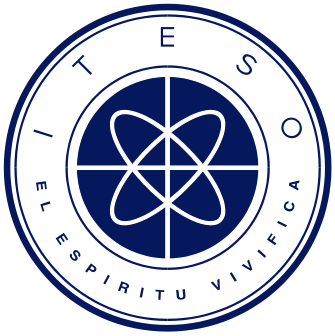
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



MEDIDOR DE VELOCIDAD DE PROTOCOLO CAN

Tesina para obtener el grado de:

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presentan: Feliciano Angulo Angulo
Manuel Zambrano Miranda

Director: Dr. José Luis Pizano Escalante

San Pedro Tlaquepaque, Jalisco. Julio de 2018.

Agradecimientos

Agradecemos a Dios el habernos permitido cursar esta especialidad, así como también a las instituciones que lo hicieron posible como, la universidad ITESO por proporcionar la infraestructura necesaria para el desarrollo de la Especialidad en Sistemas Embebidos.

De igual manera, agradecemos al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la asignación de las becas para la realización de nuestros estudios:
Feliciano Angulo Angulo: beca No. 856789
Manuel Zambrano Miranda: beca No. 860042

Damos gracias también al Dr. Luís Rizo Domínguez, por su excelente labor como coordinador de la especialidad, ya que con su motivación, consejos y recomendaciones encontramos la solución a todos los retos que se nos fueron presentando. Así mismo al Dr. José Luís Pizano Escalante, Mtro. Miguel Ángel Corral Chagolla y a la Dra. Lorena Michele Brennan Bourdon, quienes con sus recomendaciones ayudaron a guiar este trabajo.

De igual manera, no podemos dejar de agradecer a los profesores Francisco Martínez Chávez, Luís Garabito Siordia, César Alejandro Carrillo Jiménez, Sergio Nicolás Santana Sánchez, Héctor Rivas Silva y Carlos Arturo García Camarena. Les agradecemos especialmente por el esfuerzo, dedicación y calidad con que transmitieron su conocimiento y experiencia profesional durante el programa de especialidad.

Por último, queremos agradecer a nuestra familia, en especial a nuestros padres, por estar siempre a nuestro lado, por ofrecernos los medios para nuestra superación profesional y por su incansable apoyo para cada uno de nuestros proyectos, por esto y más les estaremos eternamente agradecidos.

Abstract

The technological advances of the automotive industry have a tendency of incorporating more electronic components, which are communicated with the CAN protocol. For this reason, these electronic devices require tools that help validate and diagnose their proper function. However, these tools can be expensive and often do not perform the desired task. For this reason, the objective of this project is to develop an algorithm that measures the communication speed of the BUS CAN protocol, which will serve as the basis for the development of a robust and economic tool for the CAN BUS data analysis. The developed algorithm was based on the measurement of the pulse width (bit timing) of the CAN BUS frame, which positively measured the BUS communication speed. In addition, other functions such as identification of data bytes from the sent message and its data identifier were added to the algorithm, resulting in a more complete solution than initially projected. To test the effectiveness of the algorithm, its logic sequence was coded in C language considering the layered software architecture known as AUTOSAR and its development in the FRDM-K64F board from NXP company.

Resumen

Los avances tecnológicos de la industria automotriz tienen una tendencia a incorporar más componentes electrónicos, que se comunican con el protocolo CAN. Por esta razón, estos dispositivos electrónicos requieren de herramientas que ayuden a validar y diagnosticar su correcto funcionamiento. Sin embargo, estas herramientas pueden ser costosas, y a menudo, no realizan la tarea deseada. Por esta razón, el objetivo de este proyecto es desarrollar un algoritmo que mida la velocidad de comunicación del BUS CAN" que servirá como base para el desarrollo de una herramienta robusta y económica para el análisis de datos del BUS CAN. El algoritmo desarrollado se basó en la medición del ancho de pulso (del inglés, bit timing) de la trama del BUS CAN, que mide positivamente la velocidad de comunicación del BUS. Además, otras funciones como la identificación de bytes de datos del mensaje y su identificador de datos fueron agregados al algoritmo, dando como resultado una solución más completa de lo inicialmente proyectado. Para probar la efectividad del algoritmo, su secuencia lógica fue codificada en lenguaje C considerando la arquitectura de software por capas conocida como AUTOSAR y su desarrollo en la tarjeta FRDM-K64F de NXP.

Lista de figuras.

Lista de tablas

Lista de abreviaturas y acrónimos.

ARM	Nombre de la empresa ARM del inglés, *Advanced RISC (Reduced Instruction Set Computer) Machine*
AUTOSAR	*Arquitectura de software (del inglés, **AUT**omotive **O**pen **S**ystem **Ar**chitecture)*
BUS	*Red de comunicación*
CAN	*Controlador de área local (del inglés, Controller Area Network)*
CANoe	*Software desarrollado por Vector para usar la herramienta VN1630A*
CMSIS-DAP	*Estándar de comunicación para microcontroladores (del inglés, Cortex Microcontroller Software Interface Standard)*
COM	*Puerto de comunicación (del inglés, Communication Port)*
CRC	*Comprobación de redundancia cíclica (del inglés, Cyclic Redundancy Check)*
DLC	*Código de longitud de dato (del inglés, Data Length Code)*
DMA	*Acceso directo a memoria (del inglés, Direct Memory Access)*
ECU	*Unidad de control electrónica (del inglés, Electronic Control Unit)*
ECU AL	*Capa de abstracción de datos de la unidad de control electrónica (del inglés, ECU Abstraction Layer)*
FPU	*Unidad de punto flotante (del inglés, Floating Point Unit)*
FRDM-K64F	*Tarjeta de desarrollo de bajo costo para microcontroladores Kinetis K64F*
FTM	*Flex Timer Module*
GPIO	*General Purpose Input Output*
IDE (bit)	*Bit de extension de identificador*
IDE	*Integrated Development Environment*
ID	*Identifier*
ISO	*International Organization for Standardization*
JTAG	*Joint Test Action Group*
Mbps	*Mega bit por Segundo (unidad de transmisión de datos)*
MCAL	*Microcontroller Abstaction Layer*
MCU	*Microcontrolador (del inglés, Micro Controller Unit)*
OpenSDAv2	*Adaptador para comunicación y depuración serial de uso libre (del inglés, Serial and Debug Adapter)*
OSI	*Modelo de Interconexión de Sistemas Abierto (del inglés, Open Systems Interconnection)*
PC	*Computadora personal (del inglés, Personal Computer)*
PCB	*Tarjeta de circuito impreso (del inglés, Printed Circuit Board)*
RAM	*Memoria de acceso aleatorio (del inglés, Random Access Memory)*
RTE	*Entorno de tiempo de ejecución (del inglés, Run Time Environment)*
RTR	*Solicitud de transmisión remota (del inglés, Remote Transmission Request)*
RX	*Línea de recepción (de la abreviatura del inglés, receive)*

SN65HVD230	*CAN Transceiver fabricado por Texas Instruments*
SPI	*Interfaz periférica serial (del inglés, Serial Peripheral Interface)*
SRR	*Solicitud de substitución remota (del inglés, Substitute Remote Request)*
SWC	*Componente de software (del inglés, Software Component)*
SWD	*Cable de depuración serial (del inglés, Serial Wire Debug)*
TX	*Línea de transmisión (de la abreviatura del inglés, transmit)*
UART	*Transmisor receptor asincrono universal (del inglés, Universal Asynchronous Receiver Transmitter*
USB	*BUS serial universal (del inglés, Universal Serial Bus)*
VN1630A	*Hardware desarrollado por Vector para conectar una PC en una red CAN*

Contenido

Introducción.

En la actualidad, principalmente en la industria automotriz, es indispensable el uso del protocolo CAN (del inglés, *Controller Area Network*) [1], para la comunicación entre los dispositivos que controlan el funcionamiento de un sistema completo (automóvil, en el caso de la industria automotriz).

CAN es un protocolo serial desarrollado por Bosch [2] en 1983, derivado de la necesidad de reducir los costos y complejidad de las conexiones entre los dispositivos de los automóviles. A estos dispositivos se les conoce como ECU (por sus siglas en inglés, *Electronic Control Unit*) y, actualmente, un vehículo contiene en promedio 35 ECUs comunicándose entre sí para controlar diferentes funciones [3].

Debido a la naturaleza del estándar del protocolo CAN, cualquier dispositivo puede conectarse o desconectarse de la red en cualquier momento sin causar ningún problema a los ECUs que ya se encuentran conectados. Dicha característica puede ser aprovechada al momento de realizar tareas, tanto en el desarrollo de nuevos prototipos como para realizar pruebas de diagnóstico de comunicación a una red ya conformada.

Al conectar un nuevo dispositivo en la red, la siguiente acción es configurar el controlador de CAN para que pueda enviar y/o recibir mensajes. Dicha configuración requiere conocer la velocidad a la cual todos los dispositivos del BUS están comunicándose.

En la actualidad existen dispositivos capaces de conectarse a una red CAN para su análisis. La mayoría de ellos a precios muy elevados y poco accesibles para instituciones educativas o proyectos de empresas pequeñas; además, dichos dispositivos no permiten realizar una conexión automática, sino que requieren ciertos pasos previos por parte del usuario para definir la velocidad con la cual se van a conectar en el BUS.

Es por esto que el presente proyecto está enfocado en resolver la problemática, al diseñar y desarrollar un algoritmo para determinar la velocidad a la cual se realiza la comunicación en las redes de bus CAN, con el fin de facilitar las labores, tanto a desarrolladores como a técnicos de pruebas, y conocer la velocidad a la cual debe conectarse un nuevo dispositivo a una red CAN de velocidad desconocida y así poder iniciar un proceso de análisis y diagnóstico.

El algoritmo para la detección de velocidad del protocolo CAN tiene la ventaja de ser implementado en la mayoría de las arquitecturas de microcontroladores y tarjetas de desarrollo para el mercado de sistemas embebidos. Por lo tanto, será parte de un acervo de investigación para el desarrollo de una herramienta rápida y sencilla en la prueba y medición de velocidad de tráfico de datos en sistemas con comunicación basados en redes CAN.

El objetivo es crear un algoritmo que calcule la velocidad en una red CAN, que sirva como base para el desarrollo de una herramienta de medición económica y versátil para el análisis y validación de un sistema conectado a una red CAN.

1. Antecedentes.

El objetivo es la implementación de un algoritmo para calcular la velocidad de un BUS de CAN, para usarse como base en algún sistema de diagnóstico de redes CAN.

Actualmente en el mercado ya existe una infinidad de herramientas destinadas al diagnóstico y validación de redes de BUS CAN, así como muchos proveedores, marcas y modelos, en capacidades y precios muy variados que se pueden comprar incorporando esta funcionalidad (detección de velocidad de BUS CAN). Tal sería el caso de las herramientas de *Vector Informatik GmbH* [4], con el hardware *VN1630A* [5] en conjunto con el software *CANoe* [6] que proveen un medio de comunicación con las redes CAN y para lo cual ofrecen la posibilidad de escanear la velocidad en un BUS de CAN antes de conectarse a dicha red.

Sin embargo, tomando en cuenta que las herramientas de *Vector* [4] proveen una amplia gama de características y funcionalidades, además de detectar la velocidad en un BUS de CAN, también cuentan con la capacidad de lectura de multi tramas, depuración y separación de mensajes, validación de los elementos conectados, validación por inyección de mensajes erróneos, pruebas de sistema a través del BUS CAN, etc. Pero a un precio muy elevado y poco accesible para empresas pequeñas, universidades, profesionistas independientes o emprendedores.

Además de que no existe información detallada de cómo es que dichas herramientas logran el escaneo de velocidad de BUS CAN, resulta inviable la inversión en una herramienta con tal costo y funcionalidades adicionales, si la mayoría de estas no serán utilizadas.

Otra herramienta similar a las de *Vector* [4] es el hardware *CANUSB* fabricada por *LAWICEL* [7], que es menos costoso que las antes mencionadas de *Vector* [4], con menos capacidades, dado que no ofrece un software para facilitar el análisis de los datos en un BUS CAN como *CANoe* [6], ya que solo brinda una interfaz USB con un puerto COM para enviar diferentes comandos a través de este último. Dichos comandos se encuentran en el manual de usuario [8], y de acuerdo a los cuales no existe una forma de identificar automáticamente la velocidad del BUS,

sino que se debe indicar mediante un comando la velocidad a la cual se pretende establecer la conexión.

Por otro lado, existen otras opciones como la alternativa de *Microchip* [9], la herramienta *CAN BUS Analyzer Tool* [10] que se comercializa como un producto fácil de usar, de bajo costo que permite desarrollar y depurar redes CAN de alta velocidad. La herramienta ofrece, además del hardware para la conexión física a la red, una interfaz de usuario no tan avanzada como la de *Vector CANoe* [6], pero permite realizar varias cosas como monitorear todos los mensajes de la red, hacer filtros, transmitir mensajes, crear logs y agrupar mensajes CAN de trasmisión. Todas estas características la hacen muy versátil ya que permite depurar las redes CAN fácil y rápido [11]. Sin embargo, *CAN BUS Analyzer* [10]. tampoco permite detectar la velocidad de una red, por lo que, al igual que las herramientas citadas, se debe configurar manualmente la velocidad con la cual se quiere conectar.

Las herramientas disponibles en el mercado para el análisis de redes CAN son muchas, pero pocas proveen la capacidad de detectar su velocidad, de las aquí mencionadas solamente lo hace *Vector* [4] y su software *CANoe* [6]. Sin embargo, por su costo elevado es inviable la inversión para la detección de la velocidad.

2. Marco Teórico.

El marco teórico bajo el que se sustenta nuestro proyecto involucra temas de gran importancia, que deben ser mencionados antes de abordar el tema objetivo que es ¿Cómo desarrollar un algoritmo para la detección de velocidad del protocolo CAN? Y para lo cual se tocarán los siguientes temas:

- Modelo OSI.
- Protocolo CAN.
- Arquitectura AUTOSAR.

