UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

TESIS

DISEÑO E IMPLEMENTACION DE UNA RED CAN BÁSICA

Qué para obtener el Título de
INGENIERO ELECTRÓNICO

Presenta
Francisco Javier Bedolla Arreola

Asesor de Tesis
Ingeniero Electricista Félix Jiménez Pérez

Morelia Michoacán, Enero de 2014

Agradecimientos

- *A mis incansables padres que siempre estuvieron conmigo en las buenas y en las malas por darme la mejor herencia que puede haber en el mundo ser un profesionista.*
- *Al Ing. Félix Jiménez Pérez por su paciencia, por las cosas que he aprendido de él ya que sin su ayuda no hubiera sido posible este trabajo.*
- *A todos los maestros que sin sus enseñanzas no hubiera sido capaz realizar este trabajo.*
- *A mis compañeros por los momentos que vivimos en la universidad*.

Dedicatoria

- *A mi madre Paula Arreola por todas esas noches de desvelo que llevo junto a mí, así como los sustos que tuvo que soportar producto de experimentos fallidos. Gracias a su apoyo hoy soy la persona que ella soñó.*

Resumen

En este trabajo se presenta el diseño de una red CAN básica en modo FIFO mejorado utilizando el módulo ECAN del microcontrolador PIC18F4580 y para adaptar las señales a los valores de voltaje especificados por el protocolo CAN se utiliza un transceptor MCP2551. La red está diseñada para comunicarse a una velocidad de 500 kHz y constituida por dos nodos a los cuales se pueden conectar en un futuro un máximo de 40 nodos más sin la necesidad de modificar el código del programa de los nodos originales. En los primeros capítulos se presentan los conceptos básicos para comprender la comunicación vía protocolo CAN, como se realiza el acceso múltiple por detección de portadora, el método de arbitrariedad, descripción de los dos tipos de mensajes utilizados para la comunicación CAN, es decir, mensajes estándar y extendidos, así como una explicación del estándar SAE J1939, estándar con el cual ha sido diseñada la red. La red se encuentra diseñada mediante el sistema multitarea para hacer más sencillo las modificaciones del código en dado caso de querer agregar nuevas funciones a la red. Para comprobar que el sistema multitarea se encuentra funcionando de manera correcta se realiza simulación de las tareas programadas en los microcontroladores mediante el software MpLab IDE de Microchip®. Para comprobar que la configuración del protocolo se encuentra funcionando adecuadamente se realiza una depuración línea a línea del programa principal del sistema. Finalmente se realizan pruebas funcionales de la red y se realiza una tabulación de los resultados obtenidos.

Palabras Clave:
PIC18F4580.
MCP2551
FIFO
ECAN.
SAE J1939.

Abstract

In this paper the design of a basic CAN network is presented using the enhanced ECAN FIFO module of PIC18F4580 microcontroller, the signals are adapted to the voltage values specified by the protocol with a MCP2551 CAN transceiver. The network is designed to communicate nodes at 500 kHz and consists of two nodes which could be connected in the future up to 40 nodes without modification on the program code of the original nodes. The basic concepts are presented in the first chapters to understand the communication via CAN protocol, such as multiple access is through carrier sensing, the method of arbitrariness ,description of the two types of messages used for CAN communication , it is mean , standard and extended messages, as well an explanation of SAE J1939 standard which the network is designed. The network is designed under multitasking system to make easier code modifications in case you want to add new features to the network. To verify that the multitasking system is running correctly we simulate each task programmed with MPLAB IDE software. To check out the configuration protocol is working properly it is necessary carry out a debugging line by line of the main program's code of the system. Finally network a functional tests are performed and a tabulation of the results is performed.

Keyword:
PIC18F4580.
MCP2551
FIFO
ECAN.
SAE J1939.

Contenido

Agradecimientos i
Dedicatoria ii
Resumen iii
Abstract iv
Contenido v
Lista de Figuras viii
Lista de Tablas xi
Lista de Snippet xi
Lista de Símbolos y Abreviaturas xii
Capítulo 1 1
Descripción del protocolo CAN-bus 1
1.1. Introducción 1
1.2. Antecedentes. 2
1.3. Objetivo. 4
1.4. Justificación. 4
1.5. Metodología 5
1.6. Contenido de la tesis. 5
Capítulo 2 6
Fundamentos básicos del CAN-bus 6
2.1. Introducción 6
2.2. Protocolo de comunicaciones CAN-bus. 8
2.3. Acceso múltiple por detección de portadora, con detección de colisión (CSMA/CD) 13
2.3.1. Método de arbitrariedad. 15
2.4. Comunicación basada en mensajes. 16
2.4.1. Descripción de los Mensajes “frames” (bloques de información) CAN. 18
2.4.2. Manejo de Errores en el CAN bus. 25
2.5. Estándar SAE J1939. 35
2.5.1. Formato y uso de los Mensajes (J1939-21) 35
2.5.2. Nombres y direcciones (J1939-81) 38
2.5.3. Transmisión de Mensajes (J1939-21 y J1939-7x) 40
2.5.4. Recepción de Mensajes (J1939-21 y J1939-7x) 41

2.5.5. Interpretación de un Mensaje J1939. 41
Capítulo 3 44
Microcontrolador y Controlador CAN. 44
3.1. Introducción. 44
3.2. Microcontrolador PIC18F4580. 44
3.2.1. Módulo CAN en el microcontrolador PIC18F4580. 45
3.2.2. Modos de funcionamiento del módulo CAN en el microcontrolador 46
3.2.3. Modos Funcionales del Módulo CAN. 50
3.2.4. Transmisión de Mensajes CAN 52
3.2.5. Recepción de Mensajes 54
3.2.6. Filtros y Máscaras de aceptación de Mensajes 56
3.2.7. Configuración del Baud Rate 57
3.2.8. Sincronización 60
3.2.9. Programación del tiempo de los segmentos. 62
3.3. CAN Transceiver. 63
3.3.2. Transceptor (Transceiver) CAN MCP2551. 64
3.3.3. Funcionamiento como transmisor. 64
3.3.4. Funcionamiento como receptor. 65
3.3.5. Protección Interna. 65
3.3.6. Modos de operación. 65
3.4. CAN Explorer (Explorador CAN). 66
Capítulo 4 69
Diseño de una red Can 69
4.1. Introducción 69
4.2. Diseño de la red. 69
4.2.1. El Circuito Integrado LM35 70
4.2.2. Transductor de presión MPX5010 71
4.2.3. Diseño de un nodo CAN mediante un microcontrolador PIC18F4580 72
4.2.4. Acondicionamiento de la señal de los Sensores. 74
4.3. Diseño del Software. 79
4.3.1. Tiempo Real “MultiTasking” 79
4.3.2. Manejo de Tiempo del Procesador. 80
4.3.3. Inicio del Programa “Estructura Main”. 87
4.3.4. Control del Programa “Ciclo while(1)”. 90

Capítulo 5 96
Pruebas y Resultados 96
5.1. Validación de los periodos (Mplab SIM). 96
5.2. Pruebas en depuración (PICkit 2). 99
5.3. Pruebas en tiempo de ejecución. 107
5.3.1. Pruebas al Nodo 1. 107
5.3.2. Pruebas al Nodo 2. 109
Capítulo 6 112
Conclusiones 112
6.1. Conclusiones. 112
6.2. Trabajos futuros. 114
Apéndice A 115
A.1. Función “J1939_Initialization” de la biblioteca J1939 115
A.2. Función “InitMicro”. 118
A.3. Función “init_AD”. 119
A.4. Función “Timer_Ini”. 120
A.5. Función “LCD_Init”. 121
A.6. Función “Estructura main”. 122
Bibliogrfía 125

Lista de Figuras

2.1. Modelo de referencia OSI…….. 10
2.2. Señal codificada con el método NRZ…… 11
2.3. Elementos básicos de un nodo…….. 12
2.4. Acceso múltiple por detección de portadora……. 14
2.5. Arbitrariedad del bus CAN…… 15
2.6. Comunicación del bus CAN…….. 16
2.7. Petición remota…….. 17
2.8. Frame (Mensaje) de datos…….. 20
2.9. Campo de arbitrariedad para mensajes estándar y extendidos…….. 21
2.10. Campo de control y tabla de verdad DLC…… 22
2.11. Campo de datos…… 23
2.12. Campo ACK…….. 23
2.13. Campo EOF……... 24
2.14. Composición del mensaje remoto…….. 24
2.15. Mensaje estándar (Dato/remoto), con ID idénticos……... 25
2.16. Globalización de Errores……... 27
2.17. Mensaje de Error Activo…… 28
2.18. *Bit sttufing*……. 28
2.19. Campos del mensaje con *bit-sttufing*……. 29
2.20. Área o campos del mensaje en los que se realiza *Bit-Monitoring*……. 29
2.21. Campos para el cálculo del Error CRC…… 30
2.22. Generación del polinomio CRC…… 31
2.23. Campos en los que se genera un error de formato……. 31
2.24. Estados de error de un nodo CAN……. 32
2.25. a) Contador TEC, b) Contador REC…….. 33
2.26. Mensaje de sobreflujo…… 34
2.27. Identificador de 29 bits…….. 36
2.28 Formato de mensaje (J1939-21)…… 36
3.1. Microcontrolador PIC18F4580…….. 45

3.2.	Buffers de transmisión……..	53
3.3	Segmentos del *Time Quanta*……..	57
3.4.	Alargando un periodo de bit (Agregando un SJW al segmento de fase 1)……	61
3.5.	Acotando un periodo de bit (Quitando un SJW del segmento de fase 2)……..	61
3.6.	Imagen de un CAN Transceiver conectado a un módulo CAN……	64
3.7.	CAN Transceiver MCP2551…….	64
3.8.	CAN Explorer……	67
4.1.	Diagrama de bloques de la red CAN…….	70
4.2.	Sensor LM35…….	71
4.3.	Transductor de presión MPX5010…….	71
4.4.	Sensor de presión con filtro RC pasa-bajos para filtrado de ruido……	72
4.5.	Conexiones mínimas entre el microcontrolador PIC18F4580 y el transceiver MCP2551……..	73
4.6.	Nodo CAN terminales DB9 para conexión al bus...…….	74
4.7.	Amplificador No-Inversor……..……..	75
4.8.	Amplificador No-Inversor con ganancia de 4.97……..	76
4.9.	Acondicionamiento de la señal del sensor LM35……..	77
4.10.	Circuito de acondicionamiento para la señal del sensor MPX5010..……...	78
4.11.	Conexión entre el MCU y un LCD 16x2……..	79
4.12	Diagrama de tiempo del mecanismo multitasking de 4 tareas……..	81
5.1.	Herramienta MPLAB SIM para simulación de las interrupciones……	96
5.2.	Colocación del break point en la función “high_isr”……	97
5.3.	Configuración del Simulador…….	97
5.4.	Stopwatch……..	98
5.5.	Barra de herramientas……	98
5.6.	Lectura de tiempo realizada por el Stopwatch……..	99
5.7.	Aplicación PICkit 2 para depuración…..……...	100
5.8	Pantalla de detección del PICkit 2……	100
5.9.	Registros y variable agregado al *watch*…….	101
5.10.	Compilación del proyecto……..	101
5.11.	Grabado del programa en el microcontrolador para depuración.....	102

5.12.	Breakpoint necesario para entrar a la función J1939_Initialization…..……....	102
5.13.	Interrupción del programa en la función "J1939_Initialization"…….....	103
5.14.	Ventana emergente con el código de la función J1939_Initialization…….	103
5.15.	Cambio en el valor del registro CANSTAT y la variable J1939_Address……	103
5.16.	Sub-función SetECANMode...……..	104
5.17.	Registro CANCON = 0x80h…….	104
5.18.	ECANCON = 0x90h y BSEL0 = 0xE0h……...	105
5.19.	BRGCON1 = 0x01, BRGCON2 = 0xBF y BRGCON3 = 0x87……	105
5.20	Módulo CAN en Modo Normal y ECAN adaptado para el estándar J1939…..	105
5.21.	Configuración de los pines del puerto B para operación CAN…….	106
5.22.	Reclamo de la dirección asignada al nodo…….	106
5.23	Espera para sincronización de nodos…….	106
5.24.	Selección del programador……	107
5.25.	Medición y despliegue de datos medidos por el nodo 1……	108
5.26.	Medición y despliegue de datos medidos por el nodo 2……	110

Lista de Tablas

2.1. Evolución del protocolo CAN……... 8
2.2. Campos del formato frame estándar…….. 19
2.3. Campos del formato frame extendido…… 20
2.4. Ejemplo PGN……. 38
2.5. Estructura del nombre…… 38
2.6. Número de byte ocupado por los campos del nombre……... 39
2.7. Identificador CAN (ID-CAN)…… 42
2.8. Bytes de Datos……... 42
3.1. Tabla de verdad Filtros/Mascaras…….. 56
4.1. Registros Configurados por la función J1939_Initialization.…… 88
5.1. Resultados obtenidos. Datos enviados por el nodo 2 y recibidos en el nodo 1…. 109
5.2. Resultados obtenidos. Datos enviados por el nodo 1 y recibidos en el nodo 2…. 111

Lista de Snippet

4.1. Configuración del vector de interrupción de alta prioridad……. 84
4.2. Orden a seguir para habilitación de interrupciones……. 85
4.3. Función “high_isr” para modo multitarea…… 86
4.4. Inicialización y configuración de la red CAN……. 90
4.5. “Task 0” lectura del estado del interruptor…….. 91
4.6. “Task 1” Construcción y transmisión del mensaje…..... 92
4.7. “Task 2” Recepción, decodificación y despliegue del mensaje……... 93
4.8. “Task 3” Lectura del convertidor analógico-digital……. 94

Lista de Símbolos y Abreviaturas

CAN	Red de área de control
ECAN	CAN mejorado
ECU	Unidad de control electrónico
SAE	Sociedad de Ingenieros Automotrices
CiA	Automatización del protocolo CAN
CAL	Capa de aplicación del protocolo CAN
DLL	Capa de enlace de datos
SDS	Sistema distribuido inteligente
DCS	Sistema de control distribuido
PID	Control proporcional, integral derivativo
OSEK	Sistema abierto y sus interfaces para la electrónica automotriz
ID	Identificador
ISO	Organización Internacional para la Estandarización
OSI	Interconexión de sistemas abiertos
NZR	No regreso a cero
CSMA/CD	Acceso múltiple con detección de portadora con detección de colisiones
IDE	Identificador de extensión
GND	Tierra
V+	Voltaje positivo
RTR	Petición de transmisión remota
SOF	Comienzo de mensaje
DLC	Longitud del código de datos
SRR	Petición sustituta de transmisión remota
CRC	Verificación de redundancia cíclica
ACK	Campo de reconocimiento
EOF	Fin de mensaje
REC	Contador de error de recepción
TEC	Contador de error de transmisión
PGN	Número de parámetro de grupo

DP	Datos de página
PDU	Unidad de protocolo de datos
PF	PDU format
PS	PDU specific
TP_BAM	Mensaje de anuncio de mensajes broadcast
TP_CM	Gestión de conexión
DT	Transferencia de datos
EOM	Reconocimiento de fin de mensaje
FIFO	Primero en entrar, primero en salir
Rx	Receptor
Tx	Transmisor
MCU	Microcontrolador
MAB	Buffer de ensamble de mensaje
TQ	Time Quanta
IPT	Tiempo de procesado de información
SJW	Modificación al segmento de sincronización
LCD	Display de cristal liquido
Hz	Hertz
SR	Slew rate
M	Mega
K	Kilo
M	Mili
N	nano
S	Segundos
µ	Micro
F	Frecuencia
F	faradios
T	Tiempo
°C	Grados celsius
Ω	Ohms
R	Resistencia

Comentario [i1]: Escribe su definicion

Vss	Voltaje negativo
Vcc	Voltaje positivo
IC	Circuito integrado
JP	Jumper
V	Volts
A/D	Analógico-digital
Cd	Corriente directa
Δ	Ganancia
CA	Controlador de la aplicación
Vref -	Voltaje de referencia negativo
Vref+.	Voltaje de referencia positivo
OSC	Oscilación
H	Hexadecimal
D	decimal
Pa	Pascales

Capítulo 1

Descripción del protocolo CAN-bus

1.1. Introducción.

Debido al creciente desarrollo de los procesos de control en la industria, los sistemas de control distribuido han ido evolucionando constantemente y a su vez la necesidad de disponer de dispositivos cada vez más "inteligentes" para realizar el control o supervisión remota, en procesos de fabricación, almacenamiento o distribución de información. Debido a que en la mayoría de las ocasiones estos se encuentran sometidos a entornos industriales, es necesario un protocolo de comunicaciones que simplifique y economice la comunicación entre varios subsistemas de diferentes fabricantes, esto sobre una red común o en un mismo bus.

De ahí la necesidad de implementar un red para sistemas distribuidos que permita comunicación robusta de datos, sistemas multímaestros, flexible en su configuración, detección de errores, retransmisión automática y recepción de datos de múltiples fuentes o subsistemas. El protocolo de comunicaciones CAN-bus permite obtener estas características.

La red de área de control (CAN, *Controller Area Network*) es un protocolo de comunicaciones seriales para sistemas distribuidos con control en tiempo real muy eficiente y con un alto nivel de integridad de la información. Inicialmente creado en Alemania por la empresa BOSCH, para aplicaciones automotrices como un método que permitiría la comunicación robusta entre las centrales electrónicas basadas en microprocesadores.

La implementación de una red CAN cumple las necesidades de comunicación de un amplio rango de aplicaciones con altas velocidades de transmisión de datos y a un bajo costo de cableado. Por ejemplo, en la electrónica automotriz, las unidades de control electrónica (ECUs), sensores y sistemas antideslizantes pueden ser conectados al mismo tiempo usando una red CAN, con velocidades hasta de 1 Mbit/s, además de que esta red es

fiable para conectar la electrónica de la carrocería del automóvil, como faros y/o ventanas eléctricas reduciendo la amplia circuitería que implica este tipo de conexiones.

Además de lograr la compatibilidad entre diferentes implementaciones de redes CAN. Y la posibilidad de implementar un nodo de diagnóstico dentro de la misma red, el cual permita la detección de errores en la transmisión de datos, la correcta recepción de mensajes de los nodos o subsistemas conectados a la red, así como la detección de fallas de los subsistemas distribuidos a lo largo del bus.

1.2. Antecedentes.

En febrero de 1986, Robert Bosch introdujo el sistema serial CAN-bus al congreso SAE (Sociedad de Ingenieros Automotrices) en Detroit. Diseñado para soportar mensajes pequeños (hasta 8 bytes), acceso multímaestro (las colisiones son resueltas por prioridad), y con un alto grado de fiabilidad (15 bits CRC por cada mensaje). A mediados de 1987, Intel fue uno de los pioneros en el desarrollo de circuitos integrados que soportaban el protocolo CAN, el integrado 82526. Posteriormente, Phillips Semiconductors introdujo el controlador CAN 82C200. Actualmente, cerca de 20 fabricantes de circuitos integrados producen dispositivos con interfaces CAN, siendo estos instalados en la mayoría de los automóviles fabricados actualmente en Europa y equipados con al menos una red CAN. Además de ser usado en otro tipo de transportes, desde trenes hasta navíos, así como en control industrial, esto debido a que el protocolo CAN es uno de los más dominantes en lo que se refiere al control distribuido. Solo en 1999, cerca de 60 millones de controladores CAN fueron fabricados para aplicaciones específicas, y más de 100 millones de dispositivos CAN fueron vendidos en el año 2000.

Originalmente el protocolo CAN bus fue desarrollado para ser usado en automóviles compactos, pero la primera aplicación de este bus se realizó en otras áreas del mercado. Especialmente en el norte de Europa, el protocolo CAN-bus fue muy popular en sus primeras apariciones en la industria textil y en control de máquinas. A principios de los 90´s, fueron tales las circunstancias que se logró encontrar un grupo de usuarios que hicieron posible la estandarización de las diferentes soluciones que ofrecía el protocolo. A inicios de 1992, Holger Zeltwanger, reunió a los usuarios y fabricantes para establecer el "*CAN in Automation*" (CiA), asociación internacional de usuarios y fabricantes dedicados al desarrollo del protocolo CAN. Una de las primeras tareas del CiA fue la especificación de una capa (*Layer)* para la aplicación del protocolo (CAL, *CAN Application Layer*). Debido a que el protocolo es una implementación exclusiva de la capa de enlace de datos

(*Data Link Layer "DLL")*, no había estándares para el intercambio de datos entre diferentes aplicaciones. Aunque el enfoque de CAL fue académicamente correcto y utilizable para aplicaciones industriales, tuvo un inconveniente: cada usuario tenía que diseñar un nuevo perfil de comunicaciones.

A partir de 1993, dentro del proyecto Esprit ASPIC, un consorcio Europeo encabezado por la compañía BOSCH desarrolló un prototipo que posteriormente se convertiría en el protocolo llamado CANopen, un perfil basado en CAL para redes internas en las áreas de producción. En 1995, CiA lanzo el perfil de comunicaciones CANopen.

A principios de los 90's se desarrollaba en paralelo al CANopen un perfil "*Higher-layer*" de comunicaciones: los ingenieros de la compañía Cincinnati Milacron de ingenieros mecánicos de Estados Unidos comenzaron una empresa en conjunto con Allen Bradley y Honeywell Microswitch dedicada a un proyecto de control y comunicaciones basadas en el protocolo CAN. Sin embargo, después de un corto tiempo, miembros importantes de la empresa cambiaron de prioridades y la empresa se vino abajo. Como resultado, Allen Bradly y Honeywell Microswitch se separaron y continuaron con el proyecto de forma separada. Lo que condujo a dos protocolos *higher-layer* 'DeviceNet' y 'Smart Distributed System' (SDS), bastantes similares entre sí, por lo menos en las capas inferiores de comunicación.

A comienzos de 1994, Allen Bradley cambio la especificación DeviceNet a 'Open DeviceNet Vendor Association' (ODVA), el cual impulso la popularidad del protocolo DeviceNet. Honeywell fracaso al hacer algo similar con SDS, por lo que hizo que SDS fuera casi una solución exclusiva para Honeywell Microswitch. DeviceNet fue desarrollado específicamente para la automatización en industrias, y posteriormente convirtiéndose en una competencia directa de protocolos como Profibus-DP e Interbus. Ofreciendo funcionalidades fuera de la plataforma plug-and-play, DeviceNet se convirtió en el líder de los sistemas basados en un bus, particularmente en el área de automatización industrial.

Con DeviceNet y CANopen, dos capas de aplicación se encontraban disponibles para ser usadas en el protocolo CAN, ambas con diferentes áreas de aplicación. DeviceNet es optimizado para la automatización en la industria y CANopen enfocado a sistemas embebidos con cualquier tipo de control. Lo que ha hecho obsoleto las capas de aplicación específica; la necesidad de capas de aplicación específica es historia (excepto, y casi para algunos sistemas embebidos específicos).

Por otro lado los fabricantes de circuitos integrados quienes habían implementado módulos CAN dentro de sus circuitos se enfocaron a la industria automotriz. Desde mediados de los 90's Infineon Technologies (Siemens) y Motorola han vendido grandes cantidades de controladores CAN a fabricantes de vehículos Europeos. Como consecuencia, a finales de los 90's Far Eastern, fabricante de circuitos integrados comenzó a producir controladores CAN. A partir de 1992, Mercedes Benz comenzó a utilizar CAN en sus automóviles de lujo. Posteriormente Volvo, Saab, Volkswagen y BMW emplearon CAN, por último Renault y Fiat integraron redes CAN en sus vehículos.

1.3. Objetivo.

Debido a la importancia que ha obtenido el protocolo CAN bus en aplicaciones de control distribuido dentro de la industria, es importante conocer y desarrollar aplicaciones con este tipo de bus. En este trabajo se implementa una red CAN, mediante microcontroladores PIC18F4580 de MICROCHIP, esta red se encuentra enfocada a sistemas de control distribuido, que son utilizados en la mayoría de la industria del sector automotriz, sector en el que nos enfocaremos. Sin embargo este protocolo puede utilizarse en cualquier sistema de control distribuido.

1.4. Justificación.

Debido al incremento en el número de dispositivos electrónicos en los automóviles y en dispositivos con varios subsistemas (robots, máquinas, procesos industriales, bordadoras etc.) y la complejidad del cableado entre ellos. Se pensó en la posibilidad de conectar todos estos dispositivos a un único bus, el cual tendría que ser fiable, robusto, con alta inmunidad al ruido, y además que permitiría transmisiones a altas velocidades en entornos expuestos a perturbaciones como son altas temperaturas, vibraciones e interferencias electromagnéticas, como por ejemplo en los automóviles. Además de ser capaz de auto-diagnosticarse y diagnosticar a los dispositivos conectados a la red así como verificar el correcto funcionamiento de los mismos y detectar rápidamente el dispositivo de la red que se encuentra fallando. Permitiendo una distribución de los sistemas eléctricos y electrónicos, en pequeños circuitos llamados nodos facilitando el mantenimiento de estos. Asimismo es de interés del autor aprender el funcionamiento y diseño de una red de comunicación CAN utilizado para control distribuido, particularmente en automóviles.

1.5. Metodología.

Investigación y revisión bibliográfica del protocolo CAN-bus desde un punto teórico, posteriormente la implementación de una red CAN básica y realización de pruebas a la red para corroborar lo aprendido en la parte teórica.

1.6. Contenido de la tesis.

En el capítulo 1 se presenta una breve introducción a este trabajo, la situación actual de los sistemas de control distribuido y los beneficios del protocolo CAN-bus. También se menciona una breve historia sobre el protocolo CAN y los principales impulsores del mismo.

En el capítulo 2 se presentan conceptos básicos del protocolo CAN, así como los elementos que forman una red CAN, el método de acceso al medio utilizado por el protocolo. La comunicación basada en mensajes que utiliza y el manejo de errores que emplea para asegurar la estabilidad de la red. También una breve explicación del microcontrolador a utilizar, el uso del *CAN controller* y del *Transceiver* en la red CAN.

En el capítulo 3 se presenta el hardware requerido (microcontrolador, controlador CAN y transceptor CAN) para el diseño de una red CAN básica.

En el capítulo 4 se presenta el diseño de la Red CAN, diseño del hardware y software de la red.

En el capítulo 5 se mencionan las pruebas realizadas a la red (protocolo CAN), así como resultados obtenidos de las pruebas.

En el capítulo 6 se presentan las conclusiones generales, aportaciones y trabajos futuros.

Capítulo 2

Fundamentos básicos del CAN-bus

2.1. Introducción.

Los sistemas de control distribuido (DCS, *Distributed Control System*), son aplicados por lo general, a un sistema de fabricación, proceso o cualquier tipo de sistema dinámico, en el que los elementos de tratamiento no son centrales en la localización (como el cerebro), sino que se distribuyen a lo largo de todo el sistema con cada componente o sub-sistema controlado por uno o más procesadores. En donde todo el sistema de controladores está conectado mediante redes de comunicación y de monitoreo.

Debido a que el desarrollo del control distribuido en la industria va en paralelo al de comunicaciones, cada vez es más necesario disponer de dispositivos inteligentes para realizar el control o la supervisión remota, tanto de procesos de fabricación, como de almacenamiento o distribución. Ya que los sistemas o redes de comunicación empleados en entornos industriales se encuentran sometidos a problemas muy específicos que condicionan su diseño y funcionamiento, a diferencia de las redes de datos o redes de oficina.

Los DCS, son sistemas usados para el control de procesos industriales continuos u orientados por lotes, como en refinadoras de petróleo, industrias petroquímicas, estaciones generales de generación de energía, industrias farmacéuticas, industria automotriz, etc. Los DCS son conectados a sensores y actuadores, y usados por ejemplo como puntos controlar el flujo de material a través de la planta en cuestión.

Un DCS típico, consiste en controladores digitales (típicamente and/or) distribuidos estratégicamente, capaces de ejecutar de 1 a 256 o más lazos de control en un mismo panel de control. Los dispositivos de entrada/salida (I/O) pueden estar integrados al controlador o conectados remotamente por medio de una red de área. Hoy en día los controladores tienen amplias capacidades de procesamientos, además de poder realizar diferentes tipos de control como son: control proporcional, proporcional integral y proporcional integral

derivativo (P, PI y PID) en un mismo circuito integrado, esta clase de circuitos integrados generalmente realizan control del tipo lógico y secuencial. Los DCS más modernos son capaces de soportar redes neuronales y aplicaciones de control difuso (*Fuzzy Control*).

Los Sistemas de Control Distribuido pueden usar uno o más centros de control y ser configurados en cualquiera centro de control o vía remota mediante una computadora. La comunicación local entre DCS es llevada a cabo por una red de control a través de cables de par trenzados, cables coaxiales o fibra óptica.

En 1983 la compañía Alemana, líder en fabricación de autopartes Robert Bosch tomo la decisión de desarrollar un protocolo de comunicación orientado a sistemas distribuidos, operados en tiempo real y partiendo de las exigencias que toda compañía requiere. Esto dio comienzo para el desarrollo del protocolo CAN. Debido a que una empresa de autopartes no podría lograr nada con un concepto de este tipo, Robert Bosch tomó la decisión de formar una asociación con universidades (particularmente con la Escuela Técnica Superior de Braunschweig), donde el profesor Wolfhard Lawrenz sugirió el nombre de Red de Área de Control (CAN) y con los principales fabricantes de circuitos integrados de la época, harían este sueño una realidad.

En 1991 el primer vehículo producido con el protocolo CAN salió a la venta, con cinco unidades de control electrónico y un bus CAN operando a 500 kbits/s. Durante el mismo periodo, las necesidades 'internas' (aplicaciones en autos) y necesidades 'externas' (aplicaciones industriales) salieron a relucir por lo que fue necesario buscar apoyo en la SAE (Sociedad de Ingenieros Automotrices) y la OSEK (Sistemas abiertos y sus interfaces para la electrónica automotriz) para la industria automotriz y por el protocolo CAN en automatización (CiA, CAN *in Automotation*). En la tabla 2.1 se muestra la evolución del protocolo CAN durante sus primeros 20 años.

AÑO	Desarrollo
1983	Desarrollo del protocolo CAN en R. Bosch GmbH.
1985	Versión 1.0 de las especificaciones CAN.
	Primeras relaciones establecidas entre Bosch y los fabricantes de circuitos.
1986	Trabajo de estandarización en ISO.
1987	Primer prototipo del circuito integrado CAN.
1989	Primeras aplicaciones industriales.
1991	Especificaciones del protocolo extendido CAN 2.0: • Parte 2.0A: IDs (Identificadores) de 11 bits. • Parte 2.0B: IDs de 29 bits.
	El primer vehículo (Mercedes clase S) equipado con cinco unidades de comunicación a 500 kbits/s.
1992	Creación del grupo de usuarios CiA (CAN en automatización).
1993	Creación del grupo OSEK.
	Aparece la primera capa de aplicación (CAL) de CiA.
1994	El primer estándar en ISO, llamado *high* y PSA (Peugeot Citroen) *low speed*, es completado, y Renault se une a OSEK.
1995	Grupo de trabajo en Estados Unidos con SAE.
1996	CAN es aplicado en la mayoría de los sistemas de control de motores de los vehículos de lujo Europeos.
	Un mayor número de participantes se unen a OSEK.
1997	Los principales productores de chips ofrecen familias de componentes con CAN.
	La CiA es constituida por 300 miembros empresariales.
1998	Aparecen nuevos estándares relacionados con CAN.
1999	Etapa de desarrollo de redes time-triggered CAN (TTCAN).
2000	Expansión del equipo *CAN-linked* en todos los vehículos y aplicaciones industriales.
2001	Introducción industrial del TTCAN en tiempo real.
2003	Japoneses y americanos comienzan a implementar bus CAN.
2008	Se producen aproximadamente de 65 a 67 millones de vehículos con 10-15 nodos CAN por vehículo.

Tabla 2.1 Evolución del protocolo CAN.

2.2. Protocolo de comunicaciones CAN-bus.

CAN (acrónimo del inglés, *Controller Area Network*), es un protocolo de comunicaciones desarrollado por la firma alemana Robert Bosch GmbH, basado en la topología bus para la transmisión de mensajes en entornos distribuidos. Además ofrece una solución a la gestión de la comunicación entre múltiples unidades de control electrónico (ECUs). Y es uno de los cinco protocolos utilizados en el estándar OBD-II para diagnóstico de automóviles.

El protocolo CAN es un protocolo normalizado de comunicaciones, con lo que se simplifica y economiza la tarea de comunicar subsistemas de diferentes fabricantes sobre una red común o bus. Su procesador anfitrión (host) delega la carga de comunicaciones a través de un periférico inteligente, por lo tanto el procesador anfitrión dispone de mayor tiempo para ejecutar sus propias tareas.

CAN es un protocolo basado en mensajes, el cual es un concepto de comunicaciones de datos, que describe una relación entre un productor y uno o más consumidores. Es un protocolo diseñado específicamente para aplicaciones automotrices, en donde la información que se va a intercambiar se descompone en mensajes, a los cuales se les designa un identificador (ID) y se encapsulan en tramas para su transmisión. Las principales características del protocolo CAN bus son:

- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Flexibilidad en la configuración.
- Recepción de múltiples fuentes con sincronización de tiempos.
- Sistema robusto en cuanto a consistencia de datos.
- Sistema multímaestro.
- Detección y señalización de errores.
- Retransmisión automática de tramas erróneas.
- Distinción entre errores temporales y fallas permanentes de los nodos de la red, y desconexión autónoma de nodos defectuosos.

El protocolo CAN fue desarrollado, inicialmente para aplicaciones en automóviles y por lo tanto la plataforma del protocolo es resultado de las necesidades existentes en el área automotriz. La Organización Internacional para la Estandarización (ISO, por sus siglas en inglés) define dos tipos de redes CAN: una red de alta velocidad (hasta 1Mbps); bajo el estándar ISO 11898-2, destinada para controlar el motor e interconectar las unidades de control electrónico (ECU); y una red de baja velocidad tolerante a fallas (menor o igual a 125 Kbps), bajo el estándar ISO 11519-2, dedicada a la comunicación de los dispositivos electrónicos internos de un automóvil como son control de puertas, quemacocos, luces, asientos, entre otros.

De acuerdo al modelo de referencia OSI (*Open System Interconnection*, Modelo de interconexión de sistemas abiertos), la arquitectura de protocolos CAN incluye tres capas: física, de enlace de datos y de aplicación, además de una capa especial para la gestión y control del nodo llamada capa de supervisión. En la figura 2.1 se muestra como se encuentra constituido el modelo OSI.

