7 SIMULACIÓN DE CONTROL MEDIANTE PIC

Para establecer una comparación entre el trabajo desarrollado con en PLC y el posible uso de microcontroladores PIC se hace uso de software de simulación de prototipos electrónicos como lo es PROTEUS, el cual da la posibilidad de utilizar una gran gama de componentes electrónicos entre ellos una amplia variedad de PIC. El desarrollo de los códigos de los microcontroladores se estableció usando el software PIC C Compile, el que permite el uso de lenguaje de programación basado en C, el cual tiene una serie de instrucciones genéricas enfocadas a la configuración de los PIC sin tener que estar revisando el set de instrucciones de cada uno en particular.

Una vez desarrollada dicha simulación, su manejo se hace mediante una interfaz en la PC la cual dará el control del movimiento de los motores así como la manipulación de algún otro tipo de datos que se requieran. Para ello es necesario contar con un tipo de comunicación entre la PC y el prototipo simulado, con lo cual se tiene una simulación aún más completa de cómo quedaría un control de los motores de un brazo robótico mediante la tecnología mencionada.

7.1 DISEÑO DEL CONTROL MEDIANTE PIC

7.1.1 PIC16F876A

El PIC16F876A es el utilizado en las simulaciones por ser uno de los más comunes en el mercado y dada la posibilidad de obtener muestras gratuitas por parte de la empresa que los fabrica. Forma parte de la subfamilia PIC16F87X de microcontroladores PIC (*Peripheral Interface Controller*) de gama media de 8 bits, fabricados por Microchip Technology Inc. Las características principales de esta familia son las siguientes:

- CPU de arquitectura RISC (*Reduced Instruction Set Computer*).
- Set de 35 instrucciones.
- Frecuencia de reloj de hasta 20MHz (ciclo de instrucción de 200ns).
- Todas las instrucciones se ejecutan en un único ciclo de instrucción, excepto las de salto.
- Hasta 8K x 14 palabras de Memoria de Programa FLASH.
- Hasta 368 x 8 bytes de Memoria de Datos tipo RAM.
- Hasta 256 x 8 bytes de Memoria de Datos tipo EEPROM.
- Hasta 15 fuentes de Interrupción posibles.
- Modo de bajo consumo (Sleep).
- Tipo de oscilador seleccionable (RC, HS, XT, LP y externo).
- Rango de voltaje de operación desde 2,0V a 5,5V.
- Convertidor Analógico/Digital de 10 bits multicanal.
- 3 Temporizadores.
- 2 módulos de captura/comparación/PWM.
- Comunicaciones por interfaz USART (Universal Synchronous Asynchronous Receiver Transmitter).
- Puerto Paralelo Esclavo de 8 bits (PSP).
- Puerto Serie Síncrono (SSP) con SPI e I^2C.

De las características mencionadas cabe destacar la memoria EEPROM, los temporizadores, los módulo CCP en modo PWM y el puerto serie e I2C, las cuales son usadas para el desarrollo del control de los motores.

TIMERS

Los temporizadores o TIMER son módulos integrados en el PIC, los cuales permiten realizar cuentas ya sean internas o de eventos externos. Cuando se usan internamente se usa el término timer mientras que cuando se usan externamente son contadores, por lo que mediante estos bloques se hará el conteo de los pulsos generados por el encoder de cada motor.

El TIMER0 es un contador de 8 bits, incrementado físicamente contando los eventos externos conectados al pin RA4/TOCK1 o mediante programación contando los pulsos internos del reloj.

Cuenta con un preescalador, el cual es un divisor de frecuencia configurable a dividir entre 2, 4, 8, 16, 32, 64, 128 ó 256. Cabe mencionar que la frecuencia de conteo será una cuarta parte de la frecuencia del reloj, y posteriormente dicha frecuencia de conteo es posible dividirla usando el preescalador.

El TIMER1 es otro contador el cual se diferencia por manejar 16 bits, su preescalador sólo puede dividirse entre 2, 4 y 8 y además cuenta con 3 modos de funcionamiento:

- Temporizador. Incrementándose cada ciclo de instrucciones.
- Contador Síncrono. El cual sincronizará la señal externa con la fase del reloj interno.
- Contador Asíncrono. La señal externa no está asociada al reloj interno, por lo tanto se puede seguir incrementando aún en modo SLEEP del PIC.

Cabe destacar que el TIMER1 en modo contador únicamente tomará los flancos de subida como flanco activo y una vez configurado se debe producir un flanco de bajada para comenzar el conteo.

El TIMER2 es de 8 bits, posee un preescalador con divisiones entre 4 y 16 y un postescalador divisible hasta entre 16. Asimismo el TIMER2 puede ser usado como base de tiempo para la modulación en ancho de pulso (PWM).

Para el uso de los timer en lenguaje C existen las siguientes funciones de configuración:

```
setup_timer_0(modo);

setup_timer_1(modo);

Setup_timer_2(modo, periodo, postescalador);
```

El parámetro modo es cambiado por la configuración de conteo interno o externo, y el uso de los escaladores, para el caso del TIMER2 el periodo es un valor entero de 8 bits, y el postescalador es un valor entre 1 y 14

Asimismo para la lectura o escritura en los timer el lenguaje C tiene las siguientes funciones:

```
set_timerX(valor);

valor=get_timerX();
```

MÓDULO CCP

El módulo CCP permite realizar tres funciones básicas basadas en el manejo de los timer:

- Comparador: compara el valor del timer con el valor de un registro para la realización de cierta acción.
- Captura: obtiene el valor del timer en un momento específico, fijado en la programación del PIC.
- PWM: genera una señal de tren de pulsos.

