5. SECCIÓN DEL MICROCONTROLADOR (PROGRAMACIÓN)

En este capítulo se describe la parte más robusta del proyecto, la cual es la programación del microcontrolador. Ésta se compone esencialmente de dos partes, la programación del controlador de los motores a pasos y la del controlador difuso. Primero se hace una pequeña descripción del dispositivo (PIC16F874) y posteriormente, de los periféricos empleados de éste. Después se explica como se implementó el sistema operativo, que es el encargado de administrar cada una de las partes del proyecto realizado; es decir, el que constituye la estructura del programa. Posteriormente se explican las rutinas por separado que conforman tanto los controladores mencionados, como pasos intermedios importantes.

5.1 Descripción del dispositivo

En la presente sección se da una breve descripción de los microcontroladores y de su estructura. Además se mencionan algunas de las características principales del microcontrolador PIC16F874 que es el utilizado en este proyecto.

Los microcontroladores son circuitos integrados que contienen muchas de las mismas cualidades que una computadora de escritorio, tales como la CPU, la memoria, etc., pero no incluye ningún dispositivo de "comunicación con seres humanos", como monitor, teclado o ratón. Esto es debido a que los microcontroladores son diseñados principalmente para aplicación de control de máquinas, más que para interactuar con humanos.

Algunas de las características básicas de los microcontroladores que los diferencian de los microprocesadores pueden apreciarse en la tabla 5.1-1.

MICROCONTROLADOR	**MICROPROCESADOR**
Juego de instrucciones reducidas	Juego de instrucciones extensas
Periféricos incluidos	Necesita periféricos externos
Es para aplicaciones de control	Tiene aplicaciones de propósito general
Es de bajo costo	Es caro

Tabla 5.1-1
Diferencias entre un microcontrolador y un microprocesador

Los microcontroladores se encuentran divididos por la longitud de sus palabras de instrucción en tres familias mencionadas a continuación.

1. Base-Line: Tiene una longitud de palabra de 12 bits.
2. Mid-Range: Tiene una longitud de palabra de 14 bits.
3. High-End: Tiene una longitud de palabra de 16 bits.

Estructura del dispositivo.

Cada parte del dispositivo puede clasificarse en uno de los siguientes tres grupos:

1. Núcleo

El núcleo pertenece a las características básicas, que son requeridas, para que el dispositivo funcione. Incluye dispositivos como la CPU, la ALU, el juego de instrucciones, etc.

2. Periféricos

Los periféricos son los dispositivos que los diferencian de los microprocesadores. Estos están para interfase con el mundo exterior (como los pines de I/O de propósito general, controladores de LCD, entradas A/D y salidas PWM), y para tareas internas como el conservar diferentes bases de tiempo (temporizadores).

3. Funciones o características especiales

Son todos aquellos detalles que ayudan a que el microcontrolador pueda ser más barato, rentable y flexible. La familia de rango medio ofrece varias de estas características adicionales, como el temporizador de perro guardián, modo de bajo consumo (dormido), osciladores RC interno, programación serial ya soldado, etc.

Características principales del microcontrolador PIC16F874

Este microcontrolador presenta las siguientes características principales:

Memoria de programa: 7168 bytes y 4096 palabras de instrucción

RAM de datos: 192

Velocidad máxima: 20MHz

Puertos I/O: 33

Temporizadores: 3 y el de perro guardián

Opción de consumo de bajo voltaje

Memoria EEPROM de datos de 128 bytes

5.1.1 Interrupciones

Las interrupciones son una característica muy importante que tienen los microcontroladores y permiten su gran versatilidad en la administración de diversas tareas. Éstas, posibilitan que el microcontrolador suspenda (si así es configurado) una tarea que se encuentra ejecutando en un determinado momento para atender alguna otra tarea que es avisada mediante la interrupción de algún periférico. Cuando una interrupción ocurre el microcontrolador salta el vector de atención de interrupciones, en donde corresponde al usuario asignar mediante programación la prioridad de cada interrupción con la que este trabajando y realizar la rutina de servicio de interrupciones (ISR).

