

Implementación de un Circuito de Control en un Brazo Robótico (Diciembre 2010)

Rios Limón Fernando, González Franco Vicente, Quirino de Dios Ildelfonso, Guerra Marín Rubén

Centro de Enseñanza Técnica y Superior

Abstract— Este proyecto consiste en el diseño de circuitería electrónica y programación de un microcontrolador, para llevar a cabo el control de un brazo robótico manipulado con el uso de *joysticks*. Se incluyen aspectos teóricos relevantes para el desarrollo del proyecto y una sección de implementación donde se explican los distintos módulos del circuito de control realizado. Es un producto de evaluación para las asignaturas de Electrónica Analógica e Instrumentación y Control.

I. INTRODUCCIÓN

Es difícil definir con precisión a un robot, debido a que su campo de aplicación es amplio y no siguen un protocolo estricto que defina su construcción. En cambio, son dispositivos que pueden realizar las tareas que le sean asignadas y dependen directamente de su diseño mecánico, electrónico y algoritmos de control. Sin embargo podemos decir que un robot es un manipulador reprogramable y multifuncional diseñado para mover materiales, piezas o dispositivos especiales a través de movimientos variables programados para realizar una variedad de tareas.

La idea general de utilizar un robot es tener un mecanismo programable para facilitar las tareas que son realizadas por los seres humanos. Existen una gran variedad de robots que se encuentran bajo distintas arquitecturas. Dentro de los robots se encuentran aquellos de usos industriales que pueden ser denominados también poliarticulados. Nos enfocaremos en el brazo robótico. Este es un manipulador que consta de diferentes grados de libertad o aquellos movimientos que son posible realizar por el mecanismo. El brazo que se utilizó para el desarrollo del proyecto es un *XR-ROBOT* fabricado por la empresa *RHINO ROBOTICS*. Ver figura 1

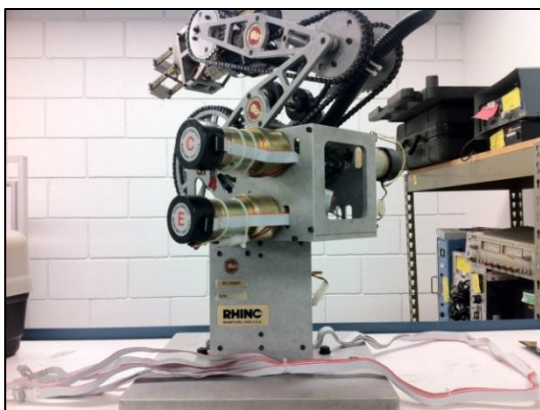


Fig. 1. Vista frontal de XR-ROBOT fabricado por RHINO Robotics

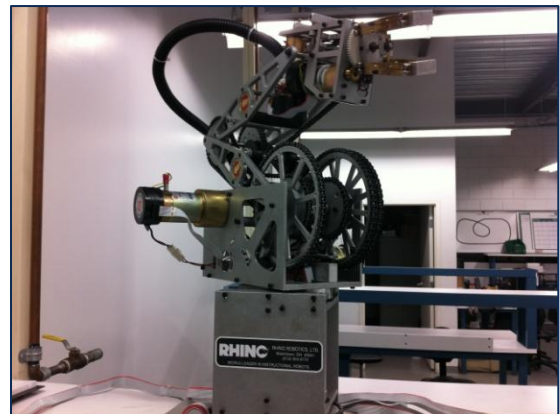


Fig. 2. Vista trasera de XR-ROBOT fabricado por RHINO Robotics

Con lo anterior, se quiere decir que el proyecto no consistió en el diseño mecánico de un manipulador debido a factores de tiempo y gastos monetarios. En cambio, el proyecto tiene como enfoque realizar la circuitería necesaria para llevar a cabo el control de un mecanismo existente. El brazo robótico con el que se trabajó tiene 5 grados de libertad lo que permite tener un buen control de los movimientos a realizar por el robot.

Se tienen como objetivos específicos del proyecto, diseñar un circuito de control que logre tener un dominio de los distintos motores con los que cuenta. Además, permitir que el usuario mediante el uso de palancas de mando o *joysticks* tenga el control de todos los cinco grados de libertad que posee el mecanismo.

El proyecto busca cubrir los distintos conocimientos adquiridos en las asignaturas de Electrónica Analógica I e Instrumentación y Control. El enfoque de la primera asignatura se le da al trabajar con transistores, específicamente con el diseño de un puente H. En cuanto a la segunda, debido a que es una materia que forma parte de la especialidad en Robótica y Automatización Industrial, se tuvo un especial enfoque en el uso de dispositivos de medición como lo serían los sensores y los actuadores para controlar un mecanismo, en este caso los motores de corriente directa o DC. Es posible considerar a un joystick como sensor si se toma en cuenta que se obtienen lecturas de variaciones de valores de resistencia, los cuales son manipulados por un usuario. A continuación se establece un marco teórico para documentar los aspectos que fueron implementados a lo largo del proyecto.

