CAPITULO 5 IMPLEMENTACIÓN PRÁCTICA DEL NODO CAN, COMO PROGRAMAR EL MICROCONTROLADOR.

En este capítulo se explicará la programación del dispositivo, así como el procedimiento a seguir para llegar al desarrollo final del programa.

Un objetivo planteado para el desarrollo de este proyecto fue programar en lenguaje de alto nivel debido a que en la actualidad éstos lenguajes se manejan en la industria automotriz; ésta afirmación toma sustento en la experiencia obtenida durante la etapa de estudio y desarrollo del proyecto, así como la realización de prácticas profesionales en la empresa *MAGNETI MARELLI*. Ésta empresa realiza programación de micro controladores CAN que se encuentran en los tableros de los automóviles de Volkswagen.

En dicha empresa se trabajó con los microcontroladores de ATMEL, es por esto que en capítulos anteriores se mencionó el conocimiento sobre el uso de este tipo de dispositivos. Durante el periodo en el que se trabajó con ellos, se pudo hacer la comparación entre dichos microcontroladores y los microcontroladores de Microchip, por lo que se llegó a la conclusión de que ambos microcontroladores implementaban las mismas características. Por otra parte se observó que las herramientas de desarrollo como lo son los emuladores y los ambientes de programación de Microchip son superiores en cuanto a versatilidad. Un ejemplo claro es el manejo de los *break points*, es decir, el ambiente de programación del AVR Studio. Éste programa se utiliza para programar los microcontroladores ATMEL; sin embargo, no es capaz de

detener el flujo del programa en cualquier punto a diferencia del MPLAB IDE que puede realizar el paro en cualquier punto deseado.

Otra de las razones por las cuales se recomienda el uso de los productos Microchip para la programación y puesta en marcha de proyectos, es que ésta universidad tiene una diversa gama de recursos para el desarrollo y simulación de los mismos. En contraste, no se tiene acceso a productos ATMEL lo cual implica un problema si se requiere hacer simulaciones o practicar con tablillas de prueba al desarrollar un proyecto.

5.1 CAN VS ECAN

Con la llegada de la familia de microcontroladores PIC18FXX8X que incluyen el módulo *Enhanced Control Area Network* (ECAN), los diseñadores pueden escoger entre el original módulo CAN presente en la familia de microcontroladores PIC18FXX8 y los nuevos dispositivos ECAN.

BasicCAN y FullCAN son otros dos conceptos que se utilizan para nombrar a los módulos CAN y ECAN respectivamente. Ambos módulos son capaces de manipular completamente el protocolo. El diseñador es libre de escoger cualquiera de los dos conceptos de CAN, pero debe tener en cuenta lo siguiente:

- Si el campo de identificadores se encuentra principalmente expresado en un máximo de 256 valores posibles y excepcionalmente en 11 bits, existe solo una opción económica, llamada BasicCAN.
- BasicCAN satisface generalmente el 90% de las aplicaciones industriales.
- FullCAN puede ser la solución idónea, cuando la velocidad de procesamiento es considerable.

Tanto el módulo CAN como el ECAN presentan plataformas estables y confiables para el desarrollo de aplicaciones basadas en CAN, sin embargo, cada uno es adecuado para ambientes diferentes. La siguiente tabla resume las principales características de cada una de las soluciones.

	Transmission – Low Traffic (duty < 10%)	**Transmission – High Traffic (duty > 10%)**
CAN	Good – Excellent Trade-Off Processing Time X Code Size	Higher processing times and possible delays due to priority inversions
ECAN	Almost no processing time but larger code(1)	Good – Up to 9 transmission buffers in Modes 1 and 2 and low probability of delays due to priority inversions
	Reception – Low Traffic (duty < 10%)	**Reception – High Traffic (duty > 10%) including RTR Messages**
CAN	Good – As in transmission, an excellent Trade-Off Processing Time X Code Size	High processing times and possible overload due to low number of buffers (2) and firmware-based filtering
ECAN	Almost no processing time but larger code(1)	Good – Up to 8 reception buffers in Modes 1 and 2. Low overhead in normal message reception and automatic RTR treatment (non-firmware based).

Tabla 1. Principales características de los módulos CAN y ECAN

De la tabla 1 se obtienen las siguientes conclusiones:

Los dispositivos CAN son adecuados para:

- Ambientes de bajo tráfico.
- Ambientes donde pocos mensajes (hasta 6 mensajes) deben ser recibidos y procesados por el sistema.
- Ambientes donde no se requieren grandes cantidades de información.
-

Por otra parte, ECAN es recomendado para:

- Ambientes de tráfico alto.
- Ambientes en donde el sistema debe seleccionar, recibir y procesar un gran número de mensajes (típicamente, hasta 16 mensajes). Para este tipo de ambientes el MODO 1 (*Enhanced legacy*) es bastante adecuado ya que cuenta con 16 filtros programables disponibles.
- Ambientes donde grandes cantidades de información son requeridas. Es este caso el MODO 2 (FIFO *mode*) es muy adecuado ya que maneja grandes ráfagas de datos de manera muy ordenada y sencilla.
-

5.2 Módulo CAN del microcontrolador.

Como se mencionó en capítulos anteriores, el lenguaje de programación utilizado en el desarrollo de éste proyecto, fue el lenguaje de alto nivel *C*. Para poder utilizar este lenguaje de programación dentro del ambiente del MPLAB IDE se utilizó el compilador C18, que es adecuado para la programación de microcontroladores de la familia PIC18.

Es importante aclarar que no se intenta escribir un manual de cómo utilizar el compilador, ni como utilizar el ambiente de desarrollo MPLAB, por lo que solo se detallarán los puntos de mayor relevancia. Está implícito que antes de comenzar a programar el módulo CAN, fue necesario hacer un estudio previo para poder utilizar el MPLAB IDE, aprender a utilizar el compilador C18, usar las librerías precargadas del compilador y el uso del lenguaje *C*, así como las variantes que presenta el compilador en cuanto a la programación de sus registros. En el capítulo 2 se hizo mención de éstos temas y se proporcionó una guía rápida de cómo comenzar a trabajar con dichas herramientas.

La principal fuente de información que sirvió de guía al programar el modulo CAN, fue la hoja de datos del micro controlador, ahí se encuentra la información necesaria para programar los registros. Además, la hoja de datos proporciona información relevante acerca del funcionamiento del protocolo y de una manera más práctica, adentra al funcionamiento del mismo.

Durante la explicación se presentarán imágenes con los registros de configuración de los que se esté hablando. El compendio total de los registros del módulo CAN ó ECAN se puede encontrar en la hoja de datos del microcontrolador.

En este apartado se explicará el funcionamiento del módulo CAN integrado en la familia del micro controlador PIC18, así como las principales características de dicho módulo.

5.2.1 Modos de operación CAN

El módulo CAN cuenta con los siguientes módulos de operación:

- *Configuration mode.*
- *Disable mode.*
- *Normal mode.*
- *Listen only mode.*
- *Loopback mode.*
- *Error recognition mode.*
-

Para tener acceso a alguno de los modos anteriormente mencionados es necesario configurar los *REQOP* bits, excepto *error recognition mode* al cuál se accede a través de los *CANRXM bits*. Para saber si se ha entrado a un modo, basta con monitorear los *OPMODE bits*. El usuario debe verificar que el dispositivo ha accesado al modo respectivo antes de realizar las operaciones pertinentes.

5.2.1.1 Configuration mode.

El módulo CAN debe ser inicializado antes de su activación. Éste modo de configuración es requerido poniendo en “1” el *REQOP2 bit*; cuando el estatus del *OPMODE2 bit* tiene un nivel alto, la inicialización puede ser desarrollada. Después de esto, los registros de configuración, máscaras de aceptación y filtros de aceptación pueden ser escritos.

5.2.1.2 Disable mode.

Solo el *WAKIF* bit puede ser configurado en este modo. Si el *REQOP*<2:0> es configurado como “001”, el módulo entrará en *disable mode*. Entrar a este módulo es lo mismo que sí deshabilitáramos los *enable bits*. Si el *WAKIE* esta en “1”, el procesador recibirá una interrupción al momento en el que el bus CAN detecta un bit “dominante”, como ocurre con el SOF (*Start of Frame*).

5.2.1.3 Normal mode.

En éste modo el dispositivo activamente monitorea todos los mensajes del bus y genera *acknowledge bits*, tramas de error, etc.

5.2.1.4 Listen only mode.

Éste modo puede ser usado para aplicaciones de monitoreo de bus. El modo de solo escucha es un modo silencioso, lo que significa que el dispositivo no generará mensajes mientras se encuentre en este estado**,** incluyendo las banderas de error o señales de *acknowledge*. Los filtros y máscaras pueden ser usados para el filtrado o pueden ser puestos todos en ceros para permitir que cualquier identificador pase. Este modo es activado configurando los *REQOP bits* en el registro *CANCON*.

5.2.1.5 Error recognition mode.

El módulo puede ser configurado para ignorar todos los errores y recibir cualquier mensaje. Éste modo es activado configurando los bits *RXM* <1:0> que se encuentran en el registro *RXBnCON* como “11”.

5.2.2. Transmisión CAN

El PIC18CXX8 implementa tres buffers de transmisión. Cada unos de estos *buffers* ocupa 14 bytes de *SRAM* y se encuentran mapeados dentro del mapa de memoria del dispositivo. El *bit*

TXREQ debe ser limpiado para indicar que no existan mensajes pendientes por ser transmitidos. Para iniciar una transmisión al menos los registros *TXBnSIDH*, *TXBnSIDL* y *TXBnDLC* deben ser cargados. Si bytes de datos se encuentran presentes en el mensaje, entonces el registro *TXBnDm* debe ser cargado. Si el mensaje maneja identificadores extendidos, los registros *TXBnEIDH*, *TXBnEIDL* y el *bit EXIDE* deben ser configurados. El *bit TXIE* debe ser configurado para habilitar o deshabilitar las interrupciones al enviar los mensajes. Además el bit de prioridad debe ser configurado según se requiera.

5.2.2.1nicio de la transmisión.

Con el objetivo de iniciar la transmisión de un mensaje, el bit *TXREQ* debe ponerse en nivel alto, cuando este bit es puesto en “1” los bits *TXABT*, *TXLARB* y *TXERR* son reiniciados.