Los periféricos de los microcontroladores poseen cada uno su respectiva interrupción, aunque algunos de ellos pueden tener más de una. Las interrupciones pueden ser habilitadas o deshabilitadas; ya sea de manera individual (enmascarando cada una) o de manera general. La deshabilitación general puede hacerse de los periféricos, dejando únicamente habilitadas las interrupciones del Timer0, interrupción en cambio y la interrupción externa; o deshabilitando todas las interrupciones. Todos los enmascaramientos (particulares o generales) se realizan por software.

5.2 Periféricos empleados

A continuación se describirán los periféricos empleados en el desarrollo de este proyecto. La descripción será de manera muy general, explicando únicamente su funcionamiento, para ver detalles de configuración y más específicos, referirse al manual [4].

5.2.1 Timer0

El módulo del Timer0 tiene las siguientes características:

- Temporizador/contador de 8 bits
- Permite lectura y escritura
- Preescalador de 8 bits programable por software
- Fuente de reloj seleccionada para ser interna o externa
- Interrupción en el desbordamiento de FFh a 00h
- Selección de flanco para reloj externo

En general, el Timer0 es un contador o temporizador, de acuerdo al modo que se seleccione. Cuenta un evento externo en el pin T0CKI; en modo de temporizador, cuenta desde el número colocado en su registro con un periodo variable. En ambos casos cuando ocurre su desbordamiento (FFh-00h) presenta una interrupción. Su fuente de reloj puede seleccionarse entre diversas opciones, tanto internas como externas. Siempre está funcionando, por lo que debe cuidarse deshabilitar su interrupción cuando no es deseada.

5.2.2 Timer1

El Timer1 es un contador/temporizador de 16 bits, formados por dos registros de 8 bits que permiten lectura y escritura. En conjunto, los registros cuentan desde 0000h hasta FFFFh y vuelve a 0000h. La interrupción del Timer1 si está habilitada, ocurre en el desbordamiento de FFFFh a 0000h. Este módulo tiene tres modos de operación:

- Temporizador síncrono
- Contador síncrono
- Contador asíncrono

En modo contador el Timer1 se incrementa en cada flanco de subida del reloj en el pin T1CKI; en modo temporizador, en cada ciclo de instrucción. Este temporizador si puede ser apagado, mediante software; cuenta además con la opción de trabajar con un cristal externo.

5.2.3 Convertidor análogo/digital (A/D)

El convertidor A/D es un módulo que permite la conversión de una señal analógica de entrada, al número digital de 8 bits equivalente. Cuenta con 8 entradas analógicas o canales, que son seleccionadas por software. La salida de un capacitor de muestreo y retención es la entrada al convertidor, que obtiene el resultado por aproximaciones sucesivas. El voltaje de referencia se selecciona por software entre un valor externo (pin V_{REF}) y el voltaje de alimentación (V_{DD}). Realmente la resolución del convertidor es de 10 bits, pero el resultado se muestra en un registro de 8 bits, por lo que tiene que elegirse si se justifica a la derecha o a la izquierda. Es importante permitir que el capacitor de muestreo y retención se cargue, para que la conversión resulte en el valor que está en el pin y no en uno

erróneo, para ello hay que dejar transcurrir un pequeño tiempo de carga. Para empezar la conversión se activa por software; al terminar ésta, el módulo produce una interrupción.

5.2.4 Puertos

Los pines de entrada/salida (I/O) de propósito general son los periféricos más simples. Permiten al microcontrolador monitorear y controlar otros dispositivos. Para agregar flexibilidad y funcionalidad, algunos pines están multiplexados con funciones alternas. Estas funciones dependen de los periféricos con los que cuenta el microcontrolador. Cuando un periférico está funcionando, la función de pin de I/O de propósito general no se puede utilizar.