Cada módulo CCP posee un registro de 16 bits el cual puede ser usado para capturar el valor del timer, para comparar el valor del registro con el TIMER1 o como registro de 10 bits para el ciclo de trabajo de una señal PWM.

En este caso el de interés es el modo PWM, el cual permite obtener en los pines CCPx una señal periódica en la cual se puede variar su ciclo de trabajo.

El compilador C tiene ciertas funciones para el manejo del módulo CCP, para la configuración del mismo se usa:

setup_ccpx(modo)

en este caso el *modo* será CCP_PWM, para definir el ciclo de trabajo se usa:

set_pwm_duty(valor);

donde *valor* es un dato de 8 ó 16 bits.

MODOS DE COMUNICACIÓN.

Los PIC utilizan, dos modos de transmisión en serie:

- El puerto serie síncrono (SSP)
- La interfaz de comunicación serie (SCI) o receptor transmisor serie síncrono-asíncrono universal.

PUERTO SERIE SÍNCRONO (SSP)

El SSP se suele utilizar en la comunicación con otros microcontroladores o con periféricos. Las dos interfaces de trabajo son:

- Interfaz serie de periféricos (SPI). Comunicación entre microcontroladores de la misma, o diferente, familia en modo maestro esclavo; Full dúplex.
- Interfaz Inter-circuitos (I^2C). Interfaz con capacidad para comunicar microcontroladores y periféricos; Half dúplex.

MÓDULO USART

La función del módulo USART es recibir y transmitir datos en serie, ya se síncrona o asíncrona. Mientas que la señal síncrona utiliza una seña de reloj y una línea de datos, las transmisiones asíncronas emplean reloj en el emisor y en el receptor, los cuales deben ser de igual frecuencia y deben estar en fase, para ello la frecuencia del reloj se establece antes de la transmisión de datos configurando la velocidad. Cada trama de datos tiene un tamaño fijo y poseen un bit de arranque y un bit de parada, los cuales permiten realizar la sincronización, su ventaja se presenta al poder transmitir y recibir simultáneamente ya que usa dos líneas (full-dúplex) una transmisora o TX y otra receptora o RX.

Para el uso de esta comunicación los PIC tienen dos pines específicos para este fin, que son RC6/TX y RC7/RX.

Para el uso del módulo USART en C se debe establecer la configuración con la directiva:

#USE RS232 (opciones)

donde se configuran los parámetros como velocidad de transmisión, pines utilizados, etc.

Para la transmisión de datos existen varias funciones dentro del lenguaje C, las cuales son:

putc(cdata)

putchar(cdata)

donde *cdata* es un carácter de 8 bits, el cual será enviado usando el pin de TX.

puts(string)

donde string es una cadena de caracteres constante o matriz de caracteres terminada con un 0.

printf(fname, cstring, values)

donde *cstring* es una cadena de caracteres o matriz de caracteres, *fname* son las funciones a utilizar para escribir la cadena indicada y los valores son valores a incluir en la cadena separados por comas.

Para la recepción de datos las funciones existentes son:

value = getc()

value = getch()

valuer = getchar();

donde *value* es un carácter de 8 bits. Estas funciones esperan recibir un carácter por la línea y después devolver su valor.

Existe además una función que indica si existe un valor entrante, la función *kbhit*(), será 0 si no hay datos entrantes, y será 1 en el momento en que haya un carácter listo para ser leído.

INTERFAZ I²C

El modo de trabajo I2C se basa en la comunicación a través de 2 hilos. Cada dispositivo conectado al bus tiene una dirección específica. Puede configurarse como comunicación de un maestro y varios esclavos o una configuración multimaestro. En ambas instancias el maestro es el que tiene la iniciativa en la transferencia, decide con quién se comunicará, si será envío o recepción de datos y el momento en que finalice. Cuando el maestro inicia la comunicación envía la dirección del dispositivo con el que se comunicará y los esclavos comparan este dato con su dirección y el último bit de estos datos indica si será envío o lectura de datos.

Los hilos usados en el bus I2C son dos líneas de colector abierto: la señal del reloj SCL o pin RC3 y la línea de datos SDA o pin RC4. Se deben utilizar unas resistencias externas (pull-up) para asegurar un nivel alto cuando no hay dispositivos conectados al bus.

La transmisión de las señales se hace mediante un bit de inicio, posteriormente se envían los datos y al final se envía un bit de parada, como se muestra en la Figura 7.1.

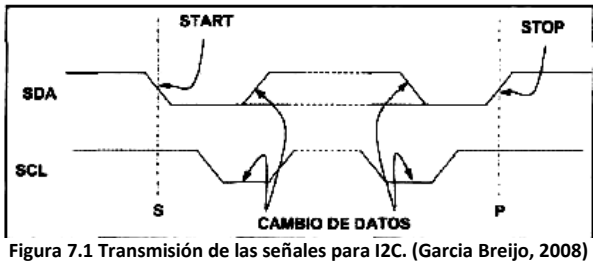

Figura 7.1 Transmisión de las señales para I2C. (Garcia Breijo, 2008)

Una vez que el maestro envía la dirección o datos, el esclavo genera un bit de reconocimiento, para informar que recibió datos, en caso de que el maestro no reciba dicho dato, la comunicación se interrumpe, generando una señal de Stop. En caso de que el maestro sea el que recibe datos, será éste el que envíe el bit de reconocimiento, como se muestra en la Figura 7.2.

