



**SEP**

**DGIT**

**SEIT**

---

---

**INSTITUTO TECNOLÓGICO DE PUEBLA**

**CONTROL POR VELOCIDAD PARA UN  
ROBOT DE DOS GRADOS DE LIBERTAD**

MEMORIA DE RESIDENCIA PROFESIONAL

**QUE PARA OBTENER EL TÍTULO DE:**

**INGENIERO ELECTRÓNICO**

**P R E S E N T A :**

**MIGUEL ANGEL PÉREZ XOCHICALE**

AUTORIZADO PARA INICIAR TRAMITES DE TITULACIÓN

---

---

**M.C. ESTEBAN TORRES LEON.**

**PUEBLA, PUE.            MAYO 2003.**

# Í N D I C E

	Página.
Índice.....	i
Índice de figuras.....	vi
Índice de Tablas.....	ix
Introducción.....	x
Justificación.....	xii
Objetivos Generales.....	xiii
Objetivos Particulares.....	xiii
Alcances.....	xiv
Limitaciones.....	xiv
 <b>Capítulo I. Introducción al sistema</b>	
1.1 Fundamentos de robótica.....	1
1.1.1 Robot.....	1
1.1.2 Historia del robot.....	2
1.1.3 Clasificación de los robots.....	4
1.1.4 Impacto de la robótica.....	7
1.2 Definición de control.....	7
1.2.1 Tipos de control.....	8
1.2.2 Control por velocidad con PWM.....	9
1.3 Sistema operativo.....	11
1.4 Definición de programar.....	13
1.4.1 Programación de Lenguaje grafico LabVIEW.....	13
1.4.2 Programación de los robots.....	14
1.5 Comunicación serie.....	15
1.5.1 El puerto serie.....	15
1.5.2 El estándar RS-232.....	15
1.5.3 Especificaciones del puerto serie RS-232.....	16
1.5.4 El UART.....	18
1.6 Estructura del robot del proyecto.....	22
1.6.1 El puente H 7125.....	22
1.7 Sensorización.....	24
 <b>Capítulo II. LabVIEW</b>	
2.1 Introducción.....	25
2.2 LabVIEW.....	26
2.2.1 Por que utilizar LabVIEW.....	26
2.2.2 Aplicaciones de LabVIEW.....	27
2.3 Ambiente LabVIEW.....	27
2.3.1 Programación gráfica con Labview.....	28



3.7 Modulación de anchura de pulsos.....	77
3.7.1 Determinación de frecuencia y configuración de PWM, para el microcontrolador.....	81
3.8 Comunicación serie asincrona.....	84
3.8.1 Generador de Baudios.....	84
3.8.2 Transmisión asíncrona.....	86
3.8.3 Recepción asíncrona.....	86
3.9 MPLAB.....	87
3.9.1 Herramientas de MPLAB.....	88
<b>Capítulo IV. Desarrollo del sistema</b>	
4.1 Introducción.....	89
4.2 Control por velocidad en LabVIEW.....	89
4.2.1 Diseño de los Sub VI para el programa de control por velocidad.	89
4.2.1.1 Inicialización de rutinas.....	90
4.2.1.2 Selector de 10bits en dos cadenas para configurar el ciclo útil del PWM.....	91
4.2.1.3 Selector de 10 bits para eje de Y.....	92
4.2.1.4 Indicadores de Switches límite para los dos ejes.....	93
4.2.1.5 Indicadores de Switches límite para cada uno de los ejes.....	95
4.2.1.6 Indicadores de funciones en 16F877.....	96
4.2.2 Implementación de los Sub VI para el programa de control por velocidad.....	97
4.3 Estructura del programa para el control por velocidad con PIC16F877....	106
4.3.1 Configuración del microcontrolador.....	106
4.3.2 Configuración de comunicación serie.....	106
4.3.3 Diagrama de flujo del programa.....	106
4.3.4 Rutina programa general.....	107
4.3.5 Rutina de comparación.....	108
4.3.6 Subrutinas de comparación.....	109
4.3.6.1 Rutina DATO_A.....	109
4.3.6.2 Rutina DATO_B.....	112
4.3.6.3 Rutina DATO_C.....	115
4.3.6.4 Rutina DATO_D.....	116
4.3.6.5 Rutina DATO_E.....	117
4.3.7 Conclusiones.....	118
4.3.8 Listado de programa.....	118
4.4 Diagramas de conexión del sistema.....	118

4.4.1 Diagrama para lograr la comunicación serie.....	118
4.4.2 Diagrama que simula los Switches límite.....	119
4.4.3 Diagrama de conexión con el puente H.....	119
4.4.4 Diagrama completo.....	120
<b>Capítulo V. Pruebas y resultados</b>	
5.1 Introducción.....	121
5.2 Comunicación entre dispositivos para el control por velocidad.....	122
5.2.1 Activación del control eje X.....	123
5.2.1.1 Indicadores de límites.....	125
5.2.2 Activación del control eje Y.....	126
5.2.2.1 Indicadores de límites.....	127
5.2.3 Activación de Home X – Y.....	127
5.2.4 Activación de Home X.....	128
5.2.5 Activación de Home Y.....	129
<b>Capítulo VI. Conclusiones y perspectivas</b>	
Conclusiones.....	131
Perspectivas.....	132
<b>Apéndices</b>	
<u>Apéndice A.</u> Hoja técnica del PIC16F87X.....	133
<u>Apéndice B.</u> Hoja técnica del MAX232.....	145
<u>Apéndice C.</u> Grabador de PICs.....	151
<u>Apéndice D.</u> Descripción de instrucciones para programar un PIC.....	158
<u>Apéndice E.</u> Listado de programa .....	170
<b><u>Referencias.</u></b>	180

## Índice de Figuras

	Página
Figura 1-1 Manipulador de tipo neumático MHU Junior.....	4
Figura 1-2. Programación gestual de un robot.....	5
Figura 1-3. Robot controlado por computadora.....	5
Figura 1-4. Señales con diferentes ciclos de trabajo.....	11
Figura 1-5. Transmisión de datos serie.....	22
Figura 1-6. Recepción de datos serie.....	22
Figura 1-7 Puente H 7125.....	23
Figura 2-1 Panel frontal de un VI.....	29
Figura 2-2 Diagrama de bloques de un VI.....	30
Figura 2-3 Diagrama de bloques para ejemplo para crear un sub VI.....	34
Figura 2-4 (a) Porción seleccionada en el diagrama de bloques, con el cual se creara un nuevo Sub VI.....	34
Figura 2-4 (b) Seleccionando Edit >> Create Sub VI crearemos el sub VI de la porción seleccionada.....	34
Figura 2-5 Configuración del puerto serie.....	35
Figura 2-6 Escritura por el puerto 0 con entrada de datos tipo cadena.....	36
Figura 2-7 Conversión de datos de entrada tipo cadena.....	36
Figura 2-8 Retardo de 10ms.....	37
Figura 2-9 Cerrando el puerto 0 ( COM1 ).....	37
Figura 2-10 Panel frontal para elegir dato a transmitir con su respectivo indicador.....	38
Figura 2-11 Tiempo de transmisión de datos para el PWM de 10bits.....	40
Figura 2-12 PWM configurado con 10 bits para un 50% de ciclo útil.....	41
Figura 2-13 Generación de 10 bits en dos cadenas una con 2bits y otra con 8 bits.....	41
Figura 2-14 Transmisión de 10 bits para el control del ciclo útil del PWM.....	42
Figura 2-15 VI para mandar 10bits.....	42
Figura 2-16 Lectura del dato cada 10ms en el puerto serie y comparación de dato.....	43
Figura 2-17 Activa PWM con los botones indicadores y displays.....	44
Figura 2-18 Indicadores y displays para activación y desactivación de PWM...	45
Figura 2-19 Activación de indicadores para los switches.....	46
Figura 2-20 Panel frontal de los indicadores de switches .....	47
Figura 2-21 Indicador que recibió 108d y lo desactiva si recibe 121d.....	48
Figura 2-22 Lee dato 108d y enciende indicador, si lee 121d se apaga el	48

indicador.....	
Figura 2-23 Secuencia de estructura para desactivar indicador Control eje X...	49
Figura 2-24 Secuencias de estructura cuando el valor de la resta es 1.....	49
Figura 2-25 Lee dato 108d y enciende indicador, si lee 121d se apaga el indicador.....	50
Figura 3-1 Arquitectura Von Newmann.....	53
Figura 3-2 Arquitectura Harvard.....	54
Figura 3-3 Diagrama de pines.....	57
Figura 3-4 Diagrama a bloques del PIC16F877.....	58
Figura 3-5 Organización de la memoria de programa tipo FLASH en el PIC16F877.....	59
Figura 3-6 Bancos y bits de direccionamiento.....	60
Figura 3-7 En la parte superior se muestra como se carga el PC. Abajo se muestra la carga del PC con las instrucciones CALL y GOTO.....	61
Figura 3-8 Formas de seleccionar el banco y la dirección de memoria RAM en los direccionamiento directo e indirecto.....	62
Figura 3-9 Distribución de la memoria RAM en cuatro bancos con 368 bytes útiles.....	63
Figura 3-10 Formato general de las instrucciones.....	65
Figura 3-11Conexion del oscilador.....	70
Figura 3-12 Direccionamiento directo con instrucción GOTO.....	73
Figura 3-13 Direccionamiento directo con instrucción CALL.....	74
Figura 3-14 Asignación de los bits de los registros CCPxCON para los módulos CCP1 y CCP2. ....	78
Figura 3-15 Salida de PWM.....	79
Figura 3-16 Estructura interna del módulo CCP1 en modo PWM.....	80
Figura 4-1 Diagrama de bloques activación de rutinas.....	90
Figura 4-2 Panel frontal de la activación de rutinas.....	91
Figura 4-3 Diagrama de bloques selector de 10bits.....	91
Figura 4-4 Panel frontal de selector de 10bits para el ciclo útil de PWM.....	92
Figura 4-5 Selector de 10bits para PWM eje Y.....	92
Figura 4-6 Panel frontal Selector de ciclo útil para PWM eje Y.....	93
Figura 4-7 Comparación de datos recibidos para activación de Switches límite	94
Figura 4-8 Panel frontal de datos para activación de Switches límite.....	94
Figura 4-9 Datos de switches X y Y.....	95
Figura 4-10 Panel frontal de datos de switches X y Y.....	95
Figura 4-11 Diagrama de bloque de indicador de funciones en 16F877.....	96
Figura 4-12 Panel frontal de indicador de funciones en 16F877.....	97
Figura 4-13 Botón DETENER TODO.....	97
Figura 4-14 Recepción de datos cada 10ms.....	98

Figura 4-15 Diagrama de bloques control de PWM e indicadores de switches	
límite eje X .....	98
Figura 4-16 Diagrama de bloques del control de eje X.....	99
Figura 4-17 Panel frontal de control del eje X.....	100
Figura 4-18 Control de PWM eje X y Y.....	100
Figura 4-19 Panel frontal de control de PWM eje X y Y.....	101
Figura 4-20 Diagrama de bloques de variables locales de indicadores de	
switches límite.....	101
Figura 4-21 Inicializadores de rutinas en el microcontrolador PIC16F877.....	102
Figura 4-22 Indicadores de funciones en el microcontrolador PIC16F877.....	103
Figura 4-23 Indicador de PIC en conexión.....	103
Figura 4-24 Diagrama de bloques de Control por velocidad.....	104
Figura 4-25 Panel frontal Control por velocidad.....	105
Figura 4-26 MAX232 Comunicación serie.....	118
Figura 4-27 Switches límite para los dos ejes.....	119
Figura 4-28 Conexión de puente H.....	119
Figura 4-29 Diagrama de conexión del sistema.....	120
 Figura 5-1 Inicialización del programa para control por velocidad.....	122
Figura 5-2 PIC EN CONEXIÓN.....	123
Figura 5-3 (a) Activación del control eje X .....	124
Figura 5-3 (b) Activación del control eje X .....	124
Figura 5-4 Límite superior X interrumpido.....	125
Figura 5-5 Activación del control eje Y.....	126
Figura 5-6 Límite superior Y interrumpido.....	127
Figura 5-7 Activación de HOME X-Y .....	128
Figura 5-8 Activación HOME X.....	129
Figura 5-9 Activación HOME Y .....	130

## Índice de Tablas

Tabla 3-1 Características del 16F877.....	57
Tabla 3-2 Descripción de campos de OPCODE.....	64
Tabla 3-3 Opciones de Directiva LIST.....	68

## ***Introducción***

En la actualidad, el campo de la electrónica tiene un amplio panorama de aplicaciones, una de las más importantes es la robótica.

En el Instituto Nacional de Astrofísica Óptica y Electrónica, en el área donde se desarrolló el proyecto existen diversos problemas a resolver relacionados con robots, que deben de realizar determinadas tareas, de este modo el control por velocidad para un robot de dos grados de libertad de la primera versión del proyecto, se cubrió parcialmente.

Con base a lo anterior y para realizar el control por velocidad para un robot de dos grados de libertad, se emigra al software a LabVIEW para el manejo y desarrollo del control, y un microcontrolador PIC 16F877 apto para suministrar las señales necesarias para el manejo del puente H.

El presente proyecto está conformado por los siguientes capítulos, dando una breve descripción a continuación

Capítulo I. Se presenta el marco teórico, tocando temas como: fundamentos de robótica historia, clasificación e impacto. Seguido por el control por velocidad con PWM, el sistema operativo, lenguajes de programación, comunicación serie, estructura del robot del proyecto y sensorización.

Capítulo II. Se revisa el lenguaje de programación G LabVIEW, además se define LabVIEW, el ambiente de programación en LabVIEW, la estructura de un instrumento virtual y

los módulos para la comunicación serie realizando transmisión y recepción de datos, y la estructura del programa para el control por velocidad.

Capítulo III. Se describe el microcontrolador PIC16F877, definiendo la estructura del microcontrolador, los comandos para la programación del microcontrolador, programación del PIC16F877, modulación por ancho de pulso, comunicación serie asíncrona y MPLAB como software ensamblador.

Capítulo IV. Se describe el desarrollo del sistema, diseñando SubVI's en LabVIEW para obtener los controles necesarios para el control por velocidad, así como la estructura del programa para el microcontrolador, describiendo con diagramas de flujo las rutinas de programación y finalmente la implementación de los diagramas electrónicos de conexión.

Capítulo V. Se mencionan las pruebas y resultados, a partir de evaluación del sistema en la comunicación serie entre LabVIEW y PIC16F877, control del ciclo útil de eje X y Y, control de límites e indicadores, control de home para eje X, control de home eje X y control de home eje Y.

Capítulo VI. Se mencionan las conclusiones y perspectivas, tomando en cuenta que dentro del capítulo V se evaluó el sistema completo, por esta razón se tiene un porcentaje de lo que se alcanzo dentro del proyecto presente.

## ***Justificación***

El presente proyecto, control por velocidad para un robot de dos grados de libertad, es necesario para la institución, ya que se deben de realizar pruebas de velocidad y posición.

En la primera versión se pretendió alcanzar el control por velocidad para el robot, pero debido problemas con la tarjeta de moviendo no se alzazo satisfactoriamente, de este modo se espera alcanzar el objetivo emigrando a otro software y hardware, superando la tecnología ocupada en la primera versión.

Dentro del área donde se realizó el proyecto, uno de los problemas más importantes, es el personal para el desarrollo de proyectos que muchas veces es esporádico por esto, la institución motiva al desarrollo del proyecto utilizando LabVIEW como software de manejo y un microcontrolador PIC como interfase para el puente H y los motores.

## ***Objetivos Generales***

Diseño, implementación, aplicación y evaluación, de un control para un robot de dos grados de libertad con LabVIEW y un microcontrolador PIC16F877, realizando por módulos cada uno de los controles aplicados al robot.

## ***Objetivos Particulares***

Realizar el control de velocidad aplicándole LabVIEW y programando cada una de las rutinas necesarias y requeridas por el microcontrolador PIC16F877

Realizar el programa para el control de velocidad con un microcontrolador PIC16F877, con sus respectivas rutinas necesarias para su objetivo.

Implementar la comunicación entre el programa de control en LabVIEW y el microcontrolador PIC16F877 para evaluar el sistema.

Evaluación del sistema completo, poniendo en prueba, los controles diseñados, tanto en LabVIEW, como en el microcontrolador, dando las respectivas conclusiones y perspectivas del proyecto.

## *Alcances*

Tener un sistema para el control por velocidad de un robot de dos grados de libertad superando la primera versión con una interfaz hombre maquina, para facilitar el control utilizando LabVIEW y un microcontrolador PIC16F877 para el desarrollo del presente proyecto.

Obtener respectivas pruebas y resultados en el menor tiempo posible con respecto a la primera versión para llegar a un punto donde las conclusiones del proyecto nos puedan dar una visión mas amplia de los objetivos planteados y posiblemente el surgimiento o implementación de nuevas ideas durante el desarrollo del proyecto.

## *Limitaciones*

En el área del proyecto se debe de tomar en cuenta que algunos de los proyectos desarrollados dentro de la institución, manejan información que es de tipo confidencial, lo que evita el desarrollo de algunos aspectos importantes en el proyecto para que se pueda analizar con la profundidad necesaria para un análisis de lo que se pretende desarrollar.

Otro factor importante el tiempo que no es lo suficiente para realizar un análisis detallado, debido a que mucha información, que se puede obtener, y dedicarse al análisis de cada parte es cuestión de tiempo, sabiendo que es necesario también tener conocimientos previos del funcionamiento general del equipo, así como la obtención de diagramas de conexión del robot, por otro lado en cuanto a la disposición del equipo se tienen fuentes de alimentación necesarias para cada uno de los proyectos que se realizan en esta área, y debido al desarrollo de nuevos proyectos es necesario una fuente de alimentación para estos, la fuente de alimentación

suministra un voltaje de 0 a 50V de corriente directa que son variables con una corriente de 15A, solo se cuenta con una fuente de alimentación para el desarrollo de este proyecto y otro mas.

En el caso de que alguien más pueda continuar el desarrollo de este proyecto es importante considerar las limitaciones anteriores que se tuvieron al realizarlo. En el desarrollo del proyecto y la elaboración es necesaria la información de teoría de control, conocimiento previo del manejo de LabVIEW, programación del leguaje para programar el microcontrolador. La información tratada es bastante extensa, por esta razón es tratada para ser lo mas breve posibles con el fin de que ningún detalle con el desarrollo del proyecto no sea citado.