Modelo OSI

El Modelo de Interconexión de Sistemas Abierto, mejor conocido como "Modelo OSI" (*Open System Interconnection*, por sus siglas en inglés) [12], es un modelo que fue creado en 1980 por la "Organización Internacional de Normalización", conocida como "ISO" (*International Organization for Standardization*, por sus siglas en inglés) [13] y publicado en 1984 por el mismo organismo. Es un modelo de referencia que es tomado para el desarrollo de protocolos de comunicación que quieren ser normalizados como parte de un standard.

El modelo OSI no explica la composición de un protocolo de comunicación en específico, es más bien un modelo general que describe la arquitectura bajo la cual se recomienda la construcción de dicho protocolo. El modelo es detallado con base a siete diferentes capas o niveles de software, que tienen como finalidad la transmisión de información de forma segura y sin errores desde un punto origen hasta un punto destino.

Las capas que componen el modelo OSI cumplen con el propósito de la separación de tareas, para hacer del protocolo un método de comunicación modular, confiable y de fácil mantenimiento. Cada una de estas tareas a su vez está compuesta por uno o más procesos llamados servicios, los cuales se encargan de resolver la tarea asignada a esa capa [12]. En la figura 2-1 se

puede observar una representación gráfica de las siete capas del modelo OSI, desde la capa superior (capa de aplicación), hasta la capa inferior (capa física).

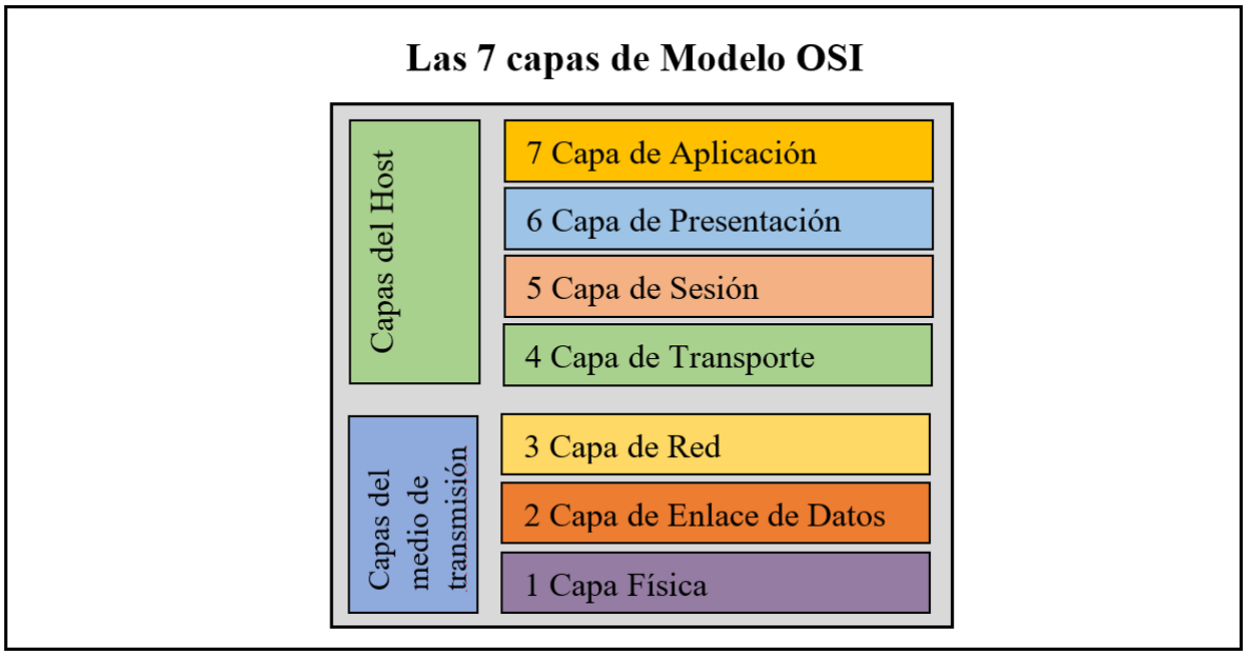


Fig. 2- 1. Representación gráfica de las siete capas del modelo OSI.

Capa física

Para el estándar del modelo OSI esta es la primera capa, y es la encargada de definir el medio por el cual se transmitirán los datos, ya sea inalámbrico, por medio de cables o fibra óptica. También define los materiales y las características mecánicas que deben tener el medio conductor en cuestión y sus conectores, así como la cantidad de hilos del mismo. De igual manera aquí también se definen los niveles de las señales eléctricas y frecuencias de operación [12].

Capa de enlace de datos

La segunda capa se encarga, como su nombre lo dice, de gestionar el acceso de los datos al medio de transmisión; les asigna direcciones para que lleven un orden, para facilitar la transmisión de datos de un nodo a otro. Así mismo, define métodos para la detección de errores y asegurar que la información llegue de forma correcta a su destino [12].

Capa de red

La capa de red es la tercera capa del modelo OSI y se encarga de definir las rutas de red para una o más redes y, de esta manera, asegurar que los datos lleguen desde el origen a un destino. Gracias a este mecanismo es que se pueden definir las direcciones de dos o más nodos en una o más redes de ellos, permitiendo con esto la comunicación entre nodos, aunque estos no estén conectados físicamente [12].

Capa de transporte

La cuarta capa del modelo OSI es la capa de transporte y su tarea principal es la de facilitar el transporte de datos de un nodo a otro, mediante la separación de la información en forma de paquetes de datos, asignándoles un orden para su identificación y posterior envío. Este método facilita el envío de grandes cantidades de datos de forma ordenada, para que posteriormente el nodo destino los pueda identificar y volver a acomodar en su forma original [12].

Capa de sesión

La quinta capa del modelo OSI se encarga de proporcionar un servicio denominado de "autenticación", es decir, que una vez que se ha establecido una conexión entre dos nodos de red, la mantienen por medio de una contraseña que asegura un canal de comunicación seguro entre ambos nodos, tanto emisor como receptor [12].

Capa de presentación

La sexta capa del modelo OSI tiene como principal tarea la de tomar los datos y traducirlos a un formato que sea común y entendible para todos los nodos de la red. Este servicio de traducción puede incluir un proceso de cifrado y compresión de los datos y, de esta manera, facilitar el proceso de transferencia [12].

Capa de aplicación

La séptima y última capa del modelo OSI, la capa de aplicación es la encargada de dar la presentación y el tratamiento adecuado a los datos para que el usuario final puede interpretar la información de forma adecuada, ya que hace las funciones de un intérprete entre el usuario y los servicios de las capas inferiores [12].

Mecánica de trabajo del modelo OSI

La mecánica de funcionamiento del modelo OSI está compuesta por una serie de tareas distribuidas en cada capa, cada una de estas tareas a su vez puede estar conformado por uno o más servicios encargados de completar dicha tarea. Los servicios realizados entre tareas es lo que permite el intercambio de datos de una capa hacia otra [12].

Cada una de las capas toma los datos de la capa anterior y le agrega información adicional (datos de control) característica de esa capa. A este proceso se le llama encapsulación de datos. Una vez que los datos han pasado por cada una de las siete capas, son enviados al nodo destino, donde este a su vez va retirando los datos de control que le fueron anexados en su contraparte, realizando esta operación en cada capa, y obteniendo como resultado final la información original a como se tenía antes de ser enviada [12].

Este proceso de encapsulamiento y des encapsulamiento es lo que permite la transmisión de datos de un nodo a otro. En la figura 2-2 se muestra la forma en la que un nodo origen, definido como “Nodo A”, encapsula la información para su posterior transmisión hacia un nodo destino, denominado como “Nodo B”.

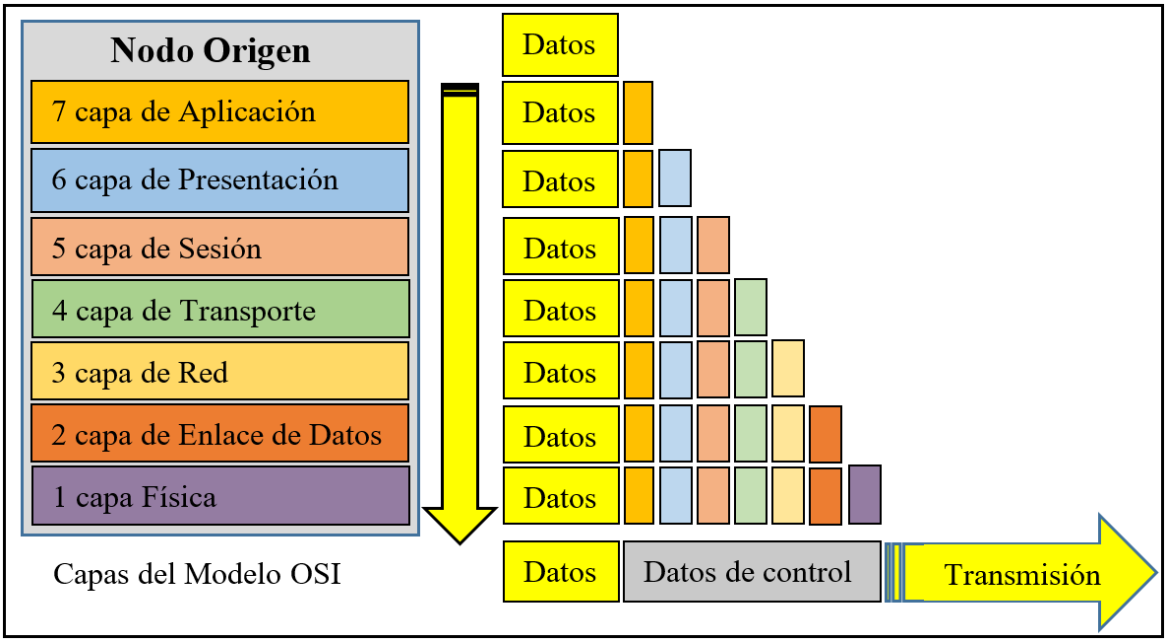


Fig. 2- 2.Proceso de encapsulado de datos en el nodo origen "Nodo A".

Por otro lado, el nodo destino "Nodo B" se encarga de hacer el proceso inverso con la información recibida (des encapsulado de datos). En la figura 2-3 se muestra el proceso de des encapsulado de los datos recibidos.

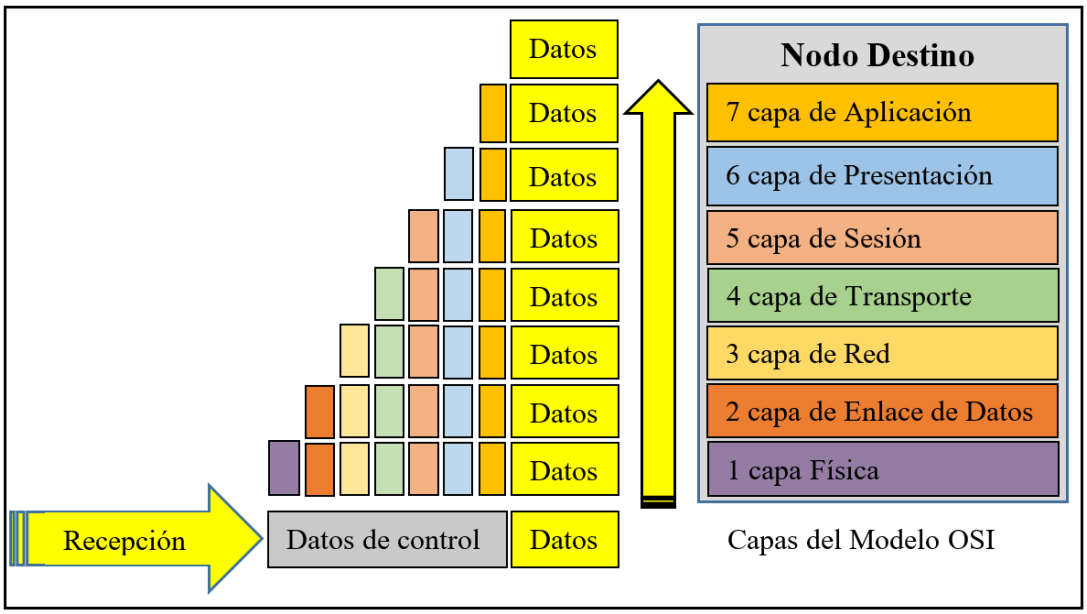


Fig. 2- 3. Des encapsulado de datos provenientes del "Nodo A" por parte del "Nodo B" (recepción de datos).

Protocolo CAN

El protocolo de Control de área Local (*Controller Area Network*, por sus siglas en inglés), mejor conocido como protocolo CAN [1], es un protocolo de comunicaciones desarrollado por la empresa alemana Robert Bosch GmbH [2], la cual comenzó su desarrollo en el año 1983, lanzando su publicación en 1986. Posteriormente los fabricantes de semiconductores Intel [14] y Philips [15] lanzaron al mercado en 1987 los primeros chips controladores de CAN destinados para adaptar las señales de este nuevo protocolo.

Posteriormente Bosch fue liberando nuevas versiones del protocolo CAN, haciendo mejoras y adecuaciones del mismo. En 1991 se liberó la versión CAN 2.0, la cual consta de dos partes, la versión CAN 2.0A conocida como la versión estándar de CAN y que está compuesta por identificadores de mensajes de 11 bits, y la versión CAN 2.0B que es conocida como la versión extendida de CAN y está compuesta por identificadores de 29 bits.

Debido al éxito y aceptación del protocolo CAN por parte de la mayoría de los fabricantes de vehículos automotores fue que en 1993 se publicó el estándar ISO 11898 [1]. Pero a medida que se han ido haciendo mejoras y adecuaciones con el paso del tiempo, este estándar se ha extendido, llegando a componerse de varios documentos que describen diferentes partes y aspectos del mismo. A continuación, se muestra un listado de los documentos que conforman el estándar, así como la fecha de su última actualización y una reseña de su contenido.

- ISO 11898-1: 2015: Señales de capa física y capa de enlace [1].
- ISO 11898-2: 2003: Unidad de acceso al medio para alta velocidad [16].
- ISO 11898-3: 2006: Baja velocidad, tolerancia a fallos, interfaz dependiendo el medio [17].
- ISO 11898-4: 2004: Comunicación y su tiempo de disparo [18].
- ISO 11898-5: 2007: Unidad de acceso al medio de alta velocidad en modo de bajo consumo [19].
- ISO 11898-6: 2013: Unidad de acceso al medio de alta velocidad con funcionalidad de arranque selectivo [20].