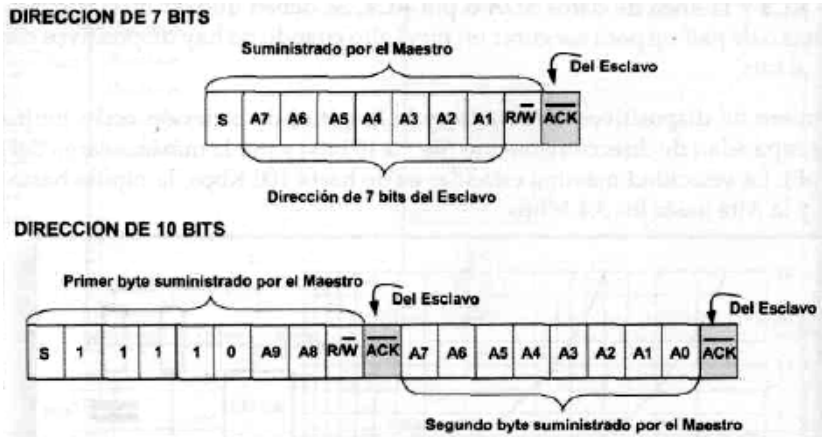

Figura 7.2. Formato de los datos enviados. (Garcia Breijo, 2008)

Para el uso de la comunicación I2C se debe configurar con la directiva:

#use I2C(opciones)

Las funciones asociadas son:

I2C_WRITE(dato). El dato es un entero de 8 bits enviado por el bus.

I2C_START(). La cual inicializa la transmisión de datos.

I2C_READ(). El dato recibido será un entero de 8 bits.

I2C_STOP(). Finaliza la transmisión.

I2C_POLL(). Se pone en 1 si se recibió un dato en el buffer y un 0 cuando no se ha recibido.

I2C_SLAVEADDR(adr). Especifica la dirección del dispositivo en modo esclavo.

7.1.2 PROGRAMACIÓN DE LOS PICS

Para el control de cada motor se tendrán algunas funciones simples como la asignación de dirección de giro usando pines de salida de alguno de sus puertos conectados a los pines de entrada del puente H, otra tarea aún más compleja es implementar una señal PWM usando el Timer2 conectado al pin Enable del puente para hacer el control de la velocidad. Y finalmente la función que podría ser la principal es la de llevar el conteo de los pulsos de los encoders para ello se usan dos timer en modo contador para que uno sea preescalado y cuente únicamente un pulso cada 128, mientras que el otro contador contará cada dos pulsos, lo cual dará un valor real del número de pulsos contados. Dadas las funciones a realizar se debe usar un PIC por cada motor a controlar, tomando esta referencia se presentan dos necesidades: establecer comunicación entre los dispositivos y con la PC así como repartir tareas ente ellos.

Para resolver ambos problemas de manera sencilla es posible utilizar un PIC maestro, el cual controle las comunicaciones y PIC esclavos que llevan la carga de interactuar con los motores y sus respectivos encoders. De este modo en la Figura 7.3 se muestra un esquema del modelo diseñado.

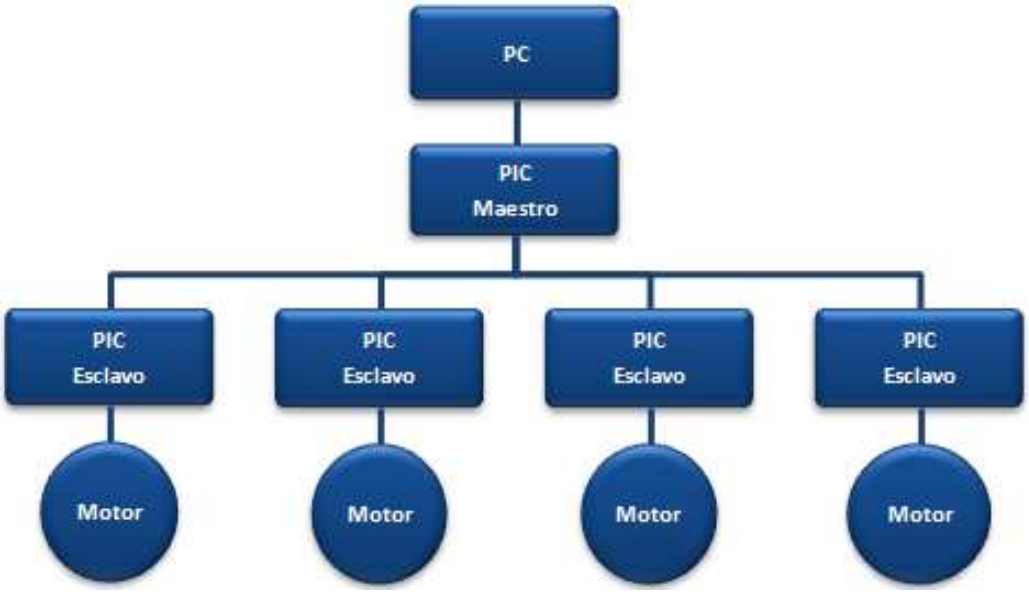

Figura 7.3. Diagrama de configuración para el control de motores.

Una vez establecido el esquema, se observa que la comunicación entre el PIC maestro y los esclavos, dado que son varios es posible hacerla usando comunicación I^2C, estableciendo una dirección a cada PIC, de esta forma si se quieren agregar más PIC bastará con asignarles una dirección y programar su código en el maestro. Ahora bien para la comunicación del maestro con la PC se usa una comunicación serial.

Para la programación de las labores a desarrollar por cada PIC esclavo se seguirá el esquema de la Figura 7.4.

Figura 7.4. Diagrama de pasos de PIC esclavo.