# I.

## Introducción al sistema

### 1.1 Fundamentos de robótica

#### 1.1.1 Robot

Mucha gente tiene una idea errónea de lo que es un robot y esta definición puede variar de persona a persona, para aclarar la idea de lo que es un robot, consideraremos tres tipos de robots. Estos son Robots de Ciencia Ficción, Robots de juguete y los verdaderos robots.

El primero que inicia es con el Robot de Ciencia Ficción. La palabra robot fue usada por primera vez en un juego de ciencia ficción por Karel Čapek y escrita por Czech. El juego llamado “R.U.R. Rossums Universal Robots” esto se escribió en 1921.

En el lenguaje de Czech la palabra robota originalmente significa “Esclavo trabajador”. Esto da como resultado una confusión entre la gente y los robots y así se inicia la tradición del robot de ciencia ficción que interpretan papeles de malos en las historias en las cuales aparecen.

Por otro lado la palabra robotics de Isaac Asimov, quien uso por primera vez la palabra para describir la ciencia robot.

Asimismo quien es el padre de la ciencia ficción de robots, escribió la primera historia acerca de un robot en 1940, incluso definió las tres leyes para los robots de forma humanoide. Para el segundo tipo de robots que son los de juguete, de los cuales podemos encontrar muchos ejemplos los cuales son tomados de modelos de robots de ciencia ficción, muchos de los robots tienen la apariencia de humanoides.

El último de tipo de robots son identificados como robots reales, este término se puede subdividir en tres tipos

- a) Robots Industriales – que son la mayoría.
- b) Robots experimental o científicos – el segundo más popular tipo de robots verdaderos
- c) Robots de servicio – tienen una participación menor a las otras citadas.

### **1.1.2 Historia del robot**

Por siglos la gente ha construido mecanismos imitando las partes del cuerpo humano. Los egipcios construyeron brazos mecánicos para sus estatuas, los sacerdotes operaban los brazos quienes exigían se actuara bajo la inspiración de los dioses. Los griegos construyeron estatuas operadas hidráulicamente

En el siglo 18 en Europa, intrigados por el mecanismo de los títeres, se construyeron autómatas, con los cuales se accionaban representaciones convincentes de humanos y animales que impactaron al público. A mediados del siglo 20 los autómatas programables se usaron para entretenimiento, después de la mitad del siglo 20 diferentes factores contribuyeron al desarrollo de la robótica.

A continuación se da una breve cronología de los diferentes robots, de los más representativos, que se han creado a lo largo de la historia de la humanidad

### **Cronología**

Mediado siglo XVIII Muñecas mecánicas que ejecutaban piezas musicales (J. De Vaucanson).

1801 Telar programable para la urdimbre (J. Jacquard).

1805 Muñeca mecánica que hacía dibujos (H. Maillardet).

1946 Controlador que registraba señales eléctricas por medios magnéticos y las reproducía para accionar una máquina mecánica (G. D. Devol).

1951 teleoperadores para manejar materiales radiactivos (Goertz y Bergstrand).

- 1952 Máquina de control numérico (Instituto de Massachusetts).
- 1954 Robot (C. W. Kenward).
- 1954 Sistemas para transferencia de artículos programada.
- 1959 Robot comercial accionado con interruptores de fin de carrera y levas (Planet Corporation).
- 1960 Robot Unimate basado en transferencia de artículos programado (Devol).
- 1961 Robot Unimate instalado en Ford Motor Company para fundición en troquel.
- 1966 Robot para pintura por pulverización (Trallfa, firma noruega).
- 1968 Robot Shakey que se desplazaba por el suelo, utilizaba sensores, táctiles, cámara de visión (SRI- Instituto de Investigación Stanford).
- 1971 El brazo (stanford arm) era de accionamiento eléctrico.
- 1973 Primer lenguaje d robot: WAVE (SRI).
- 1974 Robot Irb6 de accionamiento eléctrico (ASEA).
- 1974 Robot para soldadura por arco para estructuras de motocicletas. (Kawasaki).
- 1974 Robot TTT con control por computadora.
- 1975 Robot para operaciones de montaje (Olivetti).
- 1976 RCC para la inserción de piezas en la línea de montaje (Labs. Charles Stark)
- 1978 Robot PUMA para tareas de montaje (Unimation).
- 1978 Robot TTT para operaciones de taladro, y circulación de materiales en componentes de aviones (Cincinnati Mialacron).
- 1979 Robot SCARA para montaje (Universidad de Yamanashi).
- 1980 Robot con visión por computadora.
- 1981 Robot de Impulsión directa(Universidad de Carnegie- Mellon).
- 1982 Robot RS-1 para montaje con su lenguaje el AML. (IBM).
- 1983Robot APAS sistema para montaje programable adapteble (Westinhause Corp)
- 1984 Robots programados fuera de línea.

Aunque mucho se habla de robots y se les da una interpretación de mecanismos en forma de humano, esto de alguna manera al estilo Hollywoodense, en la industria es otro el panorama. Los robots realmente son sistemas mecánicos pero que se utilizan para la producción. En otros casos para investigación y para entretenimiento.

### 1.1.3 Clasificación de los robots

La clasificación de los robots puede verse desde diferentes puntos de vista, ya sea por su espacio de trabajo, por su geometría, por su tecnología, etc. Con el desarrollo de controladores rápidos basados en microprocesadores y el empleo de servos en bucle cerrado en un robot, se estableció con exactitud la posición real de los elementos del robot y el error con la posición deseada. Esta evolución dio origen a una serie de tipos de robots, que se citan a continuación:

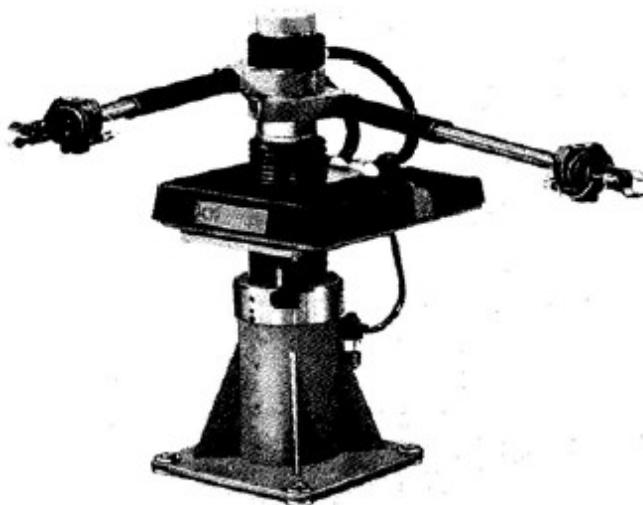
#### a).- Manipuladores

Son sistemas mecánicos multifuncionales, con un sencillo sistema de control que permite gobernar el movimiento de sus elementos, de los siguientes modos:

**Manual:** Cuando el operario controla directamente la tarea del manipulador.

**De secuencia fija:** Cuando se repite, de forma invariable, el proceso de trabajo prefijado.

**De secuencia variable:** Se pueden alterar algunas características de los ciclos de trabajo.



**Figura 1-1.** Manipulador de tipo neumático MHU Júnior 305

Comúnmente se utilizan a los manipuladores en operaciones básicas que pueden realizar óptimamente (funciones de trabajo sencillas y repetitivas), ver la figura 1-1.

## b)-. Robots de repetición o aprendizaje.

Son manipuladores que repiten una secuencia de movimientos, según se los enseñe el operario.

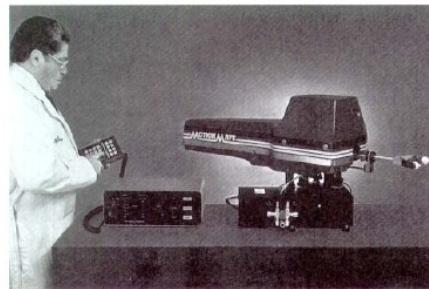
Para enseñar al robot se puede utilizar:

- Una pistola de programación
- Pulsadores o teclado
- Joysticks
- Un maniquí

Los robots de aprendizaje son los más conocidos, y el tipo de programación que incorporan, recibe el nombre de **gestual**, ver la figura 1-2.

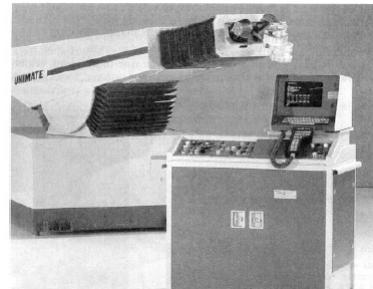
## c).- Robots con control por computadora.

Son manipuladores o sistemas mecánicos multifuncionales, controlados por una computadora. En la computadora se hará un programa con el cual se logrará mover al robot. El control por computadora dispone de un lenguaje específico o dedicado a robot, compuesto por instrucciones adaptadas al robot, con las que se puede confeccionar un programa de aplicación utilizando sólo la terminal de la computadora, no el brazo.



**Figura 1-2.** Programación gestual de un robot.

A este programa se le denomina **textual** y se crea fuera de línea (off-line), es decir, sin la intervención del manipulador. Ver la figura 1-3.



**Figura 1-3.** Robot controlado por computadora

Las ventajas que ofrece este tipo de robots le permite que se instale en el mercado rápidamente, lo que exige la preparación urgente de personal calificado, capaz de desarrollar programas similares a los de tipo informático.

#### **d).- Robots inteligentes.**

Son similares a los del grupo anterior además de que son capaces de relacionarse con el mundo que les rodea a través de sensores y tomar decisiones en tiempo real (auto programables). Actualmente son muy poco conocidos y se hallan en fase experimental. La visión artificial, el sonido de máquina y la Inteligencia Artificial, son las ciencias que más se están estudiando para su aplicación en los robots inteligentes.

#### **e).- Micro-robots**

Con fines educativos, de entretenimiento o investigación, existen numerosos robots de formación micro-robots a un precio muy económico y, cuya estructura y funcionamiento son similares a los de aplicación industrial,. Atendiendo a otros conceptos, hay diferente maneras de clasificar a los robots, pero inicialmente la que se ha expuesto es la que facilita mejor una primera aproximación al estudio de la Robótica.

La definición del “Robot industrial” y consecuentemente su clasificación, variará con el progreso, y lo que hoy se considera un robot avanzado, pasará a ser un recuerdo histórico, como ha sucedido con las computadoras.

#### **1.1.4 Impacto de la robótica**

La Robótica es una nueva tecnología, que surgió por los años 60's. En pocos años a desbordado cualquier previsión, dejando a un lado fantasías pero provocando impactos de diferente índoles: educación, automatización industrial, competitividad, sociolaboral.

##### **Impacto en la educación**

El auge de la Robótica y la imperiosa necesidad de su implantación en numerosas instalaciones industriales, requiere el concurso de un buen número de especialistas en la materia. La Robótica es una tecnología multidisciplinaria y hace uso de todos los recursos de vanguardia de otras ciencias afines, que soportan una parcela de su estructura, destacando las siguientes áreas:

- Mecánica

- Cinemática
- Dinámica
- Matemáticas
- Automática
- Electrónica
- Informática
- Enemiga y actuadores eléctricos, neumáticos e hidráulicos
- Sensores o traductores
- Visión artificial
- Sonido de maquina
- Inteligencia Artificial

La Robótica es una combinación de todas las disciplinas expuestas más el conocimiento de la aplicación a la que se enfoca, por lo que su estudio se hace en áreas de Ciencias e Ingeniería Superior. También su estudio se realiza en áreas de Informática en los vértices dedicados al Procesamiento de Imágenes, Inteligencia Artificial, Lenguajes de Robót y Programación de tareas, etc.

En la investigación brinda un basto y variado campo de trabajo, lleno de objetivos y en estado inicial de desarrollo.

A nivel educativo permiten a los centros de enseñanza complementar un estudio teórico de la Robótica, con las prácticas y ejercicios de experimentación e investigación adecuados.

## **1.2 Definición de Control**

Cuando necesitamos control es importante considerar que se requiere la consecuencia del resultado de una acción que siempre sea la misma, despreciando perturbaciones y disturbios que tengan efecto con la acción, el concepto de control tiene la siguiente característica:

Control es el proceso de que causa a un sistema variable permanecer lo mas cerca de un valor específico, el cual es llamado valor de referencia.

### 1.2.1 Tipos de control

En este caso es necesario más de un sistema variable que se refiere a un valor específico, teniendo como valores de referencia un sistema variable para ser controlado.

En controlador o sistema de control es el procesamiento de información del dispositivo con el cual manipula, flujo de energía, material u otro tipo de fuentes asociadas a sistemas físicos el cual es hecho con un sistema general de funciones en algún modo específico, enfrentando cambios arbitrarios en el sistema por el ambiente o del sistema físico del dispositivo.

La complejidad del control varía según los parámetros que se gobiernan, pudiendo existir las siguientes categorías:

**Controlador de posición:** Solo interviene en el control de la posición del elemento terminal. Puede actuar en modo punto a punto o bien en modo continuo, en cuyo caso recibe el nombre de control continuo de trayectoria

**Control cinemático:** Cuando además de la posición se regula la velocidad.

**Control dinámico:** Se tiene en cuenta también las propiedades dinámicas de manipulador, motores y elementos asociados.

**Control adaptativo:** Además de lo indicado en los anteriores controles, también se considera la variación de las características del manipulador al variar la posición.

**Control en lazo abierto y en lazo cerrado:** El control puede llevarse a cabo en lazo abierto o en lazo cerrado. En el caso del **control de lazo abierto**, se produce una señal de consigna que determina el movimiento, pero no se analiza si se ha realizado con exactitud o sea producido un error, al efectuarse en la realidad.

Ejemplo de sistemas de control de lazo abierto son:

- Control de motores paso a paso (PAP) o con movimiento cuantificado.
- Sistemas con salidas temporizadas.
- Sistemas con movimiento predeterminados.

El control de lazo abierto tiene muchas causas de error (inercia, interferencia, fricciones, desplazamiento, etc.), y si bien es muy simple y económico no se admiten en las aplicaciones

industriales, donde es fundamental la exactitud en la receptividad de los movimientos. Sin embargo, en robots dedicados a la enseñanza y el entrenamiento, este tipo de control esta muy extendido.

Los sistemas de robot industrial poseen un **control en lazo cerrado**. Este control hace uso de un transductor o sensor de la posición real de la articulación o del efecto final, cuya información se compara con el valor de la señal de mando o consigna, que indica la posición deseada. El error entre estas dos magnitudes, se trata de diversas formas para obtener una señal final, que aplicada con elementos motrices variará la posición hasta que coincida con la deseada.

### 1.2.2 Control por velocidad con PWM

La modulación por ancho de pulso (Pulse Width Modulation o PWM) se basa en el valor medio de una señal periódica que es igual a al integral entre 0 y el periodo de la función de la señal respecto al tiempo, dividiendo todo ello por el valor del periodo. En el caso de una señal lógica y considerando el periodo constante, podemos considerar que la función no es continua si no en trozos. Entonces obtenemos que el valor medio de la señal durante el periodo fijado es igual al valor de la señal en estado “alto” multiplicado por el tiempo en estado “alto” mas el valor de la señal en estado “bajo” multiplicado por el tiempo en estado “bajo”, y dividido todo ello por el tiempo total (periodo).

$$\bar{V} = \frac{V_H * T_H}{T_{TOTAL}} + \frac{V_L * T_L}{T_{TOTAL}}$$

Según este planteamiento, si generamos las señales de control al puente H, a una frecuencia fija y variamos proporcionalmente cuantas veces mandamos la señal de activación, en un sentido y cuantas veces mandamos la señal de detención, podremos variar la tensión media de alimentación del motor y por lo tanto la velocidad.

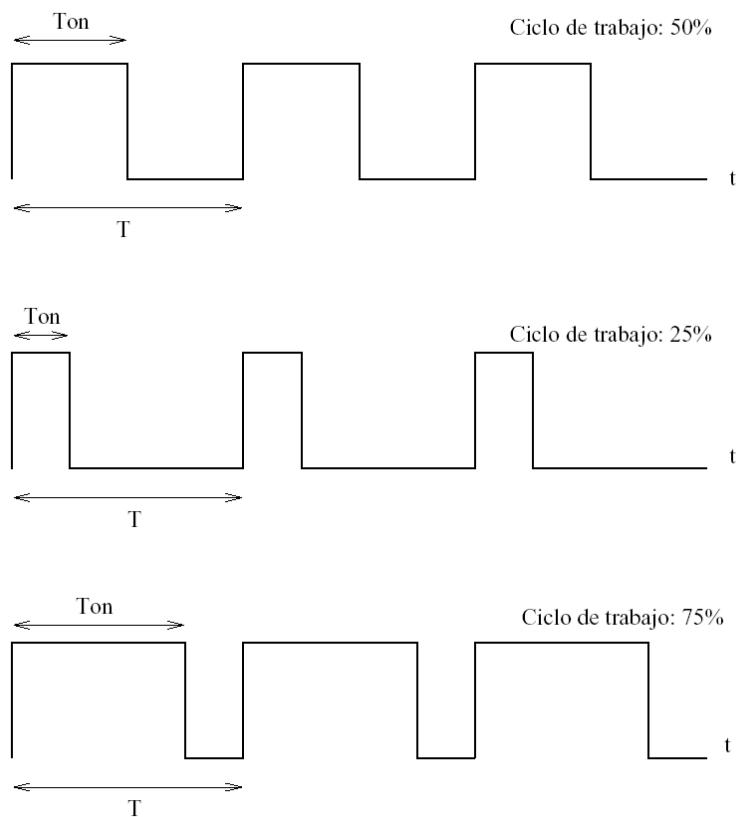
#### Señales PWM

Con las señales PWM podemos establecer la tensión media, cambiando la anchura de un pulso. Consideremos las señales de la figura 1. Todas ellas son de la misma frecuencia (periodo T).

Dentro del control por velocidad es necesario tener un dispositivo capaz de generar el PWM, por esta razón, se adopta utilizar un **microcontrolador PIC16F877**, el cual dentro de sus principales características es tener dos terminales para la modulación por ancho de pulso, el ciclo de trabajo se determina mediante la 10 bits que están en dos registros especiales para el PWM, variándolos de 0 a 1023 en decimal.

Dichos **registros de velocidad** son aplicados para cada motor, donde se especifica el valor medio de la tensión a aplicar (valor 0 implica tensión de 0v, valor 1023 la máxima tensión aplicada al motor).

