^{11} o 2018 deferentes identificadores de mensaje, mientras que el identificador extendido de 29 bits provee 2^{29} o 537 millones de identificadores.

2.1.2 Los campos de bits del CAN Estándar

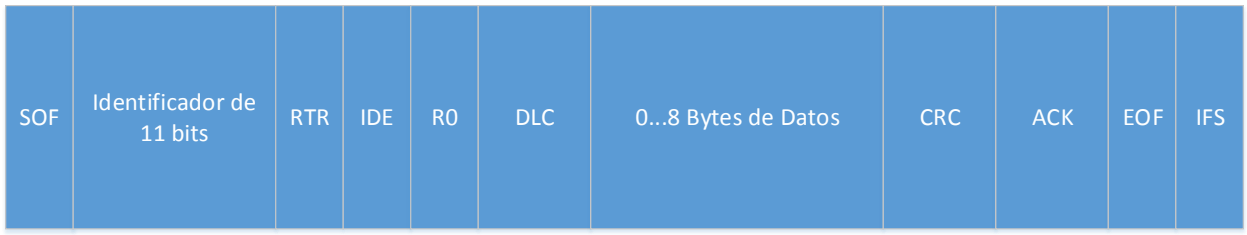


Figura 2: Campos de datos del CAN estándar.

El significado de cada uno de los campos de bits es el siguiente:

- **SOF.-** El bit dominante de inicio de frame, el bit SOF marca el inicio de un mensaje, y es utilizado para sincronizar los nodos en la red después de estar libre.
- **Identificador.** - El identificador estándar de CAN de 11 bits establece la prioridad del mensaje. Entre más bajo sea el valor binario más alta es la prioridad.
- **RTR.** - el bit de solicitud de transmisión remota única (RTR) es dominante cuando la información es requerida por otro nodo. Todos los nodos reciben la petición, pero el identificador determina al nodo específico. La información de respuesta también es recibida por todos los nodos y utilizada por el nodo interesado. De esta forma toda la información utilizada en un sistema es uniforme.
- **IDE. -** El bit dominante de extensión simple del identificador (IDE) indica que un identificador estándar sin extensión está siendo transmitido.
- **R0.-** Bit reservado (para posibles usos en el futuro en caso de enmiendas en el protocolo)
- **DLC. -** Los 4 bits del código de ancho de datos(DLC) contiene el número de bytes que están siendo transmitidos.
- **Data. -** Pueden ser enviados hasta 64 bits de datos de aplicación.
- **CRC. -** Los 16 bits (15 más un delimitador) del chequeo de redundancia cíclica(CRC) contienen la suma de comprobación (número de bits transmitidos) de la aplicación pasada para la detección de errores.
- **ACK. -** Cada nodo que recibe un mensaje preciso sobrescribe el bit recesivo del mensaje original con un bit dominante, indicando que un mensaje libre de error ha sido enviado. Cuando un nodo receptor detecta un error deja el bit en valor recesivo, descarta el mensaje y el nodo que envía repite el mensaje después del

arbitraje. De esta forma, cada nodo reconoce la integridad de los datos. El ACK es de 2 bits uno es el bit de reconocimiento y el otro es un delimitador.

- **EOF. -** Este campo de 7 bits de fin de trama, indica el final de un mensaje de CAN y deshabilita el relleno de bits, indicando un error de relleno de bits cuando este se encuentra en valor dominante. Cuando ocurre que en una sucesión de bits llegan 5 con el mismo valor lógico durante una operación normal, un bit del nivel lógico opuesto es introducido en los datos.
- **IFS. -** Este espacio inter trama de 7 bits, contiene el tiempo requerido por el controlador para mover de manera correcta la trama recibida a su correcta posición en el área de buffer de mensaje.

2.1.3 Los campos de bits del CAN Extendido

Figura 3: Campos de datos del CAN extendido.

Los mensajes de CAN extendido son lo mismo de los mensajes estándar más la adición de los siguientes campos de bits:

- **SRR. -** el bit de sustituto de petición remota (SRR) remplaza al bit RTR en los mensajes estándar.
- **IDE-A.-** Un bit recesivo en la extensión del identificador, indica que vienen más bits del identificador, los 18 bits que vienen después del campo IDE.
- **R1.**- Un bit adicional ha sido incluido para futuras adecuaciones del estándar.

2.1.4 Arbitraje

Una de las características fundamentales de CAN es el estado lógico opuesto entre las líneas del bus, y la entrada del controlador y la salida del receptor. Normalmente, un valor lógico en alto está asociado con un uno y un valor lógico bajo está asociado con un cero, pero no es así con el bus de CAN.

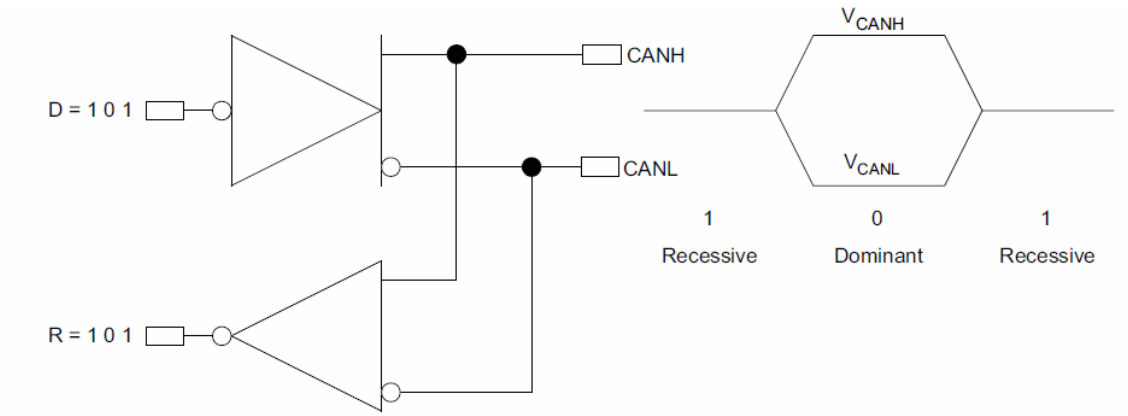


Figura 4: La lógica invertida del bus de CAN.

El acceso al bus es manejado por eventos y se lleva a cabo de manera aleatoria. Si dos nodos tratan de ocupar el bus de manera simultánea, el acceso es implementado con un arbitraje a nivel de bits no destructivo. No destructivo significa que el nodo que gana el arbitraje continúa enviando su mensaje, sin que este sea destruido ni corrompido por otro nodo.

El alojamiento de la prioridad de los mensajes en el identificador es una característica de CAN que lo hace especialmente atractivo para su uso en ambientes de control de tiempo real. Entre más bajo el valor binario del identificador del mensaje más alta es su prioridad. Un identificador que consiste enteramente de ceros es el mensaje con mayor prioridad en la red ya que mantiene el bus en modo dominante la mayor parte del tiempo. Por lo tanto, si dos nodos comienzan a transmitir de manera simultánea, el nodo que envía un último bit del identificador como cero (dominante) mientras que el otro nodo envía un uno (Recesivo) retiene el control del bus de CAN y sigue enviando hasta completar su mensaje. Un bit dominante siempre sobrescribe a un bit recesivo en el bus de CAN.

2.2 USB

USB son las siglas de Universal Serial Bus (Bus universal en serie). Es un bus estándar industrial que define los componentes físicos y los protocolos para la conexión entre la computadora y uno o más dispositivos (AXELSON J. , 2015).

El host es una PC o cualquier otro tipo de computadora que contenga un controlador USB y una concentradora raíz. Esos componentes trabajan juntos para permitir al sistema operativo la comunicación con los dispositivos en el bus. El controlador del host formatea los datos para transmitirlos al bus y traducen los datos recibidos a un formato que los componentes del sistema operativo puedan comprender. El controlador del host también lleva a cabo otras funciones referentes a la administración de las comunicaciones en el bus. La concentradora raíz, en combinación con el controlador del host, detecta los dispositivos anexados o removidos, controla las peticiones del controlador del host y pasa la información entre los dispositivos y el controlador.
Los dispositivos son los periféricos y concentradores adicionales que se conectan al bus. Un concentrador tiene uno o más puertos para la conexión de los dispositivos. Cada dispositivo debe de contener la circuitería y código para saber cómo comunicarse con el host. Algunas de las características del bus USB son las siguientes:

- Banda de paso, disponibilidad desde algunos kilobits a varios megabits.
- Transferencia síncrona y asíncrona en el mismo bus.
- Varios tipos de periféricos en el mismo bus.
- Posibilidad de conectar hasta 127 periféricos.
- Tiempo de respuesta garantizado.
- Flexibilidad a nivel banda de paso.
- Fiabilidad y control de errores.
- Perfectamente integrado en la PC, plug and play (conectar y usar).
- Coste reducido en la versión de baja velocidad.
- Posible expansión del bus.

2.2.1 Topología

La topología, o el arreglo de conexiones, en el bus es de estrella escalonada. Al centro de la estrella se encuentra un concentrador. Cada punto en la estrella es un dispositivo que se conecta a un puerto en el concentrador. El número de puntos en cada estrella puede variar de dos, cuatro, o siete puertos. Cuando existen múltiples concentradores en serie, se puede pensar en ellos como si fueran una capa, o serial, uno después del otro.

La estrella escalonada solo describe las conexiones físicas. En la programación, todo lo que importa es la conexión lógica. Para comunicarse, el host y el dispositivo no necesitan saber cuántos concentradores tiene que atravesar la información.

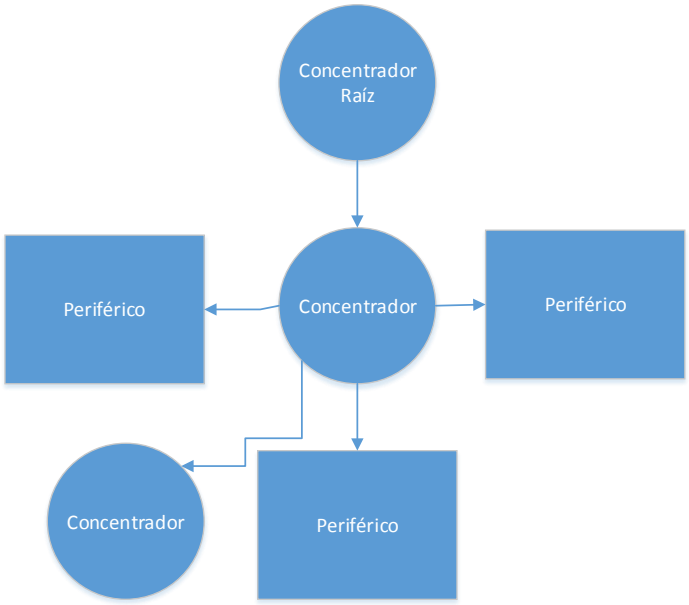


Figura 5: Topología de estrella del protocolo USB

2.2.2 Interfaz física

A nivel eléctrico, el cable USB transfiere la señal y la alimentación sobre cuatro hilos.

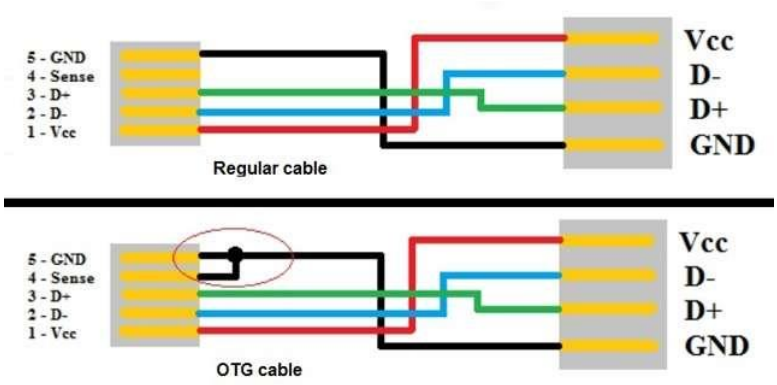


Figura 6: Distribución de señales en cables USB

A nivel de alimentación, el cable proporciona la tensión nominal de 5V. Es necesario definir correctamente el diámetro del hilo con el fin de que no se produzca una caída de tensión demasiado importante en el cable. Una resistencia de terminación instalada en la línea de datos permite detectar el puerto y conocer su configuración.

A nivel de señal se trata de un par trenzado con una impedancia característica de 90 Ohms. La velocidad puede ser tanto de 12 Mbits/s como de 1.5 Mbits/s. La sensibilidad del receptor puede ser de, al menos, 20 mV y debe de poder admitir un buen factor de rechazo de tensión en modo común. El reloj se transmite en el flujo de datos, la codificación es de tipo NRZI, existiendo un dispositivo que genera un bit de relleno que garantiza que la frecuencia de reloj permanezca constante. Cada paquete va precedido por un campo de sincronismo.

En la parte de consumo, cada sección puede proporcionar una determinada potencia máxima siendo el PC el encargado de suministrar la energía. La PC gestiona el consumo, teniendo la capacidad de poner en reposo o en marcha un periférico USB. En reposo, este reduce su consumo (si el dispositivo lo permite), quedándose la parte USB funcional. Esta gestión está orientada especialmente a los equipos portátiles.

2.2.3 Componentes en la comunicación USB

Host: Dispositivo maestro que inicia la comunicación (Generalmente, el equipo de cómputo).

Hub: Dispositivo que contiene uno o más conectores o conexiones internas hacia otros dispositivos USB, el cual habilita la comunicación entre el host y los diversos dispositivos. Cada conector representa un puerto.

Dispositivo Compuesto: Es aquel dispositivo con múltiples interfaces independientes. Cada una tiene una dirección sobre el bus, pero cada interface puede tener un diferente driver en el host.

Puerto USB: Cada host soporta varios buses, cada conector en el bus representa un puerto USB por lo tanto sobre el bus puede haber uno o varios conectores, pero solo existe una ruta en cada bus y solo un dispositivo de ese bus puede transmitir información a un tiempo.

Driver: Es un programa que habilita a las aplicaciones para poder comunicarse con el dispositivo. Cada dispositivo sobre el bus debe de tener un controlador, algunos periféricos utilizan el controlador instalado en el sistema operativo.

Puntos Terminales: Es una localidad especifica dentro del dispositivo. El punto terminal es un buffer que almacena múltiples bytes, típicamente es un bloque de memoria de datos o un registro dentro del microcontrolador. Todos los dispositivos deben de soportar en punto 0. Este punto terminal es el que recibe todo el control y las peticiones de estado durante la enumeración cuando el dispositivo se encuentra en el bus.

Tuberías: Es un enlace virtual entre el host y el dispositivo USB, este enlace configura los parámetros asociados con el ancho de banda, que tipo de transferencia se va a utilizar, dirección del flujo de datos y el tamaño máximo y mínimo de los paquetes.

Cada enlace está caracterizado por su banda de paso, su tipo de servicio, el número de punto terminal y el tamaño de los paquetes.