Inicialmente en la función Main se llama la subfunción leer la cual sirve para recibir datos como se muestra en la Figura 7.5.

```
void leer(void){            //función para leer datos por I2C como esclavo
   do{
      valor2=0;     //variable para mantener ciclo while hasta tener datos
      if(i2c_poll()==TRUE){//espera hasta recibir datos
      dato=i2c_read();      //leer datos recibidos
      if(dato==0xA2){       //comparar datos con direccion del pic
      dato=i2c_read();      //leer dato y guardarloen variable
      valor2=1;             //variable para terminar ciclo
      }
   }
   }while(valor2!=1);
```

Figura 7.5. Función Leer en lenguaje C para PIC Compiler.

Se observa que una vez llamada la función leer se mantendrá haciendo un ciclo hasta que reciba los datos.

Posteriormente el dato recibido será almacenado en la memoria EEPROM, esto se hará para cada uno de los 3 datos necesarios para el movimiento del motor, los cuales son velocidad, sentido de giro y número de cuentas a moverse.

```
do
{
  do{
     delay_ms(050);
     leer();                    //leer dato de velocidad
     write_eeprom(0x00,dato); //escribir en EEPROM
     leer();                    //leer dato de sentido
     write_eeprom(0x01,dato); //escribir en EEPROM
     leer();                    //leer dato de cuenta
     write_eeprom(0x02,dato); //escibir dato en EEPROM
   }while(valor2!=1);          //hacer ciclo mientras valor2 sea dif de
```

Figura 7.6. Código para escribir en memoria E^2PROM

Una vez que se tienen los datos serán cargados en variables y se analizan para establecer el sentido de giro como se muestra en la Figura 7.7.

```
valor2=0;                    //reseteo de variable para mantener ciclo
Vel=read_eeprom(0x00);       //leer velocidad de la EEPROM
Sen=read_eeprom(0x01);       //leer sentido de la EEPROM
Cuenta=read_eeprom(0x02);    //leer cuenta de la EEPROM

if(Sen=='R')           //comparacion del sentido de giro
{                      //si se recibe R
MoverDer();            //habilitar funcion de giro a la derecha
}
else if(Sen=='L')      //si se recibe L
{
MoverIzq();            //habilitar giro a la izquierda
}
else if(Sen=='S')      //si se recibe S
{
Stop();                //habilitar frenado rapido del motor
}
else                   //en caso de no recibir alguna de estas se detendra
{                      //al motor
Stop();                //habilitar frenado rapido del motor
}
```

Figura 7.7.Análisis de variables para establecer sentido de giro.

Aquí se observa que se está haciendo uso de subfunciones MoverDer(), MoverIzq() y Stop(), en las cuales se habilitan los pines para que el puente H haga girar al motor en cierto sentido, dichas subfunciones se pueden ver en la Figura 7.8.

```
void MoverDer()       //funcion para giro a la derecha
{
bit_set(portb,6);     //poner en 1 pin 6 del puertoB
bit_clear(portb,7);   //poner en 0 pin 7 del puerto B
}

void MoverIzq()       //funcion para giro a la izquierda
{
bit_clear(portb,6);   //poner en 0 pin 6 del puerto B
bit_set(portb,7);     //poner en 1 pin 7 del puerto B
}

void Stop()           //funcion para detener giro con frenado rapido.
{
bit_set(portb,6);     //poner en 1 pin 6 del puerto B
bit_set(portb,7);     //poner en 1 pin 7 del puerto B
}
```

Figura 7.8. Funciones para giro de motor.

Para establecer la velocidad de giro se envía una señal PWM al pin ENABLE del puente H, para la configuración de esta señal utiliza la subfunción PWM que se muestra en la Figura 7.9, en la cual después de configurar la señal se establece el ciclo de trabajo, de este modo el puente H tiene todas las señales necesarias para mover el motor.

```
void PWM()                //funcion para crear PWM
{
setup_timer_2(T2_DIV_BY_16,255,1);  //configura timer2
setup_ccp1(CCP_PWM);                //configura modulo CCP como PWM
set_pwm1_duty(Vel);                 //configura ciclo de trabajo
}                                   //con variable Vel
```

Figura 7.9. Configuración de la subfunción PWM.

Para el conteo de los pulsos, como se mencionó anteriormente se usan 2 timers, en este caso será el TIMER0 y el TIMER1, se usa la subfunción Contar() la cual tiene la codificación que se muestra en la Figura 7.10.

```
void Contar()         //funcion para contar pulsos de encoder
{
x=get_timer0();       //obtener numero de cuentas del timer0
y=get_timer1();       //obtener numero de cuentas del timer1
if(y==128)            //comparacion para limpiar el timer1 una
{                     //vez que ha llegado a 128 pulsos
set_timer1(0);        //pues el timer0 ya llevara una cuenta mas
}
write_eeprom(0x02,x);//escritura de dato de timer0 en dir 02 de EEPROM
write_eeprom(0x03,y);//escritura de dato de timer1 en dir 03 de EEPROM
}
```

Figura 7.10. Lectura de Timer asociados al encoder.

En este punto hace falta hacer la comparación entre las cuentas que se llevan y las cuentas que se recibieron como dato, dadas las limitantes del software de simulación esto no se llevó a cabo, dejando el conteo de pulsos como una teoría de su configuración y uso.

En el caso del PIC maestro el esquema que se usará será el que se muestra en la Figura 7.11.

Figura 7.11 Diagrama de funciones del PIC maestro.

La recepción de los datos en el PIC maestro, se hará por medio de comunicación serie, recibiendo como datos: el PIC que se quiere configurar para abrir comunicación con él, sentido de giro y la velocidad de giro. El PIC maestro se mantendrá a la espera de la tercia de datos como se muestra en la Figura 7.12.

```
if(kbhit())             //espera hasta que reciba datos
{                       //por puerto serie
  valor=fgetc(COM_PC);  //recepcion de dato de pic esclavo
  sentido=fgetc(COM_PC);//recepcion dato de sentido degiro
  Vel=fgetc(COM_PC);    //recepcion de dato de velocidad
```

Figura 7.12. Comunicación por puerto Serie del PIC maestro.