La transmisión comenzará cuando el dispositivo detecte que el bus esta disponible. Cuando la transmisión es completada satisfactoriamente, el bit *TXREQ* es limpiado; el bit *TXBnIF* se activará y una interrupción será generada siempre y cuando el bit *TXBnIE* esté en nivel alto. Si se presenta un error en la transmisión el bit *TXREQ* seguirá en alto y una de las siguientes banderas de condición se activara: *TXERR, IRXIF o TXLARB.*

Para mayor comprensión del proceso de transmisión de mensajes, en la figura 1 que se muestra a continuación, se puede observar un diagrama de flujo que explica claramente cómo se lleva a cabo este proceso.

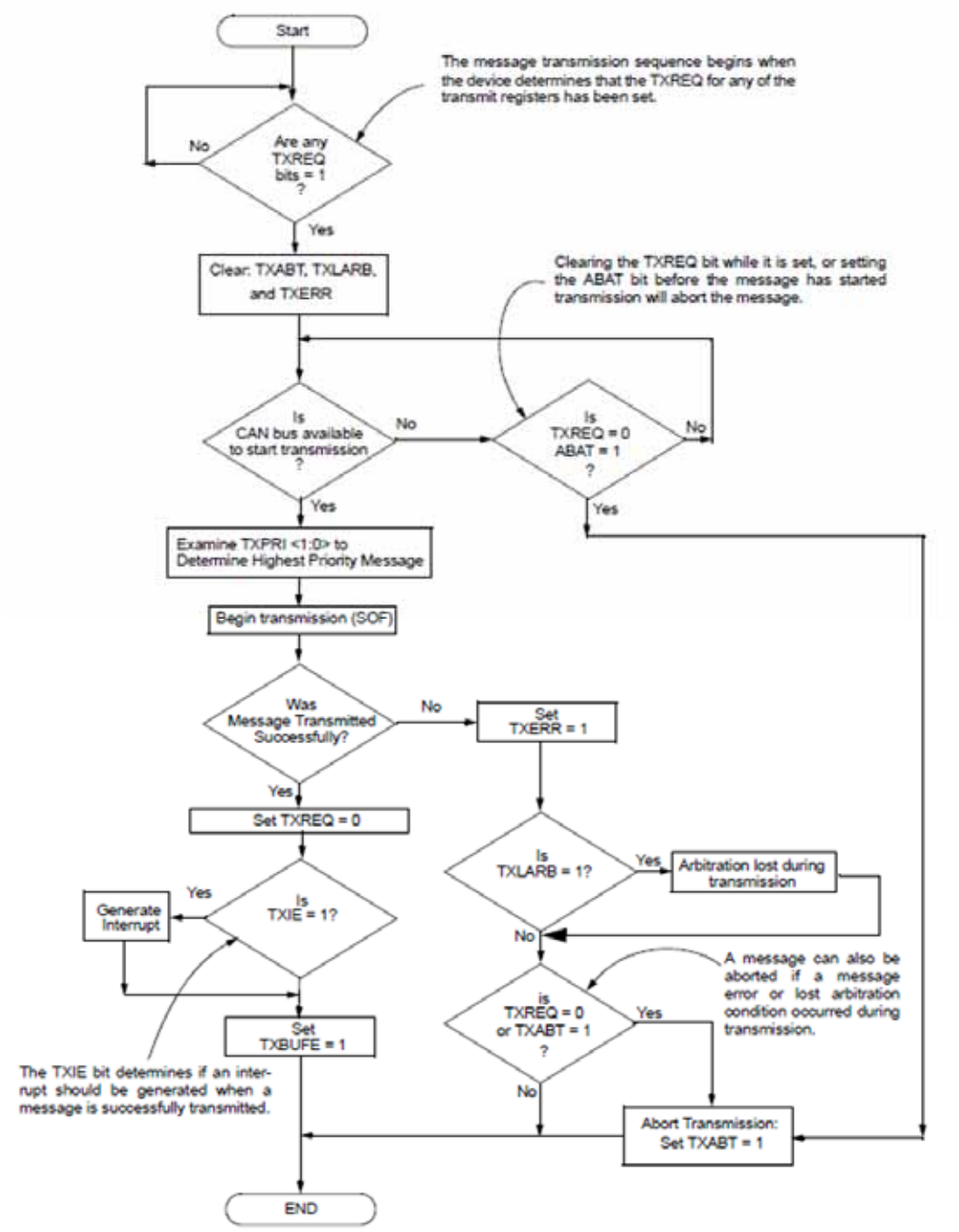


Figura 1. Diagrama de flujo del proceso de transmisión de un mensaje.

5.2.3 Recepción CAN

El PIC18 incluye 2 buffers de recepción con múltiples filtros de aceptación para cada uno; existe otro buffer separado llamado *Message Assembly buffer* (MAB), el cual actúa como un tercer buffer de recepción.

5.2.3.1 Buffers de recepción

- **MAB:** Recibe el siguiente mensaje del bus.
- **RXB0 y RXB1:** Pueden recibir cualquier mensaje del protocolo.
- **RXBnIF:** Este bit se activa cuando un mensaje es recibido dentro del buffer. Este bit debe ser apagado por el MCU para que otro mensaje pueda ser recibido por el buffer.
- **RXBnIE:** Si este bit se encuentra activado, se generará una interrupción para indicar cuando un mensaje válido ha sido recibido.
-

5.2.3.2 Prioridad de recepción

- El buffer **RXB0** es el buffer de alta prioridad y cuenta con dos filtros de aceptación de mensajes.
- El buffer **RXB1** es el buffer de baja prioridad y cuenta con cuatro filtros de aceptación de mensaje.
- Además se cuenta con dos máscaras de aceptación disponibles, una para cada buffer.

Cuando un mensaje es recibido, los bits <3:0> del registro *RXBnCON* indicarán el número del filtro de aceptación que habilitó la recepción y también nos indican cuando el mensaje recibido es una trama remota.

Los bits *RXM* configuran los modos especiales de recepción, es decir, si se configuran como”00” queda habilitada la recepción de cualquier mensaje válido. Por otra parte, si los bits *RXM* se configuran como “01” o “10”, la recepción solo aceptará mensajes con identificadores estándar o extendidos respectivamente.

Los bits *RXM* tienen mayor transcendencia que el bit *EXIDE*. Si los bits *RXM* se configuran en “11”, el buffer recibirá todos los mensajes, sin importar el valor de los filtros de aceptación. Además, si un mensaje presenta un error antes de finalizar la trama, esa parte del mensaje ensamblada en el *MAB* antes del error, se cargará dentro del buffer. Este modo presenta grandes ventajas al momento de debuggear un sistema CAN y podría no ser utilizado en un ambiente normal de sistema.

Para mejor comprensión del proceso de recepción de mensajes, en la figura 2, se observa un diagrama de flujo que explica la metodología de este proceso.

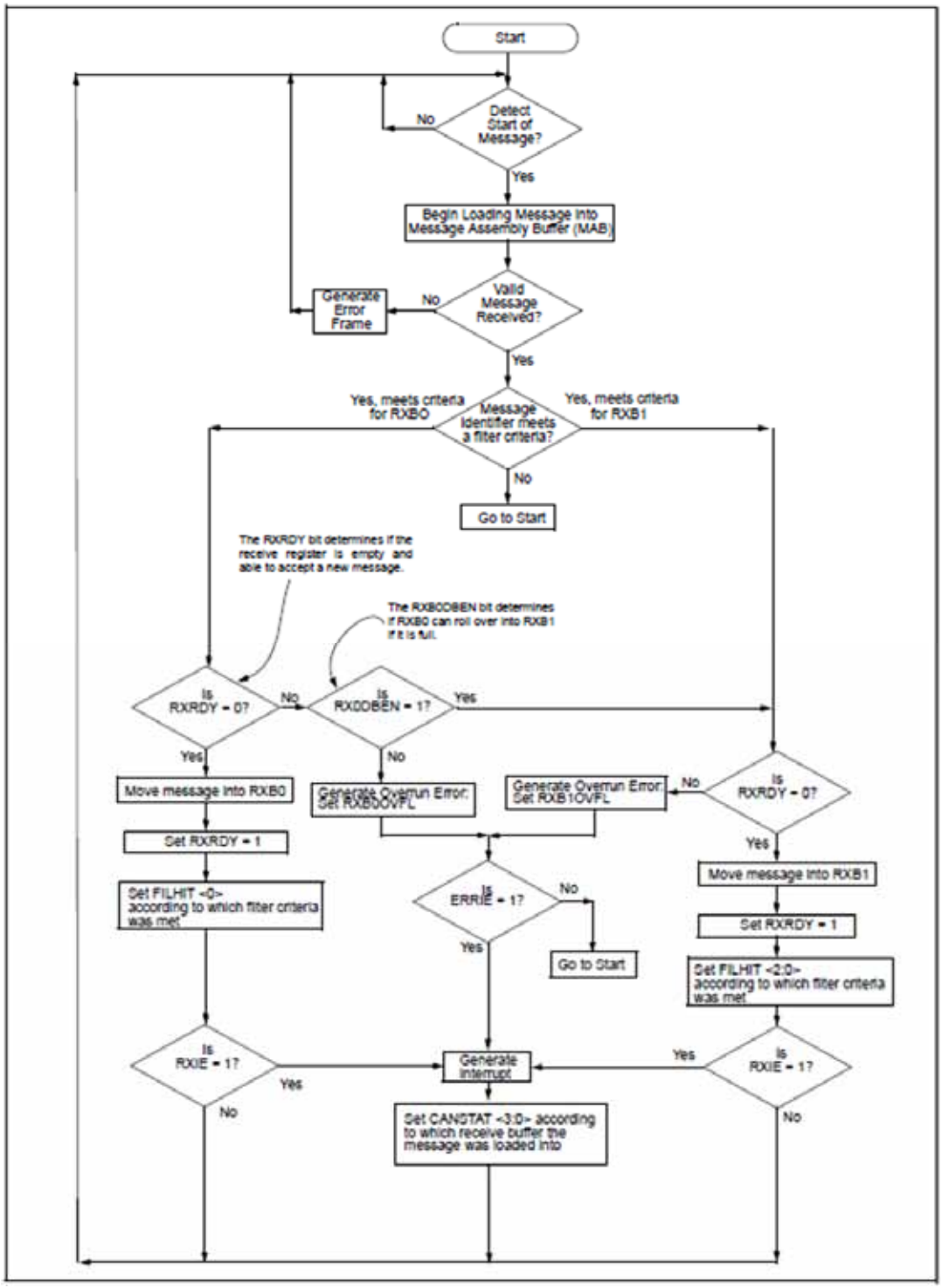


Figura 2. Diagrama de flujo del proceso de recepción de un mensaje.