II. MARCO TEÓRICO

Los transistores BJT o de unión bipolar, se encuentran contruidos con tres regiones de semiconductor dopado y dos uniones conocidas como *pn*. Las regiones son conocidas como base, colector y emisor. La base es la región más delgada y con menor dopaje. El colector es la región más grande del transistor BJT. El emisor es la región con mayor dopaje. El flujo de corriente en un transistor BJT se basa en la existencia de electrones libres y huecos. Los dos tipos de unión bipolar son *nnp* y *pnp*. [1]

Un transistor puede operar como un interruptor o *switch* en las regiones conocidas como corte y saturación. En corte las dos uniones *pn* se polarizan inversamente, de manera que no haya corriente de colector. El transistor funciona como un *switch* abierto entre colector y emisor, tal como se observa en la Fig. 3. En saturación las dos regiones *pn* se polarizan directamente y la corriente de colector es máxima. El transistor se comporta como un *switch* cerrado entre emisor y colector, tal como se aprecia en la Fig. 4. [1] Estos conceptos son parte esencial para comprender el funcionamiento del puente H que se implementa en el proyecto.

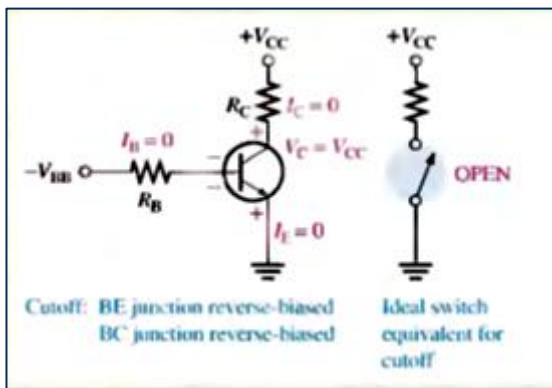


Fig. 3. Transistor BJT en su región de corte

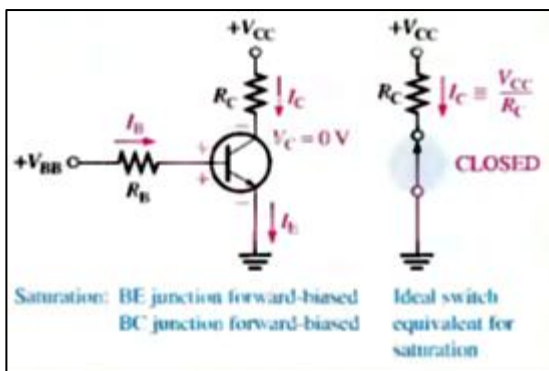


Fig. 4. Transistor BJT en su región de saturación

El puente H es un sistema de conmutación que tiene dos señales de entrada digitales y estas a su vez sirven para controlar el giro de motores de corriente directa (motores DC). Cuando el sistema detecta un 1 lógico en una de sus entradas de control y un cero lógico en la otra, el motor es conectado a la fuente de alimentación con determinada polaridad y si las señales de entrada se invierten, entonces el motor es polarizado en sentido contrario y hace girar el motor en sentido inverso.

El puente H es llamado de esa manera debido a que tiene 4 elementos en las esquinas y el motor conectado horizontalmente en el centro formando una forma de la letra H. EL factor clave es que hay en teoría cuatro elementos de conmutación o *switches* dentro del puente, los *switches* son activados en pares diagonalmente opuestos permitiendo el flujo de la corriente como se muestra en la figura 5. La corriente fluye y el motor empieza a girar en dirección positiva y al activar los otros dos *switches* diagonalmente opuestos, el motor gira al lado opuesto, es decir, en dirección negativa. [2]

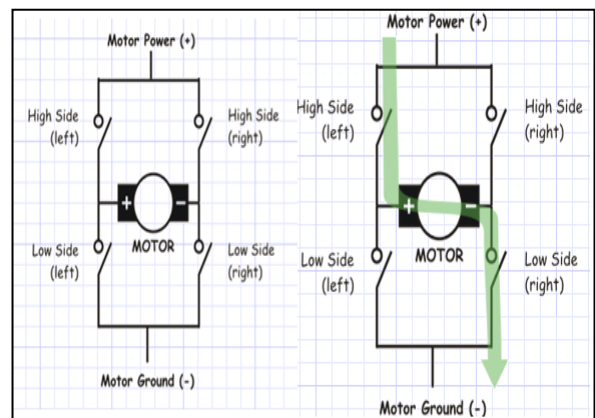


Fig. 5. Representación gráfica de un puente H.

Para construir un puente H, se pueden utilizar diferentes elementos de *switches* desde *switches* manuales, relevadores y transistores (BJTs, MOSFETs y otros). Los transistores más sencillos de implementar son los BJTs ya que son fáciles de manejar y no son tan sensibles como los MOSFETs. Se requieren transistores PNP y NPN para hacer los *switches* en sentido directo e inverso, los más comunes son 2N2222 y TIP 31 Y 32 (Fig. 6) [3], [4].

Existen ya, circuitos integrados que contienen puentes H capaces de controlar dos o más motores, tal es el caso del integrado LN293B, que es un driver de 4 canales capaz de proporcionar una corriente de salida de hasta 1A por canal. Cada canal es controlado por señales de entrada compatibles TTL y cada pareja de canales dispone de una señal de habilitación que desconecta las salidas de los mismos.

Disponen de una patilla para la alimentación de las cargas que se están controlando, de forma que dicha alimentación es independiente de la lógica de control. La Figura 7 muestra el encapsulado de 16 pines, la distribución de patillas y la descripción de las mismas.