Posteriormente se hará un comparativo para saber a qué PIC enviar la información recibida, como se muestra en la Figura 7.13.

```
      if(valor=='A'){       //comparacion de valor con PIC A
      Slave(0xA2);          //escritura de datos a esclavo A2
      fprintf(COM_PC,"PIC A"); //enviar por puerto serie mensaje del pic
      }                        //esclavo usado
      else if(valor=='B'){  //comparacion de valor con PIC B
   Slave(0xA4);             //escritura de datos a esclavo A4
   fprintf (COM_PC,"PIC B");//mensaje por puerto serie del pic usado
}
   else if(valor=='C'){     //comparacion de valor con PIC C
   Slave(0xA6);             //escritura de datos a esclavo A6
   fprintf (COM_PC,"PIC C");//mensaje por puerto serie del pic usado
}
   else if(valor=='D'){   //comparacion de valor con PIC
   Slave(0xA8);             //escritura de datos a esclavo A6
   fprintf (COM_PC,"PIC D");//mensaje por puerto serie del pic usado
}
   else {                   //en caso de que no sea ningun valor
   putc(valor);             //regresa por puerto serie el dato escrito
   fprintf (COM_PC,"NINGUNO");//e imprime que no se envio a ningun PIC
  }
```

Figura 7.13. Comparativo de datos recibidos en PIC maestro y envío a PIC esclavo.

Aquí se puede observar que se hace uso de una subfunción llamada Slave(dir),donde *dir* es la dirección de cada PIC esclavo, de este modo dentro de la subfunción como se aprecia en la Figura 7.14, se envían los datos en el orden que la programación del PIC esclavo los requiere, es decir la velocidad establecida, el sentido de giro y por último el número de cuentas.

```
void Slave(PICS)           //Funcion para comunicacion
{                          //de datos con PIC esclavo
do{                        //seleccionado.
   delay_ms(250);
   escribir(Vel,PICS);     //Uso de funciòn escribir para enviar Velocidad
   if(sentido=='L')        //comparaciòn de variable sentido para
   {                       //no enviar dato incorrecto de sentido.
   escribir(sentido,PICS); //Funciòn escribir para enviar sentido en este
   }                       //caso "L"
   else if(sentido=='R')   //compara sentido con letra R
   {
   escribir(sentido,PICS); //Funciòn escribir para enviar sentido en este
   }                       //caso "R"
   else                    //Si la variable sentido no recibio una L o R
   {                       //el sentido sera S para detener al motor.
   sentido='S';            //Sentido se establece como S
   escribir(sentido,PICS); //Funcion escribir para enviar sentido en este
   }                       //caso "S"
   escribir(cuenta,PICS);  //Funcion escribir para envio de cuentas a moverse
   valor=1;                //variable para terminar ciclo while
   }while(valor!=1);
}
```

Figura 7.14. Subfunción de PIC maestro para envió de datos a PIC esclavo.

Al igual que en los PIC esclavos se tiene una subfunción que escribe datos mediante I2C dicha codificación se puede observar en la Figura 7.15.

```
void escribir(dato,direccion){//Funciòn para esritura de datos por I2C
        delay_ms(50);          //retrasos para recepcion de datos
        i2c_start();           //Inicia transmisiòn de datos
        delay_ms(50);
        i2c_write(direccion); //Envia variable direccion
        delay_ms(50);
        i2c_write(dato);       //Envia variable dato
        delay_ms(50);
        delay_ms(50);
        i2c_stop();            //Detiene comincaciòn
        delay_ms(50);
        valor1=1;              //variable para terminar ciclo while
}
```

Figura 7.15. Subfunción para envío de datos usando I²C

Después de enviar los datos al PIC esclavo, regresará por el puerto serie el nombre de PIC al que se enviaron los datos.

7.1.3 SIMULACIÓN

Una vez que se tienen los códigos, se construirá el modelo en PROTEUS para su simulación. Para ello se hará uso de cuatro PIC esclavos, cada uno acoplado a un motor mediante un puente H L293D; también irán conectados a los cables que son usados para la comunicación I2C con el PIC maestro, el cual tendrá acoplado un puerto serie virtual por el cual recibirá los datos desde la PC. En la Figura 7.16 se puede apreciar el diseño final del prototipo de simulación.