La Primera señal tiene un ciclo de trabajo del 50 %. La mitad del tiempo la señal está a nivel alto y la otra mitad a nivel bajo. El valor medio de su tensión es 2.5v. La segunda señal tiene un ciclo del 25 %. Una cuarta parte del tiempo está a nivel alto y tres cuartas partes a nivel bajo. El valor medio de la tensión es de 1'25v. Finalmente, la tercera señal tiene un ciclo del 75 %: tres cuartas partes a nivel alto y una cuarta parte a nivel bajo. Tensión media: 3'75v. Según la anchura del pulso (Ton), conseguimos un valor medio u otro. Los motores de corriente continua responden a este valor medio y por tanto cambiando este valor conseguimos cambiar su velocidad.



**Figura 1-4. Señales con diferentes ciclos de trabajo**

### **1.3 Sistema operativo**

Es necesario hacer un breve comentario sobre las características que debería tener el sistema operativo que soporte el lenguaje con los requerimientos específicos del sistema. Generalmente la decisión de que sistema operativo utilizar depende de la decisión de velocidad del computador usado; si es suficiente rápido como para soportar un sistema y un lenguaje que manteniendo los requerimientos de respuesta a los eventos externos en tiempo acotado (y suficientemente corto), entonces la solución vale la pena, pues simplifica enormemente en la programación.

Para el manejo de LabVIEW debe de asegurarse que la computadora personal tenga Windows 98 o posterior instalado.

### **Clasificación de lenguajes de programas de robots**

Aunque ningún lenguaje de programación hoy en día cumple todos los requerimientos especificados anteriormente, hay varios que resultan útiles, dependiendo de la tarea específica a que se le destine.

Veamos la división de algunos lenguajes principales. En primer lugar estableceremos una clasificación por la sintaxis y complejidad. Aquí se distinguen tres tipos:

- Secuenciadores de instrucciones: simplemente almacenan y posteriormente repiten una secuencia de posiciones y acciones en un orden mas o menos fijo. Tales como posiciones y acciones se aprenden de varias maneras: Mediante movimiento de robot con un joystick, ratón o teclado especial suspendido del techo (“leach pendant”) Mediante movimiento manual y almacenamiento de las posiciones de los encoders. En este caso se pueden ejecutar los movimientos a diferente velocidad de la que se almacenaron
- Extensiones a lenguajes clásicos: son los módulos específicos para el manejo de sensores y actuadores, mas estructuras de datos adaptados, conservando la sintaxis general y control del flujo del lenguaje escogido. Se puede basar en BASIC, PASCAL, C, etc.

- Lenguajes específicos para robots: fueron diseñados por firmas comerciales para ser vendidos junto con sus manipuladores, teniendo en cuenta los sensores y actuadores a que se deben de conectar. Todos ellos incorporan el manejo de señales de los sensores, y de acuerdo a sus valores pueden cambiar en tiempo real el flujo del programa.

Como la programación de lenguajes tiene que evolucionar, el poder de los lenguajes tiene que ir creciendo. Un revolucionario concepto dentro de la programación, fue introducido por National Instruments en 1986 con LabVIEW 1. Fue un gran programa con muchas limitaciones de funcionalidad de programación, ya que la primera versión fue escrita específicamente para propósitos de adquisición de datos. Desde entonces hasta ahora con su nueva versión LabVIEW express 7.0, es uno de los más populares programas de adquisición de datos entre ingenieros, investigadores y científicos.

Por otra parte y respecto al nivel de abstracción que permiten a la hora de especificar la tarea, podemos clasificar estos lenguajes en:

- Orientados al robot: sus primitivas son comandos de movimiento para el robot, o repetición de lectura de sensores. Se deja al usuario la tarea de establecer explícitamente y de modo secuencial, cuáles deben de ser los movimientos a ejecutar, así como también cuál será el comportamiento de los sensores en cada instante
- Orientado a tareas: permiten al usuario especificar qué debe de hacer el robot, pero no necesariamente cómo debe de hacerlo, al menos hasta cierto punto. Las especificaciones de tarea se pueden dar en forma textual, o ayudándose de un interfase gráfico.

## **1.4 Definición de programar**

Programar es la identificación y especificación de una serie de acciones básicas, las cuales se ejecutan en un orden específico logrando una tarea o realizando algún proceso específico. Hay muchos niveles de programación en la computadora. En este caso usando LabVIEW, denominado como un lenguaje G por su programación gráfica.

### **1.4.1 Programación de Lenguaje gráfico LabVIEW**

Cuando escribimos líneas de códigos, ligas y compilamos un programa convencional, en realidad no sabemos si hay errores bajo el proceso de compilación. LabVIEW sin embargo, indica el estado de cualquier error bajo el proceso de compilación de tu desarrollo de código. Los lenguajes de programación convencional, usan regularmente archivos de texto, como sus códigos fuente, en cambio en LabVIEW se usa un archivo llamado VI (Virtual Instrument) como código fuente.

Los lenguajes convencionales usualmente, tiene tres niveles de código: (1) Alto nivel: el código fuente, estos programas se pueden crear tecleando líneas de comandos; (2) Medio nivel: El objeto de códigos contienen códigos ensamblados; y (3) Bajo nivel código maquina. Por otro lado LabVIEW tiene dos niveles: alto y bajo. El nivel alto de código con los programadores creadores de VIs y bajo nivel códigos de VIs que son compilados dentro de LabVIEW, representado en términos de conexiones entre iconos.

LabVIEW puede usarse no solo para adquisición de datos o aplicaciones de instrumentos de control, si no que también se usa para propósito de aplicaciones generales, tales como base de datos, programa de análisis de datos, redes de comunicación o un simple programa de juego. LabVIEW también tiene la variante de añadir herramientas de kits para una aplicación del usuario específico, tales como procesamiento de imágenes, análisis de datos de procesamiento digital de señales (DSP), aplicaciones en Internet, y muchas otras. Como un resultado del tamaño de aplicaciones de LabVIEW es el crecimiento que ha tenido desde 1986, y LabVIEW seguramente tiene lugar como una nueva programación de lenguaje de G en el campo de adquisición y análisis de datos, como bien otras aplicaciones surgen cada día.

#### **1.4.2 Programación de robots**

La programación de los robots es sustancialmente a la de los ordenadores aislados, aun cuando un programa escrito en un lenguaje para robots semeje a cualquier otro programa convencional. El hecho de estar controlando sistemas reales añade complejidad debido a un cierto grado de impredecibilidad que debemos de asumir y manejar. Es necesario detallar todos los puntos por los que debe de pasar en función del tiempo, especificar las posiciones, velocidades y aceleraciones de cada articulación como función del tiempo, y al mismo tiempo a las señales de los sensores de cada articulación y procesarlas, mas las posibles señales de interrupción provocadas por sensores externos, si lo hay.

Esto explica que sean necesarios lenguajes para especificar las tareas en términos accesibles a los humanos. Estos lenguajes serán usados para dos fines:

**Definir** la tarea que el robot tiene que realizar.

**Controlar** el robot mientras la realiza.

De lo dicho anteriormente se entiende que un lenguaje pretenda ser usado para programar robots deberá ser en algún sentido, diferente a los lenguajes convencionales. Esas diferencias son debidas a que:

El entorno del robot no puede describirse solo en términos cuantitativos; usa relaciones espaciales (arriba de, debajo de, sobre,...) e incluso temporales, o de causalidad (A no puede ponerse en B hasta que B no este en X,...)

El robot opera en un mundo real impreciso y el modelado del mundo que el programa maneja puede no dar cuenta debida de él, precisamente a causa de la imprecisión.

Además hay que tener en cuenta que los programas de robot deben atender a señales sensoriales que se producen en instantes impredecibles y por tanto, deben de estar preparados para interrumpirse en cualquier momento.

El concepto de programación en LabVIEW, es más intuición visual, y el usuario se familiariza debido a que la representación gráfica es más aceptable que líneas de código. Tal es el nuevo concepto que tiene la creación de una nueva expresión, *lenguaje G*, el cual muchos programadores lo aceptan y lo entienden.

## **1.5 Comunicación serie**

La comunicación serial es un tema amplio. Por esta razón nos enfocarnos solo en la comunicación de puerto serie asíncrona RS-232, primeramente en como trabaja en una PC, y enfatizando los conectores y la UART`s que es vista mas frecuentemente en las PCs de hoy en día. Después definiremos un puerto serie y RS-232, permitiéndonos llegar al corazón del puerto serie, la UART (Universal Asynchronous Receiver-Transmitter), atreves de conectores del puerto externos y cables.

### **1.5.1 El puerto serie.**

Un puerto serie es un conector y la relación electrónica que tiene es que usa un numero de entradas y salidas estándares para interfase comunicación de datos de equipos (DCE data communications equipment) y terminal de datos de un equipo (DTE data terminal equipment). El conector del puerto serie es usualmente de 9-terminales (DB-9) o un 25-terminales (DB-25), para otros tipos también existen (RJ-11, RJ-12, 8-pin DIN, RJ-45 o DB-37).

El manejo de los datos y señales del puerto serie son dirigidos por la transmisión-recepción universal asíncrona (UART). La UART usualmente es diseñada como un chip produciendo una señal eléctrica que es aceptada por los estándares serie de entrada salida.

### **1.5.2 El estándar RS-232**

Muchas PC`s, el estándar usual para entradas y salidas serie es llamada RS-232, pero hay otros estándares similares tal como el RS-422 y el RS-485 son también bastante comunes.

RS-232 es una especie de conexión serie descrita en una especificación escrita por Asociaciones de industrias electrónica (EIA Electronics Industries Association), en conjunción Asociaciones de industrias de telecomunicaciones (TIA), definiendo estándares para la transferencia tradicional de datos serie. Formalmente el estándar RS-232 es llamado EIA/TIA-232-F.

### **1.5.3 Especificaciones del puerto serie RS-232**

Las especificaciones del puerto serie RS-232 definen los tipos de comunicaciones RS-232 de equipos que involucran las características de señales, eléctricas y mecánicas de los puertos RS-232

#### **DCE y DTE**

La especificación RS-232 define dos tipos de comunicaciones de equipos: Data terminal Equipment (DTE) y Data Circuit-Terminating Equipment (DCE).

Los dos se difieren en el asignamiento de terminales -para propósitos prácticos en un PC puede ser considerado como DTE y para un MODEM como DCE. Una conexión RS-232 requiere un DTE y un DCE, o usar un cross-over comúnmente llamado null-modem que hace la conexión como si tuviéramos un DTE y un DCE (como la conexión entre dos PCs con un cable serial).

#### **Señales**

El estándar RS-232 define 25 líneas de señales en una interfase, en realidad solo se usan 9 de estas líneas. Solo tres de estas líneas RD, TD, y GND son requeridas para la comunicación de datos serie bi-direccional. Las restantes son DCD, DTR, DSR, RTS, DTS y RI que son asignadas para variedad de propósitos control.

#### **Eléctricas**

**Voltajes.** Las señales del RS-232 son indicadas por un voltaje diferenciado con respecto a tierra, y puede variar entre +3 a +15 volts y -3 a -15 volts.

Las líneas de control usan lógica positiva para indicar cualquier estado. Así un voltaje positivo en un cable portador con una señal de control (DCD,DTR,DSR,RTS,CTS, y RI ) indican que la señal de control es descrita como “True“. Y un voltaje negativo en una señal de control es descrito como “False”.

Las líneas de datos usan lógica negativa, lo que significa que un voltaje negativo en cable portador de señales de datos (RD o TD) es descrito como “True” y un voltaje positivo en el cable es interpretado como “False”.

Tiempos RS-232 también define el tiempo como una señal eléctrica. Una conexión RS-232 difiere entre los bits de una cadena de datos serie por colección de información acerca del voltaje de esta línea de datos. En simples términos, esto se hace por monitoreo de la línea a través de un bit de inicio y entonces se lee el estado de la línea para predefinir los intervalos, con cada intervalo representando el próximo bit de la cadena de datos. El tiempo del intervalo es definido por la velocidad de datos de la conexión, este proceso hace la conexión serie seguida por un reloj dentro de cada byte, a través del tiempo entre un byte de datos y el próximo que no es dictado por el reloj.

El número de lecturas tomadas dentro de un byte es determinado por el número de bits de datos para la conexión, abierta la conexión tenemos los bits de paridad y la configuración del bit stop. Una vez leído el bit stop, la conexión espera para el próximo bit start.

## Mecánicas

Cada línea en una interfase RS-232 es asignada para un número de terminales para varios conectores que puede usar el RS-232

RS-232 síncrono VS asíncrono

Las señales de RS-232 pueden ser síncronas o asíncronas.

Las señales síncronas del RS-232 son sincronizadas por un reloj que dicta el tiempo de cada bit que es mandado. El tiempo suministrado por un reloj que es compartido por ambos lados de la conexión serie, cada lado esta consciente del tiempo del próximo byte de datos.

Las señales de RS-232 asíncronas son delineadas por cambios de voltaje que son identificados como el Start y stop de cualquier byte de datos. Dentro de cualquier byte de datos, el receptor es aplicado a un reloj para medir los elementos de transmisión de datos, y muestrear los niveles de voltajes dentro del byte el número de tiempos que corresponde al número de bits discretos de datos esperados por el byte.

#### **1.5.4 El UART**

El núcleo o corazón central de cualquier puerto serie es un UART o Universal Asynchronous Receiver-Transmitter. Un UART es generalmente un circuito integrado. Este contiene el software necesario para convertir una cadena de datos paralelos de 8-bits dentro de bytes a un formato serie de solo bits y viceversa. Cuando se termina de transmitir un dato en una línea serie, el dato es transmitido en bit a tiempo, cuando se recibe un dato desde la línea serie, el UART convierte los datos serie convirtiéndolos en datos paralelos para que los use el CPU de la computadora. Para transmitir un byte de datos, el UART toma rupturas dentro de la constitución de bits, y paquetes que tienen un byte que satisfactoriamente puede ser identificado y reensamblado por el puerto receptor serial UART. Este proceso es llamado “framing” de un byte.

El UART transmite un byte a través de cables de la conexión serie. El UART también controla la forma datos manejados entre los buffers receptores y transmitidos del CPU de la computadora.

#### **Bits de STOP y START**

Cuando un byte es enmarcado (framing), el UART añade bits que indican el inicio y terminación de un byte. Sin los bits de start y stop, el flujo de datos serie debería ser diferenciado de una cadena. El UART añade un bit de inicio y opcionalmente 1 o 2 bits de stop. El voltaje del segundo bit de stop es el mismo como el del primero, también en términos prácticos un segundo bit de stop es innecesario. En pocos casos 1 ½ bits de stop pueden ser usados; esto en efecto simplemente mantiene el voltaje del bit stop para 1 ½ duración de un solo bit de stop.

#### **Paridad de bits**

Cuando un byte es enmarcado (framing) para transmitir, el UART también puede añadir un bit llamado “bit de paridad” para proporcionar un significado de chequeo ese dato recibido coincidiendo el dato que fue transmitido. La paridad de bits de RS-232 puede tener uno de cinco posibles configuraciones: none, even, odd, mark, o sapace.

Paridad “None”. Si la paridad es configurada como “none” no hay bit de paridad y es añadido al principio del dato mandado.

Paridad “Even”. Si la paridad es habilitada como “even” el UART mira cada bit de dato del byte que es preparado para enviar, y cuenta el número de bits de datos con un valor lógico de 1. Si la cuenta es un numero par, entonces un bit de paridad con un valor lógico de 1 es creado y añadido para el fin del byte, después de los bits de datos y antes del bit stop.

Cuando este byte es recibido en el otro lado de la conexión serie, la recepción UART cuenta el numero de bits de datos con un valor lógico de 1, determinando la cuenta si es par o impar. Entonces compara esta evaluación del status par o impar con el status indicado por el bit de paridad. Si ellos coinciden todo esta bien, si no coinciden un error es reportado.

Paridad “Odd”. Si la paridad es habilitada como “Odd”, el UART crea un bit de paridad solo como cuando se crea un bit de paridad “Even”, excepto el valor lógico “1” es configurado cuando la cuenta de bits con un valor lógico de 1 es impar. La recepción UART evalúa la calidad del dato recibido solo como lo hiciera con el bit de paridad “Even”.

Paridad “Mark”. Si la paridad es habilitada en Mark, la transmisión UART añadirá un bit de paridad con un valor lógico de 1 para el byte. La recepción del UART checa que este bit permanece cuando es recibid.

Paridad “Space”. Si la paridad space es habilitada en “space”, la trasmisión UART añadirá un bit de paridad con un valor lógico 0 al byte. La recepción UART checara el bit si permanece en 0 cuando es recibido.

Una pareja de cosas que pueden verse aquí; primero que cada lado de una conexión serie necesita para entenderse un bit de paridad suponiendo en el otro lado. Sin la propiedad de configuración de paridad el dato no tiene sentido cuando se recibe o se tiene un sentido inverso. Segundo la paridad “mark” y “space” no son realmente para el empleo de banda ancha: estos son creados para datos solos y tienen un malo significado de fijación de calidad de datos.

## **Bits de datos**

Los datos seriales de RS-232 pueden ser configurados para especificar el numero de bit de datos en un marco (frame). El numero de bit de datos que puede ser usado es de 4, 5, 6, 7, o 8. Cuando el UART manda los bits de datos, se manda un bit de datos del byte en orden del bit menos significativo (LSB) al bit de mayor significativo (MSB)

### Buffers and triggers

Muchos puertos serie de PCs están equipados con buffers UART, o pequeñas porciones de memoria para almacenar grupos de bytes en transito a través del UART. Los buffers mejoran la eficiencia de interconexiones del puerto CPU/serie, y ayudan a reducir los errores de transmisión llamados “overrun errors” esto ocurre cuando un nuevo byte arriba al puerto serie antes del previo byte que tiene a la izquierda el UART.

Los buffers del puerto serie son llamados “FIFO” debido a su operación First-In, First Out modo FIFO: los bytes se ponen dentro de un buffer son liberados desde el buffer antes de que los bytes arriben después.

Cuando un UART necesita el CPU para proceso de datos dentro o fuera de un buffer, esto, esto genera en el hardware una petición de interrupcion, o IRQ. Las peticiones de interrupción son muchas veces niveles básicos de flujo de control: esto indica al CPU de la computadora el manejo del puerto serie y ese puerto serie necesita algún tipo de atención.

Un buffer receptor UART almacena los datos recibidos desde el puerto serie, y reúne los bytes para el CPU los colecte.