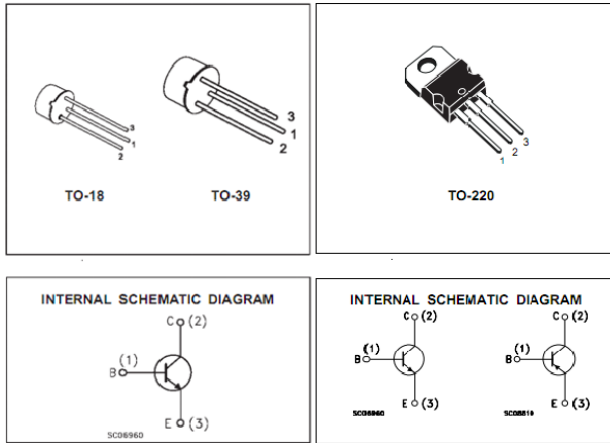


Fig. 6. Transistores BJT 2N2222 y TIP 31,32

Pin	Nombre	Descripción	Patillaje
1	Chip Enable 1	Habilitación de los canales 1 y 2	
2	Input 1	Entrada del Canal 1	
3	Output 1	Salida del Canal 1	
4	GND	Tierra de Alimentación	
5	GND	Tierra de Alimentación	
6	Output 2	Salida del Canal 2	
7	Input 2	Entrada del Canal 1	
8	Vs	Alimentación de las cargas	
9	Chip Enable 2	Habilitación de los canales 3 y 4	
10	Input 3	Entrada del Canal 3	
11	Output 3	Salida del Canal 3	
12	GND	Tierra de Alimentación	
13	GND	Tierra de Alimentación	
14	Output 4	Salida del Canal 4	
15	Input 4	Entrada del Canal 4	

Fig. 7. Descripción de los pines del LN293B

Por lo general, todo sistema que procesa información binaria para controlar un proceso analógico requiere una etapa de entrada analógico – digital y una etapa de salida digital –analógica (convertidores ADC y DAC). Para reducir costos en los diseños que no requieren alta resolución en la etapa de salida, es posible sustituir el DAC por un algoritmo de Modulación por Ancho de Pulsos (PWM – Pulse Width Modulation). Una *unidad PWM* permite asignar cierta duración de tiempo en alto o en bajo a un dato digital de n bits que se considera salida de la etapa procesadora.

El comparador determinará si el dato aplicado a la entrada de la unidad es igual al valor binario del contador que cambia constantemente. El tiempo que durará la señal en alto depende de la cantidad de pulsos de reloj que se apliquen hasta que el contador presente un dato binario mayor o igual al de la entrada. A la salida de la unidad PWM es necesario conectar un filtro *RC* (Pasa Bajas) para determinar el nivel analógico propuesto por el filtro. El periodo completo de un ciclo PWM es igual al producto del periodo del reloj de la señal de referencia (reloj del sistema) con $2n$, donde n es el número de bits del contador propuesto. Queda dado por la siguiente fórmula [5]:

$$T_{PWM} = (T_{reloj})(2^n)$$

La principal ventaja de un circuito de PWM sobre un controlador que se base en la variación lineal de la potencia suministrada a una carga mediante cambio resistivo es la eficiencia. A una señal de control del 50%, el PWM usará cerca del 50% de la potencia total, de la cual casi toda será transferida a la carga. En un controlador tipo resistivo, de un 50% de potencia que se quiera transferir a la carga se estima que le puede llegar cerca de un 71%. El otro 21% se pierde en forma de calor. [6]

El *gripper* es la pinza que se encuentra en el motor A de nuestro brazo. Su función es la de agarrar objetos. La forma en la que trabaja es mediante un botón, el cuál al ser presionado cierra la pinza si es que estaba abierta, o la abre de ser lo contrario.

III. IMPLEMENTACIÓN

1. MÓDULO ADC

El PIC 18f4550 posee un módulo de conversión analógico a digital para poder registrar un voltaje de entrada y transformarlo en información binaria para poder ser usado como parámetros de los actuadores que en este proyecto se usa para la modulación de pulsos que más adelante se explicara a detalle.

Los procesos generales del módulo ADC consiste en definir entradas análogas del PIC. Estas entradas análogas están implementadas generalmente en el puerto A, no cualquier entrada puede ser definida como entrada análoga. Además de la entrada análoga, también se tiene que usar voltajes de referencia para comparar el voltaje de entrada.

Para el comienzo de la conversión, únicamente se debe indicar cuándo comenzar, ya que el PIC automáticamente realiza la conversión con una resolución de 10 bits guardados en 2 registros de 8 bits cada uno y cuya justificación (forma de llenar los registros) se puede modificar.

1.1 Programación del módulo ADC.

En la programación del módulo de ADC del PIC se utilizan los siguientes registros:

- ADCON0
- ADCON1
- ADCON2
- ADRESH
- ADRESL

Los registros son localidades de memoria de 8 bits también llamado byte, donde el programa ya tiene una función definida para es parte de la memoria para poder acceder a su información y tenerla como parámetro de operaciones. A continuación se explica cada uno de los registros anteriormente mencionados.