En las últimas actualizaciones del estándar ISO 11898-1 [1] se anexan los detalles sobre la más reciente versión de CAN, denominada como CAN FD (*flexible data-rate*). Esta última versión de CAN es más rápida que el clásico CAN 2.0, ya que CAN FD puede transmitir datos a más de 1Mbps llegando hasta los 2Mbps. El estándar CAN FD es compatible con el clásico CAN 2.0 ya que los nodos capaces de transmitir con este estándar pueden leer y transmitir datos con el estándar del CAN 2.0, pero los módulos que usan el estándar CAN 2.0 no son compatibles con CAN FD.

Trama de CAN.

Como se mencionó anteriormente, una trama de CAN 2.0 tiene dos versiones, una estándar (CAN 2.0A) y una extendida (CAN 2.0B). Para saber que versión de CAN 2.0 es la que se está leyendo hay que averiguar si el identificador del mensaje es de 11 o de 29 bits. En la figura 2-4 se puede apreciar una representación de las dos versiones de CAN 2.0, con los bits más representativos de cada trama.

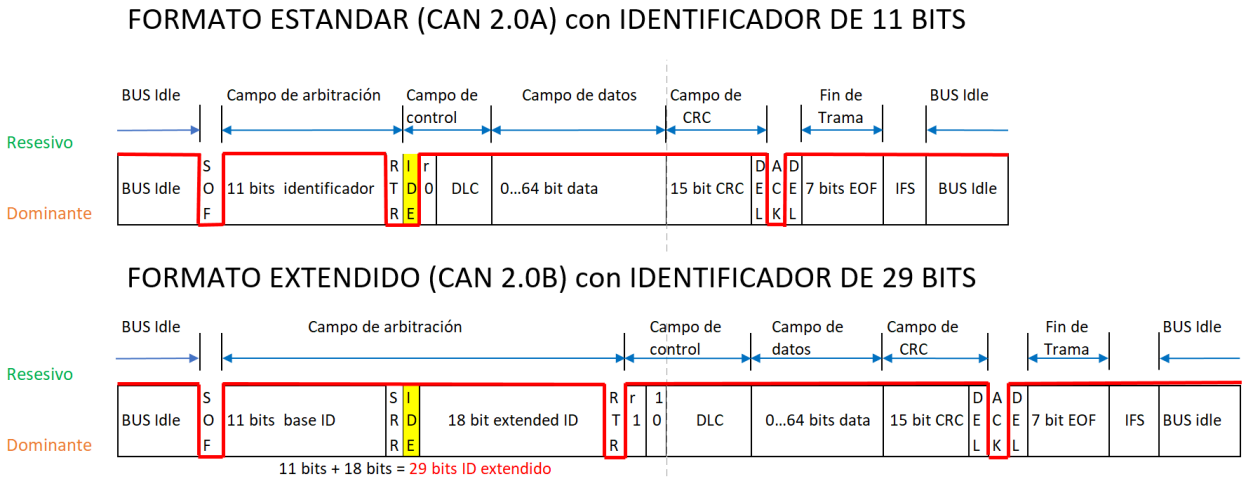


Fig. 2- 4. Tipos de tramas para CAN 2.0

Para saber de qué tamaño es el identificar de la trama, el estándar de CAN define que al terminar la trama en la parte del identificador de 11 bits.

Sí el bit IDE se encuentra en estado dominante, la trama será definida como del tipo estándar, por el contrario, si este bit se mantiene en estado recesivo, la trama se convierte a la versión extendida. De igual manera, se puede observar que la trama de CAN 2.0 en sus dos versiones está compuesta por otros elementos, los cuales se explican en las tablas 2-1 y 2-2.

Tabla 2-1. Campos que forman la trama de CAN estándar.

Nombre del campo	Longitud (bits)	Descripción
Bit de inicio	1	Marca el inicio de la transmisión.
ID	11	Identificador (único) y también representa la prioridad de la trama.
RTR	1	Petición de transmisión remota: Dominante (0) para tramas de datos, y Recesivo (1) para tramas de peticiones remotas.
IDE	1	Bit de extensión de identificador: Dominante (0) para el formato estándar (identificador 11 bits) y Recesivo (1) para el formato extendido (identificador 29 bits).
r0	1	Bit reservado: debe ser Dominante (0), pero también es aceptado como Recesivo (1).
DLC	4	Código de longitud de datos: define el número de bytes de datos en el mensaje, entre 0 y 8. Si este campo es mayor que 8, el mensaje será de 8 de cualquier forma.
Campo de datos	0-64 (0-8 bytes)	Datos de la trama (la longitud del campo viene dada por el código de longitud de datos o DLC).
CRC	15	Verificación por redundancia cíclica. Código que verifica que los datos fueron transmitidos correctamente.
Delimitador CRC	1	Debe ser recesivo (1).

ACK	1	Bit de reconocimiento: El transmisor emite recesivo (1) y cualquier receptor emite dominante (0).
Delimitador ACK	1	Debe ser recesivo (1).
Fin de trama EOF	7	Debe ser recesivo (1).

Como se mencionó anteriormente, si el bit IDE permanece en estado recesivo (1), nos indica que la trama tendrá un identificador extendido de 29 bits y, por lo tanto, el bit RTR pasa a ser nombrado como SRR, el cual es explicado en la tabla 2-2.

Tabla 2-2. Campo extra que forman la trama de CAN extendido.

Nombre del campo	Longitud (bits)	Descripción
SRR	1	Bit de substitución de transmisión remota: debe ser Recesivo (1)

Existen muchas otras características que distinguen al protocolo CAN, pero para efectos prácticos del actual proyecto no serán mencionados, ya que están fuera del objetivo planteado.

Arquitectura AUTOSAR

La "Arquitectura de Sistemas Abierta para el Sector Automotriz", mejor conocida como AUTOSAR (***AUT****omotive* ***O****pen* ***S****ystem* ***AR****chitecture*, por su acrónimo en inglés) [21], es una arquitectura de software abierta y estandarizada. Esta arquitectura de software fue desarrollada en acuerdo por diferentes empresas del ramo automotriz, de entre los cuales se destaca la participación de fabricantes de automóviles, proveedores de auto partes y desarrolladores de herramientas de software [22].

La asociación AUTOSAR ésta formada por cinco diferentes grupos, cada grupo formado en base a su categoría, estas categorías se definen de la siguiente manera:

- "**Socios Núcleo**" (*Core Partners*): este grupo ésta integrada por nueve empresas que son las fundadoras de la asociación AUTOSAR [23].
- "**Socios Premium**" (*Premium Partners*): este grupo hace uso de los estándares AUTOSAR, colaborando con los grupos "Socios Núcleo" y "Socios Desarrolladores" para definir y dar mantenimiento a los estándares AUTOSAR en sí mismo [24].
- "**Socios Desarrolladores**" (*Development Partners*): este grupo hace uso de los estándares AUTOSAR, colabora con los grupos "Socios Núcleo" y "Socios Premium" para definir y dar mantenimiento a los estándares AUTOSAR en sí mismo [25].
- "**Miembros Asociados**" (*Associate Partners*): este grupo está formado por miembros que usan de forma activa los estándares AUTOSAR más recientes [26].
- "**Socios Asistentes**" (*Attendeed*): en esta categoría los socios también colaboran con las otras categorías para definir los propios estándares de AUTOSAR [27].

En las categorías mencionadas los socios actualmente hacen uso de los estándares y además colaboran entre sí para definir los estándares de AUTOSAR.

La unión de todos estos fabricantes se llevó a cabo con el objetivo de crear y establecer estándares abiertos para arquitecturas de componentes electrónicos en el sector automotriz. Y de esta manera sentar las bases para proveer de una infraestructura básica para el desarrollo de módulos, interfaces de usuario, control para distintos dominios, etc.

Además de estandarizar funciones básicas de sistema, esto facilita la portabilidad tanto de componentes, módulos y arquitecturas de software de una variedad de fabricantes, a las diferentes variantes de vehículos y plataformas. Y de igual manera permitiendo la portabilidad e integración de múltiples proveedores y el mantenimiento de nuevas versiones del software a lo largo del ciclo de vida del vehículo.

Es importante mencionar que AUTOSAR está diseñado bajo en el concepto de arquitectura por niveles. Dicha arquitectura cuenta con tres principales niveles:

- Software de aplicación.
- Runtime environment.
- Software básico.

Entre estos niveles de software, el "Software de aplicación" es el que contiene todas las funcionalidades. El de "Runtime Environment" trabaja como un bus virtual, ya que es la interfaz común de comunicación entre el nivel de "Software de aplicación" y el nivel de "Software básico". Ya por otra parte, el nivel de "Software básico" es el encargado de procesar los mensajes de entrada y salida, directamente con el hardware (registros del micro controlador). En la figura 2-5 se observa una representación de la arquitectura por capas de AUTOSAR [21].

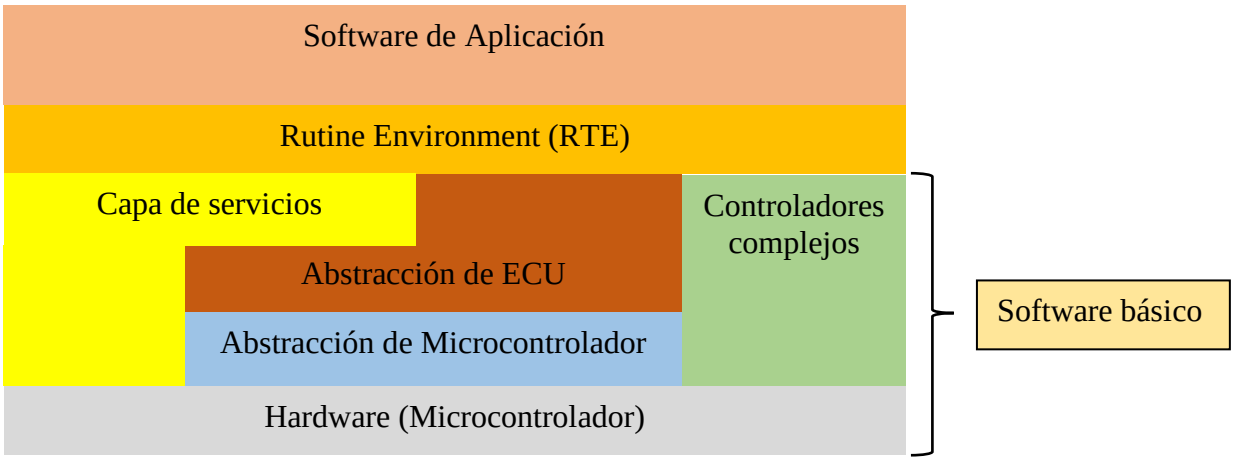


Fig. 2- 5. Diagrama básico de la arquitectura por niveles.

3. Metodología.

Hardware utilizado.

NXP FRDM-K64F Freedom

Se utilizó la tarjeta de desarrollo FRDM-K64F Freedom de la firma NXP [28], la cual es parte de un set de herramientas de hardware y software para evaluación y desarrollo. Esta es ideal para hacer prototipos de aplicaciones embebidas, es pequeña pero robusta, ya que su microcontrolador está basado en la arquitectura de ARM® Cortex ® -M4 core [29].

La FRDM-K64F [30] está basada en hardware embebido open-source, ofrece varias opciones de comunicación serial, programación flash y depuración. Incluye interfaz CMSIS-DAP, que permite comunicarse con la PC a través del puerto USB para cuestiones de programación y depuración [31]. En la figura 3-1 se pueden observar detalles de la interfaz.

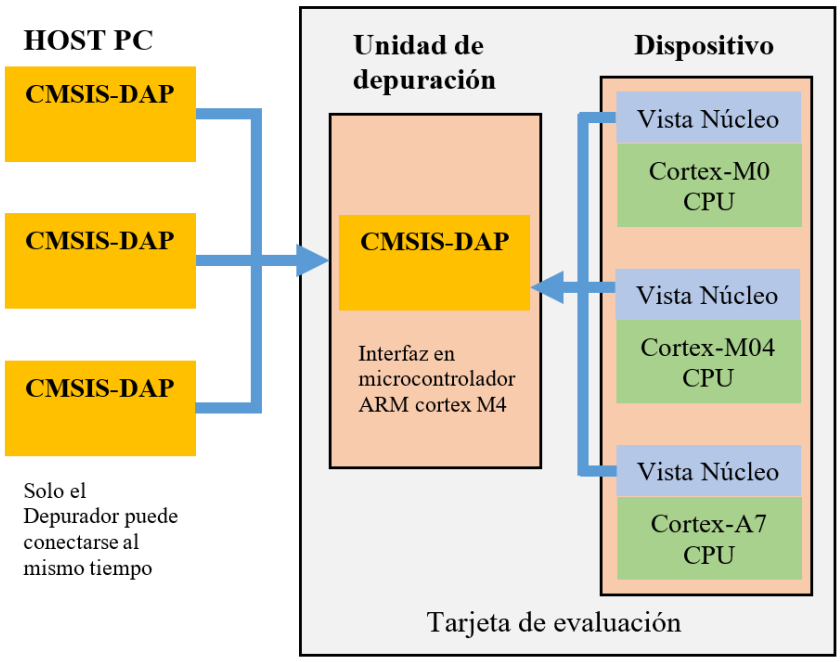


Fig. 3- 1. CMSIS_DAP interfaz.

El microcontrolador que viene con esta tarjeta es el MK64FN1M0VLL12 [32], el cual es un Microcontrolador ARM® Cortex®-M4 core con unidad de punto flotante FPU. Entre muchas de sus características se mencionarán a continuación aquellas que se consideran útiles para un proyecto de este tipo.

- Tiene un gran performance además de contar con una unidad de punto flotante.
- Memoria flash de 1 MB y RAM de 25 KB
- Múltiples modos bajo consumo y un controlador de DMA de 16 canales.
- 2 ADC y 2 DACs
- USB, Ethernet, CAN, UART y SPI para interfaces de comunicación.
- *Flex Timer* modules con 2 u 8 canales.