La mayoría de los puertos son bidireccionales; esto es que puede ser programado como entrada o salida. Esta elección depende del registro conocido como TRIS, en el cual, cada bit tiene correspondencia con cada pin del puerto. Un 1 en el bit del pin correspondiente, lo habilita como entrada y un 0 como salida.

En general un puerto en modo de entrada, al leerlo, nos da en su registro los valores lógicos que corresponden a los niveles TTL que se encuentran físicamente en los pines. Un puerto de salida, al escribirlo, transmite el correspondiente valor del registro a niveles TTL en los pines.

5.2.5 Interrupción en cambio

Esta característica permite generar una interrupción, cuando ocurre un cambio en el nivel de los pines 4 al 7 del puerto B. Está característica es sólo aplicable cuando estos

pines se encuentran configurados como entradas. El valor actual en estos pines (RB4:RB7) es comparado con el de la última lectura del puerto y si se detecta una condición de diferencia en cualquiera de ellos, se genera la interrupción de este periférico si está habilitada.

5.3 Estructura del programa

En esta sección se da un panorama global del funcionamiento del programa así como de la interacción de sus partes. Podemos tener dos grandes grupos, las subrutinas que se ejecutan dentro del sistema operativo y las que se ejecutan en la rutina de servicio a interrupciones.

Partiendo de un estado de espera en el sistema operativo, la serie de eventos que se suceden en la secuencia del programa son los siguientes:

1. Para salir de su estado de espera el sistema recibe la interrupción en cambio producto de un botonazo por parte del usuario.
2. Se ejecuta la rutina de adquisición de datos que tiene como entradas los valores analógicos de la interfaz con el usuario y da como salida los valores digitales correspondientes a estos.
3. Se ejecuta la rutina del control difuso que toma los valores anteriores como entradas y da como salidas los resultados de la aplicación del controlador.
4. Se ejecuta una secuencia de adaptación de datos dentro del sistema operativo, que toma las salidas del controlador y los adapta al número de pasos que se moverán los motores.

5. Se ejecuta la rutina que controla el movimiento de los motores, para que el dispositivo apunte en la altura adecuada sobre la bola y regular la fuerza del tiro; tomando para ello los valores ajustados en el paso anterior.
6. Se ejecuta el tiro, el microcontrolador como salida a esta secuencia manda un impulso al exterior que activa el sistema de disparo del pistón (sección 6.1).
7. Se da un retraso de un segundo. Se quita el impulso.
8. Se ejecuta una vez más la rutina que controla el movimiento de los motores, esta vez con la acción inversa a la descrita en el paso 5, es decir, reestablece altura y fuerza a su posición de origen (centrado).
9. Regresa a su posición de espera para ejecutar la rutina nuevamente.

Las rutinas que se ejecutan en el sistema operativo son las del control difuso, la de adaptación de datos y la de ejecución del tiro. Las demás son parte de la rutina de ISR ya que maneja interrupciones de los periféricos.

Como parte final de esta sección y para facilitar la apreciación global del funcionamiento del sistema operativo, se muestra en la figura 5.3-1 un diagrama a bloques de él.