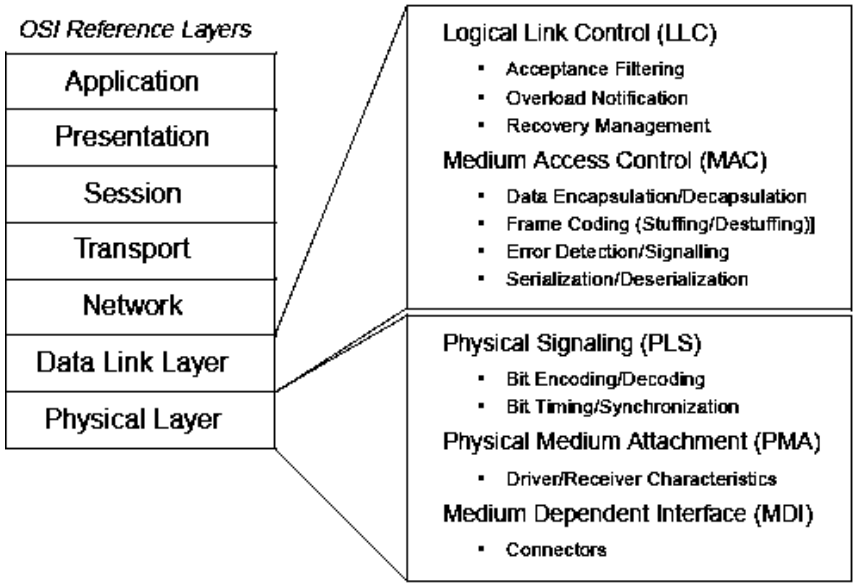


Figura 2.1 Modelo de referencia OSI.

El protocolo CAN es implementado en las dos capas más bajas del modelo de referencia OSI. El medio físico de comunicación del modelo fue dejado fuera a propósito de las especificaciones del protocolo CAN para permitir al diseñador del sistema que adaptará y optimizará el protocolo de comunicaciones en múltiples medios de comunicación de mayor flexibilidad.

- **Capa física (*Physical Layer*):** define los aspectos del medio físico para la transmisión de datos entre nodos de una red CAN, los más importantes son niveles de señal, representación, sincronización y tiempos en los que los bits se transfieren al bus (tiempo de bit).

- **Capa de enlace de datos (*Data Link Layer*):** define las tareas independientes del método de acceso al medio, además de que una red CAN brinda soporte para procesamiento en tiempo real a todos los sistemas que lo integran, el intercambio de mensajes que demanda dicho procesamiento requiere de un sistema de transmisión a altas frecuencias y retrasos mínimos. En redes multímaestro, la técnica de acceso al medio es muy importante ya que todo nodo activo tiene los derechos para controlar la red y acaparar los recursos. Por lo tanto la capa de enlace de datos define el método de acceso al medio así como los tipos de tramas para el envío de mensajes.

CAN es un protocolo de comunicaciones serie multímaestro que soporta control distribuido en tiempo real con un alto nivel de seguridad y multiplexación para conectar ECUs. El establecimiento de una red CAN para interconectar los dispositivos electrónicos internos de un vehículo tiene la finalidad de sustituir o eliminar gran cantidad de cableado. Las ECUs, sensores, sistemas deslizantes, etc. se conectan mediante una red CAN a velocidades de transferencia de datos de hasta 1 Mbps.

La transmisión de mensajes a través del CAN bus funciona de un modo parecido a una conferencia telefónica, en donde, un nodo conectado a la red puede enviar mensajes y el resto de los nodos conectados a la red reciben dicho mensaje, si los datos contenidos en el mensaje resultan interesantes para cierto nodo, este los utiliza para alguna tarea en específico. Un mensaje está constituido primeramente de un ID, que representa la prioridad del mensaje, y hasta ocho bytes de datos. Estos serán transmitidos en forma serial dentro del bus. La señal generada por los nodos es codificada con el método *no regreso a cero* (NZR, *non-return-to-zero*), es decir, que para una señal generada el nivel de voltaje de esta siempre permanecerá constate entre bits de un mismo valor, sin importar el estado del bit, es decir, si el bit es un cero lógico o si el bit es un uno lógico. La figura 2.2 muestra un ejemplo de una señal codificada con el método NRZ.

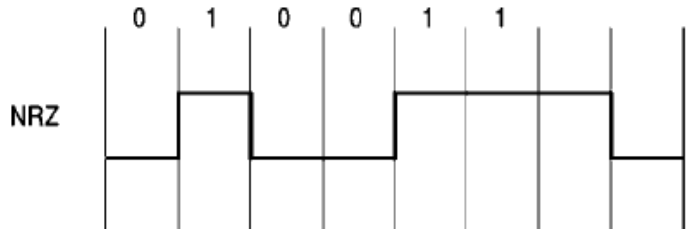


Figura 2.2 Señal codificada con el método NRZ.

Cuando un nodo necesita enviar información a través de una red CAN, puede ocurrir que varios nodos intenten transmitir simultáneamente. CAN resuelve lo anterior al asignar prioridades mediante el ID de mensaje, donde dicha asignación se realiza durante el diseño del sistema en forma de números binarios y no puede modificarse dinámicamente. El ID con el menor número binario es el que tiene mayor prioridad. En la figura 2.3 se muestran los elementos básicos de un nodo CAN, el cual está constituido por:

- **Procesador Principal** (*Host*)**:** decide que mensajes recibir y que mensajes desea enviar. Al procesador principal pueden ser conectados sensores, actuadores y dispositivos de control.

- **Controlador CAN** (CAN *controller*, hardware con un reloj síncrono)
  - Recepción: el controlado CAN almacena los bits recibidos del bus hasta que un mensaje entrante esté disponible, y que posteriormente será buscado por el procesador principal (por lo general después de que el controlador CAN haya provocado una interrupción).
  - Envío: el procesador principal almacena los mensajes a transmitir al controlador CAN, quien transmitirá los bits en forma serie por el bus.

- **Transceiver (Transceptor)**
  - Receptor: adapta los niveles de señal del bus a los niveles que el controlador CAN espera y cuenta con circuitos de protección que protegen el controlador CAN.
  - Transmisor: convierte la señal del bit a transmitir recibido del controlador CAN, en una señal que se envía en el bus.

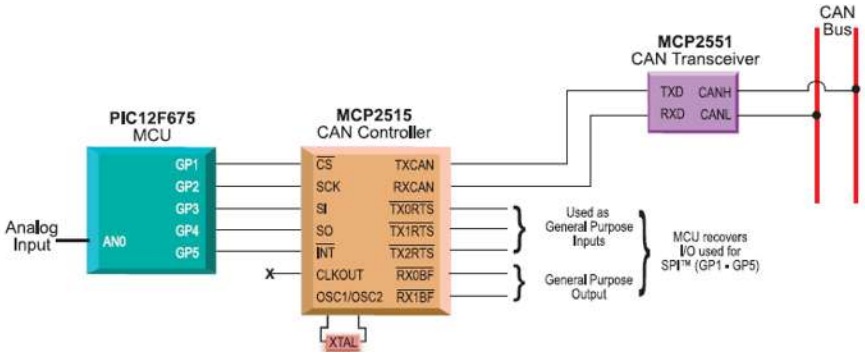


Figura 2.3 Elementos básicos de un nodo.

El método de acceso al medio utilizado es el de Acceso Múltiple por detección de portadora, con Detección de Colisiones (CSMA/CD, *Carrier Sense Multiple Access with Collision Detection*).

Una red CAN puede ser configurada para trabajar con dos diferentes formatos de mensajes (*frames*): el formato estándar (CAN 2.0A) o el formato extendido (CAN 2.0B). La única diferencia entre formatos es que el formato estándar cuenta con un ID de 11 bits de longitud y el formato extendido con uno de 29 bits de longitud, de los cuales 11 bits son de ID base y 18 bits de extensión de ID. La distinción entre el formato base y el extendido la realiza mediante un bit llamado IDE (Identificador de Extensión), que es transmitido como un bit dominante en el caso de un frame estándar, y transmitido como un bit recesivo en el caso de un frame extendido.

2.3. Acceso múltiple por detección de portadora, con detección de colisión (CSMA/CD).

El protocolo de comunicaciones CAN es un protocolo CSMA/CD. Acceso múltiple por detección de portadora, quiere decir, que cualquier nodo en la red debe monitorear continuamente el bus para que en un periodo de inactividad tratar de enviar un mensaje por el bus (Detección de portadora). Una vez que el periodo de inactividad haya ocurrido, cualquier nodo conectado al bus tiene la misma oportunidad de transmitir un mensaje (Acceso múltiple). La figura 2.4 muestra cómo se lleva acabo el protocolo CSMA.

De acuerdo con este método, los nodos en la red que necesitan transmitir información deben esperar a que el bus esté libre; cuando se cumple esta condición, dichos nodos transmiten un bit de inicio. Al iniciar una transmisión cada nodo lee el bus, bit a bit durante la transmisión del mensaje y comparará el valor transmitido con el valor recibido; mientras los valores sean idénticos, el nodo continuará transmitiendo; si se detecta una diferencia de los bits, se lleva a cabo un mecanismo que consiste en asignar el medio de comunicación (señalización del bus) a alguno de los nodos que tratan de transmitir información.

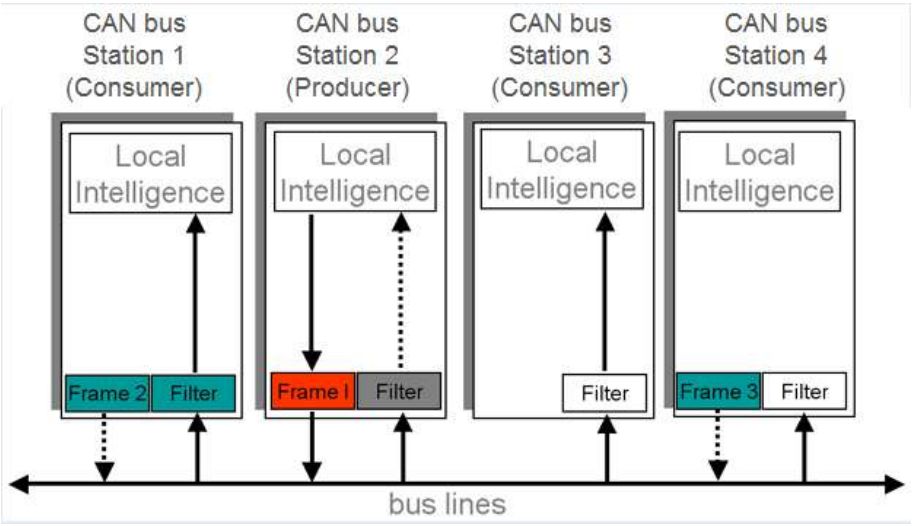


Figura 2.4 Acceso múltiple por detección de portadora.

CD detección de colisión. Quiere decir que si dos nodos comienzan a transmitir al mismo tiempo, dichos nodos detectaran una '*colisión'*, para la cual el CAN bus llevara a cabo un método de resolución de conflictos.

En el protocolo CAN se utiliza un método de resolución de controversia bit a bit no destructivo. Lo cual significa que los mensajes permanecen intactos después de que dicha controversia se ha resuelto incluso si durante esta se detectaron colisiones en el bus. Esta resolución se lleva a cabo sin alteración o retrasó del mensaje de alta prioridad.

Hay un par de conceptos que se requieren saber para el método de resolución de controversias bit a bit no destructivo. Primero, definir los estados lógicos como dominantes o recesivos, en donde un estado lógico dominante es aquel que se encuentra a un nivel lógico 0 (GND) y un estado lógico recesivo se encuentra a un nivel lógico 1 (V+). Segundo, el nodo transmisor debe estar continuamente monitoreando el bus para ver si el estado lógico que está tratando de enviar actualmente se encuentra en el bus.

Un bit dominante siempre ganará la controversia del bus sobre un bit recesivo, en consecuencia el mensaje con ID (identificador usado en el proceso de arbitrariedad de mensajes) más pequeño, será el mensaje de alta prioridad.

2.3.1. Método de arbitrariedad.

Si dos o más nodos comienzan a transmitir al mismo tiempo después de haber encontrado el bus en estado *idle* (inactivo), la colisión de mensajes es evitada gracias al método de acceso CSMA/CD implementado en el bus. Una vez que los nodos han encontrado el bus en un estado *idle*, cada nodo conectado al bus envía los bits de su ID de mensaje y monitorea los niveles del bus. Mientras los bits enviados de los nodos conectados al bus sean iguales, no pasará nada en el bus.

En la figura 2.5 se muestra un ejemplo del envío de los IDs de mensajes de 3 nodos tratando de ganar la controversia existente en el bus. Se observa que en el bit 5 los nodos 1 y 3 envían un identificador dominante o bit dominante. El nodo 2 manda un bit recesivo pero lee un bit dominante del bus, por lo tanto, el nodo 2 pierde la arbitrariedad del bus y cambia su estado a un modo en el cual solo escucha y solo transmite bits recesivos. Posteriormente en el bit 2 el nodo 1 pierde la arbitrariedad ante el nodo 3. Lo que significa que el ID de mensaje del nodo 3 tiene el valor binario más pequeño que el ID de los mensajes de los nodos 1 y 2, por lo tanto, es el de mayor prioridad de los 3 nodos. De esta forma el nodo con el mensaje de mayor prioridad gana la controversia del bus sin perder tiempo, es decir, no tendrá que repetir el mensaje. Una vez que el nodo 3 haya terminado su transmisión los nodos 1 y 2 enviaran nuevamente sus IDs de mensaje tratando de ganar una vez más la controversia en el bus.

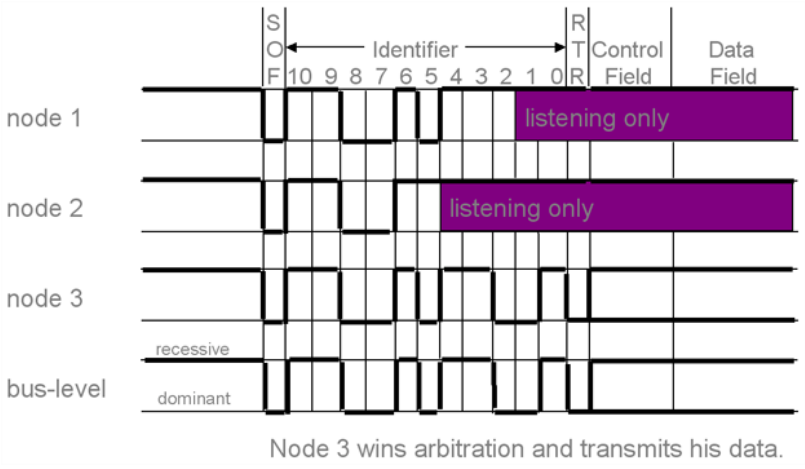


Figura 2.5 Arbitrariedad del bus CAN.

2.4. Comunicación basada en mensajes.

El protocolo CAN es un protocolo basado en mensajes, no un protocolo basado en direcciones. Esto significa que los mensajes no son transmitidos de un nodo a otro nodo basándose en direcciones, el principio utilizado es uno llamado “Mensaje Broadcast”, idea derivada de un transmisor que difunde una señal de radio y cualquier receptor puede recibir, además asegura que a todas las unidades de control conectadas al bus recibirán la misma información.

Dentro de los mensajes se encuentra integrada la prioridad de los mismos, además de contener los datos a ser transmitidos. Todos los nodos en el sistema reciben los mensajes transmitidos en el bus (si el mensaje es recibido con éxito los nodos responderán con un *acknowledge* (reconocimiento)). Después de recibir los mensajes es tarea de cada nodo decidir si el mensaje (información) recibido tendría que ser descartado o aceptado para ser procesado, este proceso es mostrado en la figura 2.6. Por lo que es necesario implementar un filtro de aceptación en cada nodo conectado al bus CAN.

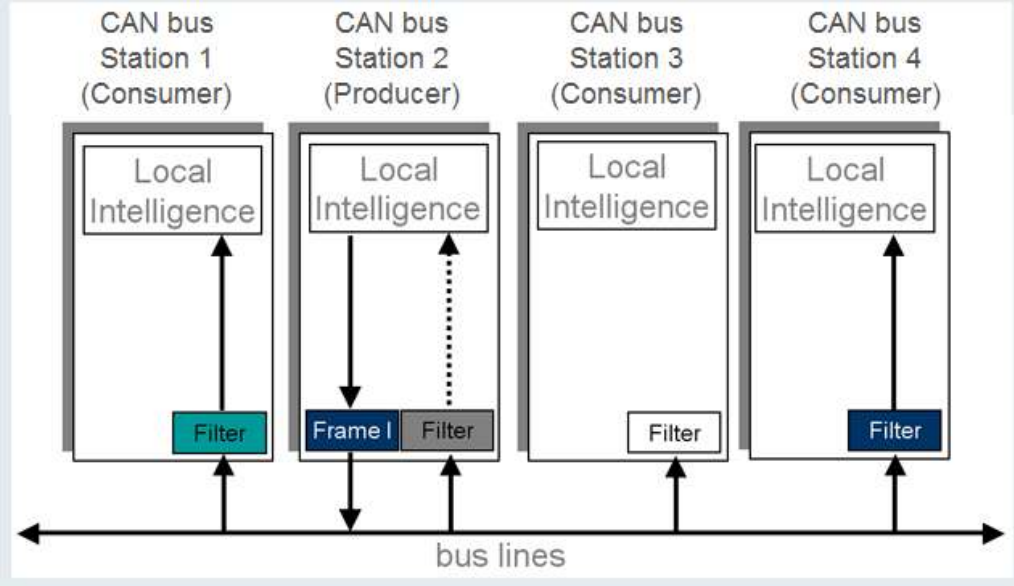


Figura 2.6 Comunicación del bus CAN.

Un mensaje puede ser destinado a un nodo en particular, o a varios nodos, esto dependiendo de la manera en que la red y el sistema hayan sido diseñados y/o configurados.

Por ejemplo, un sensor de la bolsa de aire de un automóvil puede ser conectado vía CAN a un router con un sistema seguro de un solo nodo. Este nodo puede tomar información de otro router y enviar esta información a otros nodos conectados a un router diferente. Entonces, todos los nodos conectados a la red reciben al mismo tiempo el último mensaje enviado por el sensor de la bolsa de aire, si el mensaje fue recibido adecuadamente, envían un acknowledge, y deciden si utilizan la información recibida o rechazan dicha información.

Otra característica importante del protocolo CAN es la habilidad que un nodo tiene para pedir información de otros nodos. Esta característica es llamada Petición de transmisión a distancia (RTR (*Remote Transmision Request,* por sus siglas en inglés)). Esta característica es muy diferente a la descrita en el párrafo anterior, ya que un nodo en vez de esperar información enviada de un nodo en particular, este nodo hace una petición específica para que le sea enviada la información que él solicite. La figura 2.7 ilustra un ejemplo de una petición remota.

El bit RTR es como un bit de pregunta, donde el nodo que tiene la respuesta produce en una segunda comunicación el dato o información solicitada. Este mensaje o *frame* de datos también puede ser recibido por otros nodos, quienes también están interesados en tal información. Los *frames* y datos remotos son identificados por un campo específico, dentro del ID de mensaje.

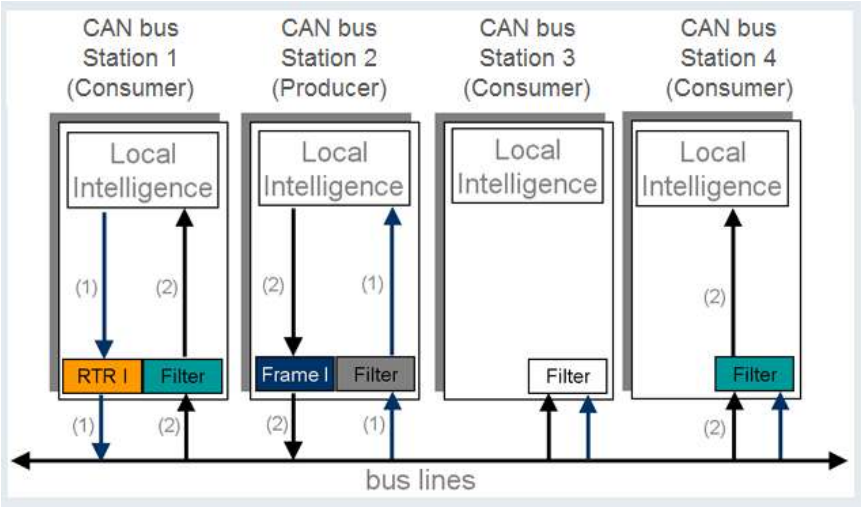


Figura 2.7 Petición remota.

Por ejemplo, un sistema basado en mensajes de un automóvil frecuentemente obtiene actualizaciones de sensores específicos como son los sensores de las bolsas de aire, pero no con frecuencia recibe actualizaciones de otros sensores como lo son: los sensores de presión de aceite o los de batería baja; para asegurarse de que estos estén funcionando correctamente. Periódicamente, el sistema puede solicitar datos de estos sensores y así realizar un chequeo completo del sistema. Esta característica puede ser utilizada para minimizar el tráfico en el la red CAN y mantener la integridad de la misma.

2.4.1. Descripción de los Mensajes "frames" (bloques de información) CAN.

El protocolo CAN define cuatro diferentes tipos de mensaje "frames". El primero y más común es el frame de datos (*Data frame*). Este tipo de frame es usado cuando un nodo transmite información a otro nodo o a todos los nodos conectados en una red CAN. El mensaje remoto (*Remote frames*), es el segundo tipo de mensaje, básicamente es un data frame con un bit RTR que significa que es una petición de transmisión remota. Los otros dos tipos de frames son para manejo de errores. Uno llamado frame de error (*Error frame*) y el segundo llamado error de sobreflujo (*Overload error*). Los frames de error son generados por nodos que detectan cualquier error definido en el protocolo CAN. Y los errores de sobreflujo son generados por nodos que requieren más tiempo para procesar un mensaje que ya ha sido recibido.

Frame de datos (Data frame).

Un frame de datos es producido por un nodo conectado al bus CAN cuando desea transmitir un dato o si le han solicitado una transmisión remota de otro nodo. Dentro de un frame puede ser enviado un dato de hasta 8 bytes. Existen dos diferentes formatos de mensajes:

- **Formato Estándar de mensajes:** con identificador de 11 bits.
- **Formato Extendido de mensajes:** con identificador de 29 bits.

El frame de datos está constituido por varios campos que proveen información adicional acerca del mensaje. En la tabla 2.2 y 2.3 se muestran los campos que constituyen tanto al formato estándar como al formato extendido de mensajes.

La implementación en el modo pasivo de la especificación CAN 2.0B del protocolo CAN no es posible almacenar o transmitir frames de datos extendidos; sin embargo en el modo activo de la especificación CAN 2.0B es posible almacenar y transmitir tanto frames de datos en formato estándar como en formato extendido.

Nombre del campo	**Longitud (bits)**	**Descripción**
***Start-of-frame* (SOF)**	1	Indica el inicio de la transmisión.
Identificador (ID)	11	Identificador del dispositivo y de la información a enviar, el cual a su vez representa la prioridad del mensaje.
***Remote Transmission Request* (RTR)**	1	El estado del bit debe ser dominante (0).
***Identifier Extensión bit* (IDE)**	1	Debe ser un bit dominante (0) (Opcional).
***Reserved bit* (r0)**	1	Bit reservado (puede ser dominante/recesivo, por lo general un bit dominante).
Data length code* (DLC)	4	Numero de bytes del dato (0-8 bytes).
Data field	0-64 (0-8 bytes)	Dato a ser transmitido (longitud en bytes dedicados al campo DLC).
***Cyclic Redundancy Check* (CRC)**	15	Checa la redundancia cíclica.
CRC delimiter	1	Deber ser un bit recesivo (1).
***Acknowledge slot* (ACK)**	1	El transmisor envía un bit recesivo y cualquier receptor puede responder con un bit dominante.
ACK delimiter	1	Debe ser un bit recesivo.
***End-of-frame* (EOF)**	7	Bits recesivos.

Tabla 2.2 Campos del formato frame estándar.

En la figura 2.8 se muestra un frame de datos. Un frame de datos es transmitido después de un periodo de espera del bus. El frame inicia con el bit SOF (*start of frame)* el cual es enviado como un bit dominante, este bit es el encargado de sincronizar a los nodos conectados en el bus. El bit SOF puede ser puesto en el bus durante un estado de reposo, o también durante una interrupción de una transmisión y al final del espacio llamado interframe. A este bit le sigue el campo de arbitrariedad (*Arbitration Field*), usado para dar prioridad a los mensajes.

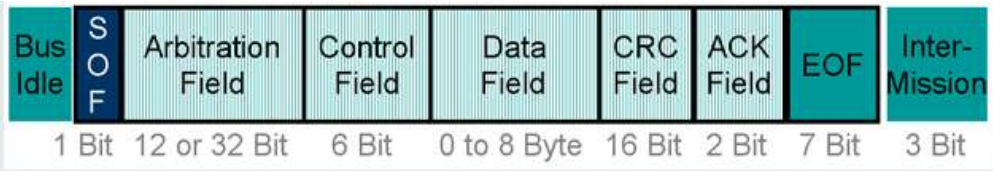


Figura 2.8 Frame (Mensaje) de datos.

Nombre	Longitud (bits)	Descripción
***Start-of-frame* (SOF)**	1	Indica el inicio de la transmisión de datos.
Indentifier A	11	Primera parte del identificador del dispositivo y la información a enviar, el cual a su vez representa la prioridad del mensaje.
***Substitute Remote Request* (SRR)**	1	Debe ser un bit recesivo (opcional).
***Identifier Extension bit* (IDE)**	1	Bit recesivo (opcional).
Indentifier B	18	Segunda parte del identificador del dispositivo e información.
***Remote transmission Request* (RTR)**	1	Debe ser un bit dominante.
***Reserved bits* (r0,r1)**	2	Bits reservados (dominantes/recesivos, generalmente bits dominantes).
Data length code* (DLC)	4	Numero de bytes del dato (0-8 bytes).
Data field	0-8 bytes	Dato a ser transmitido (longitud dedicada al DLC).
***Cyclic Redundancy Check* (CRC)**	15	Debe ser un bit recesivo.
CRC *delimiter*	1	Bit recesivo.
***Acknowledge slot* (ACK)**	1	El transmisor envía un bit recesivo y el receptor (cualquier) responde con un bit dominante.
ACK delimiter	1	Bit recesivo.
***End-of-frame* (EOF)**	7	Bits recesivos.

Tabla 2.3 Campos del formato frame extendido.

*Es posible transmitir un valor DLC entre 9 y 15, ya que este campo está compuesto por 4 bits, a pesar de esto el dato (información) a enviar se encuentra limitado a 8 bytes. Algunos controladores CAN permiten transmisión y/o recepción del campo DLC mayor a 8, pero la longitud de información seguirá limitada a 8 bytes.

En el campo de resolución de controversia (*Aribitration Field*) pueden existir mensajes en formato estándar o en formato extendido. Como se mencionó anteriormente la longitud del identificador (ID) de los mensajes en formato estándar es de 11 bits, los cuales constituyen la base del identificador (ID base) en el formato extendido. El ID es seguido por el bit RTR, que en mensajes de datos es enviado como bit dominante y en mensajes remotos como bit recesivo. El ID base es seguido por el bit IDE (*Identifier Extension* o Identificador de Extensión) en los mensajes extendidos, que es transmitido como bit dominante en el formato estándar (dentro del campo de control) y como un bit recesivo en el formato extendido. En caso de una colisión del bus, el formato estándar prevalece sobre el formato extendido.

El formato extendido comprende dos secciones: el ID base con 11 bits y el ID extendido con 18 bits. El bit SRR (*Substitute Remote Request*) sustituye al bit RTR y es transmitido como un bit recesivo. En caso de que este bit sea enviado como bit dominante, los nodos receptores tendrán que ignorarlo. Pero no es ignorado por el bit *stuffing* ni por la controversia del bus, debido a que el bit SRR es recibido antes que el bit IDE, un nodo receptor no puede decidir instantáneamente si recibe un bit RTR o un bit SRR, ya que solo el bit IDE decide si el mensaje es un mensaje estándar o uno extendido. La figura 2.9 muestra el campo de resolución de controversia para los dos formatos (estándar y extendido) de mensajes.

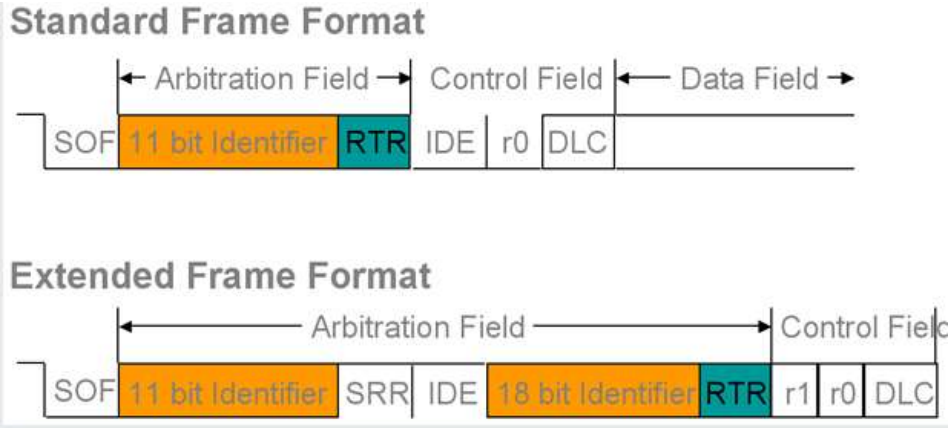


Figura 2.9 Campo de arbitrariedad para mensajes estándar y extendidos.

Nota: Si la red CAN está diseñada con el protocolo CAN 2.0B, esta es capaz de transmitir y recibir mensajes con ID estándar e ID extendido a través del bus.

Posteriormente al campo de resolución de controversia le sigue el campo de control (*Control Field*), quien se encarga específicamente del número de bytes de datos contenidos en el mensaje. El formato del campo de control es similar para los formatos de mensajes estándar y extendidos. El campo de control en el formato estándar incluye el código de longitud de datos (DLC, *Data Length Code*), el bit IDE, que es transmitido como bit dominante y el bit r0 (reservado) también enviado como bit dominante. En el formato extendido el campo de control incluye el DLC y dos bits reservados r1 y r0. Los bits reservados tienen que ser enviados como bits dominantes, aunque los nodos receptores aceptan bits dominantes y recesivos en todas las combinaciones posibles. La figura 2.10 muestra la tabla de verdad del campo DLC.

No. of Data Bytes	Data Length Code (DLC)			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d/r	d/r	d/r

Figura 2.10 Campo de control y tabla de verdad DLC.

El campo siguiente es el campo de datos (*Data Field*), que consiste en el número de bytes de datos especificados en el campo DLC, quien tiene una longitud de 4 bits y como se mencionó anteriormente es transmitido en el campo de control. El número de bytes permitido para un mensaje de datos está en el rango de 0 a 8 bytes. En la figura 2.11 se muestran los bytes mínimos y máximos permitidos por el campo DLC.

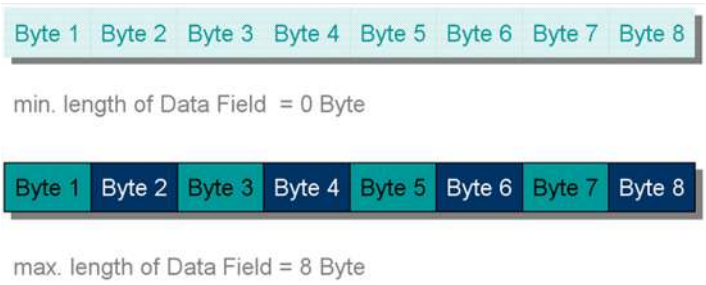


Figura 2.11 Campo de datos.

A continuación se encuentra el campo CRC (Verificación de redundancia cíclica), este campo está constituido por 15 bits y un bit recesivo delimitador (*CRC delimiter*), un valor CRC de 15 bits es calculado por el nodo transmisor y transmitido por el campo CRC. Posteriormente todos los nodos en la red que reciban el mensaje, calcularan un CRC y verificaran que los valores CRC coincidan. Si estos valores no coinciden, un mensaje de error y un error CRC serán generados.

Al campo CRC le sigue el campo de reconocimiento o campo ACK (*Acknowledgement Field o ACK Field*), utilizado para indicar si el mensaje fue recibido correctamente y el cual consta de 2 bits de longitud, un bit llamado ACK slot y otro llamado ACK delimiter. Un nodo transmisor envía dos bits recesivos en el campo ACK. Un nodo receptor, quien ha recibido un mensaje correctamente, reporta esto al nodo transmisor enviando un bit dominante durante el bit ACK slot. Si el nodo transmisor detecta un ACK positivo, que es un bit ACK slot dominante, el transmisor sabrá que por lo menos un nodo en la red recibió correctamente el mensaje. En la figura 2.12 se ilustra el campo ACK.

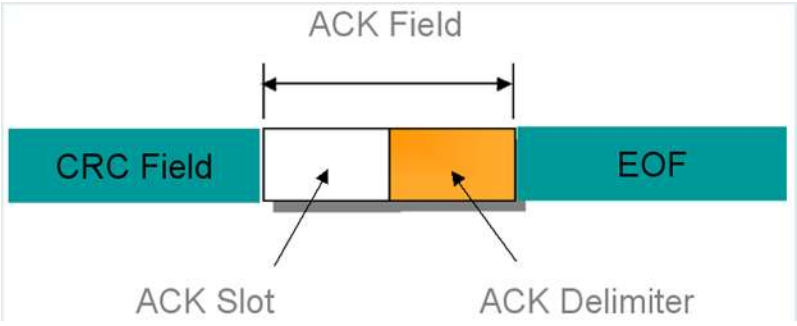


Figura 2.12 Campo ACK.

Por último se encuentra el campo *End of frame* (EOF), que consiste de 7 bits recesivos. Este campo fue introducido debido a que un mensaje de error causó una falla CRC global y esta debería haber sido transmitida dentro de los mensajes de datos o remotos. Entre dos mensajes deben existir tres bits recesivos, que constituyen el campo de intermisión (*Intermission Field*), durante la intermisión, ningún nodo puede comenzar a transmitir. La única acción posible, es la señalización de una condición de sobreflujo. La figura 2.13 muestra el campo EOF.

Figura 2.13 Campo EOF.

Solicitud de Frame Remoto (Remote/Request Frame).

Como se mencionó anteriormente es posible que un nodo pueda necesitar información de cierto tipo, que este no tenga, para llevar acabo cierta tarea. En este caso, el nodo que necesite dicha información puede iniciar una petición para la transmisión de los datos que necesita obtener de algún nodo en específico, enviando un mensaje o frame remoto. Este tipo de mensaje se encuentra formado por seis campos mostrados en la figura 2.14, en vez de los siete campos con los que cuentan los mensajes de datos (figura 2.8.).

