

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



ANALIZADOR DE REDES CAN

Tesina para obtener el grado de:

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presentan: Ing. Adriana Elizabeth Gutiérrez Gómez.

Director: Dr. Luis Rizo Domínguez.

San Pedro Tlaquepaque, Jalisco. Septiembre de 2017.

Agradecimientos

Quiero agradecer a Dios el haberme permitido cursar esta especialidad así como también a las instituciones que lo hicieron posible como CONACYT (Becario No. 823467) y Continental Automotive Periférico y sobre todo a mi jefe de área.

A mi familia y muy en especial a mi compañero de vida que me brindo todo su apoyo y comprensión en el transcurso de la especialidad.

Y por supuesto a mi asesor de proyecto que con su apoyo fue posible la culminación de mi especialidad y proyecto de titulación.

Abstract

Communication in the automotive industry is essential for the development of newly installed technology in vehicles and is used to monitor the proper functioning of the car. One of the most used protocols in automotive communication is CAN (Controller Area Network), however, the monitoring of the interfaces of this system has a high-cost for the automotive industry. In the present project, a low-cost module for the detection of problems in the communication of a CAN network node is presented using a Tiva™ C Series TM4C123G LaunchPad development card with an ARM® Cortex®-M4F based on a TM4C123G MCU. By means of an embedded system, it was possible to develop the software that monitors CAN networks in a test equipment. This software can be adapted to the needs of the units that will be monitored through a friendly and easy to understand environment for any user based on the CAN library (Texas Instruments). In conclusion, the development of the software for CAN communication helps to identify problems in the communication between nodes and in real-time, which translates into a shorter downtime of the test equipment.

Resumen

Hoy en día, la industria automotriz cuenta con un estándar internacional en cuanto a la comunicación entre los dispositivos que componen un auto por ejemplo: sistema de frenos (ABS), transmisión (Power Train), tablero del auto (Cluster) por mencionar algunos. Estos dispositivos tienen la necesidad de comunicarse entre sí lo cual se realiza con un protocolo de comunicación CAN. Uno de los protocolos de mayor uso en la comunicación automotriz sin embargo, el monitoreo de las interfaces de este sistema tiene un alto costo para la industria automotriz. En el presente proyecto se planteó la necesidad de realizar un módulo de bajo costo para la detección de problemas presentados en la comunicación de una red de nodos CAN por medio de una tarjeta de desarrollo modelo Tiva™ C Series TM4C123G LaunchPad con un ARM® Cortex®-M4F basado en un MCU TM4C123G. En dicho sistema embebido se logró desarrollar el software que permite monitorear redes de CAN en un equipo de prueba. Este software puede adecuarse a las necesidades de las unidades que va a monitorear a través de un entorno amigable y fácil de entender para cualquier usuario basado en la API de CAN (Texas Instruments). En conclusión, el desarrollo del software para la comunicación CAN ayuda a identificar problemas en la comunicación entre nodos en tiempo real, lo que se traduce en un menor tiempo de paro en los equipos de prueba por fallas detectadas con respecto a la comunicación CAN.

Listado de figuras

Figura 2-1	Representación del modelo OSI	7
Figura 2-2	Comunicación entre Sistema	8
Figura 2-3	Capas modelo OSI	11
Figura 2-4	Arquitectura de capas del protocolo CAN	13
Figura 2-5	Secciones capa física	14
Figura 2-6	Niveles Alta velocidad	15
Figura 2-7	Niveles Baja velocidad	15
Figura 2-8	Inserción de bit	16
Figura 2-9	Segmentos del tiempo de bit	17
Figura 2-10	Principio de derivación del periodo de bit	17
Figura 2-11	Arquitectura de un nodo CAN con un transceptor	20
Figura 2-12	Arquitectura de un nodo CAN	23
Figura 2-13	Lógica del bus CAN	25
Figura 2-14	Formato de trama de datos CAN	28
Figura 2-15	Formatos de trama de datos CAN, estándar y extendida	29
Figura 2-16	Formato de trama de remota CAN	31
Figura 2-17	Formato de trama de error CAN	32
Figura 2-18	Formato de trama de sobrecarga CAN	33
Figura 2-19	Diagrama de flujo para el manejo de errores	36
Figura 3-1	Tarjeta Tiva C Series LaunchPad	39
Figura 3-2	Diagrama a bloques del controlador CAN	45
Figura 3-3	Tramas de CAN de Datos/Remota	47
Figura 3-4	Conexión de PC	50
Figura 4-1	Plataforma para desarrollo de software	52
Figura 4-2	Funciones de inicialización del software	53
Figura 4-3	Ciclo infinito y lectura del bus CAN	56
Figura 4-4	Tramas de CAN	57

Figura 4-5	Conexión entre la tarjeta SAM v71 y Texas para la obtención de tramas	58
Figura 4-6	Conexión SAM V71 con Tiva C Series TM4C123G LaunchPad	59
Figura 4-7	Tramas de CAN observadas en el osciloscopio	60

Lista de tablas

Tabla 2-1	Lógica del bus CAN	25
Tabla 3-1	Interruptores de usuario y señales GPIO	40
Tabla 3-2	Características Microcontrolador TM4C123GH6PM	42
Tabla 3-3	Especificaciones principales EKT4C123GH6PM	43
Tabla 3-4	Señales de red de área del controlador (64LQFP)	46

Listado de abreviaturas y acrónimos.

ABS	<i>(Del inglés, Anti-lock Break System, Sistema de Frenos Antibloqueo)</i>
ASC	<i>(Automatic Slip Control, Control Automático de Deslizamiento)</i>
ASIC	<i>(Application Specific Integrated Circuit, Circuito Integrado de Aplicación Específica)</i>
API	<i>(Application Program Interface, Interfaz de Programación de Aplicación Específica)</i>
ASR	<i>(Anti Slip Regulator, Regulador Antideslizamiento)</i>
CAL	<i>(Application Layer, Capa de Aplicación CAN)</i>
CAN	<i>(Controller Area Network)</i>
CIP	<i>(Control and Information Protocol, Protocolo de Control e Información)</i>
CMS	<i>(CAN Based Message Specification, Especificación de Mensajes Basada en CAN)</i>
CPU	<i>(Central Processing Unit, Unidad Central de Proceso)</i>
CRC	<i>(Cyclic redundant Check, Verificación de Redundancia Cíclica)</i>
CSMA/CD+AMP	<i>(Carrier Sense Multiple Access with Collision Detection and Arbitration Message Priority, Acceso Múltiple por Detección de Portadora, con Detección de Colisiones y Arbitraje por Prioridad de Mensajes)</i>
DLL	<i>(Data Link Layer, Capa de Enlace de Datos)</i>
DSP	<i>(Digital Signal Processing, Procesamiento Digital de Señales)</i>
E/S	<i>(I/O Input Output, Entradas/Salidas)</i>
ECU	<i>(Electronic Control Unit, Unidad de Control Electrónico)</i>
EDS	<i>(Electronic Data Sheet, Hoja de Datos Electrónica)</i>
EFF	<i>(Extended Frame Format, Formato de Trama Extendida)</i>
EML	<i>(Error Management Logic, Lógica de Gestión de Errores)</i>
ESP	<i>(Electronic Stability Program, Programa Electrónico de Estabilidad)</i>
FB	<i>(Fieldbus, Bus de Campo)</i>
FMS	<i>(Food Machinery Sales, Sistemas de Empaquetamiento de Comida)</i>
HLP	<i>(High Layer Protocol, Protocolo de Capa Alta)</i>
HVAC	<i>(Heating, Ventilating and Air Conditioned, Sistema de Calefacción, Ventilación y Aire Acondicionado)</i>

iCC	<i>(Internacional CAN Conference, Conferencia Internacional de CAN)</i>
ICE	<i>(In Circuit Emulator)</i>
IDB	<i>(Intelligent Data Bus, Bus Inteligente de Datos)</i>
IDE	<i>(Integrated Development Enviroment, Entorno de Desarrollo Integrado)</i>
IEC	<i>(Instituto de Electrónica y Computación)</i>
ISO	<i>(International Organization for Standarization, Organización Internacional para la Normalización)</i>
ISR	<i>(Interrupt Service Routine, Rutina de Servicio a Interrupción)</i>
LCD	<i>(Liquid Crystal Display, Visualizador de Cristal Líquido)</i>
LDSU	<i>(Link Service Data Units, Unidades de Datos del Servicio de Enlace)</i>
LLC	<i>(Logic Link Control, Control de Enlace Lógico)</i>
LME	<i>(Layer Management Entity, Entidad de Gestión de Capa)</i>
LMT	<i>(Layer Management, Gestión de Capa)</i>
MAC	<i>(Medium Access Control, Control de Acceso al Medio)</i>
MAC-LME	<i>(Entidad de Gestión de Capa MAC)</i>
MCNet	<i>(Mobile Communication Network, Red de Comunicación Móvil)</i>
MCU	<i>(Microcontroller, Microcontrolador)</i>
MDI	<i>(Medium dependent Interface, Interfaz Dependiente del Medio)</i>
MMI	<i>(Man-Machine Interface, Interfaz Hombre Máquina)</i>
MOST	<i>(Media Oriented System Transport)</i>
MSN	<i>(The Modules are to Serve the Network)</i>
MVB	<i>(Multiple Vehicle Bus)</i>
NMT	<i>(Network Management, Gestión de la Red)</i>
NRZ	<i>(Non Return to Zero, No Retorno a Cero)</i>
OS	<i>(Operating System, Sistema Operativo)</i>
OSI	<i>(Open Systems Interconnection, Interconexión de Sistemas Abiertos)</i>
PC	<i>(Personal Computer, Computadora Personal)</i>
PDA	<i>(Personal Digital Assistant, Asistente Digital Personal)</i>
PDO	<i>(Process Data Object, Objetos de Proceso)</i>
PLC	<i>(Programmable Logic Controller, Controlador Lógico Programable)</i>

PLD	<i>(Programmable Logic Device, Dispositivo Lógico Programable)</i>
PLL	<i>(Phase Locked Loop)</i>
PLS	<i>(Physical Signalling, Subcapa de Señalización Física)</i>
PLS-LME	<i>(Gestión de Fallos del Bus)</i>
PMA	<i>(Physical Medium Attachment, Conexión al Medio Físico)</i>
PMS	<i>(Philips Message Specification, Especificación de Mensajes de Philips)</i>
PROFIBUS-PA	<i>(PROFIBUS Profile for Process Automation)</i>
PSC	<i>(Process Segment Controllers, Controladores de Proceso Segmentado)</i>
RAM	<i>(Random Access Memory, Memoria de Acceso Aleatorio)</i>
RJW	<i>(Resynchronization Jump Width)</i>
RTOS	<i>(Real Time Operating System, Sistema Operativo en Tiempo Real)</i>
RTR	<i>(Remote Transmission Request, Petición de Transmisión Remota)</i>
SAE	<i>(Society of Automotive Engineers, Sociedad de Ingenieros Automotrices)</i>
SBS	<i>(Sensortronic Brake System, Sistema de Frenado Sensortronic)</i>
SDO	<i>(Service Data Objects, Objetos de Servicio)</i>
SDS	<i>(Smart Distributed System)</i>
SeeCAN	<i>(Sistema Educativo para la Enseñanza del Protocolo de Comunicaciones CAN)</i>
SFF	<i>(Standard Frame Format, Formato de Trama Estándar)</i>
SJW	<i>(Synchronization Jump Width)</i>
TDM	<i>(Time-Division Multiplexing, Multiplexación por División de Tiempo)</i>
TTCAN	<i>(Time Trigger CAN, Protocolo CAN Accionado por Tiempo)</i>
TTL	<i>(Transistor Transistor Logic, Lógica de Transistor a Transistor)</i>
USB	<i>(Universal Serial Bus)</i>
WCS	<i>(Weight Classification System, Sistema de Clasificación de Peso)</i>
WTB	<i>(Wide Train Bus)</i>

Contenido

Introducción	1
1. Antecedentes.....	3
2. Marco teórico	6
2.1. INTRODUCCIÓN PROTOCOLO DE COMUNICACIONES CAN	6
2.2. MODELO OSI	9
2.2.1 Las siete capas del modelo OSI	10
2.3. ARQUITECTURA DE CAPAS	12
2.4. CAPA FÍSICA.....	13
2.4.1 Subcapa de señalización física.....	14
2.4.2 Representación de bits.....	14
2.4.3 Temporización de bits.....	16
2.4.4 Mecanismos de sincronización de bits.....	18
2.5. SUBCAPA DE UNIÓN AL MEDIO FÍSICO.....	20
2.6. SUBCAPA DE INTERFAZ DEPENDIENTE DEL MEDIO	21
2.6.1 Medio físico	21
2.6.2 Medio de transmisión eléctrico	22
2.7. CAPA DE ENLACE DE DATOS	23
2.7.1 Control de enlace lógico.....	23
2.7.2 Funciones de la subcapa LLC	24
2.7.3 Control de acceso al medio	25
2.7.4 Trasmisión de mensaje.....	27
2.7.5 Trama de datos.....	28
2.7.6 Trama remota	30
2.7.7 Trama de error.....	31
2.7.8 Trama de sobrecarga	32
2.7.9 Espacio entre tramas	33
2.7.10 Codificación de tramas.....	34
2.7.11 Validación de tramas.....	34
2.7.12 Detección y manejo de errores.....	34
2.7.13 Mecanismos de detección de errores.....	35
2.7.14 Manejo de errores.....	36
3. Metodología	39
3.1. HARDWARE UTILIZADO	39
3.1.1 Descripción general de la serie Tiva™ C.....	41
3.1.2 Características Microcontrolador TM4C123GH6PM.....	41
3.1.3 Especificaciones.....	42
3.2. DESCRIPCIÓN FUNCIONAL.....	43
3.2.1 Microcontrolador	43
3.3. MÓDULO DE RED CAN	44
3.3.1 Diagrama a bloques controlador CAN.....	45
3.3.2 Descripción de la señal.....	46

3.3.3	Descripción funcional	46
3.3.4	Inicialización	48
3.4.	SOFTWARE UTILIZADO.....	49
3.4.1	Code Composer Studio	50
4.	Resultados	51
4.1.	EJECUCIÓN DE LA COMUNICACIÓN CAN	57
A.	TÍTULO DEL PRIMER APÉNDICE, CON UN TÍTULO MUY LARGO PARA QUE SE OBSERVE CÓMO SE ACOMODA EL TEXTO	¡ERROR! MARCADOR NO DEFINIDO.
B.	OTRO EJEMPLO DE TÍTULO DE APÉNDICE.....	¡ERROR! MARCADOR NO DEFINIDO.

Introducción

En la actualidad, la industria automotriz representa una parte fundamental para el progreso económico de un país. Es decir, el desarrollo tecnológico de los países depende del capital humano capacitado que desee incorporarse al mundo de la modernidad y por lo tanto, ser más competitivos a nivel mundial.

Los vehículos modernos utilizan muchos sistemas, los cuales ayudan a monitorear los estados de los sensores del auto y debido al actual número de sistemas introducidos en los automóviles, se ha hecho necesario el uso de protocolos de comunicación para conectar los elementos por medio de un canal denominado bus. En este caso, el bus que analizaremos es CAN que sirve para transmitir datos entre sistemas complejos, instalados en los autos para mejorar la eficiencia y desempeño en la comunicación. [1]

CAN (del inglés, *Controller Area Network*) fue desarrollado en 1983 por Bosch. CAN es un sistema de comunicación serial diseñado para tiempo real para sistemas distribuidos y ampliamente utilizados en la industria. El uso de los protocolos de CAN en las computadoras de los vehículos como ECM- Módulo de Control del Motor (del inglés, *Engine Control Module*), TCU Control de la Transmisión Automática (del inglés, *Transmission Control Unit*), y ESP – Programa Electrónico de Estabilidad (del inglés, *Electronic Stability Program*), entre otros. Estos dispositivos intercambian información entre ellos para facilitar u optimizar los sistemas de control de funciones (inyección, embrague, sistema de frenado, etc.) [2]

Actualmente, el protocolo CAN es más el utilizado en la industria automotriz y se caracteriza por su robustez y velocidad en la transmisión de altos volúmenes de datos. Se utiliza principalmente para aplicaciones en tiempo real como la gestión del motor, transmisión y sistema clásico de frenado (hidráulico).

Cada uno de estos sistemas recoge datos de muchas partes del vehículo a través de los sensores y los comparte por medio del bus de CAN con otros sistemas. Información que debe reunirse para que el sistema tenga una visión completa del proceso y sea transmitida al conductor del auto por medio del tablero e indicadores en el vehículo. La información transmitida es de suma importancia, para el desempeño óptimo de un vehículo y esta será la parte esencial del trabajo ya que nos interesa saber que datos se transmiten por medio del bus CAN.

En el presente proyecto se realizará un dispositivo, por medio del cual se podrá gestionar, analizar y monitorear los datos entre el clúster (tablero indicador en un auto) y un equipo de pruebas funcionales. Lo anterior, con el fin de revisar posibles fallas entre la comunicación de estos dos elementos.

1. Antecedentes

El bus CAN surge a los inicios de la década de los ochentas de la necesidad de encontrar una forma de interconectar los distintos dispositivos de un automóvil de una manera sencilla y reduciendo significativamente las conexiones. Este desarrollo fue posible gracias a un grupo de ingenieros de la compañía alemana Robert Bosch GmbH, quienes estudiaron la posibilidad de aplicar sistemas de bus seriales dentro de los automóviles con la finalidad de satisfacer las demandas de la Sociedad de Ingenieros Automotrices (SAE, del inglés, *Society of Automotive Engineers*) y como resultado se obtuvo la especificación del protocolo CAN. [3]

La investigación inicio en 1983, la compañía Bosch empezó un proyecto interno para desarrollar una red de interconexión entre los diversos dispositivos electrónicos instalados en los automóviles, con la finalidad de agregar mayor funcionalidad y reducir el cableado del sistema. Dicho proyecto fue dirigido por Siegfried Dais, Martin Litschel y el Dr. Uwe Kiencke. Durante la especificación del sistema de bus serial se integró la empresa fabricante de automóviles Mercedes-Benz y el fabricante de semiconductores Intel Corp. En febrero de 1986, Robert Bosch GmbH presentó oficialmente el sistema de bus serial CAN en el congreso de SAE celebrado en la ciudad de Detroit en los Estados Unidos de América (E.U.A). [4]

A pesar de que el bus CAN fue desarrollado para aplicaciones en automóviles, las primeras implementaciones en Europa, fueron en áreas ajenas al sector automotriz, como la fabricación de elevadores en Finlandia, sistemas médicos en Holanda y maquinas textiles en Suecia por mencionar algunas.