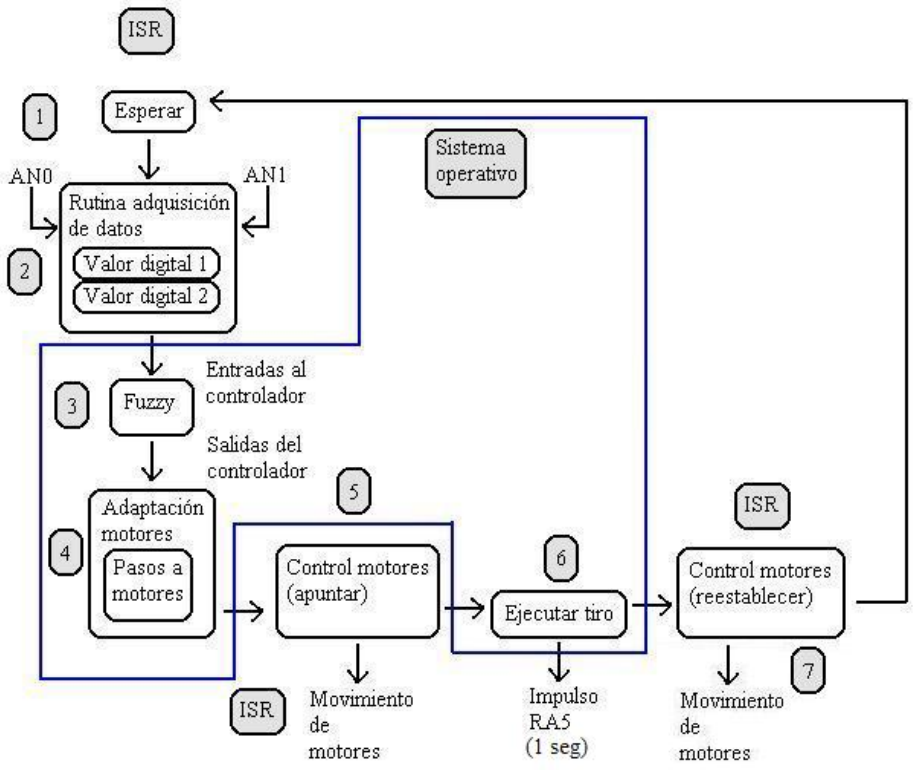

Figura 5.3-1
Funcionamiento del programa

5.4 Adquisición de datos de interfaz externa

El siguiente apartado describe el proceso mediante el cual se convierten los dos valores analógicos provenientes de la interfaz del usuario (entradas del controlador difuso) en sus equivalentes números digitales. El voltaje de referencia es el voltaje de la fuente V_{DD} (5V). Dichos valores se encuentran en el pin AN0 Y AN1 del microcontrolador. Los valores de salida se entregan en los registros: VALOR_PRIMER_POT y VALOR_SEGUNDO_POT.

La secuencia de la rutina es la siguiente:

1. El programa recibe una interrupción de cambio. En la ISR cambia el canal de adquisición del convertidor A/D al primer canal (AN0). Como se mencionó, tiene

que esperar un tiempo de adquisición. Para ello se programa el Timer0 con el tiempo de adquisición mínimo cuyos cálculos se muestran al final de este apartado (fórmula 4). Regresa al programa principal.

2. Recibe la interrupción del Timer0 que indica que ya terminó el tiempo de adquisición. En la ISR activa la conversión y regresa al programa principal.
3. Recibe la interrupción de que la conversión esta lista. En la ISR lee el valor de la primera conversión y lo guarda en el registro VALOR_PRIMER_POT. Seguido a esto, cambia el canal de adquisición al segundo canal (AN1) y configura nuevamente el Timer0 para dar el tiempo necesario de adquisición. Regresa al programa principal.
4. Se repiten los pasos 2 y 3, salvo que el resultado es guardado en el registro VALOR_SEGUNDO_POT.

El diagrama a bloques de la adquisición de datos externos se muestra en la figura 5.4-1.