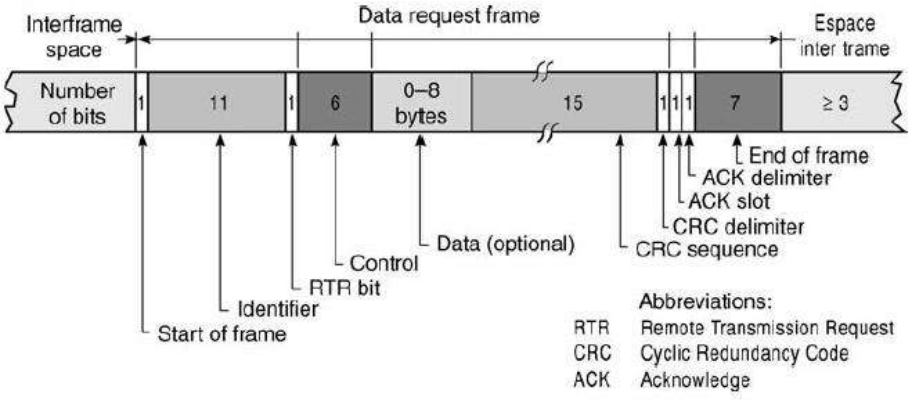


Figura 2.14. Composición del mensaje remoto.

Existen dos diferencias entre los mensajes de datos y los remotos. Primero, el bit RTR es transmitido como un bit dominante en los frames de datos y como un bit recesivo en los frames remotos y segundo en los frames remotos el campo de datos es un campo opcional, es decir, puede o no haber campo de datos.

En caso de que un mensaje de datos y uno remoto cuenten con el mismo ID al iniciar una transmisión al mismo tiempo, el frame de datos gana la controversia en el bus, debido a que el bit RTR es dominante y este sigue al ID del mensaje. En la figura 2.15 se muestra un frame de datos y uno remoto con IDs idénticos y como el mensaje de datos gana la controversia del bus debido al bit RTR.

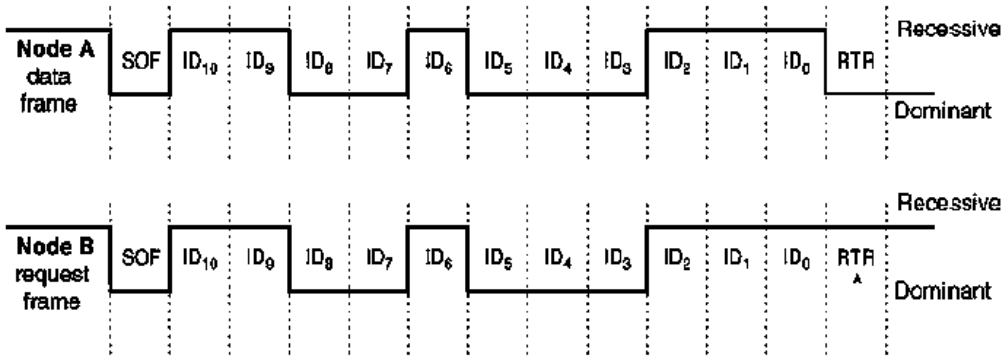


Figura 2.15. Mensaje estándar (Dato/remoto), con ID idénticos.

Debido a que el bit RTR por definición siempre es un bit recesivo en los frames remotos, se puede decir, para un identificador estándar, que un frame de datos siempre tendrá mayor prioridad sobre el frame remoto, lo cual es razonable debido a que el frame de datos contiene la información que supuestamente fue requerida en ese instante por el mensaje remoto.

2.4.2. Manejo de Errores en el CAN bus.

El manejo de errores es una de las partes más importantes del protocolo CAN, y es esencial entender el mecanismo de detección de errores, con el fin de beneficiarse de todas sus ventajas. Uno de los propósitos de este mecanismo no solo es habilitar las fallas ocurridas en el hardware y/o perturbaciones detectadas en el mismo, sino también, y especialmente, localizar estas fallas y/o perturbaciones para tratar de intervenir si es posible.

Cualquier dispositivo que se ajuste al protocolo CAN debe tener dos contadores internos, el “Contador de errores de transmisión” y el “Contador de errores de recepción”, con el propósito de grabar (contar) los errores ocurridos durante la transmisión o recepción de mensajes. Este mecanismo es muy parecido al juego de serpientes y escaleras: donde a veces se sube, y posteriormente se puede caer, aunque en el protocolo CAN enfocado a un lenguaje técnico:

- Si el mensaje es transmitido o recibido correctamente, el contador en cuestión es decrementado.
- Si el mensaje contiene un error, el contador en cuestión es incrementado.

El protocolo CAN cuenta con distintos métodos para detectar elementos parásitos o que no estén funcionando adecuadamente. En el protocolo existen varias fuentes de errores, los cuales se encuentran clasificados de acuerdo a su jerarquía. Los diferentes tipos de errores que pueden ocurrir se encuentran en la capa física, los cuales son errores de bit y errores en el bit *stuffing*; y en los mensajes (*frame layer*), los cuales se dan por información errónea o debida a que su estructura interna (de los mensajes) ha sido modificada o violada.

En todos los errores, la presencia de estos debe ser señalizada por un mensaje de error, el cual es generado en el bus para informar a la red CAN que se ha producido algún error.

Los pasos que sigue el protocolo CAN-bus para el manejo de errores son:

1. Detectar un error global o local.
2. Transmitir una bandera de error (*Error flag*), globalizar el error.
3. En caso de un error local la bandera de error será seguida por una bandera de error de superposición que a su vez será seguida por un delimitador de error (*delimiter error*).
4. El mensaje transmitido será rechazado por los nodos de la red.
5. Los contadores de errores (*Error Counters*) de cada nodo serán incrementados.
6. La transmisión del mensaje será repetida automáticamente.

Globalización de Errores Locales.

Durante la transmisión de mensajes si algún nodo conectado a la red detecta un error en el mensaje este envía una bandera de error a los demás nodos conectados a la red. En la figura 2.16 se muestra un ejemplo de globalización de errores.

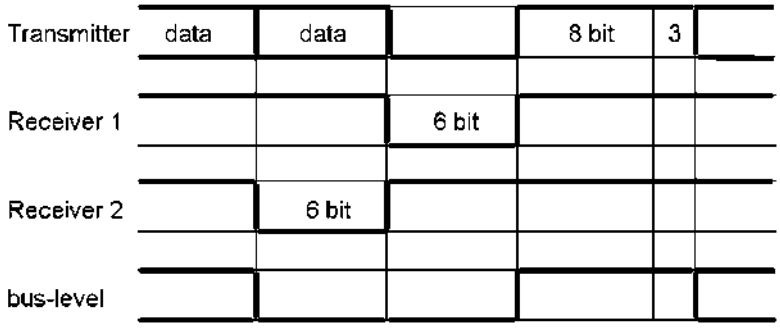


Figura 2.16. Globalización de Errores.

En la figura 2.16 se muestra la transmisión de un mensaje dentro de una red CAN de 3 nodos, en donde el receptor 2 (nodo 2) detecta un mensaje con error local en el dato recibido por lo que inmediatamente transmite un mensaje con una bandera de error (Mensaje de error) a través del bus. Posteriormente en el bit 6 del mensaje transmitido por el nodo 2 al receptor 1 (nodo 1), este nodo (nodo 1) detecta una violación de la regla del bit *stuffing*, en consecuencia el nodo 1 transmite otro mensaje de error a través del bus, lo que ocasiona que toda transmisión de mensajes sea abortada. Esta acción evita que otros nodos conectados en la red acepten mensajes y por lo tanto se asegura la coherencia de datos en toda la red. Después de que un mensaje de error ha abortado el tráfico de datos en la red y después del campo *error delimiter* y del campo de intermisión (*intermission field)*, todo nodo conectado en la red tratará de acceder al bus nuevamente y retransmitir mensajes a través del bus.

Mensaje de Error Activo.

Un mensaje de error activo (*Active Error Frame*) es generado por cualquier nodo que detecte un error en el bus. El mensaje de error está constituido por un *error flag y un error delimiter.* Los bits dominantes del *error flag* sobrescriben el mensaje de datos erróneo y provocan una retransmisión del mismo. Debido al mecanismo de globalización de errores una superposición de *error flags* puede ocurrir y no permitir la retransmisión de datos. El

campo *error delimiter* está constituido por 8 bits recesivos y permite que los nodos del bus reinicien las comunicaciones sin alguna perturbación después de un error.

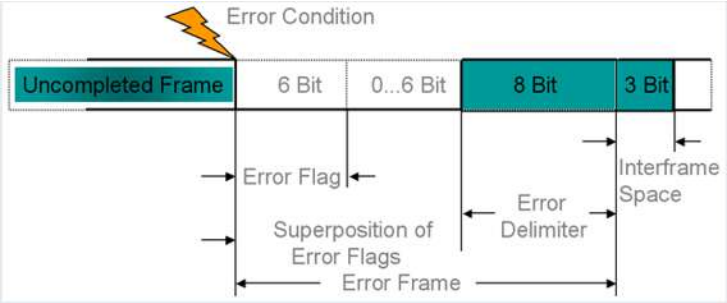


Figura 2.17. Mensaje de Error Activo.

Error en el Bit Stuffing.

Debido al uso del método NRZ (*Non-return-to-zero*),el flujo de bits en un mensaje permanece constante (mismo valor (nivel)) después de cierto tiempo ocasionando que uno o varios nodos detecten una anomalía en la red. Por lo que se decidió introducir (durante la transmisión de mensajes) deliberadamente, después de 5 bits del mismo valor (dominante o recesivo), un bit intencional con el valor contrario a los 5 bits anteriores, para indicar que la transmisión va bien. Estos bits son llamados *stuffing bits.* Estos bits son removidos, por el receptor CAN mediante un método llamado *"desttufing"*, ya que son usados únicamente para la transmisión de mensajes. En la figura 2.18 se muestra un mensaje antes de utilizar *stuffing bits* y otro mensaje con *stuffing bits,* así como, un mensaje "*desttufed*" por el receptor.

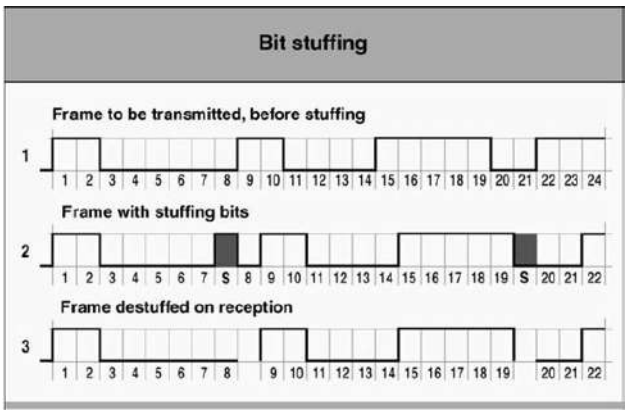


Figura 2.18. *Bit Sttufing*.

En la figura 2.19 se muestran los campos del mensaje en los cuales se utiliza el método *sttufing bit*, los campos restantes pueden ser o no ser modificados mediante este método.

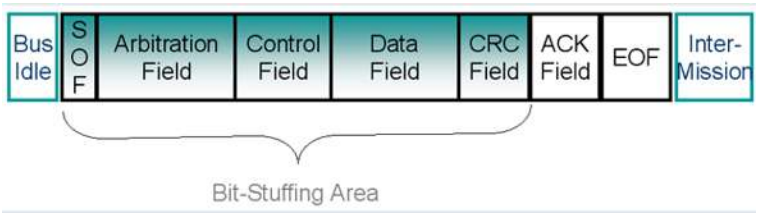


Figura 2.19 Campos del mensaje con *bit-sttufing*.

Si un nodo detecta en el momento del sexto bit un bit con el mismo valor (dominante o recesivo) durante el área *bit sttufing*, entonces se generara un mensaje de error al término del mensaje. Un error en el bit sttufing ocurre si más de cinco bits consecutivos con la misma polaridad son detectados entre el bit SOF (*start of frame*) y el *CRC delimiter*. Por lo que se generara un mensaje de error y el mensaje tendrá que ser repetido posteriormente.

Error de bit.

Todos los nodos de la red realizan monitoreo de bits (*Bit-Monitoring*), entonces, un error de bit ocurre si un transmisor envía un bit dominante pero detecta un bit recesivo en el bus o envía un bit recesivo y detecta un bit dominante en el bus. Por lo que se generara un mensaje de error el cual será transmitido en el siguiente ciclo de reloj.

Cuando un bit dominante es detectado en vez de un bit recesivo, durante el campo de controversia o el campo *acknowledgement slot*, se dice que no ocurre ningún error, ya que estos campos deben encontrarse disponibles para sobrescribir en ellos un bit dominante y mantener la funcionalidad de los campos de controversia y de reconocimiento de mensajes. En la figura 2.20 se muestran los campos en los que se realiza el monitoreo de bits.

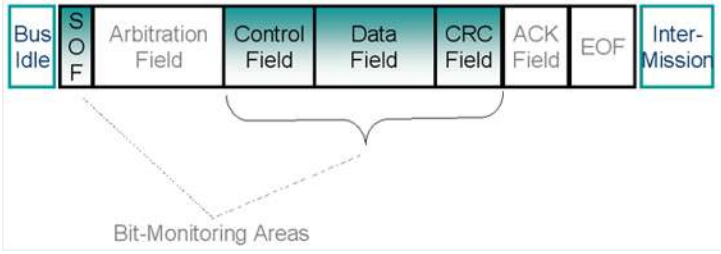


Figura 2.20. Área o campos del mensaje en los que se realiza *Bit-Monitoring*.

Error CRC (*Cyclic Redundancy Check*)

Con el campo de redundancia cíclica el transmisor calcula una suma de revisión para la secuencia del bit CRC, la cual se realiza a partir del bit SOF (*start of frame*) hasta el final del campo de datos. La secuencia CRC es transmitida dentro del campo CRC del mensaje de datos o del mensaje remoto o de petición. Un error CRC ocurre, si el resultado de la suma calculada por el receptor no es el mismo que el enviado o recibido en la secuencia CRC. Por lo tanto el receptor descartará el mensaje en cuestión y transmitirá un mensaje de error al final del bit ACK delimiter, a menos de que una bandera de error haya sido generada por alguna otra condición de error y se encuentre está bandera en proceso de transmisión o siendo transmitida en el bus. En la figura 2.21 se muestran los campos en los cuales se lleva acabo el cálculo del error CRC.

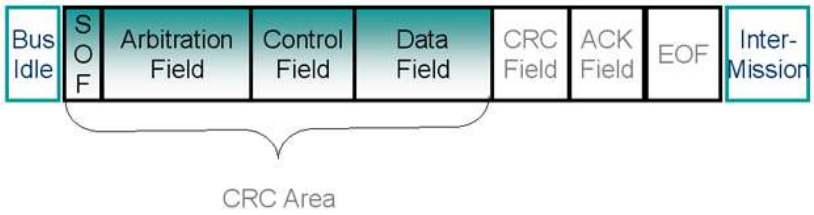


Figura 2.21. Campos para el cálculo del Error CRC.

El receptor calcula el CRC de la misma forma que el transmisor, la figura 2.22 muestra el procedimiento para el cálculo del error CRC, dicho cálculo realiza los siguientes pasos:

1. El mensaje es considerado como un polinomio y posteriormente dividido por un polinomio generador:

$$x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$$

2. El residuo de la división, originado por la división módulo 2 del polinomio es la secuencia CRC que es transmitida junto con el mensaje.

3. Por último el receptor divide el mensaje incluyendo la secuencia CRC por el polinomio generador.

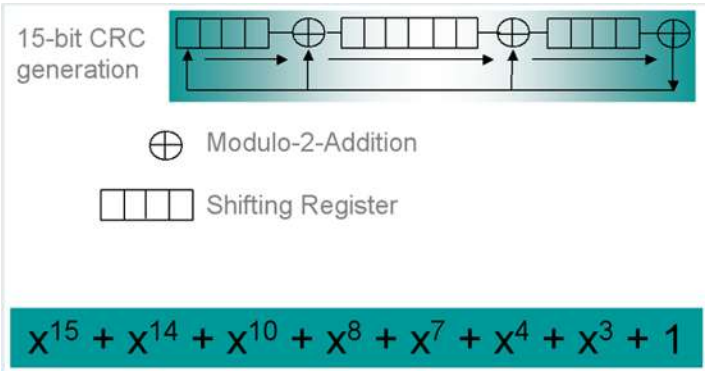


Figura 2.22. Generación del polinomio CRC.

Error de reconocimiento.

En el campo de reconocimiento (*Acknowledge Field*) de un mensaje, el nodo transmisor realiza un chequeo del bit *acknowledge slot* (el cual es transmitido como un bit recesivo), si el campo contiene un bit dominante. Si este es el caso, quiere decir que al menos un nodo ha recibido correctamente el mensaje. Si este bit es recesivo, ningún nodo ha recibido apropiadamente el mensaje y por lo tanto se genera un error de reconocimiento. Posteriormente un mensaje de error es generado y el mensaje original será transmitido nuevamente después del campo de intermisión.

Error de formato.

Si algún nodo detecta un bit dominante en cualquiera de los siguientes segmentos del mensaje: *End of frame*, *Interframe Space*, *Acknowledge Delimiter o CRC Delimiter*, el protocolo CAN define esto como una forma de violación y un error de formato es generado. Y el mensaje original es reenviado después del campo de intermisión. La figura 2.23 muestra los campos en los que se genera un error de formato.

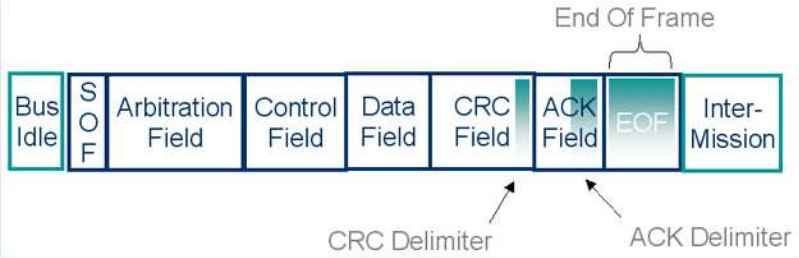


Figura 2.23. Campos en los que se genera un error de formato.

Estados de Error de un nodo CAN.

Para distinguir entre fallas temporales y permanentes, cada controlador del bus CAN conectado a la red cuenta con dos contadores de errores: el REC (Contador de error de recepción, por sus siglas en inglés, *Receive Error Counter*) y el TEC (Contador de error de transmisión, por sus siglas en inglés, *Transmit Error Counter*). Estos contadores son incrementados cada que se detectan errores, tanto en la transmisión como recepción de mensajes; y decrementados cuando la transmisión y/o recepción se realizan en forma correcta. Dependiendo de los valores de los contadores los estados de los nodos van cambiando: el estado inicial en el que el controlador CAN se encuentra es en Error Activo (*Active Error*), el controlador CAN cambia a Error Pasivo (*Passive Error*) si existe cierta acumulación de errores en los contadores.

Un nodo conectado al bus se encuentra en Error Activo cuando ambos contadores (REC y TEC) se encuentra en el rango de 0 a 127, cuando los contadores exceden estos valores el nodo cambia a Error Pasivo. El controlador del CAN bus fallará cuando exista una acumulación extensa de errores en los contadores entonces el controlador se desconectara del bus, por lo que el controlador del CAN bus se encontrará en estado *Bus-off*. Lo cual significa que el controlador estará en un estado de alta impedancia y no podrá recibir ni transmitir mensajes, el controlador solo podrá salir de este estado mediante un reset por software. Después del reset el controlador del CAN bus debe esperar a que pasen 128 x 11 bits recesivos para transmitir un mensaje. Esto debido a que algunos nodos pueden tener pendientes algunas transmisiones remotas. No se recomienda realizar un reset por hardware debido a que el tiempo de espera no sería el necesario para poder reiniciar la transmisión. En la figura 2.24 se observan los estados de error de un nodo CAN.

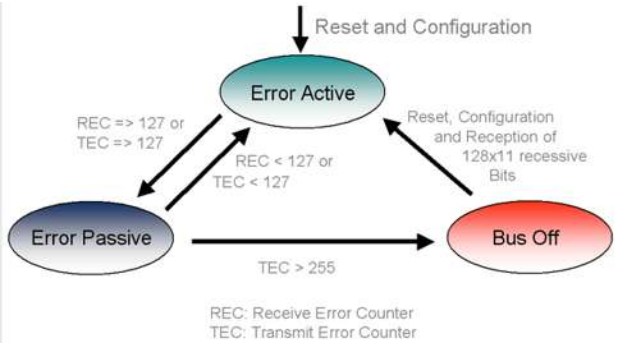
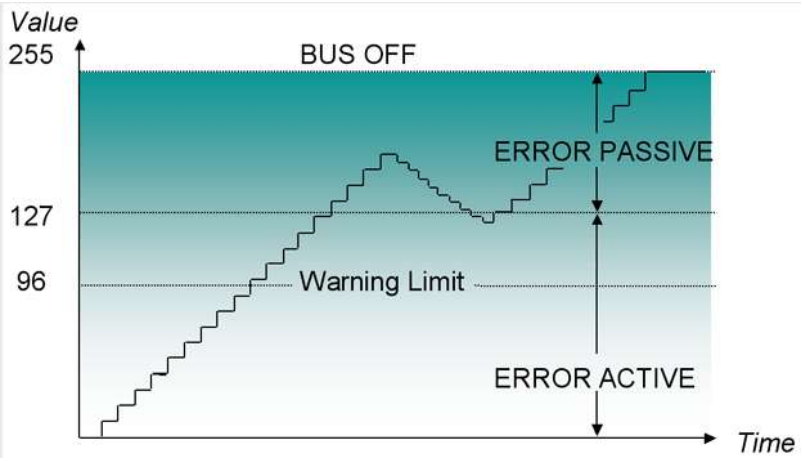
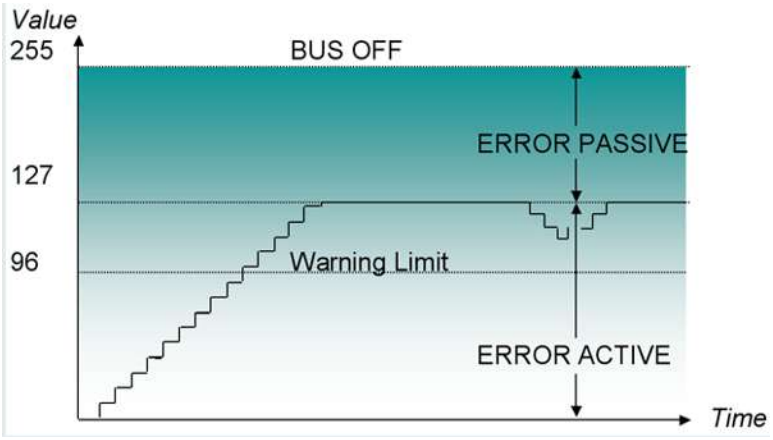


Figura 2.24. Estados de error de un nodo CAN.

En la figura 2.25a y 2.25b se muestran los estados en los cuales puede caer el controlador CAN dependiendo del número de errores que estos registren. Cuando los valores de los dos contadores se encuentren en el rango de 0-96 el nodo se encuentra en estado de error activo por debajo de la zona límite o de riesgo, es decir, el nodo se encuentra en estado normal y puede recibir y transmitir mensajes, además puede enviar mensajes de error activo (bits dominantes), que permiten destruir mensajes con algún error detectado, cuando los contadores exceden la zona de riesgo el controlador CAN genera una advertencia (una interrupción poniendo una bandera de error). Entre los valores 97 y 127, el nodo se encuentra en error activo pero el bus está fuertemente perturbado. Cuando cualquiera de los dos contadores se encuentre entre los valores 128 y 255 propiciara un cambio de estado en el nodo, a estado de error pasivo, y continuara transmitiendo y recibiendo mensajes normalmente, solo que transmitiendo mensajes de error pasivo (bits recesivos), sin poder destruir los mensajes con errores detectados. Si alguno de los contadores excede el valor de 255, el nodo en cuestión cambia a estado de *bus off*, es decir, todas las actividades CAN del nodo se detienen y este liberará el bus (estado recesivo), además de que el nodo no podrá transmitir ni recibir mensajes. Por lo que es necesario un reset por software del nodo en cuestión.



a).



b).

Figura 2.25. a) Contador TEC, b) Contador REC

Mensaje de sobreflujo.

Un mensaje de sobreflujo (*overload frame*) puede ser generado por un nodo, si debido a condiciones internas del nodo, este no se encuentra disponible para recibir el siguiente mensaje o si durante la zona interframe uno de los primeros dos bits es dominante. Otra condición de sobreflujo, es cuando un mensaje es recibido correctamente por los receptores, aun cuando el último bit del campo EOF (*End of frame*) es recibido como bit dominante. Con los mensajes de sobreflujo los nodos realizan una petición para retrasar el inicio del siguiente mensaje que transmitirán. Un nodo solo puede transmitir 2 mensajes consecutivos de sobreflujo. Este tipo de mensajes son similares a los de errores activos, solo que estos no incrementan los contadores de error y no causan retransmisión de mensajes. La figura 2.26 muestra un mensaje de sobreflujo y en donde se lleva acabo.

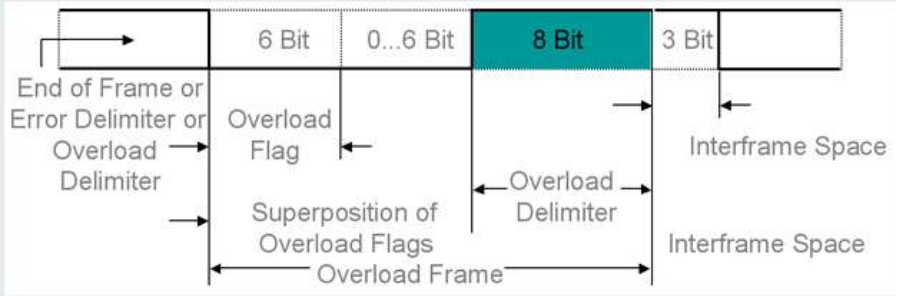


Figura 2.26. Mensaje de sobreflujo.

2.5. Estándar SAE J1939.

El estándar J1939 es una serie de recomendaciones descritas por el SAE, que fueron desarrolladas para disponer de una arquitectura en la cual múltiples sistemas electrónicos (ECU's) de un automóvil puedan comunicarse entre sí. Este fue desarrollado por el Sub-comité de Redes de Comunicación y Control de Autobuses y Camiones Eléctricos y por el Comité de Electrónicos, aunque su aplicación no solo se centra para camiones y autobuses. El estándar J1939 ha sido implementado en una amplia gama de automóviles y sistemas de transporte.

Este estándar proporciona un protocolo de comunicación a través de una red CAN. La red CAN debe estar compuesta por lo menos de dos "Unidades de Control Electrónico" interconectadas entre sí. Como se menciona en la especificación SAE J1939-11. Las cuales deben de estar conectadas a través de un cable de par trenzado, con una velocidad de datos de 250 kbits/segundos.

La capa física esta descrita en la especificación J1939-11, y describe la interfaz eléctrica del bus, el enlace de datos es especificado por SAE J1939-21, la cual nos dice las reglas de construcción de los mensajes, el acceso al bus y la detección de errores de transmisión. La capa de aplicación (J1939-71 y J1939-73) define los datos que deberá de contener un mensaje para poder ser transmitido a través de la red.

2.5.1. Formato y uso de los Mensajes (J1939-21)

El propósito de la mayoría de los mensajes del estándar J1939-21 es ser mensajes tipo broadcast. Lo cual significa que los datos son transmitidos a través de la red sin un destino específico. Esto permite que cualquier dispositivo conectado a la red utilice los datos sin requerir de un mensaje de respuesta adicional. Permitiendo que revisiones futuras de software sean realizadas con mayor facilidad (por ejemplo, asignación de direcciones). Si un mensaje debe de ser dirigido a un dispositivo en particular, una dirección específica de destino puede ser incluida dentro del identificador del mensaje. Por ejemplo, alguna petición de un valor de torque especifico del motor en lugar del valor de torque de control de frenado.

El estándar J1939 utiliza el identificador de 29 bits definido dentro del protocolo CAN 2.0B (CAN extendido) mostrado en la figura 2.27. Este identificador es utilizado ligeramente diferente en un mensaje con una dirección específica (PDU Format) comparado con un mensaje usado como broadcast (PDU specific).

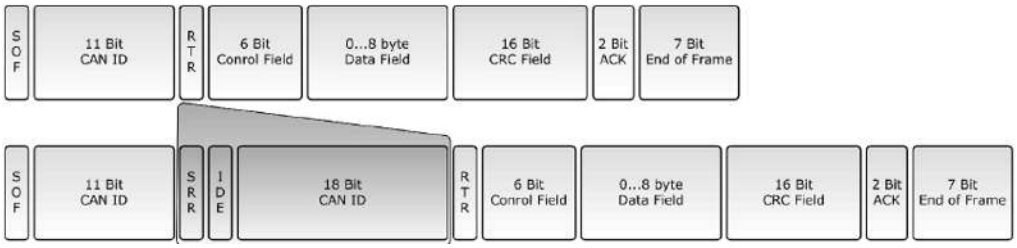


Figura 2.27. Identificador de 29 bits

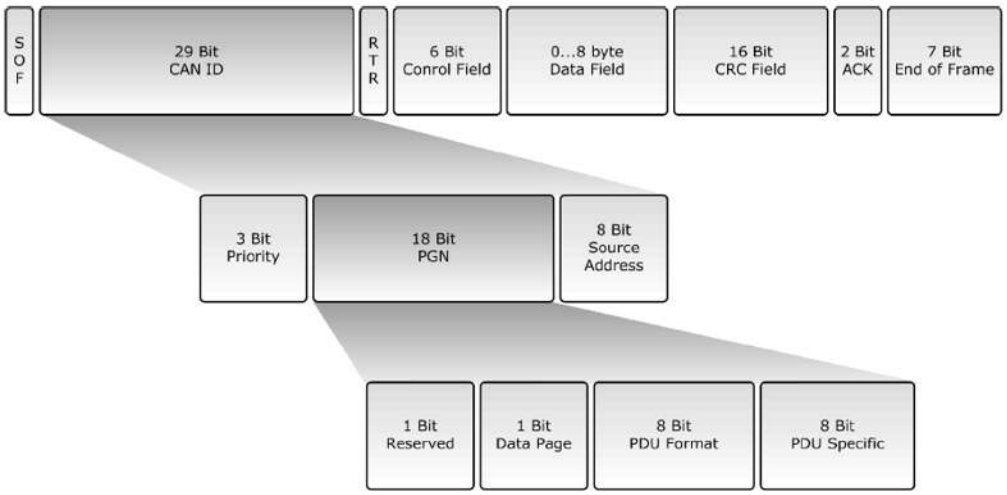


Figura 2.28. Formato de mensaje (J1939-21)

Los parámetros incrustados en el identificador de 29 bits del mensaje son divididos en tres secciones:

- Prioridad.
- PGN (Número de Parámetro de Grupo).
- Dirección de Origen de 8 bits.

Los bits *SOF, SRR, e IDE* definidos en el protocolo CAN son ignorados en este estándar y el bit *RTR* (petición remota) siempre es colocado a cero lógico en el estándar J1939.

Los primeros tres bits (3 bits priority) mostrados en la figura 2.28, son utilizados para controlar la prioridad del mensaje durante el proceso de resolución de controversias. Un valor de cero (000) cuenta con la mayor prioridad, y un valor de siete (111) con la menor prioridad. La prioridad más alta típicamente está reservada para los mensajes de control de alta velocidad, por ejemplo, el mensaje de control del torque de la transmisión del motor. Por el contrario los mensajes con datos que no son críticos en cuanto a tiempo son los de menor prioridad, como la velocidad del automóvil.

El siguiente bit se encuentra reservado y deberá ser colocado como un cero lógico en mensajes transmitidos. El siguiente bit del identificador es el selector de Datos de Página (DP), es decir, en que página de datos (0 o 1) el mensaje será definido en el estándar J1939.

El campo PDU (Unidad de Protocolo de Datos) format (PF) determina si el mensaje puede ser transmitido con una dirección de destino o si este se transmitirá como mensaje broadcast. La interpretación del campo PDU specific (PS) cambia dependiendo del valor que tome el campo PF:

- Si el campo PF se encuentra entre 0 y 239, el mensaje es direccionado (PDU format) y el campo PS contiene la dirección de destino.
- Si el campo PF está entre los valores 240 y 255, entonces el mensaje es un mensaje broadcast (PDU specific) y el campo PS contiene la Extensión de Grupo.

La Extensión de Grupo expande el número de posibles mensajes de Grupos de Parámetros broadcast que pueden ser representados mediante el identificador. El termino Número de Grupo de Parámetro (PGN) es utilizado para referirse al valor del bit de Reserva, y a los campos DP, PF y PS combinados dentro de un solo valor de 18 bits. En la tabla 2.4 se muestra un ejemplo de un Número de Grupo de Parámetro con un identificador (ID) 0xCF004EE dividido en los diferentes grupos mencionados anteriormente.

0x0C				0xF0	0x04	0xEE
000	011	0	0	11110000	00000100	11101110
---	Prio	R	DP	PF	PS	SA

Tabla 2.4 Ejemplo PGN.

En donde:

- PGN = Los campos R, DP, PF y PS, los cuales tienen un valor de 0xF004.
- PF = 0xF0 = 240, por lo tanto es un PDU specific, es decir, un mensaje broadcast.
- PS = 0x04, Extensión de Grupo = 4.

Los últimos 8 bits del identificador contienen la dirección del dispositivo al cual se transmitirá el mensaje. La dirección es la etiqueta asignada a un dispositivo conectado en una red para proporcionar una manera para accesar a esté. En una red CAN, cada dirección asignada debe de ser única (254 disponibles). Lo que significa que dos dispositivos diferentes (ECUs) no pueden usar la misma dirección.

2.5.2. Nombres y direcciones (J1939-81)

El nombre es una etiqueta de 64 bits (8 bytes) la cual asigna a cada ECU una identidad única. Este se encuentra compuesto por 10 campos y tiene la estructura mostrada en la tabla 2.5 y en la tabla 2.6 el número de byte ocupado por cada campo.