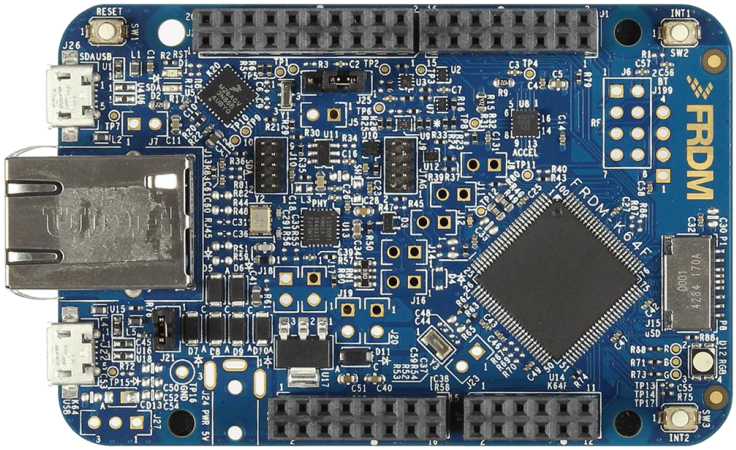


Fig. 3- 2. Vista general de la tarjeta con todos sus componentes.

CAN transceiver

Se utilizó un transceiver de CAN para la adquisición de señal desde un bus de CAN, este nos permite conectarnos en un bus de voltaje diferencial y obtener una señal digital 0-1(0 V – 3.3 o 5 V). Se podría utilizar casi cualquier transceiver de CAN disponible en el mercado, pero después de una búsqueda para seleccionar el más adecuado se optó por usar el VP230 CAN transceiver

[33], el cual se compró directamente en AMAZON [34] en un PCB que ya viene montado el chip listo para usar. En la figura 3-3 se muestra una imagen del empaquetado.

Fig. 3- 3. PCB con transceiver de CAN

Características del transceiver

- Opera con 3.3 V.
- Compatible con el standard ISO 11898-2 [16], High Speed CAN Physical Layer standard (transceiver).
- Soporta velocidades de transmisión arriba de 1 Mbps.
- Ideal para aplicaciones automotrices, robótica, control, etc.

Software utilizado

MCUXpresso IDE

Se utilizó MCUXpresso IDE v10.1.1 [Build 606] [2018-01-02] para el desarrollo del código en lenguaje C. [35]

MCUXpresso IDE es un ambiente de desarrollo fácil de usar basado en Eclipse para microcontroladores ARM Cortex-M [29]. MCUXpresso IDE tiene un modo avanzado de edición, fácil compilación y depuración. La tarjeta usada en este proyecto se conecta fácilmente con este

IDE para programarse y depurarse. La figura 3-4 muestra la vista principal que se tienen en la ventada del MCUXpresso IDE. [35]

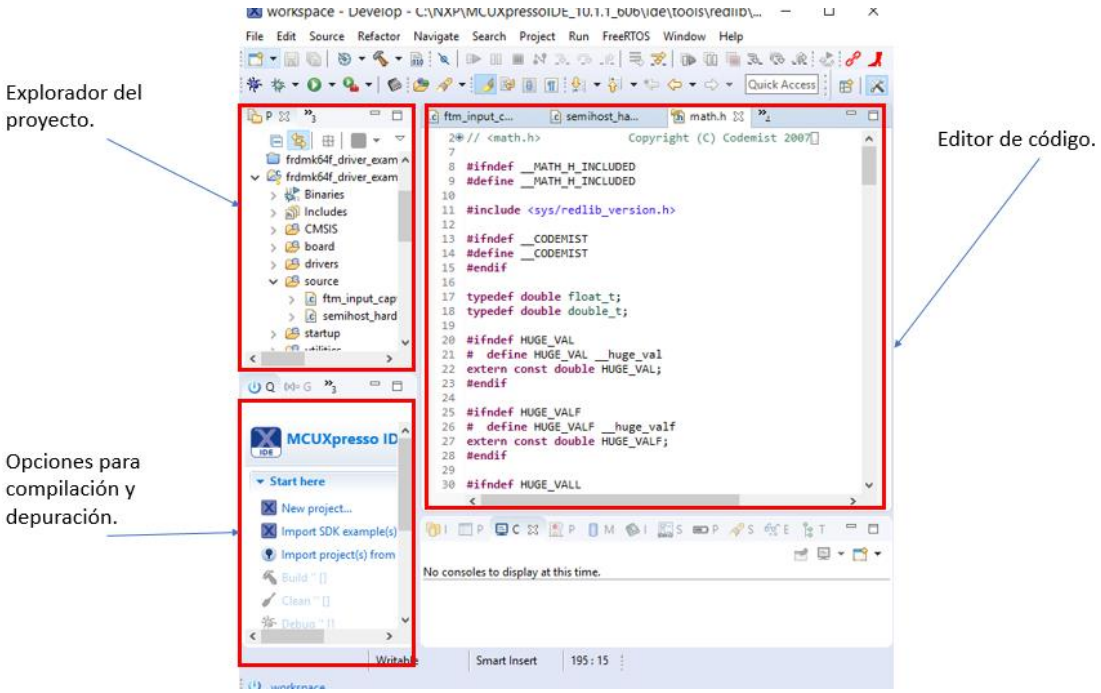


Fig. 3- 4. Ventana principal del MCUXpresso IDE.

Este mismo IDE cuenta con un asistente que permite crear ejemplos de cómo usar los diferentes periféricos y módulos del microcontrolador en la tarjeta FRDM-K64F, por lo cual fue de gran ayuda para entender cómo funciona el módulo *Flex Timer*, ya que nos ayuda a cumplir con el objetivo de nuestra aplicación que es capturar los flancos de bajada y subida en la línea RX del bus de CAN.

Tera Term

Tera Term [36] es un programa de licencia libre para la emulación de diferentes terminales, entre ellas una terminal serial, razón por la cual se está utilizando este programa para poder conectarnos a través de un puerto COM de la computadora con el puerto serial de la tarjeta para

recibir y enviar datos por UART. En la figura 3-5 se puede apreciar como este programa es una simple ventana donde es posible enviar y recibir datos.

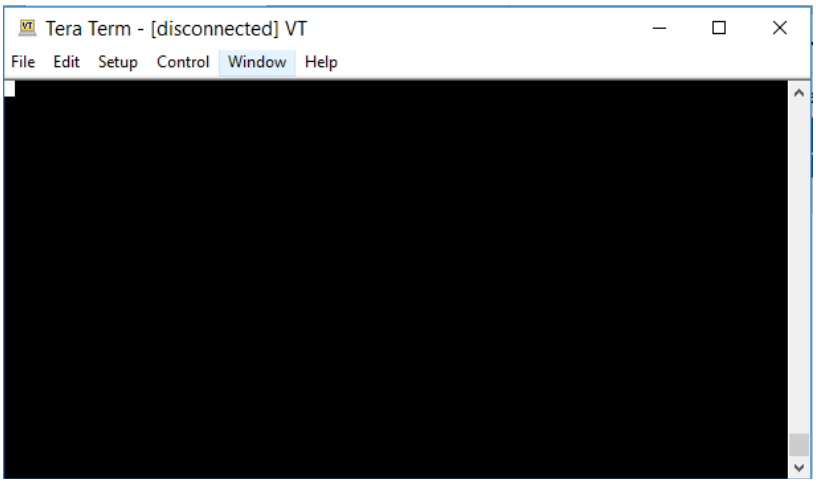


Fig. 3- 5. Ventana principal del Tera Term.

Implementación

Como solución se propone la captura de los flancos de subida y bajada de la línea RX en el transceiver de CAN, el cual debe de estar conectado a un BUS funcional de CAN para poder recibir la señal. La figura 3-6 muestra cómo se puede obtener la señal digital a partir de un voltaje diferencial de un bus real de CAN con un transceiver de CAN, en dicho transceiver las líneas CANH y CANL deben de estar conectadas al bus de CAN y la línea RX se conecta a un GPIO del microcontrolador, adaptando la señal para realizar las capturas de los flancos [1].

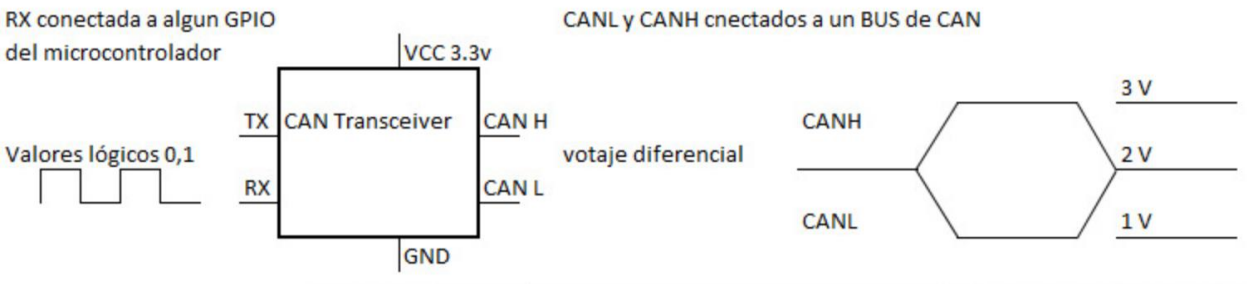


Fig. 3- 6. Perspectiva del transceiver en la solución.

Dicha captura de flancos se realiza usando el módulo *Flex Timer* del microcontrolador, este módulo se explica en la sección 3.3.1.

Flex Timer

El microcontrolador K64 posee un módulo de *Flex Timer* de 2 a 8 canales, dicho módulo tiene varias funcionalidades como PWM, comparador de salidas y capturador de entradas [32], siendo esta última la funcionalidad que nos ayuda a capturar los flancos de la línea RX en el transceiver.

En la figura 3-7 se observa cómo se debe conectar el transceiver con la tarjeta que estamos usando; basta con alimentarlo con el voltaje correcto y conectar la línea RX en el puerto PTC1(conexión mostrada en color naranja).

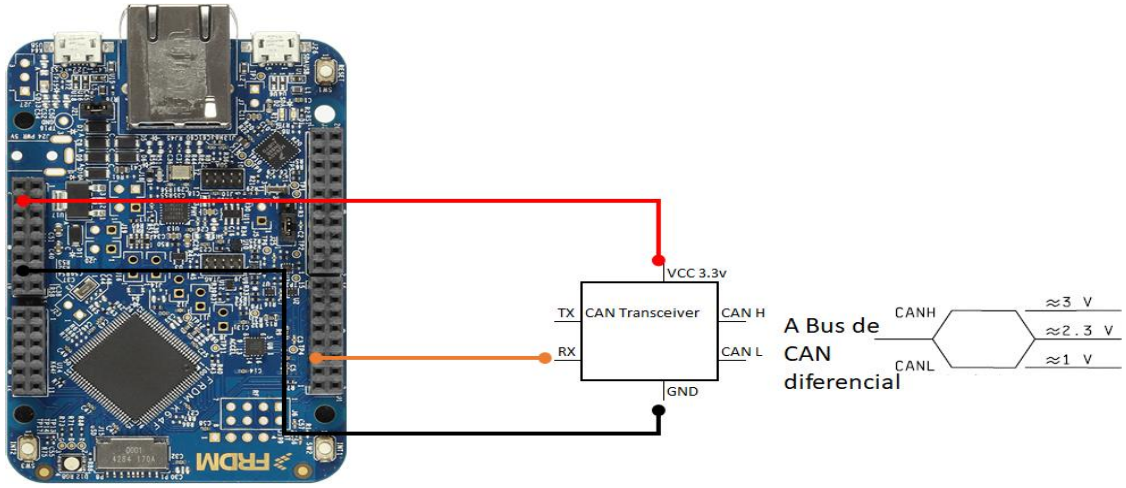


Fig. 3- 7. Conexión del transceiver con la tarjeta.

Partiendo de la conexión mostrada en la figura 3-7 ya se puede empezar a hablar de cómo programar el microcontrolador para capturar los flancos de subida y bajada en RX, esto será descrito en la sección 3.3.2.

Algoritmo

Se implementó un algoritmo para obtener el *baude rate* de un bus de CAN a partir de la medición del *BitTiming* (tiempo de bit), y además como medida de robustez, decodificar la trama de CAN para validar que se trata de un bus de CAN en donde estamos conectados.

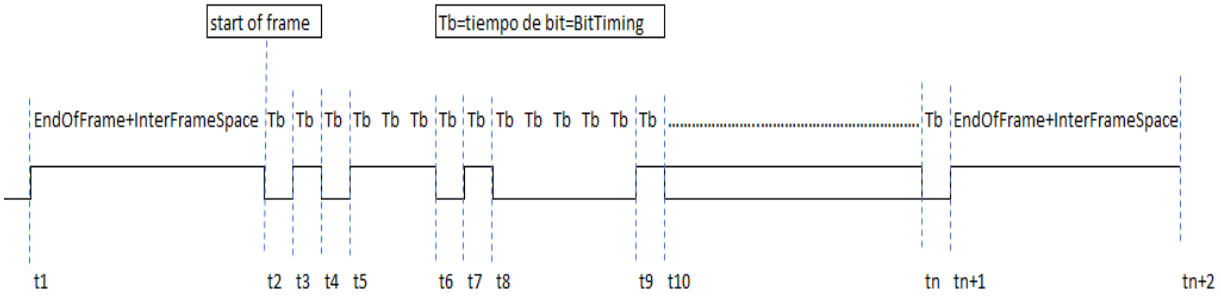


Fig. 3- 8. Ejemplo de una trama de CAN.

Los pasos para lograr el cálculo toman en cuenta las características mostradas en la figura 3-8, y se describen de la siguiente manera:

1. Capturar el tiempo en el que ocurre cada flanco de subida o bajada, en la figura 3-8 están definidos como t_1, t_2, …, t_n, t_{n+1} y t_{n+2}.

2. Ya con los tiempos capturados se puede calcular la duración de los anchos de pulso, esto se hace restando $t_2 - t_1$, $t_3 - t_2$, y así sucesivamente, por lo que se puede generalizar la formula a (3-1):

$$\boldsymbol{AnchoDePulso} = t_n - t_{n-1} \qquad (3\text{-}1)$$

3. Comparar todos los anchos de pulsos calculados para determinar cuál fue el menor y definir este como el tiempo de bit (*BitTiming*). Estas comparaciones se pueden hacer al mismo tiempo que se están calculando los anchos de pulso en el paso anterior.