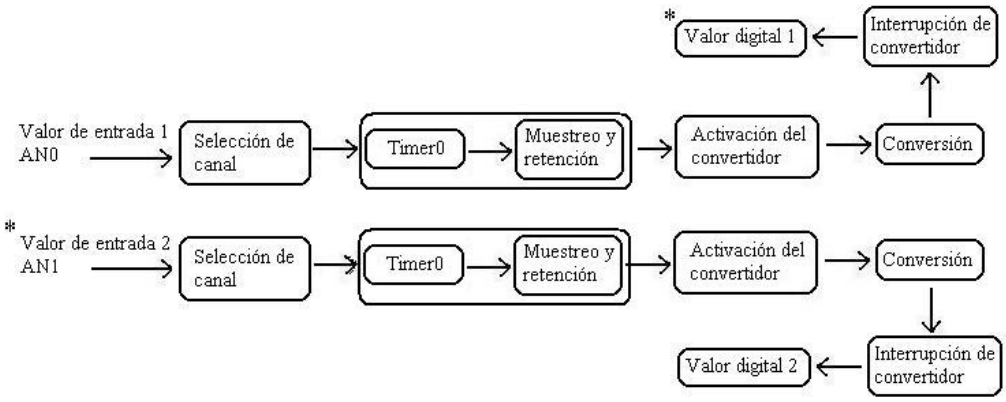

Figura 5.4-1
Adquisición de datos externos

Para el cálculo del tiempo de adquisición, se utiliza la fórmula (4).

$$T_C = -(51.2\ pF)(1\ k\Omega + R_{SS} + R_S)\ \ln(1/511)$$

$$T_{ACQ} = T_{AMP} + T_C + T_{COFF}$$

$$T_{COFF} = [(Temp - 25°C)(0.05\ \mu s/°C)]$$

(4) Fórmulas para el cálculo del tiempo de adquisición

$R_{SS} = 7k\Omega$

$R_S = 10k\Omega$

$T_{AMP} = 5\mu s$

Temp = 50ºC

Lo cual nos da un tiempo de carga del capacitor de muestreo y retención de 5.724µs y un tiempo de adquisición de 11.974µs; para fines de dar un margen se utilizó arbitrariamente un tiempo de 25 µs.

5.5 Implementación del controlador difuso

La presente sección describe la parte más ardua de programación del proyecto. Existen varias razones; por ejemplo, que presenta una gran cantidad de accesos a subrutinas, lo que conlleva accesos a pila que deben ser manejados con la debida atención para evitar saltos erróneos que provoquen que el programa se pierda. También está por supuesto la implementación de cada una de las subrutinas; tanto auxiliares (multiplicación, división, etc.) como aquellas que forman la estructura misma del controlador difuso descritas en el capítulo 4. En general la rutina del control difuso toma los valores leídos en

la rutina de adquisición de datos (sección 5.4) como las entradas al controlador difuso y devuelve los valores de salida en RES_FUERZA y RES_ALTURA. La rutina del control difuso, se ejecuta de acuerdo a la secuencia presentada a continuación.

5.5.1 Programación del fuzificador

Para llevar a cabo la fuzificación, en primer lugar se llama a la rutina que calcula las membresías de la entrada uno. Ésta sigue la siguiente secuencia:

1. Lee el valor del registro VALOR_PRIMER_POT que toma como entrada.
2. El siguiente paso es revisar en que cluster se encuentra este valor, una vez localizado indica esto en el registro CLUSTERS_IN1 activando los bits equivalentes del cluster al que pertenece.
3. Basado en los límites de la figura 5.5.1-1, maneja el segmento de la sección donde pertenece el valor para mandar llamar la subrutina que calcula la membresía.
4. La rutina calcular membresía mediante subrutinas de multiplicación y división, calcula la membresía sobre la parte de pendiente negativa. Para ello utiliza las fórmulas (1) y devuelve el resultado en el registro MEMBRESÍA.
5. Por la simetría del modelo, la membresía calculada (la parte de pendiente negativa), debe sumar 1 con la del otro cluster al que pertenece el valor. Por ello ésta última se calcula mediante una resta, para ahorrar tiempo de cálculo. Los valores de membresía del cluster x es guardado en el registro MEM_IN1_Cx.

Esta misma secuencia es hecha para el segundo espacio sólo que el valor se toma del registro VALOR_SEGUNDO_POT y los valores de membresía resultantes son guardados en los registros MEM_IN2_Cx (donde x va de 1 hasta 5).