En 1992, Mercedes-Benz implemento CAN en sus automóviles de clase alta. Y aunque en un principio el bus se limitaba a interconectar las unidades de control electrónico que cuidaban del control del motor, pronto conectarían además las unidades de control necesarias para los componentes electrónicos. Para ello, se implementaron dos sistemas de bus CAN separados físicamente, y conectados a través de ‘gateways’. Actualmente, muchos otros fabricantes de coches

como “BMW”, “Renault”, “Fiat”, “Saab”, “Volkswagen” o “Volvo”, han seguido su ejemplo y suelen implementar redes CAN en sus automóviles.

Otra de las cosas sobresalientes ocurrió en el año 2000 cuando ISO formó un grupo de trabajo integrado por empleados de “Bosch”, académicos y expertos de la industria de los semiconductores, quienes definieron un nuevo protocolo basado en CAN, el llamado TTCAN – Comunicación por Disparo de Tiempo CAN (del inglés, *Time-Triggered-Communication-Area-Network*). Esta extensión permitía tanto la transmisión de mensajes simultáneos, como la implementación de un bucle cerrado para el control del bus. Además, gracias a que el protocolo del bus no fue modificado, dichos mensajes podían seguir siendo enviados con el mismo sistema físico del bus.

Derivado de la importancia del protocolo CAN y su consolidación en la industria automotriz, se investiga la implementación de un módulo observador de un protocolo de comunicación CAN que permite observar que dos entidades de un sistema de comunicación se comuniquen entre sí (tester y unidad a probar), así como validar el intercambio de mensajes generados entre el tester y la unidad de prueba.

2. Marco teórico

2.1. Introducción protocolo de comunicaciones CAN

CAN es conocido como un protocolo de comunicación en serie de alta confiabilidad, robustez y performance, el cual es implementado mediante un desarrollo de capa de aplicación para el control de sistemas distribuidos en tiempo real. Sus principales características son:

- Priorización de mensajes.
- Sistema multi-maestro.
- Configuración flexible.
- Velocidad de transmisión media (hasta 1 Mbit/s).
- Señalización y detección de fallas.

Para detectar errores a nivel de mensajes, el protocolo implementa tres mecanismos de verificación de redundancia cíclica (CRC), chequeo de formato de trama y acuse de recibo (ACK). En la detección de errores a nivel de bit el protocolo implementa dos mecanismos, primeramente monitorea el nivel del bus de cada nodo conectado, con el fin de detectar la diferencia entre el bit enviado y el recibido. El segundo mecanismo es conocido como Bits de Relleno (del inglés, *Bit stuffing*) en donde CAN utiliza la codificación NRZ (no retorno a cero) y la técnica de bit stuffing, que consiste en insertar un bit complementario cada cinco bits iguales consecutivos transmitidos.

CAN está basado en las dos primeras capas del Modelo OSI (del inglés, *Open System Interconnection*). Las especificaciones de este protocolo se encuentran regidas bajo el estándar ISO-11898 [5].

El Modelo OSI es un modelo dividido en 7 capas, diseñadas como un estándar para telecomunicaciones entre sistemas electrónicos (Figura 2-1), podemos observar que tipo de información se procesa en cada capa que se muestra en la columna uno. En la segunda columna,

se encuentran las capas que forman el modelo OSI y en la tercera columna, las capas que usa el protocolo CAN.

El modelo OSI fue diseñado para ser implementado en cualquier hardware que requiera comunicarse con otro hardware sin tener en cuenta la estructura interna específica o la tecnología de comunicación.

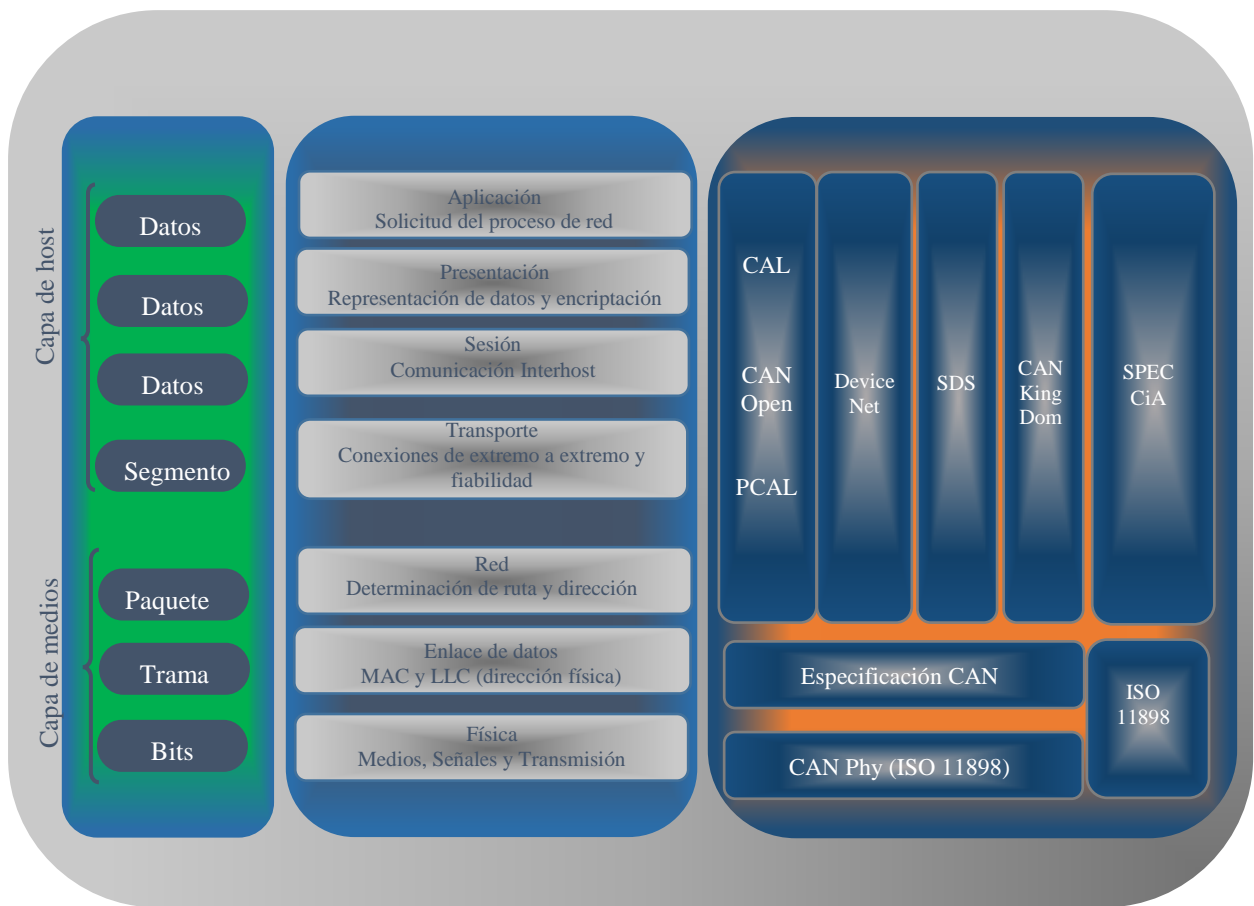


Figura 2-1. Representación del modelo OSI. La columna 1 representa el procesamiento de datos, la columna 2, las capas sistema OSI y la columna 3, las capas usadas por CAN.

Cada capa está diseñada para satisfacer una función específica en el proceso de telecomunicaciones peer-to-peer (Figura 2-2). Es importante mencionar que un protocolo no requiere ser situado en todas las capas del Modelo OSI. Muchos protocolos solo supervisan una o dos capas y esta flexibilidad permite a los desarrolladores combinar diferentes tipos de protocolos en diferentes arquitecturas de red. [6]

El modelo OSI tiene un conjunto de funciones que deben realizarse para que los paquetes de datos puedan viajar de un sistema a otro como se mencionó anteriormente y se muestra en la Figura 2-2.- Comunicación desde el sistema A al sistema B.

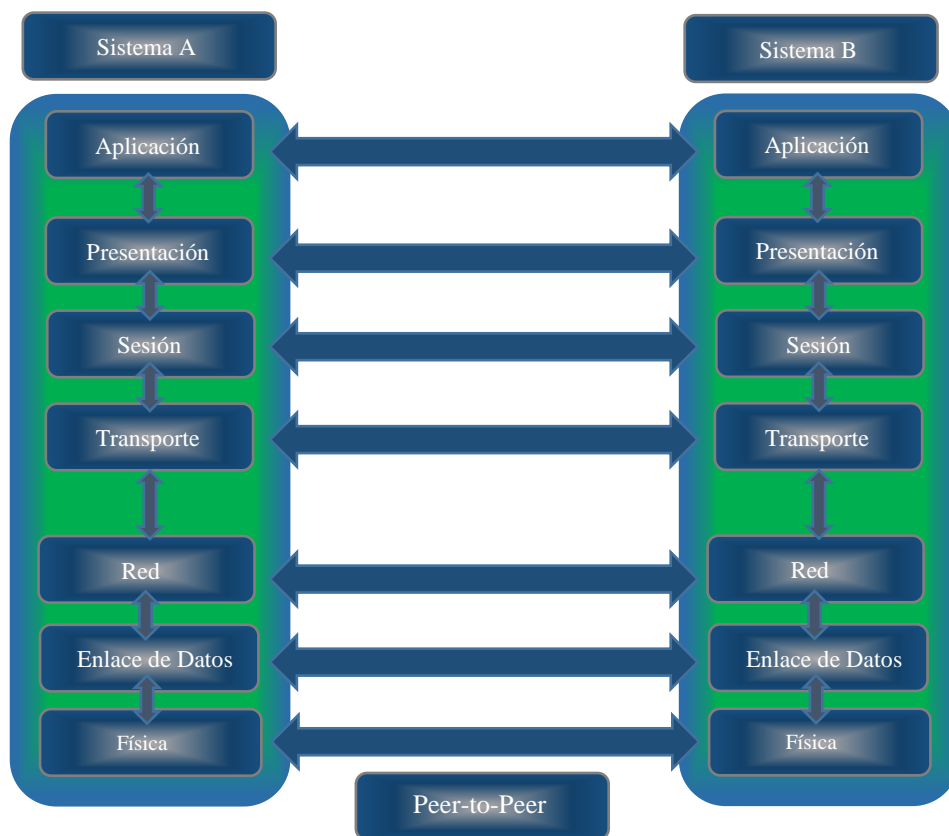


Figura 2-2 Comunicación entre Sistema A y Sistema B.

2.2. Modelo OSI

Las últimas décadas, el modelo OSI se ha caracterizado por un enorme crecimiento de redes, sin embargo, estas redes fueron desarrolladas usando implementaciones de hardware y software diferentes. Debido a esto, eran incompatibles y se volvió compleja la comunicación entre redes con diferentes especificaciones. Por tal motivo, la Organización Internacional para la Normalización (ISO) realizó investigaciones acerca de los esquemas de red. La ISO vio la necesidad de crear un modelo de red que ayudara a los diseñadores a implementar redes que pudieran comunicarse y trabajar en conjunto, lo cual llevo a la elaboración del modelo de referencia OSI en 1984.

El modelo de referencia OSI es un marco que se puede utilizar para comprender como viaja la información a través de una red. Por ejemplo, se usa para visualizar como la información o los paquetes de datos viajan desde los programas de aplicación, hasta otros programas de aplicación ubicados en otra computadora de la red. En el modelo de referencia OSI, hay siete capas numeradas, cada una de las cuales ilustra una función de red específica. [7]

ISO determinó una serie de principios a considerar para definir el conjunto específico de capas en la arquitectura OSI, y aplicó esos principios para definir las siete capas de la arquitectura OSI.

Para lograr definir las siete capas del modelo OSI, se tomaron en cuenta algunos criterios determinados por ISO, que se mencionan a continuación:

- 1) *“No crear tantas capas que dificulten la ingeniería de sistemas, es la ciencia de describir e integrar estas capas.*
- 2) *Crear un límite en un punto donde la descripción de los servicios puede ser pequeño y el número de interacciones a través de los límites frontera se minimiza.*
- 3) *Crear capas separadas para manejar funciones que son visiblemente diferentes en el proceso realizado o la tecnología involucrada.*

- 4) *Colecciona funciones similares en la misma capa.*
- 5) *Seleccionar los límites en un punto que la experiencia pasada haya demostrado tener éxito.*
- 6) *Crear una capa de funciones fácilmente localizadas para poder ser rediseñada en su totalidad y cambiar sus protocolos de una manera importante para tomar ventajas de los nuevos avances en arquitectura de hardware o software sin cambiar los servicios e interfaces con las capas adyacentes.*
- 7) *Crear un límite donde puede ser útil en algún momento en el tiempo para tener la interfaz correspondiente estandarizada.*
- 8) *Crear una capa cuando hay una necesidad de un nivel diferente de abstracción en el manejo de datos, por ejemplo, morfología, sintaxis, semántica.*
- 9) *Habilitar cambios de funciones o protocolos dentro de una capa sin afectar a las otras capas.*
- 10) *Cree para cada capa las interfaces con su capa superior e inferior solamente.*
- 11) *Crear un subgrupo más y la organización de las funciones para formar subcapas dentro de una capa en los casos en que distintos servicios de comunicación lo necesitan.*
- 12) *Crear, donde sea necesario, dos o más subcapas con una funcionalidad común, y por lo tanto mínima, para permitir el funcionamiento de la interfaz con las capas adyacentes.*
- 13) *Permitir el desvío de las subcapas". [8]*

2.2.1 Las siete capas del modelo OSI

La dificultad de trasladar información entre computadoras se divide en siete problemas más pequeños y de tratamiento más simple en el modelo de referencia OSI. Cada uno de los siete

problemas más pequeños está representado por su propia capa en el modelo como lo muestra la Figura 2-3. Las siete capas del modelo de referencia OSI de acuerdo a ISO son:

Capa 7: La capa de aplicación, es la que proporciona los servicios de red a los programas usados por los usuarios en las maquinas. Es la unica capa que no tiene comunicación con alguna otra capa OSI. Aquí se define la sincronizacion y arreglos sobre los procedimientos para la recuperacion de errores y control de datos entre los nodos que integran la red.

Capa 6: La capa de presentación, esta capa se asegura que la información que envía la capa de aplicación sea leída por cualquier otra capa de aplicación. En caso de ser necesario puede traducir de varios formatos de datos a un formato común para lograr una comunicación exitosa entre capas.

Capa 5: La capa de sesión se encarga de administrar las sesiones entre los hosts que se comuniquen entre sí. Facilita servicios a la capa de presentación, sincroniza la comunicación entre las capas de presentación de los hosts y administra el intercambio de datos. Esta capa tiene habilidades para lograr una transferencia de forma eficiente ya que cuenta con un registro de excepciones de los problemas entre la capa de sesión, presentación y aplicación.

Capa 4: La capa de transporte se encarga de la transferencia de los datos entre el emisor y el receptor asegurando que no contenga errores. Mantiene el flujo en la red y proporciona un transporte de datos confiable de la maquina origen a la maquina destino.

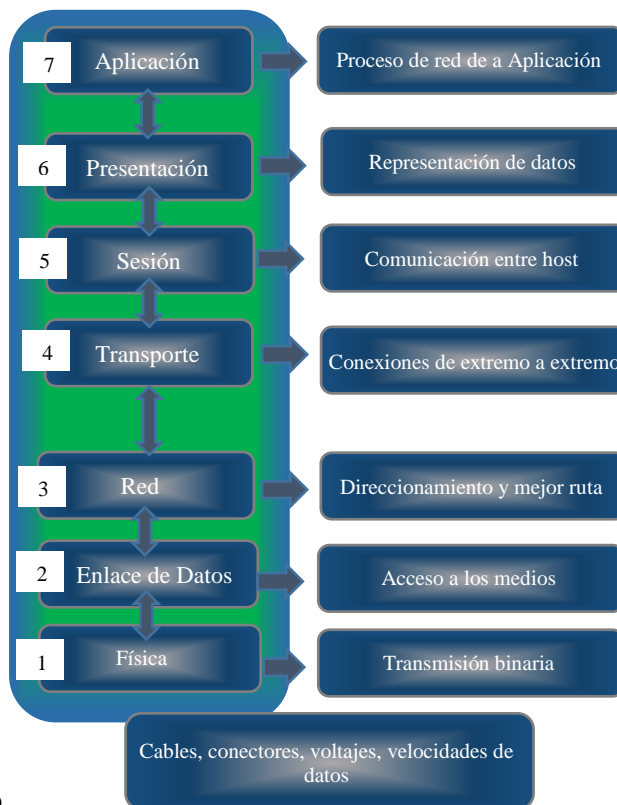


Figura 2-3. Capas modelo OSI y relación con el procesamiento de datos.

Capa 3: **La capa de red**, es una capa compleja que proporciona conectividad y selección de ruta entre dos sistemas de hosts que pueden estar ubicados en redes geográficamente distintas.

Capa 2: **La capa de enlace de datos**, proporciona el tránsito de datos confiable a través de un enlace físico. Al hacerlo, la capa de enlace de datos se ocupa del direccionamiento físico (comparado con el lógico), la topología de red, el acceso a la red, la notificación de errores, entrega ordenada de tramas y control de flujo.

Capa 1: **La capa física**, define las especificaciones eléctricas, mecánicas, de procedimiento y funcionales para activar, mantener y desactivar el enlace físico entre sistemas finales. Las características tales como niveles de voltaje, temporización de cambios de voltaje, velocidad de datos físicos, distancias de transmisión máximas, conectores físicos y otros atributos similares son definidas por las especificaciones de la capa física. [9]

2.3. Arquitectura de Capas

El protocolo CAN sólo define las dos primeras capas del Modelo OSI (física y enlace), aunque no especifica la interfaz al medio físico (medio de transmisión y conectores). La norma ISO 11898 es el estándar internacional para la comunicación de alta velocidad usado en el protocolo de bus CAN. Esencialmente este estándar define las capas física y de enlace de datos. [10]

La capa física se subdivide en tres subcapas, como se muestra en la Figura 2-4.

