IMPLEMENTACIÓN DE UNA RED INDUSTRIAL CAN BAJO UN SISTEMA SCADA

JULIÁN ANDRÉS VIDAL ILLERA
MILTON SERGIO ZÚÑIGA GALINDEZ

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
POPAYÁN
2006

IMPLEMENTACIÓN DE UNA RED INDUSTRIAL CAN BAJO UN SISTEMA SCADA

JULIÁN ANDRÉS VIDAL ILLERA
MILTON SERGIO ZÚÑIGA GALINDEZ

Anexo A

Diseño y Configuración del Software de Programación para cada PIC18F258 del Nodo Maestro y los Nodos Esclavos

Director
OSCAR AMAURY ROJAS ALVARADO
Ingeniero Electrónico Especialista en Informática Industrial

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
POPAYÁN
2006

CONTENIDO

1 DISEÑO Y CONFIGURACIÓN DEL SOFTWARE DE PROGRAMACIÓN PARA CADA PIC18F258 DEL NODO MAESTRO Y LOS NODOS ESCLAVOS

Para la programación de cada PIC18F258 se hace uso del software MPLAB proporcionado por la empresa Microchip. El programa esta implementado haciendo uso del lenguaje de programación ANSI C. Las liberarías necesarias para la programación son generadas por el software de la Aplicación "*Maestro*". MPLAB se encarga de compilar el programa y generar el código hexadecimal, útil por ser el lenguaje que entiende el PIC.

Para hacer mas legible el código se deben tener en cuenta las siguientes disposiciones: las funciones, las constantes y las variables se escribirán con negrilla, los comentarios con letra cursiva seguidos de la doble barra (//) y los argumentos y salidas con letra normal. A continuación se presenta el código del nodo maestro, del nodo esclavo 1 y por último del nodo esclavo 2.

1.1 NODO MAESTRO

// **SECCIÓN 1**

```
1. #include  <p18f258.h>          // Incluir la librería del PIC
2. #include <adc.h>               // Incluir la librería para las funciones ADC
3. #include <delays.h>            // Incluir la librería para las funciones "delays"
4. #include "UARTIntC.h"          // Incluir la librería para las funciones USART
5. #include "can.h"               // Incluir la librería para las funciones CAN
```

// **SECCIÓN 2**

```
6. //******************DECLARACION DE FUNCIONES******************
7. void low_isr(void);
8. void Enviar_Datos(int DatoNivel, int DatoTemperatura);
```

```
9.  unsigned char* division_num(int num, unsigned char *string);
10. unsigned char  valor_ascci(unsigned char x);
```

// **SECCIÓN 3**

```
11. //****************** INTERRUPCIÓN CAN*****************************
12. #pragma interrupt HighISR save=section(".tmpdata")
13. void HighISR(void)
14.     {
15. CANISR();
16.     }
17. #pragma code highVector=0x08
18. void HighVector (void)
19.     {
20. _asm goto HighISR _endasm
21.     }
22. #pragma code              // Regresa a la línea donde se generó la interrupción
```

// **SECCIÓN 4**

```
23. //****************INTERRUPCIÓN USART*******************************
24. #pragma code uart_int_service = 0x18
25. void uart_int_service(void)
26.     {
27. _asm       goto low_isr   _endasm
28.     }
29. #pragma code
30. #pragma  interruptlow low_isr save=section(".tmpdata")
31. void low_isr(void)
32.     {
33.     UARTIntISR();
34.     }
```

// **SECCIÓN 5**

```
35. void main()
36. {
37. struct CANMessage RX_Message, TX_Message;
38. char idNivel = 0x01, idTemperatura = 0x02, idControl = 0x03;
39. char datoControlNivel= 0, datoControlTemperatura = 0;
40. int datoTemperatura = 0, datoNivel = 0, cont = 0, datoNivelH = 0, datoNivelL = 0,
    datoTemperaturaH = 0,datoTemperaturaL = 0;
```