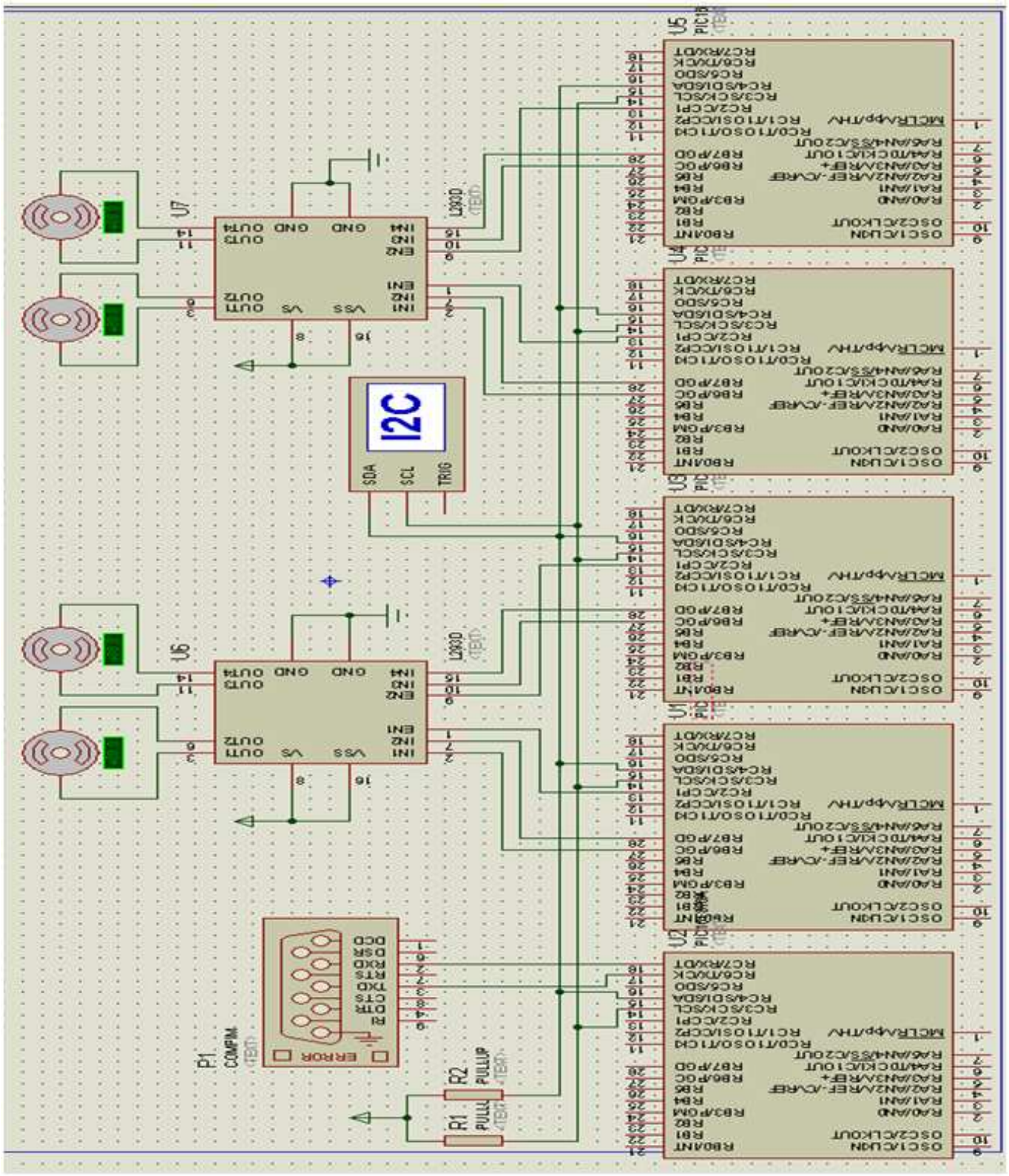

Figura 7.16. Diseño para simulación en PROTEUS

7.1.4 CREACIÓN DE UNA INTERFAZ

Para el envío de datos desde la PC, se desarrolló una interfaz en la cual fuese más sencilla la creación de botones de control para cada motor y establecer la comunicación serial para el envió de la tercia de datos necesarios en el PIC maestro, para esta tarea se creó un panel en Visual Basic 6.0, quedando como se muestra en la Figura 7.17.

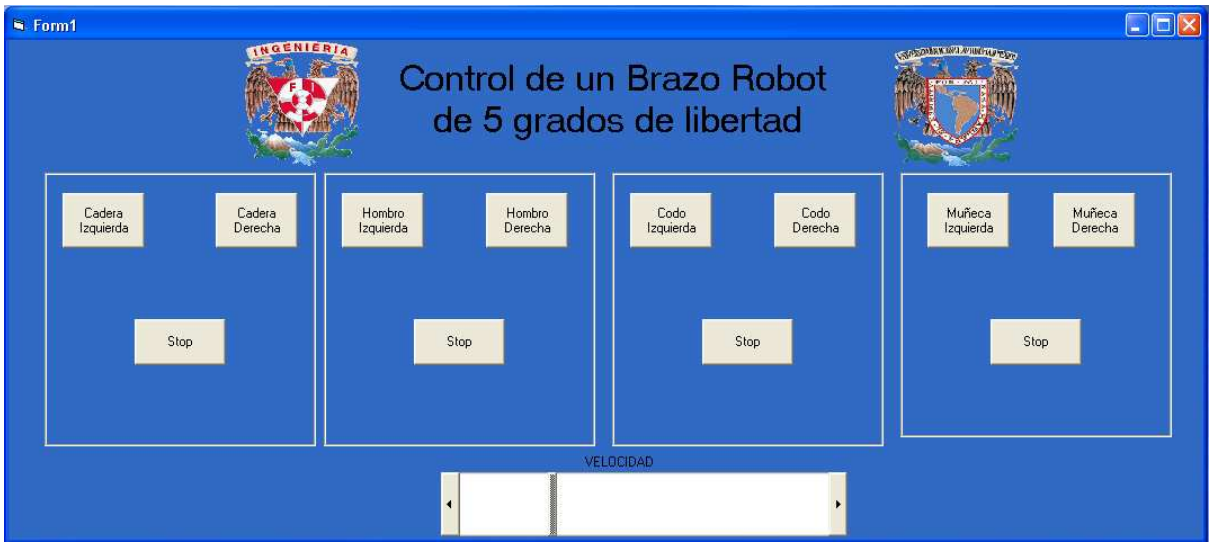

Figura 7.17. Panel de Visual Basic para control de motores.

En este panel se dividieron los botones por cada motor que será manipulado. Existen dos botones asociados al sentido de giro de cada motor, así como un botón de paro, en la parte inferior se tiene un scroll el cual nos proporciona el dato de la velocidad. Cada botón tendrá 3 parámetros a enviar por el puerto serial: una letra que lo identifique como articulación, otra que indique el sentido de giro y el valor de velocidad. En la siguiente tabla se muestran los datos que envía cada botón.