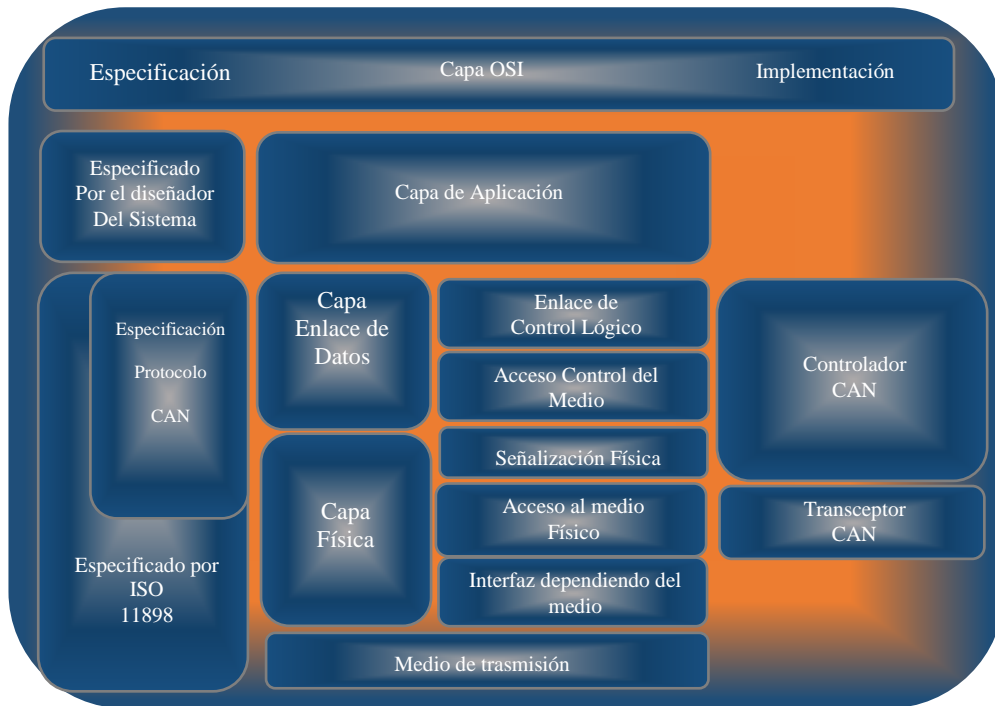


Figura 2-4 Arquitectura de capas del protocolo CAN.

2.4. Capa Física

El medio de comunicación se deberá transmitir y recibir varias señales a la vez, el objetivo del bus CAN es la comunicación entre varios nodos. Además, debe de representar los estados recesivo y dominante.

El diseño de una red CAN varía de acuerdo a las necesidades de desempeño y por lo cual, se deben considerar los requisitos de la capa física. [11]

La especificación del protocolo CAN [12] no define una capa física, sin embargo, los estándares ISO 11898 [13] e ISO 11519 [14] establecen las características que deben cumplir las aplicaciones para la transferencia en alta y baja velocidad. Las características definidas para la

capa física deben implementarse en todos los nodos que se encuentran conectados dentro de una red CAN.

La capa física se divide en tres secciones o subcapas, como se puede ver a continuación Figura 2-5.

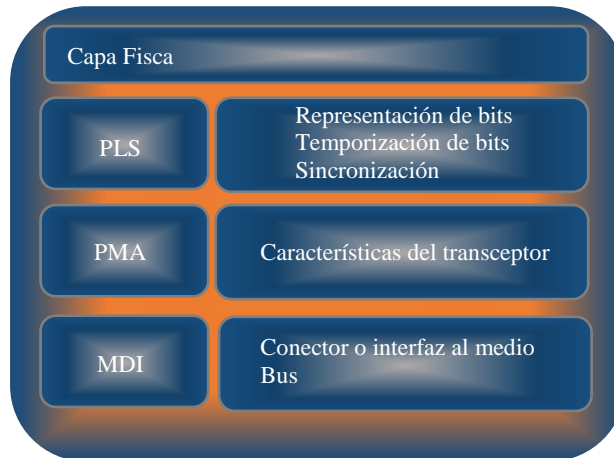


Figura 2-5. Secciones capa física.

2.4.1 Subcapa de señalización física

La subcapa de señalización física PLS (del inglés, *Physical Signalling*) define las funciones relacionadas con la representación, tiempo y sincronización de bits, y esta implementada en los controladores del protocolo CAN.

2.4.2 Representación de bits

Una trama CAN esta codificada de acuerdo con el método NRZ (del inglés, *Non Return to Zero*), en donde se establece que durante todo el tiempo de bit se genera un nivel de señal que puede ser dominante (d) o recesivo (r).

Es importante definir los niveles eléctricos que se manejan para poder decodificar la información que se está recibiendo. Un bit dominante es un “0” lógico presente en algún nodo de la red. Es nombrado bit dominante debido a que hace desaparecer los “1” lógicos de la red. Por otro lado, el bit recesivo es un “1” lógico, como se muestra en Figura 2-6 y Figura 2-7.

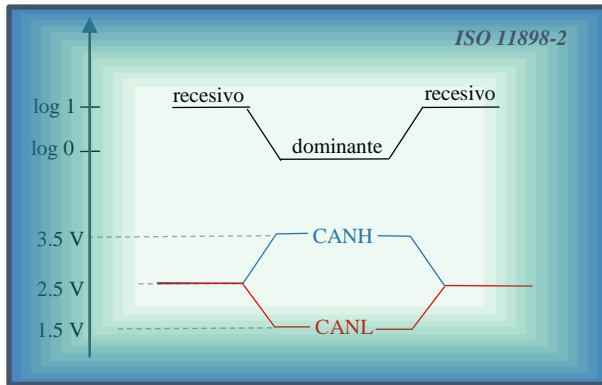


Figura 2-6. Niveles Alta velocidad (High-Speed) CAN.

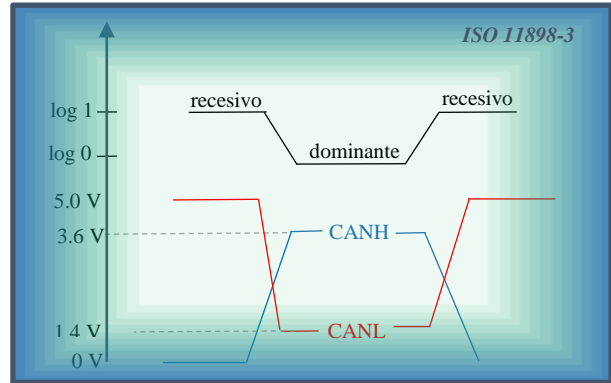


Figura 2-7. Niveles Baja velocidad (Low-Speed) CAN.

El bus CAN utiliza dos líneas eléctricas para representar los bits dominantes y bits recesivos, estas líneas se denominan CAN_High (CANH) y CAN_Low (CANL).

Como se pueden apreciar en las imágenes anteriores, los niveles lógicos se definen:

- Alta velocidad (High Speed) CAN
 - Nivel lógico “1” -> diferencia de voltaje < 0,5 V
 - Nivel lógico “0” -> diferencia de voltaje > 0,9 V
- Baja velocidad (Low Speed) CAN
 - Nivel lógico “1” -> diferencia de voltaje de 5 V
 - Nivel lógico “0” -> diferencia de voltaje de 2V

En el caso de transmitir un alto número de bits en el mismo estado, la codificación NRZ no proporciona flancos que puedan utilizarse en la sincronización. Por ello, se implementó el procedimiento de inserción de bit (*bit-stuffing*) que asegura la transmisión de una trama donde sólo puede haber un máximo de cinco bits consecutivos en el mismo estado como se muestra en la Figura 2-8.

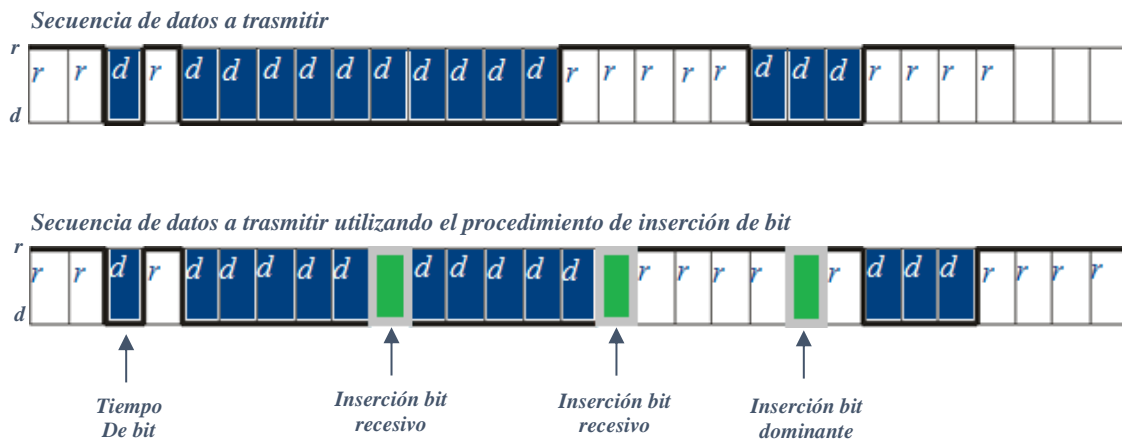


Figura 2-8. Ejemplo del procedimiento de inserción de bit.

2.4.3 Temporización de bits

El protocolo CAN es flexible para determinar los parámetros de velocidad de transferencia, punto de muestra de bit y número de muestras realizadas en un periodo de bit. Al diseñar una red CAN se deben considerar los siguientes conceptos:

- *Tiempo de bit (t_b):* es tiempo de duración de un bit.
- *Velocidad de transferencia nominal (f_b):* son los bits por segundo que un transmisor ideal emite sin resincronización.
- *Tiempo de bit nominal:* se obtiene mediante la ecuación 1-1 y se divide en cuatro segmentos de tiempo no traslapados (Figura 2-9). La longitud de los segmentos de tiempo en un intervalo de bit está definida por múltiplos enteros de la unidad básica de tiempo (t_q *time quantum*) derivada del periodo de oscilador t_{CLK} (Figura 2-10). El parámetro t_q es la unidad de tiempo discreta más pequeña utilizada por un nodo CAN.

$$t_b = \frac{1}{f_b} \quad \text{Ecuación 1.1}$$

Los cuatro segmentos que forman un tiempo de bit nominal son:

- *Segmento sincronización (Sync_Seg)*: se utiliza para compensar los diferentes nodos en el bus mediante un flanco dentro del mismo segmento.
- *Segmento de tiempo de propagación (Prop_seg)*: se utiliza para compensar los tiempos de retardos físicos originados por la propagación de la señal en el bus y por los retardos internos de los nodos [12]–[14].

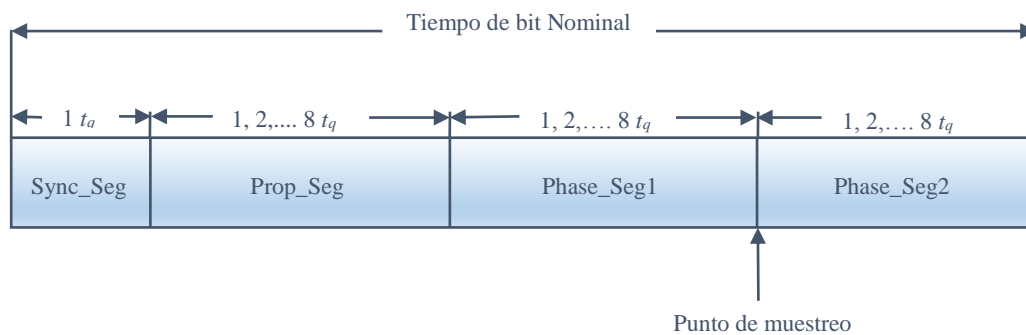


Figura 2-9. Segmentos del tiempo de bit.

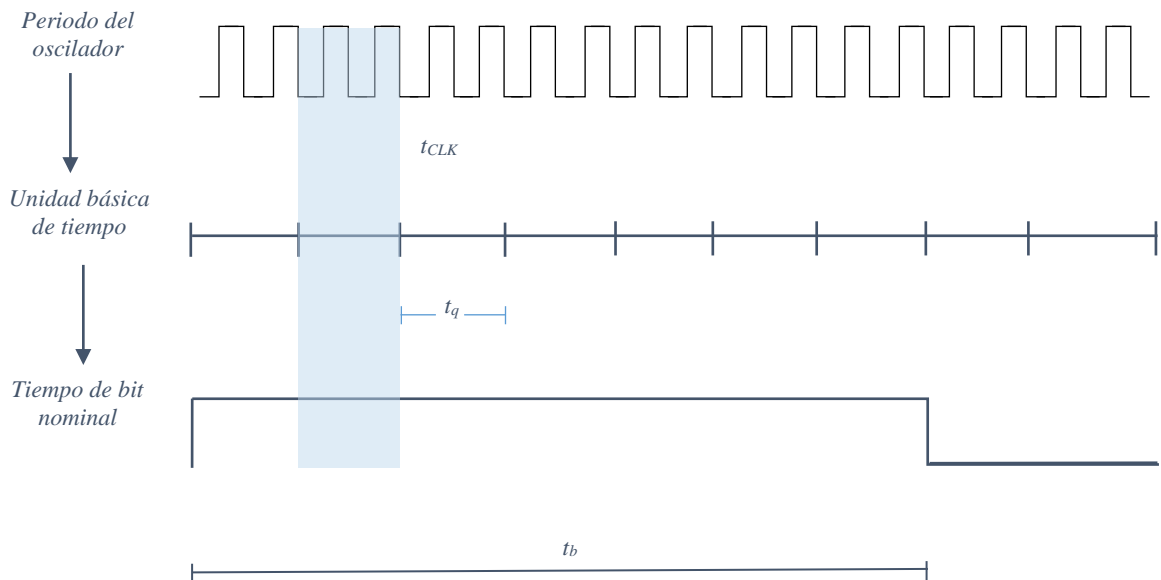


Figura 2-10. Principio de derivación del periodo de bit.

- *Segmentos de memoria temporal de fase1 (Phase_Seg1)*: segmento de longitud a la medida que se utiliza para compensar variaciones de tiempo entre nodos, puede incrementarse durante la resincronización.
- *Segmentos de memoria temporal de fase2 (Phase_Seg2)*: segmento de longitud a la medida que se utiliza para compensar variaciones de tiempo entre nodos, puede reducirse durante la resincronización. [12]–[14]

El número total de unidades básicas en un tiempo de bit debe ser programable al menos de 8 a $25 t_q$.

- *Punto de muestreo (sample point)*: intervalo de tiempo en el que se lee y se descifra el nivel del bus y da el valor de bit respectivo.
- *Tiempo de procesamiento de la información*: es el fase de tiempo que comienza con el punto de muestreo y se usa para calcular el nivel de bit siguiente [12]–[14].

2.4.4 Mecanismos de sincronización de bits

La transferencia de datos en el protocolo CAN es síncrona, esto incrementa la capacidad de transmisión. Para lograrlo, se requiere un método de sincronización de bits debido a que solo está disponible un bit de inicio al comienzo de la trama, lo cual no es suficiente para mantener sincronizado el muestreo de bit del receptor con el del transmisor, y por lo tanto, se requiere realizar una resincronización continua del receptor. [15]

En la red CAN, todos los nodos que la forman se sincronizan con su propio oscilador, debido a que pueden presentarse des fase en algunos nodos. De esta manera, los controladores CAN cuentan con un mecanismo de sincronización para equilibrar dichos desplazamientos mientras reciben una trama de CAN.

Se cuentan con dos tipos de sincronización

- *Sincronización al inicio de la trama (Hard synchronization)* en donde posteriormente a una sincronización, se reinicia el tiempo de bit al finalizar el segmento de sincronización, sin tomar en cuenta un error de fase, lo cual obliga al flanco a caer dentro del segmento de sincronización del bit reiniciado.
- *Resincronización dentro de la trama (Soft synchronization)* es cuando la dimensión de un error de fase del flanco que genere la resincronización es menor o igual al valor programado del RJW (del inglés, *Resynchronization Jump Width*), el efecto de la resincronización es el mismo que el de una sincronización cuando la magnitud del error de fase es mayor que el valor de RJW:
 - Si el error de fase es positivo: el segmento de fase 1 se prolonga por una cantidad igual al valor de RJW;
 - Si el error de fase es negativo: el segmento de fase 2 se reduce por una cantidad igual al valor de RJW.

Ambas formas de sincronización siguen las siguientes reglas de acuerdo a la especificación de CAN versión 2.0 [12]:

1. *“Dentro de un tiempo de bit sólo se permite una sincronización.*
2. *Para efectos de sincronización se utiliza un flanco sólo si el valor detectado en el punto de muestreo anterior difiere del valor del bus inmediatamente después del flanco.*
3. *Se realiza una sincronización siempre que haya un flanco recesivo a dominante, y cuando el bus esté desocupado (bus idle).*
4. *Todos los flancos recesivos a dominantes, opcionalmente los flancos dominantes a recesivos en el caso de bajas velocidades de transferencia, que satisfagan las reglas 1 y 2 se utilizan para una re sincronización, a excepción de que un nodo que se encuentre transmitiendo un bit dominante no puede realizar re sincronización como resultado de*

una flanco recesivo a dominante con un error de fase positivo, si sólo son utilizados para la re sincronización flancos recesivos a dominantes”.

2.5. Subcapa de unión al medio físico

La subcapa de unión al medio físico (PMA, del inglés, *Physical Medium Attachment*), consiste en la conexión de un controlador CAN y el medio físico. Aquí se especifican las propiedades funcionales y eléctricas las que debe tener el transceptor para hacer posible la transmisión/recepción entre un nodo y la red. También, algunas de sus implementaciones en circuitos integrados o discretos facilitan la detección de fallos en el bus.

La interfaz eléctrica con el bus se basa específicamente en un transmisor y un receptor. En la Figura 2-11 podemos apreciar el diagrama a bloques básico de un transceptor, que consiste en un comparador en la recepción y un amplificador en la salida.

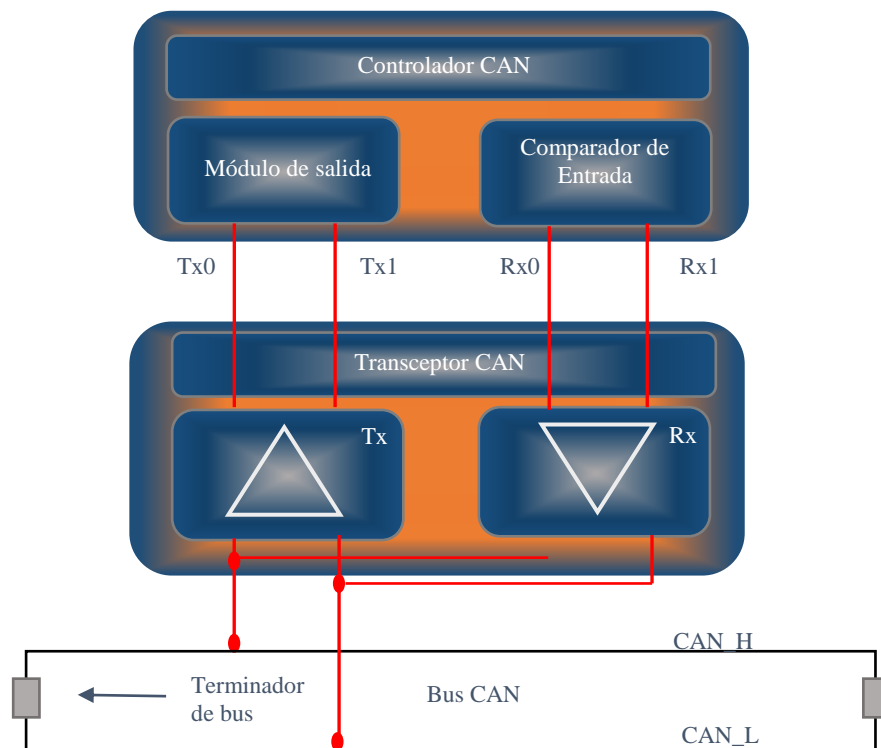


Figura 2-11. Arquitectura de un nodo CAN con un transceptor.