// **SECCIÓN 6**

// **Sección 6.1**

```
41. // Declaración de variables USART
42. unsigned char chData;
43. unsigned char j = 0;
44. unsigned char i = 0;
45. unsigned char l = 0;
46. unsigned char k = 0;
47. unsigned char rTemp[5], rNivel[5], recibidos[11];
```

// **Sección 6.2**

```
48. // Definición de bits
49. TRISBbits.TRISB5    =    0;    // Bit de transmisión CAN
50. TRISBbits.TRISB6    =    0;    // Bit de recepción de la señal de nivel
51. TRISBbits.TRISB7    =    0;    // Bit de recepción de la señal de temperatura
52. PORTBbits.RB5       =    0;
53. PORTBbits.RB6       =    0;
54. PORTBbits.RB7       =    0;
55. TRISCbits.TRISC0    =    0;    // Bit de verificar envío de datos USART
56. TRISCbits.TRISC1    =    0;    // Bit de secuencia de inicio
57. TRISCbits.TRISC2    =    0;    // Bit de verificar recepción de datos USART
58. PORTCbits.RC0       =    0;
59. PORTCbits.RC1       =    0;
```

```
60. PORTCbits.RC2       =    0;
61. TRISCbits.TRISC3    =    0;    // Bit de error CAN
62. PORTCbits.RC3       =    0;

// Sección 6.3

63. // Priorizamos y habilitamos las interrupciones, e inicializamos los módulos CAN y
    USART
64. CANInit();                    // Se inicializa el módulo CAN
65. INTCONbits.GIE      = 1;      // Se habilitan las interrupciones globales
66. INTCONbits.PEIE     = 1;      // Se habilitan las interrupciones periféricas
67. RCONbits.IPEN       = 1;      // Habilitamos niveles de prioridad de interrupción
68. UARTIntInit();                // Se inicializa el módulo USART

// Sección 6.4

69. // Secuencia de inicio
70. PORTCbits.RC1       = 1;
71. Delay10KTCYx( 100 );
72. PORTCbits.RC1       = 0;
73. Delay10KTCYx( 100 );
74. PORTCbits.RC1       = 1;
75. Delay10KTCYx( 100 );
76. PORTCbits.RC1       = 0

// Sección 6.5

77. while(1)                      // Este ciclo se repetirá indefinidamente
78. {
79. PORTBbits.RB5                 = !PORTBbits.RB5;

// Sección 6.6

80. // Recibe mensaje de los nodos esclavos
81. if(CANRXMessageIsPending())   // Chequea si hay un mensaje CAN en el
    "buffer" de recepción
```

```
82. {
83.     RX_Message = CANGet();          // Obtiene el mensaje
84. // Recepción de la trama de datos de la variable de nivel
85.     if(RX_Message.Remote == 0 && RX_Message.Address == idNivel)
86.     {
87.     // Extracción del dato de nivel
88.     datoNivelH = RX_Message.Data[0];
89.     datoNivelL = RX_Message.Data[1];
90.     datoNivel  = (datoNivelH)*255 + datoNivelL;
91.     PORTBbits.RB6               = !PORTBbits.RB6;
92.     Enviar_Datos(datoNivel,datoTemperatura);
93.     }

// Sección 6.7

94. // Recepción de la trama de datos de la variable de temperatura
95. if(RX_Message.Remote == 0 && RX_Message.Address == idTemperatura)
96.     {
97.     // Extracción del dato de temperatura
98.     datoTemperaturaH = RX_Message.Data[0];
99.     datoTemperaturaL = RX_Message.Data[1];
100.    datoTemperatura = (datoTemperaturaH)*255 + datoTemperaturaL;
101.    PORTBbits.RB7               = !PORTBbits.RB7;
102.    Enviar_Datos(datoNivel,datoTemperatura);
103.    }

// SECCIÓN 7

104.    // ********************RECEPCIÓN DE DATOS DEL PLC*****************************
105.    do{
106.    if(                 !(vUARTIntStatus.UARTIntRxError)                &&
        !(vUARTIntStatus.UARTIntRxOverFlow)                                 &&
        !(vUARTIntStatus.UARTIntRxBufferEmpty))
107.    {
108.    // Recibe los datos del "buffer" USART
```

```
109.  if(UARTIntGetChar(&chData))
110.          {
111.          recibidos[j]          =          chData;
112.          j++;
113.          }
114.  }
115.  if(chData == 'F') PORTCbits.RC2          =          !PORTCbits.RC2;
116.  }while(chData!='F');
117.  recibidos[j]    =          0;
118.  j = 0;
119.  if(recibidos[0]=='T')
120.  {
121.  l = 0;
122.  do{
123.          rTemp[l]          =          recibidos[l+1];
124.          l++;
125.  }while(recibidos[l+1]!='N');
126.  rTemp[l]=0;
127.  // Se obtiene el dato de control de temperatura
128.  datoControlTemperatura    =          atoi(&rTemp);
129.  k=0;
130.  l++;
131.  do{
132.          rNivel[k]          =          recibidos[l+1];
133.          k++; l++;}
134.  while(recibidos[l+1]!='F');
135.  rNivel[k]          =          0;
136.  // Se obtiene el dato de control de nivel
137.  datoControlNivel          =          atoi(&rNivel);
138.  }
```