Estos enlaces se definen y crean durante la inicialización del USB. Siempre existe un enlace virtual 0 que permite tener acceso a la información de configuración del periférico. Existen dos tipos de tuberías definidas en el estándar de USB, las de flujo y de mensaje. Las primeras se tratan de un flujo sin formato USB definido, esto significa que por medio de esta tubería se puede enviar cualquier tipo de dato y soporta las transferencias bulk, isócronas e interrupt. La segunda ellas es un tipo de enlace virtual el cual si tiene un formato de datos definido y solo soporta la transferencia Control.

2.2.4 Tipos de transferencia de datos

El enlace virtual o pie puede ser de cuatro posibles tipos:

Control: Modo utilizado para realizar configuraciones. Existe siempre sobre el punto terminal 0. Todos los dispositivos USB deben de soportar este tipo de transferencia. Los datos de control sirven para configurar al periférico en el momento de conectarse al bus. Algunos controladores específicos pueden utilizar este enlace para transmitir su propia información de control. Este enlace no tiene perdida de dato, puesto que los dispositivos de detección de recuperación de errores están activos a nivel de protocolo.

Bulk: Este modo se utiliza para la transmisión de importantes cantidades de información. Como el tipo de control, este enlace no tiene pérdida de datos. Este tipo de transferencia es útil cuando el tiempo de transferencia no es crítico, como, por ejemplo, el envío de un archivo a la impresora o la recepción de datos desde un escáner. En estas aplicaciones, la transferencia es rápida, pero puede esperar si fuera necesario. Solo los dispositivos de media y alta velocidad utilizan este tipo de transferencia.

Interrupt: Modo utilizado para transmisiones de pequeños paquetes, rápidos, orientados a la interacción con el humano (ratón, punteros, teclados).

Este tipo de transferencia fue diseñado para dispositivos que deben de recibir atención periódicamente lo utilizan los dispositivos de baja velocidad.

Este estilo de transmisión garantiza la transferencia de pequeñas cantidades de datos. El tiempo de respuesta no puede ser inferior al valor especificado por la interfaz.

Isócronas o flujo en tiempo real: Modo utilizado para la transmisión de audio y video comprimido. Este tipo de transmisión funciona en tiempo real y es el de mayor prioridad.

2.2.5 Enumeraciones

Cuando se conecta un dispositivo USB a la PC se produce el proceso de enumeración, el cual consiste en una comunicación bilateral para que el dispositivo se presente y le diga al host cuáles son sus parámetros. Estos son algunos de los parámetros:

- Consumo de energía expresada en unidades de carga.
- Número y tipo de puntos terminales.
- Clase del producto.
- Tipo de transferencia, etc.

El proceso de enumeración es iniciado por el host cuando detecta que un nuevo dispositivo ha sido conectado al bus. El host le asigna una dirección al dispositivo y habilita su configuración permitiendo la transferencia de datos sobre el bus.

2.3 AVT

La AVT es una interfaz de comunicación de múltiples interfaces, entre las que se encuentran: CAN de alta y baja velocidad, LIN, J1850 y K-LINE. Este dispositivo es desarrollado por la empresa americana avt-hq, la cual se encuentra en la ciudad de Davidsonville en el estado de Maryland.

2.3.1 Conexión

La interfaz AVT-853 cuenta con una conexión ethernet la cual permite conectar varios dispositivos a una red TCP/IP, en este modelo el envío y recepción de los datos se lleva a cabo abriendo un flujo por medio de la dirección ip y el puerto, a diferencia del modelo AVT-852 el cual es la base de esta investigación, la cual se conecta a la computadora por medio de un puerto USB. Cuenta en su interior con un convertidor USB-SERIAL es cual es provisto por la empresa FTDI chips (RILEY, 2017). Una vez que el equipo se encuentra conectado y se instala el controlador adecuado, podemos conectarnos por medio de un puerto serial y una velocidad de transmisión específica. Del lado de la red de la comunicación, cuenta con un conector DB-15 el cual divide todos los posibles protocolos en un mismo conector.

Pin #	Description	Notes
1	CAN4_SWC	Bi-directional (only when CAN4 is configured as Single Wire CAN) [This signal goes through JP1]
2	J1850 VPW bus +	Bi-directional [This signal goes through JP3]
3	CAN4_H	Bi-directional (only when CAN4 is configured as 2-wire CAN)
4	Ground	pins #4 and #5 are connected together internally
5	Ground	pins #4 and #5 are connected together internally
6	CAN0_H	Bi-directional
7	K-Line LIN (channel 5) KWP (channel 6)	Bi-directional [This signal goes through JP2]
11	CAN4_L	Bi-directional (only when CAN4 is configured as 2-wire CAN)
13	V-Batt supply	Sourced by external equipment (vehicle)
14	CAN0_L	Bi-directional

Figura 7: Relación de pines del conector DB-15 de la AVT-852

2.3.2 Generalidades

La AVT-852 cuenta con dos puertos de CAN, siendo el CAN0 de alta velocidad de doble hilo y el CAN4 que nos permite configurar el puerto ya sea como alta velocidad igual que el puerto 0 o como baja velocidad de un solo hilo por medio de software. Este modo de funcionamiento del puerto 4 nos permite utilizar el equipo en unidades del fabricante General Motors el cual presenta un bus de baja velocidad para el diagnóstico de la unidad bajo prueba. Además, nos permite configurar hasta 32 mensajes periódicos los cuales serán controlados por hardware. Gracias a esto se puede disminuir la carga de trabajo en el PC ya que la AVT se encarga de configurarlos, iniciarlos detenerlo o borrarlos con los comandos que nos ofrece. Cada uno de los mensajes periódicos es independiente, de esta forma podemos manipular cada uno de ellos sin afectar al resto.

2.3.3 Comunicación

La AVT-852 cuenta con dos tipos de comandos para su funcionamiento, los primeros son los de configuración. Estos nos permiten inicializar la interfaz y seleccionar el funcionamiento de cada uno de sus puertos, así como sus velocidades, filtro y máscaras para la comunicación CAN. Los segundos son los que nos permiten el envío de paquetes de información a la red de CAN en la cual se encuentra conectada. Lo primero que se debe hacer para poder enviar cualquiera de los dos tipos de comando, es formar el paquete basándonos en el manual de usuario, el cual muestra comandos formados en bytes hexadecimales. Dichos bytes nos permiten seleccionar cual parte queremos configurar. Una vez que tenemos el mensaje formado, es necesario convertir la cadena de bytes hexadecimales a su respectivo arreglo de bytes con el valor en decimal. De esta forma se envía el arreglo completo para que el dispositivo lo decodifique y realice la operación seleccionada. Cuando un comando es ejecutado correctamente se nos devuelve el mismo comando, pero con el primer nibble en valor de 8. En caso de error la AVT nos informará en que byte existe el error, además de un código extra el cual nos permite identificar por medio del manual cual es el error en el comando. Cuando tenemos la respuesta de la AVT el formato viene en valores decimales, el nibble más bajo del primer byte nos indica cuantos bytes forman el paquete, de esta forma podemos extraerlos sin modificar los paquetes posteriores. Si se desea mostrar la información recibida en formato hexadecimal es necesario aplicar un algoritmo para convertir cada posición del arreglo de bytes a su equivalente hexadecimal.

2.4 Software Analizador

El software analizador es el encargado de establecer la comunicación con el dispositivo de CAN para el envío y recepción de los datos. Una vez que el sistema tiene la información recibida desde la red, le aplica un formato para convertir los datos en información relevante, mostrando información importante acerca de cada uno de los mensajes, así como de los canales con los que cuenta el dispositivo. Esta información puede variar entre sistemas, pero la mayoría cuenta con los siguientes campos para la información de los mensajes:

- Estampa de tiempo en la que se envió o recibió el paquete.
- Dirección del paquete puede ser de envío o recepción.
- Código de ancho de datos, el cual indica de cuantos bytes lleva el formato el paquete.
- Identificador del paquete enviado o recibido.
- Paquete de bytes del mensaje de CAN.

Además de la información de los paquetes, el software analizador nos muestra información sobre los canales de CAN, al igual que la información de los mensajes, la información de los canales de CAN puede variar de un sistema a otro, pero la mayoría cuenta con la siguiente información:

- Estado del canal, inicializado, detenido o en error.
- Numero de bytes total en el canal.
- Bytes por segundo en cada canal.
- Paquetes de error totales por canal.
- Paquetes de error por segundo en cada Canal.

A continuación, se muestran algunos de los sistemas que se encuentran actualmente en el mercado y una breve descripción de cada uno de ellos, así como de las empresas que los fabrican.

2.4.1 Productos en el mercado

CanAnalyzer: Es una herramienta de software para el análisis de la empresa Vector Informatik GmbH. Este software es muy utilizado a nivel global, sobre todo por las empresas automotrices y los proveedores de unidades de control, para analizar el tráfico en un sistema de bus serial.

La primera versión de Canalyzer salió al mercado en 1992, la primera herramienta de software para CAN a nivel mundial. Canalyzer ha sido actualizado continuamente desde entonces, y hoy por hoy es considerada la herramienta líder a nivel mundial para el análisis de redes de CAN. Además de su primer campo de acción, el cual es las redes electrónicas en los vehículos, también es utilizado por muchas otras industrias como: la transportación, vehículos pesado, aeroespacial y médica. La última versión de Canalyzer es la 9.0, revisado el día 26 de enero de 2017 (VECTOR, 2015).

Can Bus Analyzer: Es una herramienta para el análisis del bus de CAN, desarrollada por la empresa Microchip. Es un monitor del bus de CAN sencillo y de bajo costo el cual puede ser utilizado para depurar redes de alta velocidad. Esta herramienta soporta CAN 2.0b y el ISO11898-2. Cuenta con amplio rango de funciones las cuales permiten ser reusadas en diferentes segmentos de mercado incluyendo el automotriz, industrial, médico y marítimo. La herramienta se compone de una interfaz gráfica la cual permite analizar e interpretar de manera rápida el tráfico en el bus. Esta herramienta solo funciona con la interfaz de hardware del mismo nombre que también es desarrollada por la empresa. La última versión del software es la 2.3, revisado el 26 de enero del 2017 (Inc., 2011).

3 PROCEDIMIENTO

3.1 Requerimientos

En base a la utilización de los equipos de prueba por parte de nuestros clientes, así como el análisis de la retroalimentación obtenida en el uso de nuestros sistemas de prueba, se llega a la conclusión de desarrollar una aplicación que permita reutilizar las interfaces de CAN con los que ya se cuentan para disminuir costos y tener una herramienta que permita el análisis del bus de sin la necesidad de adquirir equipo nuevo. Así se definió que el sistema tendría que ser capaz de manipular y configurar cualquiera de las AVT-852 con las que cuenta el equipo, los cuales en algunos casos llegan a ser hasta 18 por probador. Otra parte importante es la interacción con el bus por medio del envío y recepción de paquetes desde la red de CAN. Una vez que se lean los mensajes, la ventana de registro debe mostrar las estampas de tiempo, así como los identificadores y los datos de cada uno de ellos. Se deben almacenar la información en un archivo de texto con extensión asc que sea compatible con las herramientas de análisis de cada uno de los ingenieros. De esta manera se llegó a la idea de desarrollar la aplicación LightNv la cual permita el uso de cualquiera de los 18 dispositivos AVT con los que cuenta cada uno de los equipos de prueba desarrollados por la empresa y de esta manera darles un valor agregado a las aplicaciones ya existente. Los requisitos principales que se definieron en esta etapa son los siguientes:

- La aplicación debe de ser capaz de utilizar cualquier AVT con la que cuente el equipo de prueba, siendo en una primera versión las del modelo 852 y en versiones posteriores permitir la actualización para utilizar las del modelo 853 que cuentan con una conexión Ethernet y son controladas por medio del protocolo TCP-IP.
- La aplicación debe de permitir el envío de mensajes hacia la red de CAN en la cual se encuentra conectado, permitiendo la configuración de dichos mensajes en su identificador, canal, código de ancho de datos y la información en si de una manera sencilla utilizando una tabla estilo Excel para el llenado de la información.
- La aplicación debe de permitir la recepción de mensajes en cualquiera de sus dos canales y mostrar la información de estos en una ventana que contenga una vista de lista, en la cual, se muestra la marca de tiempo en la que el

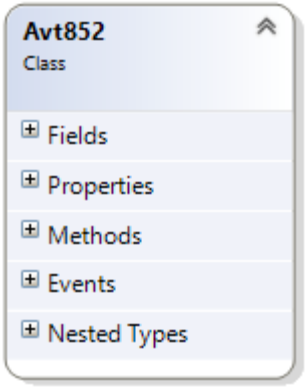
mensaje llego desde la red, el identificador, el flujo de los datos su código de ancho de datos y la información en sí de los bytes del mensaje.

- La aplicación debe de permitir el análisis de paquetes por medio del llenado de una tabla por parte del usuario, en la cual se indique cual es la información que se espera de cada paquete. Una vez que la aplicación recibe datos desde la red, estos son comparados por medio del identificador o por medio de cualquiera de los bytes de datos, mostrando así la cantidad de paquetes que han llegado a la computadora y que coinciden con los que el usuario definió en la tabla de configuración.
- La aplicación debe graficar las señales definidas por el usuario en una tabla por medio del identificador de cada señal, así como su bit de inicio y la cantidad de bits que conforman dicha señal. Debe ofrecer la selección del color de cada señal y permitir al usuario nombrarlas de una forma más comprensible.
- La aplicación debe de mostrar información acerca de los canales de CAN e información relevante acerca de la herramienta, si ésta se encuentra conectada, si fue inicializada correctamente o si existió algún error en la inicialización.
- La aplicación debe de permitir la configuración de la herramienta de CAN de manera sencilla e intuitiva mostrando el dispositivo que se encuentra conectado, así como su dirección, modelo y versión del firmware con el que cuenta.

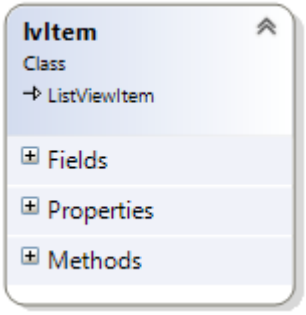
3.2 Diseño

La primera parte que se diseñó fue la librería para el manejo de la AVT, la cual puede ser reutilizada en cualquier tipo de proyecto creado en la plataforma .net ya que al ser compilada en este ambiente de desarrollo se genera una dll que puede ser referenciada y agregada a nuevos proyectos para su utilización. Esta librería nos permite comunicarnos con el dispositivo conectado para interactuar con la red de CAN por medio del envío y recepción de paquetes de información. A continuación, se muestran los diagramas de clase y una descripción de cada una de ellos.