Tabla 7.1. Datos enviados por cada botón del panel de Visual Basic

Botón	Dirección PIC	Sentido	Velocidad
Cadera izquierda	A	L	Obtenida del Scroll
Cadera Derecha	A	R	
Hombro izquierda	B	L	
Hombro derecha	B	R	
Codo izquierda	C	L	
Codo derecha	C	R	
Muñeca izquierda	D	L	
Muñeca derecha	D	R	

Para realizar la comunicación es necesario configurar el comando MSCOMM el cual habilita el puerto, para ello se usan parámetros estándar de comunicación serial. Como se muestra en la siguiente figura.

```
Private Sub Form_Load()      'Configuracion del puerto de comunicaciones
With MSComm1                 'como MsComm1
.CommPort = 3                'puerto serie com3
.RThreshold = 1              'espera 1 dato en buffer de entrada para analizarlo
.RTSEnable = True            'peticion de envio habilitada
.Settings = "9600,n,8,1"     'velocidad, paridad,bits de informacion,bit de parada
.SThreshold = 1              'espera 1 dato en buffer de salida para analizarlo
.PortOpen = True             'habilitar puerto
End With
End Sub
```

Así, por ejemplo, si se presiona el botón *Cadera Izquierda*, se envía primero una A, posteriormente una L y por último el valor del scroll como se muestra en el siguiente código.

```
Private Sub Cad_Der_Click(Index As Integer) 'Funcion del Boton Cadera Derecha al hacer clic
Dim Letra As String                         'Variable "Letra" como cadena
Dim V As String                             'Variable "V" como cadena
Letra = "A"                                 'Variable letra se le asigna valor de A
MSComm1.Output = Letra                      'Envia por puerto de comunicacion la variable Letra
Letra = "R"                                 'Variable letra se le asigna valor de R
MSComm1.Output = Letra                      'Envia por puerto de comunicacion la variable Letra
V = Chr(HScroll1.Value)                     'variable V con valor del scroll convertido a cadena
MSComm1.Output = V                          'Envia por puerto de comunicacion la variable V
End Sub
```

7.1.5 COMPARATIVO ENTRE PLC Y PIC

Es posible implementar un dispositivo Programable de Control similar a un PLC basado en un microcontrolador PIC cualquiera, lo cual hace pensar en las ventajas y desventajas que se pueden tener al usar uno u otro.

De principio se sabe que el precio de los PIC es notablemente menor al de un PLC de cualquier marca. No obstante utilizar un PLC en un ambiente industrial puede tener muchas ventajas de seguridad y confiabilidad sobre un microcontrolador.

Es por ello que se pueden analizar algunas formas de disminuir las desventajas que presenta el de menor costo utilizando algunos circuitos básicos como acoplamiento entre la planta y el controlador.Se pueden utilizar opto acopladores en las entradas del PIC para lectura de señales todo o nada (TTL) y protegerlas de picos superiores a los 5 volts. El consumo de estos circuitos es mínimo así que puede con alimentarse la misma fuente que el microcontrolador.

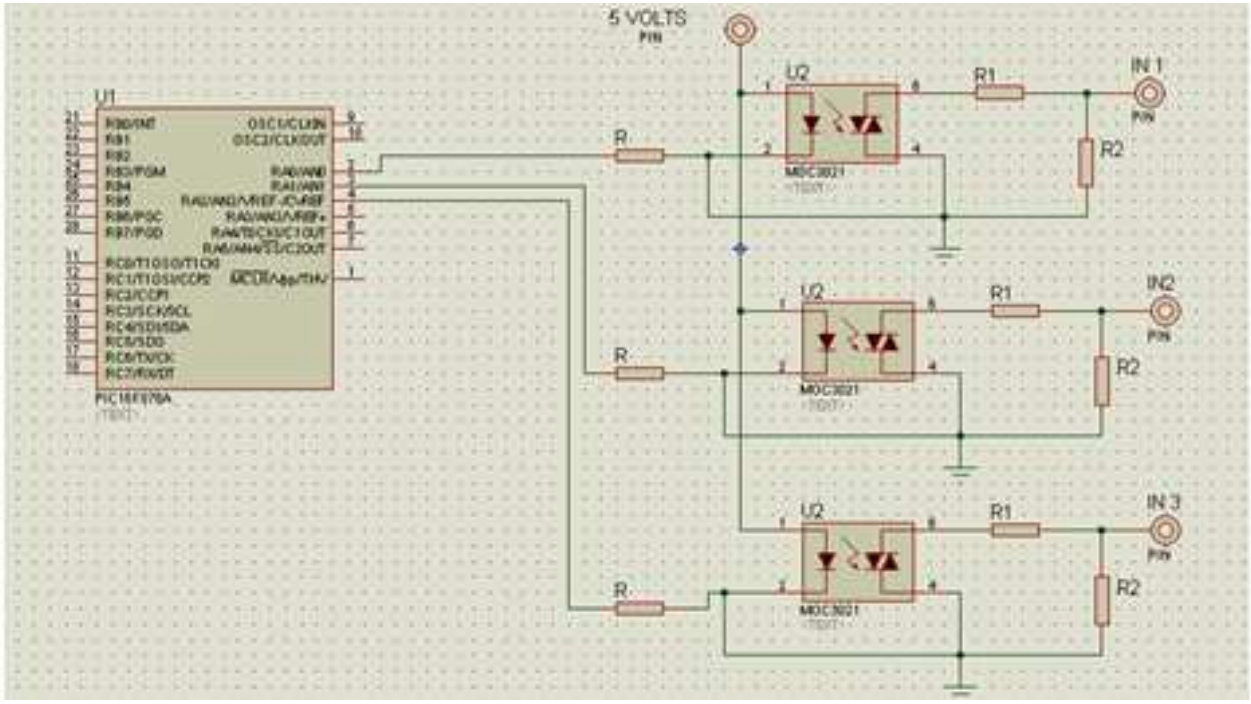

Figura 7.18. Entradas protegidas con optoacoplador MOC3021

De esta forma se puede complementar el circuito que se utilizó para la simulación, ya que no tiene contemplados los microswitches, que deberían estar conectados a las entradas digitales.