5.2.4 Filtros y mascaras de aceptación de los mensajes

Una máscara de aceptación de mensajes determina cuales bits de los filtros de aceptación aplicar. Si algún bit de la máscara de aceptación es en “0”, automáticamente ese bit será aceptado, sin importar el valor del filtro de aceptación.

A continuación se presenta una *tabla de verdad* donde se puede comprender de una mejor manera, lo mencionado anteriormente. En la tabla 2 se observan combinaciones de los bits de la máscara y el filtro de aceptación el bit del identificador es aceptado o rechazado.

Mask bit n	Filter bit n	Message Identifier bit n001	Accept or Reject bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Tabla 2. Tabla de verdad de los filtros y mascaras de aceptación

Cuando un filtro es coincidido y el mensaje es cargado dentro del buffer de recepción, el número del filtro que habilitó la recepción es cargado dentro de los bits de *FILHIT*. La asociación de los filtros esta dada de la siguiente forma:

RXB0 → RXF0, RXF1 y RXM0

RXB1 → RXF2, RXF3, RXF4, RXF5 Y RXM1

Para el buffer de recepción *RXB0*, el registro *RXB0CON* contiene el bit *FILHIT0*, el cual indica que filtro de aceptación fue el que permitió la recepción. La codificación de dicho bit se muestra a continuación:

1 = Filtro de aceptación 1 (*RXF1*)

0 = Filtro de aceptación 0 (*RXF0*)

Para el buffer de recepción *RXB1* sucede lo mismo, sólo que este buffer al contar con mayor numero de filtros, necesita 3 bits *FILHIT* para mostrar el valor del filtro que permite la recepción. Estos bits quedan codificados de la siguiente manera:

111 = Reservado.

110 = Reservado.

101 = Filtro de aceptación 5 (*RXF5*)

100 = Filtro de aceptación 4 (*RXF4*)

011 = Filtro de aceptación 3 (*RXF3*)

010 = Filtro de aceptación 2 (*RXF2*)

001 = Filtro de aceptación 1 (*RXF1*) solo posible cuando el bit de *B0DBEN* está activo.

000 = Filtro de aceptación 0 (*RXF0*) solo posible cuando el bit de *B0DBEN* está activo.

Es importante mencionar que los filtros y máscaras de aceptación solo pueden ser configurados cuando el PIC se encuentra en modo de configuración.

5.2.5 Configuración de la tasa de transmisión

Los nodos de una red CAN deben tener la misma *tasa de bit nominal*. Sin embargo, no necesariamente todos los dispositivos en la red deben tener el mismo oscilador maestro.

Para frecuencias de reloj diferentes en dispositivos distintos, la tasa de bit debe ser ajustada, configurando propiamente el preescalador de la tasa de transmisión y el número de *quanta* en cada segmento. La tasa de bit nominal es el número de bits transmitidos por segundo asumiendo que se tiene un transmisor ideal con un oscilador ideal. La tasa de bit nominal esta definida a ser de un máximo de 1Mbit/s.

Parta poder configurar la tasa de transmisión de la red CAN, se tienen que tomar en cuenta los tiempos de *quanta*, así como el número de *quantas* que se tienen por cada unos de los segmentos de bit que se explicaron en el capítulo 1. En la práctica todos éstos cálculos son transparentes al usuario, ya que existen calculadoras electrónicas y tablas preestablecidas para la definición de este factor. Más adelante, cuando se presente la programación del módulo CAN, se explicará a detalle como se calculó la tasa de transmisión de nuestra red.

5.2.6 Interrupciones CAN

Todas las interrupciones tienen una sola fuente generadora, con la excepción de la interrupción de error. Las interrupciones se pueden dividir en dos categorías: interrupciones de recepción y transmisión.

Interrupción de transmisión: Cuando esta interrupción se encuentra habilitada, una interrupción será generada cuando el buffer de transmisión se encuentre vacío y listo para ser cargado con un nuevo mensaje. El bit *TXBnIF* será activado y debe ser limpiado por la MCU.

Interrupción de recepción: Esta interrupción es generada cuando un mensaje es satisfactoriamente recibido y cargado dentro de un buffer de recepción asociado. De igual manera que en el buffer de transmisión, la bandera de interrupción del buffer de recepción debe ser limpiada.

5.3 Programa del módulo CAN con MPLAB IDE.

En las siguientes líneas se explicará como programar dicho módulo con ayuda del ambiente de desarrollo MPLAB y el compilador C18, mencionando solo los registros que fueron necesarios configurar para el desarrollo de nuestro proyecto.

Una vez configurado el MPLAB como se mencionó en el capitulo 2, ya podemos empezar a programar nuestro microcontrolador.

5.3.1 Configuración de los puertos

El módulo CAN del microcontrolador, utiliza dos puertos, uno de entrada para el *CANRX* y otro de salida para el *CANTX*. Estos bits se encuentran en el puerto B

5.3.2 Registros de interrupción

Se recomienda tener un método que inicialice las interrupciones, con el objetivo de que al inicio de la transmisión CAN éstas puedan ser utilizadas sin problema. A continuación se muestra el fragmento de código utilizado en nuestro programa.

```
void INTERRUPTS_init(void)
{
     IPR3bits.TXB0IP=1;       //high priority interrupt
     PIE3bits.TXB0IE = 1;     //reception  buffer  0  interrupt
enable

     IPR3bits.RXB0IP=1;       //high priority interrupt
     PIE3bits.RXB0IE = 1;          //reception     buffer     0
interrupt enable

     INTCONbits.PEIE =   1;
     INTCONbits.GIE = 1;

     PIE1bits.TMR1IE=1;                  //habilita           la
interrupcion del TMR1

}
```

Código de programa 1. Método de habilitación de interrupciones.

IPR3: PERIPHERAL INTERRUPT PRIORITY REGISTER 5

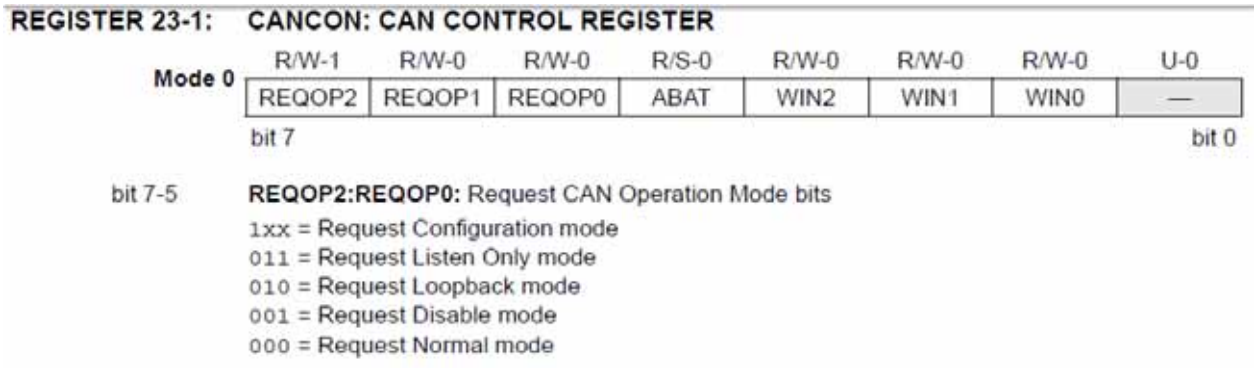
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
IRXIP	WAKIP	ERRIP	TXB2IP	TXB1IP(1)	TXB0IP(1)	RXB1IP	RXB0IP

Figura 3. Registro de prioridad de la interrupción.

En el fragmento de código anterior se le asigna prioridad alta a los registros de interrupción de los buffers de transmisión y de recepción. Los bits PEIE y GIE no deben estar desactivados si es que se requiere que se generen interrupciones por parte de los periféricos del microcontrolador. Por último, se muestra el habilitador de interrupción del TMR1.

5.3.3 Modo de configuración

Una vez configurados los puertos del microcontrolador y las interrupciones, procedemos a configurar el módulo CAN.



REGISTER 23-1: CANCON: CAN CONTROL REGISTER

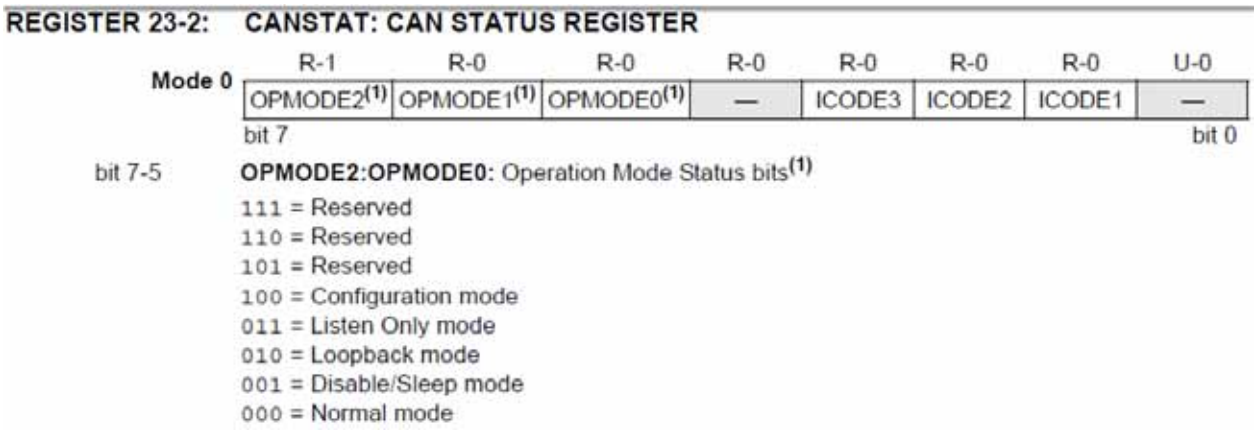
Mode 0

R/W-1	R/W-0	R/W-0	R/S-0	R/W-0	R/W-0	R/W-0	U-0
REQOP2	REQOP1	REQOP0	ABAT	WIN2	WIN1	WIN0	—
bit 7							bit 0

bit 7-5 **REQOP2:REQOP0:** Request CAN Operation Mode bits

1xx = Request Configuration mode
011 = Request Listen Only mode
010 = Request Loopback mode
001 = Request Disable mode
000 = Request Normal mode

Figura 4. Registro CANCON



REGISTER 23-2: CANSTAT: CAN STATUS REGISTER

Mode 0

R-1	R-0	R-0	R-0	R-0	R-0	R-0	U-0
OPMODE2(1)	OPMODE1(1)	OPMODE0(1)	—	ICODE3	ICODE2	ICODE1	—
bit 7							bit 0

bit 7-5 **OPMODE2:OPMODE0:** Operation Mode Status bits(1)

111 = Reserved
110 = Reserved
101 = Reserved
100 = Configuration mode
011 = Listen Only mode
010 = Loopback mode
001 = Disable/Sleep mode
000 = Normal mode

Figura 5. Registro CANSTAT.