4. En este punto, con el valor del tiempo de bit disponible ya es posible calcular la velocidad del BUS de CAN con la fórmula (3-2):

$$\boldsymbol{BaudRate} = \frac{1}{BitTiming} \tag{3-2}$$

   Sin embargo, nada nos asegura que las medidas hechas hasta el momento corresponden a un bus de CAN, por lo que el siguiente paso será validar que se ha capturado una trama de CAN.

5. Identificar todos los pulsos que correspondan a una trama de CAN. Para lograr esto, se buscan anchos de pulsos mayores o iguales a 13 *AnchosDeBit*, y los valores que se encuentren entre dos pulsos con las características anteriores corresponderán a una trama de CAN, incluyendo parte del pulso mayor o igual a 13 *AnchosDeBit* que se encuentre a la derecha.

6. Convertir todos los anchos de pulsos seccionados del paso anterior en los valores de bit correspondiente, esto partiendo de que el primer bit corresponde al *StartOfFrame* (valor en dominante), y cada nuevo pulso se le asignara el valor contrario al pulso anterior (después de dominante sigue recesivo y después de recesivo sigue dominante).
   a. Cando que un pulso es mayor a el ancho de bit calculado previamente, se debe de convertir en tantos bits como éste pulso sea múltiplo de ancho de bit, de acuerdo con la regla del protocolo. Este pulso podrá contener desde 1 bit hasta 5 bits, debido a la regla del *bit stuffing*.
7. Eliminar el *bit stuffing* con base al estándar del protocolo CAN [1], ya que en una trama no se pueden transmitir más de 5 bits consecutivos en un mismo estado, ya sea, recesivo o

dominante, para lograr esto se usa el *bit stuffing*. Esto es, después de haber transmitido 5 bits en un mismo estado se debe transmitir un bit en estado contrario a los 5 anteriores antes de continuar con la trasmisión del siguiente bit de la trama. Sabiendo lo anterior, se deben de ignorar los *bits de stuffing* para poder obtener únicamente los bits que realmente corresponden al mensaje original.

8. Obtener todas las partes del mensaje a partir de todos los bits que se han calculado hasta el momento. Con base en la figura 3-9 se definen las siguientes partes como las más importantes:
    - ID
    - Data Length Code (DLC)
    - Datos, 0 a 8 Bytes depende del DLC
    - CRC

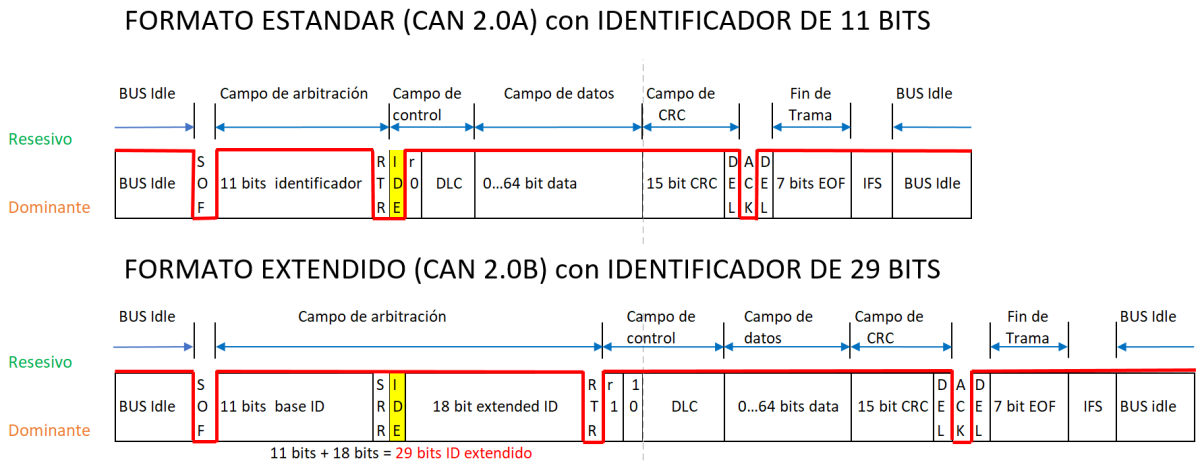


Fig. 3- 9. Formatos de la trama de CAN.

7.1 Identificar la posición de los bits RTR e IDE para saber si se trata del formato estándar o extendido:

Formato estándar: RTR (bit 12) en dominante, IDE (bit 13) en dominante.
Formato extendido: SRR (bit 12) en recesivo, RTR (bit 32) en dominante, IDE (bit 13) en recesivo.

7.2 Extraer el valor del identificador tomando los bits de las posiciones correspondientes de acuerdo con el resultado del paso anterior:

Formato estándar: Bits (1-11).
Formato extendido: Bits (1-31) omitiendo los bits de las posiciones 12 y 13.

7.3 Tomar el valor de DLC con base en si el identificador fue estándar o extendido:

Formato estándar: Bits (15-18).
Formato extendido: Bits (35-38).

El valor de DLC debe de estar en el rango de 0 a 8.

7.4 Luego de obtener el valor correcto del DLC se deben de extraer los valores de todos los datos de acuerdo con el número del DLC:

7.5 El siguiente elemento importante es la obtención del CRC, sin embargo, se optó por no calcularlo, pero para obtenerlo basta con tomar los 15 bits inmediatos después del último bit de dato.

Después de describir el algoritmo es momento de realizar la implementación en código. Dicha implementación es descrita en la sección 3.3.3.

Código: descripción de las funciones.

Para la implementación del código se decidió utilizar un enfoque de arquitectura de AUTOSAR ya que este es uno de los estándares más conocidos en la industria automotriz. El objetivo principal de esta arquitectura es una implementación lo más independiente posible del tipo de microcontrolador usado.

En la figura 3-10 del lado izquierdo se pueden ver las principales capas en AUTOSAR y del lado derecho la implementación de estas capas en nuestro proyecto. Cada capa inferior provee

una serie de interfaces (funciones) que las capas superiores pueden usar para hacer alguna petición. En las siguientes secciones se describirán a grandes rasgos las funciones usadas en cada capa y para más detalles de su implementación pueden referirse al código disponible en el repositorio del proyecto ***CAN Baude Rate Calculator*** [37].

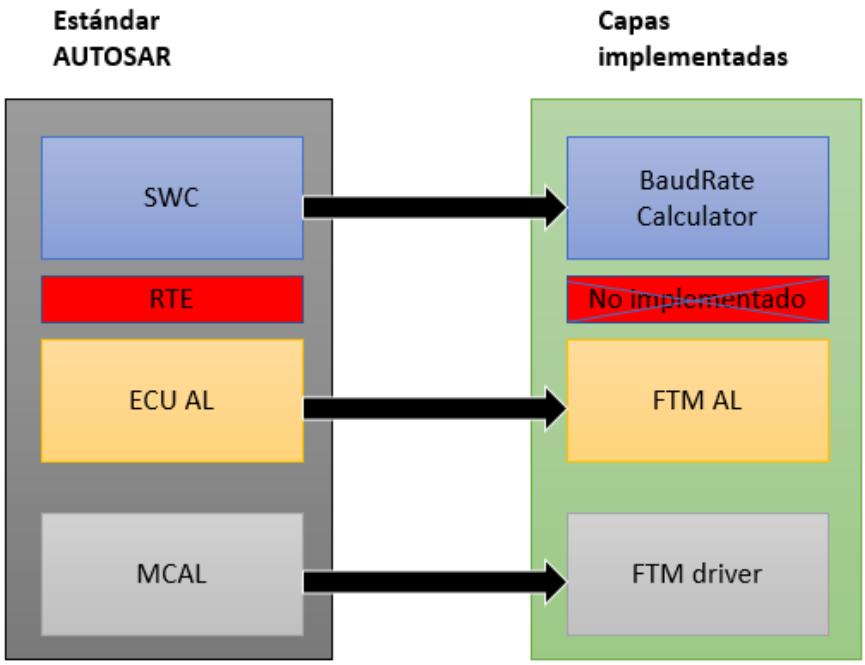


Fig. 3- 10. Capas de software AUTOSAR.

Baude Rate Calculator

Esta es la capa de aplicación, independiente completamente del hardware utilizado para calcular la velocidad de CAN después de obtener una lista de datos capturado y previamente adaptados por las capas inferiores. La figura 3-11 contiene las funciones disponibles en esta capa tanto públicas como privadas, todas descritas de la siguiente manera:

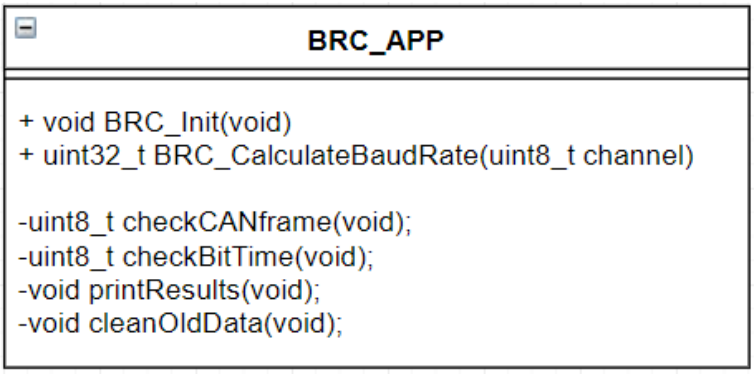


Fig. 3- 11. Funciones de la capa BRC_APP.

+ void BRC_Init(void).

Permite inicializar la aplicación y hace el llamado de inicialización de la capa inferior.

+ uint32_t BRC_CalculateBaudRate(uint8_t channel).

Inicia el proceso del cálculo del *Baud Rate* llamando a la capa inferior para que inicie captura de pulsos y le regrese los datos en un arreglo, el valor de cada dato en el arreglo representa un pulso de tiempo expresado en microsegundos. Ya con los datos en el arreglo hace el llamado a las diferentes funciones privadas para analizar los datos y calcular la velocidad del BUS CAN.

-uint8_t checkCANframe(void).

Analiza los datos para implementar los pasos del algoritmo descrito en la sección (3.3.2). Se apoya de la función "checkBitTime" para obtener el tiempo de bit.

-uint8_t checkBitTime(void).

Hace una iteración sobre el arreglo que contiene todos los anchos de pulsos para extraer el menor y definirlo como el tiempo de bit.

-void printResults(void).

Imprime los resultados del cálculo de velocidad, tales como, *Bit Time*, *Baud Rate* y detalles de una trama de CAN capturada (ID, DLC y Datos).

-void cleanOldData(void).

Limpia las variables usadas antes de iniciar un nuevo proceso de cálculo de velocidad de CAN.

FTM Abtraction Layer.

Capa de software independiente del microcontrolador, pero dependiente de los componentes conectados a él. Esta nos permite procesar los valores recibidos del *timer* en la MCAL y convertirlos a valores útiles para la capa de aplicación, es decir, convierte valores crudos del *timer* a valores de ancho de pulso en microsegundos (us). La figura 3-12 muestra las funciones aquí implementadas, y descritas a continuación:

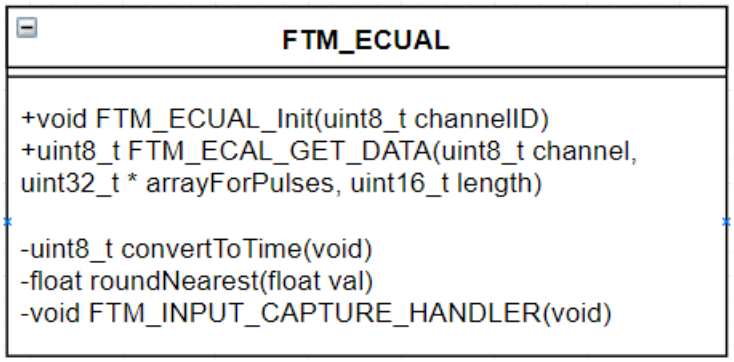


Fig. 3- 12. Funciones de la capa FTM_ECUAL.

+void FTM_ECUAL_Init(uint8_t channelID).

Aquí se hace llamado a la capa inferior para inicializar el módulo del *Flex Timer* (FTM) y configurarlo en modo de capturas tanto pulsos de bajada como de subida. Configurando el pre-escalador del módulo FTM en su valor mínimo para obtener la mayor resolución posible.

+ uint8_t FTM_ECAL_GET_DATA(uint8_t channel, uint32_t * arrayForPulses, uint16_t length).

Esta función inicia la captura de pulsos en el canal correspondiente, aunque de momento solo existe un canal, razón por el cuál no se implementó el soporte para múltiples canales.

-uint8_t convertToTime(void).

Función local de la capa actual que se usas para convertir los valores capturados del FTM driver en valores de ancho de pulso en tiempo, microsegundos (us).

-float roundNearest(float val).

Función auxiliar para redondear números del tipo “float” al entero más cercano. Se implementó esta función en particular para evitar importar alguna librería matemática, solo por usar una función en específico.

-void FTM_INPUT_CAPTURE_HANDLER(void).

Es el manejador de las interrupciones generadas por el FTM, estando en esta capa se está violando la filosofía de la arquitectura AUTOSAR [21], sin embargo, se hizo de esta manera para poder ejecutarse lo más rápido posible y permitirle que termine antes que una nueva interrupción sea generada.

Debido a esta limitante de velocidad de ejecución de la interrupción se ha limitado el alcance del actual proyecto a capturas anchos de pulso mínimos de 2us ya que se demostró que para anchos de pulso menores se tiene encolamiento de las interrupciones del FTM, lo que provoca pérdidas en los datos capturados.

FTM driver.

Para la capa correspondiente de la MCAL se utilizó software provisto por el fabricante de la tarjeta de desarrollo de NXP Semiconductor Inc. [28]. Dicho software se distribuye bajo la

licencia "*BSD License*" [38]. Dicho software es parte del paquete SDK que se puede obtener de la página oficial de NXP [39]. Además de poderse encontrar documentación con instrucciones para su descarga e instalación [40]. En la figura 3-13 se destacan las funciones utilizadas en el actual modulo, todas ellas son llamadas por la capa superior (ECUAL) para acceder a los recursos de FTM.