Aparte del arreglo de señales entre el controlador CAN y el bus, el transceptor tiene que cumplir con las siguientes funciones:

- *“Suministrar la capacidad de rendimiento al controlador CAN.*
- *Proteger al controlador CAN contra sobrecargas.*
- *Reducir la radiación electromagnética.” [10]*

Como receptor realiza las siguientes funciones:

- *“Suministrar un nivel de señal recesivo.*
- *Proteger el comparador de entrada de controlador CAN contra sobre voltajes de las líneas del bus.*
- *Suministrar suficiente sensibilidad para un mejor reconocimiento de la señal de entrada.*
- *Detectar errores en el bus como: ruptura de la línea, corto-circuito y conmutación a operación asimétrica de una sola línea” [10].*

2.6. Subcapa de interfaz dependiente del medio

En la interfaz dependiente del medio (MDI, *Medium Dependent Interfaz*) se especifican las características eléctricas mecánicas para la conexión entre el medio físico y el nodo. Estas características mecánicas de la capa física definen la interfaz con respecto al medio de transmisión y al tipo de conector.

2.6.1 Medio físico

Para realizar el método de arbitraje en el bus, el medio físico debe representar los niveles de señal recesivo y dominante. El principio se realiza con medios eléctricos.

2.6.2 Medio de transmisión eléctrico

El desarrollo de redes CAN habitualmente usa como medio físico un bus de dos hilos con retorno común controlados diferencialmente. (Figura 2-12 A) También es frecuente usar el bus de un solo hilo en aplicaciones específicas, las cuales pudieran ser los dispositivos electrónicos de un vehículo (Figura 2-12 B).

Bus de dos hilos (two wire bus): proporciona una transmisión diferencial y es resistente a los errores de modo usual, como se muestra en la Figura 2-12 A. Las líneas del bus deben contar con una resistencia terminadora en los extremos del bus con un valor de 120Ω , esto es para evitar posibles errores en la señal. Con esto se asegura la transmisión de la señal a pesar de sus niveles bajos. Así mismo, para equilibrar la interferencia de radio inducida de forma electromagnética, se usan cables de tipo par trenzado. Al contar con un manejo adecuado de los errores en el bus, es posible que la comunicación continúe en un entorno inmune al ruido aún si una línea se rompe o se encuentra en corto circuito [16].

Bus de un hilo (Single wire bus): en esta configuración se debe tener una tierra común para todos los nodos (Figura 2-12 B) que forman la red. La implementación más común es en las redes automotrices para lograr la conexión de dispositivos electrónicos internos. El bus de un hilo es vulnerable al ruido cuando no está blindado, por tal motivo se requiere realizar un desplazamiento en la señal para lograr una buena relación señal/ruido.

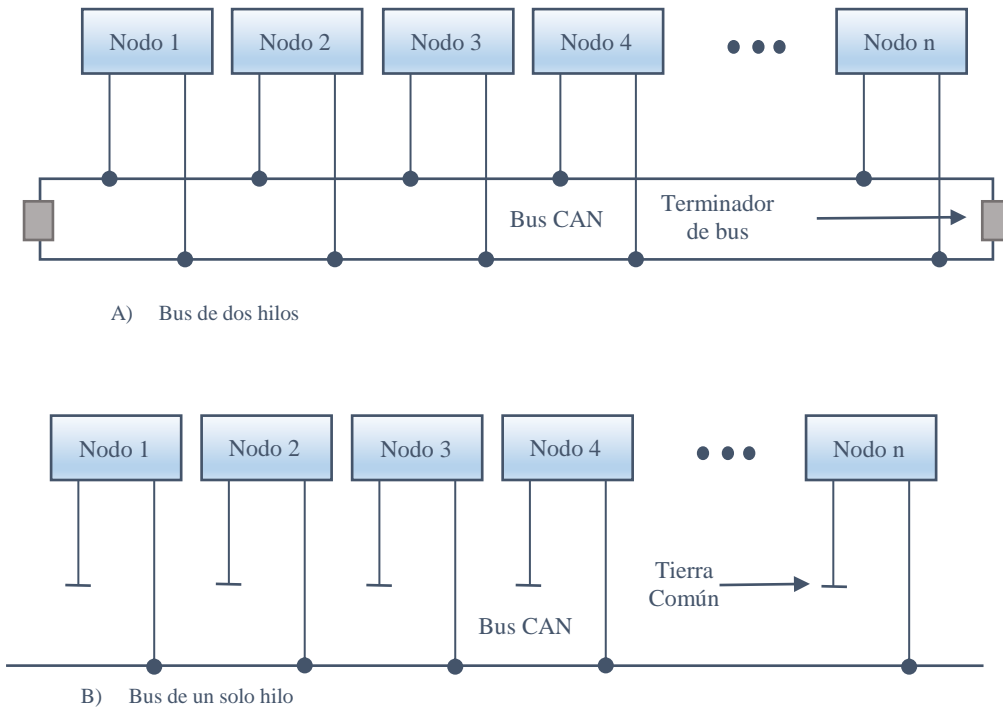


Figura 2-12. Arquitectura de un nodo CAN con transceptor. A) Bus de dos hilos. B) Bus de un solo hilo.

2.7. Capa de enlace de datos

Capa de enlace de datos (de inglés DLL, *Data Link Layer*), está dividida en dos subcapas: control de enlace lógico (de inglés LLC, *Logic Link Control*) y control de acceso al medio (del inglés MAC, *Medium Access Control*). [15]

2.7.1 Control de enlace lógico

La subcapa LLC representa la parte alta de la capa DLL es donde se definen las tareas independientes del método de acceso al medio, prestando dos tipos de servicios de transmisión sin conexión al usuario LLC (*LLC user*):

- *Servicio de transmisión de datos sin reconocimiento*: facilita al usuario LLC los medios para intercambiar unidades de datos de servicio de enlace (de inglés, LSDU, *Link Service Data Units*) sin crear una conexión de enlace de datos. La transmisión de datos puede ser punto a punto, multidifusión o difusión.
- *Servicios de petición de datos remota sin reconocimiento*: facilita al usuario LLC los medios para solicitar que un nodo remoto transmita sus LDSUs sin establecer una conexión de enlace de datos. [15]

Con base a los tipos de servicios mencionados anteriormente, se definen dos formatos de tramas de datos LLC y remota LLC. Los formatos limitan identificadores de 11 bits (formato estándar) y de 29 bits (formato extendido).

2.7.2 Funciones de la subcapa LLC

La subcapa LLC efectúa las siguientes funciones:

Filtrar mensajes (de inglés, *frame acceptance filtering*): en el identificador de una trama no se muestra la dirección destino, pero define el contenido del mensaje. Mediante esta función, todo receptor activo en la red establece si el mensaje es relevante o no para sus propósitos.

Notificar sobrecarga (de inglés, *overload notification*): en las circunstancias internas de un receptor solicitan un retraso en la transmisión de la siguiente trama de datos o remota, la subcapa LLC transmite una trama de sobrecarga. Solamente se pueden generar dos tramas de sobrecarga como máximo.

Proceso de recuperación (de inglés, *recovery management*): la subcapa LLC cuenta con la capacidad de retransmisión automática de tramas cuando una trama pierde el arbitraje o presenta errores durante su transmisión. Este servicio se confirma al usuario hasta que la transmisión se complete con éxito. [15]

2.7.3 Control de acceso al medio

La red CAN da soporte a los procesos en tiempo real a todos los sistemas que la componen. El intercambio de mensajes que solicita el procesamiento requiere de un sistema de transmisión de frecuencias altas y retrasos mínimos [15]. En redes donde todos los nodos pueden ser maestros, la técnica de accesos al medio es muy importante ya que todo nodo activo tiene derechos para controlar la red y tomar los recursos.

Para acceder al medio, los nodos CAN usan un mecanismo de arbitraje que se muestra en la Figura 2-13. CAN utiliza la función Wired-AND, la cual consiste en detectar que un nodo transmita un bit dominante para que el bus pase al estado dominante sólo en caso que todos los nodos de la red transmitan bits recesivos, el bus deberá pasar al estado recesivo (Tabla 2-1 y Figura 2-13). Es decir, que un bit recesivo representa un estado de bus que puede ser sobrescrito por un bit dominante.

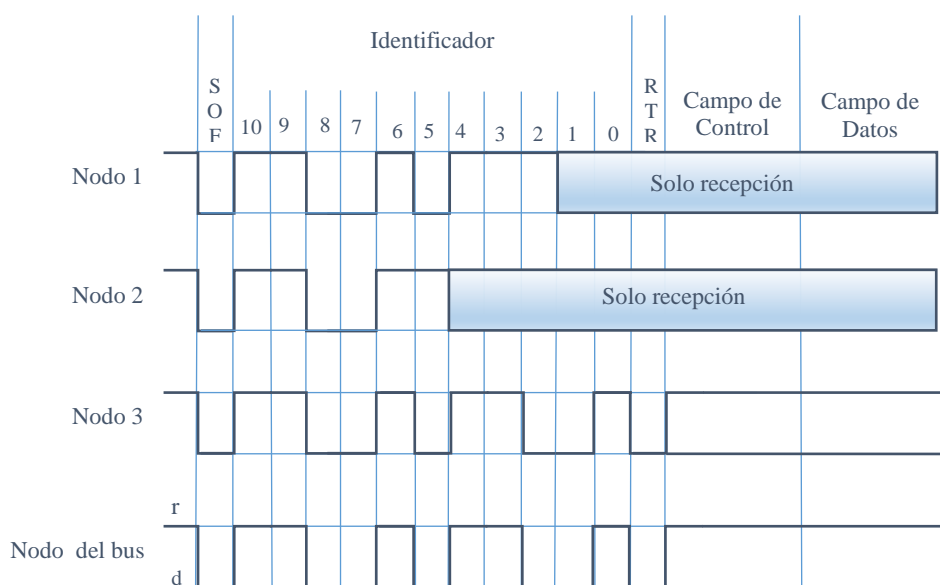


Tabla 2-1
Lógica del bus CAN

Nodo 1	Nodo 2	Nodo 3	Bus
d	d	d	d
d	d	r	d
d	r	d	d
r	d	d	d
r	d	r	d
r	r	d	d
r	r	r	r

Figura 2-13 Lógica del bus CAN.

Para transmitir información de un nodo a través de una red CAN, la capa de aplicación realiza una petición, de forma asíncrona, para transmitir una trama. En el momento que varios nodos inicien el mismo procedimiento simultáneamente, CAN asigna prioridades mediante el identificador de cada mensaje, donde dicha asignación se realiza durante el diseño del sistema en forma de números binarios y no puede modificarse dinámicamente. El identificador con el menor número binario es el que tiene mayor prioridad.

El procedimiento de acceso al medio usado es el de acceso múltiple por detección de portadora, con localización de colisiones y arbitraje por prioridad del mensaje (del inglés, CSMA/CD+AMP, *Carrier Sense Multiple Access with Collision Detection and Arbitration Message Priority*). En base a este método, los nodos en la red que necesitan enviar información deben esperar a que el bus este libre (detección de portadora). Cuando se cumple esta condición, estos nodos transmiten un bit de inicio (acceso múltiple). Cada nodo lee el bus bit a bit durante la transmisión de la trama y comparan el valor transmitido con el valor recibido; mientras los valores sean idénticos, el nodo continúa con la transmisión. Sin embargo, si se detecta una diferencia en los valores de los bits, se inicia un mecanismo de arbitraje.

En la Figura 2-13, mientras se ejecuta la transmisión del bit 5, el nodo 1 transmite un bit dominante y el nodo 2 transmite un bit recesivo; el nivel dominante sobrescribe al recesivo y los dos nodos comparan los valores transmitidos con los que reciben. Después, el nodo 2 muestra una diferencia (detección de colisión) que causa la pérdida del arbitraje e inmediatamente deja de transmitir para cambiar a modo de recepción. Entretanto, el nodo 1 continúa transmitiendo (AMP) hasta que se presenta una situación similar en la transmisión del bit 2. Al final, el nodo 3 es quien gana el arbitraje y transmite su trama. Esto se conoce como arbitraje no destructivo bit a bit, o sea, que a pesar de que varios nodos inicien la transmisión de sus tramas al mismo tiempo, el ganador del arbitraje es el mensaje con mayor prioridad, el cual continúa con la transmisión de su trama sin necesidad de retransmitirla desde el principio. Si lo anterior se detecta en un campo distinto al bit de inicio, incluyendo el campo de arbitraje y ranura ACK, se activa el proceso de control de errores por parte del administrador del nodo, quien lo considera un error de bit [11].

Al iniciar simultáneamente la transmisión de una trama de datos y una trama remota solicitada por un receptor en el proceso de arbitraje, no es posible resolverse únicamente con el identificador de la trama, sino que tiene que utilizarse el bit RTR.

El arbitraje utilizado en CAN limita la extensión máxima de la red a una velocidad de transferencia de datos específica [15].

2.7.4 Trasmisión de mensaje

El formato de trama de datos (*data frame*) cuenta con dos formatos que se caracterizan por una diferencia en la longitud del campo del identificador el primero de ellos es la trama estándar (*standard frame*) que cuenta con un identificador de 11 bits definidas en la especificación CAN 2.0 A y la segunda es la trama extendida (*extended frame*) que cuenta con un identificador de 29 bits de acuerdo a la especificación CAN 2.0B[16].

Para el control y transferencia de mensajes CAN, se definen cuatro tipos de tramas: datos (*data frame*), remota (*remote frame*), de error (*error frame*) y de sobrecarga (*overload frame*).

Las tramas remotas usan ambos formatos, estándar y extendido, y tanto las tramas de datos como las remotas se separan de tramas anteriores mediante espacios entre tramas (*interframe space*).

2.7.5 Trama de datos.

Una trama de datos está compuesta por siete campos (Figura 2-14).

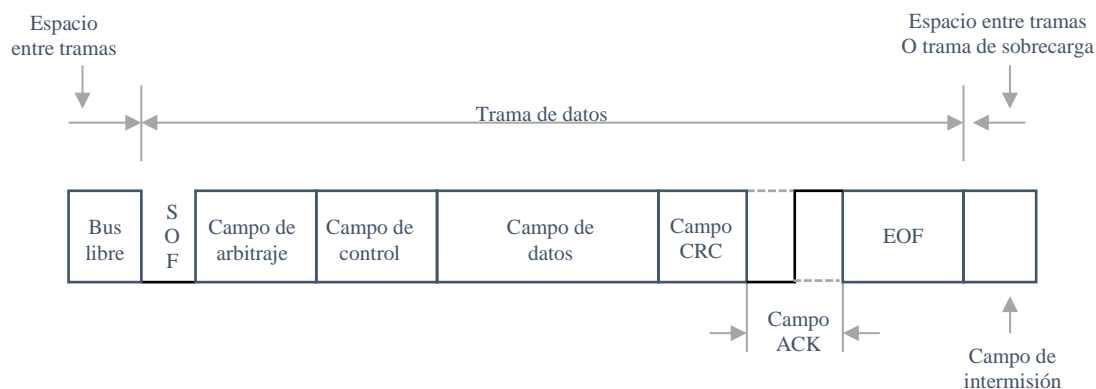


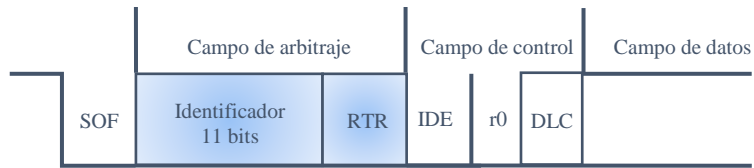
Figura 2-14. Formato de trama de datos CAN.

Inicio de trama (SOF, *Start of Frame*): muestra el inicio de una trama de datos o una trama remota, que se indica con un bit dominante que sincroniza a todos los nodos activos en la red. Se usa lo mismo para la trama estándar y extendida.

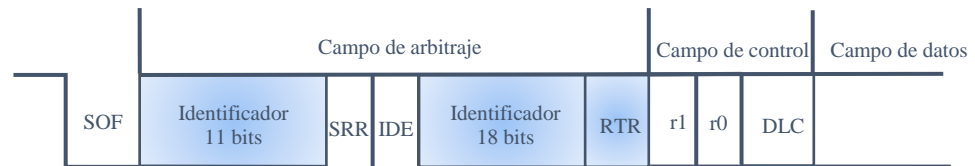
- Campo de arbitraje (*Arbitration field*): es diferente según el formato de trama (Figura 2-15).

En el formato estándar está formado por un identificador de 11 bits y el bit de petición de transmisión remota (RTR, *Remote Transmission Request*). El bit menos significativo del identificador se transmite al último y los 7 bits más significativos no pueden ser todos recesivos.

El formato extendido está formado por un identificador de 29 bits, el bit de petición remota substituta (SRR, *Substitute Remote Request*), el bit de extensión del identificador (IDE, *Identifier Extension*) y el bit RTR. El identificador se divide en dos secciones, la primera de 11 bits denominada base (*Base ID*) que corresponde al identificador del formato estándar, y la segunda sección de 18 bits, conocida como extendida (*Extended ID*). Los bits de ambas secciones del identificador se transmiten en orden de mayor a menor prioridad.



Trama estándar.



Trama extendida.

Figura 2-15. Formatos de trama de datos CAN, estándar y extendida.

El bit RTR debe ser dominante para ambos formatos de trama de datos y el bit SRR es un bit recesivo, por lo tanto, las posibles colisiones entre ambos tipos de formatos de trama que tengan el mismo valor en el campo *Base ID*, se resuelven de manera que el formato de trama estándar predomina sobre el formato de trama extendida. En dicha resolución también se involucra el bit IDE, el cual pertenece al campo de arbitraje en el caso de un formato extendido y se encuentra en el campo de control para el caso de un formato de trama estándar. La transmisión del bit IDE es dominante para el formato estándar y recesivo para el extendido.