5.3.4 configuración de la tasa de transmisión

Posteriormente, se define la tasa de transmisión del nodo. La tasa de transmisión se define en base al oscilador que se maneje. Es importante mencionar que la tasa de transmisión debe ser la misma para todos los nodos de la red, sin embrago, el oscilador maestro de cada nodo no tiene que ser necesariamente del mismo valor. Para el ajuste de este factor, se toma en cuenta el valor del oscilador y BRP (*Bud rate preescaler*) para que el valor deseado pueda ser alcanzado.

A continuación se presentan las fórmulas necesarias para configurar nuestra tasa de transmisión:

$$TBIT = \frac{1}{Nominal\ Bit\ Rate} \tag{5.1}$$

$$Nominal\ Bit\ Time = TQ\ \times \left(Sync_{Seg} + Pro_{Seg} + \ Phase_{Seg1} + \ Phase_{Seg2}\right) \tag{5.2}$$

$$TQ(\mu s) = \frac{2 \times (BRP + 1)}{Fosc\ (MHz)} \tag{5.3}$$

$$Ó$$

$$TQ(\mu s) = \left(2\ \times (BRP\ + 1)\right) \times Tosc\ (\mu s) \tag{5.4}$$

Donde:

Tbit = Tiempo de bit.

Nominal Bit Time = Tasa de bit.

Tq = Tiempo de “quanta”.

Sync_Seg = Segmento de sincronización.

Prop_Seg = Segmento de propagación.

Phase_Seg1 = Segmento de fase 1.

Phase_Seg2 = Segmento de fase 2.

En la figura 6 se presentan algunos ejemplos del cálculo de la tasa de transmisión de bits:

EXAMPLE 23-6: CALCULATING TQ, NOMINAL BIT RATE AND NOMINAL BIT TIME

TQ (µs) = (2 * (BRP + 1))/FOSC (MHz)

TBIT (µs) = TQ (µs) * number of TQ per bit interval

Nominal Bit Rate (bits/s) = 1/TBIT

This frequency (FOSC) refers to the effective frequency used. If, for example, a 10 MHz external signal is used along with a PLL, then the effective frequency will be 4 x 10 MHz which equals 40 MHz.

CASE 1:

For FOSC = 16 MHz, BRP<5:0> = 00h and Nominal Bit Time = 8 TQ:

TQ = (2 * 1)/16 = 0.125 µs (125 ns)

TBIT = 8 * 0.125 = 1 µs (10^{-6}s)

Nominal Bit Rate = $1/10^{-6} = 10^6$ bits/s (1 Mb/s)

CASE 2:

For FOSC = 20 MHz, BRP<5:0> = 01h and Nominal Bit Time = 8 TQ:

TQ = (2 * 2)/20 = 0.2 µs (200 ns)

TBIT = 8 * 0.2 = 1.6 µs ($1.6 * 10^{-6}$s)

Nominal Bit Rate = $1/1.6 * 10^{-6}$s = 625,000 bits/s (625 Kb/s)

CASE 3:

For FOSC = 25 MHz, BRP<5:0> = 3Fh and Nominal Bit Time = 25 TQ:

TQ = (2 * 64)/25 = 5.12 µs

TBIT = 25 * 5.12 = 128 µs ($1.28 * 10^{-4}$s)

Nominal Bit Rate = $1/1.28 * 10^{-4}$ = 7813 bits/s (7.8 Kb/s)

Figura 6. Ejemplos de cálculos de la velocidad de transmisión.

Los bits que deben ser configurados para la configuración de la tasa de transmisión son: *BRGCON1, BRGCON2 y BRGCON3* quedando definidos como se muestra en las figuras 7, 8 y 9 que se presentan a continuación:

REGISTER 23-52: BRGCON1: BAUD RATE CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
bit 7							bit 0

bit 7-6 **SJW1:SJW0:** Synchronized Jump Width bits
11 = Synchronization jump width time = 4 x TQ
10 = Synchronization jump width time = 3 x TQ
01 = Synchronization jump width time = 2 x TQ
00 = Synchronization jump width time = 1 x TQ

bit 5-0 **BRP5:BRP0:** Baud Rate Prescaler bits
111111 = TQ = (2 x 64)/FOSC
111110 = TQ = (2 x 63)/FOSC
:
:
000001 = TQ = (2 x 2)/FOSC
000000 = TQ = (2 x 1)/FOSC

Figura 7. Registro BRGCON1.

REGISTER 23-53: BRGCON2: BAUD RATE CONTROL REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SEG2PHTS	SAM	SEG1PH2	SEG1PH1	SEG1PH0	PRSEG2	PRSEG1	PRSEG0
bit 7							bit 0

bit 7 **SEG2PHTS:** Phase Segment 2 Time Select bit
1 = Freely programmable
0 = Maximum of PHEG1 or Information Processing Time (IPT), whichever is greater

bit 6 **SAM:** Sample of the CAN bus Line bit
1 = Bus line is sampled three times prior to the sample point
0 = Bus line is sampled once at the sample point

bit 5-3 **SEG1PH2:SEG1PH0:** Phase Segment 1 bits
111 = Phase Segment 1 time = 8 x TQ
110 = Phase Segment 1 time = 7 x TQ
101 = Phase Segment 1 time = 6 x TQ
100 = Phase Segment 1 time = 5 x TQ
011 = Phase Segment 1 time = 4 x TQ
010 = Phase Segment 1 time = 3 x TQ
001 = Phase Segment 1 time = 2 x TQ
000 = Phase Segment 1 time = 1 x TQ

bit 2-0 **PRSEG2:PRSEG0:** Propagation Time Select bits
111 = Propagation time = 8 x TQ
110 = Propagation time = 7 x TQ
101 = Propagation time = 6 x TQ
100 = Propagation time = 5 x TQ
011 = Propagation time = 4 x TQ
010 = Propagation time = 3 x TQ
001 = Propagation time = 2 x TQ
000 = Propagation time = 1 x TQ

Figura 8. Registro BRGCON2.

REGISTER 23-54: BRGCON3: BAUD RATE CONTROL REGISTER 3

R/W-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
WAKDIS	WAKFIL	—	—	—	SEG2PH2(1)	SEG2PH1(1)	SEG2PH0(1)
bit 7							bit 0

bit 7 **WAKDIS:** Wake-up Disable bit
1 = Disable CAN bus activity wake-up feature
0 = Enable CAN bus activity wake-up feature

bit 6 **WAKFIL:** Selects CAN bus Line Filter for Wake-up bit
1 = Use CAN bus line filter for wake-up
0 = CAN bus line filter is not used for wake-up

bit 5-3 **Unimplemented:** Read as '0'

bit 2-0 **SEG2PH2:SEG2PH0:** Phase Segment 2 Time Select bits(1)
111 = Phase Segment 2 time = 8 x TQ
110 = Phase Segment 2 time = 7 x TQ
101 = Phase Segment 2 time = 6 x TQ
100 = Phase Segment 2 time = 5 x TQ
011 = Phase Segment 2 time = 4 x TQ
010 = Phase Segment 2 time = 3 x TQ
001 = Phase Segment 2 time = 2 x TQ
000 = Phase Segment 2 time = 1 x TQ

Note 1: Ignored if SEG2PHTS bit (BRGCON2<7>) is '0'.

Figura 9. Registro BRGCON1.

Una vez expuestas las fórmulas que se ocupan para calcular el *bit rate* y mostrados los registros de configuración, es posible mencionar que existen técnicas mucho más prácticas para realizar estos cálculos. Lo que se recomienda en éste proyecto es utilizar las calculadoras que existen en internet para realizar dicho procedimiento. Un ejemplo de dichas calculadoras es la calculadora de “KVASER..

Una vez que se ha ingresado a la calculadora “kvaser”, nos aparecerá la siguiente pantalla:

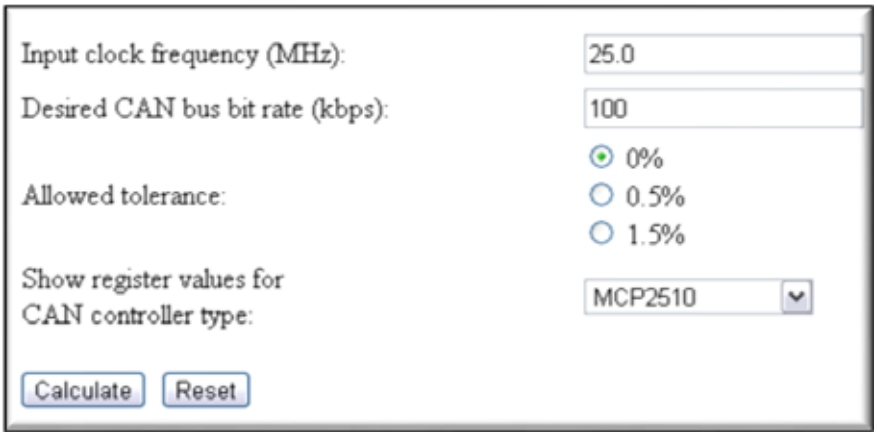


Figura 10. Calculadora "kvaser"

En la figura 10 se muestra la calculadora "kvaser". En dicha calculadora como se puede observar, sólo debemos ingresar el valor del oscilador que se está utilizando, la tasa de transmisión deseada, el porcentaje de error permitido y por último el controlador CAN utilizado.

Una vez que se han ingresado los datos, se presiona el botón calcular y se desplegará una ventana como la que se muestra en la figura 11. En dicha figura se muestran los valores ingresados cuando se tiene un oscilador de 25Mhz y se desea una tasa de transmisión de 100Kbps.

The following bus parameters are allowed, assuming

- your CAN controller is clocked with **25 MHz**,
- you want a CAN bus bit rate of **100 kbps**,
- settings leading to speed deviations of more than 0% are discarded.