El controlador original del Scorbot tiene algunas salidas extras destinadas a controlar algunos elementos de una línea de producción como bandas y sensores. Este tipo de salidas a relevador son muy comunes en los PLC, lo cual es una ventaja para este proyecto, pero es posible utilizar un circuito integrado con un arreglo de transistores Darlington para controlar relevadores con un PIC. La función del circuito integrado es amplificar la corriente para lograr excitar las bobinas de los relevadores.

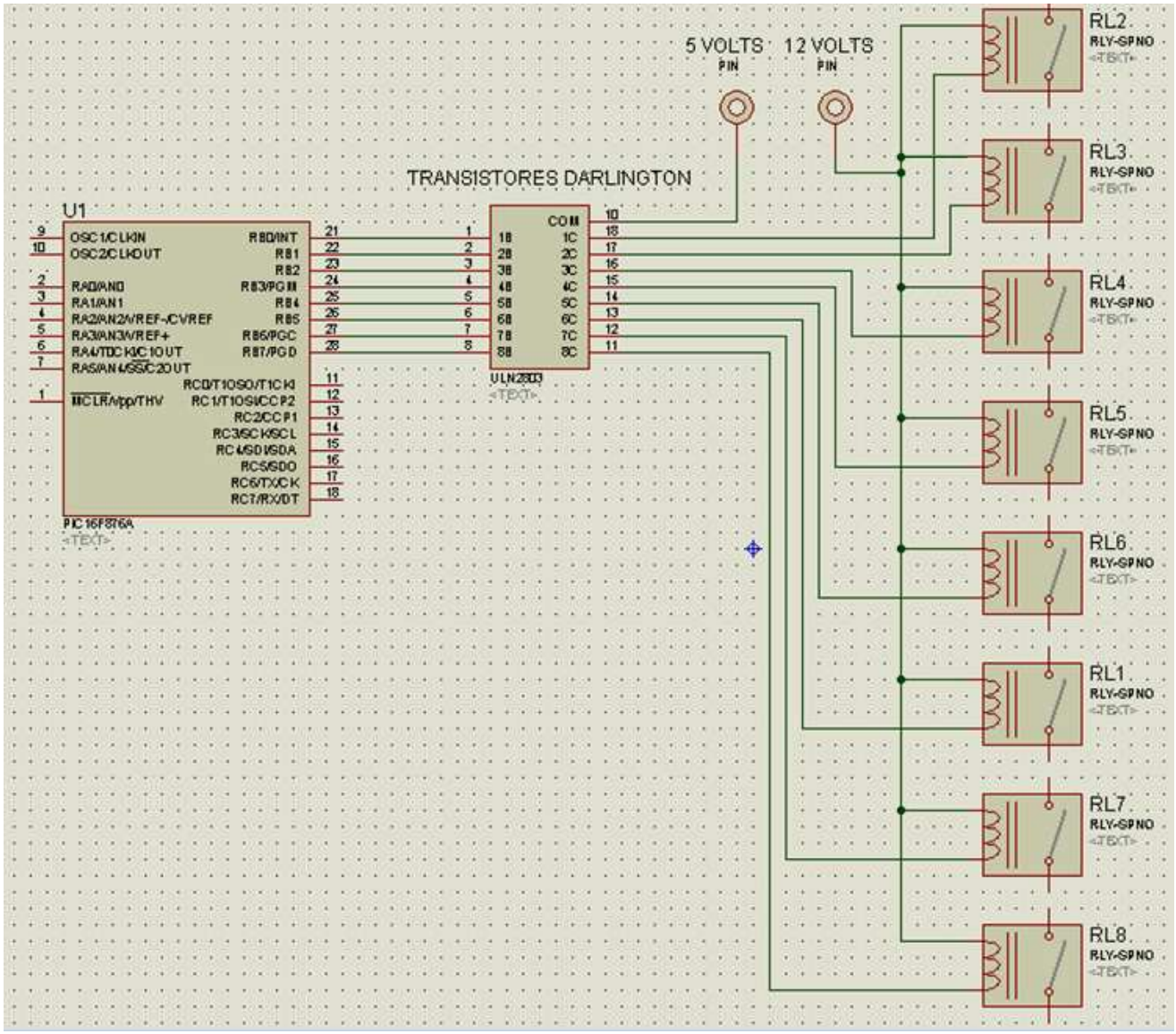

Figura 7.19. Salidas a relevador utilizando circuito ULN2803

Se puede notar el que PIC tiene muchas desventajas frente al PLC para aplicaciones donde se requieren numerosas entradas y salidas, y donde el ambiente industrial puede afectar la confiabilidad de las lecturas. Esto se debe a que algunos PLC son modulares, por lo cual incrementar el número de entradas y salidas a transistor o a relevador es muy sencillo.

Otro factor de gran importancia a favor para algunos PLC es la comunicación. Como se observó durante el desarrollo del proyecto, al tener que utilizar más de dos PIC para la simulación se hizo necesario implementar dos tipos de comunicaciones: una entre los dispositivos y otra diferente con la PC.