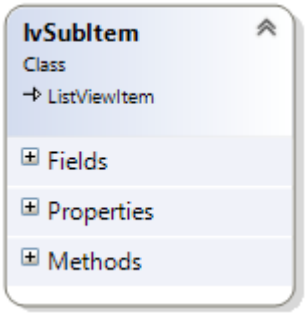
3.2.1 Diagramas de Clase



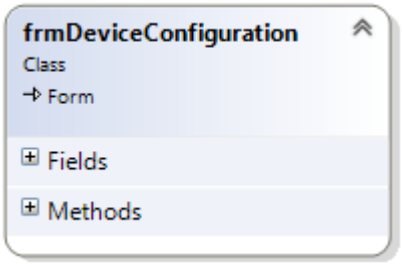
Clase de la AVT la cual permitirá la conexión y configuración con el dispositivo seleccionado por medio de un puerto y una velocidad de transmisión. Cuando la librería se conecte nos permitirá el envío y recepción de los paquetes de CAN.



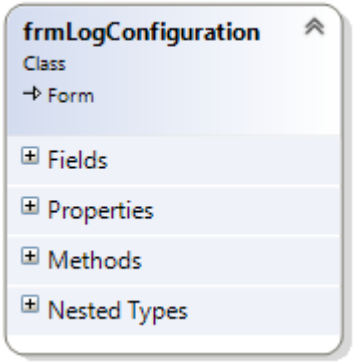
Clase que heredara desde system.windows.forms.listviewitem que nos permitirá agregar un elemento a la vista de lista, pero asignándole un icono dependiendo del tiempo de mensaje, además almacenara la estampa de tiempo y los datos del mensaje.



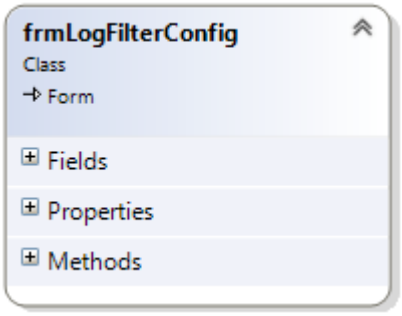
Sub elemento de la clase lvItem, para almacenar los valores de las columnas que conforman al elemento de la vista de lista.



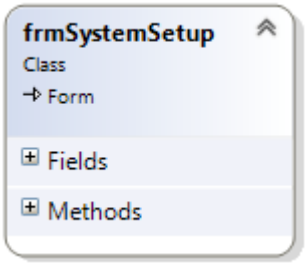
Clase del formulario que nos permitirá la configuración de los dispositivos conectados a nuestra computadora. Tanto el puerto serial y su velocidad, así como la configuración de la AVT y la velocidad de CAN. Además, se puede configurar los identificadores, filtros y máscaras de cada canal de CAN.



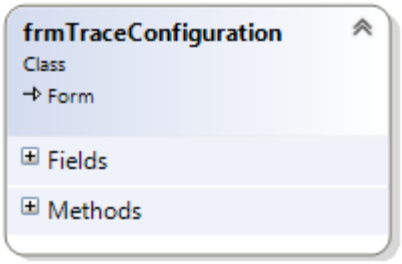
Clase para el formulario en el cual se configurarán los archivos de registro de los mensajes de CAN. En ella podremos configurar el formato del archivo, así como los valores de tiempo y rutas de almacenaje.



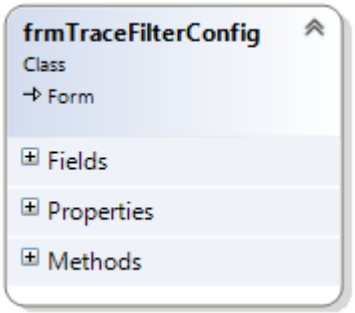
Clase del formulario en el cual se configurarán los filtros de paso y de bloqueo de los mensajes que se agregarán al archivo de registro.



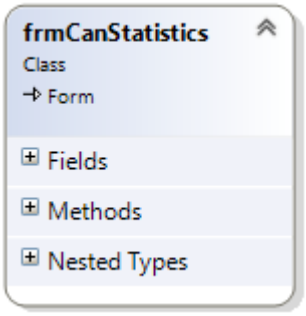
Clase para el formulario en el cual se configurarán todos los módulos que contiene el sistema, así como cada una de las ventanas que lo conforman.



Esta clase de formulario, nos permitirá la configuración de la ventana de registro, los parámetros de visualización, así como el orden de los mismos.

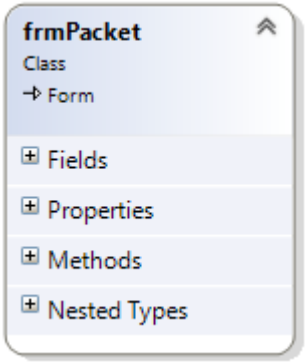


Formulario para configurar los filtros de la ventana de configuración del registro en ella se configurarán los filtros de paso y de bloque.

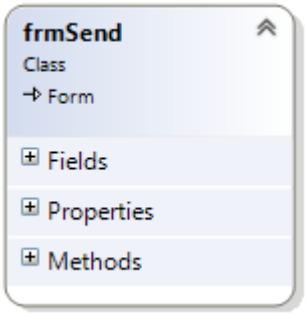


Clase del formulario que nos permitirá visualizar el estado de cada uno de los canales de CAN de la AVT conectada al sistema.

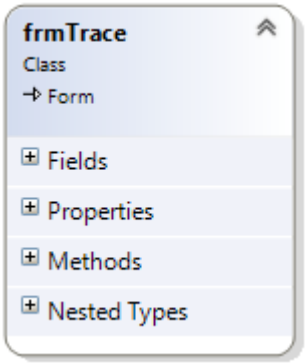
Ventana que nos permitirá visualizar los valores de los paquetes de CAN de una forma gráfica, por colores y divididas por tiempo.



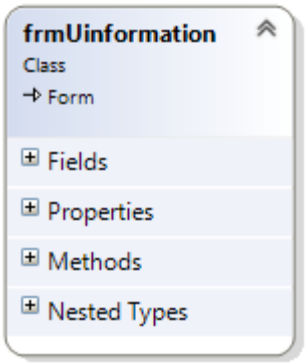
Ventana en la cual se configurarán los paquetes que se desean analizar provenientes de la red de CAN, en ella podremos seleccionar los identificadores y los bytes que se quieren revisar en cada paquete.



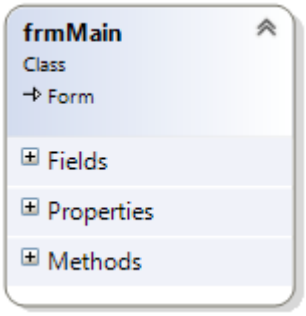
Esta clase permitirá la interacción con la red de CAN por medio del envío de mensajes, ya sean por evento o de manera periódica.



Ventana en la cual se mostrarán los paquetes enviados o recibidos desde la red, en ella se visualizará información relevante de cada mensaje y nos permitirá configurar la forma de visualización de los mismos.



Ventana que indicara al usuario el estado del sistema, así como los tiempos de ejecución y el estado del dispositivo conectado.



Formulario principal el cual servirá de contenedor para todos los demás formularios. Tendrá un registro de cada uno de sus hijos para poder llevar a cabo operaciones sobre ellos.

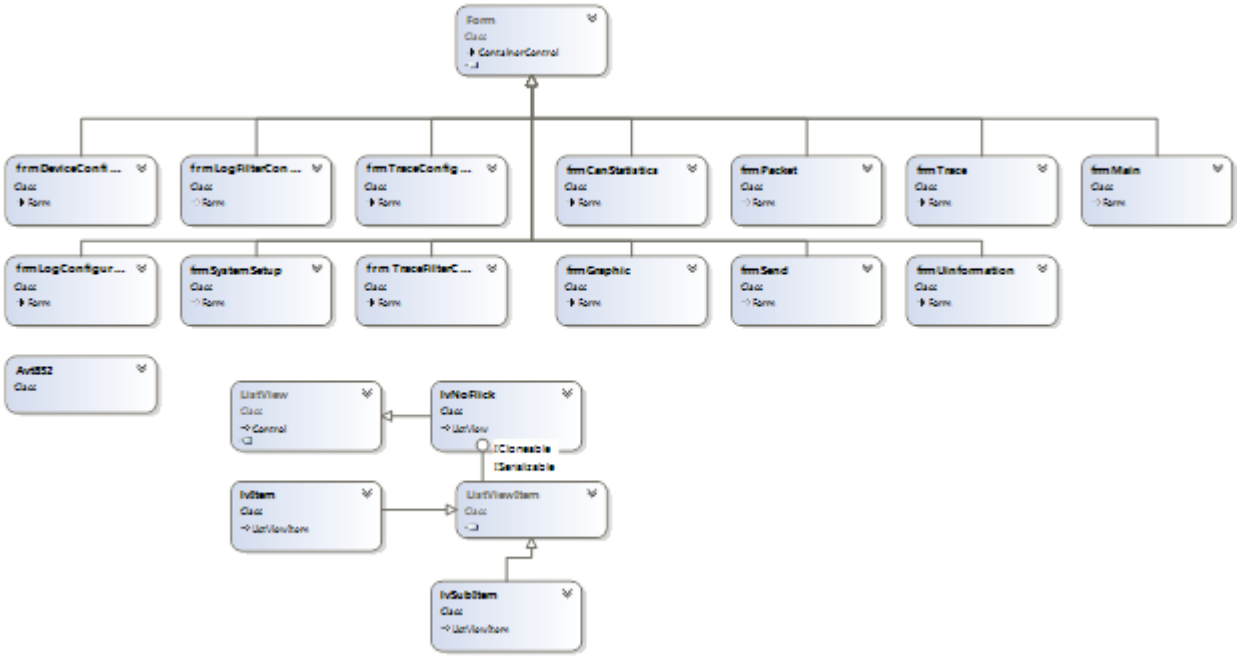


Diagrama de clases, que muestra la interacción entre todos los módulos del sistema.

3.2.2 Propiedades de la clase AVT

avtState. - Esta propiedad nos indica en qué estado se encuentra la AVT en ese momento del tiempo y nos informa si se encuentra en ejecución o recibiendo datos o en un estado de espera de inicialización

baudrateCHN0.- Velocidad de CAN con la que trabajara el canal 0 de la AVT.

baudrateCHN4.- Velocidad de CAN con la que trabajara el Canal4 de la AVT.

captureTime. - Tiempo transcurrido en milisegundos desde que se la herramienta se inicializo y comenzó la captura de paquetes del bus de CAN.

channel. - Nos permite seleccionar con cual canal de CAN va a trabajar la AVT, Canal 0, canal 4 o ambos.

connected. - Propiedad verdadera o falsa que nos indica si el equipo de cómputo ya se encuentra conectado con la AVT por medio de su puerto serial y este ya ha sido configurado correctamente.

errFramesXsecond. - En caso de existir paquetes de error en la recepción o transmisión de los mensajes de CAN, la cantidad por segundo será almacenado en esta propiedad

graphic. - Variable que almacena una instancia de la ventana o Windows form encargado de procesar los mensajes y extraer de ello las señales para que éstas se puedan visualizar por medio de series en una gráfica diferenciadas por el color.

log. - Variable que almacena una instancia de la ventana o Windows form encargado de la configuración y llenado del archivo proveniente de la ventana de registro.

packet. - Variable que almacena una instancia de la ventana o Windows form encargado del análisis de los paquetes que llegan desde la red de CAN.

portName. -Nombre del Puerto serial al cual está conectado la AVT siendo de la forma "COM#" en sistemas operativos Windows y de la forma "TTYUSB#" en los sistemas operativos Linux, en ambos casos el símbolo # indica un numero consecutivo que el sistema operativo entrega a cada uno de los dispositivos conectados.

rxData. - Cadena de datos que almacena el último mensaje que ingrese desde la red de CAN.

size. - Variable que almacena el tamaño en bytes del último mensaje que ingreso desde la red de CAN

trace. - Variable que almacena una instancia de la ventana o Windows form encargada del registro y visualización de todos los mensajes o paquetes desde la computadora hacia la red de CAN y viceversa.

3.2.3 Métodos de la clase AVT.

bStart. - Inicializa y carga la configuración seleccionada por el usuario para comenzar a recibir información desde la red de CAN.

bStop. -Detiene la ejecución de la medición y regresa la AVT a su estado de fábrica en la que se encuentra lista para recibir los comandos de configuración.

bytesToString. - Convierte las respuestas de la AVT desde arreglo de bytes a su correspondiente cadena en bytes hexadecimal.

comparePackets. -Compara dos arreglos de bytes desde su tamaño y el contenido de cada uno para verificar si ambos tienen los mismos valores en cada una de sus posiciones o si alguna de ellas es diferente.

connect. - Conecta el a la aplicación con el puerto serial seleccionado e inicia la comunicación con el dispositivo conectado para él envió y recepción de mensajes.

consumer. - Función que se ejecuta en un hilo que permanece activo extrayendo paquetes desde la cola en la que se almacena la información de cada uno de ellos para después despacharlos a las respectivas ventanas que componen el sistema.

dequeueErr. - Saca de la cola de errores el primero que entro y lo almacena en una variable de tipo cadena que contiene los datos del error.

dequeueRx. - Saca de la cola de mensajes recibidos el último que entro al buffer de almacenamiento y lo almacena en una variable del tipo cadena que contiene los datos del mensaje

disconnect. -Desconecta el Puerto serial de la AVT conectada para dejarlo libre y así poder reconectarla a la AVT o para otras aplicaciones puedan utilizarlo.

enqueueErr. -Mete en la cola de errores el último de ellos en ser detectados para después estos puedan ser extraídos y manejados por la función correspondiente.

enqueueRx. - Mete en la cola de mensajes el último que arribo al buffer de recepción del puerto serial para que después estos puedan ser extraídos y manejados por las funciones correspondientes.

enterCanMode. -Pone la AVT en modo de configuración CAN para permitir cargar los filtros y la configuración de las velocidades de cada uno de los canales.

extractBytes. - Extrae la cantidad de bytes deseada desde un arreglo de bytes de mayor tamaño.

getErrDescription. - Obtiene la descripción del error en base a un código predefinido en el manual de usuario de la AVT-852.

geNibble. - Obtiene el conjunto de cuatro bytes bajo o alto de cualquiera de los bytes de un arreglo proporcionado, esto es útil ya que la mayoría de los mensajes de configuración y error que se mandan y reciben desde la AVT contienen información sobre la acción realizada en grupos de cuatro bits.

getStringFromResponse. -Regresa una sub cadena que contiene el número de bytes deseados desde una cadena más larga en la cual se encuentra la respuesta completa o algún mensaje de CAN.

messageLen. - Regresa el ancho en bytes del mensaje que se le pasa como parámetro, este mensaje es en formato de cadena que conforman los bytes en hexadecimal.

new. - Constructor de la clase que permite instanciar los objetos con los parámetros iniciales de conexión y velocidad de los puertos.

oneByteToOneString. - Convierte un byte decimal a su correspondiente cadena de caracteres hexadecimales.

oneStringToOneByte. - Lo contrario de la función anterior convierte una cadena de un byte en hexadecimal y retorna su correspondiente valor decimal.

readStream. - Lee el flujo de datos de la conexión que se creó con la AVT por medio del puerto serial.

resetUnit. - Regresa a al AVT a su estado de fábrica en el que se encuentra preparada para recibir los nuevos comandos de configuración y ejecutarlos para ponerla en el modo deseado.

restartCounters. - Reinicia los contadores del bus de CAN con los que cuenta la librería, como son, el número de mensajes y bytes por segundo, numero de errores, etc.

sendCommandToAvt. - Envía un comando de configuración hacia la AVT, esta función está sobrecargada para poder recibir como parámetro la cadena hexadecimal del comando a enviar o su correspondiente arreglo de bytes.

sendCommandToNetwork. - Esta función envía un paquete de datos a la red de CAN por medio de la AVT, recibe como parámetros el identificador y los datos del mensaje, así como el canal por medio del cual se enviarán los datos.

setBaudrate. - Configura la velocidad de transmisión de los canales de CAN de la AVT, de manera individual pasando como parámetro el canal y la velocidad.

stringTobyte. - Convierte una cadena hexadecimal de más de un byte en su correspondiente arreglo de bytes para que estos puedan ser enviados a la AVT.

updateAvtQueryes. - Actualiza las definiciones de los errores desde un archivo csv externo el cual contiene el código y la descripción de los mismos.

updateCounters. - Actualiza los contadores internos de la librería para la comunicación CAN.

writeStream. - Escribe el flujo de datos expresados en arreglos de bytes, para él envió de comandos de configuración hacia la AVT o para él envió de los mensajes desde el equipo de cómputo a la red de CAN.