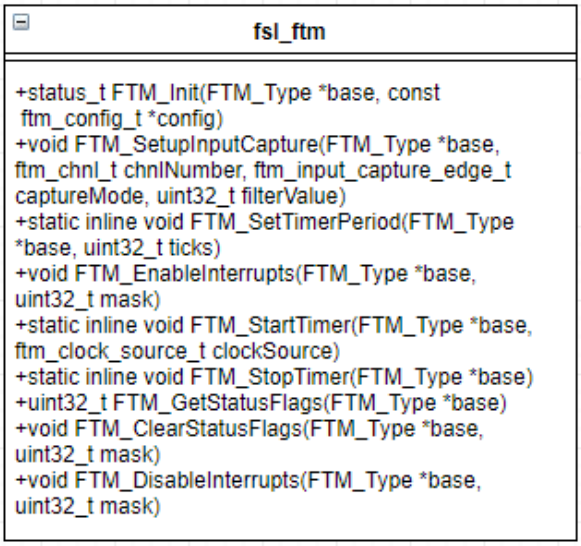


Fig. 3- 13. Funciones de la capa fsl_ftm (MCAL).

+status_t FTM_Init(FTM_Type *base, const ftm_config_t *config).

Inicializa el módulo de *Flex Timer*, recibe la dirección del periférico FTM que se desea usar y una estructura con la configuración deseada.

+void FTM_SetupInputCapture(FTM_Type *base, ftm_chnl_t chnlNumber, ftm_input_capture_edge_t captureMode, uint32_t filterValue).

Habilita la captura de pulsos en el periférico y canal recibido en los parámetros, así como el tipo de captura, esto es, flaco de subida, flanco de bajada o ambos.

+static inline void FTM_SetTimerPeriod(FTM_Type *base, uint32_t ticks).

Configura el valor del periodo del *timer* utilizado.

+void FTM_EnableInterrupts(FTM_Type *base, uint32_t mask).
Habilita las interrupciones de acuerdo con la máscara recibida en el segundo parámetro.

+static inline void FTM_StartTimer(FTM_Type *base, ftm_clock_source_t lockSource).
Inicia el Contador del *timer*, después de esto, las interrupciones que se configuraron previamente comenzarán a generarse.

+static inline void FTM_StopTimer(FTM_Type *base).
Detiene el contador del *timer*.

+uint32_t FTM_GetStatusFlags(FTM_Type *base).
Regresa el estatus de las banderas del módulo recibido, esto es útil para determinar qué tipo de interrupción ha ocurrido.

+void FTM_ClearStatusFlags(FTM_Type *base, uint32_t mask).
Limpia el estatus de las banderas del módulo recibido.

+void FTM_DisableInterrupts(FTM_Type *base, uint32_t mask).
Deshabilita las interrupciones de acuerdo con la máscara recibida.

Diagramas de la implementación.

Para facilitar de mejor manera la comprensión de la implementación del algoritmo, son agregados una serie de diagramas que expresan el comportamiento del mismo. En la figura 3-14 se aprecia un diagrama cíclico, el cual ilustra la manera en que la aplicación ***CAN Baude Rate Calculator*** es inicializado, cuando la aplicación se inicia.

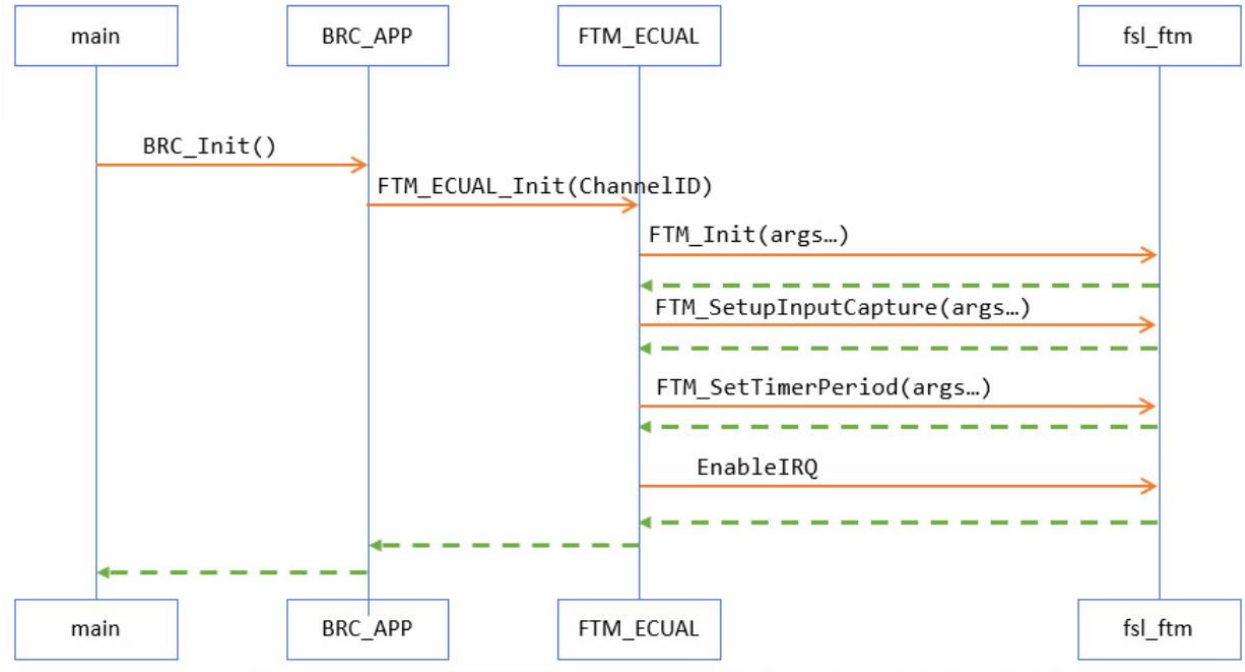


Fig. 3- 14. Secuencia de inicialización de la implementación.

El flujo de la secuencia de inicialización comienza por la parte superior izquierda con en el archivo **main**, donde se llama a la función `BRC_Init()`, que se encuentra dentro del archivo **BRC_APP.c**, se hace el llamado de la función `FTM_ECUAL_Init(ChannelID)`, la cual se encuentra dentro del archivo **FTM_ECUAL.c** dentro de este se realiza la inicialización de todos los parámetros necesarios para la configuración del módulo *Flex Timer*. Estos pasos de configuración también a otras sub funciones contenidas dentro del archivo controlador **fsl_ftm.c**, el cual contienen en su interior las funciones `FTMInit()`, `FTM_SetupInputCapture()`, `FTM_SetTimerPeriod()` y `EnableIRQ`. Para posteriormente terminar el proceso de inicialización y dar continuidad al programa en el archivo **main.c**.

Una vez que el flujo del programa ha regresado al archivo **main.c**, este da continuidad procesos posteriores, como lo son los que involucran el cálculo del *Baude Rate* de CAN. Esta serie de ciclos son ilustrados en la figura 3-15, que muestra dichos pasos de acuerdo con su secuencia y los archivos a los que debe saltar para lograr completar estos pasos.

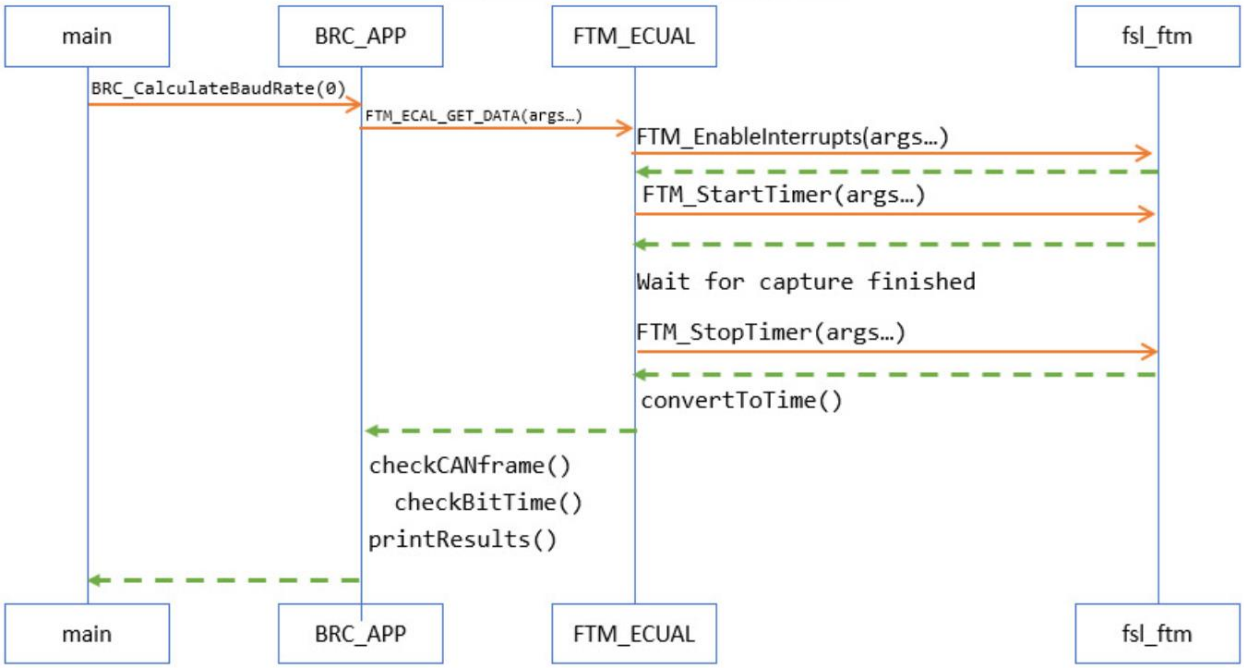


Fig. 3- 15. Secuencia de cálculo de baude rate.

Como se puede apreciar en la figura 3-15, después de haber inicializado los parámetros del módulo *Flex Timer*, se hace el llamado a la función `BRC_CalculateBaudeRate()`, contenida dentro de **BRC_APP.c**, que a su vez llama a la función `FTM_ECAL_GET_DATA()`, contenida dentro del archivo **FTM_ECUAL.c**, y es aquí donde son llamadas las funciones que habilitan las interrupciones para el que se atiendan la interrupciones del *Flex Timer*, arrancan la operación del *Timer*, esperan a que se termine la captura finalmente se detiene el *Timer* para devolver el resultado del dicho cálculo a la funciones restantes contenidas en BRC_APP.c (`FTM_EnableInterrupts()`, `FTM_StartTimer() y FTMStopTimer()`) y que son la que se aseguran de verificar la partes del *frame*, el *bit timing* y la impresión de los resultados calculados (`checkCANFrame()`, `checkBitTime() y printResult()`).

Todo lo descrito anteriormente es resumido de manera más simple como se muestra en el diagrama de la figura 3-16, donde se presentan los procesos descritos anteriormente.

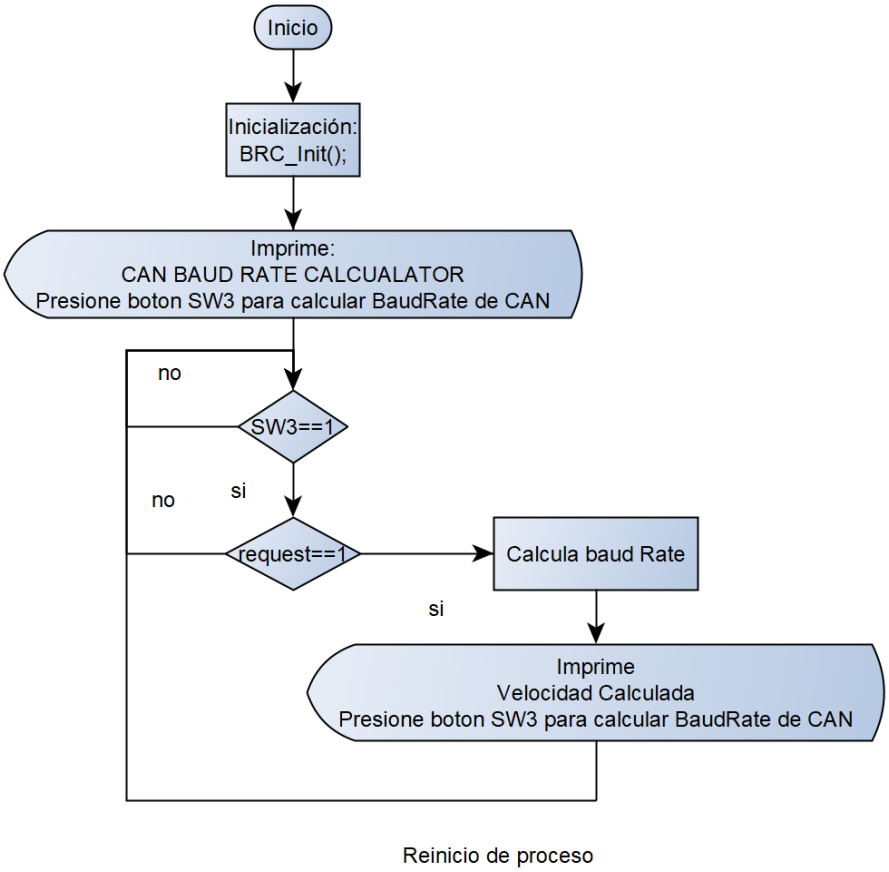


Fig. 3- 16. Diagrama de flujo de la implementación.

La figura 3-16 muestra cómo la implementación ***CAN Baude Rate Calculator*** realiza los procesos de inicialización, de periféricos para entra, salida y el módulo de *Flex Timer*, para posteriormente imprimir en la terminal los mensajes de "**CAN Baude Rate Calculator**" y "**Presione boton SW3 para calcular Baude Rate de CAN**".

Una vez en esta sección, entra a un ciclo infinito en espera de que el usuario presione el botón SW3. En el momento en que SW3 sea presionado, es disparada la interrupción que cambia el valor de la variable **request** a verdadero, permitiendo que se cumpla la condición para iniciar el proceso del cálculo de *baude rate*. Después de haber terminado el proceso, la aplicación imprime el resultado de la velocidad calculada, seguido de la leyenda "**Presione boton SW3 para calcular Baude Rate de CAN**", para posteriormente volver al inicio del ciclo infinito, y esperar que sea presionado de nueva cuenta SW3 para realizar una nueva medición.