T1	T2	BTQ	SP%	SJW	Bit Rate	Err%	CNF1	CNF2	CNF3
17	8	25	68	1	100	0	04	bf	07
17	8	25	68	2	100	0	44	bf	07
17	8	25	68	3	100	0	84	bf	07
17	8	25	68	4	100	0	c4	bf	07
3	2	5	60	1	100	0	18	80	01
3	2	5	60	2	100	0	58	80	01
3	2	5	60	3	100	0	98	80	01
3	2	5	60	4	100	0	d8	80	01

Explanations

- T1: the number of quanta **before** the sampling point.
- T2: the number of quanta **after** the sampling point.
- BTQ: the number of quanta per bit.
- SP%: the position of the sampling point, in percent of the whole bit.
- SJW: the Synchronization Jump Width.
- Bit Rate: the resulting bus speed, in kbps.
- Err%: the deviation, in percent, from the desired bit rate.
- CNF0: Configuration Register 0, in hex
- CNF1: Configuration Register 1, in hex
- CNF2: Configuration Register 2, in hex

Figura 11. Resultados de la calculadora “kvaser”.

En la figura 11 se muestran los resultados del cálculo hecho por la calculadora “kvaser”. Es recomendable utilizar un BTQ (número de *quantas* en un bit) lo más aproximado a 2, con el objetivo de aumentar la resolución del punto de muestreo. Otro aspecto importante, es que el SJW(*Synchronization Jump With*) debe ser lo mas alto posible. De lo anterior se deduce que el valor para los registros de configuración *BRGCON1, BRGCON2 y BRGCON3* son equivalentes a los registros *CNF1, CNF2 y CNF3* que se encuentran en el renglón cuatro de la figura anterior.

5.3.5 Mascaras y filtros de aceptación

Para configurar los filtros y máscaras de aceptación de nuestros buffers de recepción se debe configurar el modo de recepción; esto se logra configurando los bits *RXM* que se encuentran en el registro *RXB0CON*. En el caso del programa de ejemplo que se está describiendo, solo recibe mensajes estándar que pasen el criterio de filtrado. Por otra parte, se tienen filtros en los buffers de recepción 0 y 1, así como la máscara de aceptación cero que se encuentra asociada al buffer de recepción 0.

Los registros involucrados son los siguientes:

REGISTER 23-13: RXB0CON: RECEIVE BUFFER 0 CONTROL REGISTER

	R/C-0	R/W-0	R/W-0	U-0	R-0	R/W-0	R-0	R-0
Mode 0	RXFUL(1)	RXM1	RXM0	—	RXRTRRO	RXB0DBEN	JTOFF(2)	FILHIT0

bit 6 Mode 0:
RXM1: Receive Buffer Mode bit 1
Combines with RXM0 to form RXM<1:0> bits (see bit 5).
11 = Receive all messages (including those with errors); filter criteria is ignored
10 = Receive only valid messages with extended identifier; EXIDEN in RXFnSIDL must be '1'
01 = Receive only valid messages with standard identifier; EXIDEN in RXFnSIDL must be '0'
00 = Receive all valid messages as per EXIDEN bit in RXFnSIDL register
Mode 1, 2:
RXM1: Receive Buffer Mode bit 1
1 = Receive all messages (including those with errors); acceptance filters are ignored
0 = Receive all valid messages as per acceptance filters

bit 5 Mode 0:
RXM0: Receive Buffer Mode bit 0
Combines with RXM1 to form RXM<1:0> bits (see bit 6).

Figura 12. Registro RXB0CON.

REGISTER 23-37: RXFnSIDH: RECEIVE ACCEPTANCE FILTER n STANDARD IDENTIFIER FILTER REGISTERS, HIGH BYTE [0 ≤ n ≤ 15](1)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

bit 7-0 **SID10:SID3:** Standard Identifier Filter bits (if EXIDEN = 0)
Extended Identifier Filter bits EID28:EID21 (if EXIDEN = 1).

Note 1: Registers RXF6SIDH:RXF15SIDH are available in Mode 1 and 2 only.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 23-38: RXFnSIDL: RECEIVE ACCEPTANCE FILTER n STANDARD IDENTIFIER FILTER REGISTERS, LOW BYTE [0 ≤ n ≤ 15](1)

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDEN(2)	—	EID17	EID16
bit 7							bit 0

bit 7-5 **SID2:SID0:** Standard Identifier Filter bits (if EXIDEN = 0)
Extended Identifier Filter bits EID20:EID18 (if EXIDEN = 1).

bit 4 **Unimplemented:** Read as '0'

bit 3 **EXIDEN:** Extended Identifier Filter Enable bit(2)
1 = Filter will only accept extended ID messages
0 = Filter will only accept standard ID messages

bit 2 **Unimplemented:** Read as '0'

bit 1-0 **EID17:EID16:** Extended Identifier Filter bits

Figura 13. Registros RXFnSIDH y RXFnSIDL

REGISTER 23-41: RXMnSIDH: RECEIVE ACCEPTANCE MASK n STANDARD IDENTIFIER MASK REGISTERS, HIGH BYTE [0 ≤ n ≤ 1]

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

bit 7-0 **SID10:SID3:** Standard Identifier Mask bits or Extended Identifier Mask bits EID28:EID21

Figura 14. Registro RXMnSIDH

REGISTER 23-42: RXMnSIDL: RECEIVE ACCEPTANCE MASK n STANDARD IDENTIFIER MASK REGISTERS, LOW BYTE [0 ≤ n ≤ 1]

R/W-x	R/W-x	R/W-x	U-0	R/W-0	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDEN(1)	—	EID17	EID16
bit 7							bit 0

bit 7-5 **SID2:SID0:** Standard Identifier Mask bits or Extended Identifier Mask bits EID20:EID18

bit 4 **Unimplemented:** Read as '0'

bit 3 Mode 0:
Unimplemented: Read as '0'
Mode 1, 2:
EXIDEN: Extended Identifier Filter Enable Mask bit(1)
1 = Messages selected by the EXIDEN bit in RXFnSIDL will be accepted
0 = Both standard and extended identifier messages will be accepted

Note 1: This bit is available in Mode 1 and 2 only.

bit 2 **Unimplemented:** Read as '0'

bit 1-0 **EID17:EID16:** Extended Identifier Mask bits

Figura 15. Registro RXMnSIDL

Dichos registros quedaron programados de la siguiente manera:

```
RXB0CONbits.RXM0=1;           //receive only valid messages with
standard ID

RXF0SIDH=0b00111111;              //Acceptance    filter0    for
buffer 0
RXF0SIDL=0b11100000;              //fourth bit must be zero for
only standard identifiers

RXF1SIDH=0b11000000;              //Acceptance   filter   1   for
buffer 0
RXF1SIDL=0b00000000;              //fourth bit must be zero for
only standard identifiers


RXM0SIDH=0b11111111;          //Acceptance   filter   mask   for

buffer 0 (all zero to accept all
```

Código de programa 2. Filtros de aceptación.

La recepción de los mensajes, una vez configurados los filtros y máscaras de aceptación se generará de manera automática, cada vez que un mensaje pase por su respectivo arbitraje.

5.3.6 Transmisión CAN

Para poder comenzar a transmitir un mensaje, primero se debe verificar que el bit *TXREQ* este apagado, de esta manera se verifica que el buffer de transmisión está disponible. Posteriormente se deben cargar los buffers de datos, DLC y por ultimo el ID del mensaje a enviar. El código queda de la siguiente manera:

```
while(TXB0CONbits.TXREQ); // wait for TXREQ to clear, it could be not
neccesary

TXB0D0 =    0b00001000; // first byte
TXB0D1 =    0b00000111; // second byte

TXB0SIDH = 0b00001100; // set the ID
TXB0SIDL = 0b10000000;
TXB0DLC = 0b00000010; // set DLC = 2
TXB0CON = 0b00001000; // request to transmit the message in buffer 0
(mark it for transmission)

while(TXB0CONbits.TXREQ) //wait until transmission is complete
```

Código de programa 3. Transmisión CAN

Como se observa en las líneas de código anteriores, una vez que se cargan los buffers antes mencionados, es necesario encender el bit *TXREQ* en el registro *TXB0CON* para comenzar con la

transmisión. Es buena práctica esperar a que el bit *TXREQ* se apague para determinar cuando el mensaje ha sido transmitido.

5.3.7 Vector de interrupciones

Como se mencionó anteriormente, se utiliza el vector de interrupción de alta prioridad, en el proyecto que se esta manejando como ejemplo la *rutina de servicio a la interrupción* (ISR), enciente y apaga un led cada vez que envía o recibe un mensaje [8].

El registro de interrupción es el siguiente:

REGISTER 23-56: PIR3: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 3

Mode 0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IRXIF	WAKIF	ERRIF	TXB2IF	TXB1IF(1)	TXB0IF(1)	RXB1IF	RXB0IF

Figura 16. Registro PIR3.

Como se puede observar en la figura anterior, las banderas de interrupción que se están utilizando para el buffer de transmisión 0 y el buffer de recepción 0 se encuentran localizadas en el registro PIR3. En nuestro código el llamado se hace de la siguiente manera:

```
void my_isr(void);         //is  declared  cause  is  used  in  the
interrupt service rutine

#pragma code high_vector=0x08    //high priority vector
void interrupt_at_high_vector(void)
{
      _asm GOTO my_isr _endasm
}
#pragma code          //return to the default code section
#pragma interrupt my_isr
void my_isr (void)
{
      if(PIR3bits.TXB0IF)
      {
            PIR3bits.TXB0IF=0;
            LATDbits.LATD4=!LATDbits.LATD4 ; // we get a TX interrupt
      }

      if(PIR3bits.RXB0IF)
      {
            PIR3bits.RXB0IF=0;
            if(RXB0SIDH==IDH0 && RXB0SIDL==IDL0)
            {
                  if((RXB0D0&0x21)==0x21)
                  {
                        LATDbits.LATD5 = !LATDbits.LATD5 ;
                  }
            }
            RXB0CONbits.RXFUL=0;  //it  is  cleared  until  the  final  to
allow a new message be received
      }
}                                 //end of my_isr
```

Código de programa 4. Vector de interrupciones.

En la figura 17 que se muestra a continuación, se presenta el principio de operación del módulo CAN con el que cuenta el microcontrolador. También se pueden observar los diferentes buffers de recepción y transmisión del protocolo, así como sus respectivos filtros y máscaras de aceptación.