3.2.4 Eventos de la clase AVT.

onChannelActivityChange. - Evento que se dispara cuando la actividad de alguno de los canales de CAN cambia de estado para pasar esta información al formulario que contiene a la clase.

onError. - Informa sobre algún error en la comunicación o inicialización de la librería en algún momento determinado.

onRxData. - Informa cuando ha llegado un paquete de datos al buffer de recepción de la librería.

onTxData. - Informa cuando un paquete fue enviado hacia la red de CAN desde la AVT.

3.3 Implementación

La primera etapa que se lleva a cabo en la interacción del equipo de cómputo con la AVT es la conexión con el puerto serial en el cual se montó el driver y se enumeró en el sistema operativo, esto se lleva acabo primero creando una instancia de la clase SerialPort.

```
Friend WithEvents _port As SerialPort
```

Figura 8: Declaración de la instancia de la clase SerialPort

Esa variable será la que nos permitirá la conexión e interacción con la AVT, se debe tener en cuenta que para declarar dicha variable se debe de importar el espacio de nombres correspondiente:

```
Imports System.IO.Ports
```

Figura 9: Importación del espacio de nombres para manejo del puerto serial

Después de declarar la variable con la que manejaremos la comunicación es necesario indicarle las propiedades con la que se establecerá la conexión hacia el puerto serial en el que se encuentra montada la AVT:

```
Public Function Connect() As Boolean
        Dim found As Boolean = False

        For Each p As String In My.Computer.Ports.SerialPortNames
            If p = avtValues.Address Then found = True
        Next

        If Not found Then
            Return False
        End If

        Try
            _port = New SerialPort
            With _port
                If .IsOpen Then .Close()
                .PortName = avtValues.Address
                .BaudRate = 19200
                .ReceivedBytesThreshold = 1
                .Open()
            End With
            _connected = True
            Return True
        Catch ex As Exception
            Return False
        End Try
End Function
```

Figura 10: Función para conectar la aplicación al puerto serial

En el código anterior, inicializa la instancia con un nuevo objeto de la clase SerialPort, cerrar el puerto en caso de que este abierto. Después indicarle el puerto al que nos queremos conectar, en el formato "COM#", donde # el número de puerto que el sistema operativo asignó al puerto virtual. Si el sistema operativo es Linux, el nombre del puerto está indicado con el formato "ttyS#" o "ttyUSB#" donde # es el número que el sistema operativo asignó al nuevo puerto serial. Si la conexión es exitosa nuestra función nos regresa un valor verdadero y falso en caso de que exista un error en la conexión, este error se puede atrapar por medio de un bloque catch, este tipo de código nos permite recibir las excepciones antes de que las detecte el sistema operativo y así evitar que nuestra aplicación se cierre en caso de que exista alguno. Después de trabajar con nuestro puerto serial es necesario cerrar la conexión que creamos para que así el puerto esté disponible, aunque el sistema operativo libera todos los recursos que nuestra aplicación utilizo es recomendable que nos aseguremos que dichos recursos se liberen antes de cerrar nuestro programa, esto se lleva a cabo por medio de la siguiente función:

```
Public Function Disconnect() As Boolean
        If _connected = False Then Return True
        Try
            SendCommandToAvt("F1 A5")
            If _port.IsOpen Then _port.DiscardInBuffer() : _port.Close()
            _connected = False
            Return True
        Catch ex As Exception
            Return False
        End Try

End Function
```

Figura 11: Función para desconectar la aplicación del puerto serial

La primera acción es verificar que el Puerto en realidad se encuentra abierto antes de llamar el método para cerrarlo. También, limpiar el buffer para asegurarnos que no quedan datos en el que nos puedan causar problemas en la siguiente vez que se abra el puerto ya que estas tareas se llevan a cabo muy a menudo en aplicaciones que inician y detienen la comunicación con el puerto serial. Adicional a estas tareas de suma importancia necesitamos enviar y recibir datos hacia y desde la AVT. También debemos de convertir los comandos en hexadecimal en su correspondiente arreglo de bytes como se muestra en el siguiente código:

```
Public Shared Function StringToByte(ByVal command As String) As Byte()
        Dim bytes() As Byte = {}


        Try
            Dim commandChar() As Char = Replace(command, " ", "").ToCharArray
            Dim str As String = ""
            Dim x As Integer
            Dim count As Integer = 0
            For x = 0 To commandChar.Length - 1 Step 2
                str = str + commandChar(x).ToString + commandChar(x + 1)
                Array.Resize(bytes, count + 1)
                bytes(count) = Convert.ToByte(str, 16)
                count += 1
                str = ""
            Next
            Return bytes
        Catch ex As Exception
            Return bytes
        End Try

End Function
```

Figura 12: Conversión de cadena hexadecimal a arreglo de bytes

Esta función toma como parámetro la cadena que contiene el comando que se enviara a la AVT y nos retorna su correspondiente arreglo de bytes. Por ejemplo, si se desea enviar el comando "E1 99" que es el comando encargado de poner la AVT en modo CAN, después de llamar a la función con esta entrada el resultado será un arreglo de bytes de dos posiciones con el valor 0xE1=225 en la primera posición del arreglo y el valor 0x99=153 en la segunda posición. Este comando será enviado como arreglo de bytes a la AVT la cual lo recibirá byte tras byte y una vez que los reconozca todos, ejecutará el código correspondiente para poner la AVT lista para ejecutar el comando del modo CAN. La contraparte de esta función es la que convierte las respuestas de la AVT que estaban en un formato conformado por un de arreglo de bytes a su correspondiente cadena para que puedan ser comparadas de manera más fácil y sean más comprensibles para el programador, el código de la función es el siguiente:

```
Public Shared Function BytesToString(ByVal msgBytes() As Byte) As String
        Try

            Dim x As Integer
            Dim str As String = ""
            Dim str2 As String = ""
            For x = 0 To msgBytes.Length - 1
                If msgBytes(x) <= 15 Then
                    str2 = "0" + Convert.ToString(msgBytes(x), 16)
                Else
                    str2 = Convert.ToString(msgBytes(x),
16).ToUpper.ToUpper
                End If
                str += str2.ToString
                str += " "
            Next
            Return str.Trim
        Catch ex As Exception
            Return ""
        End Try
End Function
```

Figura 13: Conversión de arreglo de bytes a cadena hexadecimal

En esta función el retorno es la cadena ya convertida a partir del arreglo de bytes que se manda como parámetro de entrada. Como el ejemplo anterior si enviamos el comando para poner la AVT en modo CAN y si se logra de manera correcta llevar a cabo la tarea, nos enviará respuesta en formato de arreglo de bytes, de tal forma si se obtienen los valores145=0x91 y 16=0x10, se nos está indicando que la AVT ejecutó de manera correcta el comando enviado. Teniendo las funciones de conversión listas es momento de enviar y recibir datos, la primera parte se lleva a cabo por medio de las siguientes funciones:

```
Public Function SendCommandToAvt(ByVal command As String) As Boolean
        Dim byteOut() As Byte = StringToByte(command)
        Try
            WriteStream(byteOut)
            Return True
        Catch ex As Exception
            Return False
        End Try
End Function
```

Figura 14: Función para escribir los comandos hacia la AVT

```
Public Function SendCommandToAvt(ByVal command As String, ByVal response As
String, ByVal responseSize As Integer) As Boolean
        Dim byteOut = StringToByte(command)
        Dim byteToCompare = StringToByte(response)

        Try
            WriteStream(byteOut)
            Threading.Thread.Sleep(50)
            Dim byteIn() As Byte = ReadStream(responseSize)
            Threading.Thread.Sleep(50)
            Return ComparePackets(byteIn, byteToCompare, responseSize)
        Catch ex As Exception
            Return False
        End Try

End Function
```

Figura 15: Sobrecarga de la función para escribir los comandos hacia la AVT.

Ambas funciones envían el respectivo comando de cadena convertido a arreglo de bytes con la llamada a las funciones previamente mostradas. Estas funciones se encuentran sobrecargadas ya que la primera acepta como parámetro solo el comando a enviar, mientras que la segunda además del comando a enviar recibe el parámetro de la respuesta que se espera y el número de bytes que se van a comparar. En si las funciones no envían directamente la información al puerto, sino que utilizan otra función llamada WriteStream que es la encargada de escribir los datos en el puerto serial y la segunda utiliza a la funcione ReadStream que es en la que se lleva a cabo la lectura del puerto serial como se muestra a continuación:

```
Public Function WriteStream(ByVal command() As Byte) As Boolean
        If _port.IsOpen Then
            Try
                _port.Write(command, 0, command.Length)

                frmUinformation.AddApplicationMessage("Write Stream : " +
BytesToString(command), frmUinformation.MessageType.AVT)
                Return True
            Catch ex As Exception
                Return False
            End Try
        End If
End Function
```

Figura 16: Escritura de datos hacia el flujo del puerto serial.

```
Public Function ReadStream(ByVal size As Integer) As Byte()
        Dim bytein() As Byte = {}

        If _port.IsOpen Then
            Try

                Array.Resize(bytein, size)
                _port.Read(bytein, 0, bytein.Length)
                _rxdata = BytesToString(bytein)
                _port.DiscardInBuffer()
                frmUinformation.AddApplicationMessage("Read Stream : " +
_rxdata, frmUinformation.MessageType.AVT)
            Catch ex As Exception
                Return bytein
            End Try
        End If
        Array.Resize(bytein, size)
        Return bytein
End Function
```

Figura 17: Lectura de datos desde el flujo del puerto serial.

La función de lectura hace uso del método write en su sobrecarga que permite enviar arreglos de bytes, siendo sus parámetros el nombre del arreglo, la posición donde se encuentra el primer byte a enviar y el offset o cantidad de bytes a partir de la posición indicada. La función de lectura hace uso del método Read en su sobrecarga que permite almacenar los bytes leídos directamente en un arreglo de bytes, lo cual es muy conveniente para evitar conversiones innecesarias. Este método es muy parecido al de Write en la cantidad y tipo de parámetros por lo que es muy recomendable utilizar una variable global del tipo arreglo de bytes y redimensionarla durante la ejecución de la aplicación para ahorrar espacio en variables y liberar espacio innecesario para la cantidad de bytes.

3.4 Integración

Para la integración de la librería de CAN con la aplicación de control, fue necesario simular los mensajes que viajaban en la red. Esto se llevó a cabo utilizando una herramienta llamada CanAlyzer de la empresa vector. Por medio de ella se comenzaron a probar cada una de las ventanas y la librería para ver si la conexión entre ellas era positiva. Primero integramos la ventana de configuración para verificar que la AVT y la librería se inicializaban de manera correcta. Fue necesario cambiar algunos parámetros y la forma en la que el usuario seleccionaba el puerto serial. De esta forma se optó por enumerar los puertos con los que cuenta la computadora por medio de las funciones con las que cuenta MS Visual Studio, en especial MS Visual Basic que nos permite obtener el nombre de todos los puertos seriales con los que cuenta la computadora, a continuación, se muestra dicho código:

```
For Each p As String In My.Computer.Ports.SerialPortNames

            sp.PortName = p
            sp.BaudRate = 19200
            sp.ReadTimeout = 1000
            sp.WriteTimeout = 1000

            Try

                If sp.IsOpen = False Then sp.Open()
                avtValues.Address = p

                'Model
                bts = Avt852.StringToByte("F0")
                sp.Write(bts, 0, bts.Length)
                Array.Resize(bts, 4)
                sp.Read(bts, 0, 4)
                msg = Avt852.BytesToString(bts)
                avtValues.Model = msg.Substring(7).Replace(" ", "")

                If sp.IsOpen Then sp.Close()
                avtValues.configured = True

            Catch ex As Exception
                If sp.IsOpen Then sp.Close()
                avtValues.configured = False
                Me.ToolStripStatusLabelHardware.Text = "Hardware Not Found"

                Me.ToolStripStatusLabelHardware.Image =
imgLstMenu.Images(1)
            End Try


        Next
```

Figura 18: Listado y pruebas de los puertos seriales

Por medio de la función anterior se enumeran cada uno de los puertos seriales. Después se establece la comunicación con cada uno de ellos y si éste responde dentro del tiempo determinado se da por entendido que se encuentra conectada la AVT. Si se alcanza el tiempo de espera, entonces se entiende que no existe ningún dispositivo que responda a los comandos enviados. Cuando se obtiene respuesta positiva los valores se almacenan en una estructura para su uso posterior y además se obtiene el modelo del equipo, así como su número de serie para mostrarlo al usuario.