4. Resultados.

El objetivo del presente trabajo fue claro y conciso, calcular la velocidad en un BUS CAN, dicho objetivo fue logrado satisfactoriamente después de haber finalizado la implementación mencionada en la sección anterior. Se realizaron pruebas tanto en laboratorio con condiciones controladas, como en campo (conectado a un automóvil).

Resultados en laboratorio

La figura 4-1 muestra el hardware y las conexiones utilizadas para las pruebas en laboratorio, para ello se utilizaron dos nodos de CAN formando una red y enviándose mensajes cada determinado tiempo, los mensajes que estos nodos se enviaban, así como la velocidad a la que trabajaban eran conocidos para nosotros. La tarjeta FRDM-K64F se conectó a dicha red a través del transceiver de CAN y a la PC a través de USB.

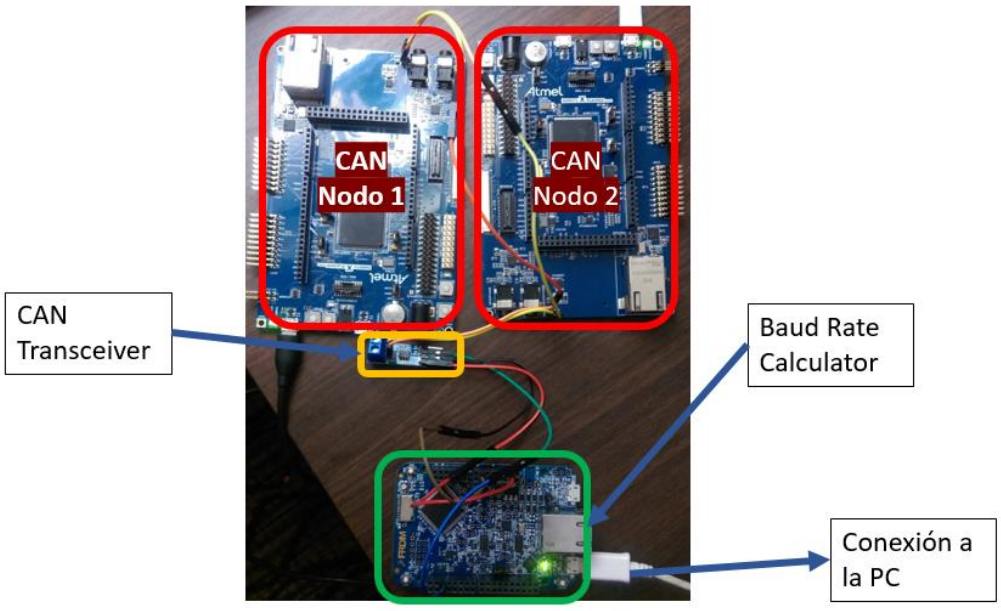


Fig. 4- 1. Conexión para pruebas en laboratorio.

Por el lado de la PC se utilzó el programa *Tera Term* para usar la terminal serial como interfaz con la tarjeta FRDM-K64F [30]. En las figuras 4-2 se puede ver el inicio de la aplicación (CAN BAUD RATE CALCULATOR) [37], después de esto se usa el botón SW3 para iniciar un muestreo de deteccion de velocidad. Una vez presionado SW3, se observa que la velocidad detectada fue 500 Kbps y además se mestran los datos de una trama de CAN, dicha trama y velocidad coinciden con la información que los nodos comparten entre si por el BUS CAN.

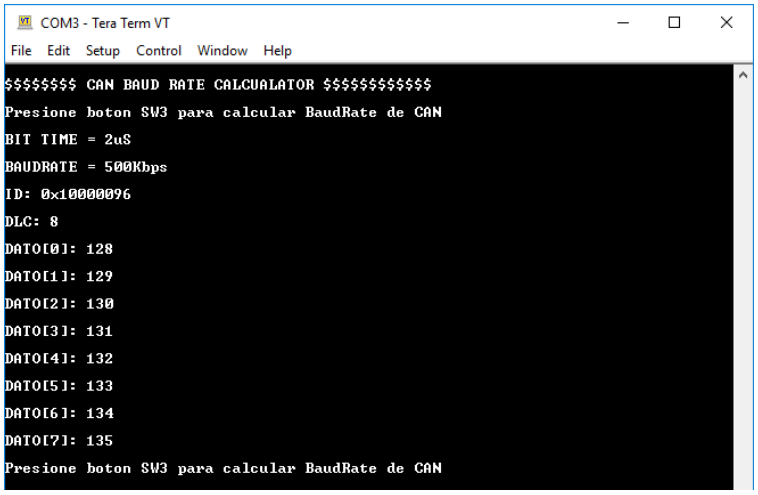


Fig. 4- 2. Resultado con ID extendido capturado.

Con base a la figura 4-3 se puede apreciar el resultado de una nueva medición de velocidad de BUS CAN, dicho resultado vuelve a coincidir con la velocidad de los nodos, 500 Kbps, pero ahora se capturó una trama diferente, esta corresponde a un mensaje con diferente ID (estándar), datos y longitud que el anterior mensaje, pero concuerdan con los mensajes que los nodos están transmitiendo.

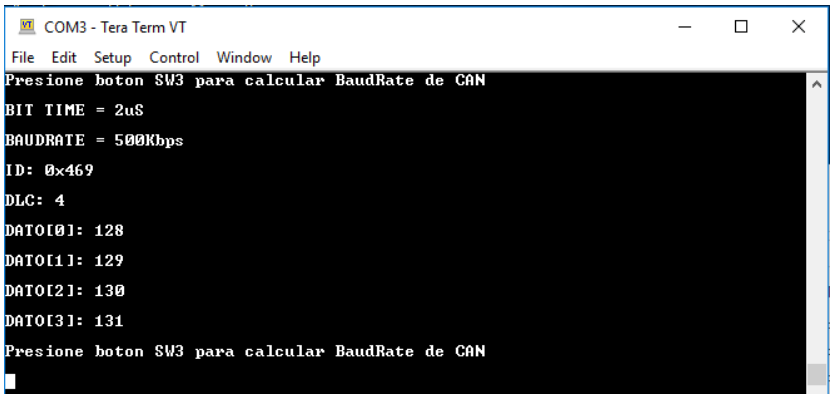


Fig. 4- 3. Resultado con ID estándar capturado.

Resultados en campo (Automóvil)

Cómo se mencionó anteriormente, también se logró obtener resultados de una red CAN en un automóvil, para esto se conectó la tarjeta FRDM-K64F con el transceiver al puerto OBD de un automovil. En la figura 4-4 se muestra la conexión con el puero OBD.

Fig. 4- 4. Conexión a conector OBD en automóvil.

En la figura 4-5 se muestran los resultados obtenidos después de inicializar la aplicación y presionar el botón SW3 en repetidas ocasiones, el resultado obtenido fue una velocidad de 500 Kbps capturando una trama con ID extendido, con datos desconocidos para nosotros dado que se conectó a un automóvil aleatoriamente.

Fig. 4- 5. Resultados en el automóvil.

Discusión

Es importante recalcar que el objetivo principal, que es generar un algoritmo para la detección de velocidad del BUS CAN se logró con éxito siguiendo los pasos descritos en la sección de la metodología. Pero además de esto, también se logró descifrar el identificador y el mensaje enviados en la trama dando como resultado una solución más completa de la proyectada. En comparación con las herramientas citadas en la sección de la introducción, se comentó que ya

existen herramientas que hacen esto, pero su costo es muy elevado y esta funcionalidad esta oculta dentro de un juego de funcionalidades extra.

Entre las ventajas obtenidas como fruto de la investigación se puede mencionar las siguientes:

- El algoritmo es migrable a cualquier otra herramienta.
- La implementación del mismo es tan económica como lo elija su desarrollador.
- Solo requiere de un microcontrolador con módulo de entrada-captura, para detección de flancos de voltaje.

Las desventajas con las que hay que liderar son las siguientes:

- El desarrollo está en una etapa muy temprana y requiere de mejoras.
- Sus funcionalidades aún son limitadas como para competir como un producto comercial,

Conclusiones

Como conclusión, podemos decir que el resultado obtenido de la investigación realizada para el desarrollo de un algoritmo detector de velocidad de protocolo CAN, tuvo resultados satisfactorios ya que en la implementación demostró ser un método fiable para esta aplicación. Además de que también se logró agregar las funcionalidades para descifrar el identificador del mensaje y los datos enviados en la trama.

Por otra parte, es importante mencionar que para fines demostrativos se usó la tarjeta FRDM-K64F de NXP, pero dicho algoritmo puede ser implementado en cualquier otra herramienta de desarrollo similar o más económica, ya que para su implementación solo se requiere que el microcontrolador seleccionado contenga un módulo de entrada-captura para la detección de flancos de subida y bajada, haciendo del mismo una opción económicamente inmejorable.

Por lo cual se deja establecida una base para que, en versiones posteriores a esta línea de investigación, independientemente del hardware que se tenga planeado utilizar, se pueda dar continuidad a la misma, agregando mejoras y funciones adicionales a las ya logradas.