1.1.1 ADCON0.

Este registro es uno de los registro de control del módulo del ADC, en el podemos definir el canal de donde vamos a

toma la entrada análoga a convertir. También se encarga de permitir la conversión y de controlar el inicio y fin de este proceso. Para poder acceder a estos bits de control, se puede hacer haciendo referencia a ellos con las siguientes denominaciones:

- CHS3:CHS0 (bit 5-2). Bits indicadores de canal de entrada análoga.
 - 0000 = Channel 0 (AN0)
 - 0001 = Channel 1 (AN1)
 - 0010 = Channel 2 (AN2)
 - 0011 = Channel 3 (AN3)
 - 0100 = Channel 4 (AN4)
 - 0101 = Channel 5 (AN5)
 - 0110 = Channel 6 (AN6)
 - 0111 = Channel 7 (AN7)
 - 1000 = Channel 8 (AN8)
 - 1001 = Channel 9 (AN9)
 - 1010 = Channel 10 (AN10)
 - 1011 = Channel 11 (AN11)
 - 1100 = Channel 12 (AN12)
 - 1101 = Unimplemented
 - 1110 = Unimplemented
 - 1111 = Unimplemented
- GO/DONE (bit 1). Estatus de la conversión; cuando el bit ADON está en "1":
 - 1 = A/D en proceso
 - 0 = A/D finalizado
- ADON (bit 0). Bit de inicio.
 - 1 = Modulo A/D habilitado
 - 0 = Modulo A/D deshabilitado

1.1.2 ADCON1.

Registro de control donde se puede indicar los voltajes de referencia como entradas del pin AN2 (voltaje negativo) y AN3 (voltaje positivo) o tomados de las fuentes de alimentación del PIC (Vss y Vdd). Además de controlar también la asignación de entrada digital o análoga de los pin designados para esta tarea. Los bits son denominados de la siguiente manera:

- VCFG0 (bit 5). Bit de configuración de voltaje negativo.
 - 1 = Vref- AN2
 - 0 = Vss
- VCFG0 (bit 4). Bit de configuración de voltaje positivo
 - 1 = Vref+ AN3
 - 0 = Vdd
- PCFG3:PCFG0 (bit 3-0). Bits de control de configuración de puerto análogo-digital. (Ver Data sheet para información de configuración)

1.1.3 ADCON2.

Registro controlador de tiempos del proceso de conversión y salida. Con este registro podemos controlar el nivel del tiempo (algún valor relacionado con la Frecuencia del PIC); los retardos requeridos para iniciar un nuevo proceso de conversión (las unidades de tiempo se conocen como TAD); y la forma de cómo acomodar los bits de salida. Los bits de control están asignados de la siguiente forma:

- ADFM (bit 7). Bit de formato de resultado.
 - 1 = justificado a la derecha
 - 0 = justificado a la izquierda
- ACQT2:ACQT0 (bit 5-3). Bits de selección de tiempo antes del inicio de conversión.
 - 111 = 20 TAD
 - 110 = 16 TAD
 - 101 = 12 TAD
 - 100 = 8 TAD
 - 011 = 6 TAD
 - 010 = 4 TAD
 - 001 = 2 TAD
 - 000 = 0 TAD
- ADCS2:ADCS0 (bits 2-0). Bits de selección de reloj A/D
 - 111 = FRC (reloj derivado del oscilador)
 - 110 = FOSC/64
 - 101 = FOSC/16
 - 100 = FOSC/4
 - 011 = FRC (reloj derivado del oscilador)
 - 010 = FOSC/32
 - 001 = FOSC/8
 - 000 = FOSC/2

1.1.4 ADRESH y ADRESL.

Estos registros guardan el resultado de la conversión A/D, al terminar la conversión el PIC automáticamente limpia el bit GO/DONE del registro ADCON0 para indicar que termino y que el resultado se encuentra en estos registros.

Al tener la conversión una resolución de 10 bits, se ocupan 2 registros para guardar el resultado, pero como los 2 registros proporcionan un total de 16 bits disponibles para guardar dicha información, queda huecos no utilizados en alguno de los registros. Para controlar en que registro queremos que se empiecen a guardar los datos, se disponen de un bit de configuración anteriormente explicado (ADFM del registro ADCON2). Pero justificación a la derecha o izquierda no nos explica a detalle cómo se acomodan. Por lo que es preciso indicar la forma de cómo se encuentran acomodado en cada formato:

- Justificado a la izquierda. ADRESH{bit9, bit8, bit7, bit6, bit5, bit4, bit3, bit2} y ADRESL{bit1, bit0, 0, 0, 0, 0, 0}.
- Justificado a la derecha. ADRESH{0, 0, 0, 0, 0, 0, bit9, bit8} y ADRESL{ bit7, bit6, bit5, bit4, bit3, bit2, bit1, bit0}.

Este registro puede ser leído y escrito en cualquier momento. Además este registro no cambia de valor cuando se reinicia el PIC, por lo que siempre guarda el último valor que se le fue escrito.

1.2 Código de programación del módulo ADC.

El código de programación en C que utiliza los registros anteriormente mencionado, se agrupo en métodos denominados “setting_ADC()” que configura el PIC para la conversión y “start_ADC()” que comienza con el procesos y no deja continuar hasta que termine. A continuación se muestra las líneas código de cada uno de los métodos con su explicación (las líneas son continuas y en el orden que se muestran).

1.2.1 “setting_ADC()”.

`TRISA = 0b00000011`. Se fija las entradas y salidas del puerto A (“1” y “0” respectivamente).

`ADCON0 = 0b00000000`. Se toma el canal 0 como primera entrada análoga y no se permite aun la conversión.

`ADCON1 = 0b00001101`. Voltajes de referencia tomados de V_{ss} y V_{dd} y se define el canal 0 y canal 1 como entradas análogas.

`ADCON2 = 0b00101111`; Justificación a la izquierda (de esta forma se facilita la adquisición de valor para el módulo PWM); 12 TAD de retardo y reloj FRC.

1.2.2 “start_ADC()”.