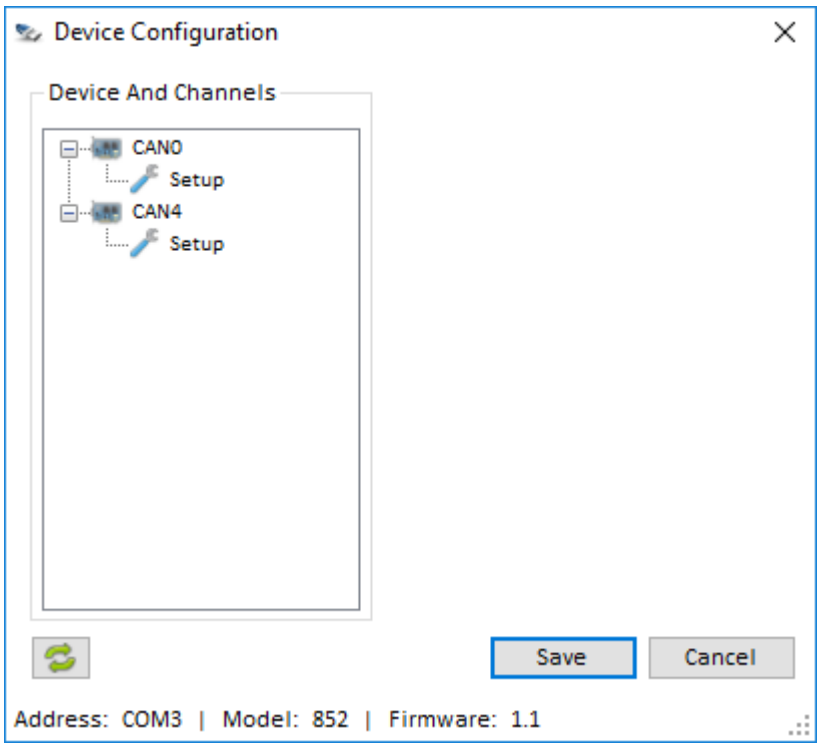


Figura 19: Ventana de configuración que muestra el dispositivo encontrado.

Una vez que se tiene identificada la dirección del dispositivo conectado, ésta es utilizada para realizar la conexión, el envío y recepción de datos por medio del puerto serial. Otra parte muy importante en el sistema es la visualización de los mensajes que se envían y se reciben desde la red. Para esto se implementó una función en cada uno de los formularios la cual despacha cada mensaje que es enviado desde y hacia la librería. Cada uno de los formularios cuenta con el código respectivo para la tarea que realiza, así la librería hace la llamada a cada una de las funciones con las que cuentan las instancias de los formularios que se dieron de alta para llevar a cabo la tarea. En el caso de la ventana trace, la información que es enviada desde la librería es

procesada a manera de elementos de la lista principal en la que se agregan un elemento por cada uno de los mensajes que han sido procesados por la librería principal. Al igual que la ventana de trace, la ventana de gráficos cuenta con una función similar pero la mayor diferencia es que ésta extrae de los mensajes las señales que el usuario dio de alta y las muestra en la gráfica en su correspondiente línea o color de señal. La ventana de visualización de paquetes extrae de cada mensaje los datos importantes de los paquetes a visualizar y lleva un conteo del número de bytes y paquetes que han arribado y que cumplen con los requisitos que el usuario dio de alta. Por último, la ventana log se encarga de grabar en un archivo los mensajes que son proporcionados por la librería. Este archivo es compatible con otras herramientas de la empresa vector ya que cuentan con la extensión necesaria y el formato que estas aplicaciones comprenden. En ese orden se fueron integrando cada uno de los formularios en la aplicación principal. Además de los formularios descritos anteriormente se agregaron otros para facilita la configuración del sistema al usuario. Una de las más importantes es la ventana de configuración del sistema que permite habilitar o deshabilitar cada uno de los módulos con los que cuenta la aplicación. De esta forma se eliminan los módulos en los que el usuario no tiene interés en la medición tan solo con seleccionar la opción deseada utilizando el botón derecho del ratón:

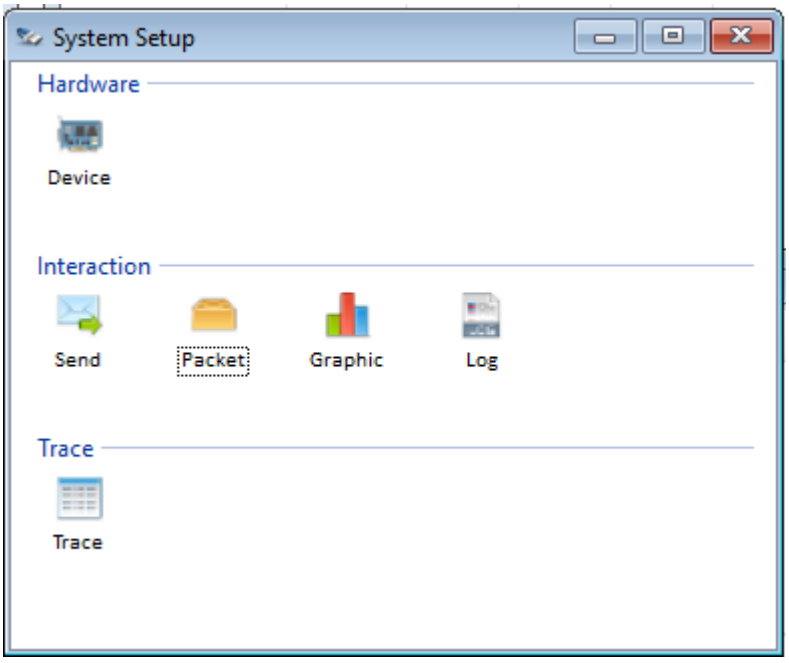


Figura 20: Ventana de configuración global del sistema

3.5 Pruebas

Para las pruebas del sistema al igual que la integración del mismo, se utilizó la herramienta de la empresa vector llamada CanAlyzer, el cual es un software analizador de CAN de dicha empresa que utiliza un dispositivo llamada CanCase también desarrollado por ellos. Por medio de esta herramienta se llevaron a cabo las tareas de envío y recepción de mensajes para probar nuestro sistema. Por medio de dichas pruebas se verificaron varias limitantes de nuestro sistema entre las cuales destacan que la velocidad de recepción de nuestra aplicación no llega a los 10ms por las mismas limitantes del puerto serial, lo que da lugar un desfase de los mensajes los cuales no son mostrados en tiempo real.

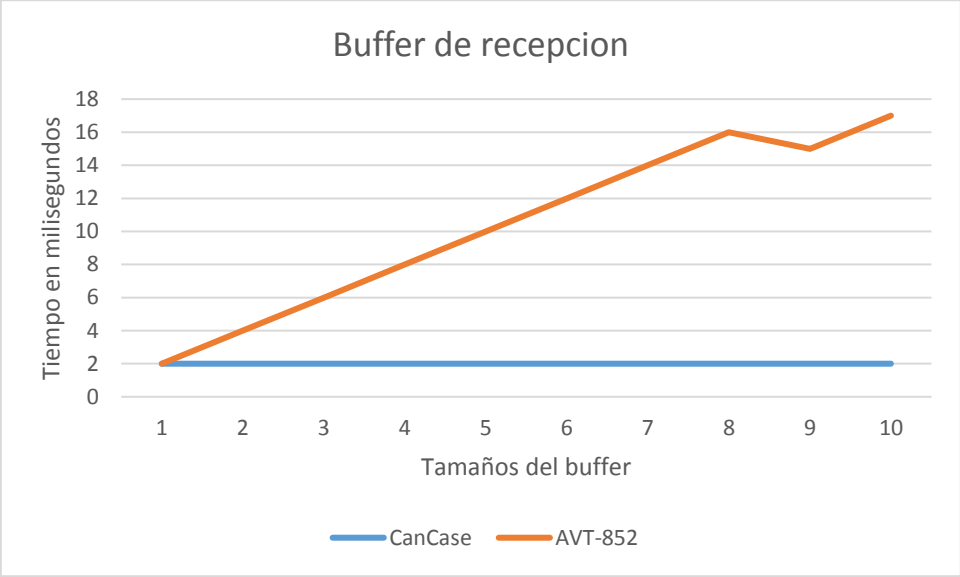


Figura 21: aumento en el buffer de recepción en comparación con CanCase.

La aplicación con la que se probó nuestro sistema cuenta con un driver nativo de USB y acceso a bajo nivel del puerto lo que le permite lecturas y escrituras más rápidas que las de nuestro sistema, en este caso no es posible cambiar este factor ya que el fabricante de la misma selecciono esta manera de transferencia. Lo que se puede hacer en versiones posteriores para atenuar esta diferencia en los tiempos de lectura es utilizar el api de Windows Win32Api. Ésta hace uso de la HAL (Capa de abstracción de hardware) de Windows. Esta capa de software trata directamente con el hardware del

equipo, en la cual no es necesario conocer información específica del hardware. La HAL proporciona rutinas que permiten utilizar un solo controlador para el dispositivo en diferentes plataformas de hardware. Esto facilita el desarrollo de la capa de aplicación para el control de los dispositivos seriales a través del uso de las rutinas que la HAL proporciona. Una ventaja es que oculta detalles dependientes del hardware como interfaces E/S y controladores de interrupción. Las funciones que ofrece la librería de Windows para el acceso a los puertos seriales desde la plataforma .net son muy parecidas a las utilizadas en los lenguajes de programación Pascal y C++, aunque sigue siendo una librería externa la forma de cambiar las librerías estándar por la que nos ofrece Windows es la siguiente:

La función Open de la clase SerialPort se sustituye por la unión de tres funciones llamadas CreateFile, BuildCommDBC y SetCommstate. La primera crea un archivo virtual para la escritura y lectura desde un dispositivo de entrada y salida como es el puerto serial. La función retorna un manejador que puede ser utilizado para acceder al archivo o dispositivo para varios tipos de entradas y salidas dependiendo de las banderas que se utilicen. La definición de la función es la siguiente:

```
HANDLE WINAPI CreateFile(
_In_     LPCTSTR               lpFileName,
_In_     DWORD                 dwDesiredAccess,
_In_     DWORD                 dwShareMode,
_In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,
_In_     DWORD                 dwCreationDisposition,
_In_     DWORD                 dwFlagsAndAttributes,
_In_opt_ HANDLE                hTemplateFile
);
```

Figura 22: Parámetros de la función para crear un archivo virtual.

La segunda función con la que se logra la creación de un manejador del puerto serial es BuildCommDBC, la cual llena una estructura que define los parámetros de control para la comunicación serial con algún dispositivo que utilice este protocolo. La definición de la función es la siguiente:

```
BOOL WINAPI BuildCommDCB(
 _In_  LPCTSTR lpDef,
 _Out_ LPDCB   lpDCB
);
```

Figura 23: Parámetros de la función para crear un manejador del puerto serial.

De igual forma la estructura DBC se conforma de la siguiente manera:

```
typedef struct _DCB {
 DWORD DCBlength;
 DWORD BaudRate;
 DWORD fBinary :1;
 DWORD fParity :1;
 DWORD fOutxCtsFlow :1;
 DWORD fOutxDsrFlow :1;
 DWORD fDtrControl :2;
 DWORD fDsrSensitivity :1;
 DWORD fTXContinueOnXoff :1;
 DWORD fOutX :1;
 DWORD fInX :1;
 DWORD fErrorChar :1;
 DWORD fNull :1;
 DWORD fRtsControl :2;
 DWORD fAbortOnError :1;
 DWORD fDummy2 :17;
 WORD wReserved;
 WORD XonLim;
 WORD XoffLim;
 BYTE ByteSize;
 BYTE Parity;
 BYTE StopBits;
 char XonChar;
 char XoffChar;
 char ErrorChar;
 char EofChar;
 char EvtChar;
 WORD wReserved1;
} DCB, *LPDCB;
```

Figura 24: Estructura que contiene la información del puerto serial.

La tercera función necesaria para abrir el puerto serial y comenzar a trabajar con él, es SetCommState. Esta función configura la comunicación con el dispositivo de acuerdo a las especificaciones seleccionadas en la estructura DCB mostrada anteriormente. La función reinicializa todas las configuraciones de control del hardware seleccionado, pero no vacía las colas de entrada y salida. La definición de la función es la siguiente:

```
BOOL WINAPI SetCommState(
 _In_ HANDLE hFile,
 _In_ LPDCB  lpDCB
);
```

Figura 25: Función para abrir la comunicación con el puerto serial.

Para suplir la función Write de la clase serialPort se utiliza su contraparte WriteFile la cual escribe los datos al archivo especificado o al dispositivo de entrada y salida. Esta función está diseñada para ambos tipos de operación síncronas y asíncronas y el formato de la función es el siguiente:

```
BOOL WINAPI WriteFile(
 _In_       HANDLE       hFile,
 _In_       LPCVOID      lpBuffer,
 _In_       DWORD        nNumberOfBytesToWrite,
 _Out_opt_  LPDWORD      lpNumberOfBytesWritten,
 _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

Figura 26: Función para escribir los datos en el puerto serial.

Para suplir la función Read de la clase SerialPort se utiliza su contraparte ReadFile la cual lee los datos desde un archivo especificado o algún dispositivo de entrada y salida. La lectura ocurre a la posición especificada por el puntero de archivo si éste es soportado por el dispositivo. Esta función está diseñada para ambos tipos de operación, síncrona y asíncrona y el formato de esta función es el siguiente:

```
BOOL WINAPI ReadFile(
  _In_        HANDLE       hFile,
  _Out_       LPVOID       lpBuffer,
  _In_        DWORD        nNumberOfBytesToRead,
  _Out_opt_   LPDWORD      lpNumberOfBytesRead,
  _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

Figura 27: Función para leer los datos desde el puerto serial.

Por último, para suplir la función Close de la clase SerialPort, la cual se encarga de cerrar el flujo con el puerto se utiliza su contraparte CloseHandle. Esta función cierra al manejador abierto anteriormente ya sea a un archivo o a un dispositivo de entrada y salida. La definición de dicha función es la siguiente:

```
BOOL WINAPI CloseHandle(
  _In_ HANDLE hObject
);
```

Figura 28: Función para cerrar la conexión con el puerto serial.

4 RESULTADOS

A lo largo del desarrollo de este proyecto de tesina, se analizó el control de la AVT-852 así como de la instrumentación y comunicación con dicho dispositivo por medio del protocolo RS-232. Se analizó la diferente documentación que ofrece el fabricante, así como las notas de aplicación. La revisión de estos tópicos permitió que se establecieran bases sólidas para la creación de una aplicación que permite la conexión con la interfaz de CAN y el equipo de cómputo, además de la creación de una librería o dll que utiliza el paradigma de orientación a objetos para facilitar la conexión de uno o más dispositivos a la aplicación principal y permita la reutilización en proyectos posteriores. Todo esto permitió que al final del proyecto se obtuviera un software o aplicación con las siguientes características:

4.1 Librería o dll de comunicación

Esta librería permite la conexión y el manejo de la AVT-852, siguiendo el patrón de diseño orientado a objetos, con lo cual, obtenemos un software reutilizable en cualquier tipo de aplicación bajo el entorno .Net framework de Microsoft©. Permite la conexión con el dispositivo, envío y recepción de comandos desde y hacia la PC, envío y recepción de mensajes desde y hacia la red, creación de mensajes periódicos, así como la manipulación de cada uno de ellos en tiempo de ejecución. Para poder utilizar la librería en un nuevo proyecto es necesario crear una referencia a la dll generada y crear una instancia de la clase Avt852. Por medio de esta instancia crearemos la conexión con el dispositivo seleccionado.