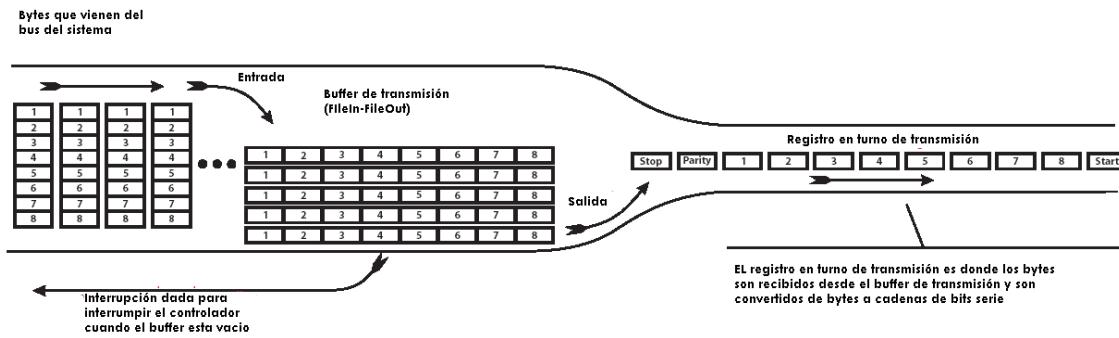
Un buffer transmite UART guarda los bytes que vienen desde el CPU, Reduce la frecuencia de interrupciones a la operación del CPU por la minimización de numeros de interrupciones al UART que debe pedir para tener el buffer lleno.

El UART tiene configuración ajustable cuando se tienen datos en los buffers que son liberados. Estos parámetros son los “trigger levels” del UART. El trigger level para una entrada buffer

indica el numero de bytes de datos requeridos para estar en el buffer antes de que el CPU responda a los colectados, el trigger level par el buffer de transmisión indica el numero de bytes de datos restantes en el buffer antes de que el CPU responda a que este llenado.

### Transmisión serie

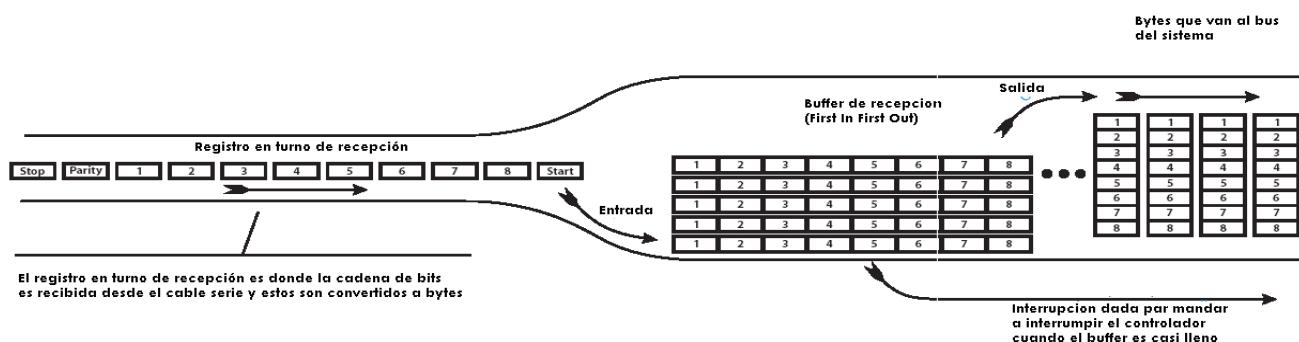
Cuando el UART transmite un dato el UART acepta bytes de datos desde el bus del sistema, y los convierte a una secuencia de bits que manda a través de una línea del puerto serie.



**Figura 1-5. Transmisión de datos serie**

### Recepción serie

Cuando se recibe un dato, el UART acepta bits de datos desde una linea del puerto serie y ensambla estos dentro de bytes para convertirlos y mandarlos hacia el bus del sistema.



**Figura 1-6. Recepción de datos serie**

## **1.6 Estructura del robot del proyecto**

La idea común que se tiene del robot , es la de un brazo mecánico articulado, pero este elemento no es más que una parte de lo que se considera técnicamente como un sistema de robot industrial, para este caso se trata de un robot de dos grados de libertad con movimientos en ronza y elevación.

El sistema de robot consta de las siguientes partes:

- 1.- Estructura de moviendo en ronza y elevación.**
- 2.- Microcontrolador PIC16F877.**
- 3.- El puente H**
- 4.- Elementos motrices.**
- 5.- Sensores**
- 6.- Fuente de alimentación**

Dentro de la estructura interna de los movimientos de ronza, que denominamos eje X y elevación eje Y. Se alojan, en muchas ocasiones, los elementos motrices, engranajes y transmisores que soportan el movimiento de las dos partes que, generalmente, suelen conformar el robot.

El número de elementos y el de las articulaciones del robot relacionan la determinación de los **grados de libertad**, que para el robot suelen ser 2 (dos), que coinciden en los movimientos independientes que posicionan las partes del robot.

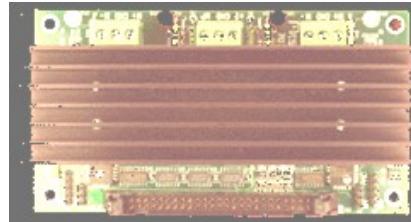
### **1.6.1 El puente H 7125**

La tarjeta 7125 es de canal dual, soporta 150 watt por canal. La 7125 deben de aplicarse 3 amperes del driver por canal con un motor alimentado con un voltaje de 12 a 48 VCD, ver figura1-3.

La 7125 puede operar con una tarjeta 6127 que es un controlador para servo motor. La máxima longitud del cable es de 10 pies (3.048mts). El pin uno del conector de 50 pines esta en

el lado izquierdo donde esta el indicador P3 viendo la tarjeta cuando estan orientadas los bloques de tornilleros a la parte superior tuya.

Tiene un header de 5 pines que es suministrado para los encoders. Un header de 3 pin que es suministrado para cada entrada de MOTORSENSE (sentido de motor).



**Figura 1-7 Puente H 7125**

Conecotor de interfase salida de pin P3

PIN#	Señal	PIN#	Señal
1	MOTOR1A (ENCODER) 25		/MOTOR1 DRV OT
3	MOTOR1B (ENCODER) 27		/MOTOR0 DRV OT
5	MOTOR0B (ENCODER) 29		MOTOR1SENCE
7	MOTOR0A (ENCODER) 31		MOTOR0SENCE
9	MOTOR1IDX	33	BIT 7
11	MOTOR0IDX	35	BIT 6
13	<b>MOTOR1PWM</b>	37	<b>BIT 5</b>
15	<b>MOTOR0PWM</b>	39	<b>BIT 4</b>
17	<b>MOTOR1DIR</b>	41	<b>BIT 3</b>
19	<b>MOTOR0DIR</b>	43	<b>BIT 2</b>
21	/MOTOR1ENA	45	<b>BIT 1</b>
23	/MOTOR0ENA	47	<b>BIT 0</b>
		49	+5 Poder

La conexión al motor y la alimentación del motor se hacen por medio de bloques de tornilleros en la 7125. La fuente de alimentación para los motores es necesario conectar la terminal negativa a la tierra lógica. Esta conexión es hecha externamente en el puente H. La conexión de los motores y la alimentación de estos es la siguiente

MOTOR 0 - (TB1)

PIN#	Señal
1	MOTOR DRIVE 1
2	POWER GROUND
3	MOTOR DRIVE 2

MOTOR 1 - (TB3)

PIN#	Señal
1	MOTOR DRIVE 1
2	POWER GROUND
3	MOTOR DRIVE 2

MOTOR POWER - (TB2)

PIN#	Señal
1	+5 (CON UNA RESISTENCIA DE 100 OHMS)
2	POWER GROUND
3	MOTOR POWER

## 1.7 Sensorización

### Necesidad e importancia de sensores

El desarrollo con éxito de la tarea de un robot depende de que este tenga la información correcta y actualizada a un ritmo suficiente mente rápido, de su propio estado y de la situación del entorno. En particular, deben conocerse posición, velocidad y aceleración de las articulaciones (al menos, una representación digital de estas magnitudes) para estar seguros de que el robot sigue una determinada trayectoria y también que alcance la posición final deseada en el instante requerido, y con la mínima o ninguna sobreoscilación.

### Sensores de posición

Como su nombre lo indica, son los que dicen la posición, o, mas exactamente, en que punto de su recorrido permitido se encuentra una articulación. Según esta sea rotacional o trasnacional, el sensor deberá tener una estructura mecánica adaptada a la medición de ángulos o de distancias.

### Optóinterruptores

Son interruptores de final de carrera (es decir, no detectan cuales la posición de la articulación, sino solo si esta ha llegado o no a un punto determinado de su recorrido, usualmente el tope). No usan contactos mecánicos, sino un fotodiodo(o fototransistor) y un LED (diodo emisor de luz) que emite frente a el. Al moverse la articulación un disco o tope acoplado a ella interrumpe la luz del LED dando al fototransistor un flanco negativo que es detectado por la circunferencia adecuada.

# II.

## LabVIEW

### 2.1 Introducción

El software es un tópico muy importante que requiere de especial cuidado, que sea flexible para futuros cambios, y preferiblemente que sea de fácil manejo, siendo lo mas poderoso e ilustrativo posible, de este modo LabVIEW es utilizado para el proyecto, cumpliendo con las características anteriores.

El software LabVIEW es un lenguaje de programación grafica el cual fue diseñado inicialmente para la adquisition de datos en 1986, desarrollando nuevas herramientas para LabVIEW, que contiene librerías de funciones y herramientas de desarrollo diseñadas específicamente para adquisition de datos y control de sistemas.

Una característica importante para el proyecto es la comunicación serie, en donde LabVIEW puede interactuar con esta, donde los controladores de instrumentos son necesarios para el manejo de la comunicación el puerto serie de comunicaciones RS-232, disponiendo de librerías de las que disponemos para tal fin, que eliminan la necesidad de aprender los comandos de programación de bajo nivel y complejos para cada instrumento.

Los dispositivos seriales para transmitir y recibir bits de datos en un orden secuencial, tienen muy baja velocidad de transmisión de datos en comparación a la comunicación por puerto paralelo. Sin embargo la comunicación serie es ampliamente usada debido a su simplicidad y la naturaleza de estar listo para correr. Por ejemplo, muchas de las computadoras en el mercado cuentan con por lo menos un puerto serie disponible, el cual puede ser tomado para cualquier hardware extra que requiera comunicación serie.

## 2.2 Labview

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) es un lenguaje de programación gráfico para el diseño de sistemas de adquisición de datos, instrumentación y control. Labview permite diseñar interfaces de usuario mediante una consola interactivo basado en software. Usted puede diseñar especificando su sistema funcional, su diagrama de bloques o una notación de diseño de ingeniería. Labview es a la vez compatible con herramientas de desarrollo similares y puede trabajar con programas de otra área de aplicación, como por ejemplo Matlab. Tiene la ventaja de que permite una fácil integración con hardware, específicamente con tarjetas de medición, adquisición y procesamiento de datos (incluyendo adquisición de imágenes).

### 2.2.1 Por que utilizar LabVIEW

Para elaborar la interfase entre la PC y el microcontrolador, se consideró que el lenguaje más apto es el LabVIEW, y las razones las siguientes:

- Es muy simple de manejar, debido a que está basado en un nuevo sistema de programación gráfica, llamada lenguaje G.
- Es un programa enfocado hacia la instrumentación virtual, por lo que cuenta con numerosas herramientas de presentación, en gráficas, botones, indicadores y controles, los cuales son muy esquemáticos y de gran elegancia. Estos serían complicados de realizar en bases como c++ donde el tiempo para lograr el mismo efecto sería muchas veces mayor.

- Es un programa de mucho poder donde se cuentan con librerías especializadas para manejos de DAQ, Redes, Comunicaciones, Análisis Estadístico, Comunicación con Bases de Datos (Útil para una automatización de una empresa a nivel total).
- Con este las horas de desarrollo de una aplicación por ingeniero, se reducen a un nivel mínimo.
- Como se programa creando subrutinas en módulos de bloques, se pueden usar otros bloques creados anteriormente como aplicaciones por otras personas.
- Es un programa que permite pasar las aplicaciones entre diferentes plataformas como Macintosh y seguir funcionando.

### **2.2.2 Aplicaciones de LabVIEW**

Labview tiene su mayor aplicación en sistemas de medición, como monitoreo de procesos y aplicaciones de control, un ejemplo de esto pueden ser sistemas de monitoreo en transportación, Laboratorios para clases en universidades, procesos de control industrial. Labview es muy utilizado en procesamiento digital de señales (wavelets, FFT, Total Distorsion Harmonic TDH), procesamiento en tiempo real de aplicaciones biomédicas, manipulación de imágenes y audio, automatización, diseño de filtros digitales, generación de señales, entre otras, etc.

Labview puede comunicarse con hardware tal como adquisición de datos, visión y dispositivos de control de movimiento, BPIB, PXI, VXI, RS-232 y RS-485. También incluye características para conectar sus aplicaciones a la Web empleando el servidor Web de Labview y estándares de software tales como red de trabajo TCP/IP y Active X.

### **2.3 Ambiente LabVIEW**

LabVIEW posee un conjunto extenso de herramientas para la adquisition, análisis, despliegue y almacenamiento de datos, así como herramientas para ayudarle a resolver problemas con su código.

### **2.3.1 Programación gráfica con Labview**

Cuando usted diseña programas con Labview está trabajando siempre bajo algo denominado VI, es decir, un instrumento virtual, se pueden crear VI a partir de especificaciones funcionales que usted diseñe. Este VI puede utilizarse en cualquier otra aplicación como una subfunción dentro de un programa general. Los VI's se caracterizan por: ser un cuadrado con su respectivo símbolo relacionado con su funcionalidad, tener una interfaz con el usuario, tener entradas con su color de identificación de dato, tener una o varias salidas y por su puesto ser reutilizables.

Cada programa realizado en Labview será llamado Instrumento Virtual (VI), el cual como cualquier otro ocupa espacio en la memoria del computador.

La memoria usada la utiliza para cuatro bloques diferentes como son:

- **EL PANEL FRONTAL:** Donde se ven los datos y se manipulan y controlan.
- **EL DIAGRAMA DE BLOQUES:** En este se aprecia la estructura del programa, su función y algoritmo, de una forma gráfica en lenguaje G, donde los datos fluyen a través de líneas.
- **EL PROGRAMA COMPILADO:** Cuando se escribe en Labview, el algoritmo escrito de forma gráfica no es ejecutable por el computador, por tanto, Labview lo analiza, y elabora un código ensamblador, con base en el código fuente de tipo gráfico. Esta es una operación automática que ocurre al ejecutar el algoritmo, por tanto no es importante entender como sucede esto. Lo que si es algo para apreciar, es que en este proceso, se encuentran los errores de confección que son mostrados en una lista de errores, donde con solo darle doble click al error, se aprecia en el diagrama de bloques, donde ocurre éste, para su corrección.
- **LOS DATOS:** Como el algoritmo maneja datos, requiere de un espacio en memoria para estos, lo que hace tomar en cuenta que el computador usado debe tener la memoria suficiente para manejálos. Por ejemplo, cuando se usan grandes matrices en cálculos se puede requerir de mucho espacio.

### **2.3.2 Diseño de la interfaz de usuario a partir de su código.**

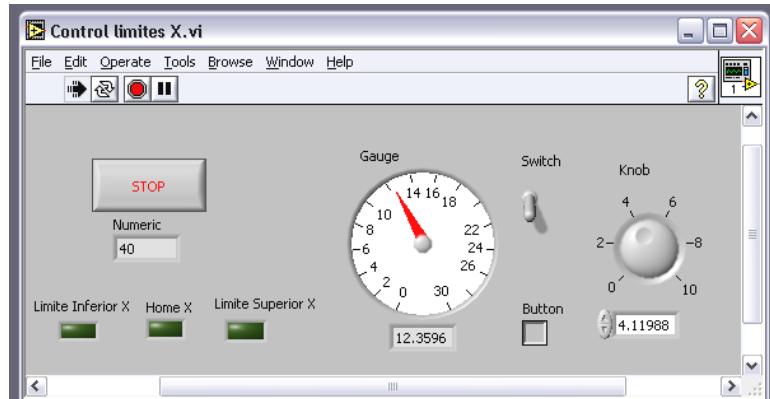
En el ambiente de trabajo de Labview existen dos paneles, el panel frontal y el panel de programación ó diagrama de bloques; en el panel frontal se diseña la interfaz con el usuario y en el panel de programación se relacionan los elementos utilizados en la interfaz mediante operaciones que determinan en sí como funciona el programa o el sistema, exactamente es la parte donde se realizan las especificaciones funcionales.

## **2.4 Estructura de un VI**

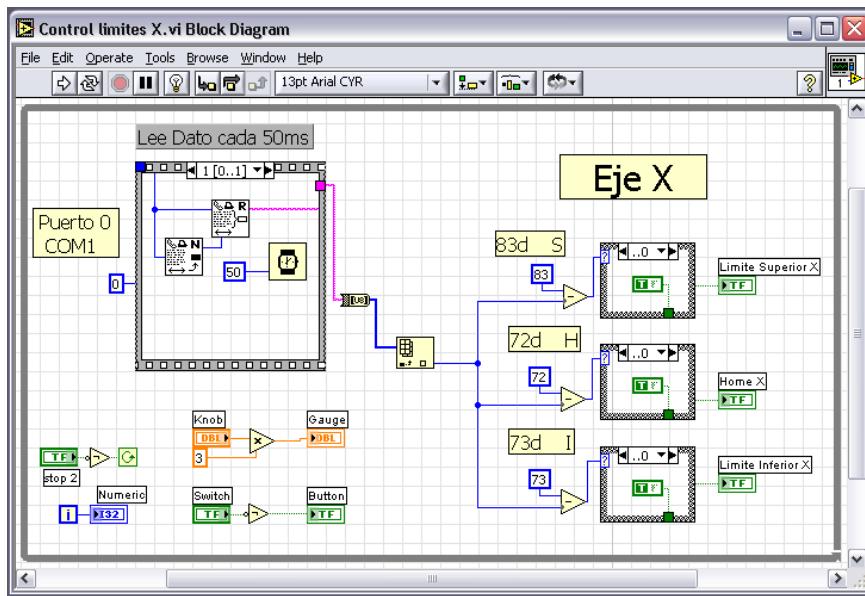
### **2.4.1 Ventanas de panel frontal y de diagrama**

Cuando se carga LabVIEW dando doble clic en el ícono de LabVIEW, se abrirán dos ventanas: una con un fondo gris, el cual es llamado panel frontal, y otra con fondo blanco, llamado panel de diagrama o diagrama de bloques. Figura 2-1 y 2-2 mostrándolos respectivamente.