// **SECCIÓN 8**

```
139.  // ********** CONTRUCCIÓN DE LA TRAMA CAN**********
140.  cont++;
```

```
141.  if(cont==15000){
142.  TX_Message.Address          = idControl;
143.  TX_Message.Ext              = 0;
144.  TX_Message.NoOfBytes        = 2;
145.  TX_Message.Data[0]          = datoControlNivel;
146.  TX_Message.Data[1]          = datoControlTemperatura;
147.  TX_Message.Remote           = 0;
148.  TX_Message.Priority         = 1;
149.  CANPut(TX_Message);
150.  cont=0;
151.  }
152.  }
153.  }
```

// **SECCIÓN 9**

```
154.  //Envío de datos de las señales de temperatura y de nivel al sistema SCADA
155.  void Enviar_Datos(int DatoNivel, int DatoTemperatura)
156.  {
157.  unsigned char i = 0;
158.  unsigned char arreglo[11];
159.  // Definición de arreglo
160.  arreglo[0]   =   'T';
161.  arreglo[5]   =   'N';
162.  arreglo[10]  =   13;
163.  PORTCbits.RC0               = !PORTCbits.RC0;
164.  division_num(DatoTemperatura,&arreglo[1]);
165.  division_num(DatoNivel,&arreglo[6]);
166.    i=0;
167.   while(i<=10)
168.        {
169.            // Clareamos el watch dog timer
170.            ClrWdt();
171.            // Envío del dato
172.            if(!vUARTIntStatus.UARTIntTxBufferFull)
```

```
173.                        {
174.                        // Enviar datos
175.                        if(UARTIntPutChar(arreglo[i]))
176.                              {
177.                              i++;
178.                              Delay1KTCYx(1);
179.                              }
180.                        }
181.          }
```

// **SECCIÓN 10**

```
182.    // División del valor obtenido de los nodos esclavos en sus unidades
183.    unsigned char* division_num(int num, unsigned char *string)
184.    {
185.    unsigned char m,miles='0',centena='0',decena='0',unidad='0',finDeLinea=13,
    cadena[4];
186.          if(num>=1000){
187.                              miles = valor_ascci(num/1000);
188.                              num         =     num%1000;
189.                              }
190.          if(num>=100 && num <1000){
191.                              centena =     valor_ascci(num/100);
192.                              num         =     num%100;
193.                              }
194.          if(num<100 && num>=10){
195.                              decena      =     valor_ascci(num/10);
196.                              num         =     num%10;}
197.          if(num<10){
198.                              unidad      =     valor_ascci(num);
199.                        }
200.          cadena[0]=miles;
201.          cadena[1]=centena;
202.          cadena[2]=decena;
203.          cadena[3]=unidad;
```

```
204.            for(m=0;m<=3;m++)
205.                    *(string + m) = cadena[m];
206.    return &cadena;
207.    }
```