`ADCON0bits.ADON = 1`. Se permite la conversión

`ADCON0bits.GO = 1`. Se inicia la conversión (línea de código no verificado como necesario)

`while (ADCON0bits.GO_DONE)`. Se espera a que este bit se limpie (poner en “0”) para continuar

2. MÓDULO PWM.

Además del módulo de ADC, el PIC18f4550 posee también un módulo de PWM donde el programador puede establecer un periodo y un *Duty Cycle*. La frecuencia o periodo (inversa de la frecuencia) son definidos por la frecuencia de oscilación del reloj de operación del PIC y el periodo establecido.

En este proyecto el módulo PWM es una señal que responde a los resultados de la conversión ADC variando el ancho de pulso, por lo que el programa toma el valor resultante (ADRESH y ADRESL) para poder cambiar los valores del Cycle Duty.

2.1 Programación del módulo PWM.

El modulo del PWM tiene la función de crear una señal cuadrada y la de mandarla a registros de salida, es decir que la señal se mantiene mientras que el PIC realiza otras funciones o tareas. Cuando se vuelve a ingresar al modo

PWM, lo que hace esta es la de cambiar los valores de la señal, pero no es la que la genera si no la que establece dicha señal.

La programación del PWM es un poco más compleja debido a que en esta parte se tiene que considerar el periodo tanto del todo el ciclo como el periodo en el que mantenemos en algo el pulso. Por lo que además de investigar los registros que se involucran en el funcionamiento de este módulo, también se tuvo que considerar ecuaciones proporcionadas por el datasheet del PIC para calcular el periodo y el Cycle duty. [7]. Las ecuaciones son las siguientes:

- (1) $PWM\ Period = [(PR2) + 1] \cdot 4 \cdot T_{osc} \cdot (TMR2\ Prescale\ Value)$
- (2) $PWM\ Duty\ Cycle = (CCPRxL:CCPxCON<5:4>) \cdot T_{osc} \cdot (TMR2\ Prescale\ Value)$

Estas fórmulas nos ayudan a saber que registros están involucrados en los valores de nuestra señal PWM. Sin embargo, hay otros registros que configuran esta modalidad que a continuación se explican.

A. PR2.

PR2 es un registro de 8 bits cuya función es la de establecer un periodo. Como se pudo ver en la fórmula 1 anteriormente mostrada, el periodo en función del valor del PR2 ya que los demás valores que participan tiene la función de permitir obtener valores proporcionales a este valor, ya que el registro solo llega a un valor 255, y no se puede ingresar a los bits de este registro ya que no es necesario.

En este proyecto el valor de PR2 se mantiene fijo debido a que solo queremos tener un cambio de pulso tiempo en alto. Sin embargo si hacemos cambios en este registro, se obtendría un modulador de frecuencias, que también es una alternativa para el control de motores.

B. T2CON.

Registro del controlador del tiempo. En el modo PWM, el reloj asignado para que controle este proceso es el “timer2” de tal modo que nuestro PWM no tiene por qué ser directo a la frecuencia del PIC. La función de este registro es la de controlar la postscala y prescala del PWM. La prescala es el único valor de este registro que participa en la frecuencia del PWM, la postscala es un valor que controla el tiempo después de que termina el proceso de PWM y de sacar la señal por los registros de salida. Los bits de control son los siguientes:

- T2OUTPS3: T2OUTPS0 (bit 6-3). Bits de selección de postscala.
 - 0000 = 1:1 Postscala
 - 0001 = 1:2 Postscala
 - 1111 = 1:16 Postscala
- TMR2ON (bit 2). Bit de encendido del “Timer2”
 - 1 = Timer2 esta encendido
 - 0 = Timer2 está apagado

- T2CKPS1:T2CKPS2 (bit 1-0). Bits de selección de prescala
 - 00 = Prescala es 1
 - 01 = Prescala es 4
 - 1x = Prescala es 16

C. CCPRxL y CCPRxH

Estos registros realizan la misma función, que es la de establecer un Cycle Duty, su valor máximo es de 255. Se diferencia en que el CCPRxL es la que el programador modifica para realizar dicha función y el CCPRxH es el que el PIC toma el valor para generar su señal de salida y es modificado al final del proceso del módulo PWM.

Como el PIC tiene manera de generar hasta 2 señales PWM con la misma frecuencia pero diferente ancho de pulso, existen varios registros CCPRxL y CCPRxH. En la denominación de estos registros, la “x” no forma parte del verdadero nombre, esta “x” es usada en la hoja de datasheet para referirse a estos registros en generar ya que todos hacen lo mismo. [7]. En la programación esta “x” se debe sustituir por el número del PWM que se programa (1 o 2).

D. CCPxCON

Los elementos usados en el módulo PWM también pueden ser usados como comparador y capturador. Debido a esto se debe contar con un registro en la que podemos seleccionar el modo que se va a utilizar para que se configuren adecuadamente los elementos. Para eso está el registro CCPxCON, es en este donde especificamos que vamos a usar el PWM.

Como el Cycle duty puede tener un valor de hasta 1023 (10 bits que se pueden conseguir en el módulo ADC) y el registro CCPRxL solo llega a 255(8 bits), es en el CCPxCON donde se pueden guardar los otros 2 bits menos significativos que según nuestra fórmula 2 (PWM Duty Cycle), son encadenado para determinar el Cycle duty. Los bits de control son los siguientes:

- DCxB1:DCxB0 (bit 5-4). Bits menos significativos del Cycle duty.
- CCPxM3:CCPxM0 (bit 3-0). Bits de selección de módulo.
 - 0000 = ningún módulo seleccionado.
 - 11xx = modo PWM.