4.2 Software de medición

El resultado del desarrollo de software, es la aplicación LightNv. Dicha aplicación será puesta en marcha en los equipos de pruebas de nuestros clientes. Primero será necesario que nuestro sistema pase las pruebas de campo correspondientes, para poder ser liberada en los laboratorios de nuestros clientes. Las pruebas funcionales y no funcionales fueron hechas en nuestra empresa en un ambiente simulado, ya que para tener llevar a cabo dichas pruebas en los equipos reales será necesario contar con el permiso de nuestro cliente, lo cual, se encuentra en proceso.

Los requisitos mínimos del sistema para el equipo de cómputo en el cual se ejecutará la aplicación LightNV, son los siguientes:

- Sistema operativo Windows 7 o posterior.
- 2Gb de RAM o más.
- 1.5 GHz de velocidad en la unidad central de procesamiento o más.
- 5 Gb libres en el disco duro o más.
- Un puerto USB libre para la conexión de la AVT-852.

La aplicación nos permite la búsqueda y configuración de cualquier AVT-852 conectada a nuestra computadora, ya sea de manera automática o manual, sin importar la versión de firmware integrado en el dispositivo.

La aplicación cuenta con varios módulos que nos permites realizar diferentes tareas, desde la conexión y configuración hasta el registro de los paquetes en la red, A continuación, se explican cada uno de los módulos con los que cuenta el sistema, así como una imagen demostrativa y un ejemplo simulado para observar el funcionamiento de cada uno de ellos.

4.2.1 Configuración del sistema

En este módulo el usuario puede configurar cada una de las partes que componen el sistema, Renombrar cada uno de los bloques que lo componen o deshabilitar cualquiera de las partes que el usuario no crea necesarias.

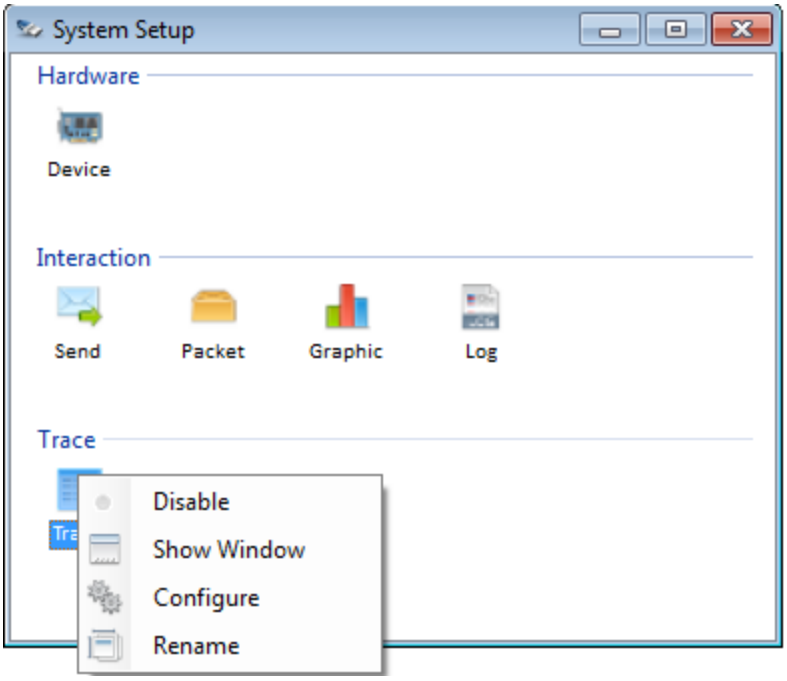


Figura 29: Ventana para la configuración del sistema

Como podemos observar en la imagen, cada uno de los módulos del sistema puede ser configurado de manera independiente. El usuario puede deshabilitar cualquiera de las partes del sistema, renombrar las ventanas para que la representación de los datos sea la adecuada o configurar los parámetros específicos de cada una de las partes. Cada módulo presenta configuraciones independientes de acuerdo a la funcionalidad y características del trabajo que realiza. Las partes que componen el sistema son las siguientes:

- Configuración del dispositivo.
- Envío de mensajes a la red.
- Monitoreo de paquetes desde la red.
- Visualización grafica de señales.
- Ventana de registro.

4.2.2 Configuración del dispositivo

Para poder llevar a cabo una medición o registro de los mensajes de CAN, es necesario que el dispositivo se encuentre conectado y configurado. Este módulo es capaz de buscar cualquier AVT-852 conectada al equipo de cómputo. Cuando se lleva a cabo una búsqueda y se obtiene el número de puerto serial en el que se encuentra conectado el equipo, el módulo continúa con la consulta del modelo y la versión de firmware instalada. Este módulo nos muestra el número de canales de CAN con los que cuenta nuestro dispositivo, además nos permite seleccionar y configurar cada uno de ellos. Las configuraciones con las que cuenta cada canal son: velocidad de CAN, filtro en formato hexadecimal, mascara en formato hexadecimal y el uso de ese canal para futuras mediciones. A continuación, se muestra la ventana correspondiente a este módulo y cada una de las partes que la conforman.

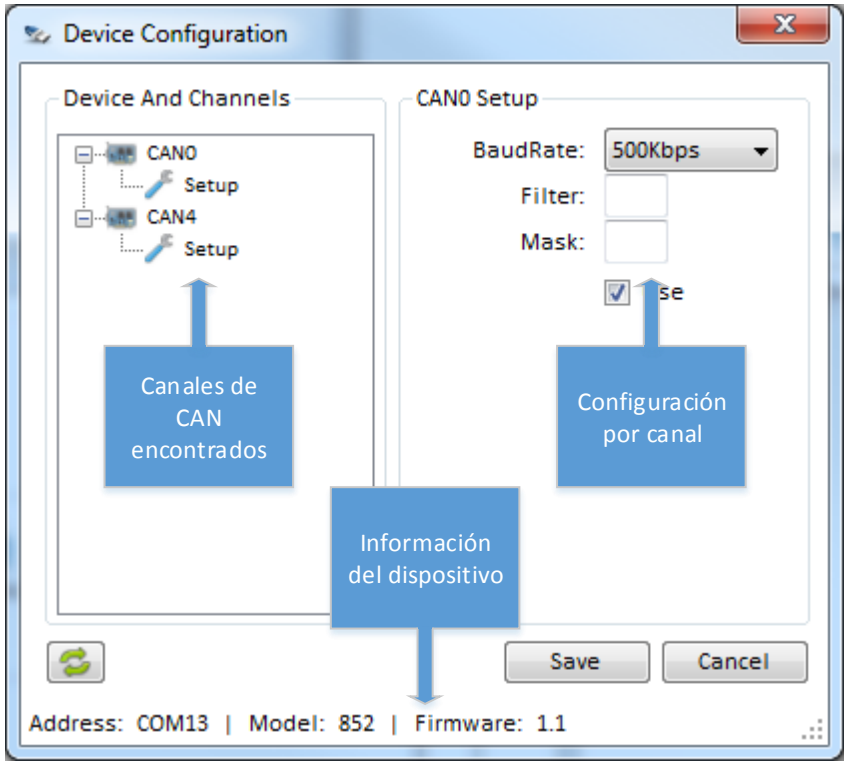


Figura 30: Ventana para la configuración del hardware conectado.

En el ejemplo que se muestra en la figura, se seleccionó solo el canal 0 de CAN para la transmisión de los datos a una velocidad de 500 kbps y no se asignó ningún filtro ni mascara, de esta manera todo el tráfico de la red se visualizara en la ventana de registro

4.2.3 Envío de mensajes a la red

Este módulo nos permite la interacción con la red de manera activa por medio del envío de mensajes o paquetes a la red de CAN. Estos mensajes pueden ser dos tipos, el primero de ellos nos permite enviar los datos en un evento asociado al botón de disparo, lo que significa que el mensaje será lanzado a la red cada vez que el usuario lleve a cabo la acción de presionar el botón asignado a cada mensaje, la segunda forma nos permite enviar los paquetes de manera periódica con un tiempo de transferencia configurable. Cabe destacar que el número de mensajes periódicos que el sistema puede enviar, depende de la cantidad de memoria con la que cuente el equipo de cómputo en el que se ejecute la aplicación. A continuación, se muestra la imagen de la ventana perteneciente a este módulo, así como cada una de las partes que lo conforman.

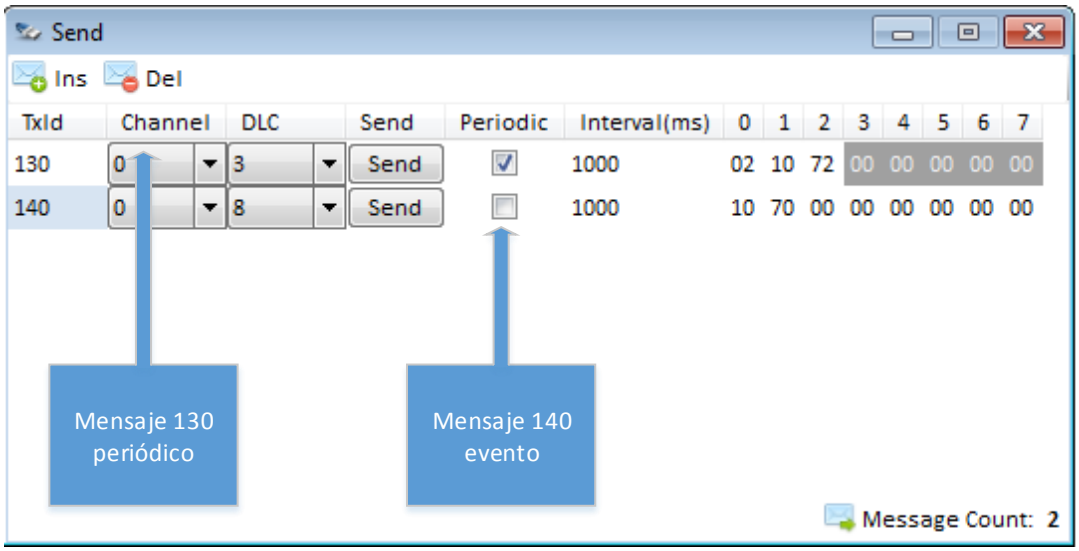


Figura 31: Ventana para la configuración de los mensajes a enviar.

En el ejemplo de la imagen anterior, se pueden observar dos mensajes. El primero de ellos es un mensaje periódico con los siguientes datos: identificador (130), canal (0), dlc (3), periódico(SI), intervalo de envío (1000 ms), datos (02 10 72). El segundo mensaje configurado en este ejemplo, es un mensaje enviado por medio de un evento, el cual se dispara cada vez que el usuario presiona el botón de enviar asignado a dicho mensaje. Este mensaje cuenta con los siguientes datos: Identificador (140), canal (0), dlc (8), periódico(NO), datos (10 70 00 00 00 00 00 00).

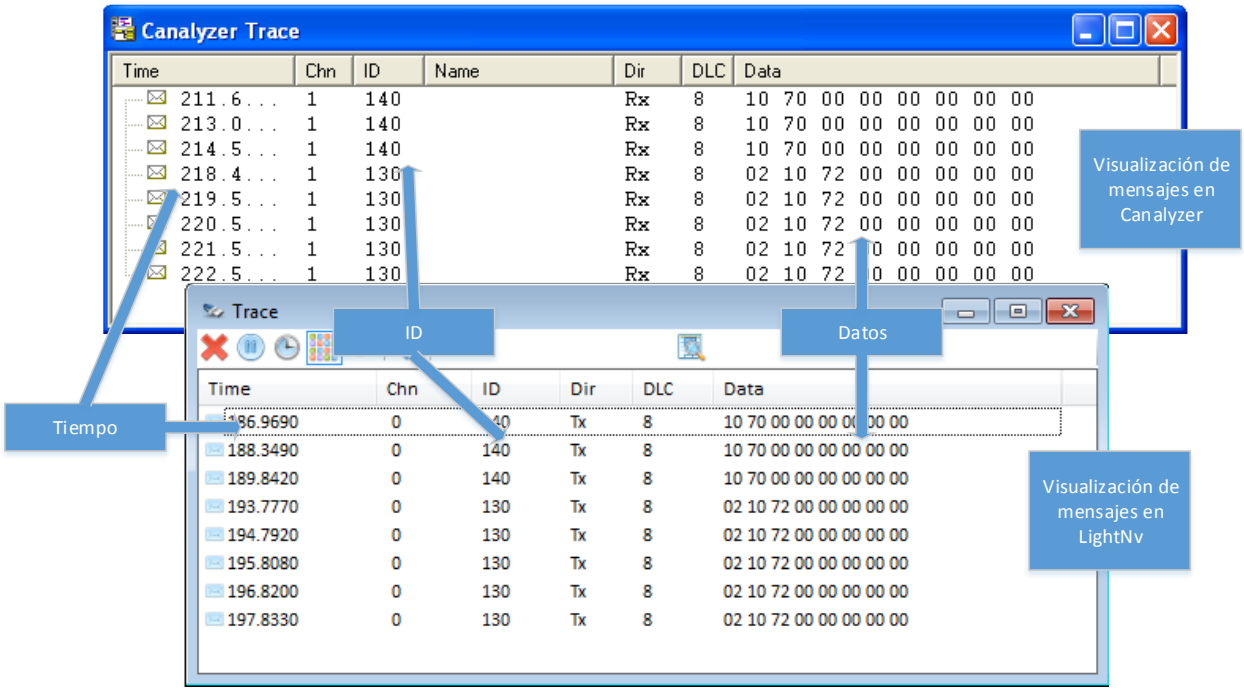


Figura 32: Registros de los mensajes enviados.

Como se puede observar en la imagen anterior, los mensajes fueron enviados correctamente a la red de CAN. Los primeros registros de ambas ventanas nos muestran el mensaje 140 el cual es enviado por medio del evento del botón, los registros siguientes nos muestra la información del mensaje 130 el cual es enviado de manera periódica cada segundo. Los tiempos en ambas ventanas no coinciden por el momento en el que se inició cada herramienta, pero la medición de cada registro es la correcta con una diferencia de un segundo.

4.2.4 Monitoreo de paquetes desde la red

En esta ventana el usuario puede analizar la información basándose en paquetes. Por medio de ella podemos visualizar en qué momento llegó el último bloque de datos correspondiente a los parámetros de configuración. Dichos parámetros pueden ser el identificador del mensaje o cualquiera de los bytes de datos con los que pueda llegar el mensaje, basta con poner el valor que se espera en el byte correspondiente para que después se haga una comparación de cada mensaje que llegue desde la red con el valor deseado. El número de paquetes permitidos para analizar depende de la capacidad de la memoria instalada en el sistema.