En el panel frontal podemos encontrar instrumentos como: indicadores (gauges), switches, perillas (knobs), y botones. Todo el alambrado y cada componente individual son ocultados en la ventana que corresponde al diagrama de bloques en LabVIEW.



**Figura 2-1 Panel frontal de un VI.**



**Figura 2-2 Diagrama de bloques de un VI.**

Los botones en la ventana del panel frontal y diagrama de bloques son:



**Run:** Cuando este botón es presionado, LabVIEW ejecutara el VI que este actualmente seleccionado.



**Run continuamente:** Este botón corre un VI continuamente, hasta que el botón de abortar la ejecución el presionado.



**Abortar ejecución:** Este detiene la ejecución de un VI.



**Pausa:** Pausa la ejecución. Para continuar se presiona el botón otra vez



**Ejecución Highlight:** La foco es encendido dando un clic en este botón. Si esta encendido. LabVIEW desplegará un flujo de señal mientras corre el VI. Un punto corre a través del cable mientras muestra valores de cada nodo de los indicadores de la señal de flujo. Este es muy útil para depurar un VI, observando cual es el flujo de la ejecución; sin embargo, la velocidad de ejecución es mucho menor que en ejecución normal.



**Step into:** Este es diferente al botón de **Run** ya que suministra la capacidad de depurar.

Si das clic en este botón se activara el botón Run, mientras que LabVIEW pasara a través de cada nodo de alambrado de cada elemento. La diferencia de la **Ejecución Highlight** es que corre un nodo y espera al próximo clic del botón, donde **Ejecución Highlight** corre (lento) sin detenerse.



**Step Over:** Mientras corres un VI con Step into, este botón permitirá bricarse cualquier nodo o elemento.



**Step out:** Este sale del modo de corre **Step into**.

13pt Arial CYR



**Configuración de texto:** Permite cambiar el estilo de letras, color, tamaño, etc.



**Alineación Objetos:** Podemos alinear objetos verticalmente y horizontalmente en la ventana. Primero, selecciona el objeto que quieras alinear, entonces selecciona uno de los modos desde el menú que aparece abajo después de que oprimes el botón.



**Distribución de objetos:** Puedes controlar el espacio entre objetos. Este termina por la primera selección de objetos que son escogidos en el menú que aparece abajo después de que oprimes el botón.



**Reordenar:** Con esta característica, se puede traer objetos al frente o hacia atrás, o moverlos hacia arriba o debajo de otro objeto. Es utilizado cuando hay múltiples objetos que están encimados y se trata para cambiar la visibilidad en la pantalla.

#### 2.4.2 Controles e indicadores

Considerando un analizador de espectros como ejemplo para explicar los objetos en el panel frontal y el diagrama de bloques, tenemos que este tiene unos switches, para seleccionar diferentes bases de tiempo o diferentes velocidades de muestreo en el panel con un display del espectro. Si tu abres este caso, verás el alambrado el icono de transformada rápida de fourier (FFT), con sus unidades y algunas otras cosas mas. En LabVIEW estos son considerados objetos, y estos son elementos fundamentales que constituyen un VI.

En LabVIEW hay dos tipos de objetos: controles e indicadores. Los controles son objetos tales como perillas (knobs), (levers), o switches que permiten entrar valores. Los indicadores son solo para propósito de desplegar como pueden ser graficas, gauges o medidores de nivel

#### 2.4.3 Comunicación por puerto serie.

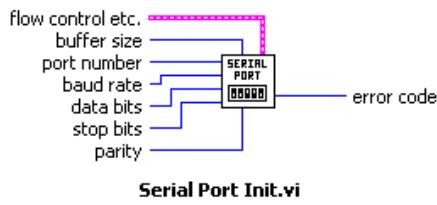
La comunicación serie es un medio popular de transferir datos entre computador y un dispositivo periférico tal como un instrumento programable en este caso un microcontrolador

PIC16F877. La comunicación serie se emplea para enviar datos, un bit a la vez, sobre una línea de comunicación hacia el receptor, de este modo con LabVIEW se logra el objetivo manejando librerías de funciones y herramientas para este fin

La velocidad de transmisión de datos serie es medida en bits por segundo (bps) o velocidad de baud (baud rate). Los bps indican como muchos bits datos pueden ser transmitidos por segundo, en cambio baud rate representa como muchos estados pueden ser transmitidos por segundo. Por ejemplo, si dos estados de datos deben de ser representados, 1 bit denotara todos los estados (cualquiera alto o bajo). En este caso el bps y el baud rate son iguales. Sin embargo, si usas cuatro estados para representar un dato, necesitas 2 bits, y el bps deberá ser dos veces el baud rate. Por lo tanto, un bps y un baud rate serán el mismo solo para representación de dos estados de datos en el sistema.

#### 2.4.3.1 VIs para la comunicación serie

Hay seis VIs para la comunicación serie en **Functions >> Input >> Instr Drivers >> Serial**, pero en esta sección solo se cubrirán cinco de los que son mas comúnmente usados.



**Serial Port INIT.vi** Este VI inicializa las especificaciones del puerto serie usando varios parámetros que se muestran en el VI siguiente.



**Bytes At serial Port.vi** Indica en la variable byte count el numero de bytes en el buffer de entrada del Puerto serie indicado en port number.



**Serial Port Write.vi** Escribe los datos en string to write en el puerto serie indicando el port number.



**Serial Port Read.vi** Lee el numero de caracteres especificados por requested byte count del puerto serie indicando el port number. Los bytes leidos se van a una salida que es el string read.



**Close Serial Driver.vi** Cierra el puerto serie especificando en port number.

#### 2.4.4 Definición de un Sub VI

Regularmente los lenguajes de programación basados en texto tales como FORTRAN, PASCAL, C, y otros, tienen un concepto de subrutina que es muy esencia para el código, ya que no puede estar disponible su código sin el uso de subrutinas. Similarmente el LabVIEW se cuenta con la correspondencia de subrutinas y estos son llamados sub VI. Como hacer subrutinas, en LabVIEW un sub VI puede tomar parámetros, ejecutar tareas y regresar resultados. Desde que LabVIEW es un lenguaje (G) de programación grafica, una de las expectativas que tiene son las diferentes formas de involucrar los sub VI, desde que un sub VI es representado como un instrumento virtual, se puede tener alambres para conexiones y objetos para componentes individuales. Por lo tanto los parámetros pasados usando alambrado, se puede llamar subrutinas para incluir objetos (sub VI) en el diagrama de bloques.

##### 2.4.4.1 Creando Sub VI

La forma que se presenta a continuación de cómo crear un sub VI, es una forma alternativa ya que dentro del proyecto realizado es mas apta por el desarrollo del control que es desarrollado en un solo VI, que posteriormente se utiliza para crear los SubVI necesarios para su flexibilidad a cambios futuros.

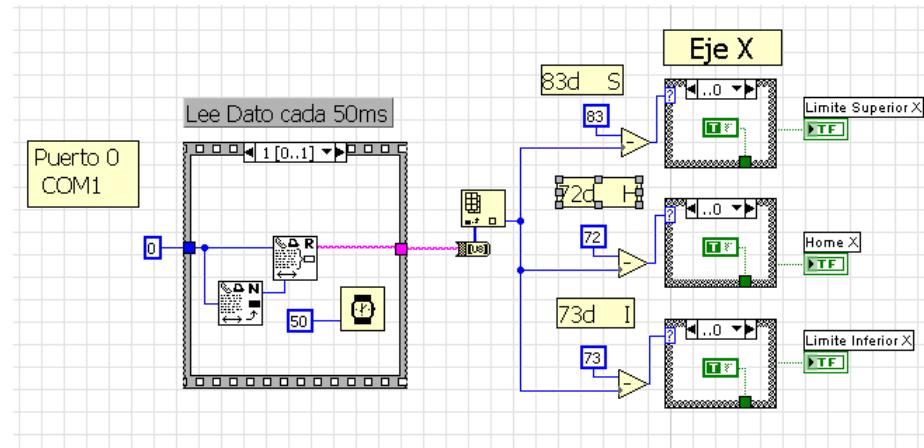
Supóngase que se tiene un VI que se muestra en la figura 2-3.

Los pasos son los siguientes:

Paso 1: Selecciona la porción con el cual tu quieras crear un SubVI.

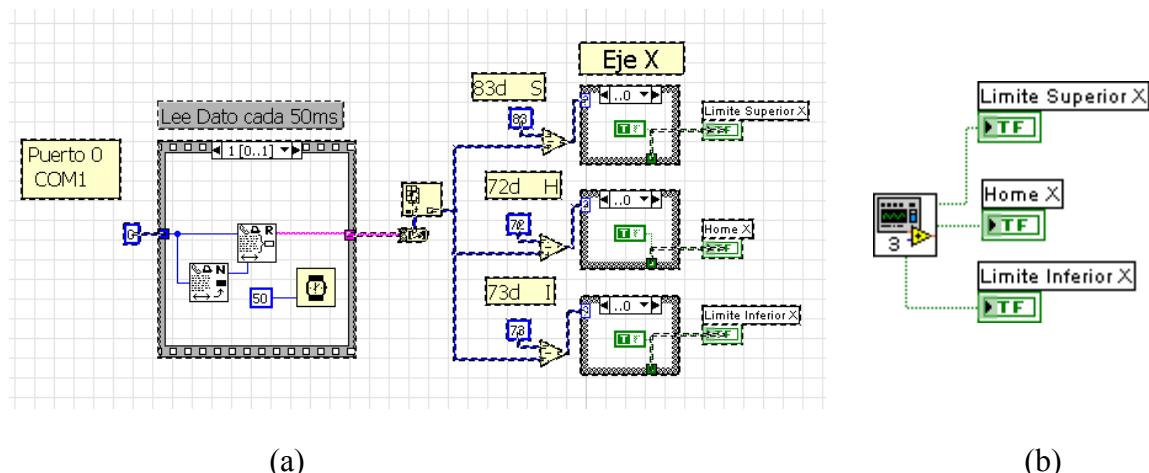
En la figura 2-3, seleccionaras el diagrama completo.

Paso 2: Selecciona del menú **Edit >> Create Sub VI**. Este creara un Sub VI con tres terminales limite superior X, Home X y limite inferior X.



**Figura 2-3 Diagrama de bloques para ejemplo para crear un sub VI**

Cada paso que es mostrado en la figura 2-4. Es el método es muy conveniente ya que no tienes que asignar cada terminal al objeto en tu Sub VI; sin embargo, esto puede causar confusiones si la estructura de entrada y salida es mas complicada.



**Figura 2-4**

**(a) Porción seleccionada en el diagrama de bloques, con el cual se creara un nuevo Sub VI.**

**(b) Seleccionando Edit >> Create Sub VI crearemos el sub VI de la porción seleccionada.**

## 2.5 Estructura del programa del control por velocidad

El siguiente programa es dividido en módulos los cuales se presentaran a continuación explicación su función de cada uno de ellos para finalmente integrarlos en el programa completo.

### 2.5.1 Escribiendo datos por el puerto serie

Aquí se presenta como configurar el puerto serie para la transmisión de datos.

Primeramente usaremos una secuencia de estructura que es como una colección de marcos de películas. Que cuando iniciamos la película las proyección de los marcos son secuenciales desde el marco 0, 1, 2, y mas.

En esta secuencia de estructura introduciremos, la configuración del puerto serie, la escritura del dato, un retardo, y cerraremos el puerto.

Configuración del puerto serie. En la sección 2.4.3.1 se determina la función los iconos para la comunicación serie para este caso los mas importantes para la comunicación por el puerto serie es la velocidad de baudios que es de 19200, 0 para el COM1, 0 para el tamaño del buffer, 8 bits de datos, 1 bit de alto, y sin paridad, ver figura 2-5. Los parámetros adoptados por el icono deben de tomarse en cuenta para el PIC16F877 y lograr una comunicación satisfactoria

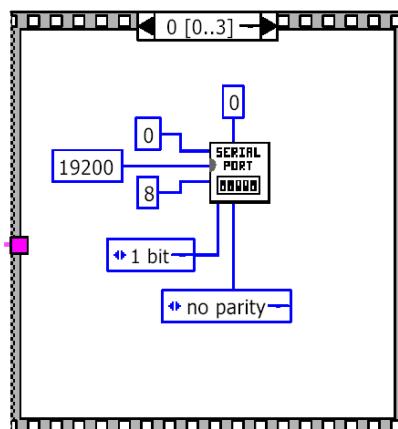
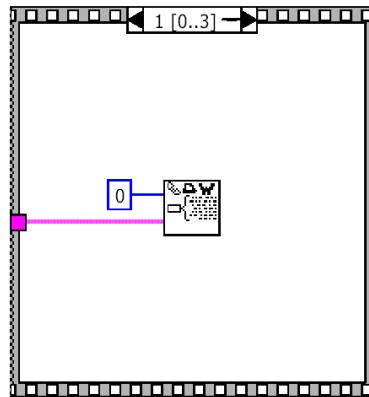


Figura 2-5 Configuración del puerto serie

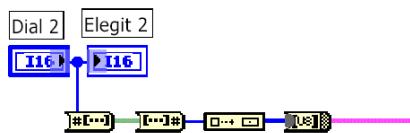
Secuencia de estructura 1; en marco de la estructura el icono de Serial Port Write, se configura el puerto que va a realizar la escritura y la cadena que se escribe, ver figura 2-6.



**Figura 2-6 Escritura por el puerto 0 con entrada de datos tipo cadena**

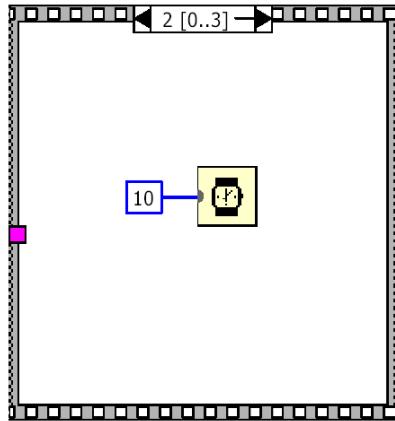
Para tener bien claro la conversión de datos se explica el siguiente diagrama de bloques Figura 2-7. Explicándolos de izquierda a derecha.

Tenemos el icono Dial 2 que nos permite la selección de datos decimales, con su indicador Elegit 2, posteriormente el numero es convertido por el icono a un array booleano de 8, 16, o 32 elementos. Después convertimos el array booleano con el icono a enteros de 32 bits sin signo y el elemento del array empieza con el menos significativo de los bits. Posteriormente el entero de 32 bits es concatenado por un array de 1 elemento con el icono para tener un array de byte sin signo, que entra al icono a bytes sin signo en representación de caracteres ASCII dentro de una cadena que entra a la terminal de string del icono Serial port write.



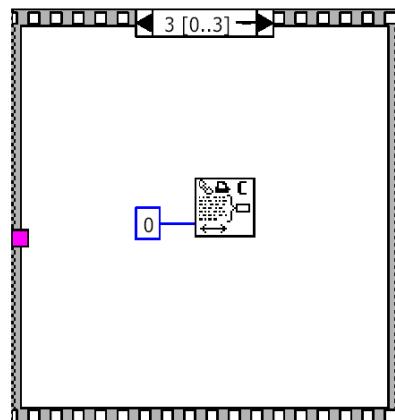
**Figura 2-7 Conversión de datos de entrada tipo cadena**

En el marco 2 tenemos la función Wait(ms)  que agrega el tiempo de espera al tiempo de ejecución del código, como se muestra en la figura 2-8, que también puede ser interpretado como un retardo, en este caso de 10ms.



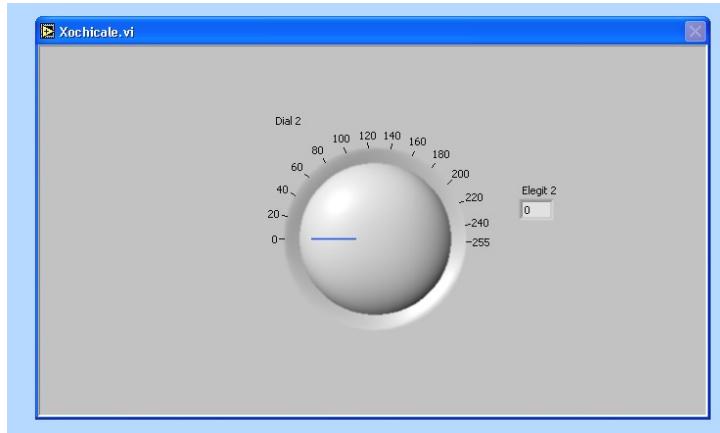
**Figura 2-8 Retardo de 10ms**

Por último el marco 3, donde utilizamos el ícono Serial port close, cerrando el puerto 0, mostrando lo en la figura 2-9



**Figura 2-9 Cerrando el puerto 0 ( COM1 )**

Con la estructura de secuencias anterior, podemos mandar de 0 a 255 datos en decimal, ya que solo podemos transmitir 8 bits y por lo tanto esto nos representa 255 datos decimales. El icono dial 2 panel frontal se muestra en la figura 2-10 donde nosotros seleccionamos el dato que se envía por el puerto serie de la computadora.