- Campo de control (*Control field*): está compuesto de seis bits, IDE/r1, r0 y cuatro bits que forman el código de longitud de datos (DLC, *Data Length Code*). El primer bit que se transmite es IDE, el cual distingue entre los dos tipos de tramas; enseguida r0, en nivel dominante y está reservado para futuras aplicaciones del protocolo CAN; finalmente, se transmite el DLC para indicar el número de octetos contenidos en el campo de datos.
- Campo de datos (*Data field*): contiene el mensaje a transmitir dentro de una trama CAN y puede tener una longitud de 0 a 8 octetos. Se transmite primero el bit con mayor prioridad.

- Campo CRC (*CRC field*): está constituido por una secuencia de 15 bits de verificación y un bit delimitador CRC (*CRC delimiter*), este último se transmite en un nivel recesivo. Mediante este campo, el receptor verifica si la secuencia de bits recibidos fue alterada. Se utiliza el polinomio generador $X^{15}+X^{14}+X^{10}+X^8+X^7+X^4+X^3+1$.
- Campo de aceptación (*ACK field*): está constituido por dos bits, ranura ACK (*ACK slot*) y delimitador ACK (*ACK delimiter*), este último siempre se transmite en un nivel recesivo. Todo nodo activo en la red CAN que recibe una trama válida, sobrescribe la ranura ACK con un nivel dominante, y con ello, el transmisor verifica que su mensaje se envió correctamente; si por el contrario ningún nodo sobrescribe dicha ranura, el transmisor considera un error de transmisión.
- *Fin de trama (EOF, End of Frame)*: tanto la trama de datos como la trama remota están delimitadas por una secuencia de 7 bits recesivos que indican el fin de trama CAN. Cuando EOF está activo se realiza una falta al procedimiento de inserción de bit. Por ello, dicho procedimiento no se aplica a este campo.

2.7.6 Trama remota

Un nodo CAN en modo receptor, puede iniciar la transmisión de su información mediante el envío de una trama remota, la cual se compone de seis campos tanto en formato estándar o extendido (Figura 2-16). Los campos de una trama remota son los mismos que los de una trama de datos, a excepción de la trama remota que no contiene el campo de datos y el bit RTR es recesivo. El valor del DLC debe coincidir con el de la trama de datos correspondiente.

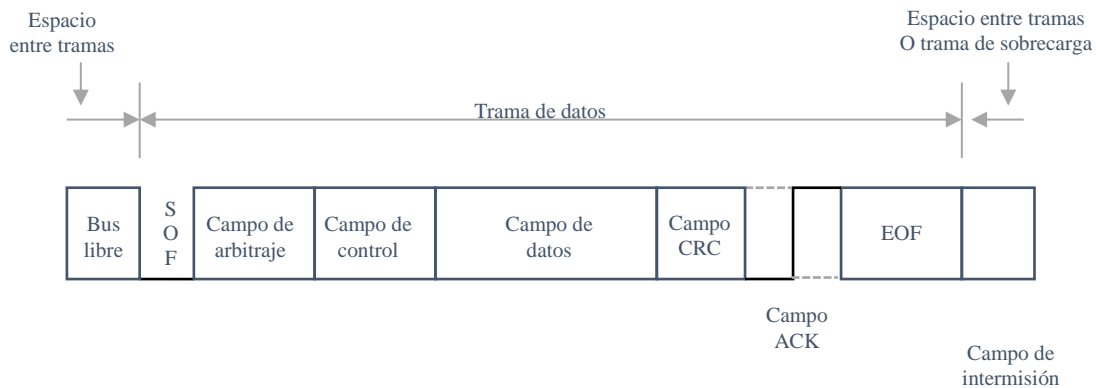


Figura 2-16. Formato de trama de remota CAN.

2.7.7 Trama de error

La trama de error señala la detección de cualquier error durante la transmisión o recepción de una trama de datos o remota, la cual viola el procedimiento de inserción de bit y ocasiona que el transmisor reenvíe la trama. Asimismo, la detección de un error durante la transmisión o recepción de una trama de sobrecarga o error, genera la transmisión de una nueva trama de error. La trama de error está formada por dos campos (Figura 2-17)

Bandera de error (Error flag): existen dos formas de representar una bandera de error:

- Bandera de error activo (Active error flag): consiste en seis bits dominantes consecutivos.
- Bandera de error pasivo (Passive error flag): está formada por seis bits recesivos consecutivos, a menos que sean sobrescritos por bits dominantes de otros nodos.

Delimitador de error (Error delimiter): una trama de error termina con una secuencia de ocho bits recesivos. Posterior a la transmisión de una bandera de error, el nodo transmite bits recesivos y verifica el nivel del bus hasta que reconozca un bit recesivo, entonces comienza la transmisión de otros siete bits recesivos. Con este mecanismo, el nodo puede determinar si fue el primero en transmitir una bandera de error y con ello detectar una condición de error.

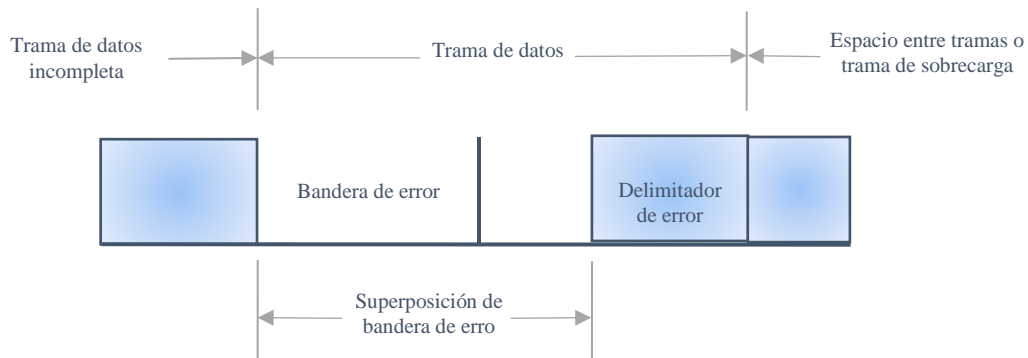


Figura 2-17. Formato de trama de error CAN.

2.7.8 Trama de sobrecarga

La trama de sobrecarga se utiliza para que un receptor solicite un retraso en la transmisión de la trama siguiente, ya sea de datos o remota, o para señalar condiciones de error relacionadas con el campo de intermisión. El protocolo CAN permite la generación de dos tramas de sobrecarga como máximo para retrasar la transmisión de la siguiente trama.

Las tramas de sobrecarga se transmiten después de detectar las siguientes condiciones de error:

- Detección de un bit dominante durante los primeros dos bits del campo de intermisión. La detección de un bit dominante en el tercer bit del campo de intermisión se interpreta como un SOF.
- Cuando un receptor detecta un bit dominante en el último bit del campo EOF, o cuando un nodo, receptor o transmisor, detecta un bit dominante en el último bit del delimitador de una trama de error o de sobrecarga.
-

Una trama de sobrecarga se considera una forma especial de trama de error y consta de los siguientes campos (Figura 2-18):

- Bandera de sobrecarga (*Overload flag*): se constituye por seis bits dominantes. La forma completa corresponde a la bandera de error activa.
- Delimitador de sobrecarga (*Overload delimiter*): está formado por ocho bits recesivos.

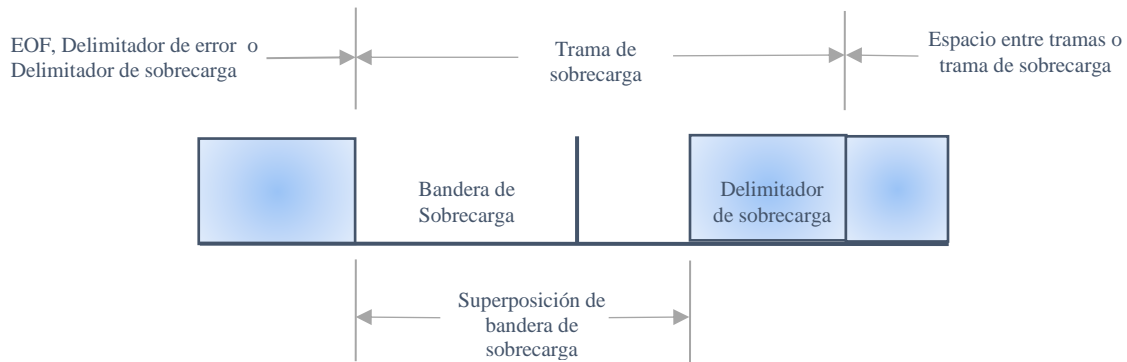


Figura 2-18. Formato de trama de sobrecarga CAN.

2.7.9 Espacio entre tramas

Las tramas de datos y remotas están separadas de tramas precedentes por un espacio entre tramas, a diferencia de las tramas de error y de sobrecarga que se transmiten en forma sucesiva, es decir, sin un espacio entre tramas.

El espacio entre tramas está formado por tres campos:

- Intermisión (*Intermission*): consiste en tres bits recesivos. Durante su transmisión, la única acción que puede realizarse es señalar una condición de sobrecarga, y no se permite que ningún nodo inicie la transmisión de una trama de datos o remota.
- Bus libre (*Bus idle*): este periodo es de longitud arbitraria y tiene un nivel recesivo hasta que algún nodo inicie la transmisión de una nueva trama.

- Suspendir transmisión (*Suspend transmission*): el espacio entre tramas contiene un tiempo de inhibición de transmisión de ocho bits para nodos que se encuentren en estado de error pasivo.

2.7.10 Codificación de tramas

Los campos inicio de trama, arbitraje, control, datos y secuencia CRC están codificados de acuerdo al procedimiento de inserción de bit. Los campos restantes, delimitador CRC, ACK y EOF, tienen un formato fijo y no siguen el procedimiento de inserción de bit; de igual forma, las tramas de error y de sobrecarga tienen un formato fijo y no se codifican por dicho procedimiento [13].

2.7.11 Validación de tramas

El instante de tiempo en el que se valida una trama difiere según el transmisor y el receptor:

Transmisor: la trama es válida si no existen errores hasta el final del campo EOF. Si existe un error, en la trama se activa el proceso de recuperación.

Receptor: la trama es válida si no existen errores hasta el siguiente bit después del campo EOF.

2.7.12 Detección y manejo de errores

Un controlador CAN cuenta con la capacidad de detectar y manejar los errores que surjan en una red. Todo error detectado por un nodo, se notifica inmediatamente al resto de los nodos.

Un nodo puede tener una alteración local permanente, lo que provoca el envío continuo de tramas de error. Para prevenir dicho comportamiento, el protocolo CAN describe un algoritmo basado en la actividad del bus que obliga a los nodos con errores permanentes a desconectarse de la red (*bus off*), y que los demás nodos no sean perturbados por los nodos defectuosos [11].

2.7.13 Mecanismos de detección de errores

Para cumplir con las demandas relativas a la seguridad en la transmisión de datos, el protocolo CAN define los siguientes mecanismos para detección de errores [13]:

- Monitoreo de bits: todo nodo verifica que el nivel del bus transmitido sea el mismo nivel en el bus, y cuando dichos valores difieren, se detecta un error de bit (bit error). El monitoreo de bits representa un mecanismo de seguridad global para la detección de todos los errores efectivos.
- Verificación del procedimiento de inserción de bit: hace referencia al hecho de detectar un error de inserción de bit cuando ocurren seis niveles consecutivos de bits con el mismo valor en un campo de trama codificado por el procedimiento de inserción de bit (stuff error).
- Verificación de redundancia cíclica de 15 bits: se presenta un error de CRC (CRC error) cuando la secuencia CRC recibida no corresponde con la secuencia CRC calculada.
- Verificación de trama: detecta un error cuando un campo de bit de formato fijo contiene uno o más bits no válidos (form error).
- Verificación de aceptación: un transmisor detecta un error de aceptación (ACK error) cuando la ranura ACK (ACK slot) no cambia a estado.

2.7.14 Manejo de errores

Cuando un nodo detecta algún tipo de error, de bit, de inserción de bit, de forma o de aceptación, inicia la transmisión de una bandera de error (*error flag*) en el siguiente bit. Cuando se detecta un error de CRC, se inicia la transmisión de una trama de error después del delimitador ACK, a excepción de que previamente se haya transmitido otra trama de error.

El manejo de errores se lleva a cabo de acuerdo con el diagrama de flujo de la Figura 2-19.

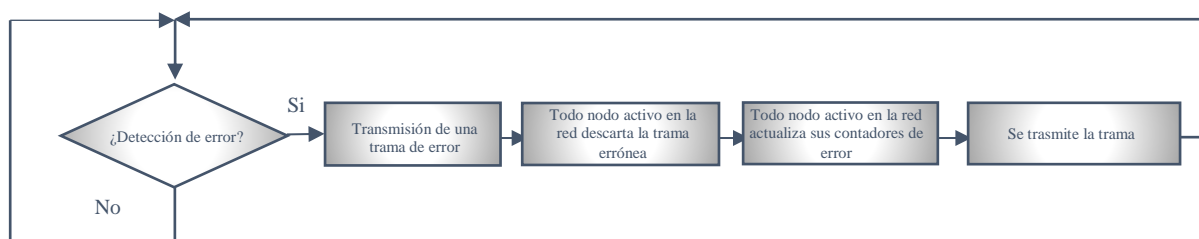


Figura 2-19. Diagrama de flujo para el manejo de errores.

La mayoría de errores de transmisión son debido a interferencias electromagnéticas. La susceptibilidad de error de un sistema de transmisión de datos se puede caracterizar mediante parámetros estadísticos tales como la velocidad de error de trama (*frame error rate*) y la probabilidad de error de bit (*bit error probability*). La velocidad de error (*error rate*) está dada por la relación entre el número de tramas erróneas y el número total de tramas transmitidas durante un periodo de tiempo, con lo cual, se describe la probabilidad de que exista una trama errónea. Por otro lado, la probabilidad de error de bit especifica la probabilidad de que exista un bit erróneo en una trama transmitida.

Los errores de transmisión no ocurren en todos los nodos de una red, sino que aparecen en nodos individuales con diversos patrones de error. El protocolo CAN tiene la capacidad de señalar los errores detectados mediante la transmisión de una bandera de error, sin embargo, existe la posibilidad de no detectar errores locales cuando se cumplen las siguientes condiciones en la distribución del error de bit:

- El nodo que transmite funciona adecuadamente y puede detectar un error al monitorear el bit y señalarlo a todo el sistema.
- Cuando todos los nodos receptores que reciben una trama errónea detectan un patrón de error no definido (indetectable). Debido a que estos patrones son muy raros, todos los nodos que recibieron la trama errónea deben mostrar un patrón de error idéntico. Esta condición es cada vez más improbable en un sistema distribuido con un gran número de nodos.

La DLL del protocolo de comunicaciones CAN garantiza un alto grado de detección de errores. Todos los errores se detectan mediante el proceso de monitoreo de bit realizado por el transmisor de la trama. Mediante dicho proceso, el transmisor puede detectar errores ocasionados a nivel global por interferencia electromagnética la cual aparece en todos los nodos, y además tiene la capacidad de detectar errores locales.

Los errores que sólo ocurren localmente, en receptores individuales, se detectan mediante la verificación de la secuencia CRC de 15 bits, realizada por el mismo receptor.

3. Metodología

3.1. Hardware utilizado

La tarjeta de evaluación Tiva™ C Series TM4C123G LaunchPad es una tarjeta de evaluación (EK-TM4C123GXL) de bajo costo. Esta plataforma de evaluación es para microcontroladores ARM® Cortex™ -M4F. El diseño de Tiva C series LaunchPad destaca la interfaz del microcontrolador TM4C123GH6PMI con el dispositivo USB 2.0, el cual cuenta con un módulo de hibernación y controla el modulador de ancho de pulso (MC PWM). También cuenta con botones de usuario programable y un LED RGB para aplicaciones personalizadas [17].

Uno de los principales motivos para usar esta tarjeta se debe a que cuenta con un módulo de red CAN en el cual se basa el desarrollo principal del proyecto. Los pines principales de Tiva C Series TM4C123G LaunchPad BoosterPack XL, usados como interfaz demuestran lo fácil que es ampliar la funcionalidad del Tiva C Series LaunchPad cuando se conecta a otros periféricos en muchos complementos BoosterPack, así como en productos futuros. La Figura 3-1 muestra una foto de la tarjeta Tiva C Series LaunchPad.

El Tiva C Series LaunchPad puede alimentarse desde una de las dos fuentes de alimentación con las que cuenta:

- Cable USB ICDI integrado (depuración, predeterminado)
- Cable del dispositivo USB (Dispositivo)

El interruptor POWER SELECT (SW3) se utiliza para seleccionar una de las dos fuentes de alimentación. También cuenta con un LED RGB. Este LED se usa en RGB precargado aplicación de inicio rápido y se puede configurar para usar en aplicaciones personalizadas. Dos botones de usuario están incluidos en el tablero. Los botones de usuario se usan en el inicio rápido precargado aplicación para ajustar el espectro de luz del LED RGB y para entrar y salir de la

hibernación. El usuario puede usar los botones para otros fines en aplicación personalizada. La placa de evaluación cuenta con un LED de energía verde.

En la Tabla 3-1 (tomada de “Tiva™ C Series TM4C123G LaunchPad Evaluation Board”) se muestra como las características físicas mencionadas están conectadas a los pines en el microcontrolador.