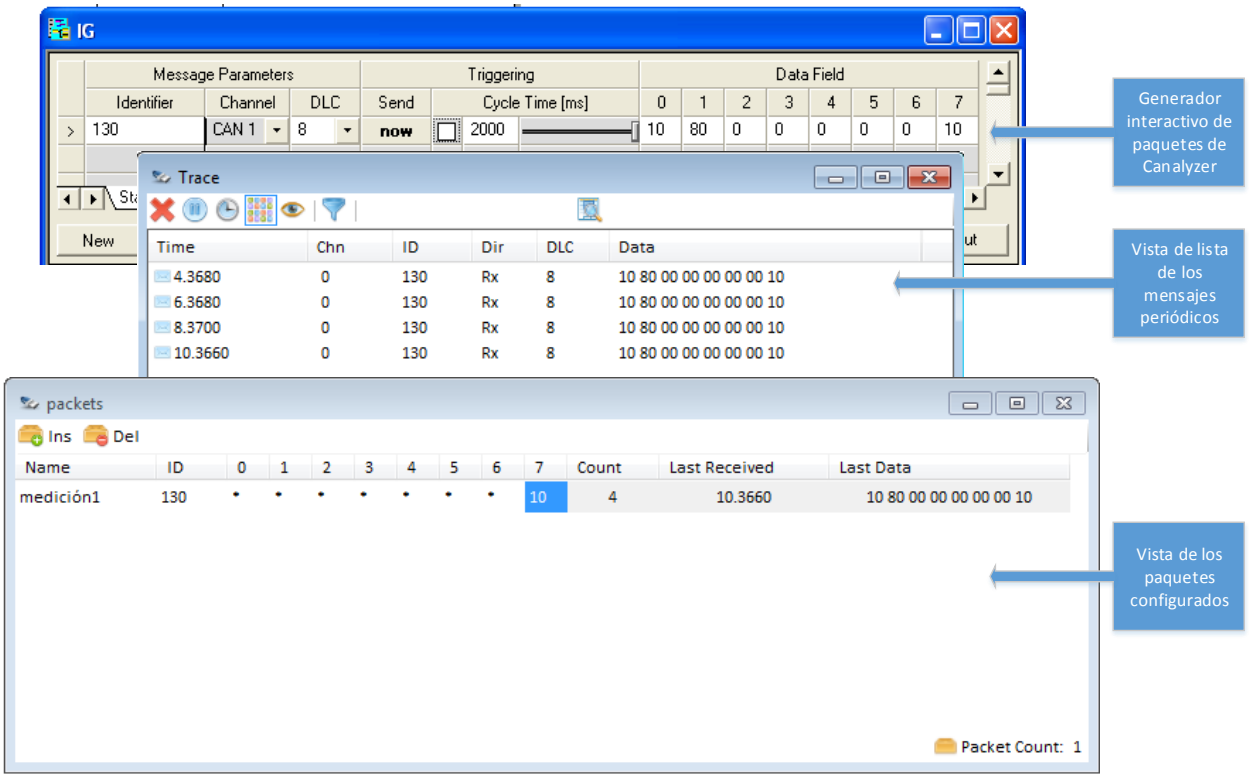


Figura 33: Recepción y procesamiento de paquetes desde la red.

En la figura superior se puede apreciar la configuración del paquete que queremos monitorear, en este caso se configuró con el nombre medición1, el identificador 130, los primeros 7 bytes no nos interesan, solo nos interesa el último byte cuando este tenga un valor de 10. Si dicho paquete ingresa a la red, la aplicación nos muestra la cantidad de paquetes con esas características, el tiempo y los datos del último mensaje recibido. En este caso nos muestra que el paquete ha ingresado 4 veces a la red, con un tiempo de 10.3660 y un paquete de datos 10 80 00 00 00 00 001 10, esto lo podemos corroborar en

la ventana de registro la cual nos despliega estos4 paquetes, el tiempo y los datos de cada uno de ellos. Este módulo es muy útil en los casos en los que se presenta un error de manera intermitente en las mediciones, nos indica en que momento del tiempo se presentó y de esa manera junto con el archivo de registro, podemos aislar las causas de dicho error.

4.2.5 Visualización grafica de las señales

Esta parte del sistema nos permite la visualización de los valores en decimal de cualquiera de los bits o conjunto de bits que conforman el paquete. Se tiene que definir un nombre de la señal, el identificador, el color, el bit de inicio y el offset, con estos datos se procesa cada uno de los mensajes que llegan a la red con el identificador definido. El sistema nos permite definir hasta 100 señales diferentes, cada una de ellas se muestra en el gráfico como una seria con el color definido.

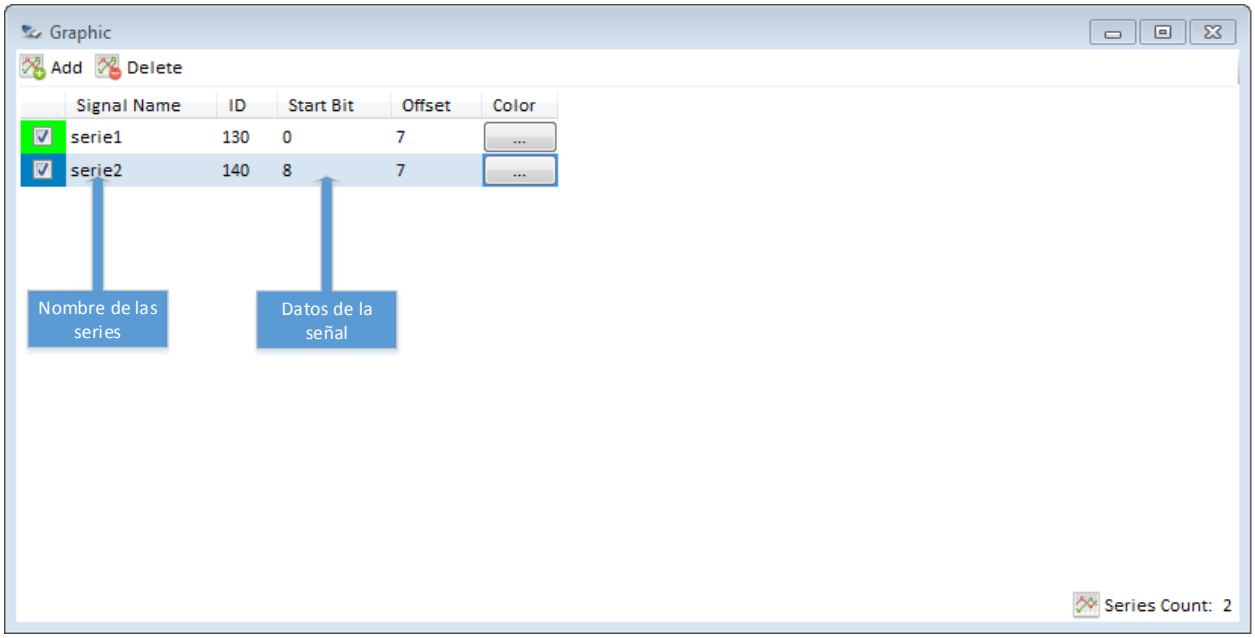


Figura 34: Definición de las series de la gráfica.

En la figura anterior se puede observar que se definieron una señal, esta contiene el identificador 130 con el nombre serie1, el bit de inicio es el 0 y el número de bits es 7, esto monitoreará el valor del primer byte del paquete y graficará dicho valor con una serie de color verde. Dada la naturaleza del sistema binario, los valores que el sistema puede graficar son solo valores enteros sin signo y sin valores decimales. Una vez que se definen las señales y se comienza con la medición, cada uno de los mensajes que llegan a la red son pasados como parámetros a este formulario, cada uno de ellos es comparado contra cada uno de los identificadores definidos, si alguno coincide, entonces se extraen los bits correspondientes y el valor obtenido se pasa como un punto a la gráfica para cada una de las series que conforman el grafico de señales.

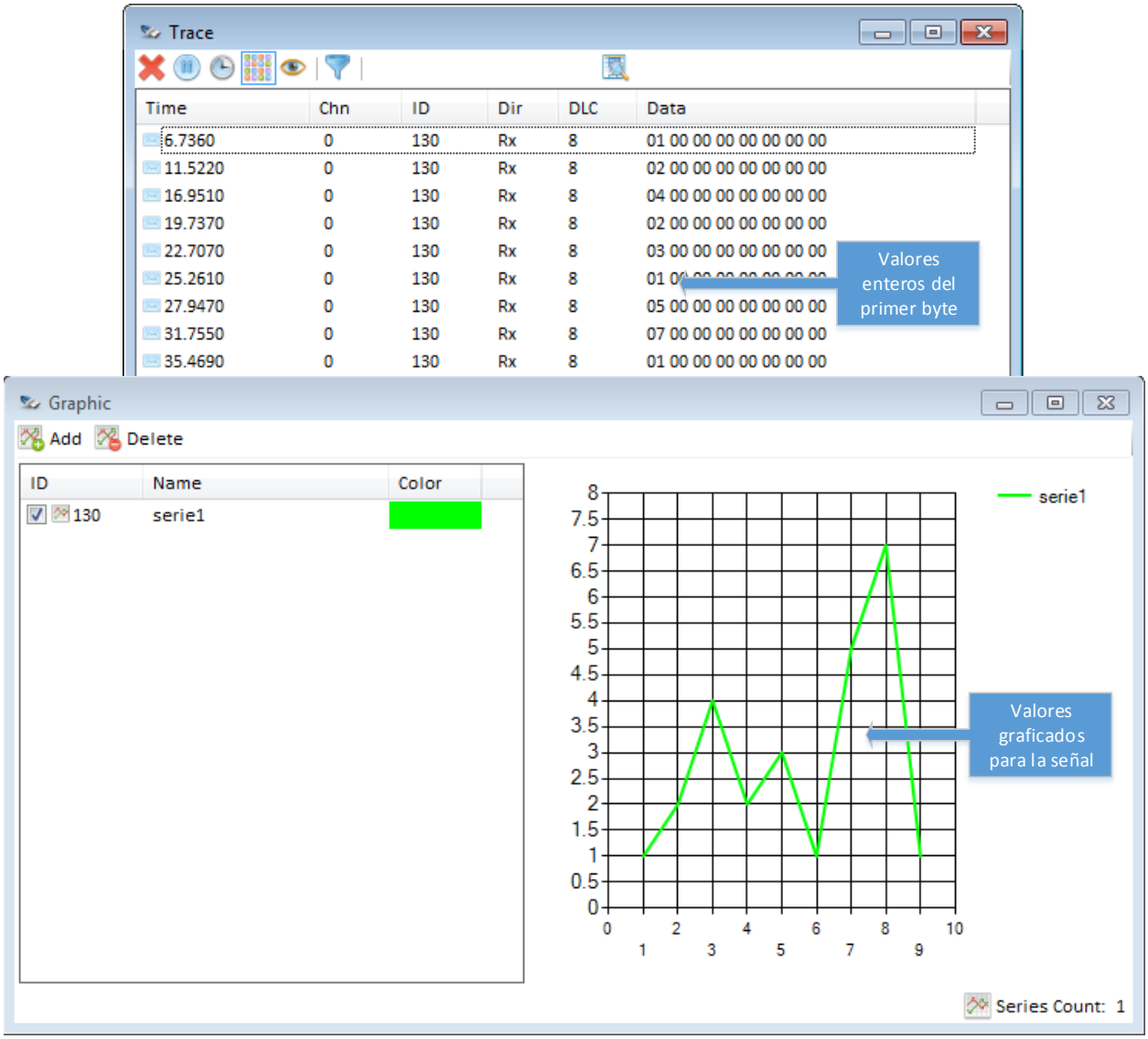


Figura 35: Visualización de los valores de la serie 1 para cada paquete.

En la figura anterior, podemos observar como los valores del primer byte de los paquetes está cambiando constantemente, además de visualizar el valor entero en la ventana de registro, también podemos ver como la serie del gráfico va tomado los valores correspondientes a la señal definida. Los valores que recibe la señal son: 01,02,04,02,03,01,05,07,01. Estos valores se pasan como puntos de la serie para cada uno de ellos.

4.2.6 Ventana de registro

Este módulo es uno de los más importantes del sistema, ya que por medio de el podemos visualizar los paquetes que entran y salen dese y hacia la red. La ventana que muestra los paquetes, cuenta con una vista de lista heredada desde los controles comunes de Windows. Esto fue necesario ya que el control original presenta un problema llamado flick, el cual se manifiesta con un borrado temporal en la lista, perceptible para el ojo humano.

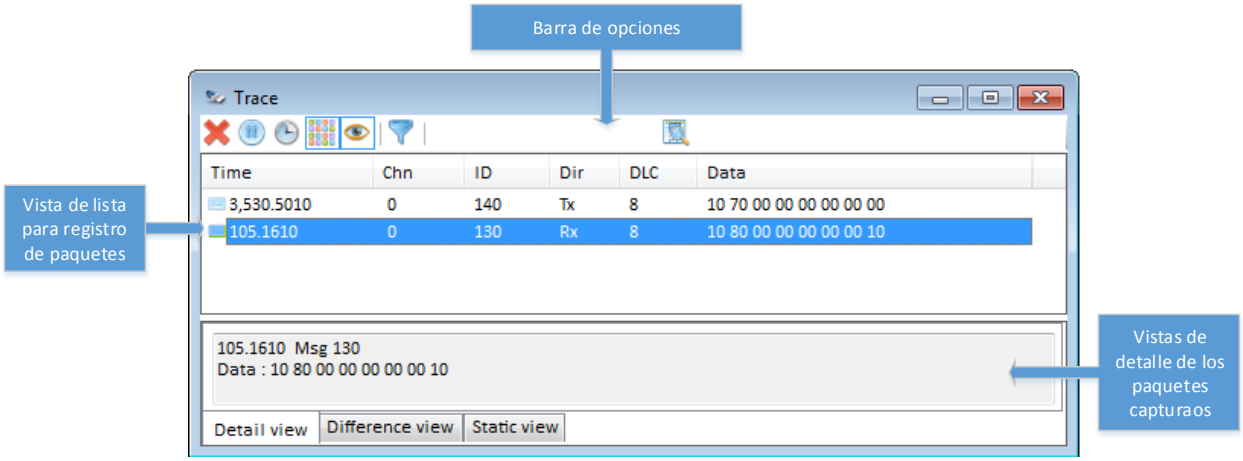


Figura 36: Ventana de registro de paquetes.

La ventana cuenta con una barra de opciones las cual nos muestra las tareas más comunes en el manejo del registro de paquetes, estas opciones nos permiten borrar la lista, pausar el registro de actividades durante un tiempo indefinido, cambiar la vista del tiempo del paquete, visualizar los detalles de cada mensaje en la parte inferior, filtrar los mensajes en base a una condición y además buscar un elemento dentro de la vista de lista.