// **SECCIÓN 11**

```
208.    // Se convierte los números a sus correspondientes valores ASCII
209.    unsigned char  valor_ascci(unsigned char x)
210.    {
211.      unsigned char ascci;
212.            switch (x)
213.            {
214.            case 0: ascci='0'; break; //correponde al codigo ASCCI del numero 0
215.            case 1: ascci='1'; break; //correponde al codigo ASCCI del numero 1
216.            case 2: ascci='2'; break; //correponde al codigo ASCCI del numero 2
217.            case 3: ascci='3'; break; //correponde al codigo ASCCI del numero 3
218.            case 4: ascci='4'; break; //correponde al codigo ASCCI del numero 4
219.            case 5: ascci='5'; break; //correponde al codigo ASCCI del numero 5
220.            case 6: ascci='6'; break; //correponde al codigo ASCCI del numero 6
221.            case 7: ascci='7'; break; //correponde al codigo ASCCI del numero 7
222.            case 8: ascci='8'; break; //correponde al codigo ASCCI del numero 8
223.            case 9: ascci='9'; break; //correponde al codigo ASCCI del numero 9
224.            }
225.     return(ascci);
226.    }
```

// **SECCIÓN 12**

```
227.    void CANErrorHandler(void)
228.    {
229.    // Secuencia para indicar si se ha presentado algún error definido por el protocolo
        CAN
230.    PORTCbits.RC3               = 1;
231.    Delay10KTCYx( 100 );
```

```
232.    PORTCbits.RC3                = 0;
233.    }
```

Como se puede observar el código esta dividido en secciones. Se explica de forma general cada una de éstas y solo se profundiza en las funciones que se relacionan con el protocolo CAN [1] y la comunicación serial USART [2].

Sección 1

Se incluye las librerías que van hacer utilizadas para acceder a las funciones CAN, USART, "*delays*", ADC y estándares del ANSI C.

Sección 2

Se declaran las funciones de usuario que se van a implementar para el desarrollo del programa. Cada función se explica mas adelante en su respectiva sección.

Sección 3

Código que se ejecuta cuando hay una interrupción CAN. Existen dos situaciones por las cuales se puede generar ésta interrupción:

1. Cuando un mensaje es recibido en el "*MAB*" ("*Buffer" de Montaje del Mensaje*). El mensaje es almacenado en uno de los dos "*buffers*" de recepción si pasa el criterio de los filtros de aceptación.
2. Cuando uno de los tres "*buffers*" de transmisión esta listo para transmitir.

El PIC18F258 tiene dos niveles de prioridad para las interrupciones: un nivel alto y un nivel bajo. La interrupción CAN esta configurada en el nivel alto.

Sección 4

Código que se ejecuta cuando se genera una interrupción USART para la comunicación serial entre el PIC y el PLC. Una interrupción USART puede generarse de dos formas:

1. Cuando se recibe un mensaje en el "*buffer*" de recepción de mensajes USART.
2. Cuando el "*buffer*" de transmisión USART esta listo para transmitir.

La interrupción esta configurada en el nivel bajo de interrupción. Para más información ver [3] (pagina 79).