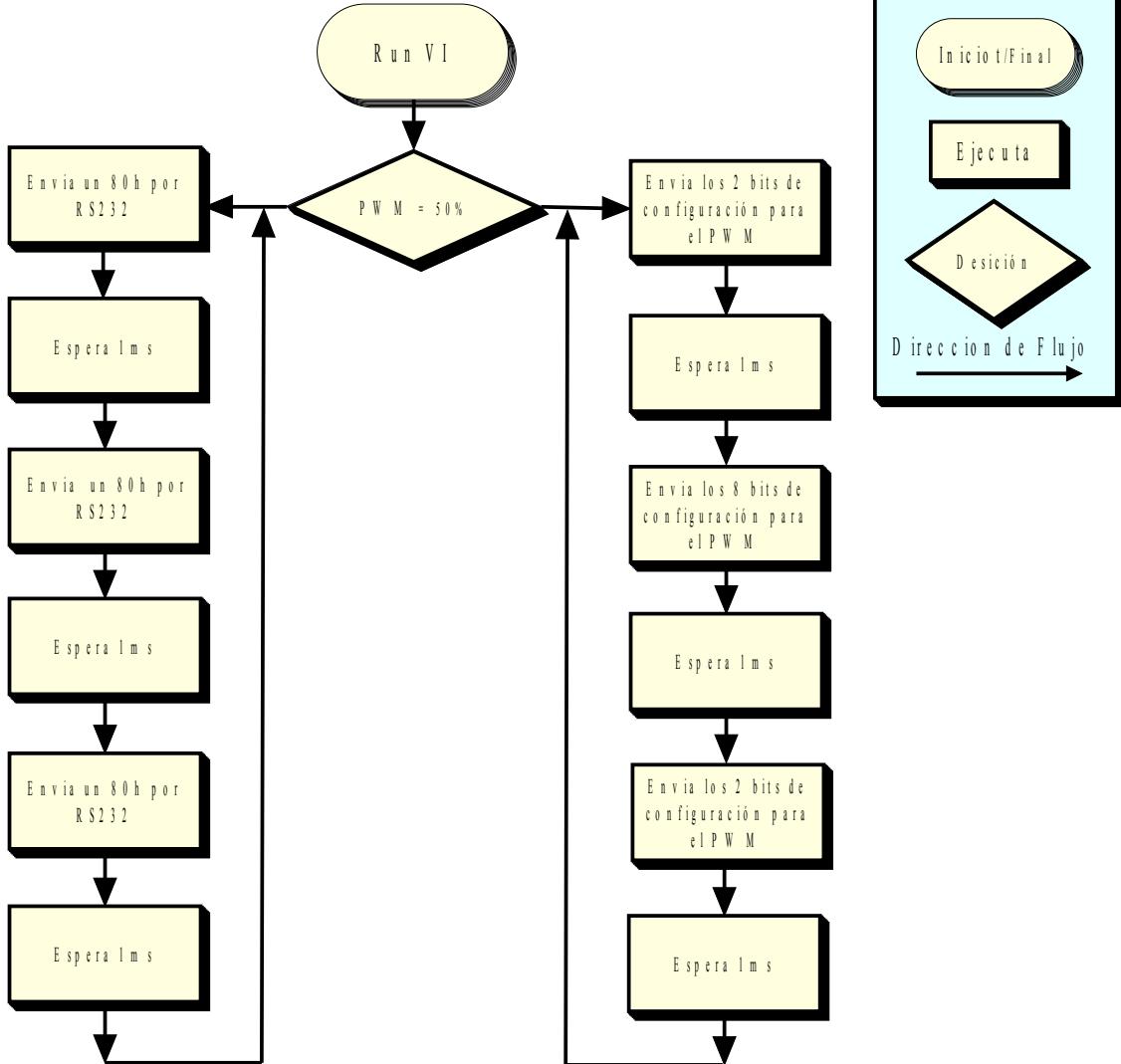


**Figura 2-10 Panel frontal para elegir dato a transmitir con su respectivo indicador**

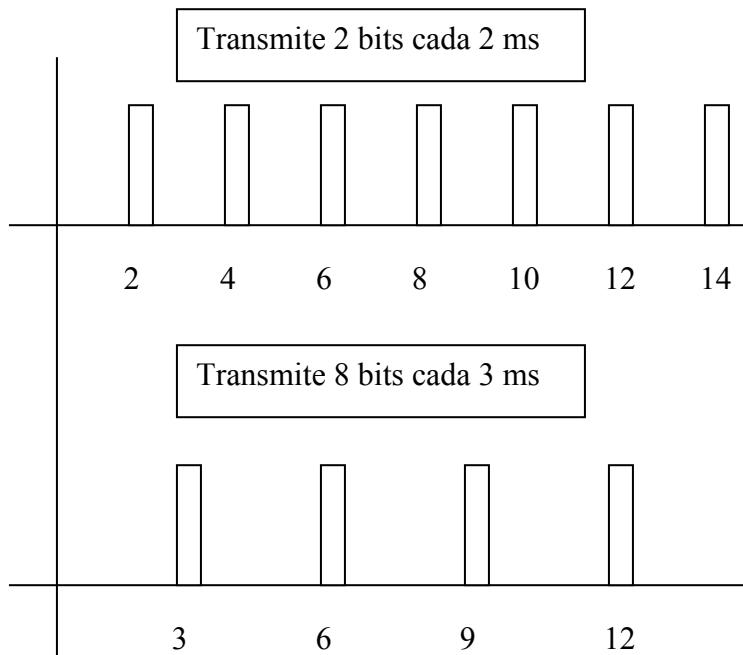
### 2.5.2 Algoritmo de transmisión de 10bits para PWM

Debido a la comunicación serie que tiene la configuración para transmitir 8 bits, es necesario implementar un algoritmo capaz de transmitir 10 bits necesario para la configuración del ciclo de trabajo del PWM del PIC16F877, ya que este cuenta con la configuración de 10bits, por la razón de tener un mejor resolución en el ciclo de trabajo del control del PWM.

V I para control de PW M 10 bits



mandar los datos por el puerto serie, se envian 8 bits, de esta forma, para poder enviar los 10 bits primeramente enviamos los 8 bits que contienen los 2 bits de configuración del PWM, y posteriormente los 8 bits que completan la resolución que requiere el PIC16F877 para la configuración del ciclo útil, en la siguiente figura 2-11 se muestran los tiempos en los que son enviados los bits y debido a que es un ciclo continuo siempre van a llevar la secuencia que determinada para enviar los datos.



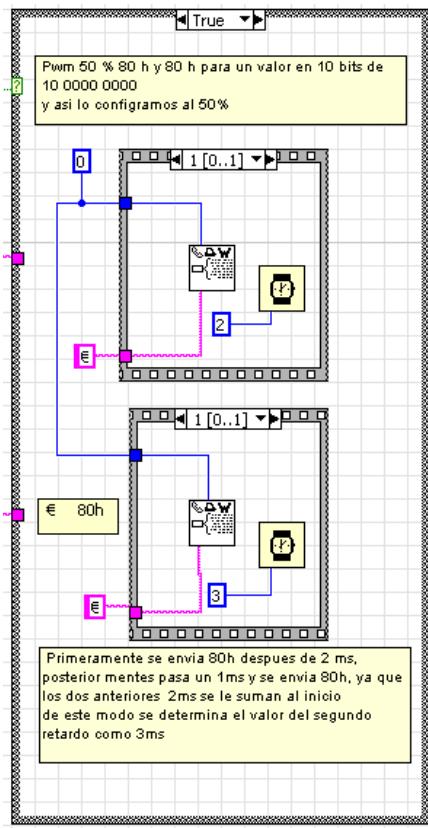
**Figura 2-11 Tiempo de transmisión de datos para el PWM de 10bits**

### 2.5.3 Instrumento virtual para transmitir los 10 bits del PWM

En el control de la velocidad se requiere una posición estática, es decir sin movimiento, debido a esto se diseña el envío de 10bits que configuran el ciclo útil del PWM a 50% cada que se oprime un botón, por otro lado esta el control del PWM que depende del usuario, añadiendo un selector de los bits que se envían decidiendo el PWM dependiendo del valor del selector.

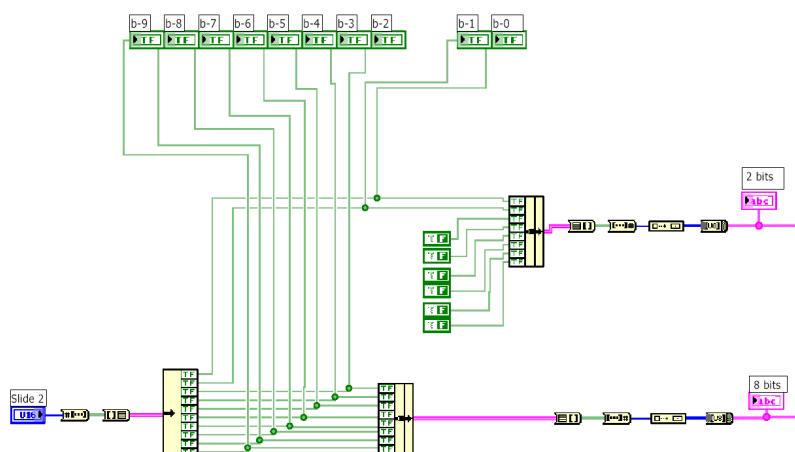
A continuación se da la explicación del diagrama de bloques y el panel frontal para cada una de las funciones iniciando con el PWM de 50%.

En una estructura de casos que contienen dos opciones falso y verdadero, elegiremos entre control de PWM y PWM al 50% con el botón , cuando el botón esta en la opción verdadera, se ejecuta la secuencia de estructura que contiene el retardo de 2ms mandando un 80 Hex por el puerto serie (figura 2-12 superior), posteriormente la secuencia de estructura que contiene el retardo de 3ms, mandando un 80 Hex, (figura 2-12 inferior) las dos estructura configuran el puerto 0 para la transmisión de dato.



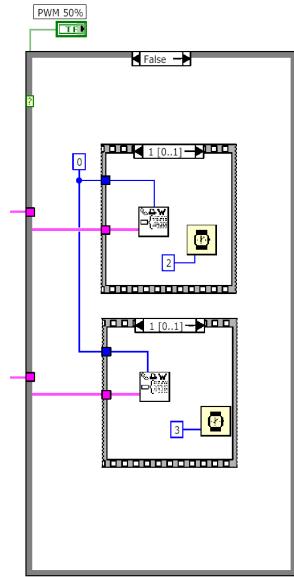
**Figura 2-12 PWM configurado con 10 bits para un 50% de ciclo util.**

Cuando el botón es puesto en falso, se tiene el control del ciclo útil de PWM, estos son generados por un selector de bits de 0 – 1023 números en decimal y convertidos de numero a array y de array a cluster que entran a un icono que expande los bits que es llamado unbundle, para mandarlos a cada uno de los conversores (bundle) que los manda al string wire del serial port write, para transmitirlos, figura 2-13.



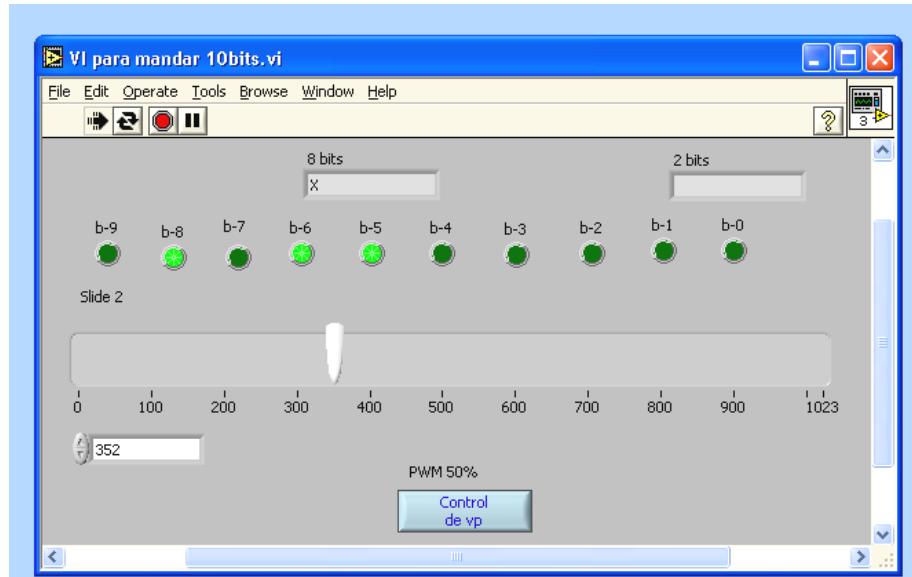
**Figura 2-13 Generación de 10 bits en dos cadenas una con 2bits y otra con 8 bits.**

Posteriormente con enviados a cada uno de los iconos para su transmisión respectiva (figura 2-14).



**Figura 2-14 Trasmisión de 10 bits para el control del ciclo util del PWM**

El panel frontal correspondiente se muestra en la figura 2-15 donde tenemos el selector de los números decimales para transmitir y el botón para PWM de 50% o Control de PWM



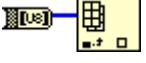
**Figura 2-15 VI para mandar 10bits**

#### 2.5.4 Recepción de datos para compararlos y entrar a una rutina determinada.

El botón PWM 2 tiene el objetivo de seleccionar por medio de una estructura de caso entre Recibir el dato y compararlo para ejecutar determinada rutina o sencillamente no hacer nada para el caso falso.

Cuando el botón PWM 2 es activado en verdadero se ejecutan el siguiente diagrama de bloques.

La secuencia de estructura (figura 2-16) que nos proporcionar lo que lee en el puerto serie

cada 10ms, este string leído se va a los iconos  para su conversión de string a un array de bytes y de un array de n-dimensión a un elemento del array, para hacer posible la resta con el dato que esperamos recibir, de este modo si esperamos un 69d y recibimos cualquier otro numero, la resta nunca no es cero, por lo tanto cuando el dato recibido es un 69d la resta es cero y ejecuta lo que tenemos en el caso de estructura.

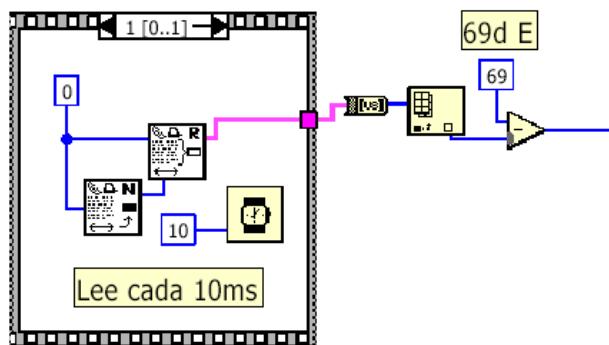
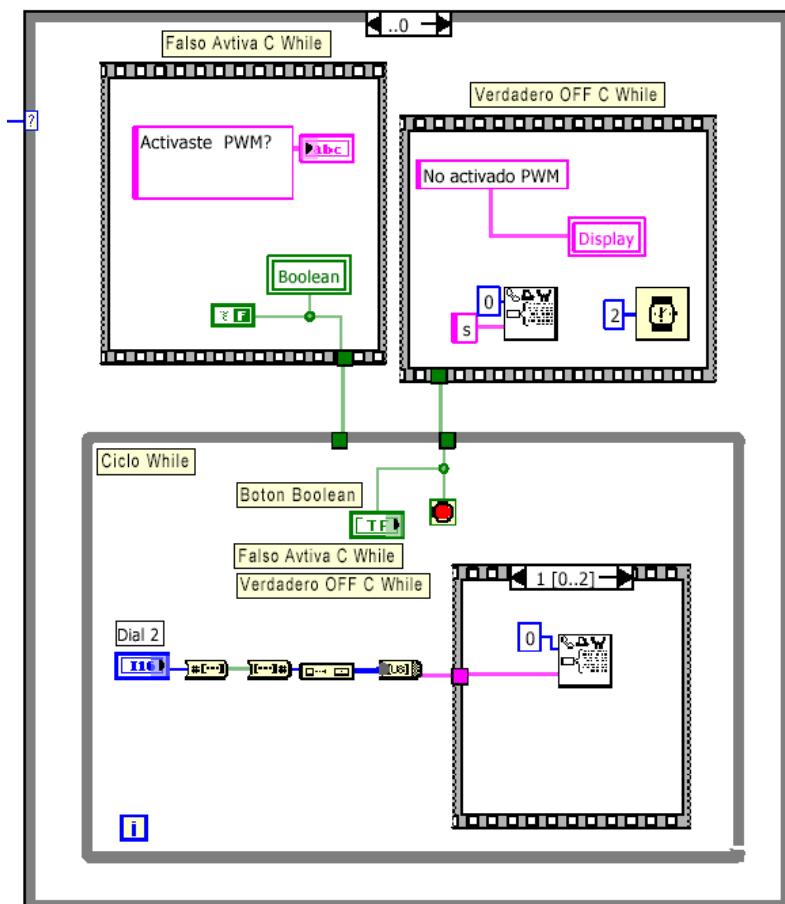


Figura 2-16 Lectura del dato cada 10ms en el puerto serie y comparación de dato.

En el momento que el dato recibo es un 69d la resta se hace cero y entra a la siguiente rutina; que nos activa un letrero donde nos indica **Activaste PWM?**, con el botón en azul cielo de leyenda **Si deseas detenerlo clic AQUÍ**.

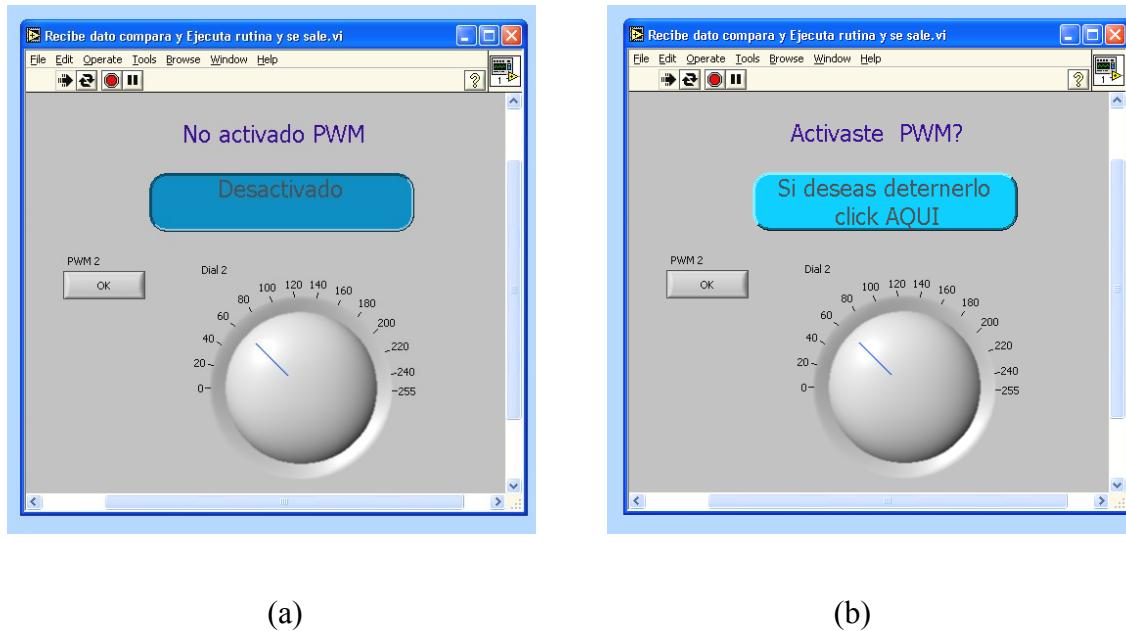
Esto es debido a que en el ciclo While tenemos un botón que llamado Boolean que tiene una variable local encontrada en la estructura de secuencia en la parte superior izquierda, dicho botón nos elige entre falso y verdadero, de este modo cuando el botón es falso activa la secuencia de estructura de la parte superior izquierda donde se coloca un display con la leyenda Activaste PWM? y así hacer posible el control del selector para transmitir datos por el puerto serie, y cuando el botón Boolean tiene el valor de verdadero desactivando el ciclo while ya que por este

icono  se detiene en verdadero y activa la estructura de secuencia que está en la parte superior derecha con la variable local Display de leyenda **No activado PWM**, transmitiendo una s después de 2ms para proporcionar la información necesario que posteriormente se ocupara con el PIC16F877, ver figura 2-17.