Tabla 3-1. Interruptores de usuario y señal de Led RGB asociadas a las salidas GPIO

GPIO Pin	Pin Function	USB Device
PF4	GPIO	SW1
PF0	GPIO	SW2
PF1	GPIO	RGB LED (Red)
PF2	GPIO	RGB LED (Blue)
PF3	GPIO	RGB LED (Green)

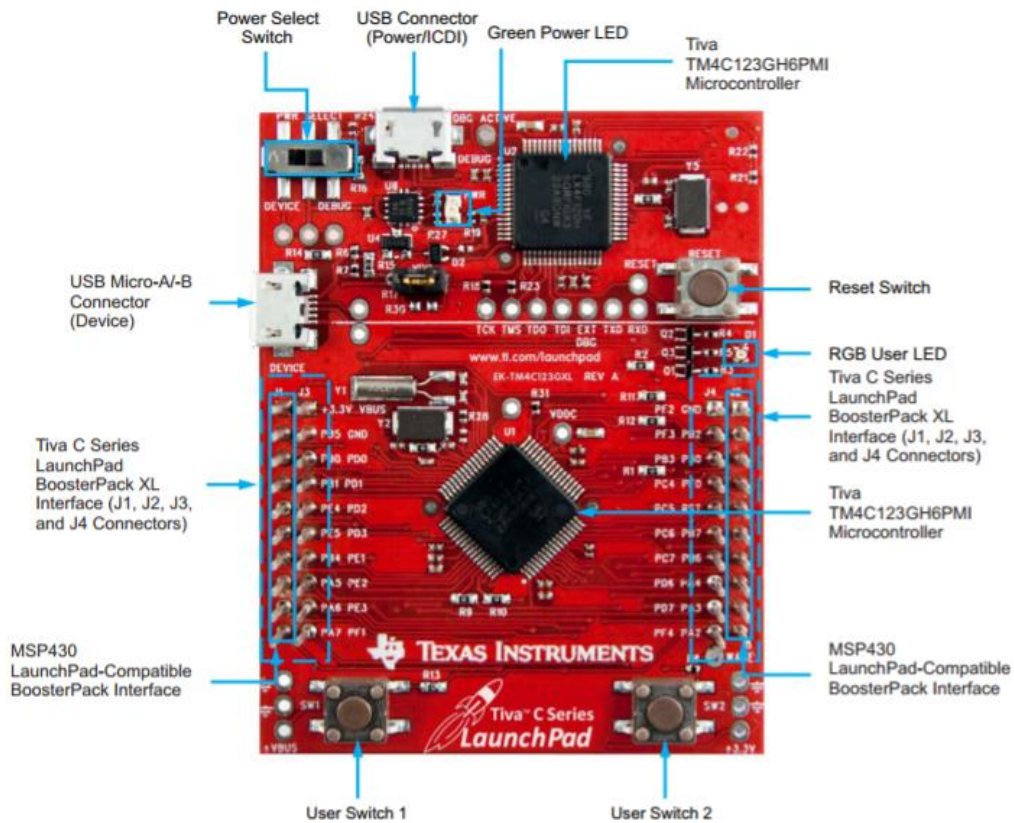


Figura 3-1. Foto de la tarjeta Tiva C Series LaunchPad.

3.1.1 Descripción general de la serie Tiva™ C

Los microcontroladores ARM Cortex-M4 de la serie Tiva™ proporcionan un rendimiento superior y avanzada integración. La familia de productos está posicionada para aplicaciones que requieren un control significativo en el controlar del procesamiento y las capacidades de conectividad tales como:

- Automotriz
- Dispositivos inteligentes de mano y de baja potencia
- Equipo de juego
- Monitoreo y control de sitios domésticos y comerciales
- Control de movimiento
- Instrumentación médica
- Equipos de prueba y medición
- Automatización de procesos
- Fuego y seguridad
- Soluciones de Smart Energy / Smart Grid
- Control de iluminación inteligente
- Transporte

3.1.2 Características Microcontrolador TM4C123GH6PM

El microcontrolador TM4C123GH6PM combina una integración compleja y un alto rendimiento en las características que se muestran en la Tabla 3-2 (Tomada de “Tiva™ TM4C123GH6PM Microcontroller Data Sheet”). [18]

Tabla 3-2. Características Microcontrolador TM4C123GH6PM

Feature	Description
Performance	
Core	ARM Cortex-M4F processor core
Performance	80-MHz operation; 100 DMIPS performance
Flash	256 KB single-cycle Flash memory
System SRAM	32 KB single-cycle SRAM
EEPROM	2KB of EEPROM
Internal ROM	Internal ROM loaded with TivaWare™ for C Series software
Security	
Communication Interfaces	
Universal Asynchronous Receivers/Transmitter (UART)	Eight UARTs
Synchronous Serial Interface (SSI)	Four SSI modules
Inter-Integrated Circuit (I ² C)	Four I ² C modules with four transmission speeds including high-speed mode
Controller Area Network (CAN)	Two CAN 2.0 A/B controllers
Universal Serial Bus (USB)	USB 2.0 OTG/Host/Device
System Integration	
Micro Direct Memory Access (μDMA)	ARM® PrimeCell® 32-channel configurable μDMA controller
General-Purpose Timer (GPTM)	Six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks
Watchdog Timer (WDT)	Two watchdog timers
Hibernation Module (HIB)	Low-power battery-backed Hibernation module
General-Purpose Input/Output (GPIO)	Six physical GPIO blocks
Advanced Motion Control	
Pulse Width Modulator (PWM)	Two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs.
Quadrature Encoder Interface (QEI)	Two QEI modules
Analog Support	
Analog-to-Digital Converter (ADC)	Two 12-bit ADC modules, each with a maximum sample rate of one million samples/second
Analog Comparator Controller	Two independent integrated analog comparators
Digital Comparator	16 digital comparators
JTAG and Serial Wire Debug (SWD)	One JTAG module with integrated ARM SWD
Package Information	
Package	64-pin LQFP
Operating Range (Ambient)	Industrial (-40°C to 85°C) temperature range Extended (-40°C to 105°C) temperature range

3.1.3 Especificaciones

En la siguiente Tabla 3-3 (Tomada de “Tiva™ C Series TM4C123G LaunchPad Evaluation Board”)[17] se muestra un resume de las especificaciones del Tiva C Series LaunchPad.

Tabla 3-3. Especificaciones principales EKTM4C123GH6PM

Parameter	Value
Board supply voltage	4.75 V _{DC} to 5.25 V _{DC} from one of the following sources: <ul style="list-style-type: none"> • Debugger (ICDI) USB Micro-B cable (connected to a PC) • USB Device Micro-B cable (connected to a PC)
Dimensions	2.0 in x 2.25 in x 0.425 in (5.0 cm x 5.715 cm x 10.795 mm) (L x W x H)
Break-out power output	<ul style="list-style-type: none"> • 3.3 V_{DC} (300 mA max) • 5.0 V_{DC} (depends on 3.3 V_{DC} usage, 23 mA to 323 mA)
RoHS status	Compliant

3.2. Descripción funcional

3.2.1 Microcontrolador

El TM4C123GH6PM es un microcontrolador ARM Cortex-M4 de 32 bits con memoria flash de 256 kB, SRAM de 32 kB y funcionamiento a 80 MHz; está integrado por un Host USB, dispositivo y conectividad OTG, un módulo de hibernación y PWM y una amplia gama de otros periféricos [17]. Para obtener mayor información se puede consultar la hoja de datos del microcontrolador TM4C123GH6PM (Número de publicación SPMS376) [18] para detalles completos del dispositivo.

La mayoría de las señales del microcontrolador se enrutan al conector de paso de 0,1 pulgadas (2,54 mm). Un multiplexor interno permite asignar diferentes funciones periféricas a cada uno de las salidas del GPIO.

El microcontrolador TM4C123GH6PM viene programado de fábrica con un programa de demostración de inicio rápido. El programa de inicio rápido reside en la memoria flash en el chip y se ejecuta cada vez que se aplica energía, a menos que la aplicación de inicio rápido haya sido reemplazada por un programa de usuario.

3.3. Módulo de red CAN

La tarjeta Tiva C Series TM4C123G LaunchPad cuenta con un módulo CAN con un estándar de bus serie compartido y multidifusión para conectar electrónicamente unidades de control (ECUs del inglés: *electronic control units*). El módulo CAN fue específicamente diseñado para ser robusto en entornos electromagnéticamente ruidoso y puede utilizar una línea equilibrada diferencial como RS-485 o un cable par trenzado más robusto. Originalmente creado para automóviles, también se usa el módulo de red CAN en muchas aplicaciones de controles integrados (como industrial y médica). Las velocidades de bits es de hasta 1 Mbps estas son posibles en longitudes de red de menos de 40 metros. Las tasas de bits reducidas permiten distancias de red más largas (por ejemplo, 125 Kbps en 500 metros).

El microcontrolador TM4C123GH6PM incluye dos periféricos de CAN con las siguientes características:

- Protocolo CAN versión 2.0 parte A/B.
- Tasas de bits de hasta 1 Mbps.
- 32 objetos con mensaje enmascarables de identificaciones individuales.
- Interrupción enmascarable.
- Deshabilitar el modo de retransmisión automática para aplicaciones CAN activadas en el tiempo (TTCAN).
- Modo de bucle invertido programable para la operación de autodiagnóstico.
- El modo FIFO programable permite el almacenamiento de múltiples objetos de mensaje.
- Se conecta sin interrupción a un transceptor CAN externo a través de las señales *CANnTX* y *CANnRX* [18].

3.3.1 Diagrama a bloques controlador CAN

En la Figura 3-2 (Tomada de “Tiva™ TM4C123GH6PM Microcontroller Data Sheet”) se pueden apreciar todos los registros internos involucrados para realizar una transmisión o recepción vía CAN en el microcontrolador.

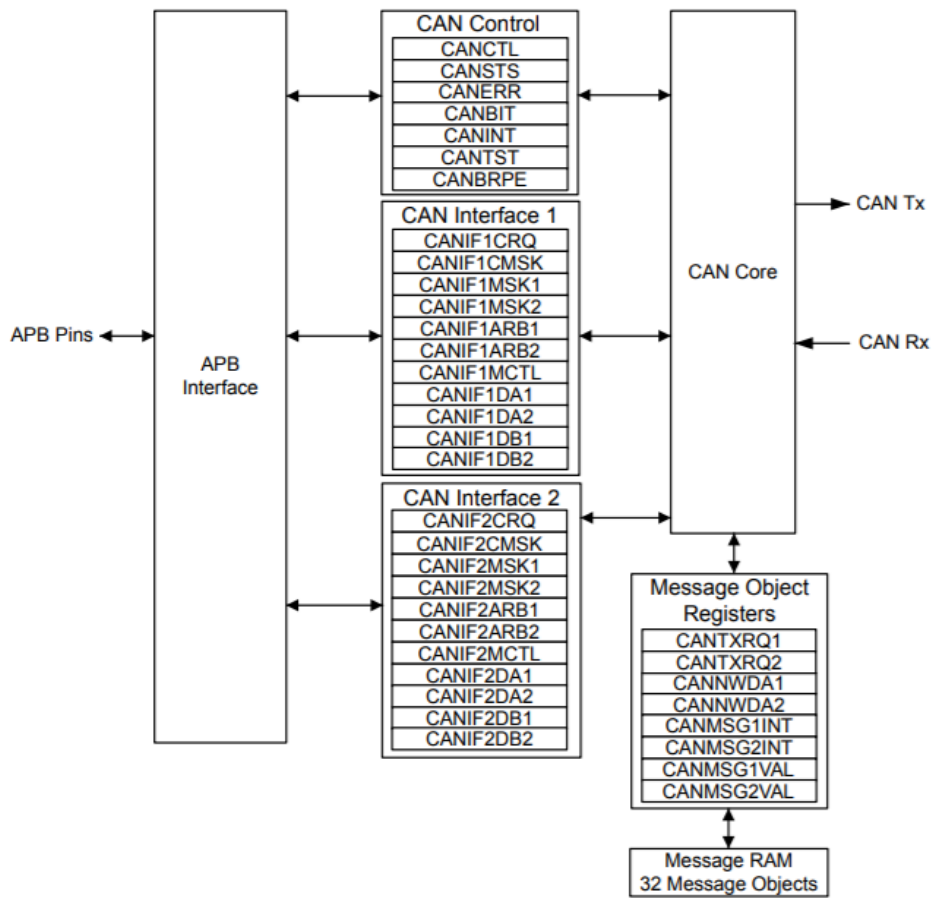


Figura 3-2. Diagrama a bloques del controlador CAN.

3.3.2 Descripción de la señal

La Tabla 3-4 enumera las señales externas del controlador CAN y describe la función de cada una. Las señales del controlador CAN son funciones alternativas para algunas señales GPIO y el valor predeterminado de las señales GPIO al reiniciar. La columna titulada "Pin Mux / Pin Assignment" en la Tabla 5, enumera las posibles ubicaciones de pin GPIO para las señales CAN.

El registro de selección (*GPIOAFSEL*) debe configurarse para elegir la función del controlador CAN. Los números entre paréntesis es la codificación que debe programarse en el campo PMCn en el registro GPIO. Para realizar la asignación del control de los puertos se hace en el registro (*GPIOCTL*) al configurar el registro se asigna la señal CAN al puerto GPIO en el pin especificado de la configuración.

Tabla 3-4. Señales de red de área del controlador (64LQFP)
(Tomada de "Tiva™ TM4C123GH6PM Microcontroller Data Sheet")

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type ^a	Description
CAN0Rx	28 58 59	PF0 (3) PB4 (8) PE4 (8)	I	TTL	CAN module 0 receive.
CAN0Tx	31 57 60	PF3 (3) PB5 (8) PE5 (8)	O	TTL	CAN module 0 transmit.
CAN1Rx	17	PA0 (8)	I	TTL	CAN module 1 receive.
CAN1Tx	18	PA1 (8)	O	TTL	CAN module 1 transmit.

3.3.3 Descripción funcional

El controlador TM4C123GH6PM CAN se desempeña con el protocolo versión 2.0 (partes A y B). Los mensajes de transferencia incluyen marcos de datos, remotos, de error y de sobrecarga con un

identificador de 11 bits (estándar) o un identificador de 29 bits (extendido), los cuales son compatibles entre sí. Las velocidades de transferencia se pueden programar hasta 1 Mbps.

El módulo CAN consta de tres partes principales:

- Controlador de protocolo CAN y manejador de mensajes
- Memoria de mensajes
- Registros de interfaz CAN

Un marco de datos contiene datos para la transmisión, mientras que un marco remoto no contiene datos y se usa para solicitar la transmisión de un objeto de mensaje específico. Las tramas de CAN datos/remoto están contruidos como se muestra en la Figura 3-3 (Tomada de “Tiva™ TM4C123GH6PM Microcontroller Data Sheet”) [18] .

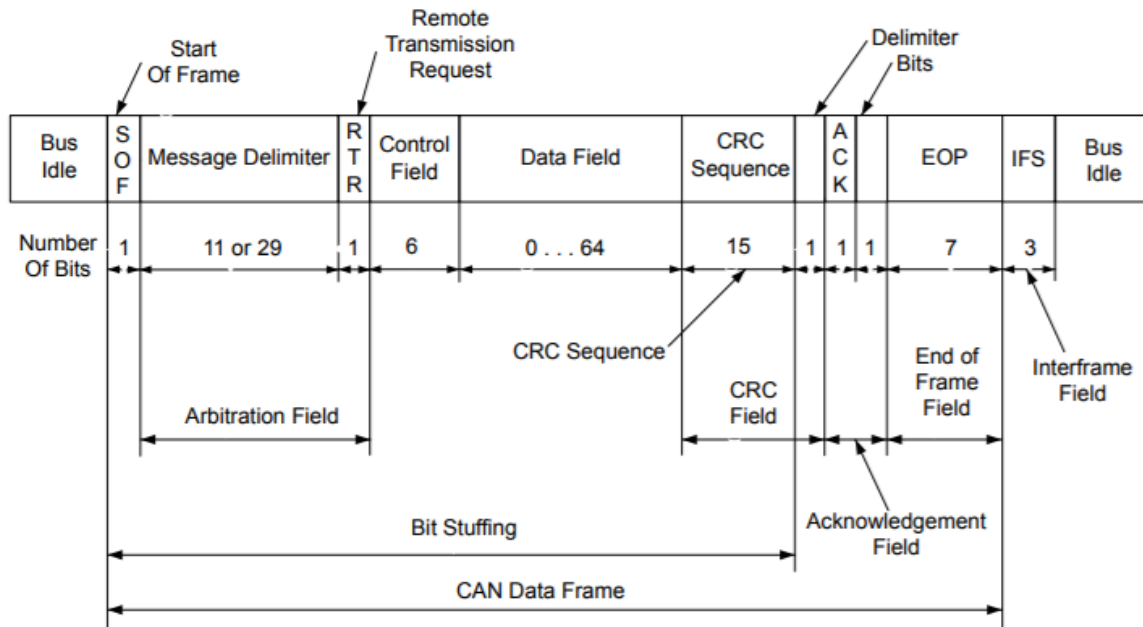


Figura 3-3. Tramas de CAN de Datos/Remota.

El controlador de protocolo transfiere y recibe los datos en serie del bus CAN y pasa los datos al manejador de mensajes. El manejador de mensajes carga esta información en el identificador apropiado basado en el filtro actual del mensaje. El controlador de mensaje también

es responsable de la interrupción de eventos, que se generan en el bus CAN. La memoria del identificador mensaje contiene 32 bloques de memoria idénticos que contienen la configuración actual y los datos de cada identificador de mensaje. También se puede acceder a esos bloques de memoria a través de los registro de interface de CAN. A la memoria de identificador de mensajes no se puede acceder directamente del mapa de memoria del microcontrolador TM4C123GH6PM, por lo que el controlador CAN proporciona una interfaz para comunicarse con la memoria de identificador de mensajes a través de dos registros de interfaz CAN que permiten comunicarse con los identificadores de mensajes. Estas dos interfaces deben usarse para leer y escribir cada mensaje. Las interfaces permiten acceso paralelo a los mensajes del controlador CAN cuando múltiples objetos pueden tener nueva información que debe ser procesada. En general, una interfaz se usa para transmitir datos y otra para recibir datos.[18]

3.3.4 Inicialización

Para usar el controlador CAN, se debe habilitar el reloj del periférico, esto se hace por medio del registro *RCGC0*. Además, el reloj del módulo *GPIO* apropiado debe estar habilitado a través del registro *RCGC2*. Para asignar las señales CAN a los pines correspondientes, es necesario configurar los campos *PMCn* en el Registro *GPIOPCTL*.

La inicialización del software se inicia configurando el bit INIT en el registro CAN Control (*CANCTL*) (con software o por un restablecimiento de hardware) o al apagar el bus, lo que ocurre cuando el error del transmisor contador excede un conteo de 255. Mientras se fija el valor del bit INIT, todas las transferencias de mensajes hacia y desde el bus CAN se detienen y la señal *CANnTX* se mantiene alta. Entrar en el estado de inicialización no cambia la configuración del controlador CAN, los mensajes o los contadores de errores. Sin embargo, algunos registros de configuración solo son accesibles en el estado de inicialización. Para inicializar el controlador CAN, se configura el registro CAN Bit Timing (*CANBIT*) así como también cada identificador de mensaje. En caso de no necesitar un identificador de mensaje, se puede etiquetar como no válido borrando el bit *MSGVAL* en el registro CAN IFn Arbitration 2 (*CANIFnARB2*). De lo contrario, el

identificador del mensaje debe inicializarse, ya que los campos del identificador de mensaje pueden no tener información válida, lo que provocará una excepción en los resultados. Los bits INIT y CCE en el registro CANCTL se deben de establecer para acceder al registro CANBIT y el registro (*CANBRPE*) donde se configura *CAN Baud Rate Prescaler* así como el *bit timing*.