Sección 5

Se declaran las variables:

- **RX_Message:** Estructura utilizada para almacenar el mensaje CAN que se extrae de uno de los dos "*buffers*" de recepción CAN.
- **TX_Message:** Estructura en donde se almacena, campo por campo, toda la trama CAN
- **idNivel, idTemperatura, idControl:** Son los identificadores estándares que corresponden a las variables de nivel de líquido, temperatura y señal de control. Incorporado en el mensaje con el identificador idControl viaja tanto la señal de control de la planta de nivel como la señal de control de la planta de temperatura.
- **datoControlTemperatura y datoControlNivel:** Son los datos que el nodo maestro recibe del PLC o sistema SCADA que corresponden a los valores de la señal de control de temperatura y control de nivel respectivamente. Datos que posteriormente son enviados a los nodos esclavos a través del protocolo CAN.
- **datoNivelH, datoNivelL, datoTemperaturaH, datoTemperaturaL:** Variables usadas para almacenar los valores de 10 bits enviados por los sensores de nivel y temperatura de las plantas. Posteriormente se usan para formar el dato de 10 bits (datoTemperatura y datoNivel) de cada señal que será enviado hacia el PLC.
- **datoTemperatura y datoNivel:** Son los datos o valores de temperatura y nivel que serán enviados hacia el PLC o sistema SCADA.

Sección 6

Sección 6.1

Declaración de las variables para la comunicación serial.

Sección 6.2

Configuración de los puertos ya sea como entrada o como salida. Algunos son banderas de información visual, por ejemplo para saber cuando el PIC esta enviando o recibiendo

mensajes CAN se utilizan LEDs conectados a los pines 5 y 6 del puerto B respectivamente. Se configura para algunos pines del puerto su estado inicial, cuando es necesario. Los pines 2 y 3 del puerto B corresponden a los pines de transmisión y recepción CAN respectivamente.

Sección 6.3

Se habilitan las interrupciones globales y periféricas. Posteriormente se configura al PIC para habilitar los niveles de prioridad de las interrupciones.

Las funciones *CANInit()* y *UARTInit()* cargan los códigos, CANDef.h yUARTIntC.def, en donde se encuentran la configuración inicial realizada en la Aplicación "*maestro*" para los módulos CAN y USART respectivamente.

Sección 6.4

Se configura una secuencia de inicio que consiste en encender un LED 2 veces intermitentemente. El objetivo es establecer una secuencia para determinar visualmente cuando el PIC se esta reseteando. Esto para efecto de depuración.

Sección 6.5

Empieza el ciclo que se repetirá indefinidamente.

Sección 6.6

Si hay un mensaje en alguno de los "*buffers*" de recepción se almacena éste en la estructura RX_Message utilizando la función *CANGet().* Si la trama es de datos y el identificador del mensaje coincide con el identificador de nivel se extrae el valor enviado en el campo de datos del mensaje - que es el dato de la medición del nivel enviado por el nodo esclavo 1 - y se lo almacena en *datoNivel*.

Sección 6.7

Si la trama es de datos y el identificador del mensaje coincide con el identificador de temperatura se extrae el valor enviado en el campo de datos del mensaje - que es el dato

de la medición de la temperatura enviado por el nodo esclavo 2 – y se lo almacena en *datoTemperatura*.

Sección 7

A través de la norma RS232 el PIC recibe los datos de control de nivel y de control de temperatura desde el sistema SCADA.

Sección 8

Se construye la trama de datos para enviar el valor de control de nivel y de temperatura con destino a los nodos esclavos. Se siguen los siguientes pasos:

1. **TX_Message.Address:** Corresponde al identificador del mensaje.
2. **TX_Message.Ext**: Indica si el mensaje tiene un identificador estándar (0) o extendido (1).
3. **TX_Message.NoOfBytes:** Indica la cantidad de bytes que contiene el campo de datos. Los valores posibles son de 1 a 8 bytes. Para la aplicación se ha escogido un valor de 2.
4. **TX_Message.Data:** Es un arreglo de 8 bytes en donde se almacenan los datos que se quieren enviar. En este campo se almacenaran los valores de las señales de control de nivel y temperatura. En el byte 1 se almacena la señal de control de la planta de nivel y en el byte 2 la señal de control de la planta de temperatura.
5. **TX_Message.Remote:** Si es cero indica una trama remota y si es uno indica una trama de datos.
6. **TX_Message.Priority:** Prioridad del **"***buffer*". Esta prioridad no hace parte del protocolo CAN, simplemente es una forma de informar al PIC cual de los "*buffers*" transmitirá primero. Los valores posibles van desde el 3 con la prioridad más alta hasta el 0 con la prioridad mas baja.
7. **CANPut:** Coloca el mensaje en el FIFO.