Número de Campo	Nombre del Campo	Longitud
1	Dirección Arbitraria	1 bit
2	Grupo Industrial	3 bits
3	Instancia del sistema del vehículo	4 bits
4	Sistema del Vehículo	7 bits
5	Reservado	1 bit
6	Función	8 bits
7	Instancia de la Función	5 bits
8	Instancia de la ECU	3 bits
9	Código del Fabricante	11 bits
10	Número Identificador	21 bits

Tabla 2.5 Estructura del nombre.

Número de Byte en un Mensaje CAN	Contenido/Significado
0	Número de Identidad, LSB
1	Número de Identidad
2	Bits 0-4: Número de Identidad, MSB Bits 5-7: Código del Fabricante, LSB
3	Código del Fabricante, MSB
4	Bits 0-2: Instancia de la ECU Bits 3-7: Instancia de la Función
5	Función
6	Bit 0: Bit Reservado Bits 1-7: Sistema del Vehículo
7	Bits 0-3: Instancia del Sistema del Vehículo Bits 4-6: Grupo Industrial Bit 7: Bit de la Dirección Arbitraria.

Tabla 2.6 Número de byte ocupado por los campos del nombre.

El propósito principal del Nombre es describir una ECU. Los valores más bajos del 0 al 127 del campo "Función", son pre-asignados a funciones o dispositivos "Estándar". Los valores del 128 al 254 dependen del Grupo de la Industria y del Sistema del Vehículo, lo que permite tener las mismas funciones en diferentes automóviles. Cuando una ECU conectada a una red CAN realiza una petición de dirección, el nombre es utilizado para determinar cuál ECU o dispositivo es el de mayor prioridad y por lo tanto obtener la dirección que ha sido requerida.

Todo dispositivo en una red CAN debe tener asociado por lo menos un nombre y una dirección. Sin embargo, varios nombres y direcciones de dispositivos pueden coexistir dentro de una misma ECU. Por ejemplo, un motor y un freno de motor, pueden estar en una misma ECU. La dirección del dispositivo define una fuente específica de comunicaciones o un destino específico para un mensaje. Mientras que el Nombre establece la funcionalidad e incluye un Número de Instancia único de la función cuando varios dispositivos del mismo tipo coexisten en la red.

Solo 254 dispositivos diferentes del mismo tipo pueden coexistir en una red CAN, debido al límite de direcciones. La dirección 255 se encuentra reservada como dirección global para mensajes de tipo broadcast, así como la dirección 254 como "Dirección Nula" utilizada por dispositivos que no cuentan con una dirección específica.

2.5.3. Transmisión de Mensajes (J1939-21 y J1939-7x)

Para que un mensaje de datos pueda ser enviado, este debe de estar construido por un dato de cabecera el cual describa al mensaje que será enviado. Mensajes con datos relacionados típicamente son enviados juntos dentro del mensaje. Cuando algún dispositivo conectado a la red no cuenta con datos disponibles de algún parámetro dado, provocará que el byte asociado a estos parámetros sea colocado como "no disponible" (valor hexadecimal 0xFF), y por lo tanto que el dispositivo receptor detecte el dato faltante.

Aquellos mensajes que requieren más de ocho bytes, deberán ser transmitidos en mensajes múltiples. Los mensajes múltiples pueden ser transmitidos en dos formas diferentes:

- Mensajes de anuncio de Mensajes Broadcast (TP_BAM).
- Gestión de Conexión (TP_CM).

Mensajes TP_BAM.

Este tipo de mensajes utiliza una dirección de destino global, es decir, que todo dispositivo conectado a la red recibirá los mensajes. La transmisión de estos mensajes comienza con un mensaje TP_CM, el cual tiene un PGN = 00x00EC00, y con un Byte de Control que indica que es un mensaje TP_BAM. Posteriormente los mensajes de datos son transmitidos como mensajes de Transferencia de Datos (DT), con un PGN = 0x00EB00.

Mensajes TP_CM.

Este tipo de mensajes son enviados punto a punto entre dos dispositivos. La transmisión comienza con un mensaje TP_CM y un Byte de Control el cual indica una Petición (RTS) para enviar un mensaje. El dispositivo receptor responderá con un mensaje TP_CM y con el Byte de Control indicando que se encuentra Listo (CTS) para enviar el mensaje.

Posteriormente el dispositivo transmisor comenzará él envió de los datos indicados en el campo CTS mediante mensajes de Transferencia de Datos. Este proceso continua hasta que el mensaje sea completamente transmitido. Una vez que el mensaje ha sido transmitido completamente, el dispositivo receptor transmitirá un mensaje TP_CM con el byte de

control que indica el Reconocimiento de Fin de Mensaje (EOM). El campo CTS contiene el número de paquetes en que será transmitido el mensaje completo.

2.5.4. Recepción de Mensajes (J1939-21 y J1939-7x)

Existen varios métodos con lo que respecta a la recepción de mensajes en una red CAN.

- Si un mensaje con una dirección específica es solicitado mediante un mensaje o comando RTS, el dispositivo que ha hecho la petición determinará si la dirección de destino coincide con la dirección solicitada por el dispositivo. Si estas coinciden, el dispositivo deberá entonces procesar el mensaje y transmitir algún mensaje de aceptación (Acknowledgment).
- Si el mensaje es derivado de una petición global, todo dispositivo (incluso el que genero el mensaje) conectado a la red deberá de recibirlo y al menos uno procesarlo y trasmitir un Acknowledgement de recibido.
- Si el mensaje es un mensaje Broadcast, todo dispositivo deberá determinar si el contenido de este es de relevancia para él.

2.5.5. Interpretación de un Mensaje J1939.

Con el siguiente ejemplo se describirán los principios para la interpretación de un mensaje en estándar J1939.

Se tiene un mensaje J1939 con la siguiente información:

- **Identificador CAN:** 0xCF00401.
- **Bytes de Datos:** 0xFF FF 82 DF 1A FF FF FF

El Identificador CAN provee la información siguiente:

Identificador CAN 0x0CF00401	
0x0C	Prioridad del mensaje Bit Reservado Página de Datos
F004	Formato de Parámetro (PF) Formato Específico (PS)
01	Dirección Fuente.

Tabla 2.7. Identificador CAN (ID-CAN)

En la tabla 2.7 se muestran los campos del identificador del ejemplo descrito en el párrafo anterior, en donde, los primeros tres bits de los primeros dos bytes (0xC0 = 0000 1100 (formato binario)) no son utilizados debido a que el identificador está constituido por 29 bits. Los siguientes 3 bits representan la prioridad del mensaje, en el caso del ejemplo, la prioridad es igual a 3 (011), el bit siguiente es el bit de reserva, los bits siguientes representan (F004) la página de datos (DP), los cuales se utilizan para determinar completamente el campo PGN.

El último byte (01) del ID-CAN es la Dirección Fuente (SA), a la cual será enviado el mensaje de datos. En el ejemplo SA = 01. El campo PGN = 0x0F004 corresponde al Controlador Eléctrico del Motor (EEC) número 1, de acuerdo al estándar J1939-71.

El campo de Bytes de Datos, está representado por la cadena de bytes, mostrada en la tabla 2.8.

Bytes de Datos	FF	FF	82	DF	1A	FF	FF	FF
Posición	1	2	3	4	5	6	7	8

Tabla 2.8. Bytes de Datos.

Los bytes de datos 1, 2, 6, 7 y 8, del ejemplo no se encuentran disponibles y por lo tanto son colocados con un valor FF. Debido a que ningún valor puede contener un valor igual a FF.

El byte 3, es el parámetro Actual del Torque del Motor, de la tabla 2.8, el valor correspondiente al byte 3 es igual a 0x82 (valor hexadecimal), que a su vez es igual a 130 (valor decimal). De acuerdo al estándar J1939-71 cada bit representa el 1% del torque del

motor, además de un offset de -125. Por lo que el valor actual en porcentaje del Torque del Motor es igual al 5%.

Los bytes 4 y 5, representan el valor de la Velocidad del Motor. El primer byte (4), es el byte menos significativo (Orden de Bytes “Intel”), es decir, la Velocidad del Motor será igual a 0x1ADF (valor hexadecimal), 6879 (valor decimal). De acuerdo al estándar J1939-71, la escala por bit es igual a 0.125 rpm sin offset, por lo que la velocidad del motor actual es igual a 859.875 rpm.

Capítulo 3

Microcontrolador y Controlador CAN.

3.1. Introducción.

Desde la invención del circuito integrado, el desarrollo de la electrónica digital ha dado lugar a dispositivos más complejos. Entre ellos los microcontroladores. Hoy en día es común encontrar microcontroladores en las cafeteras, hornos de microondas, automóviles, tarjetas de adquisición de datos, control retroalimentado, automatización, instrumentación, etc.

Cada fabricante ofrece una gran variedad de versiones de un mismo microcontrolador, los cuales se diferencian entre ellos por su capacidad de memoria, tipo de encapsulado, número y tipo de periféricos incluidos en el chip. Una posible forma de clasificarlos es por el número de bits de sus registros internos, lo que se conoce como longitud de palabra del dispositivo. Atendiendo a su longitud de palabra se clasifican en microcontroladores de 4, 8, 16 y 32 bits.

3.2. Microcontrolador PIC18F4580.

El microcontrolador PIC18F4580 de MICROCHIP cuenta con una arquitectura *Integrated CAN Controller* (controlador CAN integrado), lo que significa que este microcontrolador incluye un módulo de control para la implementación del protocolo CAN que actúa como controlador de la red CAN sin la necesidad de conectar uno externamente. Entre las características especiales que posee este microcontrolador, nos encontramos que el modulo CAN puede trabajar hasta en seis diferentes modos de operación. En la figura 3.1 se observa un microcontrolador PIC18F4580

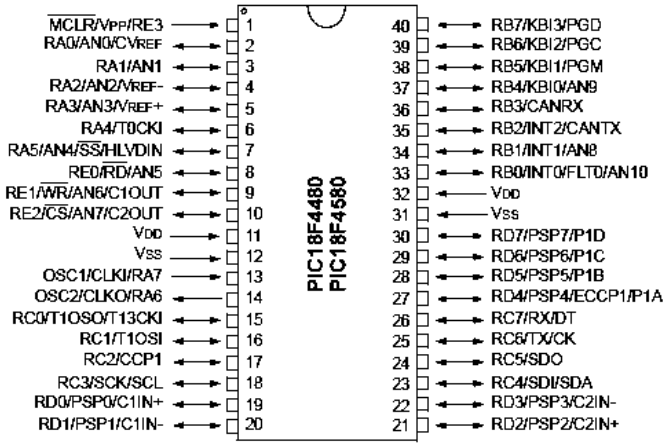


Figura 3.1 Microcontrolador PIC18F4580.

Como se explicó en el capítulo 2, el modulo CAN es una interfaz serie que permite la comunicación con otros periféricos o microcontroladores conectados en la misma red. Esta interfaz o protocolo, fue diseñada para permitir la comunicación en entornos susceptibles al ruido. Algunos de los microcontroladores cuentan con un módulo ECAN, el cual es un controlador CAN mejorado implementado en las versiones más recientes de algunas familias de microcontroladores, el cual cuenta con los protocolos CAN 2.0A y CAN 2.0B tal y como lo define BOSCH. El módulo ECAN del microcontrolador PIC18F4580 soporta los estándares pasivos (manejan únicamente mensajes estándar) como el CAN 1.2, CAN 2.0A, CAN 2.0B y el estándar activo CAN 2.0B (maneja tanto mensajes estándar como extendidos).

3.2.1. Módulo CAN en el microcontrolador PIC18F4580.

El módulo ECAN cuenta con las siguientes características:

- Implementa el protocolo CAN 1.2, CAN 2.0A y CAN 2.0B.
- Filtrado de bytes de datos.
- Manejo de mensajes o tramas estándar y/o extendidos.
- Longitud de datos de 0-8 bytes.
- Velocidad de transmisión programable hasta 1 Mbits/seg.
- Compatibilidad del módulo CAN con dispositivos de la misma familia.
- Tres modos de operación.
    - Modo 0: Modo de herencia.

    - Modo 1: Modo de herencia mejorado con soporte DeviceNet.
    - Modo 2: Modo FIFO (*First in first out*) con soporte DeviceNet.
- Soporte de mensajes remotos con manejo automatizado.
- Doble buffer de recepción con dos buffers receptores de almacenamiento de mensajes priorizados.
- Seis buffers de mensajes programables como Rx o Tx.
- 16 (identificadores estándares o extendidos) filtros de aceptación que pueden ser enlazados a una de las cuatro mascaras.
- Dos máscaras de filtros de aceptación que pueden ser asignadas a cualquier filtro.
- Un filtro de aceptación que puede ser usado como filtro de aceptación o mascara de filtro de aceptación.
- Tres buffers de transmisión dedicados con aplicación en priorización especificada y con capacidad de abortar la transmisión.
- Señalización de errores mediante interrupciones para todos los receptores y transmisores de errores CAN.
- Fuente de reloj programable.

El modulo CAN está constituido por una 'máquina de estados' (protocol engine), control de mensajes y un registro para el almacenamiento de estos. El protocol engine manipula automáticamente todas las funciones de recepción y transmisión de mensajes del CAN bus. Los mensajes a transmitir primeramente son cargados en los registros de datos apropiados. Los errores y estados del CAN bus pueden ser examinados leyendo los registros dedicados al manejo de errores y estados del CAN bus. Cualquier mensaje detectado en el bus CAN es analizado en busca de errores y comparado con los filtros para verificar si este es recibido y almacenado en uno de los dos registros de recepción. El modulo CAN soporta mensajes de datos estándar, mensajes de datos extendidos, mensajes remotos, mensajes de error y mensajes de sobreflujo en la recepción.

3.2.2. Modos de funcionamiento del módulo CAN en el microcontrolador

Modo de configuración (Configuration Mode).

El modulo CAN tiene que ser inicializado antes de ser activado para su funcionamiento. Esto es posible solo si el modulo está en modo de configuración. El modo de configuración es inicializado mediante el bit REQOP2, solamente si el bit OPMODE2 se

encuentra en 1 lógico la inicialización del módulo puede llevarse a cabo. Ya con el bit OPMODE2 = ‘1’, el diseñador puede escribir en los registros de configuración del módulo CAN por ejemplo, en los registros de máscaras de aceptación y en los registros de filtros de aceptación, dependiendo de las características de operación del módulo CAN, (Las máscaras de los filtros son usadas para determinar que bits del ID son examinados junto con los filtros y determinar si el mensaje entrante es aceptado). Una vez configurados los registros del módulo CAN este es activado colocando a cero los bits de control del registro REQOP.

Todos los registros que controlan la configuración del módulo no pueden ser modificados cuando este se encuentre en operación. El modulo CAN no permitirá entrar al modo de configuración mientras una transmisión o recepción de mensajes se encuentre en progreso. El modo de configuración funciona como un candado de protección hacia los registros:

- Registros de configuración.
- Registros de selección de modo de funcionamiento.
- Registros del bit timing.
- Registros de los filtros de aceptación de identificadores de mensajes.
- Registros de las máscaras de aceptación de identificadores de mensajes.
- Registros de control de filtros y máscaras.
- Registros de selección de máscaras.

En el modo de configuración, el modulo CAN no podrá transmitir o recibir mensajes. Los contadores de errores son limpiados y las banderas de interrupciones permanecerán sin cambios. Además el programador tendrá acceso a los registros de configuración que están restringidos en otros modos de funcionamiento.

Modo inactivo/bajo consumo (Disable/Sleep Mode).

En el modo inactivo/bajo consumo, el modulo CAN no transmitirá ni recibirá mensajes. El modulo CAN tiene la habilidad de colocar en “1” lógico el bit WAKIF (el bit WAKIF se encarga de generar una interrupción en el microcontrolador cuando se detecta actividad del CAN bus) debido a la actividad del bus; sin embargo, cualquier interrupción pendiente permanecerá activa y los contadores de errores mantendrán los valores registrados hasta ese momento.

Si los bits REQOP<2:0> son colocados en ‘001’, el modulo CAN entrará al modo *Disable/Sleep.* La función de este modo es similar a deshabilitar otros periféricos del microcontrolador. Lo que ocasiona que el reloj interno del módulo CAN sea detenido hasta que el microcontrolador detecte actividad sobre el CAN bus y el modulo CAN sea activado. En dado caso que el modulo sea activado, este esperará 11 bits recesivos sobre el bus CAN, detectando esta condición como un bus inactivo, aceptando posteriormente el comando del módulo *disable/sleep.* Los bits del registro OPMODE<2:0> = ‘001’ indican si el modulo entro correctamente al modo *disable/sleep.* El pin TXCAN permanecerá en estado recesivo mientras el modulo se encuentre en modo *disable/sleep.*

La única interrupción disponible del módulo en el modo *disable/sleep*, es la interrupción ocasionada por el bit WAKIF. El modulo se coloca automáticamente en el anterior modo de operación después de ser habilitado (*wake-up*), es decir, si el módulo cambia del modo normal al modo *disable/sleep* durante el procesamiento de información, después de un mensaje que provoque un *wake-up* en el módulo, este automáticamente cambiará al modo de operación normal que era el modo anterior en que se encontraba el módulo CAN antes de realizar el cambio de operación. En otro caso, si el procesador del módulo recibe un mensaje o interrupción *wake-up* mientras este se encontraba inactivo. Más de un mensaje puede perderse durante la transición. El número de mensajes que se pueden perder dependen del tiempo en que tarde el procesador del módulo en inicializar la actividad del bus, así como de la velocidad de los mensajes entrantes.

Modo normal (Normal Mode).

Este modo, es el modo de operación estándar de módulo ECAN de la familia de microcontroladores PIC18F2480/2580/4480/4580. En este modo de operación, el microcontrolador monitorea constantemente todos los mensajes del bus, así como los bits de reconocimiento, mensajes de error, etc. además es el único modo en que el microcontrolador, transmitirá mensajes sobre el bus CAN.

Modo de solo escuchar (Listen only Mode).

Este modo provee un método para la familia de microcontroladores PIC18F2480/2580/4480/4580 que permite recibir todos los mensajes, incluyendo los mensajes con errores. Además puede ser usado para monitorear aplicaciones del bus o para detectar la velocidad de este en situaciones de conexión dinámica (*hot plugging*). Para

detectar automáticamente la velocidad del bus, para realizar esta función es necesario que al menos otros dos nodos se encuentren en comunicación entre ellos.

El modo de solo escuchar es un modo silencioso, es decir, no se pueden transmitir mensajes mientras el módulo ECAN se encuentre en este modo, incluyendo banderas de error o señales de reconocimiento. Además los filtros y las máscaras pueden ser configurados para permitir que solo se carguen ciertos mensajes en los registros de recepción del módulo, o las máscaras de los filtros pueden ser puestas a cero para permitir que solo pase un mensaje con cierto identificador (ID). Los contadores de error son reinicializados y desactivados en este modo.

Modo bucle (Loopback Mode).

Este modo permite la transmisión interna de mensajes, de los buffers de transmisión a los buffers de recepción sin la transmisión de estos sobre el bus CAN. Además puede usarse en sistemas de desarrollo o prueba. En este modo, el bit ACK es ignorado y el microcontrolador permitirá mensajes entrantes del mismo microcontrolador, como si estos fueran transmitidos de otro nodo. Al igual que el modo anterior este también es un modo silencioso.

El pin TXCAN, es puesto como puerto de entrada/salida (I/O) mientras el microcontrolador se encuentra en modo bucle. Los filtros y las máscaras pueden ser usados para permitir que algunos mensajes en particular puedan ser cargados dentro de los registros de recepción. Las máscaras pueden ser puestas a ceros para proveer un modo en el que se acepten todos los mensajes.

Modo de Reconocimiento de Errores (Error Recognition Mode).

El modulo puede ser configurado para ignorar todos los errores y recibir cualquier mensajes. En este modo, es activado colocando los bits RXM <1:0> = ‘11’ del registro RXBnCON. Los datos que se encuentran en el buffer de ensamble de mensajes, son copiados en el buffer de recepción y pueden ser leídos mediante la interfaz del CPU.

3.2.3. Modos Funcionales del Módulo CAN.

Además de los modos de operación del módulo CAN, el módulo ECAN tiene un total de 3 modos funcionales. Estos modos son identificados como Modo 0, Modo 1 y Modo 2.

Modo 0: Modo herencia (*Legacy Mode*).

El modo 0 es diseñado para ser completamente compatible con los módulos CAN usados en los dispositivos PIC18CXX8 y PIC18FXX8. Es el principal modo de operación después de cualquier condición de reset. Como consecuencia, cualquier configuración usada en el módulo CAN de los dispositivos PIC18XX8 puede ser usado en el módulo ECAN sin necesidad de cambiar la configuración.

El modo herencia cuenta con:

- Tres buffers de transmisión: TXB0, TXB1 y TXB2.
- Dos buffers de recepción: RXB0 y RXB1,
- Dos máscaras de aceptación, una para cada buffer de recepción: RXM0 y RXM1.
- Seis filtros de aceptación, 2 para RXB0 y 4 para RXB1: RXF0, RXF1, RXF2, RXF3, RXF4 y RXF5.

Modo 1: Modo herencia mejorado (*Enhanced Legacy Mode*).

El modo 1 es similar al modo 0, con la excepción de que cuenta con más recursos que el modo 0. El modo 1 cuenta con 16 registros de filtros de aceptación (selector del mensaje a ser aceptado por el controlador CAN) y dos registros de máscaras de aceptación. El filtro numero 15 puede ser configurado como filtro de aceptación o mascara de aceptación. Además de tres buffers de transmisión y dos de recepción, cuenta con seis buffers más de mensajes. Uno o más de estos buffers pueden ser configurados como buffers de transmisión o de recepción. Estos buffers adicionales también pueden ser configurados para manejar automáticamente mensajes RTR (remotos).

Catorce de los dieciséis filtros de aceptación pueden ser asociados dinámicamente a cualquier buffer de recepción y/o a cualquier registro de mascara de aceptación. Esta capacidad puede ser utilizada para asociar más de un filtro a un buffer.

El modo 1 cuenta con:

- Tres buffers de transmisión: TXB0, TXB1 y TXB2.
- Dos buffers de recepción: RXB0 y RXB1.
- Seis buffers programables como TX o RX: B0-B5.
- Manejo automático de mensajes RTR en B0-B5.
- Dieciséis filtros de aceptación asignados dinámicamente: RXF0-RXF15.
- Dos registros de máscaras de aceptación dedicados;
    - RXF15 programable como una tercera mascara: RXM0-RXM1, RXF15.
- Filtro de datos programable para mensajes con identificadores estándar: SDFLC.

Modo 2: Modo FIFO (*First in, first out*) mejorado.

En el modo 2, dos o más buffers de recepción son utilizados para formar un buffer de recepción tipo FIFO (*first in, first out*). Este tipo de buffer no tiene relación uno a uno entre el buffer de recepción y los registros de los filtros de aceptación. Cualquier filtro que sea habilitado y enlazado a cualquier buffer FIFO de recepción puede generar aceptación y causar que el buffer FIFO sea actualizado.

La longitud de los buffers FIFO es programada por el usuario, la cual puede abarcar de 2 a 8 buffers de profundidad. La longitud FIFO es determinada por el primer buffer programado y configurado como buffer de transmisión. Por ejemplo, si el buffer 2 (B2) es programado como un buffer de transmisión, por lo tanto, el buffer FIFO está constituido por los buffers RXB0, RXB1, B0 y B1, creando un buffer FIFO de longitud 4. Si todos los buffers programables son configurados como buffers de recepción, los buffers FIFO tendrán una longitud máxima de 8.

Los recursos disponibles en el modo 2 son:

- Tres buffers de transmisión: TXB0, TXB1 y TXB2.
- Dos buffers de recepción: RXB0 y RXB1.
- Seis buffers programables como TX o RX; buffers de recepción en modo FIFO: B0-B5.
- Manejo RTR automático en B0-B5.

- Dieciséis filtros de aceptación: RXF0-RXF15.
- Dos registros de máscaras de aceptación dedicados; RXF15 programable como una tercera mascara: RXM0-RXM1, RXF15.
- Filtro de datos programable en identificador de mensaje estándar: SDFLC, compatible con el protocolo DeviceNet.

3.2.4. Transmisión de Mensajes CAN

Inicio de la transmisión

Para que el microcontrolador (MCU) tenga acceso de escritura en los buffers de mensajes, el bit TXREQ debe estar en estado lógico 0, indicando que el buffer de mensaje se encuentra limpio y que ningún mensaje está pendiente de ser transmitido. Como mínimo, los registros SIDH, SIDL y DLC deben de ser cargados. Si el mensaje a ser transmitido cuenta con algún tipo de datos, los registros de datos, también deben ser cargados. Si el mensaje utiliza identificadores de mensajes extendido, los registros EIDH:EIDL deben ser cargados y el bit EXIDE=1.

Para iniciar la transmisión, el bit TXREQ debe ser colocado a uno lógico para cada buffer que será transmitido. Cuando el bit TXREQ=1, los bits TXABT, TXLARB y TXERR serán puestos a cero lógico. Para que una transmisión sea exitosa, debe existir por lo menos un nodo con la misma velocidad de baudios en la red.

La transmisión iniciara cuando un nodo detecte que el bus se encuentra disponible. Posteriormente el nodo que comenzará la transmisión será el que tenga la mayor prioridad en el mensaje.

Abortar la Transmisión

El MCU, puede solicitar abortar un mensaje, colocando el bit TXREQ=0 asociado al buffer de mensaje correspondiente (TXBnCON ‘Registro de control del buffer de transmisión’ <3> ‘bit 3 del registro’, $0 \leq n \leq 2$’ o BnCON ‘Registro del buffer de control en modo de transmisión’ <3> ‘bit 3 del registro’, $0 \leq n \leq 5$). Colocando a uno lógico el bit ABAT (CANCON<4>) se solicitara abortar todos los mensajes pendientes. Si el mensaje no ha comenzado a ser transmitido, o si este ya ha comenzado la transmisión pero fue interrumpido por perder la controversia en el bus o por algún mensaje en él, la petición de

abortar será aceptada. De la misma manera, si un mensaje está siendo transmitido durante una petición de abortar el mensaje, y el mensaje perdió la controversia del bus o se ha detectado un mensaje con errores, el mensaje en cuestión no podrá ser retransmitido y el bit TXABT será puesto a uno lógico, indicando que el mensaje ha sido abortado satisfactoriamente.

Una vez que se ha realizado una petición de abortar el mensaje, colocando el bit ABAT a uno lógico o el bit TXABT, estos no podrán ser limpiados para cancelar la petición de abortar el mensaje. Solo el hardware del módulo CAN o por una condición de alguno de los puertos podrá limpiar los bits y cancelar la petición.

Prioridad de Transmisión

La prioridad de transmisión es una priorización interna de la familia de microcontroladores PIC18F2480/2580/4480/4580, de los mensajes pendientes de transmisión. Esta prioridad de transmisión no tiene nada que ver con la priorización implícita en la arbitrariedad de los mensajes. Internamente el MCU antes de enviar el bit Start-of-frame (SOF), compara todos los buffers que han sido encolados para su transmisión.

El buffer con mayor prioridad será el primero en enviar mensajes. Si existen dos buffers con la misma prioridad, el buffer con el número mayor será el primero en transmitir mensajes. Si los bits TXP para un buffer de mensaje en particular son puestos a unos lógicos (TXP=’11’), este buffer tendrá la prioridad más alta, y por el contrario si TXP=’00’ de algún buffer, este será el que tenga la prioridad más baja. En la figura 3.2 se muestran los buffers de transmisión del microcontrolador.

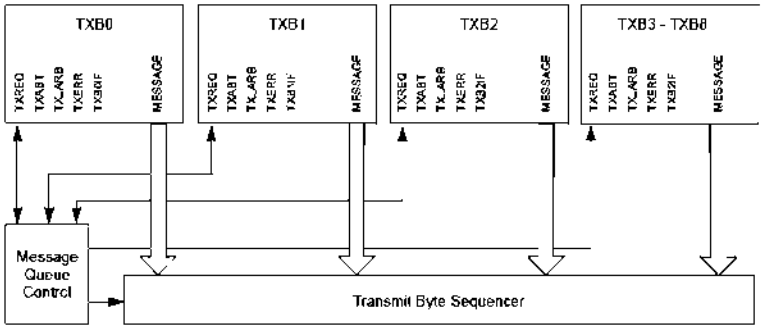


Figura 3.2. Buffers de transmisión.

3.2.5. Recepción de Mensajes

De todos los buffers de recepción, el buffer MAB (*Message Assembly Buffer*) está comprometido siempre a recibir el siguiente mensaje proveniente del bus. El MCU puede accesar a un buffer mientras otro buffer está disponible para recibir mensajes o se encuentra procesando un mensaje previo.

Todo el contenido del buffer MAB es movido hacia el buffer de recepción una vez que el mensaje es aceptado. Esto significa que sin importar el tipo de identificador del mensaje y el número de bytes de datos recibidos, todo el buffer de recepción es sobrescrito con el contenido del buffer MAB.

Cuando un mensaje es movido a cualquier buffer de recepción, el bit asociado RXFUL es puesto a uno lógico. Este bit debe ser limpiado por el MCU cuando este haya procesado completamente el mensaje en el buffer, para permitir que un nuevo mensaje sea recibido por el buffer. Una vez que un mensaje es cargado en cualquiera de los buffers, el firmware puede determinar exactamente que filtro ha causado la recepción checando los bits de los filtros en los registros RXBnCON o BnCON.

Un mensaje recibido es considerado un mensaje con identificador estándar si el bit EXID en el registro RXBnSIDL o el registro BnSIDL se encuentra a cero lógico (EXID=0). En cambio si el bit EXID=1 indica que el identificador de mensaje es extendido. Si el mensaje recibido indica ID estándar el firmware necesita leer los registros SIDL y SIDH. En caso contrario, en el que el ID se extendido, es necesario leer los registros SIDL, SIDH, EIDL y EIDH.

Cada buffer cuenta con bits RXM para configurar modos especiales de recepción. En modo 0, los bits RXM<1:0> en RXBnCON definen un total de cuatro modos de recepción. En modo 1 y 2, el bit RXM1 en combinación con el bit de mascara y filtro EXID, definen los mismo cuatro modos de recepción. Normalmente, estos bits son configurados como ‘00’ para habilitar la recepción de todos los mensajes. Cuando este es el caso, para determinar si el mensaje es estándar o extendido, es configurado el bit EXIDE en el registro de filtros de aceptación. En modo 0, si los bits RXM son configurados como ‘01’ o ‘10’, el receptor aceptará solo mensajes con IDs estándar o extendidos según sea el caso.

En modo 1 y 2, configurando el bit EXID en el registro de máscaras SIDL, se asegurara que solo mensajes con IDs estándar o extendido serán recibidos. Además cuando

un buffer es configurado como un buffer de transmisión y uno o más filtros de aceptación son asociados a él, todos los mensajes entrantes que coincidan con la configuración del filtro serán descartados.

Prioridad de recepción.

En modo 0, cuando el buffer RXB0 es el de mayor prioridad y cuenta con dos filtros de aceptación de mensajes asociados a él. RXB1 es el buffer de menor prioridad contará con cuatro filtros de aceptación asociados a él. El número menor de filtros de aceptación hace que la coincidencia en RXB0 sea más restrictiva e implique una prioridad más alta para este buffer. Además existen dos filtros de máscaras de aceptación programables disponibles, uno para cada buffer de recepción.

En modo 1 y 2, hay un total de 16 filtros de aceptación disponibles y cada uno puede ser asignado dinámicamente a cualquier buffer de recepción. El buffer con el menor número será el de mayor prioridad.

Modo FIFO mejorado.

En modo 2 cuando son configurados dos de los buffers de recepción dedicados en combinación con uno o más buffers de recepción/transmisión, son usados para crear un buffer FIFO con una longitud de 8 bytes. Cualquier filtro que haya sido habilitado puede generar una aceptación de mensajes. El buffer FIFO consiste en un arreglo de buffers de recepción contiguos. En donde el primer buffer contiguo del buffer FIFO es el RXB0 y estos se expanden hasta el buffer RXB5. La longitud máxima del buffer FIFO es limitada por la presencia o ausencia del primer buffer de transmisión. Si un buffer es configurado como buffer de transmisión, la longitud del buffer FIFO es reducida.

Para determinar si el buffer FIFO se encuentra vacío o no, el usuario puede usar los bits FP<3:0> para accesar al bit RXFUL del buffer actual. Si el bit RXFUL=0, el buffer FIFO es considerado vacío. Pero si esta es igual a uno, el buffer FIFO puede contener uno o más mensajes.

3.2.6. Filtros y Máscaras de aceptación de Mensajes

Los filtros y las máscaras de aceptación de mensajes son usados para determinar si un mensaje en el buffer de ensamble de mensajes (MAB) deberá ser cargado dentro de cualquier buffer de recepción. Una vez que un mensaje ha sido recibido dentro del MAB, el campo ID del mensaje es comparado con los valores de los filtros. Si estos valores coinciden, el mensaje será cargado en el buffer de recepción apropiado. En la tabla 3.1 se muestra la tabla de verdad, que indica como es comparado cada bit del identificador con las máscaras y los filtros para determinar si el mensaje deberá ser cargado en algún buffer de recepción. Las máscaras determinan que bit se aplica a los filtros de aceptación, es decir, si algún bit de la máscara es colocado en cero, entonces este bit será aceptado sin importar el valor del bit del filtro.

Mascara bit n	**Filtro bit n**	**Identificador de mensaje bit n001**	**Aceptado o rechazado bit n**
0	X	X	Aceptado
1	0	0	Aceptado
1	0	1	Rechazado
1	1	0	Rechazado
1	1	1	Aceptado

Tabla 3.1. Tabla de Verdad Filtros/Mascaras.

*(X = no importa).

n = bit del ID que indica si el mensaje es aceptado o rechazado.

En modo 0 los filtros de aceptación, RXF0 y RXF1, y la máscara de filtros, RXM0, están asociados con RXB0. Los filtros RXF2, RXF3, RXF4 y RXF5, la máscara RXM1, están asociados con RXB1. En modo 1 y 2, adicionalmente hay 10 filtros de aceptación, RXF6-RXF15, creando un total de 16 filtros disponibles. El filtro RXF15 puede ser utilizado como filtro de aceptación o como registro de mascara de aceptación. Cada filtro puede ser habilitado o deshabilitado individualmente, colocando un uno o cero lógico en el bit RXFENn del registro RXFCONn. Cualquiera de los 16 filtros puede ser dinámicamente asociado con cualquier buffer de recepción.

Además de filtro dinámico al buffer de asociación. En modo 1 y 2, cada filtro puede también ser dinámicamente asociado a los registros de mascara de aceptación disponibles.

Como con el filtro al buffer de asociación, también se puede asociar más de una máscara a un filtro de aceptación específico.