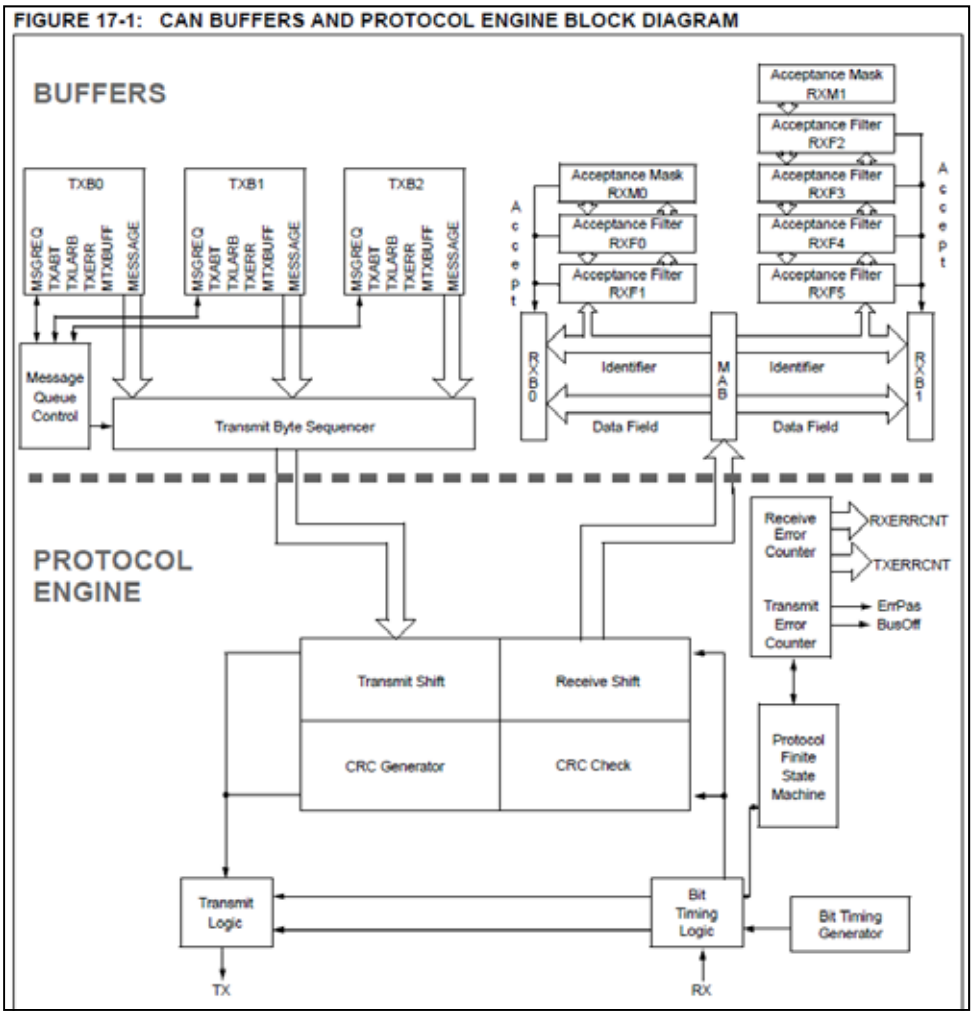


Figura 17. Diagrama a bloques del principio de operación del módulo CAN.

5.4 Módulo ECAN del microcontrolador.

El módulo ECAN, comparte varias de las características del modulo CAN. En el microcontrolador PIC18F4680 ambas versiones se encuentran en un mismo módulo. Como se mencionó anteriormente el módulo CAN es 100% compatible con el módulo ECAN.

5.4.1 Características del módulo ECAN.

Las características del módulo ECAN del microcontrolador fueron descritas en el capítulo 3 es por esto que no se presentan en este capítulo.

Es importante mencionar que a partir de este momento el módulo CAN pasará a llamarse modo 0. Esto es así, ya que el módulo ECAN de nuestro microcontrolador tiene la opción de operar en este modo de compatibilidad con el módulo CAN. Los modos de operación que tenemos disponibles en el microcontrolador son los siguientes:

- Mode 0 – Legacy mode
- Mode 1 – Enhanced Legacy mode with DeviceNet support
- Mode 2 – FIFO mode with DeviceNet support

El modo de operación a usar será el modo 1, el cual es un modelo mejorado del módulo CAN o modo 0.

Lo anterior se resume a las siguientes líneas de código:

```
CANCONbits.REQOP2 = 1;        // request configuration mode
          while(!CANSTATbits.OPMODE2);  //  wait  until  we
can begin initialization
     LATDbits.LATD0 = 1;               //we   are   now   in
configuration mode
     ECANCON=0b01000000;           //(MODO 1)
```

Código de programa 5. Modo 1 Request.

5.4.2 Baud rate configuration

La configuración del *baud rate* se lleva a cabo de la misma forma que en el modo 0.

5.4.3 Filtros y mascaras de aceptación

En el modo 1 se cuenta con 16 filtros de aceptación, dichos filtros pueden ser asociados a cualquiera de los 8 buffers de recepción que se tienen. Para esto, primero es necesario habilitar los buffers que se utilizarán y consecuentemente asociarlos al buffer de recepción deseado. Además es necesario configurar los 6 buffers *RX/TX* como se requiera

En las figuras 18 y 19 que se presentan a continuación, se puede observar las configuraciones para los registros *BSEL0, RXFCONn y RXFBCONn* (0>=n<=7).

REGISTER 23-36: BSEL0: BUFFER SELECT REGISTER 0(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0
B5TXEN	B4TXEN	B3TXEN	B2TXEN	B1TXEN	B0TXEN	—	—
bit 7							bit 0

bit 7-2 **B5TXEN:B0TXEN:** Buffer 5 to Buffer 0 Transmit Enable bit
1 = Buffer is configured in Transmit mode
0 = Buffer is configured in Receive mode

bit 1-0 **Unimplemented:** Read as '0'

Note 1: This register is available in Mode 1 and 2 only.

Figura 18. Registro BSEL0.

REGISTER 23-45: RXFCONn: RECEIVE FILTER CONTROL REGISTER n [0 ≤ n ≤ 1](1)

	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RXFCON0	RXF7EN	RXF6EN	RXF5EN	RXF4EN	RXF3EN	RXF2EN	RXF1EN	RXF0EN
	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
RXFCON1	RXF15EN	RXF14EN	RXF13EN	RXF12EN	RXF11EN	RXF10EN	RXF9EN	RXF8EN
	bit 7							bit 0

bit 7-0 **RXFnEN:** Receive Filter n Enable bits

0 = Filter is disabled

1 = Filter is enabled

Note 1: This register is available in Mode 1 and 2 only.

Figura 19. Registro RXFCONn

REGISTER 23-47: RXFBCONn: RECEIVE FILTER BUFFER CONTROL REGISTER n(1)

	bit 7							bit 0
RXFBCON0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	F1BP_3	F1BP_2	F1BP_1	F1BP_0	F0BP_3	F0BP_2	F0BP_1	F0BP_0
RXFBCON1	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-1
	F3BP_3	F3BP_2	F3BP_1	F3BP_0	F2BP_3	F2BP_2	F2BP_1	F2BP_0
RXFBCON2	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-1
	F5BP_3	F5BP_2	F5BP_1	F5BP_0	F4BP_3	F4BP_2	F4BP_1	F4BP_0
RXFBCON3	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	F7BP_3	F7BP_2	F7BP_1	F7BP_0	F6BP_3	F6BP_2	F6BP_1	F6BP_0
RXFBCON4	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	F9BP_3	F9BP_2	F9BP_1	F9BP_0	F8BP_3	F8BP_2	F8BP_1	F8BP_0
RXFBCON5	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	F11BP_3	F11BP_2	F11BP_1	F11BP_0	F10BP_3	F10BP_2	F10BP_1	F10BP_0
RXFBCON6	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	F13BP_3	F13BP_2	F13BP_1	F13BP_0	F12BP_3	F12BP_2	F12BP_1	F12BP_0
RXFBCON7	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	F15BP_3	F15BP_2	F15BP_1	F15BP_0	F14BP_3	F14BP_2	F14BP_1	F14BP_0

bit 7-0 **FnBP_3:FnBP_0:** Filter n Buffer Pointer Nibble bits

0000 = Filter n is associated with RXB0
0001 = Filter n is associated with RXB1
0010 = Filter n is associated with B0
0011 = Filter n is associated with B1
...
0111 = Filter n is associated with B5
1111-1000 = Reserved

Note 1: This register is available in Mode 1 and 2 only.

Figura 20. Registro RXFBCONn (0>n<7)

Una vez que se han habilitado, configurado y asociado los filtros a sus respectivos buffers, es momento de especificar los filtros deseados. En el proyecto se hizo de la siguiente manera:

```
RXF0SIDH=0b00001100;              //Acceptance filter0 for RXB0    (064)
RXF0SIDL=0b10000000;              //fourth bit must be zero for only
standard identifiers
RXF1SIDH=0b10101110;              //Acceptance filter1 for RXB1    (575)
RXF1SIDL=0b10100000;              //fourth bit must be zero for only
standard identifiers
RXF2SIDH=0b10001110;              //Acceptance filter2 for B0      (470)
RXF2SIDL=0b00000000;              //fourth bit must be zero for only
standard identifiers
RXF3SIDH=0b10110010;              //Acceptance filter3 for B1      (591)
RXF3SIDL=0b00100000;              //fourth bit must be zero for only
standard identifiers
RXF4SIDH=0b10010111;              //Acceptance filter4 for B2      (4BD)
RXF4SIDL=0b10100000;              //fourth bit must be zero for only
standard identifiers
```

Código de programa 6. Asociación de os filtros de aceptación.

En éste proyecto todos los filtros de aceptación fueron asociados a una sola máscara. Dicha máscara es la máscara de aceptación cero. Para poder hacer la asociación se programan dos registros, el registro *MSEL0 y MSEL1*. A continuación se presentan las líneas de código del programa que asocian la mascara 0 con los 7 filtros de aceptación utilizados, así como las especificaciones de cada uno de los registros mencionados.

```
MSEL0=0b00000000;     //Mask Zero is sellected for the seven filters
MSEL1=0b00000000;
```

Código de programa 7. Asignación de las máscaras de aceptación.

REGISTER 23-48: MSEL0: MASK SELECT REGISTER 0(1)

R/W-0	R/W-1	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
FIL3_1	FIL3_0	FIL2_1	FIL2_0	FIL1_1	FIL1_0	FIL0_1	FIL0_0
bit 7							bit 0

bit 7-6 **FIL3_1:FIL3_0:** Filter 3 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 5-4 **FIL2_1:FIL2_0:** Filter 2 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 3-2 **FIL1_1:FIL1_0:** Filter 1 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 1-0 **FIL0_1:FIL0_0:** Filter 0 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

Note 1: This register is available in Mode 1 and 2 only.

Figura 21. Registro MSEL0.