Nota: solo se mencionan estas configuraciones de bit por que los demás no forman parte del módulo PWM y se descartó su importancia.

a. Código de programación del PWM

La programación que se implementó al final en el PIC para el PWM fue uno hecho a través de librerías que implementan ya los registros anteriormente mencionado, simplificando el código a solo referirnos a los métodos como OpenPWMx(255) y otros métodos de configuración. Sin

embargo, al inicio se pensó realizarlo de forma explícita la programación, es decir, usando los registros directamente y realizar el proceso con plena conciencia de los que se hacía, por lo que a continuación se mostrara el código que implementa directamente los registros divididos en código de configuración y de inicio.

- “setting_PWM()” implementación explícita de registros.

```
TRISBbits.TRISB3 = 0;
```

```
TRISCbits.TRISC2 = 0;
```

Se habilitan salidas de la señal PWM.

```
PR2 = 0xff;
```

Se fija el periodo máximo.

```
CCP2CON = ADRESL>>2;
```

```
CCP1CON = ADRESL>>2;
```

Se recorre los bits menos significativos 2 posición a la derecha para que queden en la posición deseada en el registro CCPxCON.

```
CCP2CONbits.CCP2M3 = 1;
```

```
CCP2CONbits.CCP2M2 = 1;
```

```
CCP2CONbits.CCP2M1 = 0;
```

```
CCP2CONbits.CCP2M0 = 0;
```

```
CCP1CONbits.CCP1M3 = 1;
```

```
CCP1CONbits.CCP1M2 = 1;
```

```
CCP1CONbits.CCP1M1 = 0;
```

```
CCP1CONbits.CCP1M0 = 0;
```

Configuración de los demás bits de forma individual para no cambiar los bits de los ya anteriormente agregados.

```
CCPR1L = ADRESH;
```

```
CCPR2L = ADRESH;
```

Fijar el Cycle duty con el resultado de la conversión ADC.

```
PIR1bits.TMR2IF = 0.
```

Se limpia interrupción condicional del Timer2

```
T2CON = 0b00000000;
```

Se fija una postescala de 1, aun no se habilita el timer2 y se fija una prescala de 1.

- “start_PWM()” implementación explícita de registros

```
T2CONbits.TMR2ON = 1
```

Se habilita el timer2 para empezar con el módulo PWM

```
while (PIR1bits.TMR2IF==0)
```

Se espera a una interrupción por desborde.

- “setting_PWM()” implementación de librerías.

```
SetOutputPWM1(SINGLE_OUT,PWM_MODE_1);
```

Se selecciona una señal simple y se active el modo PWM.

```
OpenTimer2(TIMER_INT_OFF & T2_PS_1_1 &
T2_POST_1_1);
```

Se deshabilita las interrupciones se elije prescala 1 y postcala 1.

```
OpenPWM1ConfigIO();
```

Se configuran las salidas del PWM

```
OpenPWM2(255);
```

```
OpenPWM1(255);
```

Se inicializa el PWM fijando un periodo

- “start_PWMx()” implementación de librerías.

El código posee 2 metodos con el mismo algoritmo, pero refiriéndose a diferentes salidas y entradas por lo que solo se mostrara uno solo.

```
ADCON0 = 0b00000000;
```

Selección del canal analoga.

```
start_ADC();
```

Inicio de la conversión ADC.

```
if(ADRESH>140)
```

```
{
  LATAbits.LATA5 = 0;
  SetDCPWM1(((unsigned
int)ADRESH)*4);
  LATAbits.LATA4 = 1;
}
```

```
if(ADRESH<121)
```

```
{
  LATAbits.LATA4 = 0;
  SetDCPWM1((255-(unsigned
int)ADRESH)*4);
  LATAbits.LATA5 = 1;
}
```

Se verifica si el resultado de la conversión es mayor de 140 para girar en un solo sentido fijando el *Duty Cycle* con respecto al resultado. Si es menor entonces se verifica si es menor de 121 para cambiar el sentido y fijar el *Duty Cycle* con un complemento a 255.

3. MÓDULO PUENTE H

Una vez que se analizaron los aspectos del funcionamiento del transistor BJT como *switch* y la aplicación del principio en el control de motores de corriente continua o DC, se decidió implementar un circuito para lograr el control de dirección de los motores con los que cuenta el brazo robótico utilizado. El circuito implementado se aprecia en la Fig. 8.

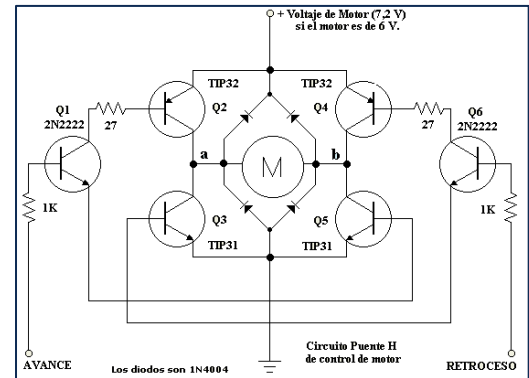


Fig. 8. Puente H utilizando transistores

Para construir el circuito que se mostró en la figura anterior, se decidió realizar el acomodo y soldadura en una placa de cobre de manera que pueda ser utilizado sin problemas para cualquier tipo de motor. Se observa el módulo de puente H que se construyó en la siguiente imagen. (Fig. 9).