Sección 9

En esta sección se ejecuta la función de *envío de datos* desde el PIC 18F258 hacia el PLC. Para esto, los dos datos a enviar, *datoTemperatura y datoNivel*, se dividen en sus dígitos y éstos a su vez se convierten al código ASCII haciendo uso de la función

división_num. Luego, los valores ASCII son ordenados en un arreglo para ser enviados por el canal serial. Para mayor claridad se da un ejemplo: si se tiene *datoTemperatura = 1024* y *datoNivel = 0128* al llamar la función *división_num* obtendremos que *arreglo = [‘T’, ‘1’, ‘0’, ‘2’, ‘4’, ‘N’, ‘0’, ‘1’, ‘2’, ‘8’, CR]*. La T es para indicar que comienza el dato de temperatura, la N es para fijar el comienzo del dato de nivel y el CR es el *Retorno de Carro*, para señalar el fin de la trama. Se recuerda que las comillas simples (‘ ’) son la forma de declarar constantes de tipo char, otra forma posible de enviar los datos es cambiar lo limitado por las comillas simples por su código ASCII decimal. Por ejemplo cambiar ‘1’ por 49.

Luego de obtener el “*array*” *arreglo* se procede ha enviar cada dato, uno a la vez. Para ello se pregunta primero si por lo menos un “*buffer*” de transmisión USART esta desocupado, si es así, se procede a almacenar el dato en el “*buffer*” y a enviarlo posteriormente. Cuando se llega al CR la función termina. Por último, el PIC envía los datos haciendo uso de la interrupción USART explicada con anterioridad.

Sección 10

Esta función divide los datos que se van a enviar al PLC. El parámetro de salida de esta función es el arreglo con los valores en código ASCII.

Sección 11

Por último se tiene la función de conversión de los dígitos de los datos a enviar a sus respectivos valores ASCII.

Sección 12

Si ocurre un error relativo al protocolo CAN, el bit 4 (RC3) del puerto C se encenderá por un instante de tiempo.