REGISTER 23-49: MSEL1: MASK SELECT REGISTER 1(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-1
FIL7_1	FIL7_0	FIL6_1	FIL6_0	FIL5_1	FIL5_0	FIL4_1	FIL4_0
bit 7							bit 0

bit 7-6 **FIL7_1:FIL7_0:** Filter 7 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 5-4 **FIL6_1:FIL6_0:** Filter 6 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 3-2 **FIL5_1:FIL5_0:** Filter 5 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

bit 1-0 **FIL4_1:FIL4_0:** Filter 4 Select bits 1 and 0
11 = No mask
10 = Filter 15
01 = Acceptance Mask 1
00 = Acceptance Mask 0

Note 1: This register is available in Mode 1 and 2 only.

Figura 22. Registro MSEL1.

Después de programar los modos de operación, *baud rate*, filtros de aceptación y máscaras de aceptación, es salir del modo de configuración y entrar al modo normal. Al entrar en modo normal, el microcontrolador se encuentra listo para comenzar a enviar y recibir mensajes. Lo anterior se logra con las siguientes líneas de código:

```
CANCONbits.REQOP2 = 0;          //request normal mode
CANCONbits.REQOP0 = 0;
CANCONbits.REQOP1 = 0;
while(CANSTATbits.OPMODE2);
```

Código de programa 8

Una vez que se ha salido del *while*, se puede asegurar que se encuentra en modo normal. En el modo normal, se puede configurar las interrupciones del dispositivo, esto es muy obvio ya que no podemos habilitar interrupciones de buffers que no estén previamente habilitados, es por esto que hasta este punto es cuando está permitido habilitarlas.

5.4.4 Transmisión y recepción del módulo ECAN

La transmisión y recepción del módulo ECAN se identifica por tener disponibles 6 buffers extras independientes delos que normalmente se tienen en el modo 0 o módulo CAN. Estos buffers pueden ser configurados de cualquiera de las dos formas; recepción o transmisión.

El proceso de transmisión y recepción se lleva a cabo de la misma manera que en el modo 0. En el caso de éste programa, la recepción se esta controlando por medio de interrupciones. En el proyecto se están habilitando los 6 buffers configurables como recepción, aparte de los dos buffers que se tienen por default. La rutina de servicio a la interrupción se muestra en el fragmento de código a continuación:

```
if(B1CONbits.RXFUL){
                        LATDbits.LATD3 = !LATDbits.LATD3 ;
                        //******rutina para almacenar un byte en la memoria******//
                //      HDByteWriteI2C(ControlByte, ADDRESS.HADR , ADDRESS.LADR, B1D0 );
//High Density Byte Write
```

Código de programa 9. Rutina de servicio a la interrupción ECAN module.

Como se puede observar, en el fragmento de código mostrado anteriormente se atiende a la interrupción generada por el buffer de recepción RXB0.

El apagado del bit RXFUL se hace hasta el final, con la finalidad de que no se reciba ningún mensaje mientras el buffer se encuentre ocupado.

Hasta este punto, ya se han explicado los aspectos más importantes de la programación del módulo ECAN. Los códigos completos, tanto del módulo CAN como del módulo ECAN se presentan en la sección de anexos de este documento.

Al dominar los aspectos de programación del módulo CAN, se es capaz de enviar o recibir cualquier dato dentro de una red. Además, al establecer la comunicación con nuestro microcontrolador, podemos realizar cualquier tarea requerida, y que pueda ser desarrollada por un microcontrolador. Entre las tareas que se pueden realizar con el microcontrolador sobresalen las siguientes:

- PWM
- ADC
- Comunicación I2C
- Comunicación ISP
- *Timers*
- USART
- I/0 Ports
- Control de *display* de cristal liquido
- Acoplar etapas de potencia
- Comparadores

- Manejo de estructuras
-

Se hace mención de estas características o módulos que tiene el microcontrolador, ya que se busca dar a entender la importancia de poder establecer una comunicación de alto nivel como lo es CAN y poder integrarla con cualquiera de las funciones anteriores. Además, durante el desarrollo de este proyecto se aprendió a utilizar las librerías, lo cual hace mucho más sencillo poder integrar cualquiera de los módulos del microcontrolador en un proyecto.

En el caso de éste proyecto, fue necesario hacer una integración de los módulos siguientes:

- CAN
- *Timers*
- *Delays*
- EUSART
- I2C
- I/0 Ports
- Manejo de estructuras

Como se puede observar, no solo fue necesario aprender a utilizar el módulo CAN, sino que también se necesitó dedicar parte del tiempo en aprender a utilizar los otros módulos con los que cuenta el microcontrolador.

5.5 Interfaz de comunicación I2C protocolo de comunicación de la memoria EEPROM

El nodo CAN cuenta con una memoria EEPROM. Esta memoria fue implementada en el nodo con el objetivo de almacenar los mensajes requeridos por la red CAN y después poder descargarlos a una PC.

La memoria que se eligió fue la 24LC512 I^2C^{TM} CMOS Serial EEPROM, ésta memoria cuenta con 512Kbit de almacenamiento, utiliza un bus I2C para la comunicación con el dispositivo maestro, y puede manejar velocidades de transmisión de hasta 400KHZ.

La memoria EEPROM establece una comunicación I2C con el dispositivo maestro. El módulo I2C se estudió de una manera profunda, con el objetivo de poder utilizar dicha memoria.

El módulo I2C con el que cuenta éste microcontrolador implementa todas las funciones *maestro – esclavo* y provee interrupciones por hardware en los bits de *Start* y *Stop* para así poder determinar cuando el bus se encuentra libre. Los dos pines que utiliza el protocolo para establecer la transferencia de datos, son los siguientes:

- Serial clock (SCL) – RC3/SCK/SCL

- Serial data (SDA) – RC4/SDI/SDA

El pin SCL se utiliza para enviar la señal de reloj del maestro y el pin SDA para la transferencia de los datos.

La secuencia de trabajo que se utilizó para poder tratar con este nuevo protocolo, fue leer la hoja de datos del microcontrolador y hacer programas básicos de comunicación, como lo es escribir un byte en la memoria, leer un byte de la memoria, escribir páginas y leer páginas. Esto se realizó investigando como configurar cada uno de los registros y generando las interrupciones pertinentes para atender a las distintas tareas de comunicación. El código de dichas pruebas de aprendizaje se proporciona en el apartado de anexos de este documento.

Al momento de investigar más acerca del uso del compilador C18, se aprendió que dicho compilador contaba con librerías especializadas para el manejo del protocolo I^2C, así como la comunicación con memorias EEPROM I^2C.

La memoria EEPROM cuenta con una serie de secuencias de bytes necesarias para poder ser utilizada, es decir la memoria I^2C tiene un FIRMWARE integrado por medio del cual se puede establecer la comunicación con el maestro.

Las funciones utilizadas en el proyecto son las siguientes:

- Control Byte
- HighAdd

- LowAdd
- Data (en caso de escritura)

Las especificaciones de cada uno de estos bytes se encuentran en la hoja de datos de la memoria.

Después de investigar y estudiar a fondo el compilador C18, se encontró la manera de manejar la variable de la manera requerida, esto se logró con el uso de las estructuras que se manejan en dicho compilador.

Dentro de las estructuras se presentan las llamadas uniones. En las uniones se almacenan los campos solapándose unos con otros en la misma disposición. Las uniones se implementan principalmente para llamar de dos maneras distintas al mismo sector de memoria [5].

A continuación se presenta un ejemplo del uso que tuvo la unión del proyecto al momento de dar la dirección en memoria de nuestra EEPROM.

```
union{
     unsigned int DIR;
          struct{
               unsigned char LADR:8;
               unsigned char HADR:8;
          }
}ADDRESS;
```

Código de programa 11. Unión.

Como se observa en el fragmento de código anterior, se reservan 16 bits en RAM para almacenar una variable del tipo *int* (DIR), de la cual se puede tener acceso a bytes alto y bajo de forma independiente. Para acceder a sus miembros se utiliza el operador punto como se muestra a continuación:

```
ADDRESS.LADR=0x00;
ADDRESS.HADR=0x00;
```

Código de programa 12. Acceso a los miembros de la unión.

En la figura 23que se muestra a continuación se observa la estructura de la unión con sus respectivos miembros.

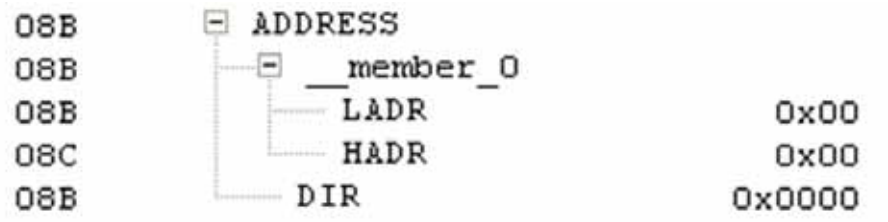


Figura 23. Unión ADDRESS en el *watch* de MPLAB IDE

Al tener la variable de 16 bits y poder acceder a sus bytes alto y bajo, fue muy sencillo utilizar las funciones de la librería I^2C.

Las funciones quedaron programadas de la siguiente manera:

```
HDByteWriteI2C(ControlByte, ADDRESS.HADR , ADDRESS.LADR, RXB0SIDH );
ADDRESS.DIR++;
```

Código de programa 13. Escritura de un byte en la memoria EEPROM.

En el fragmento de código anterior se muestra como en los parámetros de la función se escriben la parte alta y baja de la unión ADRESS y posteriormente se incrementa en 1 la variable DIR de 16 bits. Como resultado se tiene una librería que facilita la comunicación con la memoria EEPROM y una unión que permite un manejo versátil de los datos que se envían como dirección de la memoria al momento de escribir o leer la misma.

Los bits RC3 y RC4 son los que implementa la comunicación I^2C y se configuran como entradas. El registro *SSPCON1* se utiliza para habilitar el puerto serial y configurar los pines SDA y SCL como pines del puerto serial, así como seleccionar el modo de operación del protocolo, en este caso el microcontrolador debe ser configurado en modo maestro. Por último se resetean los bits de estatus en el registro *SSPSTAT*, los bits de control en el registro *SSPCON2* y se configura la velocidad de transmisión en el registro *SSPAD*. A continuación se presentan los registros involucrados en la configuración del modulo MSSP en modo I^2C.

REGISTER 17-3: SSPSTAT: MSSP STATUS REGISTER (I^2C MODE)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Slew Rate Control bit

In Master or Slave mode:

1 = Slew rate control disabled for Standard Speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for High-Speed mode (400 kHz)

bit 6 **CKE:** SMBus Select bit

In Master or Slave mode:

1 = Enable SMBus specific inputs
0 = Disable SMBus specific inputs

bit 5 **D/$\overline{A}$:** Data/$\overline{Address}$ bit

In Master mode:

Reserved.