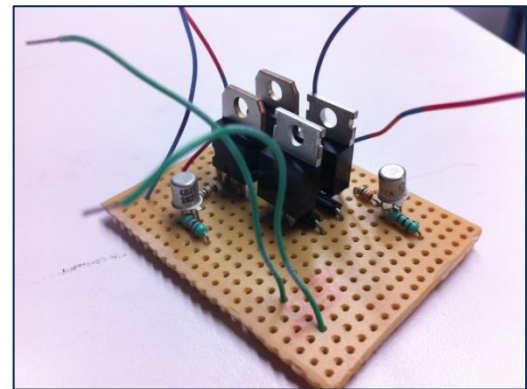


Fig. 9. Puente H montado en placa perforada

Cabe señalar que se utilizaron los transistores TIP32 y 2N2222 para el módulo de puente H construido en placa de cobre. Por ello se incluyó la información en el marco teórico de mayor relevancia, para conocer la arquitectura de este tipo de transistores. Además del módulo construido en placa perforada, se decidió utilizar el circuito integrado L293B para realizar la siguiente configuración (Fig. 10).

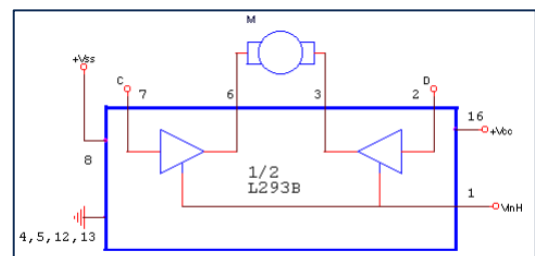


Fig. 10. Puente H utilizando circuito integrado L293B

Se debe dejar en claro que en la Fig. 7 se aprecia sólo el control de un motor, sin embargo, el circuito integrado empleado es capaz de manejar dos motores en ambos sentidos utilizando los pines de salida 3 y 4. Los pines de

control 2 y 7 son las entradas del circuito integrado, es decir, de donde se recibe la señal de salida que entrega el microcontrolador. Se sigue la siguiente tabla de verdad para controlar la dirección del motor.

V _{inh}	A	B	M
H	L	L	Parada rápida del motor
H	H	H	Parada rápida del motor
H	L	H	Giro a la izquierda
H	H	L	Giro a la derecha
L	X	X	Motor desconectado, giro libre

Fig. 11. Tabla de verdad para el control de dirección utilizando L293B

Es importante señalar que se utilizaron dos circuitos integrados L293B para el control de 4 motores DC del brazo robótico empleado. El circuito en placa se utilizó para controlar uno de los motores. En la siguiente imagen se muestra el módulo completo de puente H en conjunto con los demás módulos explicados en la presente sección de implementación.

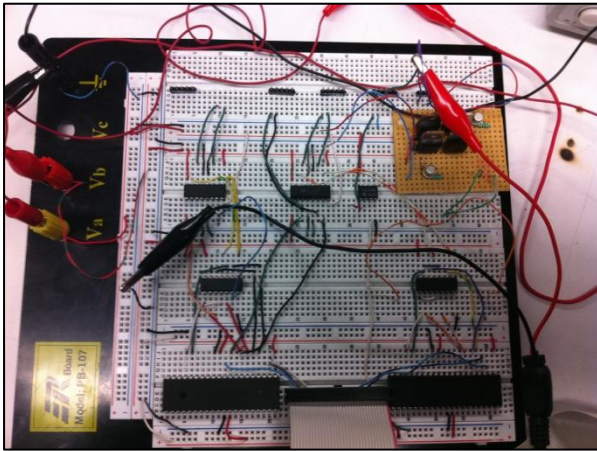


Fig. 12. Módulos del circuito de control

4. MÓDULO GRIPPER

El *gripper* funciona mediante un botón el cual activa el funcionamiento de éste. En nuestro modelo estamos usando lo que se conoce como “pull-up resistor”, el cual consiste en una resistencia que nos evita que se junten Tierra con Vcc. Nuestro puerto del PIC en el que checamos el estado del botón es el E2, y siempre que no está presionado el botón está en alto (5 V). Al presionar el botón este pasa a conectarse a Tierra, provocando que nuestro código ejecute lo debido. Se muestra un diagrama general de su conexión en la figura 13.

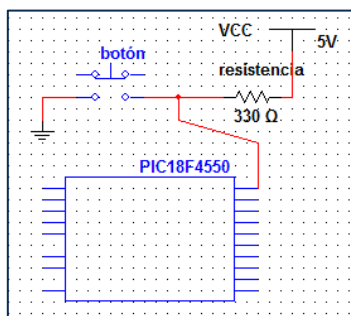


Fig. 13. Conexión del *gripper*

4.1 Programación del módulo *gripper*.

Para el código de nuestro *gripper*, hay que primero inicializar algunos parámetros:

- “int salida_*gripper* = 1” Declaramos una variable de tipo entero con la cual nos ayudaremos a saber si nuestro *gripper* está abierto (1) o está cerrado (0).
- “TRISE=0b00000100” Seguidamente declaramos los puertos E0 y E1 como nuestras salidas hacia los motores, el puerto E2 como la entrada de nuestro botón, y finalmente el resto de puertos E los declaramos como salidas por prevención.

Nuestra función de código a ejecutar para el *hfyu* será llamado “*gripper()*”, y estará llamándose desde nuestro método main en un loop infinito, para siempre estar al pendiente de cuando se presione nuestro botón.