1.2 NODO ESCLAVO 1

El código implementado en el nodo esclavo 1 es semejante al código del nodo maestro, con la diferencia de que no ejecuta la comunicación serial USART, por lo tanto solo se muestra el código pero no se incurre en explicaciones.

```
1.  #include   <p18f258.h>          // Incluir la librería del PIC
2.  #include   "can.h"              // Incluir la librería para las funciones CAN
3.  #include   <delays.h>           // Incluir la librería para las funciones “delays”
4.  #include   <adc.h>              // Incluir la librería para las funciones ADC
5.  int leer_ADC(int *ResADC);
6.  #pragma interrupt HighISR save=section(".tmpdata")
7.  void HighISR(void)
8.  {
9.  CANISR();
10. }
11. #pragma code highVector=0x08
12. void HighVector (void)
13. {
14. _asm goto HighISR _endasm
15. }
16. #pragma code  // Regresa a la línea donde se generó la interrupción
17. void main()
18. {
19. struct CANMessage RX_Message, TX_Message;
20. char idNivel = 0x01, idControl = 0x03;
21. int  datoADC=0, cont = 0, resADC[2];
22. // Definición de bits
23. TRISC                = 0x00;
24. TRISBbits.TRISB5     = 0;    // Bit de Inicio y de transmisión de la señal de nivel
25. TRISBbits.TRISB6     = 0;    // Bit de recepción de la señal de control
26. TRISBbits.TRISB7     = 0;    // Bit de error CAN
27. PORTBbits.RB5        = 0;
28. PORTBbits.RB6        = 0;
29. PORTBbits.RB7        = 0;
30. // Habilitamos las interrupciones e inicializamos el módulo CAN
31. CANInit();                   // Se inicializa el módulo CAN
32. INTCONbits.GIE  = 1;         // Se habilitan las interrupciones globales
33. INTCONbits.PEIE = 1;         // Se habilitan las interrupciones periféricas
34. // Secuencia de inicio del nodo
```

```
35. PORTBbits.RB5           = 1;
36. Delay10KTCYx( 100 );
37. PORTBbits.RB5           = 0;
38. Delay10KTCYx( 100 );
39. PORTBbits.RB5           = 1;
40. Delay10KTCYx( 100 );
41. PORTBbits.RB5           = 0;
42. while(1)                        // Este ciclo se repetirá indefinidamente
43. {
44. // Recibe mensaje del nodo maestro
45. if(CANRXMessageIsPending()) // Chequea si hay un mensaje CAN en el "buffer" de
    recepción
46. {
47.     RX_Message = CANGet();    // Obtiene el mensaje
48. // Recepción del mensaje de las señales de control
49.     if(RX_Message.Remote == 0 && RX_Message.Address == idControl)
50.     {
51.      PORTC = RX_Message.Data[0];
52.      PORTBbits.RB6              = !PORTBbits.RB6 ;
53.      }
54. }
55.  // ************** CONTRUCCIÓN DE LA TRAMA CAN******************
56. cont++;
57. if(cont==15000)
58. {
59. leer_ADC(&resADC);
60. TX_Message.Address          = idNivel;
61. TX_Message.Ext              = 0;
62. TX_Message.NoOfBytes        = 2;
63. TX_Message.Data[0]          = resADC[0];
64. TX_Message.Data[1]          = resADC[1];
65. TX_Message.Remote           = 0;
66. TX_Message.Priority         = 1;
67. CANPut(TX_Message);
```

```
68. PORTBbits.RB5                = !PORTBbits.RB5;
69. cont = 0;
70. }
71. }
72. }
73. int leer_ADC(int *ResADC)
74. {
75. // Se lee el conversor A/D para adquirir el dato del sensor de nivel
76. // Se configura el conversor A/D
77. OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF, ADC_CH0 &
    ADC_INT_OFF );
78. Delay10TCYx( 7 );        // Retardo
79. ConvertADC();            // Inicio de la conversión
80. while( BusyADC() );      // Se espera mientras se finaliza la conversión
81. ResADC[0] = ADRESH;      // Leemos el resultado de la conversión
82. ResADC[1] = ADRESL;
83. CloseADC();              // Deshabilitamos el conversor A/D
84. return 0;
85. }
86. void CANErrorHandler(void)
87. {
88. // Secuencia para indicar si se ha presentado algún error definido por el protocolo CAN
89.     PORTBbits.RB7        = 1;
90.     Delay10KTCYx( 100 );
91.     PORTBbits.RB7        = 0;
92. }
```