Cuando un filtro coincide y un mensaje es cargado dentro del buffer de recepción (por ejemplo en el buffer de recepción RXB0), el número de filtro que habilita la recepción del mensaje es cargado en los bits FILHIT. En modo 0 para RXB1, el registro RXB1CON contiene los bits FILHIT<2:0>. En modo 1 y 2, cada registro de control de buffers contiene 5 bits para verificar el número de filtro que habilitó la recepción (FILHIT<4:0>).

Si más de uno de los filtros de aceptación coinciden, los bits FILHIT serán codificados al valor binario del filtro con el número menor que ha coincidido, es decir, si el filtro RXF2 y el filtro RXF4 coinciden, los bits FILHIT serán cargados con el valor de RXF2.

Los registros de las máscaras y los filtros, solo pueden ser modificados cuando el MCU, se encuentre en modo de configuración.

3.2.7. Configuración del Baud Rate

El protocolo CAN utiliza la codificación de no regreso a cero (NRZ), lo que implica que no codifica un reloj con el flujo de datos. Entonces, el reloj de recepción debe ser recuperado y sincronizado al reloj transmisor por los nodos de recepción. El receptor debe contar con algún tipo de PLL (*Phase Lock Loop*) sincronizado a los datos de transmisión para sincronizar y mantener el reloj receptor.

El bit-timing del MCU es implementado usando un DPLL, el cual es configurado para sincronizar a los datos entrantes y proveer el tiempo nominal para los datos transmitidos. El DPLL divide cada bit-time en múltiples segmentos compuestos de periodos pequeños de tiempo llamados *Time Quanta* (T_Q), dichos segmentos son mostrados en la figura 3.3.

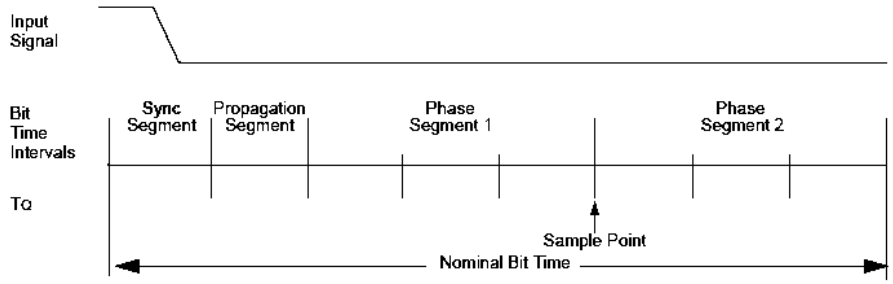


Figura 3.3. Segmentos del *Time Quanta.*

Todos los dispositivos conectados al bus CAN deben utilizar la misma velocidad del bus. Para diferentes frecuencias de reloj de dispositivos individuales, la velocidad en bauds tiene que ser ajustada por la configuración apropiada del preescaler del baud rate del número de *time quanta* en cada segmento.

El *Bit Rate Nominal,* es el número de bits transmitidos por segundo. El bit rate nominal se define como un máximo de 1 Mb/s. y está dado por:

$$\boldsymbol{T_{BIT}} = \frac{1}{Nominal\ Bit\ Rate} \tag{3.1}$$

$$\boldsymbol{T_{BIT}} = T_Q(\mu s) * número\ de\ T_Q\ por\ intervalos\ de\ bit \tag{3.2}$$

En donde $T_{BIT} = bit\ time$.

En la figura 3.3, se muestran los segmentos del bit time nominal, los cuales son:

- Segmento de sincronización (Sync_Seg).
- Segmento de tiempo de propagación (Prop_Seg).
- Segmento 1 de buffer de fase (Phase_seg1).
- Segmento 2 de buffer de fase (Phase_seg2).

Los segmentos de tiempo (y por lo tanto, el Nominal Bit Time) están, a su vez, compuestos de unidades enteras de tiempo llamados *Time Quanta o* T_Q. Por definición, el *Nominal Bit Time* se puede programar desde 8 hasta 25 T_Q. También por definición, el mínimo *Nominal Bit Time* es 1μs (1Mb/s).

La duración actual está dada por:

$$\boldsymbol{Nominal\ Bit\ Time} = T_Q * \left(Sync_{Seg} + Prop_{Seg} + Phase_{Seg1} + Phase_{Seg2}\right) \tag{3.3}$$

El *Time Quantum* es una unidad fija derivada del periodo de oscilación. Y se encuentra dado por:

$$\boldsymbol{T_Q(\mu s)} = \frac{2 * (BRP + 1)}{F_{osc}(MHz)} \quad (3.4)$$

O

$$\boldsymbol{T_Q(\mu s)} = \big(2 * (BRP + 1)\big) * T_{osc}(\mu s) \quad (3.5)$$

En donde F_{osc} es la frecuencia de oscilación del reloj, T_{osc} es el correspondiente periodo de oscilación y BRP es un entero (0 a 63) representado por valores binarios de los bits BRGCON<5:0>. Las ecuaciones 3.4 y 3.5, están basadas en la frecuencia de reloj efectiva usada por el microcontrolador.

Time Quanta

Como ya se mencionó, el Time Quanta es una unidad fija derivada del periodo de oscilación y del preescaler del baud rate. Además está relacionado con T_{Bit} y al Nominal Bit Rate.

Segmento de Sincronización

Esta parte del bit time es utilizada para sincronizar los nodos conectados al bus CAN. El flanco de la señal de entrada se espera que ocurra durante el segmento de sincronización con una duración de 1 T_Q.

Segmento de propagación.

El segmento de propagación del bit time es usado para compensar para el tiempo de retraso físico dentro de la red. Estos tiempos de retraso consisten en una señal de propagación en el bus y de un tiempo de retraso de los nodos. La duración del segmento de propagación puede ser programada desde 1 hasta 8 T_Q mediante los bits PRSEG<2:0>.

Segmentos del buffer de fase

Estos segmentos son usados para optimizar localmente el punto de muestra (Sampling point) del bit receptor dentro del bit time nominal. El sampling point ocurre entre el segmento de fase 1 y el segmento de fase 2. El final del segmento 1 determina el sampling

point dentro del bit time. El segmento de fase 1 es programable desde 1 T_Q hasta 8 T_Q de duración. El segmento de fase 2 provee un retraso antes de la siguiente transición de datos transmitidos y también puede ser programado desde 1 T_Q hasta 8 T_Q de duración. El sampling point debe ser enviado con un retraso bastante amplio o lo más próximo al 80% del bit time.

Sample point

El sample point es el instante de tiempo en el cual el nivel del bus es leído y el valor del bit recibido es determinado. El valor del bit recibido es determinado para ser el valor de la decisión mayoritaria de tres valores. Las tres muestras son tomadas en el sample point y dos veces antes, con un tiempo $\frac{T_Q}{2}$ entre cada muestra.

Tiempo de procesado de información (IPT)

El IPT es el comienzo del segmento de tiempo en el punto de muestreo que es reservado para el cálculo del nivel de bit subsecuente. Las especificaciones del protocolo CAN definen que este tiempo debe ser menor o igual a 2 T_Q

3.2.8. Sincronización

Para compensar los desplazamientos de fase entre las frecuencias de oscilación de los nodos en el bus, cada controlador CAN debe ser capaz de sincronizar la señal relevante de la señal entrante. Cuando un pulso es detectado en los datos transmitidos, la lógica interna del controlador CAN, comparara la localización del pulso con el tiempo esperado (Sync_seg).

Sincronización dura (*Hard Synchronization*)

Este tipo de sincronización se realiza solo cuando existe un cambio en la señal de recesivo a dominante durante una condición de inactividad del bus, indicando el inicio de un mensaje. Después de una sincronización dura o forzada, los contadores del bit time son reiniciados por el segmento de sincronización (Sync_seg). Este tipo de sincronización fuerza un reinicio de los segmentos de sincronización ocurridos a los límites del bit time. Si una sincronización dura ocurre, no habrá una resincronización dentro del bit time.

Resincronización

Como resultado de una resincronización, el segmento de fase 1 puede ser alargado o el segmento de fase 2 puede ser acotado. La cantidad que se agrega o se quita a los segmentos de buffer de fase tiene un límite superior dado por el periodo SJW "Modificación al segmento de sincronización" (*Synchronization Jump Width*, por sus siglas en inglés*)*. El valor del SJW será agregado al segmento de fase 1, como se muestra en la figura 3.4 o sustraído al segmento de fase 2, como se muestra en la figura 3.5.

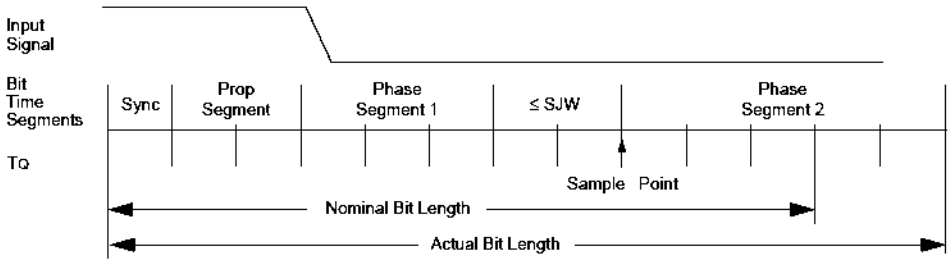


Figura 3.4 Alargando un periodo de bit (Agregando un SJW al segmento de fase 1).

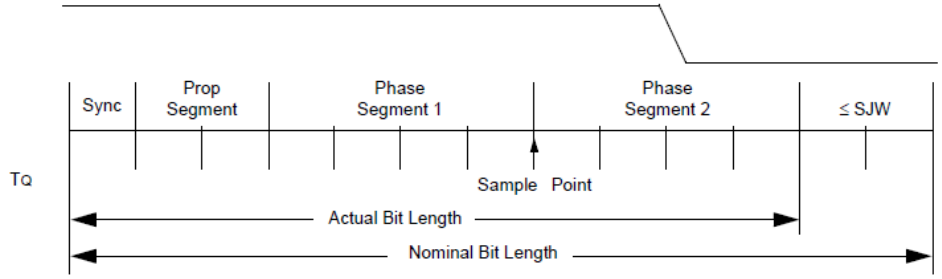


Figura 3.5 Acotando un periodo de bit (Acotando un SJW del segmento de fase 2)

La propiedad, de que solo un número máximo de bits sucesivos tenga el mismo valor, asegura una resincronización del flujo de bits durante un mensaje.

Reglas de sincronización

- Solo una sincronización dentro de un bit time está permitida.

- Un pulso será usado para sincronización solo si el valor detectado en el punto de muestreo previo (valor anterior leído del bus) difiere del valor leído inmediatamente después del pulso.
- Todos los cambios de estado de recesivo a dominante que cumplan con las reglas 1 y 2 serán usados para resincronización, con la excepción cuando un nodo transmita un bit dominante no se realizará una resincronización.

3.2.9. Programación del tiempo de los segmentos.

Algunos requerimientos son necesarios para programar el tiempo de los segmentos:

- Prop_Seg + Phase_Seg1 ≥ Phase_Seg2.
- Phase_Seg2 ≥ Sync Jump Width.

Ejemplo, asumiendo que se desea un bus CAN con baud rate igual a 125 kHz, usando un Fosc de 20 MHz. Con un Tosc de 50ns, un baud rate con prescaler de 0x04h nos da un T_Q de 500ns. Para obtener un Nominal Bit Rate de 125kHz, el Nominal Bit Time debe ser 8µs o 16 T_Q.

$$T_{OSC} = \frac{1}{20x10^6} = 50x10^{-9}\ seg = \mathbf{50}\boldsymbol{ns}$$

De la ecuación 3.4 se tiene que el time quantum es:

$$T_Q(\mu s) = \frac{2 * (BRP + 1)}{F_{osc}(MHz)} = \frac{2 * (4 + 1)}{20x10^6} = 5\ x10^{-7} seg = \ \mathbf{500}\boldsymbol{ns}$$

Con la ecuación 3.1 se tiene que el Nominal Bit Time se encuentra dado por:

$$Nominal\ Bit\ Time = T_{BIT} = \frac{1}{Nominal\ Bit\ Rate} = \frac{1}{125x10^3} = 8x10^{-6} seg = \mathbf{8}\mu\boldsymbol{s}$$

De la ecuación 3.2 se obtiene el número de T_Q necesarios para un $T_{BIT} = 8\mu s$.

$$\boldsymbol{T_{BIT}} = T_Q(\mu s) * número\ de\ T_Q\ por\ intervalos\ de\ bit$$

Lo que implica que el número total de T_Q es igual a:

$$T_Q = \frac{T_{BIT}}{T_{Q(\mu s)}} = \frac{8x10^{-6}}{500x10^{-9}} = \mathbf{16}$$

Usando 1 T_Q para el Sync_Seg, 2 T_Q para el Prop_Seg y 7 T_Q para el Phase_Seg1 y colocando el punto de muestro en el 10 T_Q después de la transición. Esto nos deja 6 T_Q para el Phase_Seg2. Debido a las reglas descritas anteriormente, el SJW tendría como máximo 4 T_Q. Típicamente, un SJW = 1 T_Q es más q suficiente.

3.3. CAN Transceiver.

El CAN Transceiver o Transceptor CAN es un dispositivo que sirve como interfaz entre el controlador del protocolo CAN y el bus físico de la red CAN. El cual provee una señal diferencial para que el controlador CAN pueda interactuar físicamente con el bus. El CAN Transceiver es un amplificador transmisor y receptor, el cual convierte una cadena de bits provenientes del controlador CAN en valores de voltajes eléctricos y viceversa además es capaz de operar a velocidades de hasta 1 Mbit/s.

Típicamente, todo nodo conectado a un sistema o red CAN debe tener un transceptor CAN para convertir las señales generadas por el controlador CAN en señales (salidas diferenciales) apropiadas para su transmisión sobre el bus. Además el transceptor actúa como un buffer entre el controlador CAN y picos de alto voltaje que puedan ser generados en el bus y causar fallas debidas a fuentes externas (transitorios eléctricos, interferencia electromagnética, descargas electrostáticas, etc.).

El CAN Transceiver es conectado al controlador o modulo CAN vía Tx (línea de transmisión) y Rx (línea de recepción). La línea de transmisión es conectada directamente al CAN bus y permite el monitoreo continuo de las señales del bus. La figura 3.6 muestra transceptor CAN conectado a un bus CAN.

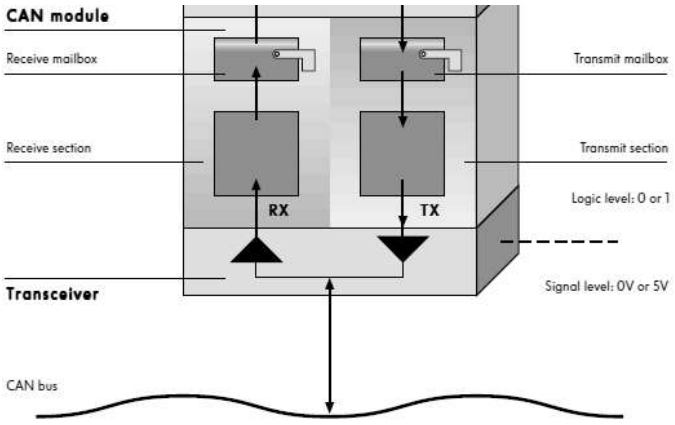


Figura 3.6. Imagen de un CAN Transceiver conectado a un módulo CAN.

3.3.2. Transceptor (Transceiver) CAN MCP2551.

El CAN transceiver MCP2551 es un dispositivo CAN de alta velocidad, tolerante a fallas que funciona como una interface entre un controlador del protocolo CAN y el bus físico. El MCP2551 de MICROCHIP provee una transmisión diferencial y una capacidad de recepción para controladores del protocolo CAN además de que es completamente compatible con el estándar ISO-11898, incluyendo requerimientos de hasta 24 volts. Y es capaz de operar a velocidades de hasta 1Mb/s. En la figura 3.7 se observa el diagrama del transceptor CAN MCP2551.

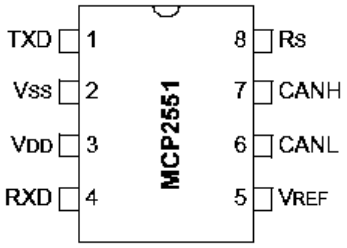


Figura 3.7 CAN Transceiver MCP2551

3.3.3. Funcionamiento como transmisor.

Los estados dominante y recesivo corresponden a los estados bajo y alto respectivamente del pin de entrada TXD. Sin embargo, un estado dominante iniciado por otro nodo CAN anulara un estado recesivo en el bus CAN.

Número Máximo de nodos.

Las salidas CAN del MCP2551 pueden manejar una carga mínima de 45Ω, permitiendo un máximo de 112 nodos conectados en la red (Dando una resistencia de entrada diferencial mínima de 20kΩ y una resistencia de terminación con un valor nominal de 120Ω). Debido a las características eléctricas del MCP2551, al conectar más de 112 nodos en la red, los mensajes transmitidos a través de esta pueden presentar perdida de datos.

3.3.4. Funcionamiento como receptor.

La salida RXD refleja el voltaje diferencial del bus entre el CANH y CANL. Los estados alto y bajo de la salida RXD corresponden a los estados recesivo y dominante respectivamente del bus CAN.

3.3.5. Protección Interna.

CANH y CANL esta protegidos contra cortocircuitos y transitorios eléctricos que puedan ocurrir en el bus CAN. Esta característica previene una destrucción de la salida transmisora durante una condición de falla.

Además el dispositivo está protegido contra corriente de carga excesiva ocasionada por un apagado térmico de la circuitería que deshabilite las 3 salidas (TXD, CANH y CANL) del transceptor CAN cuando la temperatura de unión exceda el límite nominal de 165°C.

3.3.6. Modos de operación.

El pin Rs permite seleccionar tres modos de operación diferentes:

- High-Speed
- Slope-Control
- Standby

Modo high-speed.

Este modo se selecciona conectando el pin Rs a Vss. En este modo, la salida del transmisor tiene una salida rápida y tiempos de caída rápidos para soportar las velocidades altas del bus CAN.

Modo Slope-Control.

El modo Slope-Control ayuda a reducir las interferencias electromagnéticas (EMI), limitando los tiempos de subida y caída de los pines CANH y CANL. La pendiente, o la velocidad de respuesta (SR (slew rate)), este modo es controlado conectando una resistencia externa (Rext) entre el pin Rs y el pin Vss (usualmente tierra). La pendiente es proporcional a la corriente de salida del pin Rs.

Modo Stanby.

El MCP2551 puede ser colocado en modo Stanby o "Sleep" aplicando un voltaje a Rs. En modo sleep, el transmisor es apagado y el receptor opera con corriente baja. El pin receptor del lado del controlador (RXD) continuara funcionando pero opera a una velocidad baja.

3.4. CAN Explorer (Explorador CAN).

El Explorador CAN es una herramienta de desarrollo, el cual se conecta entre el bus CAN y una PC. Con ayuda de software, el CAN explorer es capaz de monitorear la actividad de un bus para corroborar la funcionalidad de alguno de los nodos conectado a la red; el explorador CAN envía mensajes a un nodo en específico esperando recibir un mensaje en el cual el campo ACK contenga un bit dominante indicando que el nodo en cuestión ha recibido el mensaje. Si el campo ACK = 1, quiere decir que el nodo no está recibiendo el mensaje y probablemente se encuentre inactivo. En la figura 3.8 se observa el diagrama de circuito del CAN Explorer

El software utilizado para el manejo del explorador CAN es el llamado CANking desarrollado por la compañía Kvaser, el cual cuenta con un driver específicamente agregado a este software para el manejo del controlador CAN MCP2515, el cual nos permite acceso a los registros del controlador y poder configurar estos registros para diferentes aplicaciones del controlador.

Además el software permite configurar al controlador CAN MCP2515 de manera que este pueda enviar y recibir mensajes (solamente mensajes CAN en formato estándar) a través de la red, así como configurar los filtros y máscaras de aceptación permitiendo al usuario manipular los mensajes que realmente desea aceptar.

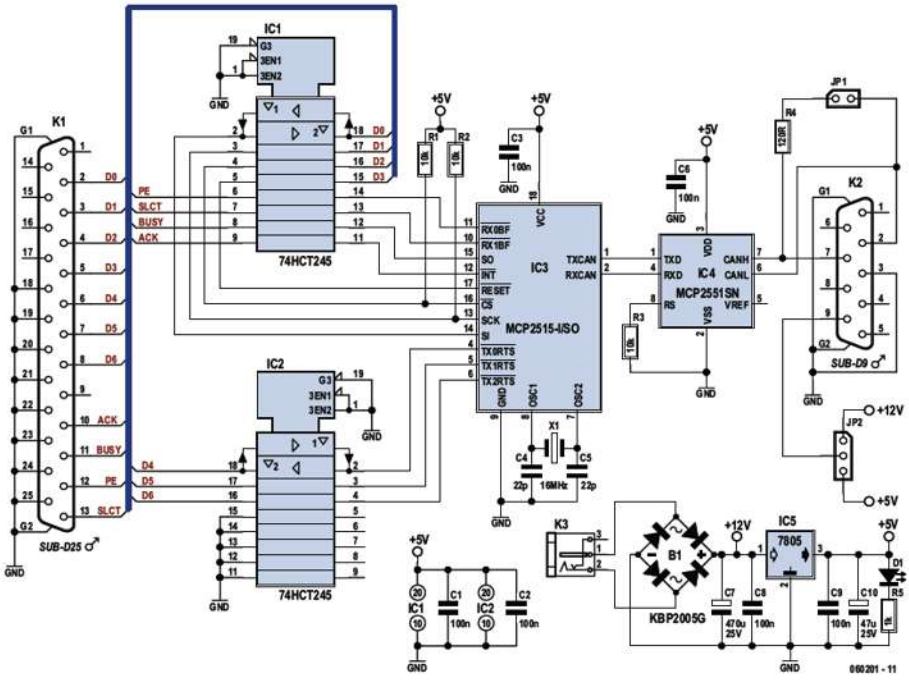


Figura 3.8. CAN Explorer.

El explorador CAN se comunica con la PC mediante el puerto paralelo, debido a que este permite una velocidad de comunicación estable y no necesita de un hardware tan robusto. El puerto paralelo debe ser configurado (en el software) de manera tal que permita el flujo de datos en forma bidireccional. El kit de desarrollo (Explorador CAN) cuenta con dos transceptores 74HCT245 (IC1, IC2) mostrados en la figura 3.8, los cuales son utilizados como buffers de señal entre el puerto paralelo y el controlador CAN MCP2515.

La comunicación hacia el controlador CAN es implementada mediante la interfaz SPI (puerto serial) con las señales producidas/recibidas por el software CANking vía pines del puerto paralelo. Las señales adicionales son utilizadas para registro/control de los estados de otras señales provenientes del controlador CAN. Además un circuito integrado MCP2551 (IC4) es utilizado como transceptor CAN.

La tarjeta elaborada cuenta con un jumper (JP1) que puede ser usado para conectar una resistencia terminadora (120 ohms) en la red, si el explorador CAN se encuentra conectado en la posición final del nodo. Esta resistencia terminadora es recomendada para reducir señales inducidas al bus. La conexión física al bus es realizada mediante un conector DB9

(K2). Los pines de este conector están asignados de forma tal que permiten alimentar la tarjeta (explorador CAN) a través del conector colocado en el bus (JP2), ya sea con 12v o 5v.

Capítulo 4

Diseño de una red Can

4.1. Introducción.

Una red CAN está organizada o constituida a base de nodos, en donde un nodo está formado por un procesador o microcontrolador, un controlador CAN, y un transceptor CAN.

Las velocidades de una red CAN puede ser de hasta 1 Mbit/s para una longitud no mayor a 40m; la velocidad de la red decrementará conforme la longitud de la misma vaya aumentando, por ejemplo, a una distancia de aproximadamente unos 500m la velocidad será de hasta 125 kbit/s.

4.2. Diseño de la red.

Una manera sencilla de entender el funcionamiento del protocolo CAN bus y del estándar J1939, es contando con una red CAN básica, la cual está constituida por dos nodos, en donde cada nodo cuenta con un microcontrolador PIC18F4580, un transceptor CAN MCP2551 y para el intercambio de datos (transmisión y recepción), un nodo monitorea temperatura y el otro nodo se encuentra monitoreando presión. Para el sensado de temperatura se utiliza el circuito integrado LM35 ya que este sensor nos entrega 10mv/°C y es de fácil interpretación; el sensor para medir la presión es el MPX5010, el cual nos entrega una salida de 0-5 volts equivalente a 0 a 10 kPa.

De acuerdo al estándar J1939 para que los mensajes sean transmitidos a una dirección específica el campo PF debe encontrarse entre 0 y 239, por lo tanto el PDU format a escoger es 92 para cuando nuestro nodo (nodo 1 o nodo 2) lea el convertidor analógico-digital y 94 para enviar la leyenda “CAN bus”, las direcciones pueden ser elegidas entre 0 y 255 (0x00 a 0xFF en hexadecimal), entonces para los nodos diseñados se eligen las

direcciones 128 para el nodo 1 y 129 para el nodo 2. En la figura 4.1 se muestra un diagrama de bloques de la red CAN.

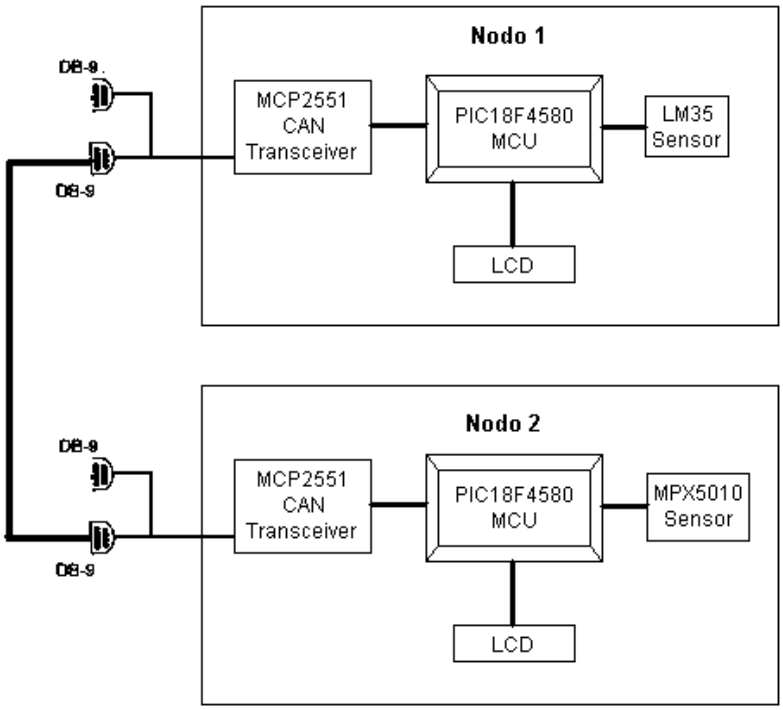


Figura 4.1. Diagrama de bloques de la red CAN.

4.2.1. El Circuito Integrado LM35

Los sensores de temperatura LM35 son una serie de circuitos integrados de precisión, cuya salida de voltaje es directamente proporcional a la temperatura en grados Celsius (Centígrados). Esta característica hace que este tipo de sensores tengan una ventaja sobre los sensores calibrados en grados Kelvin, debido a que el usuario no requiere obtener una ecuación compleja para así obtener la salida calibrada en grados centígrados.

La baja impedancia de salida, la salida lineal y su calibración inherentemente precisa, hacen que la interface para leer la salida de voltaje y la circuitería de control es realmente sencilla de implementar. Este sensor puede ser utilizado con fuentes simples (fuente positiva) o con fuentes dobles (Fuente positiva-negativa). En la figura 4.2 se observa una imagen de un sensor LM35.

Características Principales del LM35.

- Calibrado directamente a grados Celsius (Centígrados).
- Salida lineal: +10mV/°C.
- Precisión de 0.5°C @ 25°C.
- Adecuado para aplicaciones remotas.
- Voltaje de operación de 4 a 30 volts.

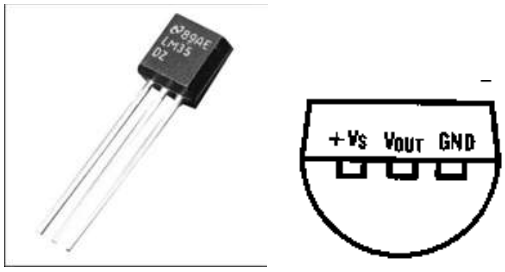


Figura 4.2. Sensor LM35.

4.2.2. Transductor de presión MPX5010

La serie de transductores piezorresistivos MPX5010, son sensores de silicio monolítico diseñados para una amplia gama de aplicaciones, particularmente para aplicaciones en las cuales sean empleados microcontroladores o microprocesadores con entradas A/D (Analógico-Digital).

Este tipo de transductores combinan técnicas avanzadas de micromaquinado, metalización en películas delgadas de silicio y un proceso bipolar, el cual proporciona una alta precisión en la señal de salida analógica y cuya señal es directamente proporcional a la presión aplicada. La figura 4.3 muestra el transductor de presión MPX5010.

Figura 4.3. Transductor de presión MPX5010

Filtrado de Ruido en el Transductor MPX5010.

Para atenuar los efectos causados por señales de ruido sobre el sensor, uno de las métodos recomendados por el fabricante, es conectar un filtro pasa bajos a la salida del transductor, así como un par de capacitores de desacople a la entrada de esté. El filtro deberá tener una frecuencia de corte de 650Hz.

El fabricante recomienda utilizar una resistencia de 750 ohms y un capacitor de 0.33µf, los cuales han sido determinados para dar mejores resultados. El filtro se muestra en la figura 4.4, así como los valores de los elementos utilizados.

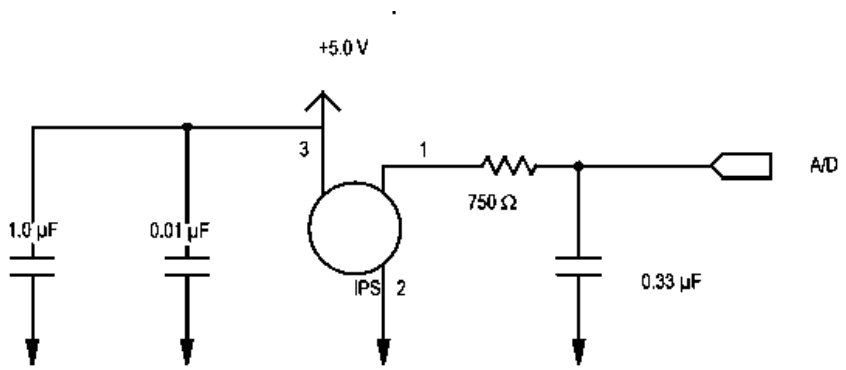


Figura 4.4. Sensor de presión con filtro RC pasa-bajos para filtrado de ruido.

4.2.3. Diseño de un nodo CAN mediante un microcontrolador PIC18F4580

El elemento principal y más importante de la red es sin duda el microcontrolador que es el encargado de recibir, transmitir y procesar los mensajes CAN, además es necesaria la conexión de un transceptor CAN con el microcontrolador para adaptar las señales del bus a los voltajes requeridos por el MCU. En la figura 4.4 se muestra el circuito esquemático con las conexiones mínimas requeridas entre el MCU y el CAN Transceiver.

Mediante las conexiones mostradas en la figura 4.5, se podría decir que se tiene un nodo CAN listo, pero se va a diseñar una red de dos nodos, por tal motivo es necesario agregar las terminales para que exista comunicación entre estos dos nodos. El estándar J1939, menciona que la comunicación del bus debe ser llevada acabó mediante un cable de par trenzado, y un conector tipo DB9, además este conector debe contar con el voltaje y la

tierra física del bus. La figura 4.5 muestra la configuración recomendada para el conector DB9 y la conexión física de la red entre nodos.

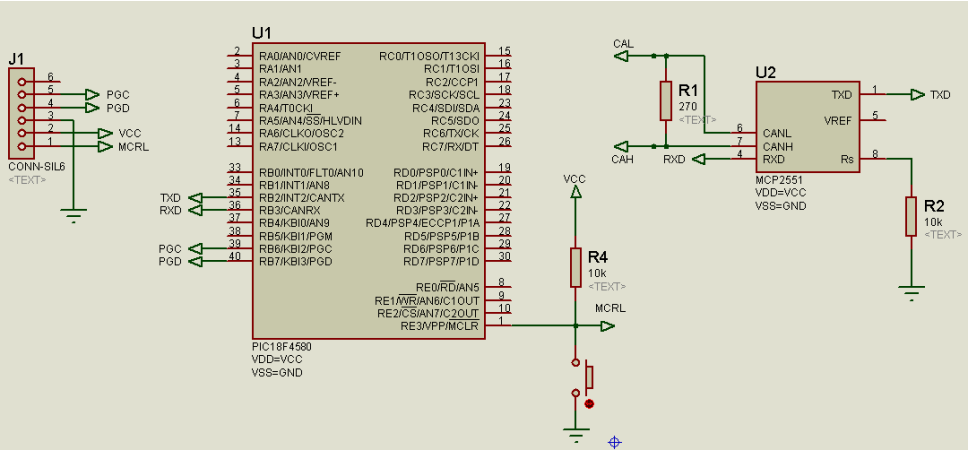


Figura 4.5 Conexiones mínimas entre el microcontrolador PIC18F4580 y el transceiver MCP2551

En los conectores “DB9” J6 y J13, únicamente se tiene conexiones en los pines 2, 7, 3 y 9, en donde:

- Pin 2: CAN-low (CAL).
- Pin 3: Tierra (GND).
- Pin 7: CAN-high (CAH).
- Pin 9: CAN V+ (CAN_V+), alimentación.

J6 y J13 se encuentran etiquetados como CONN-D9M_Entrada y CONN-D9M_Salida respectivamente, lo que significa que el conector con terminación “Entrada” será por el cual las señales provenientes de otro nodo estarán llegando a esté para ser procesadas. De forma complementaria el conector “Salida” es para conectar este nodo a otro para enviar o recibir información sobre el bus. Se colocaron dos conectores DB9, con el propósito de conectar más nodos a la red.