El código a ejecutar de la función del *gripper* es el siguiente:

- “void *gripper*(void)” Se inicializa la función del *gripper*.
 - “if(PORTEbits.RE2==0)” Se revisa que el botón haya sido presionado. En caso de ser así, se continúa al resto de la función.
 - “while(PORTEbits.RE2==0) { }” Debido a que el reloj de nuestro PIC es demasiado rápido y que el PIC ejecuta las instrucciones de manera continua, es necesario poner esta instrucción con el motivo que se detenga todo hasta que dejemos de presionar el botón, y evitar que se ejecute el código siguiente muchas veces.
 - “if(salida_*gripper* == 1)” Si la variable que declaramos para saber el estado del *gripper* nos indica que el *gripper* está abierto, continúa las ejecuciones de este bloque.
 - “LATEbits.LATE0=1” Esta instrucción hace salir 5 volts del puerto E0 de nuestro PIC, el cual pasa a una etapa de potencia haciendo que el motor de nuestro *gripper* lo haga cerrarse.
 - “Delay10TCYx (240)” Hacemos una pausa de aproximadamente 500 ms., con el motivo de que el tiempo de potencia del motor sea suficiente para cerrar completamente el *gripper*.
 - “LATEbits.LATE0=0” Le quitamos la potencia al motor cuando ya se ha cerrado completamente.

- “salida_gripper = 0” Para terminar este bloque declaramos la variable del estado del *gripper* como cerrado.
- “else if(salida_gripper == 0)” En el caso de que nuestro *gripper* esté cerrado, ejecuta este bloque.
 - “LATEbits.LATE1=1” Esta instrucción hace salir 5 volts del puerto E1 de nuestro PIC, el cual pasa a una etapa de potencia haciendo que el motor de nuestro *gripper* lo haga abrirse.
 - “Delay10TCYx (240)” Hacemos una pausa de aproximadamente 500 ms., con el motivo de que el tiempo de potencia del motor sea suficiente para abrir completamente el *gripper*.
 - “LATEbits.LATE1=0” Le quitamos la potencia al motor cuando ya se ha abierto completamente.
 - “salida_gripper = 1” Para terminar este bloque declaramos la variable del estado del *gripper* como abierto.

IV. CONCLUSIONES

El objetivo principal del proyecto fue el de lograr el control de un brazo robótico mediante el empleo de palancas de mando, por ello se dejó en claro que no era el de diseñar el mecanismo como tal, sino de controlar uno existente. De esta manera, se desarrolló un circuito de control basado en los principios de convertir señales analógicas a digital, modulación por ancho de pulso y control de dirección utilizando un puente H. Se obtuvieron los resultados deseados, pues se logró desarrollar un control basado en *joysticks* que pudiesen manipular cada uno de los grados de libertad con los que cuenta el brazo robótico empleado.

Se considera que el proyecto cuenta con un amplio potencial de continuidad, por lo que puede ser retomado en los semestres restantes de estudio. Lo anterior se justifica con el hecho de que existen una serie de mejoras que pudiesen ser añadidas para incrementar el grado de funcionalidad y eficacia del proyecto, como lo sería incluir rutinas automatizadas.

Los resultados experimentales obtenidos ayudaron a comprender la programación que fue llevada a cabo y la integración de la misma con los módulos de control que se incluyeron en la circuitería implementada en el proyecto. Gracias a la previa investigación realizada, los resultados de experimentación, esfuerzo, paciencia y asesoría, se resolvieron los problemas que se presentaron y el proyecto se considera que fue un resultado de aprendizaje exitoso.

V. RECONOCIMIENTOS

Los resultados obtenidos en el presente proyecto no hubieran sido posibles sin el constante apoyo de los maestros del Centro de Enseñanza Técnica y Superior. Agradecimientos especiales a la profesora Nataly Medina Rodríguez y al profesor Iván López Yee quienes impartieron las asignaturas de Electrónica Analógica e Instrumentación y Control, respectivamente.

VI. SOBRE LOS AUTORES

El proyecto fue desarrollado por estudiantes del 5to. Semestre de Ingeniería en Cibernética Electrónica del Centro de Enseñanza Técnica y Superior Campus Tijuana. Se pone a su disposición las direcciones de correo electrónico de cada uno.

Fernando Rios Limón
frioslimon@gmail.com

Rubén Guerra Marín
rguerra.marin@gmail.com

Vicente González Franco
vicentesonic_12@hotmail.com

Ildelfonso Quirino de Dios
iquirinods@hotmail.com

REFERENCIAS

- [1] Floyd, T., Electronic Devices, Seventh Ed., Estados Unidos: Pearson, 2005
- [2] H-Bridges: Theory and Practice, Chuck Mc Manis, disponible en <http://www.mcmanis.com/chuck/robotics/tutorial/h-bridge/index.html>
- [3] Datasheet de transistor 2N2222
- [4] Datasheet de transistor TIP 31, 32.
- [5] Herrera, J., Teoría PWM, México: CIDETEC IPN, 2008, disponible en http://www.cidetec.ipn.mx/profesores/jcrls/doctos/aplic_cidetec.pdf
- [6] Román, L., Modulador por ancho de pulso, disponible en <http://members.fortunecity.es/electronico/circuitos/modulador/modulador.html>
- [7] Microchip, PIC18F2455/2550/4455/4550 datasheet, 2006.