1.3 NODO ESCLAVO 2

El código implementado en el nodo esclavo 2 es similar al que se implementó en el nodo esclavo 1. A continuación se presenta el código.

```
1. #include <p18f258.h>      // Incluir la librería del PIC
2. #include "can.h"          // Incluir la librería para las funciones CAN
3. #include <delays.h>       // Incluir la librería para las funciones “delays”
```

```
4.  #include   <adc.h>                // Incluir la librería para las funciones ADC
5.  int leer_ADC(int *ResADC);
6.  #pragma interrupt HighISR save=section(".tmpdata")
7.  void HighISR(void)
8.  {
9.  CANISR();
10. }
11. #pragma code highVector=0x08
12. void HighVector (void)
13. {
14. _asm goto HighISR _endasm
15. }
16. #pragma code                      // Regresa a la línea donde se generó la interrupción
17. void main()
18. {
19. struct CANMessage RX_Message, TX_Message;
20. char idTemperatura = 0x02, idControl = 0x03;
21. int datoADC = 0, cont = 0, resADC[2];
22. // Definición de bits
23. TRISC = 0x00;
24. TRISBbits.TRISB5      = 0;    // Bit de Inicio y de transmisión de la señal de
    temperatura
25. TRISBbits.TRISB6      = 0;    // Bit de recepción de la señal de control
26. PORTBbits.RB5         = 0;
27. PORTBbits.RB6         = 0;
28. // Habilitamos las interrupciones e inicializamos el módulo CAN
29. CANInit();                       // Se inicializa el módulo CAN
30. INTCONbits.GIE  = 1;             // Se habilitan las interrupciones globales
31. INTCONbits.PEIE = 1;             // Se habilitan las interrupciones periféricas
32. // Secuencia de inicio del nodo
33. PORTBbits.RB5         = 1;
34. Delay10KTCYx( 100 );
35. PORTBbits.RB5         = 0;
36. Delay10KTCYx( 100 );
```

```
37. PORTBbits.RB5          = 1;
38. Delay10KTCYx( 100 );
39. PORTBbits.RB5          = 0;
40. while(1)               // Este ciclo se repetirá indefinidamente
41. {
42. // Recibe mensaje del nodo maestro
43. if(CANRXMessageIsPending())  // Chequea si hay un mensaje CAN en el "buffer" de
    recepción
44. {
45. RX_Message = CANGet();       // Obtiene el mensaje
46. // Recepción del mensaje de las señales de control
47.     if(RX_Message.Remote == 0 && RX_Message.Address == idControl)
48.     {
49.     PORTC = RX_Message.Data[1];
50.     PORTBbits.RB6          = !PORTBbits.RB6 ;
51.     }
52. }
53. // ************** CONTRUCCIÓN DE LA TRAMA CAN******************
54. cont++;
55. if(cont==15000)
56. {
57. leer_ADC(&resADC);
58. TX_Message.Address      = idTemperatura;
59. TX_Message.Ext          = 0;
60. TX_Message.NoOfBytes    = 2;
61. TX_Message.Data[0]      = resADC[0];
62. TX_Message.Data[1]      = resADC[1];
63. TX_Message.Remote       = 0;
64. TX_Message.Priority     = 1;
65. CANPut(TX_Message);
66. PORTBbits.RB5           = !PORTBbits.RB5;
67. cont=0;
68. }
69. }
```

```
}
int leer_ADC(int *ResADC)
{
// Se lee el conversor A/D para adquirir el dato del sensor de temperatura
// Se configura el conversor A/D
OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF, ADC_CH0 &
    ADC_INT_OFF );
Delay10TCYx( 7 );        // Retardo
ConvertADC();            // Inicio de la conversión
while( BusyADC() );      // Se espera mientras se finaliza la conversión
ResADC[0] = ADRESH;  // Leemos el resultado de la conversión
ResADC[1] = ADRESL;
CloseADC();              // Deshabilitamos el conversor A/D
return 0;
}
void CANErrorHandler(void)
{
// Secuencia para indicar si se ha presentado algún error definido por el protocolo CAN
    PORTBbits.RB7           = 1;
    Delay10KTCYx( 100 );
    PORTBbits.RB7           = 0;
}
```

Para más información acerca de las librerías y/o funciones usadas en los programas desarrollados anteriormente consultar [4].

REFERENCIAS BIBLIOGRAFICAS

[1] MICROCHIP TECHNOLOGY, INC. “Interrupt Based CAN Library Module”. http://ww1.microchip.com/downloads/en/DeviceDoc/canintc.readme.pdf

[2] MICROCHIP TECHNOLOGY, INC. “Interrupt Based UART Library Module (For C Language)”. http://ww1.microchip.com/downloads/en/DeviceDoc/uartintc.readme.pdf

[3] MICROCHIP TECHNOLOGY, INC. ”PIC18FXX8 Data Sheet ”. http://ww1.microchip.com/downloads/en/DeviceDoc/41159d.pdf

[4] MICROCHIP TECHNOLOGY, INC. “MPLAB C18 C Compiler Libraries”. http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_C18_Libraries_51297f.pdf