La subrutina de calcular membresías tiene la capacidad de calcular membresías sobre una pendiente negativa o positiva, indicado esto en el bit PENDIENTE_POS del registro FUZZY_REG, si es cero indica que es pendiente negativa y si es uno indica que es pendiente positiva. La membresía se calcula entre P1 y P2 (ver figura 5.5.1-2). La subrutina nos arroja un resultado manejando precisión de centésimas.

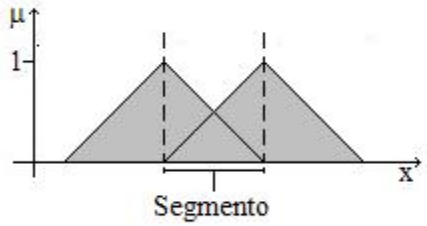

Figura 5.5.1-1
Límites de clusters manejados por el programa

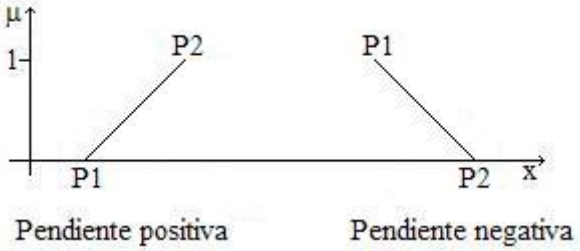

Figura 5.5.1-2
Definición de pendientes y puntos

5.5.2 Programación de la inferencia

La subrutina de la inferencia se hace en base a comparaciones, lo que equivale a ver que reglas se están cumpliendo. La inferencia del controlador de altura es más simple que la de fuerza, dado que sólo hay cinco reglas y sólo dependen de un antecedente (un espacio de

entrada) se tienen únicamente cinco comparaciones directas posibles. Para el caso de la segunda inferencia se tienen más comparaciones y éstas se encuentran anidadas debido a que son reglas compuestas (dependen de más de un antecedente) por lo cual se sigue el siguiente procedimiento.

1. Cuando las reglas formadas tienen dos condiciones, se juzga la primera de ellas (correspondiente a un espacio de entrada) y se toma el valor de membresía que corresponde al cluster de este espacio, como el que se hereda al cluster de salida correspondiente.
2. Se compara cual de las segundas condiciones (correspondiente al otro espacio de entrada) dependientes de la primera se cumple.
3. Se juzga si la membresía obtenida en el paso 2 es menor a la del paso 1 mediante una subrutina. Es decir, después de esta subrutina el valor heredado al cluster de salida en esta regla es el mínimo.
4. Para cumplir con que el valor final debe ser el máximo de estos mínimos, se lleva a cabo una subrutina que juzga si este valor de membresía es mayor que cualquier otro del mismo cluster hasta el momento. De ser así este valor se deja como el mayor actual; así se asegura que el valor con el que terminará el cluster siempre será el mayor.

De esta manera se termina con el método de inferencia máximo – mínimo. En la secuencia del programa se ejecuta primero la inferencia de la altura, seguida de su defuzificación (que se explica a continuación) y posteriormente la inferencia de la fuerza seguida de su defuzificación. Esto para hacer notar que no se ejecutan las dos inferencias de manera

consecutiva ya que se utilizan registros comunes en las dos inferencias y pueden perderse valores en la defuzificación.

5.5.3 Programación del defuzificador

La rutina es la misma para ambos controladores. La defuzificación se lleva a cabo segmento por segmento del espacio de salida. En cada segmento realiza lo siguiente:

1. Calcula la membresía del lado de pendiente positiva y la compara con la heredada y toma la menor que es la que rige el cluster.
2. Realiza la misma operación que en el paso 1, sólo que con la pendiente negativa.
3. Compara cual de las membresías es mayor y es la que agrega a la sumatoria del método del centroide en la parte del denominador.
4. Multiplica el mismo valor que agrego en el paso anterior por el valor del punto evaluado y lo agrega al numerador del método del centroide.
5. Repite los pasos 1, 2, 3 y 4 para cada punto de cada segmento del espacio de salida.
6. Realiza la división de la fórmula del método del centroide.