Para salir del estado de inicialización, debe limpiarse el bit INIT, después el bit interno (*BSP* del inglés, *Bit Stream Processor*) se sincroniza con la transferencia de datos en el bus CAN esperando la aparición de una secuencia de 11 bits recesivos consecutivos (lo que indica una condición inactiva del bus), esto sucede antes de que inicie la transferencias de mensajes en el bus.

La inicialización de los identificadores de mensaje no requiere que el CAN esté en el estado de inicialización y puede hacerse sobre la marcha. Sin embargo, todos los identificadores de mensaje deben estar configurados con identificadores particulares o configurados como no válidos antes de que la transferencia de mensajes empiece. Para cambiar la configuración de un identificador de mensaje durante el funcionamiento normal, se borra el bit *MSGVAL* que se encuentra en el registro *CANIFnARB2* para indicar que el identificador del mensaje no es válido durante el cambio. Cuando se completa la configuración, y se configura el bit *MSGVAL* indicar que el identificador del mensaje es otra vez válido.

3.4. Software utilizado

En el desarrollo del proyecto se usó la librería que proporciona el fabricante de la tarjeta de desarrollo “Tiva C Series TM4C123G LaunchPad” de Texas Instruments. La biblioteca usada es de controladores periféricos TivaWare™ de Texas Instruments®, un conjunto de controladores para acceder a los periféricos que se encuentran en la familia Tiva™ de microcontroladores ARM® Cortex™ -M.

La biblioteca de controladores periféricos está dividida en dos modelos de programación: el modelo de acceso directo al registro y el modelo para el controlador de software. Cada modelo

se puede usar de forma independiente o combinada, basado en las necesidades de la aplicación o el entorno de programación deseada. En este caso, se hizo uso de la librería para poder realizar con un menor tiempo el desarrollo del proyecto ya que si se realizara una librería para el manejo de periféricos, en especial CAN, nos consumiría un mayor tiempo en la implementación del proyecto.

3.4.1 Code Composer Studio

Para poder programar a alto nivel y crear y modificar las librerías de TivaWare, se ha seleccionado *Code Composer Studio*. Este programa ofrece una interacción más amigable con el usuario, indicando errores de forma precisa y dando la posibilidad de navegar entre archivos rápidamente.

Code Composer Studio es un entorno de desarrollo integrado (IDE) basado en Eclipse para procesadores integrados de Texas Instruments. CCSv7.0 se basa en Neon (4.6) y CDT 9.0. El software se comunica con la tarjeta desarrolladora por medio de un puerto USB (Figura 3-4) por el cual es descargado el software desarrollado en el programa *Code Composer Studio* hacia el microcontrolador.

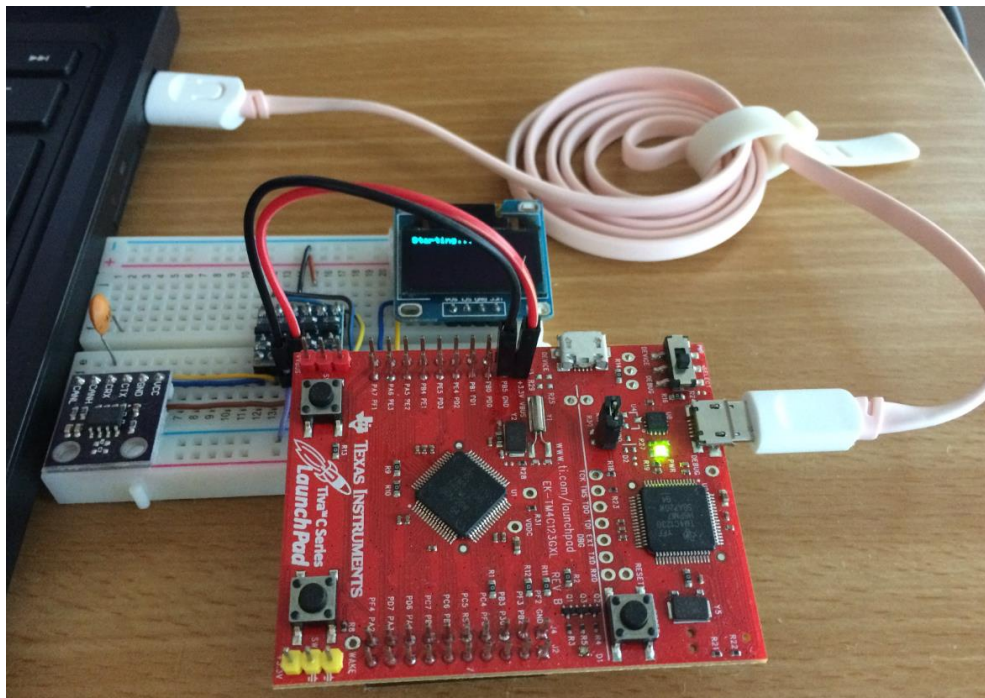


Figura 3-4. Conexión de PC (Software) a tarjeta embebida.

4. Resultados

El objetivo del software desarrollado en el sistema embebido es monitorear una red CAN en la cual se pueda ver la comunicación entre nodos y observar el tráfico en el bus de comunicación. Para realizar el monitoreo se conectó el dispositivo a la red CAN, donde recibe todas las tramas de forma pasiva. Al capturar la información por el dispositivo detector de paquetes se pueden identificar paquetes erróneos o identificar problemas presentados en la red en tiempo real. La instalación del dispositivo es muy fácil ya que no requiere una PC o un software de interface especializado con alto costo de licenciamiento como los que comúnmente se usan en la industria automotriz (Vector CAN).

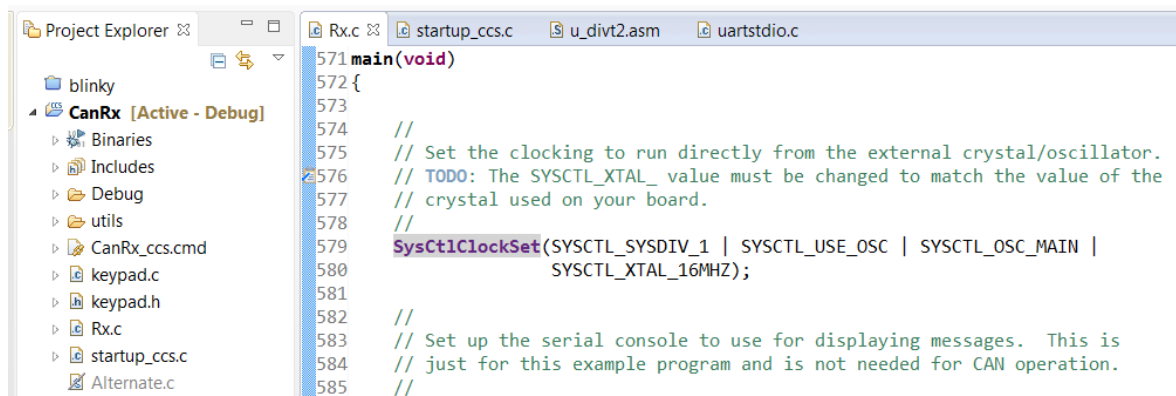
Para realizar la conexión del dispositivo lo único que se requiere es tener acceso al cableado que comunica el equipo de pruebas con la unidad a probar para conectar el CANH y CANL de la red CAN a los canales correspondientes en el dispositivo. Otro punto a considerar para iniciar el monitoreo de la red CAN es la velocidad de transmisión del equipo de pruebas con respecto a la unidad ya que es necesario contar con la misma velocidad de transmisión en el dispositivo, en caso contrario no funcionará.

Un aspecto importante para mejorar el proyecto puede ser la detección automática de la velocidad a la que se comunica los nodos en una red de CAN. Los equipos que cuentan con esta función son de alto costo, los programas más comúnmente utilizados son CANAnalyser y CANoe de Vector. Proveen información del bus de datos. Son productos costosos y complejos los cuales requieren capacitación especializada.

El proyecto en comparación con otros detectores de tramas se enfoca únicamente en redes CAN con el propósito de desarrollar una herramienta práctica y de bajo costo para el uso en la detección de fallas en equipos de prueba en área automotriz.

En la Figura 4-1 podemos observar el entorno de trabajo para desarrollo del software. En la parte izquierda de la imagen, contamos con las librerías que componen el proyecto y del lado derecho la sección de código desarrollado. El software que se usó para el desarrollo es un software gratuito proporcionado por el fabricante de la tarjeta de desarrollo (Texas Instruments). No requiere licencia por lo cual se puede descargar en cualquier equipo.

Otra mejora que pude realizarse en el proyecto para hacerlo independiente al software de desarrollo (*Code Composer Studio*) que sirve de interface para capturamos las tramas de la red, es mostrar las tramas de la red en una pantalla LCD (del inglés, Liquid Crystal Display) que nos permita ver el tráfico en el bus sin necesidad de estar ejecutando el software de desarrollo en la PC (de inglés, *Personal Computer*).



```
571 main(void)
572 {
573
574     //
575     // Set the clocking to run directly from the external crystal/oscillator.
576     // TODO: The SYSCTL_XTAL_ value must be changed to match the value of the
577     // crystal used on your board.
578     //
579     SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
580                   SYSCTL_XTAL_16MHZ);
581
582     //
583     // Set up the serial console to use for displaying messages. This is
584     // just for this example program and is not needed for CAN operation.
585     //
```

Figura 4-1. Plataforma para desarrollo de software.

Como ya se mencionó, la Figura 4-1 muestra el entorno de trabajo. Del lado derecho podemos observar la función *SysCtlClockSet (uint32_t ui32Config)* que nos ayuda a establecer la velocidad de reloj del procesador para dar el tiempo de ejecución del programa con respecto al cristal usado en la tarjeta de desarrollo.

Esta función determina la frecuencia del reloj del procesador, que también es la frecuencia del reloj de los módulos periféricos con la excepción de PWM y CAN que tiene su propio divisor de reloj. El parámetro *ui32Config* es el OR lógico de varios parámetros diferentes, muchos de los

cuales son agrupados en conjuntos donde solo uno puede ser elegido. El primer parámetro es *SYSCTL_SYSDIV_1*. Aquí se define el divisor del reloj del sistema donde en este caso estamos usando 1, el siguiente parámetro *SYSCTL_USE_OSC*, se usa para seleccionar entre un reloj oscilador de cristal (XO) o un PLL (del inglés *phase-locked loop*), continuamos con la fuente del oscilador *SYSCTL_OSC_MAIN* y por último, seleccionar el cristal apropiado con uno de los valores apropiados *SYSCTL_XTAL_xxx* y en este caso se usa *SYSCTL_XTAL_16MHZ*.

Estas son algunas de las funciones usadas en el programa principal (Figura 4-2).

```

582 //
583 // Set up the serial console to use for displaying messages. This is
584 // just for this example program and is not needed for CAN operation.
585 //
586 InitConsole();
587 InitCan0();
588
589 //Init LCD Oled
590 InitI2C();
591 InitSSD1306();
592 clearScreen();
593

```

Figura 4-2. Funciones de inicialización del software.

La función *InitConsole()*; nos ayuda a inicializar la consola para poder visualizar en la PC los datos recibidos por el dispositivo pasivo. Lo que significaría observar el tráfico en el bus de CAN.

La función *InitCan0()*; Inicializa el driver de CAN que se encuentra en la tarjeta para poder iniciar con la comunicación. Dentro de la librería contamos con las siguientes funciones:

- *extern void SysCtlPeripheralEnable (uint32_t ui32Peripheral);* Esta función habilita un periférico. En el encendido, todos los periféricos están desactivados deben ser habilitados para función específica, requerida en el desarrollo. En este caso se habilita el puerto E de la tarjeta.

- *extern void GPIOPinConfigure (uint32_t ui32PinConfig);* Configura la función alternativa de un pin GPIO (del inglés, General Purpose Input/Output). Selecciona la función periférica asociada a un pin GPIO en particular. Cada función periférica solo debe asociarse con un solo pin GPIO a la vez (a pesar de que muchos de ellos pueden estar asociados con más de un pin GPIO).
- *extern void GPIOPinTypeCAN (uint32_t ui32Port, uint8_t ui8Pins);* Configura pin(s) para usar como un dispositivo CAN. Los pines CAN deben estar configurados correctamente para que los periféricos CAN funcionen de forma adecuada.
- *extern void CANInit(uint32_t ui32Base);* Inicializa el controlador CAN después del restablecimiento, el controlador CAN se queda en estado desactivado. Sin embargo, la memoria utilizada para las tramas de mensaje contienen valores indefinidos y deben borrarse antes de habilitar el controlador de CAN la primera vez. Esto evita la transmisión o recepción no deseada de datos antes de que las tramas de mensaje estén configuradas esta función debe invocarse antes de habilitar el controlador la primera vez.
- *extern uint32_t CANBitRateSet (uint32_t ui32Base, uint32_t ui32SourceClock, uint32_t ui32BitRate);* Esta función establece los valores de tiempo de bit CAN a una configuración nominal en relación a la tasa de bits deseada.
- *extern void CANIntRegister (uint32_t ui32Base, void (*pfnHandler)(void));* Esta función registra el manejador de interrupciones en la tabla de vectores de interrupción y habilita el controlador de interrupción de CAN; las fuentes de interrupción de CAN específicas deben habilitarse usando `CANIntEnable ()`.
- *extern void CANIntEnable (uint32_t ui32Base, uint32_t ui32IntFlags);* La función habilita fuentes de interrupción específicas del controlador CAN. Solo las fuentes habilitadas causan una interrupción en el microprocesador.

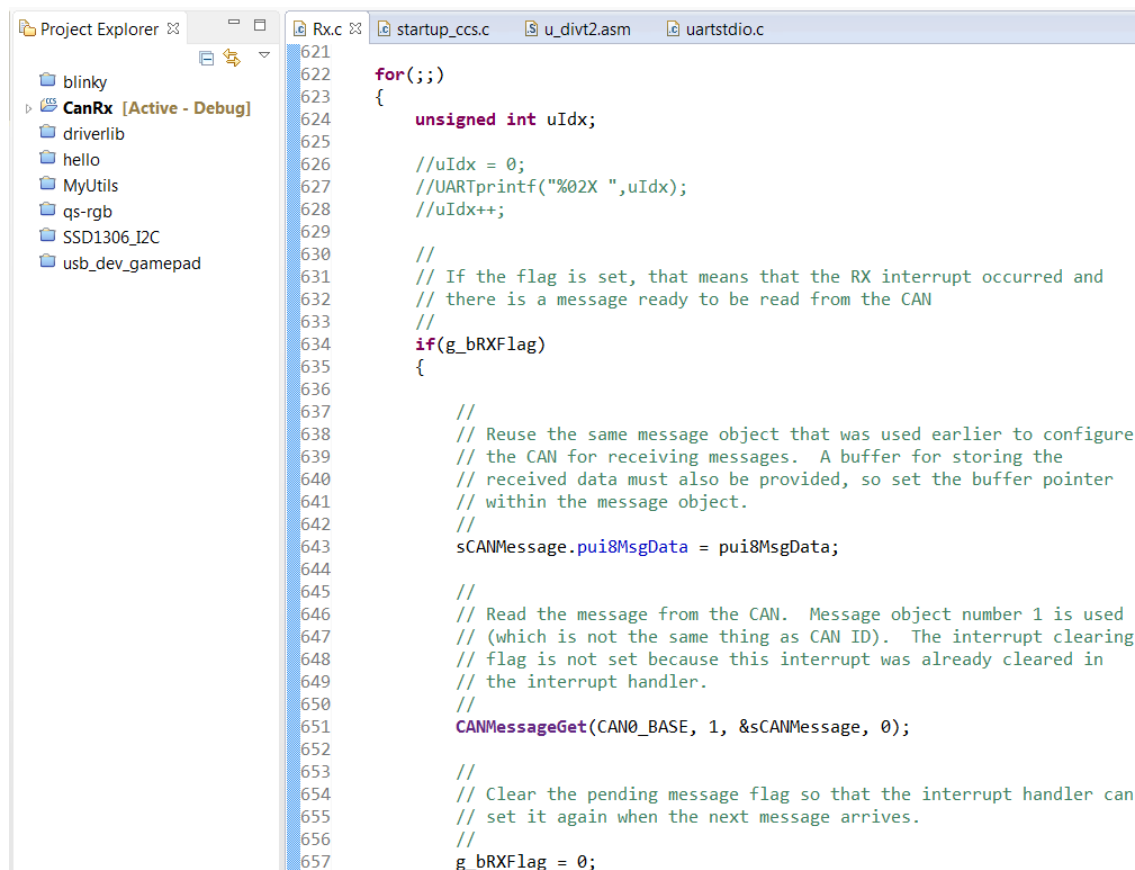
- *extern void IntEnable (uint32_t ui32Interrupt);* La interrupción especificada está habilitada en el controlador de interrupción.
- *extern void CANEnable (uint32_t ui32Base);* Habilita el controlador CAN para el procesamiento de mensajes. Una vez habilitado, el controlador automáticamente transmite cualquier trama pendiente, y procesa cualquier trama recibida. El controlador puede ser detenido llamando a *CANDisable ()*. Antes de llamar a *CANEnable ()*, *CANInit ()* debe haber sido llamado para inicializar el controlador y el reloj del bus CAN debe configurarse llamando a *CANBitTimingSet ()*.
- *extern void CANMessageSet (uint32_t ui32Base, uint32_t ui32ObjID, tCANMsgObject *psMsgObject, tMsgObjType eMsgType);* Esta función se usa para configurar cualquiera de los 32 identificadores de mensaje en el controlador CAN. Un identificador de mensaje se puede configurar para que sea cualquier tipo de mensaje CAN, así como para usar automático en transmisión y recepción. Esta llamada también permite que el identificador de mensaje sea configurado para generar interrupciones al finalizar la recepción o transmisión del mensaje. El identificador mensaje también se puede configurar con un filtro/ máscara para que las acciones solo se realicen cuando aparece un mensaje que cumple con ciertos parámetros que se ven en el bus CAN.