Cada mensaje que entra a la red desde la librería de comunicación, se envía como parámetro a una función del formulario encargada de procesar la información y separar cada una de las partes que conforman el paquete. Estos datos se almacenan en un tipo de dato heredado desde listviewitem, al cual, se le agregaron algunas propiedades para poder guardar más información para cada uno de los elementos de la vista de lista. El número máximo de elementos que la vista de lista es capaz de almacenar, depende en gran medida de la cantidad de memoria RAM con la que cuenta el equipo de cómputo, esto es muy útil ya que algunas veces las mediciones de CAN se llevan a

cabo en varios días, en los casos en los que un error es muy difícil de encontrar. La vista de lista se limpia cada vez que el usuario inicia una nueva medición lo que hace que los datos que se encontraban con anterioridad se borren y se pierdan, para mitigar la perdida de datos también se cuenta con el archivo de registro, el cual almacena la misma información que se está procesando en la ventana de registro. Otra de las herramientas con los que cuenta este formulario, es la que nos permite seleccionar uno o varios elementos de la vista de lista y después copiarlos hacia el portapapeles de Windows o en su defecto, exportarlos al formato de valores separados por comas, este formato se puede visualizar en cualquier editor de texto o en Microsoft Excel.

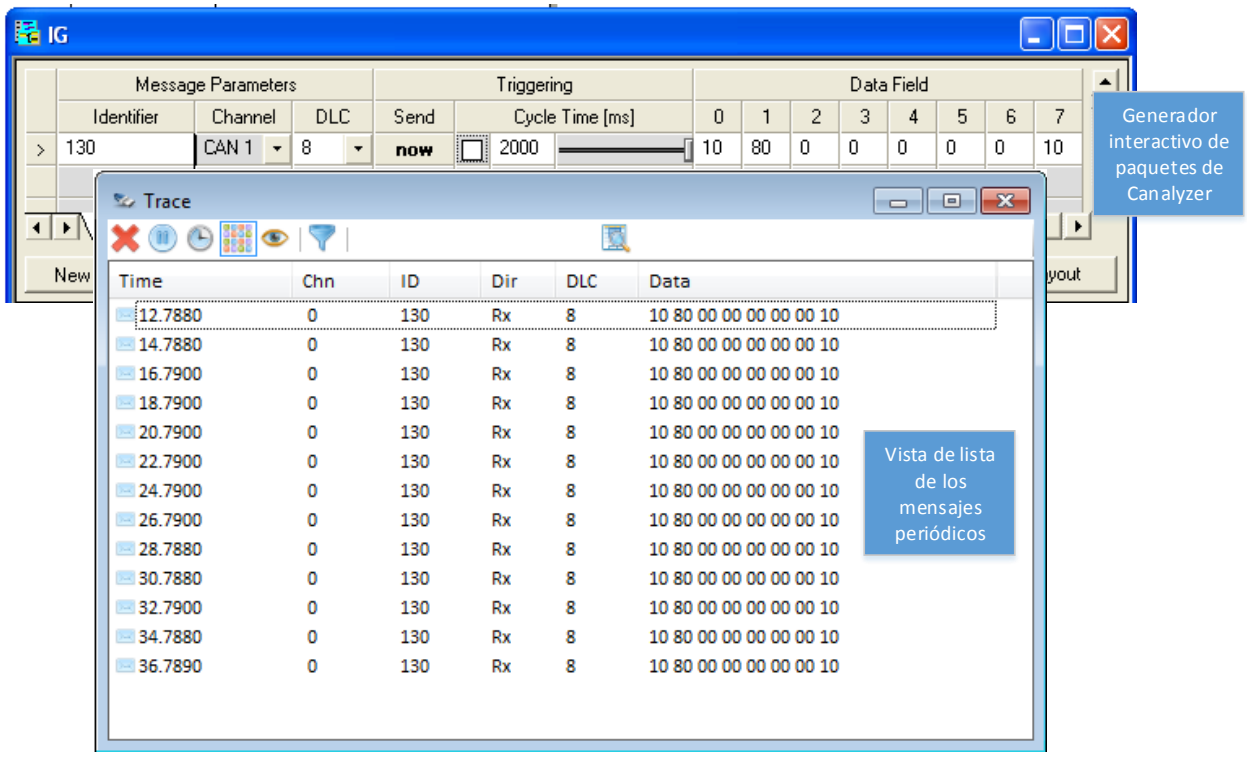


Figura 37: Registro de mensaje periódico enviado desde la red.

En el ejemplo de la imagen anterior, se utilizó el generador interactivo de Canalyzer para enviar un mensaje periódico cada dos segundos, como se puede observar en la ventana de registro de nuestro sistema, cada uno de los mensajes fue registrado con la misma diferencia de tiempo con las que fueron generados.

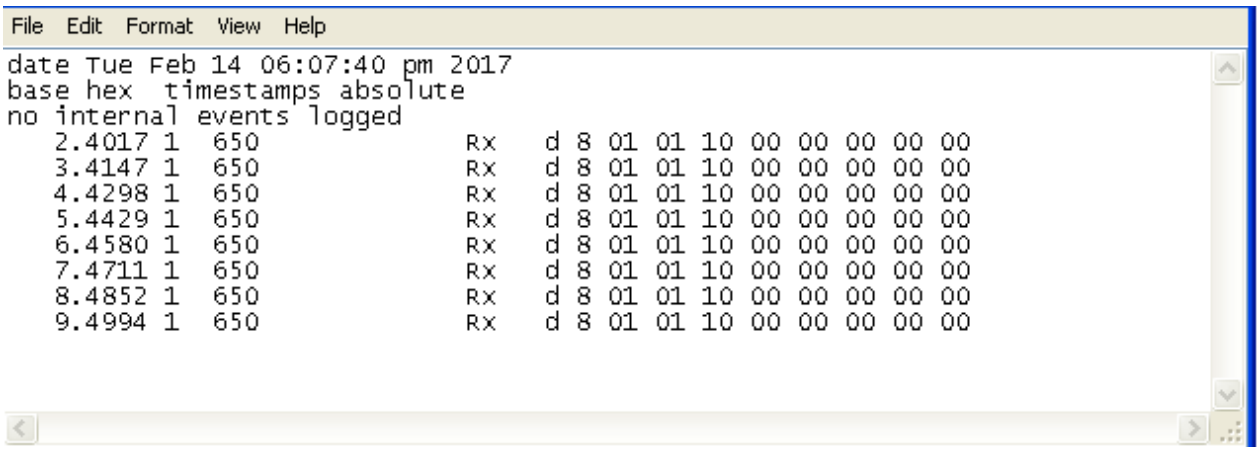


Figura 38: Archivo de registro en formato ASC.

En la imagen superior se muestra el archivo de registro de datos, el cual almacena los paquetes recibidos en un archivo de texto con extensión. ASC. Este tipo de archivos son los mismos que utiliza la aplicación Canalyzer para llevar a cabo su archivo de registro de datos, esto hace que los archivos generados por nuestra aplicación, sean totalmente compatibles con las herramientas de la empresa Vector.

4.2.7 Modelo 3D del equipo de pruebas

Figura 39: Modelo 3D del equipo de pruebas.

En la imagen superior se observa el modelo 3D del equipo de pruebas en el que se implementó el software, en la trasera se encuentran las 12 AVT para la comunicación CAN, una para cada unidad bajo prueba. En la parte frontal se encuentran los conectores DB9 para el monitoreo de la comunicación, solo se necesita conectar la unidad bajo prueba a cualquiera de las 11 AVT restantes. Por razones de confidencialidad no se utilizan fotos reales del equipo de pruebas ya que es propiedad de nuestro cliente.

4.2.8 Equipos para probar el sistema

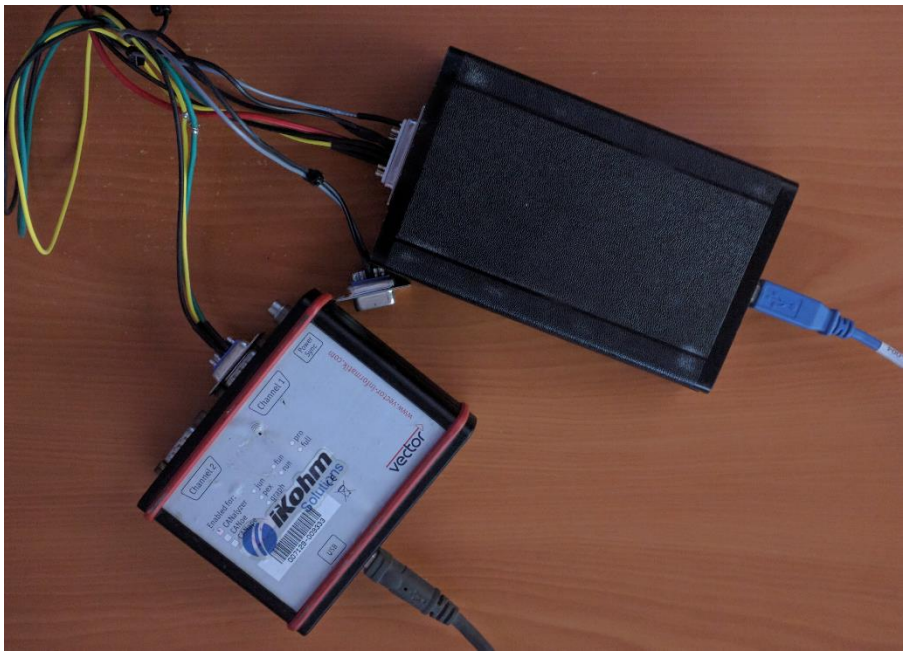


Figura 40: Equipos utilizados para las pruebas del sistema.

En la imagen se muestran ambos dispositivos utilizados para las pruebas de comunicación entre ambos sistemas, por un lado, el CanCase de la empresa vector y por el otro la AVT de la empresa avt-hq. Ambos dispositivos están conectador por medio un arnés que contiene las tres señales necesarias para la comunicación CAN: GND, CAN_H, CAN_L. Este arnés por un lado cuenta con un conector DB9 para conectarse con el CanCase y un conector DB15 para la conexión con la AVT.

5 CONCLUSIONES

En la hipótesis planteada se dijo que era posible utilizar el AVT-852 como herramienta de análisis del bus de CAN y que solo sería necesario instalar la aplicación desarrollada para que ésta pueda ser utilizada en equipos ya existentes. Esto fue totalmente posible gracias al desarrollo realizado de la aplicación. Esta aplicación permite reutilizar las AVT-852 ya conectadas a los equipos de prueba para realizar análisis de CAN, así como la integración en los equipos nuevos. Basta con crear un paquete de instalación el cual distribuya los componentes necesarios en la computadora en la que se utilizará, conectar el dispositivo al puerto USB de la computadora, seleccionar la dirección y los valores de configuración para que la aplicación pueda llevar a cabo el análisis deseado. Este desarrollo aumentó el acervo técnico de la empresa y de manera personal en la configuración y uso de este tipo de instrumentos, lo que nos permite reducir los tiempos de desarrollo en nuevos proyectos en los que sea necesario reutilizar este tipo de equipos. Además, aumentó la cartera de productos que la empresa ofrece a sus clientes y la disminución en los costos en el desarrollo de nuevas plataformas de prueba para los clientes. El desarrollo de esta aplicación, abrió una nueva oportunidad de negocios al ofrecer las herramientas desarrolladas además de los servicios para otras empresas en forma de cursos o desarrollo de librerías para este tipo de dispositivos, para que dichas empresas reduzcan la curva de aprendizajes en la utilización y programación de las AVT.

6 RECOMENDACIONES

Como se comentó anteriormente, las pruebas de nuestro sistema arrojaron que la librería con la que cuenta visual studio.net en especial vb.net para la lectura del puerto serial, pierde demasiado tiempo en cada una de las lecturas de los paquetes que se encuentran en el buffer utilizado para la recepción. Si en la aplicación que se va a desarrollar, el tiempo y la velocidad de lectura/escritura no es crítico para el manejo de los datos, se puede seguir utilizando dicha librería ya que es más fácil y rápida de implementar, pero si, al contrario, los tiempos en nuestra aplicación son muy importantes lo mejor utilizar las funciones que nos ofrece Windows, las cuales se explicaron anteriormente en el apartado de pruebas. Aunque estas funciones son un poco más complejas que las estándar, nos permite agilizar la lectura y escritura al puerto serial. Además, si queremos que nuestra aplicación cuente con más de una AVT monitoreando cada uno de los canales de CAN con los que cuenta nuestra red, la AVT-852 en algún momento será obsoleta ya que el equipo de cómputo nos pondrá la limitante del número de puertos USB disponibles. La mejor opción sería utilizar la AVT-853 que funciona de manera muy parecida, pero utiliza el protocolo TCP-IP, lo cual nos permitiría expandir el número de dispositivos utilizando un switch conectado a la tarjeta de red, así el número máximo de dispositivos conectado será el que el protocolo nos permita. Además de proporcionar mayor distancia y mayor tolerancia a fallos, también, las velocidades de lectura y escritura del puerto Ethernet superan por mucho a las velocidades alcanzadas por un puerto serial siendo de hasta 1000 Mbit/s utilizando la tecnología 1000 Base LX que contra los 14 kbit/s del serial. Debido a que la AVT-852 no cuenta con un driver propietario desarrollado para sistemas Windows y debe ser utilizada como un puerto serial lo recomendable es desarrollar la aplicación para que soporte ambos modelos y la selección del dispositivo dependa de la velocidad con la que son enviados los mensajes en la red de CAN.

7 REFERENCIAS BIBLIOGRÁFICAS

AXELSON, J. (2015). *USB Complete fifth edition.* Madison, Wisconsin: Lakeview Research.

AXELSON, J. (2015). *USB Complete Fifth Edition.* Madison, Winconsin: Lakeview Research.

CORRIGAN, S. (2008, Julio). *Introducction to the Controller Area Network (CAN).* Retrieved from www.ti.com: https://www.google.com.mx/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwj-9taY2ZDSAhUB32MKHZTVDf0QFggdMAA&url=http%3A%2F%2Fwww.ti.com%2Flit%2Fan%2Fsloa101a%2Fsloa101a.pdf&usg=AFQjCNGOJ4HpVpSftLR-I7-QNyXSoFQ9eA

D., C. (2015). *Encyclopedia of automotive engineering.* west sussex: John Wiley & Sons Ltd.

ETSCHBERGER, K. (2008). *Controller Area Network, Basics Protocols, Chips and Applications.* Leibnizstr: IXXAT Automation GmbH.

Inc., M. T. (2011, Febrero 15). *51848B.pdf*. Retrieved from http://ww1.microchip.com: http://ww1.microchip.com/downloads/en/DeviceDoc/51848B.pdf

M., D. N. (2012). *Understanding and Using the Controller Area Network Communication, Theory and Practice.* New York: Springer.

RILEY, M. (2017, Enero 22). *http://avt-hq.com/download.htm#AVT-852*. Retrieved from http://avt-hq.com: http://avt-hq.com/download.htm#AVT-852

VECTOR. (2015, Abril 1). *CANalyzer_ProductInformation_EN.pdf*. Retrieved from Vector.com: https://vector.com/portal/medien/cmc/info/CANalyzer_ProductInformation_EN.pdf

VOSS, W. (2005). *A Comprehensible Guide to Controller Area Network.* Greenfield, Mariland: Copperhill Media Corporation.

8 ANÉXOS