Es así como se obtiene el valor de salida del controlador difuso y se procede a adaptarlo al movimiento de los motores.

5.6 Adaptación de las salidas del controlador difuso, a pasos de los motores

El resultado del controlador difuso nos da un valor dentro de un espacio continuo de 0 a 252. Como tanto el motor de la válvula como el de la altura están centrados, mediante simples restas se adapta el número de pasos que se moverá y la dirección en la que lo hará.

Por ejemplo, si el resultado del controlador es mayor al centro del espacio, el motor se moverá la diferencia de esos números en dirección, supongamos derecha. Si es menor se moverá la diferencia de estos pasos en dirección, de acuerdo a la suposición anterior, a la izquierda. Estos valores se guardan en los registros PASOS_MOTOR_1 y PASOS_MOTOR_2 y se indica el sentido conveniente. Por conveniente entendemos hacía que lado gira el motor, de acuerdo si queremos que la válvula abra o cierre o si el pistón suba o baje. En el siguiente apartado (5.7) se explica el concepto de movimiento de los motores.

5.7 Controlador del motor a pasos

En esta sección se describe la parte del software que manda la secuencia de pasos necesarios para mover los motores a pasos. Como se mencionó en la sección 3.2, este programa mueve dos motores. El programa recibe el número de pasos que se moverá cada motor. Para contabilizar estos movimientos, se tienen dos registros de 8 bits llamados: CONT_PASOS_L y CONT_PASOS_H, donde el primero se maneja como una preescala. La resolución total es de 16 bits; es decir, el motor puede moverse hasta 65536 pasos consecutivamente, con manejo interno del programa. La secuencia de pasos que se maneja como salida del programa se manda a través de un puerto. La secuencia del primer motor es mandada por los cuatro bits menos significativos del puerto B y la segunda secuencia por los mismos bits del puerto D. Tiene además la capacidad de mandar la secuencia en orden invertido, por lo cual puede mover el motor en ambas direcciones.

Los movimientos del primer motor, son realizados de 11 en 11 pasos (preescala de 11), el por qué se explica más adelante en la integración del proyecto (dwiefniwf). Así que si el número de movimientos que recibe es *n*, el motor se mueve *n* series de once pasos; esto ya

está considerado al recibir el valor (adaptación). Los pasos del segundo motor, son enviados directamente los indicados. Así que para el primer motor, el número de pasos que se recibe es colocado en el CONT_PASOS_H y el CONT_PASOS_L mantiene la preescala de 11; para el segundo, el número de pasos es colocado en el CONT_PASOS_L y el CONT_PASOS_H tiene el valor de uno.

La secuencia de pasos que sigue el programa se describe a continuación:

1. El programa principal, manda a ejecutar esta secuencia de acuerdo a la estructura del sistema operativo (sección 5.3) y le indica que se trata del motor 1.
2. El programa evalúa en que dirección debe moverse dicho motor de acuerdo al bit SENTIDO_1 del registro BETA y lo indica en el bit SENTIDO del registro ALFA; salta a la secuencia de la dirección correspondiente.
3. El programa determina que es el primer paso mandado. Configura el Timer0 para dar el tiempo de espera entre paso y paso que hará que el motor se mueva de manera uniforme y a una velocidad suficientemente rápida. Manda el primer paso y regresa al programa principal.
4. Cuando recibe la interrupción del Timer0 que indica que el tiempo entre paso y paso se ha cumplido, en la ISR reconoce en que secuencia está (motor y dirección) y que ya no es el primer paso, entonces manda el paso siguiente.
5. Acto seguido, resta 1 al registro CONT_PASOS_L; si no es cero el resultado, regresa al programa principal a esperar la siguiente interrupción del Timer0 y mandar el paso siguiente. Repite los pasos 4 y 5 hasta que al restar se obtiene como resultado cero, entonces procede con el siguiente paso.