Las siguientes funciones inicializan la pantalla en la cual podemos ver el estatus del bus CAN.

```
InitI2C();
InitSSD1306();
clearScreen();
```

Después de haber realizado las inicializaciones de los drivers que son requeridos continuamos con la parte en donde el dispositivo está en espera de actividad en el bus. Esto es

realizado dentro de un ciclo infinito el cual será interrumpido al registrarse una interrupción lo cual cambia el valor de una variable de estado de 0 a 1 indicando actividad en el bus de CAN (Figura 4-3).



```
621
622     for(;;)
623     {
624         unsigned int uIdx;
625
626         //uIdx = 0;
627         //UARTprintf("%02X ",uIdx);
628         //uIdx++;
629
630         //
631         // If the flag is set, that means that the RX interrupt occurred and
632         // there is a message ready to be read from the CAN
633         //
634         if(g_bRXFlag)
635         {
636
637             //
638             // Reuse the same message object that was used earlier to configure
639             // the CAN for receiving messages. A buffer for storing the
640             // received data must also be provided, so set the buffer pointer
641             // within the message object.
642             //
643             sCANMessage.pui8MsgData = pui8MsgData;
644
645             //
646             // Read the message from the CAN. Message object number 1 is used
647             // (which is not the same thing as CAN ID). The interrupt clearing
648             // flag is not set because this interrupt was already cleared in
649             // the interrupt handler.
650             //
651             CANMessageGet(CAN0_BASE, 1, &sCANMessage, 0);
652
653             //
654             // Clear the pending message flag so that the interrupt handler can
655             // set it again when the next message arrives.
656             //
657             g_bRXFlag = 0;
```

Figura 4-3. Activación de bandera en un ciclo infinito y lectura del bus CAN.

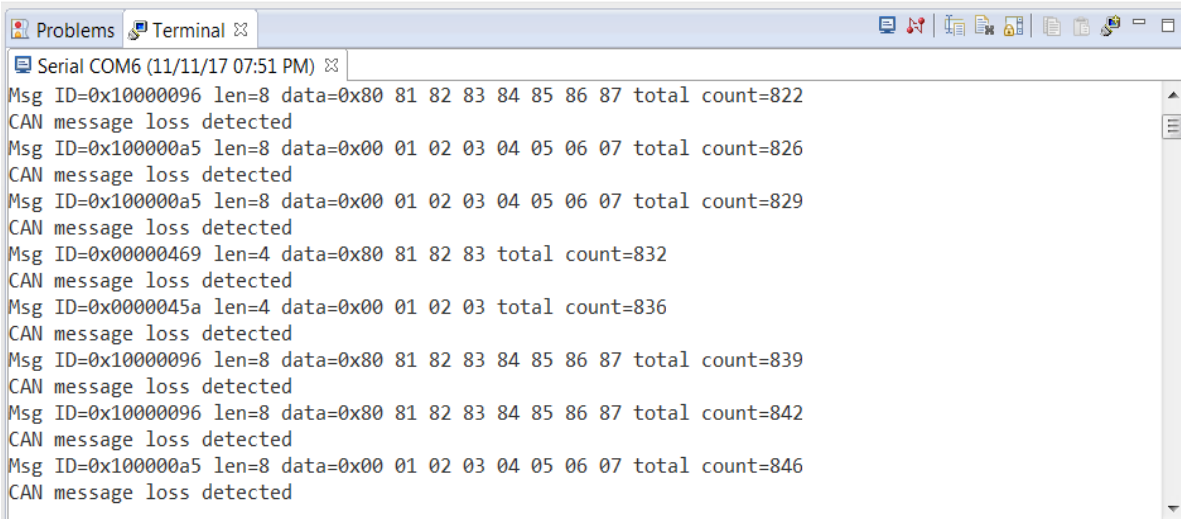
La variable que se usa como bandera para detectar actividad en el bus es, *g_bRXFlag* cuando su estado se encuentre en uno manda llamar a la función *CANMessageGet(CAN0_BASE, 1, &sCANMessage, 0);* que realiza una lectura en el bus CAN.

4.1. Ejecución de la comunicación CAN

El análisis de tráfico de la red CAN podría definirse como: "*la deducción de información a partir del dispositivo que observa el flujo de datos en la red*" [19] . Por lo tanto, el análisis de tráfico de red se puede categorizar por análisis en tiempo real, análisis por tramas y análisis por identificador.

Para obtener los resultados se tuvieron que hacer varias pruebas con diferentes dispositivos. La primera prueba realizada fue con un cluster que se encuentra en el laboratorio automotriz del ITESO (Instituto Tecnológico y de Estudios Superiores de Occidente). La conexión se realizó entre el dispositivo pasivo y la red CAN del clúster que está compuesta por varios nodos de CAN que consta de un tablero de auto, sistema de direccionales, sistema de audio etc.

En este punto, se pudo observar las tramas que se transmitían en la red, por medio de la terminal del software de desarrollo. (Figura 4-4) La conexión realizada para la obtención de tramas se puede observar en la Figura 4-5 donde podemos apreciar la conexión por medio de los pines de CAN de las dos tarjetas y a su vez la tarjeta Texas tiene comunicación con la laptop por medio del puerto serial, donde al ejecutar el software de desarrollo (*Code Composer Studio 7.2.0*) se obtienen las tramas en el bus de CAN que son enviadas por la tarjeta SAM V71.



```
Serial COM6 (11/11/17 07:51 PM)
Msg ID=0x10000096 len=8 data=0x80 81 82 83 84 85 86 87 total count=822
CAN message loss detected
Msg ID=0x100000a5 len=8 data=0x00 01 02 03 04 05 06 07 total count=826
CAN message loss detected
Msg ID=0x100000a5 len=8 data=0x00 01 02 03 04 05 06 07 total count=829
CAN message loss detected
Msg ID=0x00000469 len=4 data=0x80 81 82 83 total count=832
CAN message loss detected
Msg ID=0x0000045a len=4 data=0x00 01 02 03 total count=836
CAN message loss detected
Msg ID=0x10000096 len=8 data=0x80 81 82 83 84 85 86 87 total count=839
CAN message loss detected
Msg ID=0x10000096 len=8 data=0x80 81 82 83 84 85 86 87 total count=842
CAN message loss detected
Msg ID=0x100000a5 len=8 data=0x00 01 02 03 04 05 06 07 total count=846
CAN message loss detected
```

Figura 4-4 Tramas de CAN capturadas por la terminal del software de desarrollo.

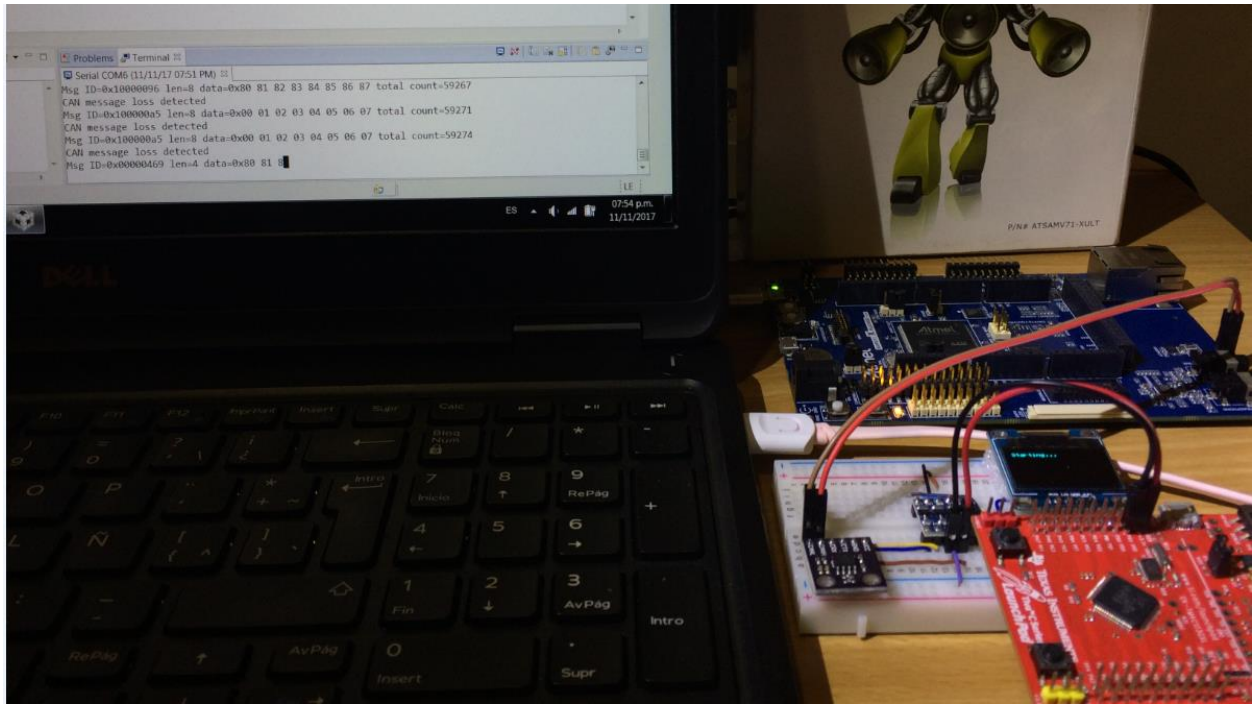


Figura 4-5 Conexión entre la tarjeta SAM v71 y Texas LaunchPad para la obtención de tramas.

Al realizar la prueba anterior se comprobó el funcionamiento del dispositivo en cuanto a la detección de tramas en tiempo real y la identificación de mensajes específicos. Este último, se realizó por medio de activación de registros en la configuración CAN que desde el controlador de CAN (hardware) filtra los identificadores de mensaje y solo se ven en el bus los identificadores deseados.

Otra prueba realizada consistió en análisis de tramas de CAN entre las dos tarjetas anteriormente mencionadas (Figura 4-6).

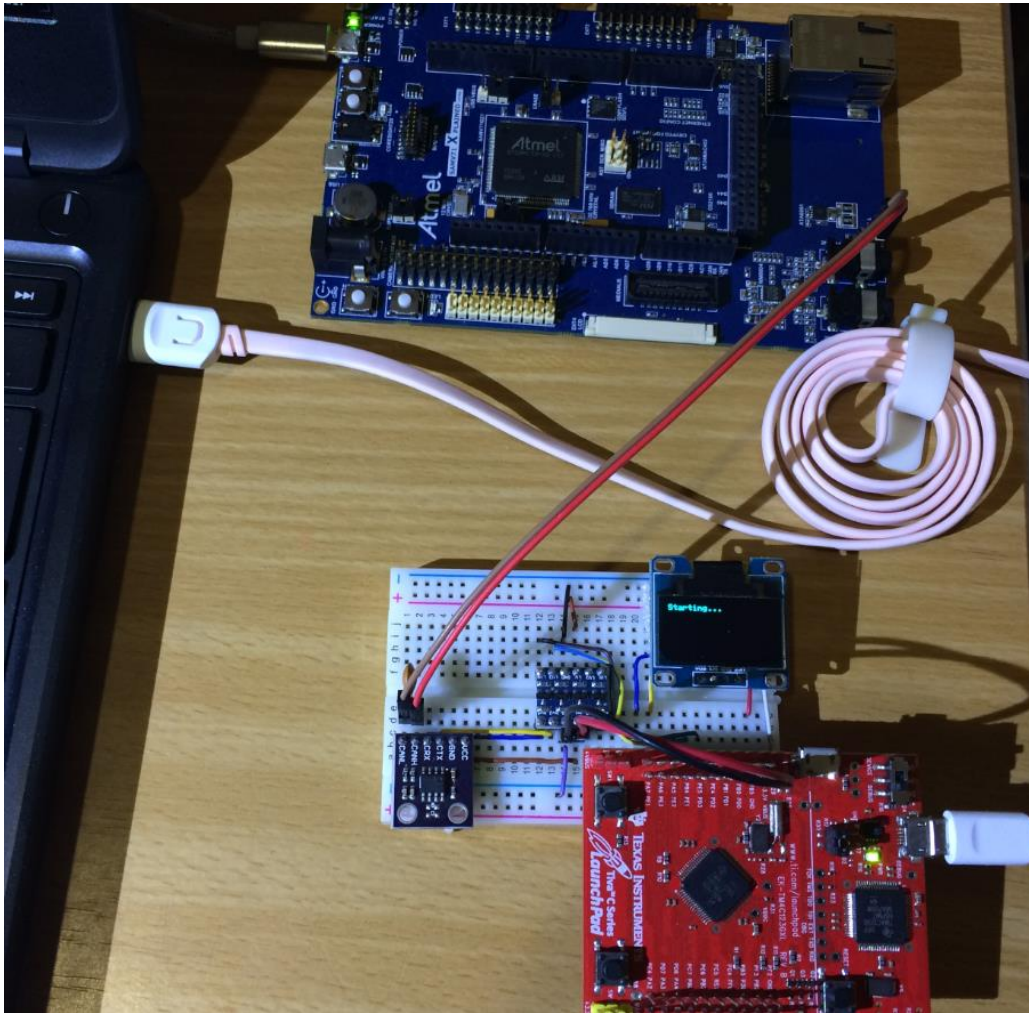


Figura 4-6 Conexión SAM V71 con Tiva C Series TM4C123G LaunchPad.

Las tramas recibidas se observaron por medio de osciloscopio como se muestra en la Figura 4-7. Al analizar las tramas por medio del osciloscopio se demostró la confiabilidad que el equipo puede tener para realizar el monitoreo en el bus CAN. También se pudo apreciar que el dispositivo es capaz de detectar dos formatos de trama (formato estándar y formato extendido).

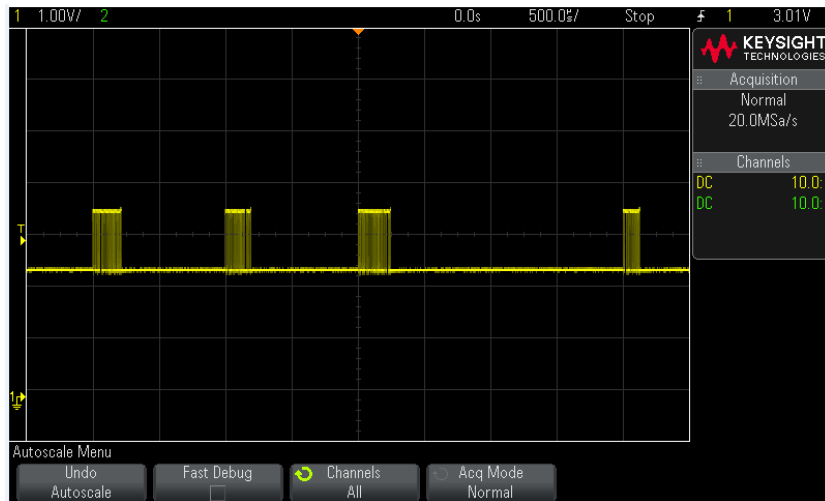


Figura 4-7. Tramas de CAN observadas en el osciloscopio.

Conclusiones

El módulo desarrollado en el presente proyecto (monitoreo bus CAN) nos permite proporcionar una herramienta en el área automotriz, que facilite la verificación en la comunicación en redes CAN. En este caso en particular, se realizaron pruebas en un BUS de 100 Mhz y 500 Mhz exitosamente. Se analizaron tanto el canal “high” como “low” del par diferencial de transmisión. Se demostró como de manera económica se puede hacer el análisis de tramas CAN y se detalla como inicializar la tarjeta de desarrollo Texas para la observación de tramas seriales; además, se muestra que módulos instanciar y sus parámetros a configurar con el fin de capturar señales seriales automotrices. Los dispositivos actuales ofrecidos por vector son robustos, costosos y requieren una capacitación por parte del proveedor, por lo que la principal ventaja de esta propuesta es que puede estar al alcance de presupuestos reducidos.

Se ha mencionado a lo largo de este capítulo, algunos aspectos importantes para desarrollo de trabajos de investigación futuros, los cuales pueden tener un impacto favorable en el desempeño de este módulo de monitoreo para redes CAN, mismos que se enlistan a continuación:

- Mostrar en la pantalla LCD las tramas monitoreadas en el bus CAN además del estado del módulo que ya se muestra.
- Realizar cambios en el software para detectar de manera automática la velocidad de la red.
- Desarrollar el almacenamiento de las tramas monitoreadas en una tarjeta SD.
- Implementar un teclado por medio del cual se pueda seleccionar opciones de un menú que se muestre en el LCD. El menú puede contar con opciones que ayuden al usuario a iniciar, finalizar comunicación, así como guardar o mostrar tramas del bus CAN.

Bibliografía

- [1] “On the communication network inside vehicles.pdf.” .
- [2] “Real-Time Systems in Automotive.pdf.” .
- [3] “CAN, a Ten Years’ Anniversarial Review.pdf.” .
- [4] U. Kiencke, K.J. Neumann, “U. Kiencke, K.J. Neumann: ‘OSEK/VDX - An Open Software Architecture for Communicating Vehicle Systems’, 3rd international CAN Conference, Paris, France, 1996.”
- [5] © 2003 ISO — All rights reserved, “ISO 7637-3:1995, Road vehicles — Electrical disturbance by conduction and coupling — Part 3: Vehicles with nominal 12 V or 24 V supply voltage — Electrical transient transmission by capacitive and inductive coupling via lines other than supply lines.”
- [6] HUBERT ZIMMERMANN, “OS1 Reference Model-The ISO Model of Architecture for Open Systems Interconnection.pdf.” .
- [7] H. Zimmermann, ““High level protocols standardization: Technical and political issue,” in *“High level protocols standardization: Technical and political issues,”* .
- [8] IFIP-WG 6. I and ISO/TC97/SC16/N23,56 pp.. Feb. 1978., ““Proposal for a standard virtual terminal protocol.”” .
- [9] “OSI Reference Model An Overview.pdf.” .
- [10] Philips Semiconductors, *PCA82C250/251 CAN Transceiver. Application Note. AN 96116. 1996.* .
- [11] Lawrenz W., “CAN System Engineering: From theoretical to Practical Applications, Springer Verlag.” .
- [12] Robert Bosch GmbH, “CAN Specification Version 2.0.” Sep-1991.
- [13] ISO, “ISO 11898: Road Vehicles Interchange of digital information Controller Area Network (CAN) for high-speed communication ISO 1993.” 1993.
- [14] ISO, “ISO 11519-2: Road Vehicles Low-speed serial data communication Part 2: Low-Speed Controller Area Network (CAN), ISO 1994.” 1994.
- [15] Estchberger, K., *Controller Area Network, Basics, Prococols, Chips and Applications, IXXAT Automation GmbH. 2001.*
- [16] “CAN in Automation, CiA Draft Standar 102 Version 2.0 CAN Physical Layer for Industrial applications, CiA April, 1994.” .
- [17] “Tiva™ C Series TM4C123G LaunchPad Evaluation Board.pdf.” .
- [18] “Tiva™ TM4C123GH6PM Microcontroller Data Sheet.pdf.” .
- [19] “Network Traffic Analysis Using Packet Sniffer.pdf.” .