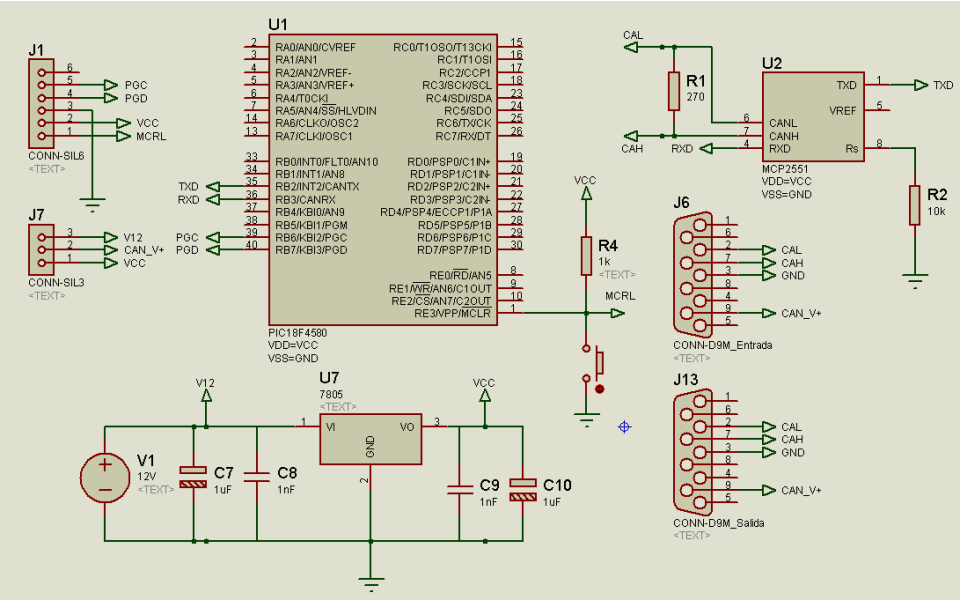


Figura 4.6. Nodo CAN terminales DB9 para conexión al bus.

J1 es un conector de 6 pines tipo macho, para conectar el programador “Pickit 2” de microchip y programar/depurar el microcontrolador (PIC18F4580). J7 es utilizado para seleccionar el voltaje de alimentación del bus “12Vcd o 5 Vcd”. Si el voltaje de alimentación del bus es de 5 V, es necesario cambiar la resistencia R1 por una de 120 Ω.

La red CAN estará constituida por dos nodos idénticos al mostrado en la figura 4.6 NODO 1 y NODO 2, en donde, el NODO 1 mide la temperatura mediante el circuito integrado “LM35” y el NODO 2 simula la presión del aceite del motor mediante el transductor de presión MPX5010.

4.2.4. Acondicionamiento de la señal de los Sensores.

Debido a que los puertos de entrada analógica del microcontrolador convierten las señales a un número digital de 10 u 8 bits y que los voltajes de referencia para una adecuada conversión son 0 a 5 Vcd, es necesario acondicionar las señales de los sensores colocarlos en el rango requerido por el convertidor A/D del microcontrolador.

Acondicionamiento del sensor LM35.

El sensor de temperatura LM35 cuenta con una salida de +10mV/°C, es decir, por cada grado centígrado que se incrementa la temperatura en el sensor tendremos a la salida 10mV, suponiendo que la lectura máxima que se desea medir es de 100 °C, el sensor proporcionará 1V a la salida como máximo por lo que es necesario amplificar la salida del sensor en 5 veces.

Para lograr una ganancia de se requiere conectar a la salida del sensor un circuito amplificador no inversor, el cual debe ser diseñado para entregar una ganancia de 5. El esquema general de un amplificador no inversor se muestra en la figura 4.7.

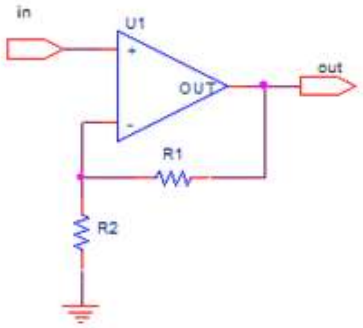


Figura 4.7. Amplificador No Inversor.

La salida de voltaje está dada por la ecuación 4.1

$$V_{out} = V_{in}\left(1 + \frac{R_2}{R_1}\right) \qquad (4.1)$$

Por lo que la ganancia estará dada mediante la ecuación 4.2

$$\Delta V = 1 + \frac{R_1}{R_2} \qquad (4.2)$$

La ganancia requerida es de 5. Por lo tanto, despejando de la ecuación 4.2 para R2, se tiene:

$$R_2 = \frac{R_1}{\Delta V - 1} \qquad (4.3)$$

Al suponer una resistencia $R_1 = 25K\Omega$.

$$R_2 = \frac{25x10^3}{5 - 1}$$

$$R_2 = \frac{25x10^3}{4} = 6.25x10^3\Omega$$

Comercialmente no se encuentran los valores obtenidos, los valores más cercanos son: 6.8kΩ y 27kΩ, lo cual afecta la ganancia previamente calculada, la nueva ganancia será:

$$\Delta V = 1 + \frac{R_1}{R_2}$$

$$\Delta V = 1 + \frac{27x10^3}{6.8x10^3} = 4.97 \approx 5$$

Al comparar la ganancia requerida con la que se obtuvo, se puede observar que estas no varían mucho una respecto de la otra, lo que implica que nuestros cálculos han sido correctos, por lo que el circuito a armar será el mostrado en la figura 4.8.

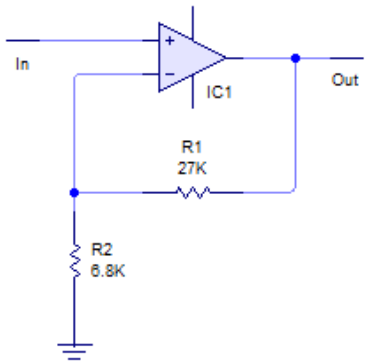


Figura 4.8. Amplificador No-Inversor con ganancia de 4.97

Por lo que el circuito completo para el sensor quedará como el mostrado en la figura 4.9 en la cual se observa que la salida del amplificador va hacia a la entrada RA1/AN1 (Pin 3), que corresponde al canal 1 del convertidor analógico-digital del microcontrolador PIC18F4580 (microcontrolador del circuito "NODO 2" conectado a la red CAN). Además se conectó a la salida del sensor un filtro R-C para eliminar ruidos que puedan afectar la señal de salida del sensor, por ejemplo: fuentes electromagnéticas como relevadores, transmisores de radio, motores, etc.

El amplificador operacional debe ser alimentado con una fuente simétrica ±V.

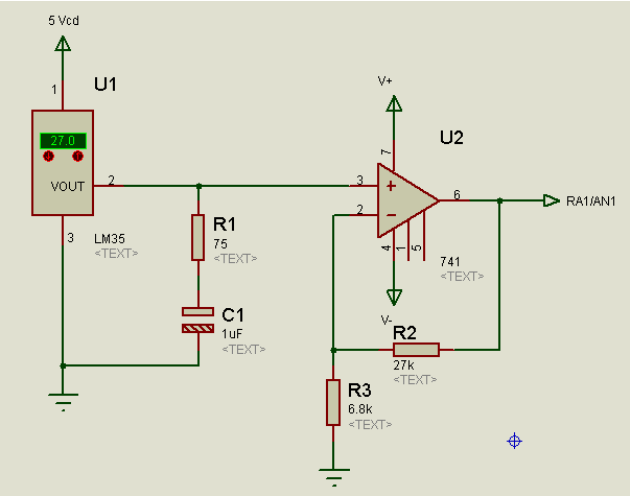


Figura 4.9 Acondicionamiento de la señal del sensor LM35.

De manera similar al sensor de temperatura, es necesario conectar una etapa para acondicionar la señal de salida del sensor MPX5010, es decir, amplificar dicha señal. Este circuito es el mismo que se utilizó para el CI LM35, de igual forma la ganancia será la misma, por lo que no es necesario repetir los cálculos, por lo tanto el circuito final se muestra en la figura 4.10.

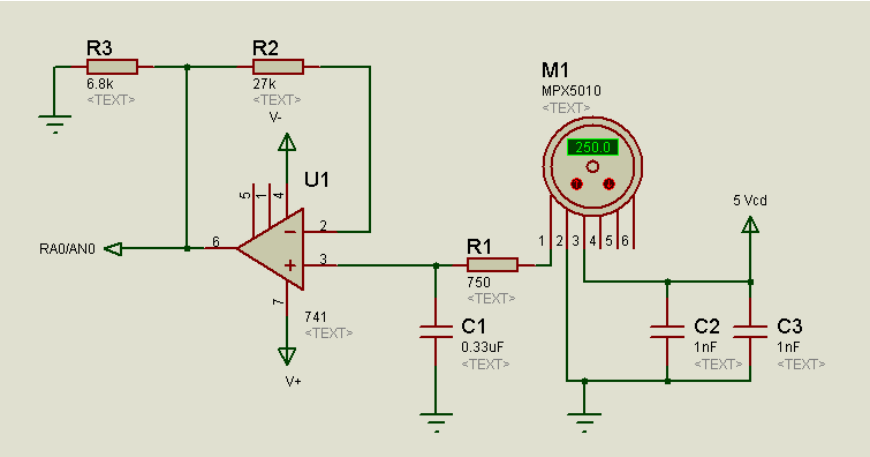


Figura 4.10. Circuito de acondicionamiento para la señal del sensor MPX5010.

La salida del circuito se conecta a la entrada RA0/AN0 (Pin 2), la cual corresponde al canal 0 del convertidor analógico-digital del microcontrolador PIC18F4580 (microcontrolador del circuito "NODO 1" conectado a la red CAN). En la figura 4.9 se observa el filtro (ver capítulo 4 sección 4.2.2) aplicado al transductor para eliminar las señales de ruido que puedan afectar la medición.

Para visualizar las lecturas realizadas por los sensores es necesario conectar un indicador alfanumérico LCD (uno por nodo) del tipo 16x2, los cuales deberán ser conectados al puerto D "PORTD" de cada microcontrolador. En la figura 4.11 se muestra la conexión del indicador alfanumérico y el microcontrolador.

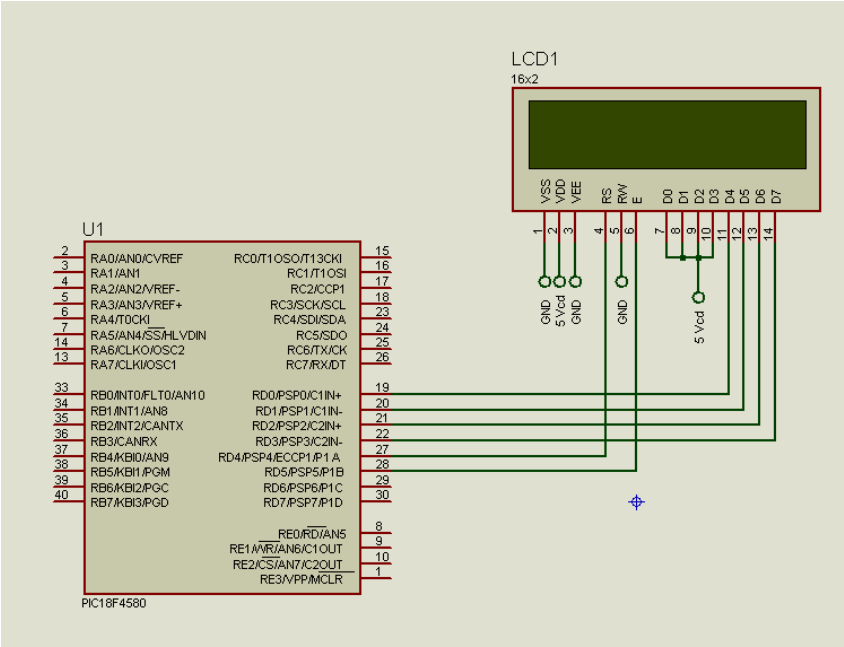


Figura 4.11. Conexión entre el MCU y un LCD 16x2.

4.3. Diseño del Software.

4.3.1. Tiempo Real "MultiTasking"

En un microcontrolador el modo multi-tarea (Multitasking), nos permite ejecutar una, dos, tres o más tareas en tiempos precisos calculados con anterioridad, para ejecutar diferentes rutinas del firmware. La Multi-Tarea requiere que el código se encuentre estructurado y dividido en pequeños fragmentos, los cuales se ejecutan en tareas (Tasks) individuales.

Multitasking es una de las mejores opciones cuando se desea que un microcontrolador o controlador ejecute varias funciones, pero que estas no se ejecuten demasiado rápido. Por ejemplo, si se deseas que todas las funciones o módulos de cierto programa sean ejecutadas máximo 10000 veces por segundo, la multitarea es una de las mejores opciones.

4.3.2. Manejo de Tiempo del Procesador.

Tarde o temprano uno termina descubriendo, que diseñar firmware significa “Manejo de Tiempo del Procesador”. Supóngase que se tiene un microcontrolador operando a una frecuencia interna de 80 MHz, lo que implica que las instrucciones se estarán ejecutando a 20 MHz. Esto significa que una instrucción se ejecuta en 50ns.

Si una instrucción se ejecuta en 50 ns, entonces en un 1 µs (un microsegundo) el microcontrolador ejecuta 20 ciclos de instrucciones, en 1 ms (un milisegundo) este realiza 20,000; y en un segundo ejecutara la asombrosa cantidad de 20,000,000. Sorprendentemente, aunque parezca que hay suficiente tiempo para cada tarea, la mayoría de los programas no trabajan de forma adecuada, debido a colisiones de tiempo entre varias funciones e interrupciones dentro del programa.

La multitarea tiene ventajas respecto al uso de interrupciones, a continuación se muestra un ejemplo: supongamos que se desea leer un valor entero (integer) en dos bytes. Si no se tiene cuidado, la lectura podría realizarse en los escenarios siguientes. Primero, se leerá el byte menos significativo “low byte” del entero. Después, antes de leer el byte más significativo “high byte” una interrupción sería ejecutada y modificaría el valor original del entero. Cuando finalice la interrupción, nuestro programa continuará leyendo el “high byte”, pero el dato ya estará corrompido, en otras palabras será erróneo.

Existen pocas técnicas que se podrían utilizar para evitar el escenario descrito anteriormente, y una de las mejores es la múlti-tarea. El mecanismo utilizado para programar múlti-tarea en un microcontrolador es simple y eficiente, adecuado para controladores industriales (incluso para microcontroladores de 2 kilo bytes de memoria de programa).

El programa del microcontrolador se encontrará encerrado dentro de un WHILE(true)-loop, con el fin de alcanzar la precisión deseada, es necesario programar las interrupciones ocasionadas por temporizadores (timers) y así hacer multitarea. Para la mayoría de las aplicaciones de control dos o máximo tres tareas “tasks” con suficientes, para comprender mejor el concepto “Multitasking” se presenta una estructura con cuatro tareas en la figura 4.12.

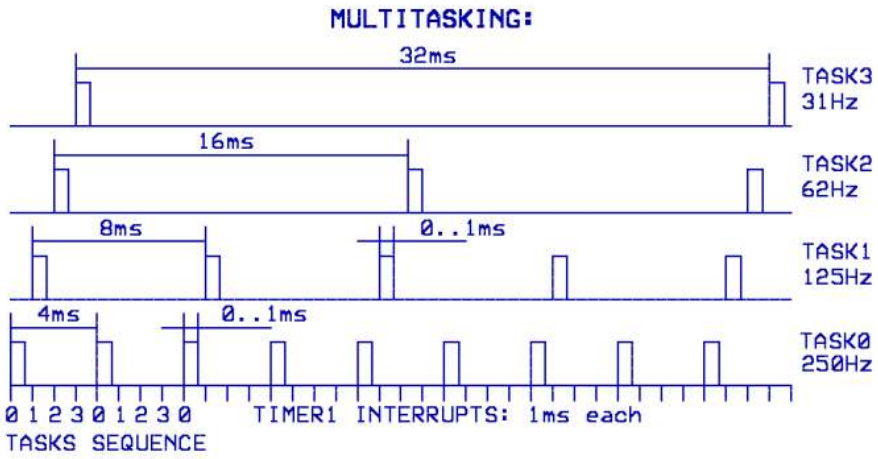


Figura 4.12. Diagrama de tiempo del mecanismo multitasking de 4 tareas.

La figura 4.11 ilustra el mecanismo utilizado para la multi-tarea. Para llevar a cabo la implementación de la multi-tarea es necesario configurar las interrupciones ocasionadas por algún timer del microcontrolador (PIC18F4580 en nuestro caso) a utilizar, la interrupción será generada cada milisegundo (para el ejemplo), enseguida se debe definir intervalos específicos de tiempo para cada una de las cuatro tareas.

- **TASK0,** se ejecutará cada 4 ms, es decir, una frecuencia de 250 Hz $\left(250\frac{veces}{segundo}\right)$.
- **TASK1,** se ejecutará cada 8 ms, es decir, una frecuencia de 125 Hz $\left(125\frac{veces}{segundo}\right)$.
- **TASK2,** se ejecutará cada 16 ms, es decir, una frecuencia de 62.5 Hz $\left(62.5\frac{veces}{segundo}\right)$.
- **TASK3,** se ejecutará cada 32 ms, es decir, una frecuencia de 31.25 Hz $\left(31.25\frac{veces}{segundo}\right)$.

El tiempo para cada tarea es solo para ejemplificar. Además observemos que, cada tarea puede ser diseñada para ejecutarse a una frecuencia de 250 Hz, o cualesquiera que se desee.

Como se mencionó, la aplicación "Multitasking" estará utilizando un timer del microcontrolador, como temporizador de la multitarea, y este generará interrupciones cada milisegundo (o cada 20 000 ciclos de instrucciones). En la figura 2.36, se observa que cada

tarea es ejecutada en un tiempo específico, independientemente de las otras tareas. De esta forma cada tarea tendrá cerca de un milisegundo para realizar todas las funciones asignadas a está. Si se desea se puede incrementar o decrementar el tiempo de ejecución de las tareas.

Para la configuración del protocolo J1939, en el cual estará montada la red, se utilizó la biblioteca J1939 de microchip ya que esta tiene las funciones de configuración de la mayoría de los aspectos importantes del protocolo. Esta biblioteca nos ofrece el protocolo de comunicación J1939 para cada Controlador de la Aplicación (CA) conectado en la red.

La principal función de la biblioteca es organizar todos los mensajes recibidos por el CA y transmitir los mensajes que algún controlador desee enviar. Para iniciar la biblioteca es necesario configurar e incluir en el proyecto los archivos "j1939.c", "j1939.h" y "j1939.def".

- J1939.c debe ser compilado y enlazado con el archivo principal del proyecto.
- El archivo de cabecera J1939.h debe ser incluido en cualquier archivo fuente que utilice las rutinas de la biblioteca o cualquier definición del protocolo J1939.
- J1939.def debe encontrarse localizado en el mismo folder del proyecto. Ya que en él se configuran el Nombre, Dirección, Prioridad, etc. de cada nodo o CA.

En el archivo J1939.h, se debe agregar la definición del microcontrolador PIC18F4580, que es el utilizado en la red; para esto agregar la definición de dicho controlador, localizar las siguientes líneas en el archivo de cabecera antes mencionado y agregar la definición del microcontrolador.

```
#if   defined(__18F248)
        #define ECAN_IS_CAN_MODULE
#elif defined(__18F258)
        #define ECAN_IS_CAN_MODULE
#elif defined(__18F448)
        #define ECAN_IS_CAN_MODULE
#elif defined(__18F458)
        #define ECAN_IS_CAN_MODULE
#elif defined(__18F4580)
        #define ECAN_IS_CAN_MODULE
#endif
```

En color rojo se muestra como se debe agregar la definición del microcontrolador. En el mismo archivo (J1939.h) configurar el modo de funcionamiento, en el cual se desea que la red se encuentre configurada, para nuestro caso la red se configura en "Modo FIFO". A continuación se presenta el segmento de código para la configuración en "Modo FIFO":

```
#ifdef ECAN_IS_CAN_MODULE
        #undef ECAN_LEGACY_MODE
        #define ECAN_LEGACY_MODE  J1939_FALSE
#endif
```

Es necesaria esta parte del código debido a que es utilizada por el archivo J1939.c para configurar el modo en el cual opera el controlador CAN del microcontrolador. El archivo (J1939.h) es el mismo para ambos nodos de la red (Nodo 1 y Nodo 2).

Otro archivo a modificar es el J1939.def, en el cual solo es necesario cambiar la línea:

- #define J1939_STARTING_ADDRESS 128, para el Nodo 1.
- #define J1939_STARTING_ADDRESS 129, para el Nodo 2.

Hasta ahora se tiene configurada la biblioteca J1939 de Microchip para que está funcione con el microcontrolador PIC18F4580.

Como se mencionó al inicio de este capítulo, los nodos estarán monitoreando temperatura y presión (nodo 1 y nodo 2, respectivamente), por lo cual es necesario configurar el convertidor analógico-digital de cada microcontrolador, los cuales deben tener la misma configuración. En el convertidor se configura de la siguiente forma:

- Voltaje de referencia 1 = Vref -.
- Voltaje de referencia 2 = Vref+.
- AN0 – AN4: como convertidores analógico-digitales.
- Resultado de la conversión: Justificada a la derecha (10 bits).
- Tiempo de conversión: Fosc/32.

La configuración descrita anteriormente se lleva a cabo configurando los registros ADCON1 y ADCON2 del microcontrolador, estos registros quedan de la siguiente manera:

- ADCON1 = 0b00001011 (binario) o 0x0B (hexadecimal).
- ADCON2 = 0b10000010 (binario) o 0x12 (hexadecimal).

La configuración completa de los convertidores se encuentra en el archivo "Inicia_AD.c" (ver apéndice C), el cual debe ser incluido en el archivo fuente que utiliza las rutinas del convertidor.

Después se configura el timer 1, se habilita el vector de interrupción de alta prioridad para la programación multitarea del microcontrolador. El timer 1 se encuentra configurado para solicitar una interrupción cada 1.00 ms o 16 000 ciclos de reloj (Frecuencia de oscilación, Fosc = 16 MHz del microcontrolador). Esta configuración debe encontrarse al inicio del archivo fuente.

La función del vector de interrupción alto (ver snippet 4.1) debe colocarse al inicio del archivo fuente como se muestra en el segmento de código mostrado a continuación. En el que se observa como el vector es posicionado en la localidad de memoria número 8.

```
void high_isr (void);
#pragma code _HIGH_INTERRUPT_VECTOR = 0x00008          //modifica el vector de
interrupciones alto
void _high_ISR (void)
{
    _asm GOTO high_isr _endasm;
}
#pragma code
/*****************High priority ISR *************************/
#pragma interrupt high_isr
```

Snippet 4.1. Configuración del vector de interrupción de alta prioridad.

Además se debe configurar los registros INTCON, RCON, PIE1 e IPR1.

- INTCON: Habilita las interrupciones de alta prioridad y las ocasionadas por los puertos del microcontrolador.
- RCON: Habilita los niveles de interrupción.
- PIE1: Interrupción ocasionada por el timer 1.
- IPR1: Interrupción del timer 1 de alta prioridad.

Debido a que son pocos bits, el snippet 4.2 muestra como habilitar las interrupciones así como el orden que se debe seguir.

```
INTCONbits.PEIE = 1;          // Habilita las interrupciones de los puertos.
INTCONbits.GIEH = 1;          // Interrupciones globales de alta prioridad.
RCONbits.IPEN = 1;            // Habilita los niveles de interrupción.
PIE1bits.TMR1IE = 1;          // Interrupción por Timer 1.
IPR1bits.TMR1IP = 1;          // Interrupción por Timer 1 de alta prioridad.
```

Snippet 4.2. Orden a seguir para habilitación interrupciones.

A continuación se configura el timer 1, para operación de 16 bits e interrupción cada milisegundo, para saber obtener el valor a cargar en los registros TMR1H y TMR1L, se utiliza la ecuación 4.4. Antes de colocar cualquier valor en estos registros, se debe configurar el registro T1CON, sin arrancar el timer 1, como muestra a continuación:

```
T1CON = 0b10000000;           //Timer 1 a 16 bits con prescaler de 1.
```

$$Valor_{cargar} = 65535 - \left(\frac{Temporización}{4 * T_{osc} * Rango_{Divisor}}\right) \tag{4.4}$$

$$T_{osc} = \frac{1}{F_{osc}} \tag{4.5}$$

$Rango_{Divisor}$ es el prescaler a utilizar para el timer 1, en este caso se utiliza un prescaler de 1 a una frecuencia de oscilación de 16MHz; sustituyendo valores en la ecuación 4.4 tenemos que:

$$Valor_{cargar} = 65535 - \left(\frac{1x10^{-3}}{4 * \left(\frac{1}{16x10^{6}}\right)}\right) = 65\,535 - 4\,000 = 61\,535$$

Es necesario cargar este valor en la parte alta y parte baja del timer 1 (TMR1H y TMR1L), es conveniente convertir este valor a un valor hexadecimal, el cual es:

$$Valor_{cargar} = 65\,535_{decimal} = \boldsymbol{F05F_{hexadecimal}}$$

El valor hexadecimal se divide en dos partes, parte alta y parte baja, la parte alta se carga en el registro TMR1H y la parte baja en TMR1L.

```
TMR1H = 0xF0;
TMR1L = 0x5F;
```

La interrupción de alta prioridad ejecuta la función "high_isr" (ver snippet 4.3), en cada desbordamiento del timer 1, es decir, cada milisegundo. Si la bandera del timer 1 (TMR1IF) se encuentra en alto, se entra a esta función (high_isr) y se verifica que tarea debe ejecutarse (modo multitarea), la decisión es tomada mediante el contador "ISR_counter". Para decidir que tarea se ejecuta se realiza una operación lógica AND con el contador "ISR_Counter" y un número binario asignado, que depende del número de veces que se desea que una tarea sea ejecutada.

Al entrar a la función se cargan los valores calculados para el timer 1 nuevamente y se reinicia la bandera del timer (TMR1IF = 0), para entrar a la función y al concluir el periodo de 1 milisegundo continuar el conteo de las tareas.

```
void high_isr(void)
{
    if(PIR1bits.TMR1IF)
    {
            TMR1H = 0xF0;
            TMR1L = 0x6F;

            PIR1bits.TMR1IF = 0;

            if((ISR_counter & 0b00000011) == 0x00)          // Task0 se repite 8 veces
            (Cada 4 ms)
                task0 = ACT;
            if((ISR_counter & 0b00000111) == 0x01)          // Task1 se repite 4 veces
            (Cada 8 ms)
                task1 = ACT;
            if((ISR_counter & 0b00001111) == 0x02)          // Task2 se repite 2 veces
            (Cada 16 ms)
                task2 = ACT;
            if((ISR_counter & 0b00011111) == 0x03)          // Task3 se repite 1 vez
```

```
            (Cada 32 ms)
                task3 = ACT;

            ISR_counter++;
            if(ISR_counter == 32)
                ISR_counter = 0;
    }
}
```

Snippet 4.3. Función "high_isr" para modo multitarea.

En el segmento de código "Función high_isr", se observa que el programa cuenta con cuatro tareas diferentes las cuales tiene diferente prioridad y se ejecutan cada cierto periodo. La tarea que se ejecuta más número de veces es la de mayor prioridad, siendo la de menor prioridad la ejecutada en menos ocasiones.

Una vez configurada la biblioteca J1939, el convertidor Analógico-Digital, las interrupciones y declarado el número de tareas y la frecuencia a que se ejecutaran, iniciaremos con la programación del nodo 1 en la función "main".

4.3.3. Inicio del Programa "Estructura Main".

En la estructura main, la función "InitMicro" es la primera que se ejecuta, la cual inicializa los puertos a utilizar (ver apéndice A, bloque de programa A.2), selecciona la frecuencia de oscilación del microcontrolador, para nuestro caso está frecuencia se basa en el oscilador interno del microcontrolador, configurada mediante el registro OSC (Fosc = 16 MHz, OSC = IRCIO7) e inicializa las variables y tareas que se utilizan en el programa; después se inicializan los convertidores analógico-digitales, función "init_AD" (ver apéndice A, bloque de programa A.3); enseguida se inicializa la función llamada "Timer_Ini" que se encarga de inicializar el timer 1 para realizar interrupciones cada milisegundo (ver apéndice A, bloque de programa A.4), estas interrupciones son las encargadas de ejecutar las tareas del microcontrolador (task0, task1, task2 y task3); a continuación se inicializa el display tipo LCD (función "LCD_Init", ver apéndice A, bloque de programa A.5), para visualizar los datos obtenidos por los convertidores analógico-digitales; finalmente se inicializa la biblioteca J1939 mediante la función "J1939_Initialization" (ver apéndice A, bloque de programa A.1), esta función es la encargada de configurar e inicializar las variables globales de la biblioteca "J1939", el modulo CAN y las interrupciones ocasionadas por los mensajes (enviados y recibidos).

La función “J1939_Initialization” es la encargada de configurar el nombre y dirección del nodo (definidos en el archivo “J1939.def”), si el parámetro de entrada de la función es un “True” booleano (ejemplo “J1939_Initialization(True)”). Los registros configurados son los mostrados en la tabla 4.1. Se recomienda que los registros sean configurados en el orden mostrado en la tabla 4.1, ya que es necesario primeramente colocar el modulo CAN en Modo configuración para poder modificar los registros involucrados en la comunicación CAN.

Registro	Valor (Hexadecimal)	Significado
CANCON	0x80	Modo Configuración
ECANCON	0x90	Modo FIFO Mejorado
BSEL0	0xE0	Buffers 5 a 3 (B5TXEN: B3TXEN:) en modo de transmisión, buffers 2 a 0 (B2TXEN:B0TXEN)en modo recepción
MSEL0	0x5C	Asigna Mascara 0 al Filtro 0 y Mascara 1 a Filtros 2 y 3
BRGCON1	0x01	SJW = 1 TQ Baud Rate = $T_Q = \frac{4}{F_{OSC}}$
BRGCON2	0xBF	SEG2PHTS = 1 SAM = 0 SEG1PH = 111 (8 TQ) PRSEG = 111 (8 TQ)
BRGCON3	0x87	WAKDIS = 1 WAKFIL = 0 SEG2PH = 111 (8 TQ)
CANCON	0x00	Modo Normal
ECANCON	0xA5	Modo FIFO Mejorado Buffer 2 de transmisión

Tabla 4.1. Registros Configurados por la función J1939_Initialization.

De la tabla 4.1, se tiene el registro BRG1CON = 0x01, diseñado para un baud-rate de 500 KHz con SJW = 1, mediante la ecuación 3.4 se tiene que:

$$T_{Q(\mu s)} = \frac{2 * (BRP + 1)}{F_{OSC}} = \frac{2 * (1 + 1)}{F_{OSC}}$$

$$T_{Q(\mu s)} = \frac{4}{16x10^6} = \mathbf{0.250\ \mu S}$$

Por lo que nuestro *Time Quanta* es igual a 0.250 µS y el tiempo de bit dado por la ecuación 4.1, debe ser 8 veces el Time Quanta (ver sección 3.2.7. Configuración del Baud Rate).

$$T_{BIT(\mu s)} = T_{Q(\mu s)} * Numero\ de\ T_Q\ por\ intervalo\ de\ bit \quad (4.1)$$

Entonces se tiene que:

$$T_{BIT} = 8 * T_Q$$

$$T_{BIT} = 8 * 250x10^{-6} = 2x10^{-6}$$

$$T_{BIT} = 2\ \mu S$$

La ecuación 4.2 nos dice que la velocidad nominal de bits está dada por:

$$Nominal\ Bit\ Rate = \frac{1}{T_{BIT}} \quad (4.2)$$

$$Nominal\ Bit\ Time = \frac{1}{2x10^{-6}}$$

$$\boldsymbol{Nominal\ Bit\ Time = 500\ KHz}$$

Por lo tanto el registro BRGCON1 se encuentra configurado correctamente para la velocidad deseada y con un SJW = 1. Una vez inicializado el protocolo J1939, se verifica que la dirección asignada al nodo (nodo 1 = 128, nodo 2 = 129) se encuentre disponible, observando el estado de la bandera "J1939_Flags.WaitingForAddressClaimContention", el cual debe ser igual a cero.

- **"J1939_Flags.WaitingForAddressClaimContention" = 0**.

Si la bandera es igual a cero, significa que la dirección se encuentra disponible, entonces es necesario espera por lo menos cinco milisegundos (5 ms), para sincronizar el nodo con la red.

En el segmento de código mostrado a continuación (ver snippet 4.4 “Inicialización y Configuración de la red CAN”) se muestran las funciones realizadas para la configuración del protocolo J1939 y la inicialización de la red CAN, las cuales deben ser llamadas antes de entrar al bucle while(1) en el que se encuentra el control e instrucciones llevadas a cabo por el programa.

```
void main (void)
{
    InitMicro();
    init_AD();
    Timer_Ini();
    LCD_Init();

/************** Inicializamos Protocolo J1939 ********************/

    J1939_Initialization( TRUE );

    // Wait for address contention to time out
    while (J1939_Flags.WaitingForAddressClaimContention)
            J1939_Poll(5);

    // Now we know our address should be good, so start checking for
    // messages and switches.

    while(1)
```

Snippet 4.4. Inicialización y configuración de la red CAN.

4.3.4. Control del Programa “Ciclo while(1)”.

Dentro del ciclo while(1), se encuentran las tareas definidas para el sistema multitarea que el programa ejecuta de forma constante:

- Task0: se repite cada 0.5 milisegundos.

- Task1: se repite cada 2 milisegundos.
- Task2: se repite cada 8 milisegundos.
- Task3: se repite cada 32 milisegundos.

En la tarea 0 “Task0” (snippet 4.5 “Lectura del estado del interruptor”), se verifica el estado actual del interruptor que indica si se ha realizado una petición de información desde el otro nodo conectado a la red. A continuación se muestra el código para la lectura del estado del interruptor (tanto para el Nodo 1 como para el Nodo 2).

```
if(task0 == ACT)
{       task0 = DESACT;
        CurrentSwitch = PORTBbits.RB5;
}
```

Snippet 4.5. “Task 0” lectura del estado del interruptor.

En la tarea 1 (snippet 4.6 “Construcción y transmisión del mensaje”) “Task1” (tanto para el nodo 1 como para el nodo 2) se verifica el estado del interruptor (variable “CurrentSwitch”), si este ha cambiado de estado lógico (0 a 1 o 1 a 0), el nodo (Nodo 1 o Nodo 2) realiza alguno de los siguientes casos:

- Sí la variable “CurrentSwitch = 0”, el PDU del mensaje es igual a la variable Read_Data (Read_Data = 92), (ver apéndice A, bloque de programa A.6). La variable Read_Data = 92, le ordena al microcontrolador que obtenga la información de los convertidores analógico-digitales del microcontrolador.
- Sí la variable “CurrentSwitch = 1”, el PDU del mensaje es igual a la variable Read_off (Read_off = 94), (ver apéndice A, bloque de programa A.6). La variable Read_off, es una instrucción nula para el microcontrolador, es decir, si el microcontrolador ve esta variable, enviara desplegar la leyenda “CAN bus” en el display LCD.