6. Resta 1 al registro CONT_PASOS_H; si no es cero, vuelve a cargar el CONT_PASOS_L con el valor de 0Bh y regresa a los pasos 4 y 5. Si el resultado de la resta es cero, entonces sabe que terminó la secuencia de pasos del primer motor y continúa al paso 7.
7. Activa el indicador para comenzar con los pasos del segundo motor, mueve un 1 al CONT_PASOS_H y el número de pasos a mover el motor al CONT_PASOS_L y regresa al sistema operativo.
8. En el sistema operativo se determina que sigue con la secuencia de motores y conoce que se trata del motor 2, se determina su dirección por el bit SENTIDO_2 del registro BETA y se guarda en el bit SENTIDO de ALFA. Repite los pasos 3, 4, 5 y 6 (el paso 6 sólo se ejecuta una vez ya que no hay preescala) y en está ocasión, continúa en vez del paso 7, con el 9.
9. Activa el bit TIRO del registro BETA, para determinar que la rutina de motores ha terminado y proceda a ejecutar el tiro.

El bit SENTIDO del registro ALFA indica, durante la ejecución de la rutina de movimiento, hacia que dirección va; mientras que SENTIDO_1 y SENTIDO_2 del registro BETA, dan al iniciar el movimiento de cada motor esta dirección. Siendo uno en SENTIDO_1 cerrar y cero abrir y en SENTIDO_2 cero indica subir y uno bajar. Cabe mencionar que siempre memoriza en que posición quedaron los motores, para que cuando vayan a realizar otro movimiento, se mande la secuencia a partir de este valor. El diagrama a bloques de esta parte del programa es mostrado en la figura 5.7-1.

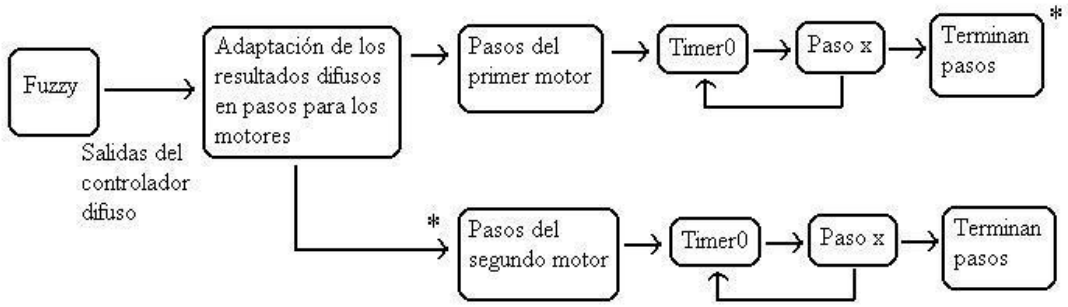

Figura 5.7-1
Controlador del motor a pasos

5.8 Simulación en MPLAB©

El programa implementado en este proyecto se simuló en el MPLAB©. Esto nos permitió, tanto encontrar errores de programación, como comprobar los resultados del controlador difuso implementado, para que al integrar el microcontrolador al circuito, tener la certeza de su correcto funcionamiento. En la sección 7.1 se mostrará la comparación de los resultados de las simulaciones de MatLab© y MPLAB©.

De acuerdo a los procedimientos presentados en este capítulo, queda implementado el programa que constituye el control total del mecanismo, desde los procedimientos mecánicos hasta el controlador difuso. En este capítulo se explica de una manera estructurada la serie de pasos que ejecuta este programa, tratándolos a detalle. Sin embargo, la descripción paso por paso, así como de todas y cada una de las subrutinas incluidas, se lleva a cabo en la documentación del programa. Éste se adjunta en un CD-ROM que conforma el apéndice. Queda únicamente por delante el tratar la interfaz de usuario, que se describe en el capítulo siguiente.