**Figura 2-17 Activa PWM con los botones indicadores y displays**

El panel frontal correspondiente al anterior diagrama de bloques se muestra en la figura 2-18 donde tenemos dos figuras que nos indican cuando el PWM es activado mandando un 69d “E” en ASCII Figura 2-18 (b) y cuando el PWM es desactivado dando clic en el botón y mandando un “s” en ASCII Figura 2-18 (a).



**Figura 2-18 Indicadores y displays para activación y desactivación de PWM**

#### 2.4.5 Recepción de datos y activación de indicadores.

Para este VI es necesario, generar las condiciones en las que posiblemente se encuentre cuando este en funcionamiento, por este motivo se agrega la secuencia de estructura que Envía una X cada 10ms, posiblemente estos puedan ser los datos que se envíen para configurar el ciclo útil del PWM en le PIC16F877, también encontramos una estructura de casos que envia un 00h, cada que es activado el botón correspondiente.

Por otro lado al momento de recibir datos, utilizamos el principio de comparación anterior “la resta”, si recibimos un 83d activamos el Indicador Limite Superior X si recibimos un 72d activamos el indicador Home X y si recibimos un 73d se activa el Limite Inferior X, de lo contrario si no es ninguno de los datos recibidos correspondientes para que las restas se haga cero ninguno es activa.

También muestra una nueva característica es la desactivación del ciclo while, que correrá hasta que sea desactiva este ciclo por medio de un botón haciendo que continúe mientras sea verdadero, si es presionado el botón se sale del ciclo.

En la siguiente figura 2-19 esta su diagrama de bloques.

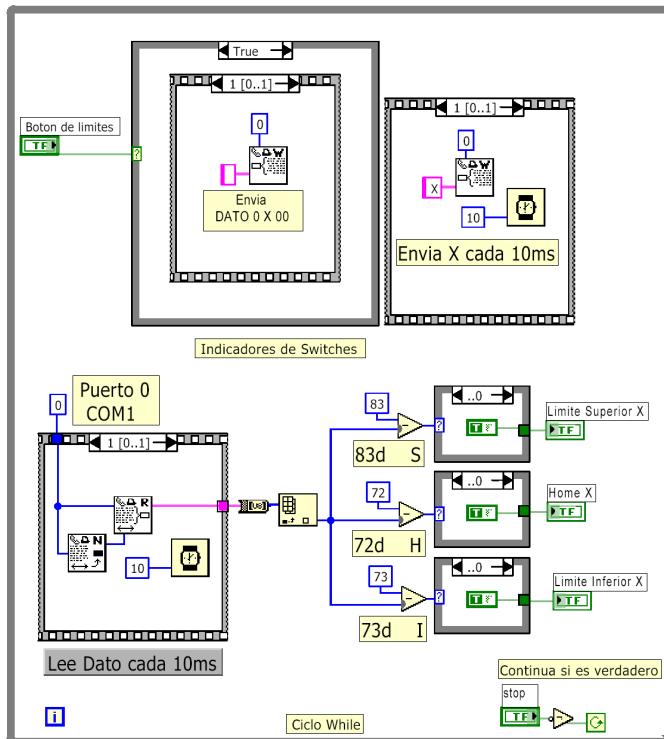


Figura 2-19 Activación de indicadores para los switches

Panel frontal de los indicadores de switches limite que se muestran en la figura 2-20.



**Figura 2-20 Panel frontal de los indicadores de switches**

## 2.5.6 Recibe un dato activando un indicador y recibiendo otro dato se desactiva

Inicialmente lee datos cada 10ms con la secuencia de estructura en la parte izquierda, de este modo los datos recibidos se van a cada una de las ramas para comparar los datos, si el dato recibido es 108d enciende el Indicador Control Eje X, si recibe un 121d desactiva la variable local Control Eje X y se apaga el indicador, ver figura 2-21.

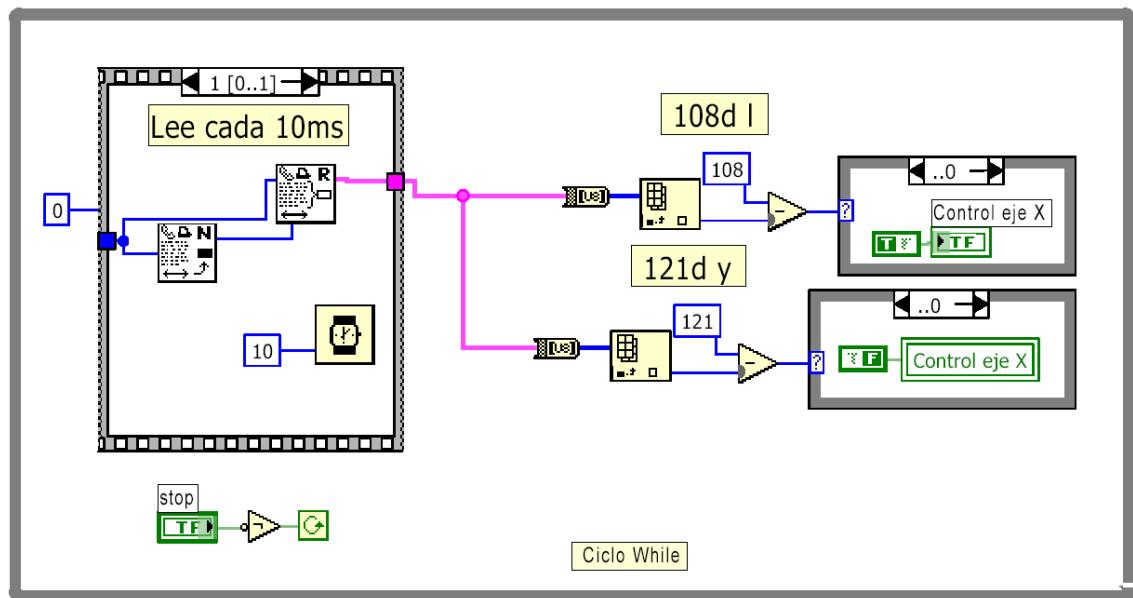


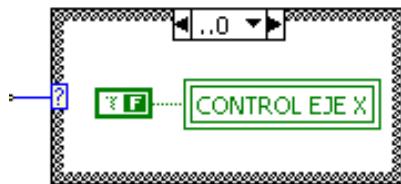
Figura 2-21 Indicador que recibió 108d y lo desactiva si recibe 121d

Cuando el dato recibido es 128d por el puerto serie se hace la conversión y activa la secuencia de estructura para poner en alto el indicador de control de eje X, ver figura 2-29.



Figura 2-22 Secuencia de estructura para el indicador Control eje X

Si el dato recibido es un 128d la resta se hace sero y entra a la secuencia de estructura que contiene, en la cual tiene una variable local de Control eje X, para desactivarla, ya que al indicador le se le asigna un bajo, ver figura 2-23

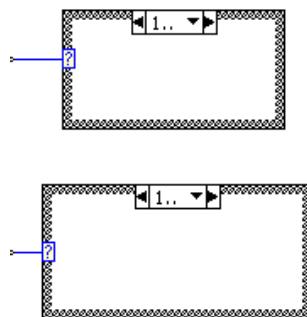


**Figura 2-23 Secuencia de estructura para desactivar indicador Control eje X**

Ya que en el caso de secuencia de estructura es necesario tener por lo menos dos secuencia, para que este activa, cuando el valor igual a uno, esta no realiza ninguna tarea por no tener ningún objeto dentro de la secuencia de estructura.

Es posible hacer una rutina, cuando la resta de 1, 2,3... de resultado pero debido a que el proceso se lo mas entendible posible se adopta la forma anterior para realizar todas las comparaciones dentro de cada una de las rutinas necesarias para su satisfactorio funcionamiento.

Secuencia de estructuras sin ningún objeto cuando la resta es 1, ver figura 2-24.



**Figura 2-24 Secuencias de estructura cuando el valor de la resta es 1**

Panel frontal del anterior diagrama de bloques representado en la figura 2-25



**Figura 2-25 Lee dato 108d y enciende indicador, si lee 121d se apaga el indicador.**

# III.

## El microcontrolador PIC16F877

### 3.1 Introducción

Desde la invención del circuito integrado, el desarrollo constante de la electrónica digital ha dado lugar a dispositivos cada vez más complejos. Entre ellos los microprocesadores y los microcontroladores, los cuales son básicos en las carreras de ingeniería electrónica.

Se pretende explicar conceptos que ya han sido probados y aceptados, considerando que su vigencia se mantendrá en un periodo mas o menos largo. El capítulo se desarrolla alrededor del microcontrolador PIC16F877, mas esta limitación no se considera importante, ya que si se comprende a fondo un microcontrolador, los demás pueden aprenderse con facilidad partiendo de la estructura del primero. Además, el estudio de un microcontrolador particular elimina la posibilidad de una presentación superficial o confusa y permite al usuario enfrentarse a los problemas reales en la práctica.

Para estar al tanto de las innovaciones se recomienda que el usuario se mantenga en contacto con la pagina electrónica de Microchip Technology, <http://www.microchip.com>, solo así podrá obtener un conocimiento completo y actualizado, tanto de los aspectos que aquí se tratan, como de los nuevos que vayan surgiendo.

A diferencia del procesamiento de aplicaciones, tal es el caso de las PC's y Workstations, el control de los elementos mediante computo requiere que existan dispositivos de control dentro de las aplicaciones. Al usuario del producto solo le interesa que es lo que necesita saber para utilizar la interfaz que se le presente (keypads, teclados, comandos), siempre y cuando funcione y/o cumpla con sus expectativas y objetivos.

En muy pocas ocasiones el usuario final conoce (o le interesa conocer) que sistema de control es el que se encuentra incrustado en su aplicación (a diferencia de las personas que adquieren una PC, que se preocupa del tipo de procesador, memoria, velocidad de reloj, etc.).

Así es que, de cualquier forma, es vital para la mayoría de los diseñadores de aplicaciones con elementos de control dentro de ellas, seleccionar los dispositivos y las compañías más indicadas de controladores. Los productos de control incrustado se encuentran en la mayoría de los sectores del mercado: sector comercial, consumibles, periféricos de computadoras, telecomunicaciones (incluyendo los productos de telecomunicación personal de emergencia), automotriz, automotiva e industrial. La mayoría de los productos de control incrustado deben satisfacer requerimientos especiales: eficiencia, bajo costo, baja potencia y un alto nivel de integración en los sistemas.

### 3.2 Definición de PIC

La palabra PIC (***Programmable Integrated Circuits = PIC***) significa Circuito Integrado Programable, y se trata de un microcontrolador, estos dispositivos electrónicos constan de miles de elementos lógicos e interconexiones y están diseñados para responder a las instrucciones de un programa.

Un microcontrolador es un dispositivo digital que acepta o lee datos aplicados a cierto numero de líneas de entrada, los procesa de acuerdo a las instrucciones secuenciales de un programa almacenado en su memoria y suministra o escribe los resultados del proceso en un cierto numero de líneas de salida.

Los datos de entrada pueden provenir de interruptores, sensores, convertidores A/D, teclados, etc. Los datos de salida pueden ser dirigidos a actuadores, displays, pantallas, convertidores D/A, etc. El programa almacenado determina como deben ser procesados los datos de entrada y, en consecuencia, que información debe enviarse a las líneas de salida.

Para un microcontrolador, la función primaria de este es ejecutar programas, sin un programa que le dé vida, un microcontrolador y todo el hardware desarrollado a su alrededor no tendrían un propósito específico. Las funciones realizadas por un microprocesador quedan definidas por un conjunto de instrucciones. El trabajo del programador consiste, precisamente, en elaborar programas, es decir en combinar estas instrucciones, que son relativamente simples, de

una manera lógica con el fin de permitir que el microcontrolador lleve a cabo la tarea final deseada.

La interacción armónica del Hardware y el software es el que confiere a los sistemas basados en su potencia y versatilidad. De esta manera, es posible que un mismo circuito pueda, por ejemplo, pasar de un sistema de alarma a convertirse en un juego de luces con solo cambiar el programa almacenado en su memoria y algunos elementos externos e, incluso, realizar ambas funciones.

Aún cuando son conocidos desde hace más de veinte años, existen en la actualidad nuevos tipos que cumplen con una serie de requisitos y características sumamente útiles.

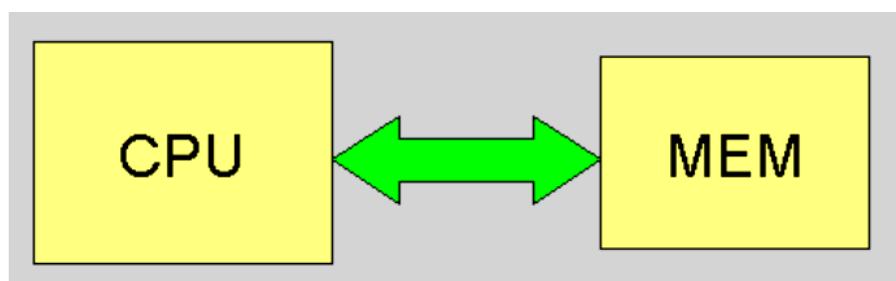
### **3.2.1 La arquitectura tradicional John Von Neumann**

La arquitectura tradicional de computadoras y microprocesadores se basa en el esquema propuesto por John Von Neumann, en el cual la unidad central de proceso, o CPU, esta conectada a una memoria única que contiene las instrucciones del programa y los datos (figura 3-1). El tamaño de la unidad de datos o instrucciones esta fijado por el ancho del bus de la memoria.

Las dos principales limitaciones de esta arquitectura tradicional son:

- a) Que la longitud de las instrucciones esta limitada por la unidad de longitud de los datos, por lo tanto el microprocesador debe hacer varios accesos a memoria para buscar instrucciones complejas.
- b) Que la velocidad de operación (o ancho de banda de operación) esta limitada por el efecto de cuello de botella que significa un bus único para datos e instrucciones que impide superponer ambos tiempos de acceso.

La arquitectura von Neumann permite el diseño de programas con código automodificable, práctica bastante usada en las antiguas computadoras que solo tenían acumulador y pocos modos de direccionamiento, pero innecesaria, en las computadoras modernas.



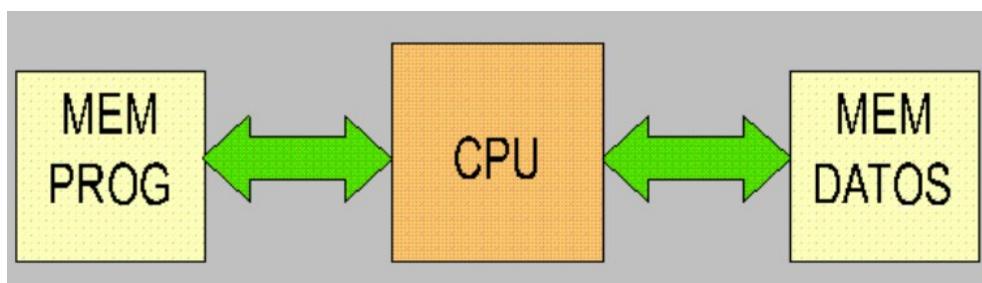
**Figura 3-1 Arquitectura Von Newmann**

### **3.2.2 La arquitectura Harvard**

La arquitectura conocida como Harvard, consiste simplemente en un esquema en el que el CPU esta conectado a dos memorias por intermedio de dos buses separados. Una de las memorias contiene solamente las instrucciones del programa, y es llamada Memoria de Programa. La otra memoria solo almacena los datos y es llamada Memoria de Datos (figura 3-2). Ambos buses son totalmente independientes y pueden ser de distintos anchos. Para un procesador de Set de Instrucciones Reducido, o RISC (Reduced Instrucción Set Computer), el set de instrucciones y el bus de la memoria de programa pueden diseñarse de manera tal que todas las instrucciones tengan una sola posición de memoria de programa de longitud. Además, como los buses son independientes, el CPU puede estar accediendo a los datos para completar la ejecución de una instrucción, y al mismo tiempo estar leyendo la próxima instrucción a ejecutar. Se puede observar claramente que las principales ventajas de esta arquitectura son:

- a) Que el tamaño de las instrucciones no esta relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa.
- b) Que el tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad de operación.

Una pequeña desventaja de los procesadores con arquitectura Harvard, es que deben poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontrarán físicamente en la memoria de programa (por ejemplo en la EPROM de un microprocesador).



## **Figura 3-2 Arquitectura Harvard**

### **3.3 Los microcontroladores PIC16F84 y PIC16F87X**

Hasta el año 2000 han sido muchos los millones de piezas del PIC16F84 que se han vendido en todo el mundo. Este modelo había venido precedido por el PIC16F84, prácticamente igual, con excepción de memoria de programa que era de tipo EEPROM en lugar de FLASH. La memoria FLASH de los PIC soporta hasta 1000 operaciones de escritura/borrado, mediante un proceso totalmente eléctrico que no precisa sacar el microcontrolador del zócalo. La memoria EEPROM para datos que tienen los PIC soporta 100000 operaciones de grabado/borrado. Con ambos tipos de memorias se tenía posibilidad de grabar y borrar eléctricamente programas, lo cual suponía que podía estar insertado en el mismo zócalo durante el proceso de diseño y depuración. Los modelos de microcontroladores programables conocidos disponían de memoria de programa EPROM, que exigía someterla durante cierto tiempo a rayos ultravioleta en el proceso de borrado. Esta operación suponía un gran inconveniente y la posible rotura de las patillas del circuito integrado cuando se realizaban muchos borrados.

Microchip ha tenido disponibles desde hace mucho tiempo microcontroladores muy potentes como los famosos PIC16C73 y 74 de la gama media, pero con memoria EPROM para el programa. Estos dispositivos alcanzaban capacidades de memoria de 4K, con 192 bytes de RAM de datos, con conversor A/D de 8 bits de varios canales de entrada, puertos de comunicación serie ((USART, I2C, SPI), módulos CCP, varios Timer y frecuencia de funcionamiento de 20MHz. Se trata de los PICs muy ricos en recursos.