Después de verificar el estado del interruptor y realizar alguna de las funciones descritas en el párrafo anterior, el microcontrolador estructura el mensaje a ser transmitido sobre la red, independientemente de la función llevada a cabo por el microcontrolador. Una vez estructurado el mensaje se actualiza el valor lógico del interruptor. Ver segmento de código “Task 1”.

```
        if(task1 == ACT)
        {
                task1 = DESACT;

/*************Instrucciones Realizadas Nodo Transmisor ******************/

                if (LastSwitch != CurrentSwitch)
                {
                        Msg.DataPage  = 0;
                        Msg.Priority = J1939_CONTROL_PRIORITY;
                        Msg.DestinationAddress = NODE1;
                        Msg.DataLength = 1;
                        Msg.Data[0] = press;

                        if (CurrentSwitch == 0)
                                Msg.PDUFormat = Read_Data;
                        else
                                Msg.PDUFormat = Read_off;

                  while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS)

                        LastSwitch = CurrentSwitch;
                }
        }
```

Snippet 4.6. "Task 1." Construcción y transmisión del mensaje.

En la tarea 2 (snippet 4.7 "Recepción, decodificación y despliegue del mensaje") "Task2", el nodo receptor decodifica el mensaje transmitido por el nodo (nodo transmisor) que recibió una petición de datos (Nodo 1 o Nodo 2), si el PDU es igual a Read_Data, en cualquiera de los nodos, un LED indicador es encendido, indicando que se ha recibido el mensaje satisfactoriamente, el cual podría contener la temperatura o la presión (Nodo 1 o Nodo 2, respectivamente), dato desplegado por un display LCD; en caso de que el PDU sea igual a Read_off, el LED indicador se apagará y el display LCD mostrara la leyenda "CAN bus". Ver segmento de código "Task 2".

```
            if(task2 == ACT)
            {
                  task2 = DESACT;

/**************** Instrucciones Realizadas Nodo Receptor ******************/

                  while (RXQueueCount > 0)
                  {
                        J1939_DequeueMessage( &Msg );

                        if (Msg.PDUFormat == Read_Data)
                        {
                              LATCbits.LATC1 = 1;
                              LM35 = Msg.Data[0];
                              if(LM35 == 0)
                              {
                                    LCD_Command(0x80);
                                    LCD_Str("Temperatura");
                                    sprintf(lectura,"%3d", (LM35));
                                    LCD_Command(0xC0);
                                    LCD_puts(lectura);
                                    LCD_Command(0xC4);
                                    LCD_Str("Grados");
                              }

                              else
                              {
                                    LCD_Command(0x80);
                                    LCD_Str("Temperatura");
                                    sprintf(lectura,"%3d", ((LM35*100)/255)+10);
                                    LCD_Command(0xC0);
                                    LCD_puts(lectura);
                                    LCD_Command(0xC4);
                                    LCD_Str("Grados");
                              }
                  }
                        else if (Msg.PDUFormat == Read_off)
```

```
                    {
                            LCD_Command(0x80);
                            LCD_Str("CAN bus      ");
                            LCD_Command(0xC0);
                            LCD_Str("             ");
                            LATCbits.LATC1 = 0;
                    }
            }
    }
```

Snippet 4.7. "Task 2". Recepción, decodificación y despliegue del mensaje.

Finalmente en la tarea 3 (snippet 4.8 "Lectura del convertidor analógico-digital")"Task 3", se realiza la lectura del convertidor analógico-digital (canal 0 o canal 1, para el nodo 1 y para el nodo 2 respectivamente), esta lectura es la que se transmite si el estado del interruptor es igual a cero. Ver segmento de código "Task3". Esta tarea realiza la lectura del canal 0 del microcontrolador como ejemplo.

```
    If(task3 == ACT)
    {
            task3 = DESACT;
            press = conv_canal(0);
            press = ((unsigned char)press << 8);
    }
```

Snippet 4.8. "Task 3". Lectura del convertidor analógico-digital.

En el segmento de código "Task 1", se observa que el mensaje es estructurado como se muestra a continuación:

- **Msg.Datapage = 0:** Nos dice que el mensaje será enviado en una sola página de datos, es decir, solo se transmitirán 8 bits (8 bits es igual a una página o 1 byte de datos).
- **Msg.Priority = J1939_Control_Priority:** Da la prioridad del mensaje, la cual es controlada por la biblioteca J1939 y se encuentra definida en el archivo J1939.def (prioridad 2 para el nodo 1 y prioridad 3 para el nodo 2).
- **Msg.DestinationAdress = Node1:** Dirección de destino del mensaje (Node1 = 129 y Node0 =128, direcciones elegidas por el diseñador de la red).

- **Msg.DataLength = 1:** Longitud del mensaje, 1 significa que el mensaje tiene una longitud de 8 bits (1 byte).
- **Msg.Data[0] = press:** Envía el dato (longitud del dato igual a 8 bits “un byte”) obtenido de los convertidores analógico-digitales del microcontrolador, para el nodo 2 se envía la presión (nodo 1 transmite la temperatura) obtenida mediante el sensor MPX5010.

Capítulo 5

Pruebas y Resultados

5.1. Validación de los periodos (Mplab SIM).

Mediante la herramienta MPLAB SIM (ver figura 5.1) del software Mplab de Microchip®, es posible comprobar que el tiempo de interrupción programado (ver sección 4.3.2 Manejo de Tiempo del Procesador) en el microcontrolador es efectivamente un milisegundo. Para verificar este tiempo se abre el proyecto en el cual se programaron estas interrupciones, una vez abierto el proyecto (Node0MultTask.mcp o Node1MultTask.mcp), se localiza la función "high_isr", en la cual se encuentra programado el tiempo de interrupción de un milisegundo y se coloca un break point al inicio de esta función para verificar que nuestro programa entra a la función cada milisegundo (ver figura 5.2).

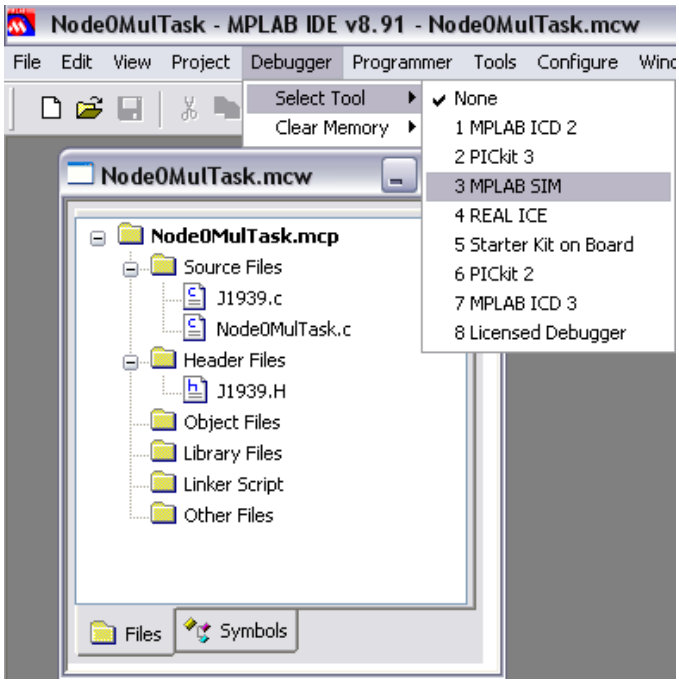


Figura 5.1. Herramienta MLAB SIM para simulación de las interrupciones.

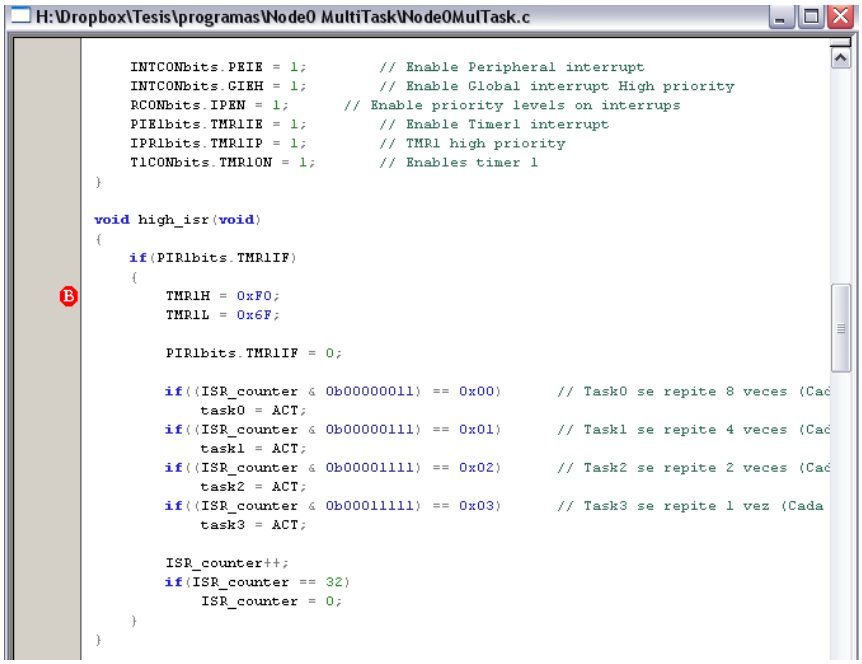


Figura 5.2. Colocación del break point en la función “high_isr”.

Enseguida configuramos las opciones del simulador en Debugger\Settings, una vez realizado lo anterior nos aparecerá una ventana como la mostrada en la figura 5.3, ahí ubicar la opción “Osc/Trace” y después “Processor Frecuency”, en la opción Processor Frecuency escribir 16 y en la opción “Units” escoger Mhz ver figura 5.3.

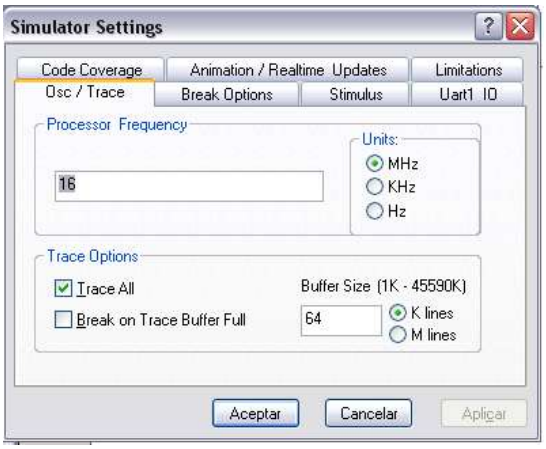


Figura 5.3. Configuración del Simulador.

Al tener configurado el simulador abrir un “Stopwatch” (Debugger\Stopwatch) para observar el tiempo que le toma al microcontrolador entrar a la función “high_isr”. La figura 5.4 muestra el Stopwatch.

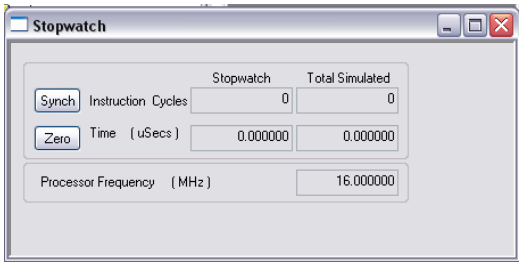


Figura 5.4. Stopwatch.

Una vez abierto el Stopwatch y configurado el simulador se corre el programa con el botón “run” ubicado en el a barra de herramientas figura 5.5.

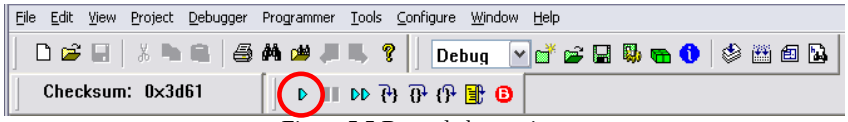


Figura 5.5 Barra de herramientas.

Al presionar el botón “run” observamos como la lectura (Time) del Stopwatch cambia a mSecs y en la función “high_isr” aparece una flecha; esta flecha nos indica que el programa ha entrado a la función y el Stopwatch nos dice el tiempo que le tomo al microcontrolador entrar a la función.

En la figura 5.6 se observa que al microcontrolador le toma un milisegundo entrar a la función “high_isr”. Por lo que los cálculos realizados en la sección 4.3.2 son correctos.

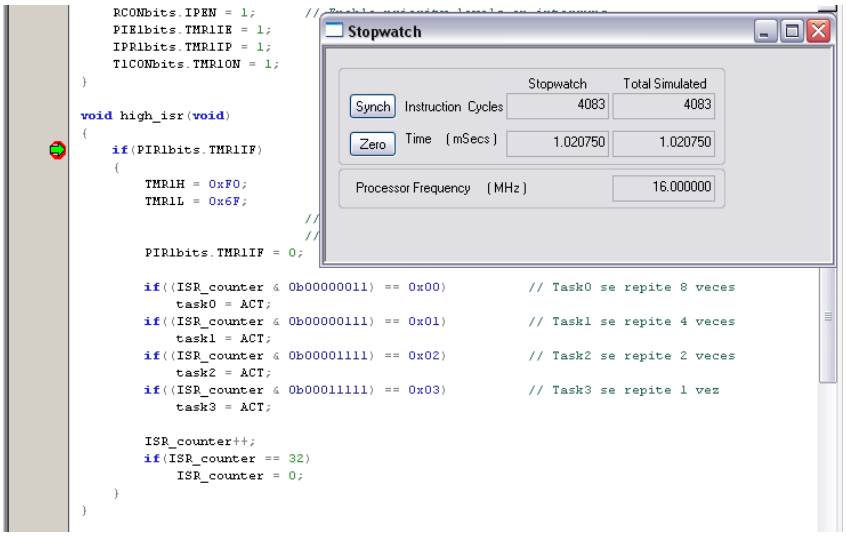


Figura 5.6. Lectura de tiempo realizada por el Stopwatch.

5.2. Pruebas en depuración (PICkit 2).

Mediante la depuración se corrigen e identifican los posibles errores que el programa pueda tener, ya que este es quemado (grabado) en el microcontrolador y se observa el funcionamiento del código línea a línea. Para nuestro caso utilizaremos la depuración para verificar la correcta configuración del módulo ECAN del microcontrolador.

De manera similar a la simulación (ver sección 5.1 Simulador Mplab) abrir la aplicación para realizar la depuración mediante del software Mplab de Microchip®, para correr esta herramienta dirigirse a la barra de herramientas después en Debugger\Select Tool\PICkit 2 (es necesario tener conectado el PICkit 2 al microcontrolador) como se muestra en la figura 5.7. Al abrir la aplicación PICkit 2 el software se conecta al microcontrolador a través del dispositivo PICkit 2, si el software ha conectado satisfactoriamente el dispositivo PICkit2 este despliega una pantalla (figura 5.8) con el texto mostrado líneas a bajo, este texto indica que el microcontrolador encontrado es el modelo PIC18F4550 y que el PICkit está listo para utilizarse.

PIC18F4580 found (Rev 0x1)

PICkit 2 Ready

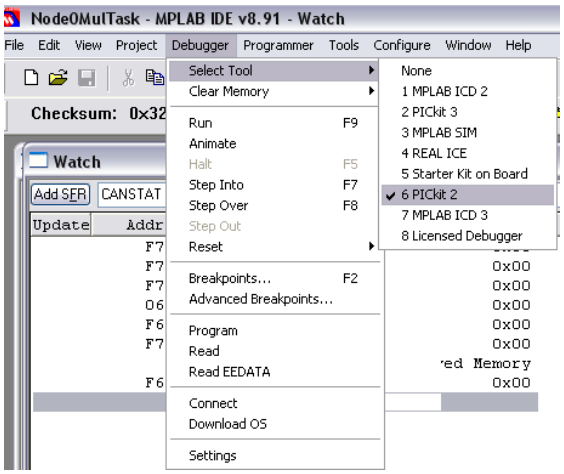


Figura 5.7. Aplicación PICkit 2 para depuración.

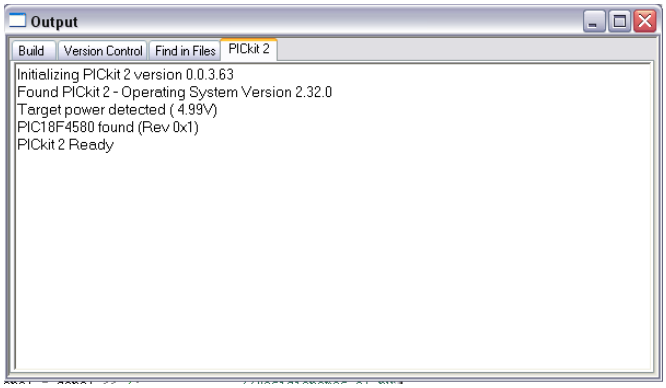


Figura 5.8. Pantalla de detección del PICkit 2.

A continuación se abre un *watch* (figura 5.9) para visualizar los registros del módulo ECAN que son configurados en la función "J1939_Initialization" (sección 4.3.3 Inicio del Programa "Estructura Main"). En el *watch* (figura 5.9) se agregan los registros: BRGCON1, BRGCON2, BRGCON2, CANCON, ECANCON, BSEL0 Y CANSTAT (Tabla 3.1), así como la variable J1939_Address, la cual contiene la dirección del nodo (Nodo 1 = 128 y Nodo 2 = 129).

Después de abrir el watch y agregar los registros y la variable a monitorear, construir el proyecto, haciendo clic en el icono "build all" de la barra de herramientas, al presionar este icono el software compila el proyecto arrojando como resultado la ventana mostrada en la figura 5.10. Cuando el software termine de compilar quemar (grabar) el programa en el

microcontrolador dando clic en el icono *"Program the target device"* ubicado en la barra de herramientas, este proceso tarda aproximadamente unos 2 segundos, una vez que el software ha grabado el programa en el microcontrolador el software despliega la ventana de *"output"* (figura 5.11).

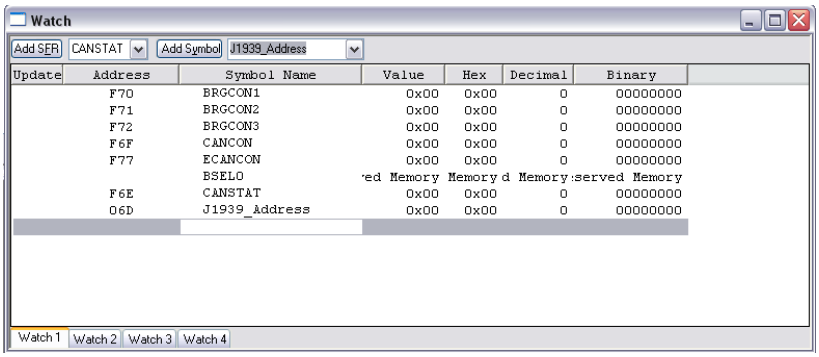


Figura 5.9. Registros y variable agregados al *watch.*

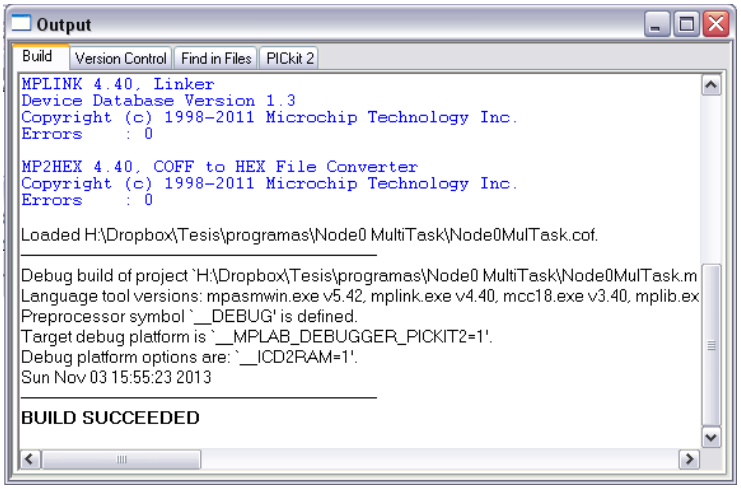


Figura 5.10. Compilación del proyecto.

Dentro de la estructura "main" del programa principal ubicar la función encargada de configurar el módulo ECAN y colocar un break point (ver sección 5.1 Simulador Mplab) como se muestra en la figura 5.12. Para nuestro caso colocar el break point en la función "J1939_Initialization(TRUE)". Enseguida correr nuestro programa, el cual parará en la función donde hemos colocado el break point (figura 5.13.), desplegando en la ventana *"output"* el texto: *Target Halted*, es decir, microcontrolador en espera o detenido. A

continuación entramos a la función haciendo clic en el icono *"Step Into"*  localizado en la barra de herramientas del software. Al entrar a la función se abre una ventana con el código de la función (figura 5.14), por otro lado el registro CANSTAT cambia su valor a 0x80h (módulo CAN en modo Configuración) y la variable J1939_Address también cambia a 0x80h (128 decimal; dirección que toma el nodo dentro de la red) como se observa en la figura 5.15.

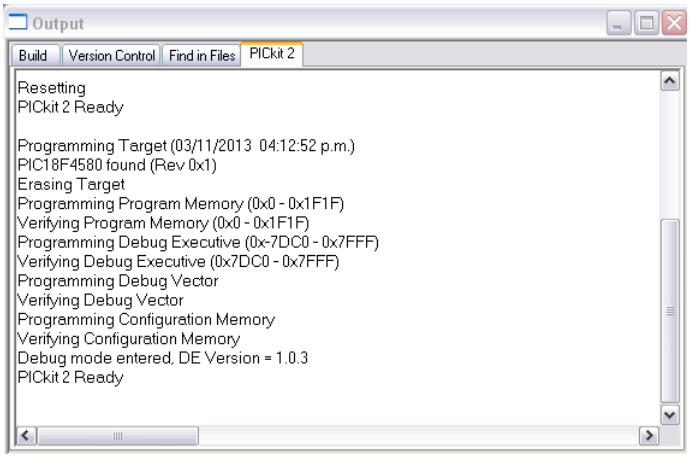


Figura 5.11. Grabado del programa en el microcontrolador para depuración.

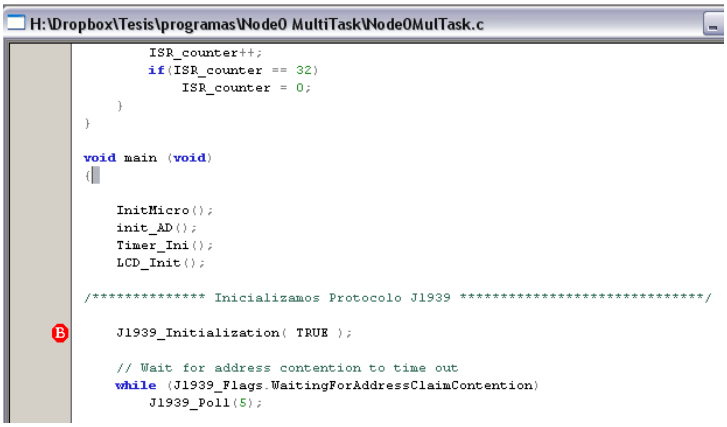


Figura 5.12. Break point necesario para entrar a la función J1939_Initialization.

Continuar haciendo clic en el icono *"Step Into"* hasta llegar a la sub-función **"SetECANMode(ECAN_CONFIG_MODE)",** localizada dentro del código, volver a dar clic en *Step Into* para dirigirse a la función (figura 5.16). Esta función coloca el módulo

CAN en modo configuración dando clic en *Step Into* hasta que el registro CANCON cambie su valor (CANCON = CANSTAT 0x80h) como se muestra en la figura 5.17.

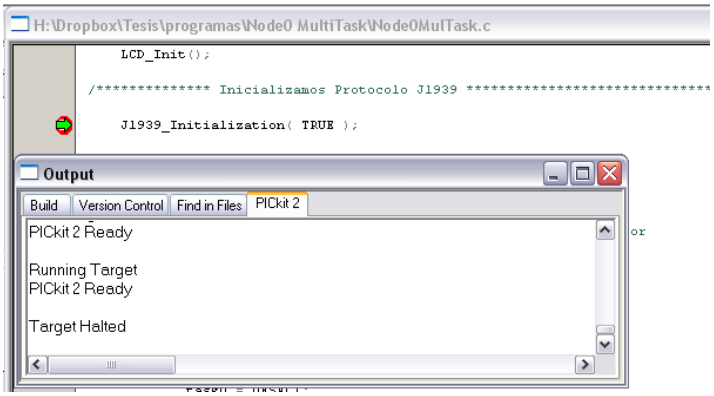


Figura 5.13. Interrupción del programa en la función "J1939_Initialization".

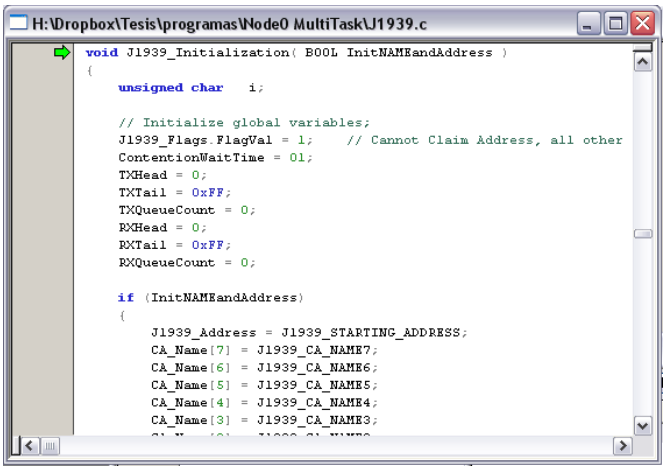


Figura 5.14. Ventana emergente con el código de la de función J1939_Initialization.

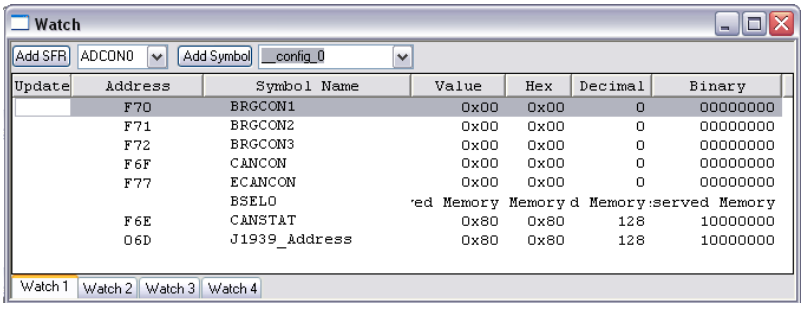
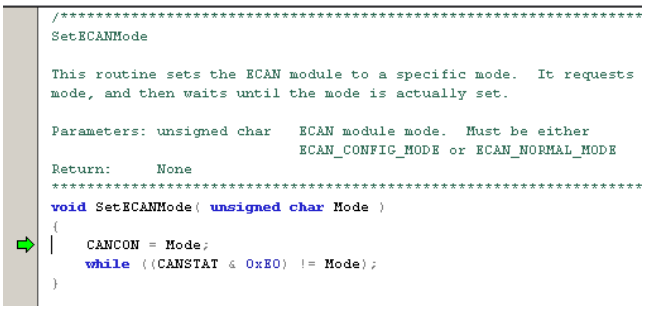


Figura 5.15. Cambio en el valor del registro CANSTAT y la variable J1939_Address.

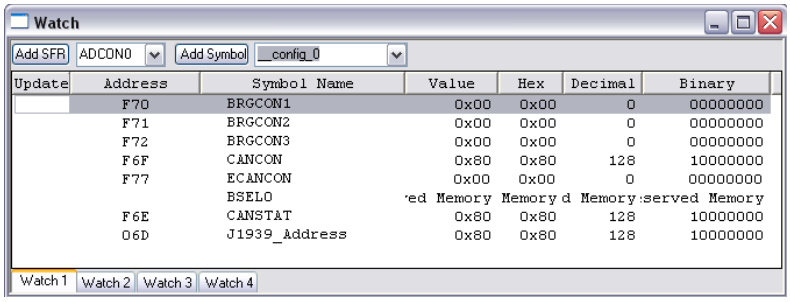


```
/*********************************************************************
SetECANMode

This routine sets the ECAN module to a specific mode.  It requests
mode, and then waits until the mode is actually set.

Parameters: unsigned char   ECAN module mode.  Must be either
                            ECAN_CONFIG_MODE or ECAN_NORMAL_MODE
Return:     None
*********************************************************************
void SetECANMode( unsigned char Mode )
{
    CANCON = Mode;
    while ((CANSTAT & 0xE0) != Mode);
}
```

Figura 5.16. Sub-función SetECANMode.



Watch

Add SFR | ADCON0 | Add Symbol | __config_0

Update	Address	Symbol Name	Value	Hex	Decimal	Binary
	F70	BRGCON1	0x00	0x00	0	00000000
	F71	BRGCON2	0x00	0x00	0	00000000
	F72	BRGCON3	0x00	0x00	0	00000000
	F6F	CANCON	0x80	0x80	128	10000000
	F77	ECANCON	0x00	0x00	0	00000000
		BSEL0	ed Memory	Memory	d Memory	served Memory
	F6E	CANSTAT	0x80	0x80	128	10000000
	06D	J1939_Address	0x80	0x80	128	10000000

Watch 1 | Watch 2 | Watch 3 | Watch 4

Figura 5.17. Registro CANCON = 0x80h.

Al salir de la sub-función SetECANMode, la función J1939_Initialization continua configurando los registros faltantes, a continuación el registro ECANCON cambia su valor a 0x90h (Modo FIFO mejorado con los filtros 1, 2 y 3 de aceptación, habilitación de los registros BRGCON2 y BRGCON3), como se observa en la figura 5.18, también cambia el valor del registro BSEL0 a 0xE0 (Buffers 3-5 en modo transmisión y buffers 0-2 en modo recepción) ver figura 5.18.

Continuar haciendo clic para configurar máscaras y filtros de aceptación hasta llegar a los registros BRGCON1, BRGCON2 y BRGCON3, los cuales toman los valores mostrados en la figura 5.19. BRGCON1 = 0x01h (SJW = 1 x TQ y TQ = (2x2)/Fosc), BRGCON2 = 0xBF (la fase 2 del segmento de tiempo en programación libre "SEG2PHTS = 1", muestreo del bus CAN realizado una sola vez "SAM = 0", SEG1PH = 111(8 TQ) y tiempo de propagación de 8 TQs "PRSEG = 111") y BRGCON3 = 0x87h (Wake-up del CAN bus deshabilitado "WAKDIS =1", filtro del bus no utilizado para Wake-up "WAKFIL = 0" y fase 2 del segmento de tiempo = 8 TQ "SEG2PH = 111").

Después de ser configurados los registros de control del *Baud Rate* del CAN bus, se llama nuevamente la sub-función "SetECANMode" para poner el módulo CAN en Modo

Normal “CANCON = 0x00h” (figura 5.20) y pueda enviar y recibir mensajes, al salir de la sub-función, se reconfigura el registro ECANCON “ECANCON = 0xA5h” (figura 5.20) colocando el buffer 2 como transmisor de mensajes CAN en el estándar J1939 deseado.

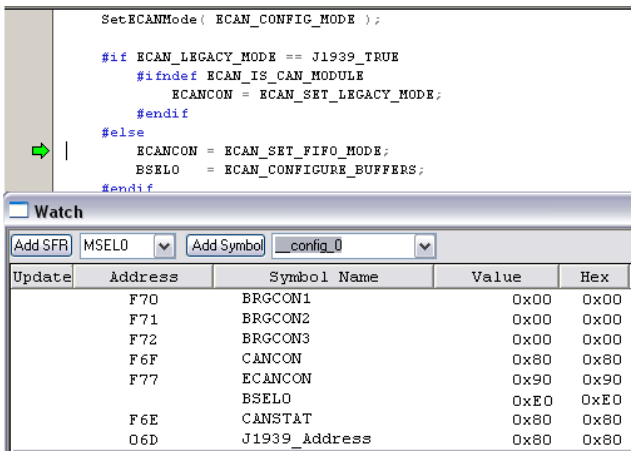


Figura 5.18. ECANCON = 0x90h y BSEL0 = 0xE0h.

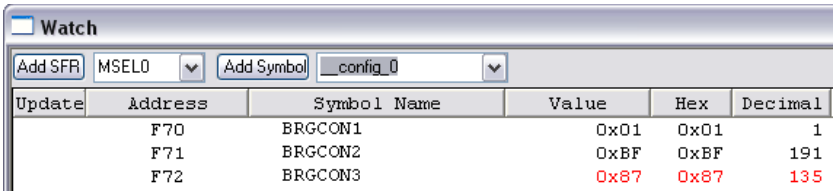


Figura 5.19. BRGCON1 = 0x01, BRGCON2 = 0xBF y BRGCON3 = 0x87.

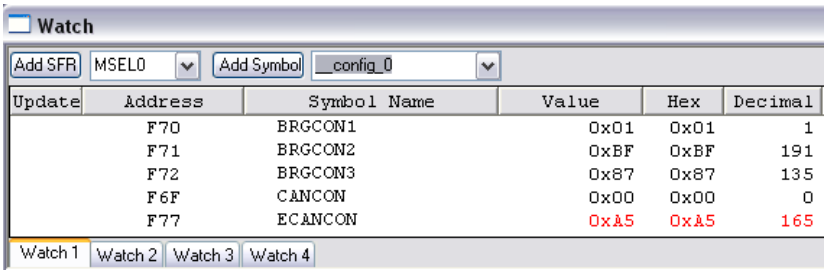


Figura 5.20. Modulo CAN en Modo Normal y ECAN adaptado para el estándar J1939.

Una vez configurado el módulo CAN y ECAN, se adapta el puerto B (PORTB2 y PORTB3) del microcontrolador para enviar y recibir mensajes en el bus (figura 5.21). PORTB2 = 0 “CANTX” (pin transmisor) y PORTB3 = 1 “CANRX” (pin receptor).

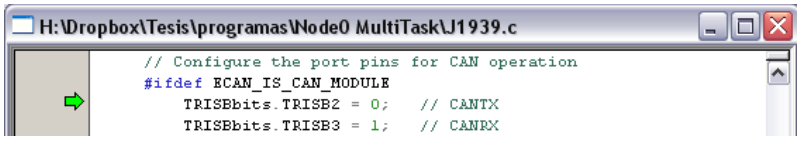


Figura 5.21. Configuración de los pines del puerto B para operación CAN.