Apéndices

A. CÓDIGO DE LA IMPLEMENTACIÓN: BAUDE RATE CALCULATOR

```
/*****************************************************************************
 *  main.c
 *  Created on: Jun 25, 2018
 *  Authors: Feliciano Angulo & Manuel Zambrano
/*****************************************************************************/

#include "fsl_debug_console.h"
#include "board.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "fsl_port.h"
#include "fsl_gpio.h"
#include "fsl_common.h"

/* Interface con aplicación Baud Rate Calculator */
#include "BRC_APP.h"

/*****************************************************************************/
 * Definitions
/*****************************************************************************/
#define BOARD_SW_GPIO BOARD_SW3_GPIO
#define BOARD_SW_PORT BOARD_SW3_PORT
#define BOARD_SW_GPIO_PIN BOARD_SW3_GPIO_PIN
#define BOARD_SW_IRQ BOARD_SW3_IRQ
#define BOARD_SW_IRQ_HANDLER BOARD_SW3_IRQ_HANDLER
#define BOARD_SW_NAME BOARD_SW3_NAME
/*****************************************************************************/
 * Variables
/*****************************************************************************/
uint8_t request = 0;
/*****************************************************************************/
 * Code
/*****************************************************************************/

void BOARD_SW_IRQ_HANDLER(void){
    /* Clear external interrupt flag. */
    GPIO_PortClearInterruptFlags(BOARD_SW_GPIO, 1U << BOARD_SW_GPIO_PIN);
    /* Change state of button. */
    request = 1;
    /* Add for ARM errata 838869, affects Cortex-M4, Cortex-M4F Store immediate
overlapping
      exception return operation might vector to incorrect interrupt */
#if defined __CORTEX_M && (__CORTEX_M == 4U)
    __DSB();
#endif
}
/*****************************************************************************/
/***** brief Main function ******/

int main(void){
    /* Define the init structure for the input switch pin */
    gpio_pin_config_t sw_config = {kGPIO_DigitalInput, 0,};
    /* Board pin, clock, debug console init */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
```

```
    /* initialize CAN BAUD RATE CALCULATOR application */
    BRC_Init();
    /* Init input switch GPIO. */
       PORT_SetPinInterruptConfig(BOARD_SW_PORT, BOARD_SW_GPIO_PIN,
       kPORT_InterruptFallingEdge);
       EnableIRQ(BOARD_SW_IRQ);
       GPIO_PinInit(BOARD_SW_GPIO, BOARD_SW_GPIO_PIN, &sw_config);
       printf("\r\n$$$$$$$$ CAN BAUD RATE CALCUALATOR $$$$$$$$$$$$\r\n");
       printf("\r\nPresione boton SW3 para calcular BaudRate de CAN\r\n");
    while (1){
       if(request){
              /* call method for calculate baud rate */
              BRC_CalculateBaudRate(0);
              request = 0;
              printf("\r\nPresione boton SW3 para calcular BaudRate de CAN\r\n");
       }
    }
}
/*************************************************************************
 *     BRC_APP.c
 *     Created on:  Jun 25, 2018
 *      Authors:     Feliciano Angulo & Manuel Zambrano
 *************************************************************************/
#include "BRC_APP.h"
#include "FTM_ECUAL.h"
/*************************************************************************
 * Definitions
 *************************************************************************/
#define ARRAY_LENGTH   200
#define MAX_FRAME_LEN  131
#define MIN_FRAME_LEN   34
#define INDEX_RTR       12
#define INDEX_RTR_X     32
#define INDEX_IDE       13
#define INDEX_IDE_X     33
#define INDEX_DATA      19
uint8_t INDEX_DLC = 15;
/*************************************************************************
 * Prototypes
 *************************************************************************/
uint8_t checkCANframe(void);
uint8_t checkBitTime(void);
void printResults(void);
/*************************************************************************
 * Variables
 *************************************************************************/
/* Este arreglo conendra los  */
//uint32_t * AppPulseWidthArray;
uint32_t AppPulseWidthArray[ARRAY_LENGTH];
uint32_t baudeRates[MAX_IN_CAP] = {0};
uint32_t br_calculated = 0;
uint8_t canDataArray[MAX_FRAME_LEN];

uint8_t captureChannelList[MAX_IN_CAP] ={IN_CAP0};

/*estructura para guardar los datos de un frame de can despues de leerlos*/
struct canMsg{
       uint32_t bit_time;
       uint8_t isFD;
       uint32_t ID;
```

```
	uint8_t DLC;
	uint8_t DATA[8];
}CAN_MSG;

/*****************************************************************************
 * Code
 *****************************************************************************/
/* Inicialización de la aplicación */
void BRC_Init(){
	/* Inicializar el bit time en un vlaor lo suficiente alto par aque sea mayor que
	un posibe bit time capturado.*/
	CAN_MSG.bit_time = 0xFFFF;
	/*initialize FTM ECUAL */
	for(uint8_t i = 0; i < MAX_IN_CAP; i++){
		/* llamada al ECUAL ara inicializar el FTM de todos los canales
disponibles.*/
		FTM_ECUAL_Init(captureChannelList[i]);
	}
}

/*********************************************************
 * Esta función inicia la captura de pulsos en el bus de CAN,
 * el resultado se gurdad en AppPulseWidthArray
 * *******************************************************/
uint32_t BRC_CalculateBaudRate(uint8_t channel){
	uint8_t success = 0;
	uint16_t tryCounter = 100;
	for(uint8_t i = 0; i < MAX_FRAME_LEN; i++){
		canDataArray[i] = 0;
	}
	for(uint8_t i = 0; i < tryCounter; i++){
		if(channel < MAX_IN_CAP){
			if(FTM_ECAL_GET_DATA(channel, &AppPulseWidthArray[0],
(uint16_t)ARRAY_LENGTH)){
				if(checkCANframe()){
					success = 1;
					break;
				}
			}
		}
	}
	if(success){
		printResults();
		return br_calculated;
	}
	else{
		printf("\r\n################### ERROR ###################\r\n");
		printf("####Fail to get Baud Rate after %d attempts ####\r\n",
tryCounter);
		return 0;
	}
}

/*********************************************************
 * Imprime el resultado con los detalles del Mensaje leido.
 * *******************************************************/
void printResults(void){
	/*check baud rate*/
	br_calculated = (1000000 / CAN_MSG.bit_time);
	printf("\r\nBAUDRATE = %dKbps\r\n", br_calculated / 1000);
	printf("\r\nID: 0x%x \r\n", CAN_MSG.ID);
```

```
	printf("\r\nDLC: %d \r\n", CAN_MSG.DLC);
	for(uint8_t i = 0; i < CAN_MSG.DLC; i++){
		printf("\r\nDATO[%d]: %d \r\n", i, CAN_MSG.DATA[i]);
	}
}
/*********************************************************************
 * Analiza los datos recibidos en AppPulseWidthArray
 * decodifica el contenido para validar si exise algun mensaje de CAN
 * además verifica el bit time
 * ********************************************************************/
uint8_t checkCANframe(void){
	uint32_t interframe_length = 0;
	uint32_t frameStartIndex = 0;
	uint32_t frameStopIndex = 0;
	uint8_t bit_counter = 0;
	uint8_t bus_level = 0;
	uint8_t data_counter = 0;
	uint8_t is_stuffing = 0;
	uint8_t index_data = 0;
	uint8_t aux = 0;
	CAN_MSG.DLC = 0;
	/* Obtener el bit time*/
	if(!checkBitTime()){
		return 0;
	}
	interframe_length = CAN_MSG.bit_time * 12;
	// Obtener la dirección de inicio y fin de un frame de CAN
	for(uint32_t i = 0; i < ARRAY_LENGTH; i++){
		if(AppPulseWidthArray[i] >= interframe_length){
			if(!frameStartIndex){
				frameStartIndex = i + 1;
			}
			else{
				frameStopIndex = i - 1;
				/*Convertir los pulsos a valores loginos 1/0
				 * remueve el bit stuffing cuando este existe.
				 * ****************************************/
				for(uint32_t i = frameStartIndex; i <= frameStopIndex; i++){
					bit_counter = AppPulseWidthArray[i]/
CAN_MSG.bit_time;
					if(bit_counter < 6){
						/*se usa is_stuffing  para omitir el primer
bit(stuffing) cuando este vale 1*/
						for(uint8_t j = is_stuffing; j < bit_counter;
j++)
						{
							canDataArray[data_counter] = bus_level;
							data_counter++;
						}
						if(bit_counter == 5){
							is_stuffing = 1;
						}
						else{
							is_stuffing = 0;
						}
						bus_level = !bus_level;
					}
				}
				if(data_counter > MIN_FRAME_LEN){
					/*cumple longitud mínima para ser un frame de CAN*/
					break;
```

```
                        }
                        else{
                                /*continuar analizando el arreglo*/
                                data_counter = 0;
                                frameStartIndex = frameStopIndex + 2;
                        }
                }
        }
        // analizar el fram guradado previamente en canDataArray
        /* extract ID */
        if(canDataArray[INDEX_RTR] == 0 && canDataArray[INDEX_IDE] == 0){
                /*standard frame*/
                CAN_MSG.isFD = 0;
                INDEX_DLC = 15;
                for(uint8_t i = 1; i < INDEX_RTR; i++){
                        CAN_MSG.ID |= canDataArray[i] << (11 - i);
                }
        }
        else if(data_counter > (MIN_FRAME_LEN + 19) && canDataArray[INDEX_RTR_X] == 0 &&
canDataArray[INDEX_IDE] == 1){
                /*extended frame*/
                CAN_MSG.isFD = 1;
                INDEX_DLC = 35;
                for(uint8_t i = 1; i < INDEX_RTR_X; i++){
                        if(i < INDEX_RTR){
                                CAN_MSG.ID |= canDataArray[i] << (29 - i);
                        }
                        else if(i > INDEX_IDE){
                                CAN_MSG.ID |= canDataArray[i] << (29 - i + 2);
                        }
                }
        }
        else{
                printf("\r\nCAN format Invalid\r\n");
                return 0;
        }
        /*Extract DLC*/
        CAN_MSG.DLC |= canDataArray[INDEX_DLC] << 3;
        CAN_MSG.DLC |= canDataArray[INDEX_DLC + 1] << 2;
        CAN_MSG.DLC |= canDataArray[INDEX_DLC + 2] << 1;
        CAN_MSG.DLC |= canDataArray[INDEX_DLC + 3];
        if(CAN_MSG.DLC > 8){
                printf("\r\nERROR in DLC rule: DLC = %d\r\n", CAN_MSG.DLC);
                return 0;
        }
        /* extract the data. */
        for(uint8_t i = 0; i < CAN_MSG.DLC; i++){
                index_data = INDEX_DLC + 4 + (8 * i);
                for(uint8_t j = index_data; j < index_data + 8; j++){
                        aux = 7 - (j - index_data);
                        CAN_MSG.DATA[i] |= canDataArray[j] << aux;
                }
        }
        return 1;
}

/*********************************************
 * Recorre el arreglo AppPulseWidthArray
 * para obtener el valor minimo de sus elementos.
 * dicho valor sera considerado como el bit time.
```

```
 * ******************************************/
uint8_t checkBitTime(void){
	CAN_MSG.bit_time = 0xFFFF;
	for(uint32_t i = 0; i < ARRAY_LENGTH; i++){
		if(AppPulseWidthArray[i] < CAN_MSG.bit_time){
			CAN_MSG.bit_time = AppPulseWidthArray[i];
		}
	}
	if(CAN_MSG.bit_time > 0){
		return 1;
	}
	else{
		return 0;
	}
}
/*****************************************************************************
 *  BRC_APP.h
 *  Created on:     Jun 25, 2018
 *  Author:         Feliciano Angulo & Manuel Zambrano
 *****************************************************************************/
#ifndef SWC_BRC_APP_H_
#define SWC_BRC_APP_H_

#include "fsl_debug_console.h"

typedef enum inputCaptureNumber{
	IN_CAP0 = 0,
	MAX_IN_CAP
} _inputCaptureNumber;

void BRC_Init();
uint32_t BRC_CalculateBaudRate(uint8_t channel);

#endif /* SWC_BRC_APP_H_ */
/*****************************************************************************/
```

Bibliografía

[1] "*ISO 11898-1: 2015*; *Road vehicles -- Controller area network* (CAN) -- *Part 1: Data link layer and physical signalling*". [En línea]. https://www.iso.org/standard/63648.html. [Consultado: 29-Jun-2018].

[2] «Inicio», Bosch en México. [En línea]. Disponible en: https://www.bosch.com.mx/. [Accedido: 28-jun-2018].

[3] Freescale Semiconductor, "Future Advancements in Body Electronics". [En línea]. «BODYDELECTRWP.pdf». Accedido 28 de junio de 2018. https://www.nxp.com/docs/en/white-paper/BODYDELECTRWP.pdf.

[4] "Vector Informatik." .[En línea]. Disponible en: https://vector.com/vi_index_en.html

[5] "VN1600 Network Interface for CAN, LIN, K-Line, J1708 and IO". [En línea]. Disponible en: https://vector.com/vi_vn1600_en.html. [Consultado: 02-Jun-2018].

[6] "CANoe - ECU Development & Test". [En línea]. Disponible en: https://vector.com/vi_canoe_en.html. [Consultado: 02-Jun-2018].

[7] "CAN tools by LAWICEL AB". [En línea]. Disponible en: http://www.can232.com/. [Consultado: 06-Jun-2018].

[8] "CANUSB Manua by LAWICEL AB l". [En línea]. Disponible en: http://www.can232.com/docs/canusb_manual.pdf[Consultado: 06-Jun-2018].

[9]"Microchip". [En línea]. Disponible en: http://www.microchip.com/about-us/company-information/about

[10] "CAN BUS Analyzer Tool". [En línea]. Disponible en: https://www.microchip.com/DevelopmentTools/ProductDetails/apgdt002

[11] "CAN BUS Analyzer User guide". [En línea]. Disponible en: http://ww1.microchip.com/downloads/en/DeviceDoc/51848B.pdf

[12] "ISO/IEC 7498-1 Open Systems Interconnection - Basic Reference Model". [En línea]. Disponible en: https://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf

[13] «Inicio», "International Organization for Standardization". [En línea]. https://www.iso.org/home.html. [Consultado: 29-Jun-2018].

[14] «Inicio», "Intel Corporation". [En línea]. Disponible en: https://www.intel.la/content/www/xl/es/homepage.html [Consultado: 29-Jun-2018].

[15] «Inicio», "Philips Corporation". [En línea]. Disponible en: https://www.philips.com/global [Consultado: 29-Jun-2018].

[16] "ISO 11898-2: 2016 CAN ". [En línea]. Disponible en: https://www.iso.org/standard/67244.html [Consultado: 02-Jul-2018]

[17] "ISO 11898-3: 2006 CAN ". [En línea]. Disponible en: https://www.iso.org/standard/36055.html [Consultado: 02-Jul-2018]

[18] "ISO 11898-4: 2004 CAN ". [En línea]. Disponible en: https://www.iso.org/standard/36306.html [Consultado: 02-Jul-2018]

[19] "ISO 11898-5: 2007 CAN ". [En línea]. Disponible en: https://www.iso.org/standard/41284.html [Consultado: 02-Jul-2018]

[20] "ISO 11898-6: 2013 CAN ". [En línea]. Disponible en: https://www.iso.org/standard/36055.html [Consultado: 02-Jul-2018]

[21] «Inicio», "AUTOSAR". [En línea]. Disponible en: https://www.autosar.org/ [Consultado: 02-Jul-2018].

[22] «Inicio», "AUTOSAR: Current Partners". [En línea]. Disponible en: https://www.autosar.org/about/current-partners/ [Consultado: 02-Jul-2018].

[23] «Inicio», "AUTOSAR: Core Partners". [En línea]. Disponible en: https://www.autosar.org/about/current-partners/core-partners/ [Consultado: 02-Jul-2018].

[24] «Inicio», "AUTOSAR: Premium Partners". [En línea]. Disponible en: https://www.autosar.org/about/current-partners/premium-partners/ [Consultado: 02-Jul-2018].

[25] «Inicio», "AUTOSAR: Development Partners". [En línea]. Disponible en: https://www.autosar.org/about/current-partners/development-partners/ [Consultado: 02-Jul-2018].

[26] «Inicio», "AUTOSAR: Associate Partners". [En línea]. Disponible en: https://www.autosar.org/about/current-partners/associate-partners/ [Consultado: 02-Jul-2018].

[27] «Inicio», "AUTOSAR: Attendees". [En línea]. Disponible en: https://www.autosar.org/about/current-partners/attendees/ [Consultado: 02-Jul-2018].

[28] «Inicio», "NXP". [En línea]. Disponible en: https://www.nxp.com/ [Consultado: 02-Jul-2018].

[29] «ARM news room», “ARM Lunches Class-Leading Cortex-M4 Processor for High Performance Digital Signal Control”. [En línea]. Disponible en: https://www.arm.com/about/newsroom/arm-launches-class-leading-cortex-m4-processor-for-high-performance-digital-signal-control.php [Consultado: 09-Jul-2018]

[30] «FRDM-K64F board», “FRDM-K64F NXP, sitio del fabricante”. [En línea]. Disponible en: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/k-seriesperformancem4/k2x-usb/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F [Consultado: 08-Jul-2018]

[31] «Keil», "About CMSIS-DAP". [En línea]. Disponible en: http://www.keil.com/support/man/docs/dapdebug/dapdebug_introduction.htm [Consultado: 11-Jul-2018].

[32] «NXP», "K64 Sub-Family Reference Manual". [En línea]. Disponible en: https://www.nxp.com/docs/en/reference-manual/K64P144M120SF5RM.pdf [Consultado: 11-Jul-2018].

[33] «TI», "VP230_SN65HVD23x 3.3-V CAN Bus Transceivers". [En línea]. Disponible en: http://www.ti.com/lit/ds/symlink/sn65hvd230.pdf [Consultado: 11-Jul-2018].

[34] «Inicio», "Amazon web site". [En línea]. Disponible en: https://www.amazon.com.mx/ [Consultado: 11-Jul-2018].

[35] «NXP Software and tools», “MCUXpresso IDE”. [En línea]. Disponible en: https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-

tools/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE?tab=Design_Tools_Tab [Consultado: 08-Jul-2018]

[36] «Inicio», "Tera Term home page". [En línea]. https://ttssh2.osdn.jp/index.html.en [Consultado: 11-Jul-2018].

[37] «Repositorio Github», "Codigo Detector de Velocidad de Protocolo CAN". [En línea]. Disponible en: https://github.com/FelicianoAngulo/CAN_BAUD_RATE_CALCULATOR.git [Consultado: 08-Jul-2018]

[38] «BSD sitio de la iniciativa», "BSD Open intiative". [En línea]. Disponible en: https://opensource.org/licenses/bsd-license.php [Consultado: 08-Jul-2018]

[39] «SDK sitio de descarga», "SDK NXP". [En línea]. Disponible en: https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/mcuxpresso-software-development-kit-sdk:MCUXpresso-SDK [Consultado: 08-Jul-2018]

[40] «NXP community forum», "Getting Started with MCUXpresso and FRDM-K64F". [En línea]. Disponible en: https://community.nxp.com/docs/DOC-334086 [Consultado: 08-Jul-2018]