In Slave mode:

1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit

1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last

Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 3 **S:** Start bit

1 = Indicates that a Start bit has been detected last
0 = Start bit was not detected last

Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 2 **R/$\overline{W}$:** Read/$\overline{Write}$ bit Information (I^2C mode only)

In Slave mode:

1 = Read
0 = Write

Note: This bit holds the R/$\overline{W}$ bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit or not $\overline{ACK}$ bit.

In Master mode:

1 = Transmit is in progress
0 = Transmit is not in progress

Note: ORing this bit with SEN, RSEN, PEN, RCEN or ACKEN will indicate if the MSSP is in Idle mode.

bit 1 **UA:** Update Address bit (10-bit Slave mode only)

1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit

In Transmit mode:

1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty

In Receive mode:

1 = Data transmit in progress (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is full
0 = Data transmit complete (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is empty

Figura 24. Registro SSPSTAT.

REGISTER 17-4: SSPCON1: MSSP CONTROL REGISTER 1 (I²C MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit

In Master Transmit mode:

1 = A write to the SSPBUF register was attempted while the I²C conditions were not valid for a transmission to be started (must be cleared in software)

0 = No collision

In Slave Transmit mode:

1 = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)

0 = No collision

In Receive mode (Master or Slave modes):

This is a "don't care" bit.

bit 6 **SSPOV:** Receive Overflow Indicator bit

In Receive mode:

1 = A byte is received while the SSPBUF register is still holding the previous byte (must be cleared in software)

0 = No overflow

In Transmit mode:

This is a "don't care" bit in Transmit mode.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit

1 = Enables the serial port and configures the SDA and SCL pins as the serial port pins

0 = Disables serial port and configures these pins as I/O port pins

Note: When enabled, the SDA and SCL pins must be properly configured as input or output.

bit 4 **CKP:** SCK Release Control bit

In Slave mode:

1 = Release clock

0 = Holds clock low (clock stretch), used to ensure data setup time

In Master mode:

Unused in this mode.

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits

1111 = I²C Slave mode, 10-bit address with Start and Stop bit interrupts enabled

1110 = I²C Slave mode, 7-bit address with Start and Stop bit interrupts enabled

1011 = I²C Firmware Controlled Master mode (Slave Idle)

1000 = I²C Master mode, clock = Fosc/(4 * (SSPADD + 1))

0111 = I²C Slave mode, 10-bit address

0110 = I²C Slave mode, 7-bit address

Note: Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.

Figura 25. Registro SSPCON1.

REGISTER 17-5: SSPCON2: MSSP CONTROL REGISTER 2 (I^2C MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN(1)	RCEN(1)	PEN(1)	RSEN(1)	SEN(1)
bit 7							bit 0

bit 7 **GCEN:** General Call Enable bit (Slave mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT:** Acknowledge Status bit (Master Transmit mode only)
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (Master Receive mode only)
1 = Not Acknowledge
0 = Acknowledge

Note: Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (Master Receive mode only)(1)
1 = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence Idle

bit 3 **RCEN:** Receive Enable bit (Master mode only)(1)
1 = Enables Receive mode for I^2C
0 = Receive Idle

bit 2 **PEN:** Stop Condition Enable bit (Master mode only)(1)
1 = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Stop condition Idle

bit 1 **RSEN:** Repeated Start Condition Enable bit (Master mode only)(1)
1 = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated Start condition Idle

bit 0 **SEN:** Start Condition Enable/Stretch Enable bit(1)
In Master mode:
1 = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Start condition Idle
In Slave mode:
1 = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
0 = Clock stretching is disabled

Note 1: For bits ACKEN, RCEN, PEN, RSEN, SEN: If the I^2C module is not in the Idle mode, these bits may not be set (no spooling) and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

Figura 26. Registro SSPCON2.

La configuración de la velocidad de transmisión de la comunicación I^2C, se lleva a cabo cargando un valor en el registro *SSPADD*. El valor asignado al registro *SSPADD* se calcula con la siguiente fórmula:

$$SSPADD = \frac{\frac{Fosc}{BitRate}}{4} - 1 \qquad (5.1)$$

Donde:

Fosc = Frecuencia de oscilación del reloj maestro.

BitRate = Velocidad de transmisión deseada.

5.6 Interfaz de comunicación USART

USART por sus siglas en inglés *Universal Synchronous Asynchronous Receiver Transmitter* es un protocolo de comunicación serial que puede ser configurado como un sistema asíncrono "full –duplex" con la capacidad de comunicarse con dispositivos periféricos, tales como terminales CRT y computadoras personales [8].

Una vez que se logró implementar la comunicación con el módulo ECAN y la comunicación entre la memoria y el microcontrolador por medio del protocolo I^2C, la siguiente tarea fue habilitar una comunicación serial USART. El objetivo principal de contar con una comunicación USART es poder enviar los datos de la red CAN almacenados en la memoria EEPROM hacia la PC.

Al igual que con la comunicación I^2C, fue necesario hacer un estudio completo del protocolo, para así poder programar cada uno de los registros de configuración que se manejan

en dicho protocolo. Los registros de configuración involucrados se reunieron en una función llamada USART_init(), dicha función y los registros de configuración se presentan a continuación:

```
void USART_init(void){
     RCSTAbits.SPEN = 1;    //habilito el puerto serial pines RC7 y RC6 como puertos seriales
     TXSTAbits.BRGH = 1;    //high speed
     TXSTAbits.TXEN = 0;    //transmision detenida!!!  hasta que este todo listo
     TXSTAbits.SYNC = 0;    //modo de TX asincrona
     TXSTAbits.TX9  = 0;    //no quiero 9 bits sino 8 bits como la deshabilite no es necesario configurar la recepcion que por default esta en 0
     //SPBRG=162;              //transmision a 9600bps ASINCRONA en high speed (25MHz)
     SPBRG=155;              //(24MHz)
}
```

Código de programa 14. Método de inicialización del módulo USART.

REGISTER 18-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled (held in Reset)

bit 6 **RX9:** 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

bit 4 **CREN:** Continuous Receive Enable bit
Asynchronous mode:
1 = Enables receiver
0 = Disables receiver
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and loads the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit
Asynchronous mode 9-bit (RX9 = 0):
Don't care.

bit 2 **FERR:** Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receiving next valid byte)
0 = No framing error

bit 1 **OERR:** Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data bit
This can be address/data bit or a parity bit and must be calculated by user firmware.

Figura 27. Registro RCSTA.

REGISTER 18-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care.
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9:** 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN:** Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled
Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC:** EUSART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB:** Send Break Character bit
Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed
Synchronous mode:
Don't care.

bit 2 **BRGH:** High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode.

bit 1 **TRMT:** Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data
Can be address/data bit or a parity bit.

Figura 28. Registro TXSTA.

Los registros involucrados en la configuración de la transmisión USART son 3, *el RCSTA, TXSTA y el SPBRG*. Básicamente lo que se configura en ellos es la habilitación de los puertos RC7 yRC6 como puertos seriales; el modo *high speed* que asegura que la transmisión este detenida hasta que se tenga todo previamente configurado; también se determina el modo de

configuración asíncrona, se establece la transmisión de una cadena de 8 bits de datos y por último, se carga el registro *SPBRG* con un valor calculado para la velocidad de transferencia deseada.

Éste valor fue calculado a partir de la siguiente formula:

$$SPBRG = \frac{\frac{Fosc}{BaudRate}}{16} - 1 \qquad (5.2)$$

Dónde:

Fosc = Frecuencia de oscilación del reloj maestro.

BaudRate = Tasa de transmisión deseada.

Por otra parte, como en el método de interrupciones está habilitando la interrupción de este módulo, se utiliza el registro TXREG para escribir el byte a enviar cada vez que dicho buffer se encuentre libre. En el caso de éste proyecto la interrupción del módulo USART se utiliza para enviar el byte leído de la memoria EEPROM.

Lo anterior es posible escribiendo el valor del *SSPBUF*; que es un registro que recibe el byte de la memoria al momento de ser leída en el registro *TXREG*. A continuación se muestra el fragmento de código que realiza esta operación.

```
if(PIR1bits.TXIF)
     {
          //PIR1bits.TXIF=0;
          TXREG=SSPBUF;
          while(!TXSTAbits.TRMT);
          TXSTAbits.TXEN=0;
     }
```

Código de programa 15. Método de interrupción del USART.

Como se mencionó anteriormente, cada vez que el buffer *TXREG* se encuentra disponible, la interrupción del módulo se presenta, en este momento es cuando se pasa el valor del *SSPBUF* al *TXREG*; el while() asegura que el dato sea enviado antes de detener el módulo USART en la siguiente línea. El módulo se detiene cada vez que se envía un mensaje, para así poder dar el control de la transmisión a la lectura de la memoria puesto que en el momento que se lee un byte de ésta, también se habilita el módulo USART. La lectura de la memoria se realiza con las líneas de código siguientes:

```
if(PORTBbits.RB5==0)
     {
          //SSPCON1|=0x28;
          numadr=ADDRESS.DIR;
          ADDRESS.DIR=0;

          /** Retardo necesario para evitar el rebote del boton**/
          Delay1KTCYx(10000000000000);
          Delay1KTCYx(10000000000000);
          Delay1KTCYx(10000000000000);
          Delay1KTCYx(10000000000000);
          Delay1KTCYx(10000000000000);
          Delay1KTCYx(10000000000000);
          Delay1KTCYx(10000000000000);

          //   BIE0=0b00000000;    //disable all the interruptions
          for(n=0; n<numadr; n++)
               {
                    LowAdd=ADDRESS.LADR;
                         HighAdd=ADDRESS.HADR;
                    HDByteReadI2C(ControlByte, HighAdd, LowAdd , &Data,
                                  Length) ;
                    ADDRESS.DIR++;
                    TXSTAbits.TXEN=1;
               }


               ADDRESS.DIR=0;
          //   BIE0=0b11111111;     //enable all the interruptions of
the CAN buffers
          }
     }
```

Código de programa 16. Método de lectura de la memoria.

Como se muestra en las líneas de código anteriores, al momento de leer una entrada digital del microcontrolador accionada por medio de un *push-button*, se genera un retardo con el fin de evitar el rebote del botón. Posteriormente se deshabilitan las interrupciones de los buffers del módulo ECAN con la finalidad de no interrumpir la transmisión y evitar la pérdida de sincronización del módulo USART.