Finalmente la función J1939_Initialization reclama (se adueña) de la dirección asignada al nodo (Nodo 1 = 128d y Nodo 2 = 129d), mediante la sub-función "J1939_AddressClaimHandling" (figura 5.22), al salir de esta función nuestro nodo tiene asignada la dirección que se le declaro en el archivo "J1939.def"

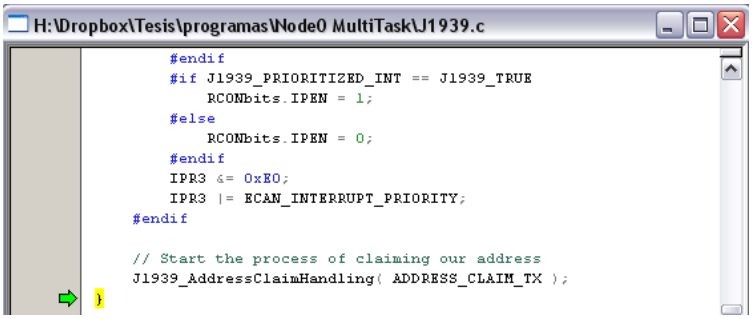


Figura 5.22. Reclamo de la dirección asignada al nodo.

Finalmente tenemos inicializado el protocolo J1939 y el bus CAN como se diseñó en el capítulo 4.3 "Diseño del Software" (ver tabla 4.1). Al salir de la función J1939_Initialization, el programa espera por lo menos 5 segundos para que los nodos (nodo 1 y nodo 2) se sincronicen en la red (figura 5.23).

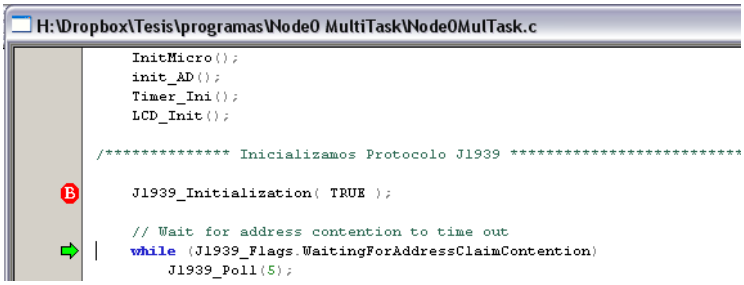


Figura 5.23. Espera para sincronización de nodos.

5.3. Pruebas en tiempo de ejecución.

Para realizar las pruebas en tiempo de ejecución es necesario programar el microcontrolador en el microcontrolador de manera que este quede corriendo de forma permanente en el microcontrolador. Para programar el programa en el microcontrolador, primero se va a la opción "Programmer" del software Mplab y se selecciona "PICkit 2" (ver figura 5.24) para realizar el quemado del programa en el microcontrolador.

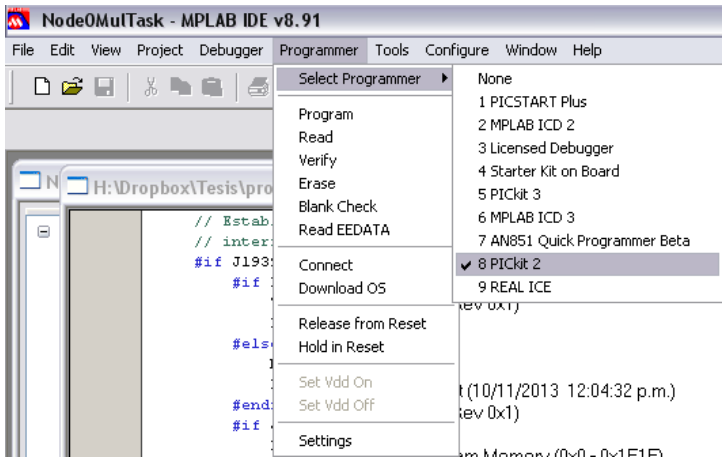


Figura 5.24. Selección del programador.

5.3.1. Pruebas al Nodo 1.

Despliegue correcto de los datos leídos en el nodo 2 y recibidos en el nodo 1.

Como se mencionó en el capítulo 4 (sección 4.2.3 "Red CAN"), el nodo 1 monitorea el transductor de presión MPX5010, pero en el display tipo LCD que se encuentra conectado en este nodo se despliega lo medido (temperatura del transductor LM35) en el nodo 2.

Para verificar que el display LCD conectado al nodo 1 despliega correctamente los datos (temperatura) obtenidos por el convertidor analógico-digital del nodo 2, se conecta un multímetro en la salida del sensor LM35 para monitorear la salida este y otro multímetro en la salida de la etapa de acondicionamiento, así monitoreamos lo que el nodo 2 se encuentra midiendo.

A continuación tomar lecturas cada que el sensor LM35 aumente un grado centígrado, para lograr que el sensor detecte un aumento en la temperatura se puede calentar este mediante una fuente de calor externa como puede ser un encendedor, un cautín, o con las manos. En la tabla 5.1 se muestran los resultados de 10 lecturas realizadas al nodo 1.

Para visualizar el aumento de temperatura en el display LCD, se presiona el interruptor conectado en el nodo 2, para que el nodo 1 reciba la orden y despliegue los datos en el display, como se muestra en la figura 5.25.

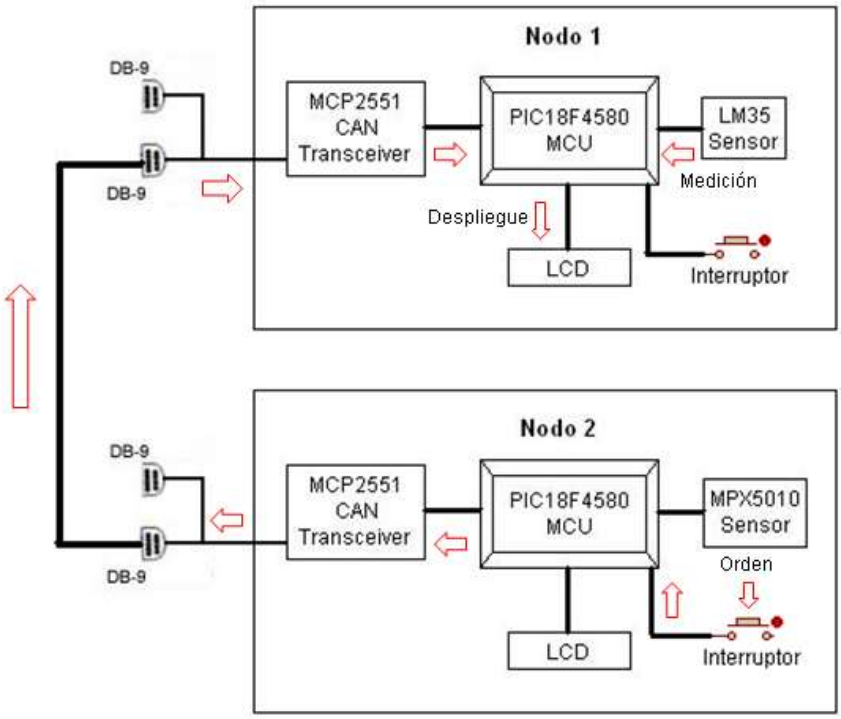


Figura 5.25. Medición y despliegue de datos medidos por el nodo 1.

De la tabla 5.1, se observa que el nodo 2 se encuentra leyendo correctamente la salida de la etapa de acondicionamiento del sensor LM35 (ver capítulo 4 sección 4.2.4 “Acondicionamiento de la señal de los sensores”), así como el envío correcto de estos. Por otro lado el nodo 1 despliega la temperatura de manera correcta sensada y transmitida desde el nodo 2.

Vout_LM35 (mV)	Vout_acondicionada (V)	Despliegue LCD (°C)
255.0	1.29	25 grados
264.6	1.33	26 grados
271.3	1.36	27 grados
284.4	1.42	28 grados
289.3	1.45	29 grados
301.5	1.501	30 grados
313.0	1.56	31 grados
328.2	1.64	32 grados
336.4	1.68	33 grados
347.2	1.74	34 grados

Tabla 5.1. Resultados obtenidos. Datos enviados por el nodo 2 y recibidos en el nodo 1.

5.3.2. Pruebas al Nodo 2.

Despliegue correcto de los datos leídos en el nodo 1 y recibidos en el nodo 2.

Como se mencionó en el capítulo 4 (sección 4.2.3 "Red CAN"), el nodo 2 monitorea el sensor de temperatura LM35, pero en el display LCD que se encuentra conectado en este nodo se despliega lo medido (presión del transductor MPX501) en el nodo 1.

Para verificar que el display conectado al nodo 2 despliega correctamente los datos (presión) obtenidos por el convertidor analógico-digital del nodo 1, se conecta un multímetro en la etapa de acondicionamiento, así monitoreamos lo que el nodo 1 se encuentra leyendo.

A continuación tomar lecturas cada que el sensor MPX5010 aumente dos kilo pascales, para lograr que el transductor aumente la presión, en el pivote del transductor se conecta una pequeña manguera para soplar a través de esta (el valor en kPa se obtiene utilizando la ecuación de conversión de voltaje a kPa ecuación 5.1) y así lograr que cambie la presión del transductor. En la tabla 5.2 se muestran los resultados de 10 lecturas realizadas al nodo 2.

$$\boldsymbol{P_{kPa}} = \frac{\left(\frac{V_{out}}{V_{SMPX5010}}\right) - 0.04}{0.09} \qquad (5.1)$$

Para visualizar el aumento de presión en el display, se presiona el interruptor conectado en el nodo 1, para que el nodo 2 reciba la orden y despliegue los datos en el display, como se muestra en la figura 5.26.

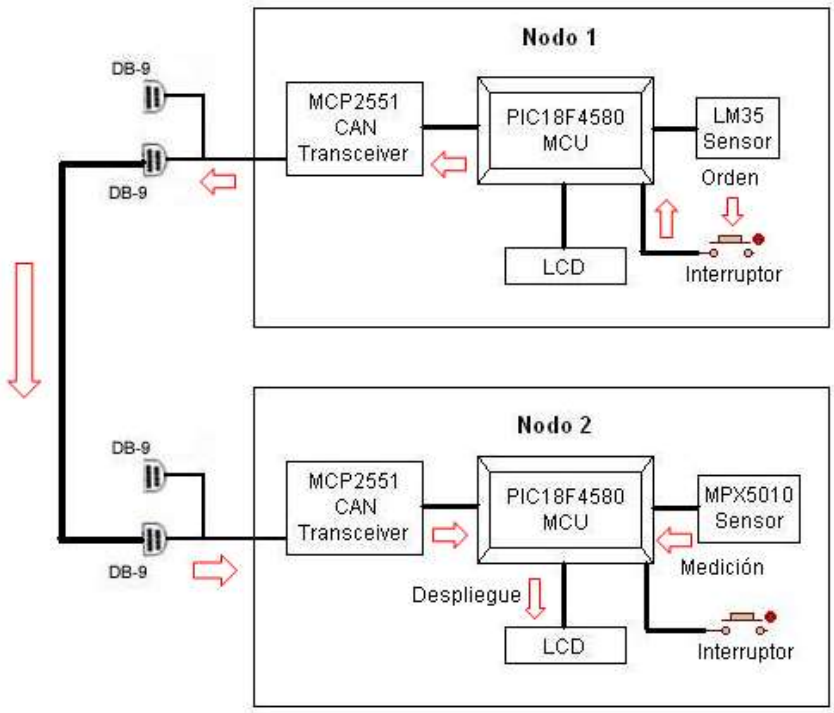


Figura 5.26. Medición y despliegue de datos medidos por el nodo 2.

De la tabla 5.2, se observa que el nodo 1 se encuentra leyendo correctamente la salida de la etapa de acondicionamiento del transductor MPX5010 (ver capítulo 4 sección 4.2.4 "Acondicionamiento de la señal de los sensores"), así como el envío correcto de estos. Por otro lado el nodo 2 despliega la presión de manera correcta medida y transmitida desde el nodo 1.

Vout_acondicionada (V)	Despliegue LCD (kPa)
1.238	2 kPa
1.31	8 kPa
1414	17 kPa
1.546	28 kPa
1.613	34 kPa
1.74	45 kPa
1.824	52 kPa
1.92	60 kPa
2.07	73 kPa
2.134	78 kPa

Tabla 5.2. Resultados obtenidos. Datos enviados por el nodo 1 y recibidos en el nodo 2.

Capítulo 6

Conclusiones

6.1. Conclusiones.

Como podemos ver a lo largo de este trabajo, los sistemas de control y monitoreo distribuido son de gran importancia en la industria ya que el control y supervisión de los procesos de fabricación pueden ser de manera remota. Además este tipo de sistemas simplifica la comunicación entre diferentes sistemas o subsistemas de diferentes fabricantes.

Por otro lado podemos observar que uno de los protocolos con mayor demanda para implementaciones en control distribuido es el protocolo CAN. Debido a que este protocolo permite control en tiempo real, sistema multímaestro y un alto nivel de integridad de información a altas velocidades de transmisión de datos, sobre todo en entornos con altas interferencias electromagnéticas. Debido a que este protocolo es altamente inmune a ruidos eléctricos y/o interferencias electromagnéticas es muy utilizado en el sector automotriz, para la comunicación entre las computadoras (ECUs) y subsistemas de los automóviles. Ya que este protocolo se encuentra implementado en las dos capas más bajas del sistema OSI, que son la capa física y la capa de datos.

Debido a que el protocolo CAN es un protocolo normalizado este permite comunicar subsistemas de diferentes fabricantes conectados en la misma red. Ya que puede delegar la carga de comunicaciones a un solo nodo o a distintos nodos a la vez y así mantener la integridad y velocidad del bus. Además de que este protocolo envía la información en paquetes de datos llamados “mensajes”, en los cuales se encuentra la prioridad del mensaje que es asignada por el diseñador de la red y hasta ocho bytes de datos.

Cabe destacar que este protocolo utiliza el método CSMA/CD, el cual consiste en que el mensaje con el identificador de mayor prioridad es el primero en ser procesado por el nodo maestro o nodos, así como encargarse de mantener la información intacta de los

mensajes a que son rechazados en el red debido a colisiones ocasionadas por mensajes que quieren obtener la prioridad del bus, para lo cual el bus utiliza el método de arbitrariedad.

Otra de las características importantes del protocolo CAN es que este puede transmitir mensajes tipo broadcast, es decir, enviar mensajes sobre el bus con el propósito de que todo nodo conectado a la red reciba los mensajes pero solo el nodo que necesite la información contenida en el mensaje sea el único en utilizarla. Por otro lado puede transmitir mensajes direcciones y/o nodos específicos. Además de que dichos mensajes pueden ser de datos, solicitar información de un nodo y/o mensajes de error.

Durante el diseño de la red se pudo comprender como el transceptor CAN adapta los niveles de voltajes del bus para que los mensajes sean transmitidos a través de este, ya que el módulo ECAN del microcontrolador envía los mensajes en tensión TTL (5V.) y el transceptor CAN adapta estos mensajes a tensiones superiores (12V.). También determina el número máximo de nodos que se pueden conectar en el bus. Además es necesario colocar el módulo CAN en modo configuración para poder establecer el modo en el cual se utilizará la red CAN. Una vez configurado la red CAN es necesario establecer el módulo ECAN en modo de operación normal para poder transmitir y recibir mensajes a través de la red.

En este caso la red se configuró en modo FIFO, debido a que este modo ofrece recursos mayores a ser utilizados. Nos ofrece tres buffers de transmisión y dos de recepción además del manejo automático del bit RTR. Por otro lado observamos que para poder iniciar la transmisión de un mensaje es necesario colocar el bit TXREQ = 1 y que se encuentre por lo menos un nodo con la misma velocidad de baudios en la red. Se vio que el microcontrolador PIC18F4580 cuenta con un mecanismo de priorización independiente al implícito en el protocolo CAN, el cual primeramente compara todos sus buffers que han sido encolados para la transmisión y el buffer con mayor prioridad es el primero en enviar los mensajes. También como el microcontrolador es capaz de accesar a un buffer de recepción mientras otro buffer está disponible para recibir otro mensaje. Uso de filtros y máscaras de aceptación para determinar los mensajes a ser cargados en el buffer de ensamble de mensajes (MAB). Una vez cargados en el buffer MAB se comparan los valores de los filtros con el campo ID del mensaje, si dicho valores coinciden en mensaje será cargado para su lectura.

De las pruebas funcionales se pudo observar cómo se sincronizan los nodos dentro de la red mediante el PLL interno de los microcontroladores de cada nodo conectado al bus. Para nuestra red, los nodos conectados en al bus se encuentran sincronizados a una velocidad en bauds o baud rate de 500 kHz.

Durante el diseño de la red, observamos cómo es necesaria una etapa de acondicionamiento de señales para algunos tipos de sensores, para que el microcontrolador pueda realizar una adecuada conversión analógico-digital. En nuestro caso el acondicionamiento de los sensores LM35 y MPX5010. Además de la facilidad de modificaciones que se le pueden realizar al software, debido a que este se encuentra estructurado por el método de programación por múlti-tarea, además de que permite que el microcontrolador ejecute varias funciones, siempre y cuando se cuiden los tiempos del procesador.

De igual forma, se comprendió el uso de la biblioteca J1939 de microchip para la transmisión de los mensajes a través de la red diseñada. Primero configurar el archivo J1939.def, para declarar la dirección de los nodos y la velocidad de bauds del bus. Además en el archivo J1939.c se descubrió que le hacían falta unas definiciones para poder utilizar dicha biblioteca con el microcontrolador seleccionado para este trabajo. Por otro lado fue de gran ayuda el método de simulación y depuración para realizar una configuración exitosa del protocolo CAN bus, ya que se pudo revisar línea por línea el código que se programó en el microcontrolador.

6.2. Trabajos futuros.

Finalmente, cabe señalar que es del interés del autor continuar con el estudio del protocolo CAN, así como el ampliar la red CAN conectando más nodos a esta, además de añadir un nodo dedicado a detección de errores dentro del bus, además de conectar el explorador CAN elaborado ya que por razones de tiempo no fue posible conectarlo a la red. Es también del interés del autor profundizar más en el control automotriz para diseñar una mejor red CAN con instrumentos, actuadores y sensores específicos del sector automotriz.

Apéndice A

A.1. Función "J1939_Initialization" de la biblioteca J1939.

Esta rutina es llamada cada que el sistema con estándar J1939 es inicializado. Esta función es la encargada de inicializar las variables globales, los periféricos del microcontrolador, así como sus interrupciones. Además es también la encargada de iniciar el proceso de reclamo de dirección para cada CA. Si el CA tiene un nombre y dirección predefinidos por el diseñador, la rutina llama a estos colocando un TRUE booleano como parámetro de entrada de la función. Pero si el nombre y dirección son configurados automáticamente (Self-configurable o Arbitrary Address Capable), el CA deberá inicializar el nombre y la dirección antes de llamar a la función "J1939_Initialization", enseguida llamar a esta función con un FALSE booleano como parámetro de entrada.

NOTA: El nombre es inicializado mediante el vector CA_Name, el cual tiene una longitud de un byte. La dirección es inicializada por la variable J1939_Address.

NOTA: Esta rutina NO habilita las interrupciones globales. Estas deben ser habilitadas por cada CA.

Parámetros: *Booleanos* Para inicializar o no inicializar el nombre y dirección.
Regreso: *None*

```
void J1939_Initialization( BOOL InitNAMEandAddress )
{
	unsigned chari;

	// Initialize global variables;
	J1939_Flags.FlagVal = 1;  // Cannot Claim Address, all other flags
cleared.
	ContentionWaitTime = 0l;
	TXHead = 0;
	TXTail = 0xFF;
	TXQueueCount = 0;
	RXHead = 0;
	RXTail = 0xFF;
	RXQueueCount = 0;

	if (InitNAMEandAddress)
	{
		J1939_Address = J1939_STARTING_ADDRESS;
		CA_Name[7] = J1939_CA_NAME7;
		CA_Name[6] = J1939_CA_NAME6;
		CA_Name[5] = J1939_CA_NAME5;
		CA_Name[4] = J1939_CA_NAME4;
		CA_Name[3] = J1939_CA_NAME3;
		CA_Name[2] = J1939_CA_NAME2;
		CA_Name[1] = J1939_CA_NAME1;
		CA_Name[0] = J1939_CA_NAME0;
```

```
    }
    CommandedAddress = J1939_Address;

    // Put the ECAN module into configuration mode and set it to the
    // desired mode.  Then configure the extra buffers for receive or
    // transmit as selected by the user.
    SetECANMode( ECAN_CONFIG_MODE );

    #if ECAN_LEGACY_MODE == J1939_TRUE
           #ifndef ECAN_IS_CAN_MODULE
                  ECANCON = ECAN_SET_LEGACY_MODE;
           #endif
    #else
           ECANCON = ECAN_SET_FIFO_MODE;
           BSEL0   = ECAN_CONFIGURE_BUFFERS;
    #endif

    // Set up mask 0 to receive broadcast messages.  Set up mask 1 to
    // receive messages sent to the global address (or eventually us).
    RXM0SIDH = 0x07;
    RXM0SIDL = 0x88; //0x80;
    RXM0EIDH = 0x00;
    RXM0EIDL = 0x00;
    RXM1SIDH = 0x00;
    RXM1SIDL = 0x08;
    RXM1EIDH = 0xFF;
    RXM1EIDL = 0x00;

    // Set up filter 0 to accept only broadcast messages (PF = 240-255).
    // Set up filter 2 and 3 to accept only the global address.  Once we
    // get an address for the CA, we'll change filter 3 to accept that
    // address.
    RXF0SIDH = 0x07;
    RXF0SIDL = 0x88;
    RXF2SIDL = 0x08;
    RXF2EIDH = J1939_GLOBAL_ADDRESS;
    RXF3SIDL = 0x08;
    RXF3EIDH = J1939_GLOBAL_ADDRESS;

    // If we're in Legacy Mode, we need to set up filters 1, 4,
    // and 5 also, since we can't disable them.
    #if ECAN_LEGACY_MODE == J1939_TRUE
           RXF1SIDH = 0x07;
           RXF1SIDL = 0x88;
           RXF4SIDL = 0x08;
           RXF4EIDH = J1939_GLOBAL_ADDRESS;
           RXF5SIDL = 0x08;
           RXF5EIDH = J1939_GLOBAL_ADDRESS;
    #endif

    #if ECAN_LEGACY_MODE == J1939_FALSE
           // Set mask 0 to filter 0, and mask 1 to filters 2 and 3.
           MSEL0    = 0x5C;

           // Leave all filters set to RXB0.  The filters will apply to
           // all receive buffers.
           RXFBCON0  = 0x00;
           RXFBCON1  = 0x00;
           RXFBCON2  = 0x00;

           // Enable filters 0, 2, and 3.  Disable the others.
           RXFCON0  = 0x0D;
           RXFCON1  = 0x00;
```

```
	#endif

	// Set up bit timing as defined by the CA
	BRGCON1 = ECAN_BRGCON1;
	BRGCON2 = ECAN_BRGCON2;
	BRGCON3 = ECAN_BRGCON3;

	// Put the ECAN module into Normal Mode
	SetECANMode( ECAN_NORMAL_MODE );

	// Give the network management transmit buffer the highest priority.
	SET_NETWORK_WINDOW_BITS;
	MAPPED_CON = 0x03;

	// Configure the port pins for CAN operation
	#ifdef ECAN_IS_CAN_MODULE
		TRISBbits.TRISB2 = 0;	// CANTX
		TRISBbits.TRISB3 = 1;	// CANRX
	#else
		#if defined(__18F4680) || defined(__18F4585) || defined(__18F2680)
|| defined(__18F2585)
			TRISBbits.TRISB3 = 1;	// CANRX - Changed to accomodate
18F4680 - CG
		#endif
		#if defined(__18F8680) || defined(__18F8585) || defined(__18F6680)
|| defined(__18F6585)
			TRISGbits.TRISG3 = 1;	// CANRX
		#endif
	#endif

	// Establish the ECAN interrupt priorities and enable the ECAN receive
	// interrupt.  The caller must enable global interrupts when ready.
	#if J1939_POLL_ECAN == J1939_FALSE
		#if ECAN_LEGACY_MODE == J1939_TRUE
			TXIntsEnabled = 0;
			PIE3 = ECAN_RX_INT_ENABLE_LEGACY | ECAN_ERROR_INT_ENABLE;
		#else
			BIE0 = ECAN_BUFFER_INTERRUPT_ENABLE;
			PIE3 = ECAN_RX_INT_ENABLE_FIFO | ECAN_ERROR_INT_ENABLE;
		#endif
		#if J1939_PRIORITIZED_INT == J1939_TRUE
			RCONbits.IPEN = 1;
		#else
			RCONbits.IPEN = 0;
		#endif
		IPR3 &= 0xE0;
		IPR3 |= ECAN_INTERRUPT_PRIORITY;
	#endif

	// Start the process of claiming our address
	J1939_AddressClaimHandling( ADDRESS_CLAIM_TX );
}
```

A.2. Función “InitMicro”.

Esta rutina es la encargada de inicializar la frecuencia de oscilación del microcontrolador, los periféricos y las variables locales a ser utilizadas por el programa principal.

Parámetros: *None*
Regreso: *None*

```
void InitMicro(void)
{

	OSCCON = 0b01100011;		//Oscillator Control Register
					//Idle Enable bit, 4 MHz, Internal
					oscillator block
	OSCTUNEbits.PLLEN = 1;		//PLL enabled for INTOSC (4 MHz and 8 MHz
only)

/*************** Inicializacion de Puertos y variables ******************/
	PORTB = 0;
	TRISB = 0;
	INTCON2bits.RBPU = 0;			// Pull ups enabled
	TRISBbits.TRISB5 = 1;			// Switch pin
	TRISD = 0;				// LCD pins
	LATD = 0;

	PORTA = 0;				// Inicialize PORTA
	LATA = 0;
	CMCON = 0x07;
	TRISA = 0x0F;

	PORTC = 0;
	TRISC = 0;
	TRISCbits.TRISC1 = 0;			// LED pins

	ISR_counter = 0;
	task0 = DESACT;
	task1 = DESACT;
	task2 = DESACT;
	task3 = DESACT;

}
```

A.3. Función “init_AD”.

Esta rutina es la encargada de inicializar los convertidores analógico-digitales del microcontrolador, la precisión de los convertidores y la elección del canal que realiza la conversión.

Parámetros: *unsigned char* Número de canal que realiza la conversión.
Regreso: *unsigned int* Resultado de la conversión analógico-digital.

```
/****** Rutina para el control del convertidor A/D *******/


/***** Inicializar el Convertidor ******/

void init_AD(void)
{
	ADCON1 = 0b00010000;		//Vref-, Vref+ (AVdd) AN10-AN0 (Analógicos)
	ADCON2 = 0b10111010;		//Right Justified, 20 TAD, Conversión clock
					//Fosc/64
}

/***** Conversión del A/D *****/

unsigned int conv_canal(unsigned char canal)
{
	unsigned char canal_temp = 0b00111000;		//Temporal para
actualización de canal.
	unsigned int canal_res = 0;			//Resultado de la
conversion.

	canal = canal << 2;			//Posicionamos el número de canal
	ADCON0 = canal & 0b00111100;		//Actualizamos el canal y prendemos
	el convertidor

	ADCON0bits.ADON = 1;			//Habilitamos el ADC

	canal = 0;
	Delay10TCYx (8);			// 20 Us espera a que se
establezcan los voltajes
	ADCON0bits.GO_DONE = 1;			//Inicia la conversión.

	while(ADCON0bits.GO_DONE){};

	canal_res = (int)ADRESH << 8;
	canal_res = canal_res + ((int)ADRESL);

	return (unsigned int)canal_res;		//Regresamos el valor de la
conversión
}
```

A.4. Función “Timer_Ini”.

Esta rutina es la encargada de inicializar y arrancar el timer encargado de realizar las interrupciones de alta prioridad del microcontrolador.

Parámetros: *None*
Regreso: *None*

```
void Timer_Ini(void)
{
	T1CON = 0b10000000; // Timer1 16-bit operation
			    // 1:1 Prescale value
	TMR1H = 0xF0;       // Interrupción cada 16,000 ciclos o 0.001 s
	TMR1L = 0x5F;       // Coloca Timer1 = 65,535 - 4000 = 60535 = F05F
			    // Duración de un ciclo c = 62.5 ns -- FCY = 16 MHz

	INTCONbits.PEIE = 1;            // Enable Peripheral interrupt
	INTCONbits.GIEH = 1;            // Enable Global interrupt High priority
	RCONbits.IPEN = 1;              // Enable priority levels on interrups
	PIE1bits.TMR1IE = 1;            // Enable Timer1 interrupt
	IPR1bits.TMR1IP = 1;            // TMR1 high priority
	T1CONbits.TMR1ON = 1;           // Enables timer 1
}
```

A.5. Función “LCD_Init”.

Esta rutina es la encargada de inicializar el display tipo LCD en modo escritura utilizando el puerto D del microcontrolador para escribir sobre el display.

Parámetros: *None*
Regreso: *None*

```
void LCD_Init(void)
{
	//Inicializa los pines del Micro
	// 12000 ciclos = 1mSeg
	TRISD = 0b11000000;
	ADCON1 = 0x0F;

	//Delay 15 mSeg
	Delay10KTCYx(6);
	LCD_RS = 0;
	LCD_Nibble(0x03);

	//Delay 4.1 mSeg
	Delay10KTCYx(2);
	LCD_Nibble(0x03);

	//Delay 100 uSeg
	Delay100TCYx(4);
	LCD_Nibble(0x03);

	LCD_Nibble(0x02);
	LCD_Command(0x28);
	LCD_Command(0x0C);
	LCD_Command(0x01);
	LCD_Command(0x06);
}
```

A.6. Función “Estructura main”.

En la estructura main se encuentran las cuatro tareas que realiza nuestro programa cíclicamente. Al inicio de la estructura se inicializa el microcontrolador (ver A.2), enseguida los convertidores analógico-digitales (ver A.3), después de inicializados los convertidores se inicializa el timer (ver A.4), a continuación se inicializa el display LCD (ver A.5) y finalmente el protocolo J1939 (ver A.1). Dentro del ciclo while se encuentran las tareas, en donde:

- *task0:* es la encargada de verificar el cambio de estado del switch.
- *task1:* estructura el mensaje a ser transmitido. Dependiendo el estado del switch es el mensaje que se estructura.
- *task2:* tarea realizada por el nodo receptor; es la encargada de decodificar el mensaje recibido y de realizar el despliegue de esta información a través del display LCD.
- *task3:* lee los convertidores analógico-digitales (presión y/o temperatura) del microcontrolador.

Parámetros: *None*
Regreso: *None*

```
void main (void)
{

	InitMicro();
	init_AD();
	Timer_Ini();
	LCD_Init();

/************** Inicializamos Protocolo J1939 ******************************/

	J1939_Initialization( TRUE );

	// Wait for address contention to time out
	while (J1939_Flags.WaitingForAddressClaimContention)
		J1939_Poll(5);

	// Now we know our address should be good, so start checking for
	// messages and switches.


	while(1)
	{

		if(task0 == ACT)
		{
			task0 = DESACT;

			CurrentSwitch = PORTBbits.RB5;
```

```
            }

            if(task1 == ACT)
            {
                  task1 = DESACT;

/******************** Instrucciones Realizadas Nodo Transmisor*****************/

                  if (LastSwitch != CurrentSwitch)
                  {
                        Msg.DataPage = 0;
                        Msg.Priority = J1939_CONTROL_PRIORITY;
                        Msg.DestinationAddress = NODE1;
                        Msg.DataLength = 1;
                        Msg.Data[0] = press;

                        if (CurrentSwitch == 0)
                              Msg.PDUFormat = Read_Data;
                        else
                              Msg.PDUFormat = Read_off;

                    while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS){};

                      LastSwitch = CurrentSwitch;

                  }

            }

            if(task2 == ACT)
            {
                  task2 = DESACT;

/******************** Instrucciones Realizadas Nodo Receptor ******************/

                  while (RXQueueCount > 0)
                  {
                        J1939_DequeueMessage( &Msg );

                        if (Msg.PDUFormat == Read_Data)
                        {
                                    LATCbits.LATC1 = 1;
                                    LM35 = Msg.Data[0];

                                    if(LM35 == 0)
                                    {
                                          LCD_Command(0x80);
                                          LCD_Str("Temperatura");
                                          sprintf(lectura,"%3d", (LM35));
                                          LCD_Command(0xC0);
                                          LCD_puts(lectura);
                                          LCD_Command(0xC4);
                                          LCD_Str("Grados");
                                    }

                                    else
                                    {
                                          LCD_Command(0x80);
                                          LCD_Str("Temperatura");
                                          sprintf(lectura,"%3d",
((LM35*100)/255)+7);
                                          LCD_Command(0xC0);
```

```
                                        LCD_puts(lectura);
                                        LCD_Command(0xC4);
                                        LCD_Str("Grados");
                                }
                        }
                        else if (Msg.PDUFormat == Read_off)
                                {
                                        LCD_Command(0x80);
                                        LCD_Str("CAN bus        ");
                                        LCD_Command(0xC0);
                                        LCD_Str("                ");
                                        LATCbits.LATC1 = 0;
                                }
                        }
                }

                if(task3 == ACT)
                {
                        task3 = DESACT;

                        press = conv_canal(1);
                        press = ((unsigned char)press<<8);
                }

                        // Since we don't accept the Commanded Address message,
                        // the value passed here doesn't matter.
                        J1939_Poll(20);

        }

}
```

Bibliogrfía

[Paret 2005]

Dominique Paret, Multiplexed Networks for embedded systems, CAN, LIN, Flexray, Safe-by-wire, París: John Wiley & Sons LTD, 2005.

[POPA 2005]

O. G. POPA, Learn Hardware, Firmware and Software design, Canada: Corollary Theorems Ltd, 2005.

[Softing 2012]

Industrial Softing. Páginas secundarias. Estados Unidos. 17 de mayo de 2012. www.softing.com/home/en/industrial-automation/products/can-bus/more-can-bus.html.

[Todopic 2007]

Todopic. Páginas secundarias. Argentina. 19 de noviembre de 2007. http://www.todopic.com.ar/foros/index.php?topic=19182.msg139742#msg139742.

[kvaser 2013]

kvaser. Páginas secundarias. USA. 20 de agosto de 2013. http://www.kvaser.com/en/about-can/higher-layer-protocols/36.html.

[BridgeWay 2010]

BridgeWay. *J1939 Data Mapping Explained*. USA, 2010.

[Microchip 2009]

Microchip, PIC18F2480/2580/4480/4580 Data sheet, 28/40/44-Pin, Enhanced Flash Microcontrollers with ECANTM , 10-bit A/D and nanoWatt Technology, USA, 2009.

[Microchip 2004]

Microchip, AN930, J1939 C Library for CAN-Enabled PICmicro® Microcontrollers , USA, 2004.

[Elektor electronics 2008]

Fredi Krüger, "CAN Explorer, Versatile PC-CAN Interface", *Bus Systems*, págs. 38-41, Febrero 2008.