El PIC16F84 tiene memoria FLASH, pero con capacidad de 1K de palabras, solo tiene un Timer y 16 líneas de E/S digitales y el modelo normal soporta de 10MHz, aunque el A llega a los 20MHz.

Los nuevos PIC16F87X se pueden considerar como una combinación de las virtudes del PIC16F84 con la inclusión de los recursos de los 16C73 y 74. Incorporan la memoria FLASH, con capacidad 4K y 8K palabras de 14bits, sin cambiar la estructura interna del procesador y conservando el mismo repertorio de instrucciones.

La memoria RAM de datos de los PIC16F87X posee una capacidad de 192 bytes en dos modelos y de 2368 bytes en los otros dos. Aunque superan ampliamente los 68 bytes del PIC16F84, mantienen la misma estructura basada en cuatro bancos de 128 bytes cada uno.

La memoria de datos no volátil de 64 bytes tipo EEPROM que tenia el PIC16F84 en los nuevos PIC16F87X de 28 patitas sube hasta 128 bytes, y en los de 40 patitas a 256 bytes.

En los PIC16F87X se manejan hasta 14 posibles fuentes de interrupción y 3 Timer. El número de puertos se ve aumentado, con 3 en los de 28 patitas y hasta 5 en los de 40 patitas. Además, los nuevos PIC incorporan los siguientes recursos inexistentes en el PIC16F84:

- 1) Dos módulos CCP. Son capaces de capturar y comparar impulsos. La captura se efectúa con una precisión de 12.5ns y una resolución de 16 bits, mientras que la comparación con igual resolución alcanza una precisión de 200ns. Además, la sección PWM varía la anchura de los impulsos, técnica muy empleada en el control de motores.
- 2) Comunicación serie. En esta subfamilia se ha potenciado muchísimo el tema de las comunicaciones y en cuanto a la serie tipo admite dos modelos. La típica USART, orientada a la comunicación entre subsistemas o máquinas (RS-232) y la MSSP, destinada a la comunicación entre diversos circuitos integrados y que admiten el protocolo I2C, SPI.
- 3) Comunicación paralelo. En los PIC16F84/7 de 4 patitas está disponible de protocolo PSP, más rápido que la comunicación serie, pero que hipoteca muchas líneas de E/S: ocho del puerto D y tres del puerto E.
- 4) Conversor A/D. En todos los PIC16F87X existe un conversor Analógico/Digital de 10 bits con 5 canales de entrada en los microcontroladores de 28 patitas y 8 en los de 40 patitas.

### 3.4 Estructura del PIC 16F877

#### 3.4.1 Características

El diagrama de pines de estos PIC'S es el siguiente:

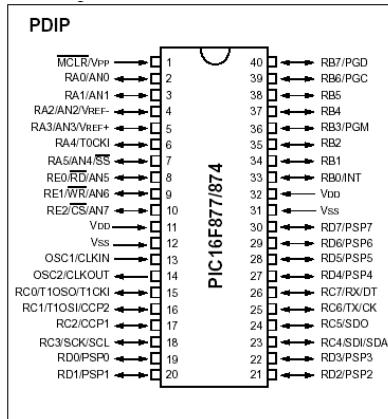


Figura 3-3 Diagrama de pines

Este PIC existe con velocidad de operación de 4 y 20 MHz. Basta con aprender 35 instrucciones básicas para poder programar un PIC y asignarle una tarea específica. Este dispositivo tiene tres tipos de memoria, memoria de programa FLASH, memoria de datos, y una memoria de programa EEPROM., puertos de entrada/salida, un registro de trabajo llamado W, un convertidor analógico-digital de 10 bits, timers entre otros bloques más.

La siguiente tabla muestra algunas características del dispositivo:

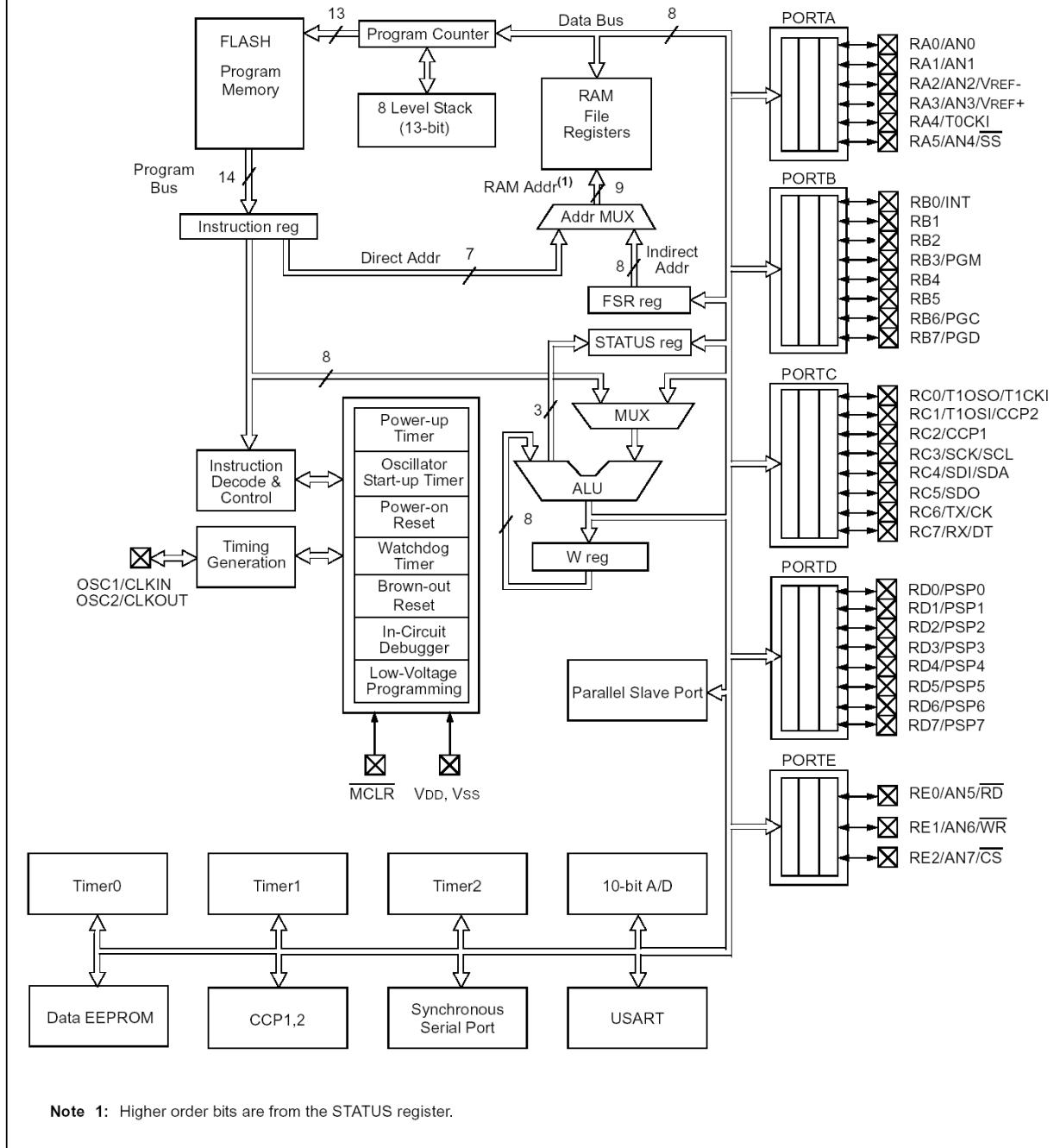
CARACTERÍSTICA/PIC	16F877
Frecuencia de operación	4-20MHZ
Memoria de Programa FLASH (14-bits words)	8 k
Memoria de datos(bytes)	368 bytes
Memoria de datos EEPROM	256 bytes
Interrupciones	14
Puertos I/O	PORTS A,B,C,D,E
Timers	3
Captura/comparación/módulos PWM	2
Comunicación serial	MSSP, USART
Comunicación paralela	PSP
Convertidor A/D de 10-bits	8 canales de entrada

### **Tabla 3-1 Características del 16F877**

#### **3.4.2 Diagrama de bloques**

La figura 3-4 muestra el diagrama a bloques del PIC 16F877 de 40 pines:

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes

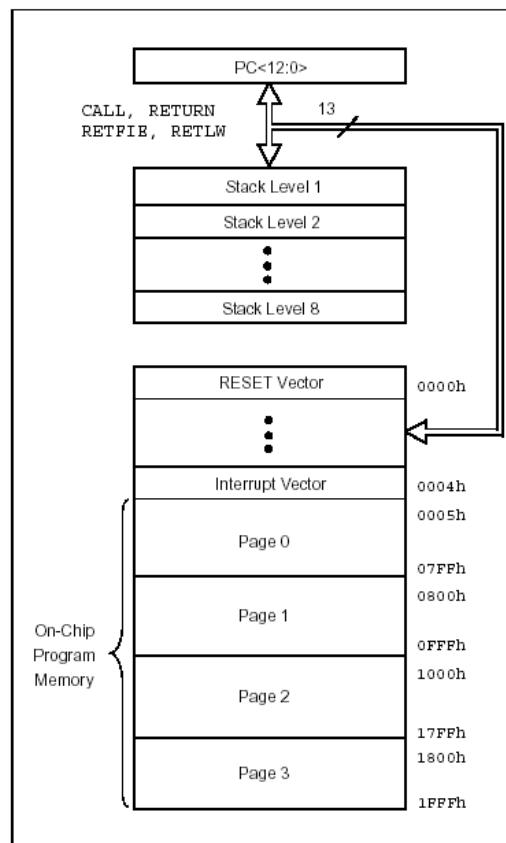


**Figura 3-4 Diagrama a bloques del PIC16F877**

### 3.4.3 Organización de la memoria

Existen tres bloques de memoria en estos dispositivos. La memoria de programa FLASH, EEPROM y la memoria de datos o de registro.

El PIC 16F877 tiene 8k x 14 bits de palabras de memoria programa FLASH. Dicha memoria esta dividida en paginas de 2 K de palabras y esta direccionada con el PC, que tiene una tamaño de 13bits. La pila, que tiene 8 niveles de profundidad, es transparente para el usuario, es decir, funciona automáticamente y no dispone de instrucciones para guardar o sacar de ella información. Con la instrucción CALL y con las interrupciones el valor del PC se salva en el nivel superior. Con las instrucciones RETURN; RETFIE y RETLW el valor contenido en el nivel superior de la pila se carga en PC. Al poseer la pila 8 niveles le corresponde al programador preocuparse por los anidamientos en las subrutinas para no sobrepasar dicho valor. El vector de reset se encuentra en la dirección 0000h y el vector de interrupción de encuentra en la dirección 0004h. Por lo que en la memoria de programa se encuentran reservadas las primeras 5 localidades de memoria y se debe de empezar a programar el PIC a partir de la dirección 0005h.



**Figura 3-5 Organización de la memoria de programa tipo FLASH en el PIC16F877**

La memoria de datos tiene posiciones implementadas en RAM y otras en EEPROM. En la sección RAM, se alojan los registros de operativos fundamentales en el funcionamiento del

procesador y en el manejo de todos sus periféricos, además de registros que el programador puede usar para información de trabajo propia de la aplicación. La RAM estática consta de 4 bancos con 128 bytes cada uno. En las posiciones iniciales de cada banco se ubican registros específicos que gobiernan al procesador y sus recursos.

En la Figura 3-6 se presentan los cuatro bancos de la RAM, indicando las primeras posiciones de cada uno los nombres de los registros que contienen.

Para seleccionar el banco que se desea acceder en la RAM se emplean los bits 6 y 5 del registro de estado, denominados RP1 y RP0 respectivamente, según el código siguiente:

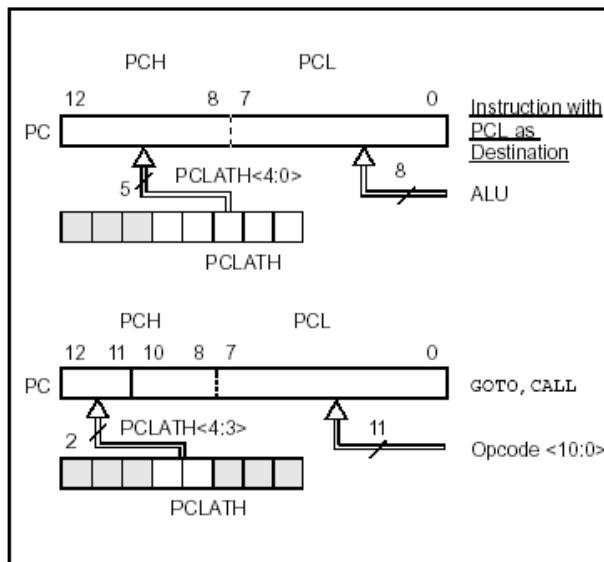
RP1:RP0	Bank
00	0
01	1
10	2
11	3

**Figura 3-6 Bancos y bits de direccionamiento**

#### 3.4.4 Registros específicos para el control de la memoria de programa

Los 13 bits contenidos en el PC, que direccional la memoria de código, están guardados en dos registros específicos. El registro PCL guarda los 8 bits de menos peso y se puede leer y escribir. Los bits <12:8> del PC se aloja en el registro PCH, que al no poder ser leído ni escrito, se accede a el a través del PCLATH.

Las instrucciones de salto CALL y GOTO sólo proporcionan 11bits de la dirección a saltar. Esto limita el salto dentro de cada banco de 2 K. Cuando se desea salir del banco actual hay que programar correctamente los bits PCLATH <4:3> que seleccionan al banco. Es labor del programador modificar bits en las instrucciones (Figura 3-7)



el valor de dichos CALL y GOTO.

**Figura 3-7 En la parte superior se muestra como se carga el PC. Abajo se muestra la carga del PC con las instrucciones CALL y GOTO.**

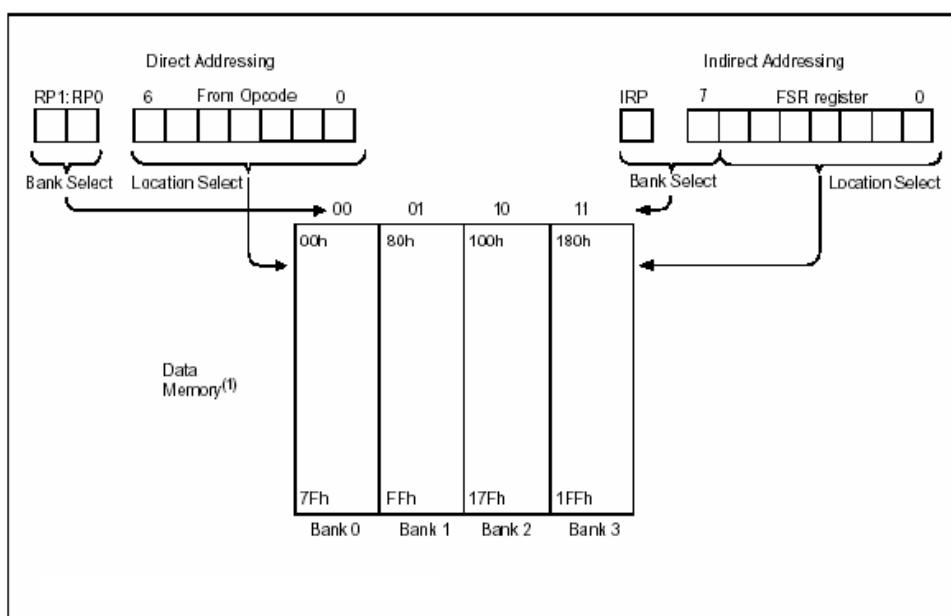
### 3.4.5 Control de memoria de datos

Para direccional la memoria RAM de datos estructurada en 4 bancos de 128 bytes cada uno existen dos modos diferentes:

Direccionamiento indirecto

Direccionamiento directo

En el modo de Direccionamiento directo, los bits RP1 y RP0 del registro estado <6:5> se encargan de seleccionar el banco, mientras que la dirección dentro del banco la determinan 7 bits procedentes del código OP (OPCODE) de la instrucción. Para el direccionamiento indirecto se usa el registro FSR (Feedback Shift Register) vinculado con el bit IRP (Indirect Register Port) del registro estado <7>. Figura 3-8



**Figura 3-8 Formas de seleccionar el banco y la dirección de memoria RAM en los  
direcccionamiento directo e indirecto**

En la siguiente figura 3-9 las posiciones con trama oscura no están implementados físicamente y siempre se leen como 0. Los registros específicos marcados con la nota (1) no se hallan implementados en el PIC16F873, y los que están marcados con (2) son reservados y se mantienen limpios.

File Address	File Address	File Address	File Address
Indirect addr.(*)	00h	Indirect addr.(*)	80h
TMR0	01h	OPTION_REG	81h
PCL	02h	PCL	82h
STATUS	03h	STATUS	83h
FSR	04h	FSR	84h
PORTA	05h	TRISA	85h
PORTB	06h	TRISB	86h
PORTC	07h	TRISC	87h
PORTD <sup>(1)</sup>	08h	TRISD <sup>(1)</sup>	88h
PORTE <sup>(1)</sup>	09h	TRISE <sup>(1)</sup>	89h
PCLATH	0Ah	PCLATH	8Ah
INTCON	0Bh	INTCON	8Bh
PIR1	0Ch	PIE1	8Ch
PIR2	0Dh	PIE2	8Dh
TMR1L	0Eh	PCON	8Eh
TMR1H	0Fh		8Fh
T1CON	10h		90h
TMR2	11h	SSPCON2	91h
T2CON	12h	PR2	92h
SSPBUF	13h	SSPADD	93h
SSPCON	14h	SSPSTAT	94h
CCP1RL	15h		95h
CCP1RH	16h		96h
CCP1CON	17h		97h
RCSTA	18h	TXSTA	98h
TXREG	19h	SPBRG	99h
RCREG	1Ah		9Ah
CCP2RL	1Bh		9Bh
CCP2RH	1Ch		9Ch
CCP2CON	1Dh		9Dh
ADRESH	1Eh	ADRESL	9Eh
ADCON0	1Fh	ADCON1	9Fh
	20h		A0h
General Purpose Register		General Purpose Register	
96 Bytes		96 Bytes	
Bank 0	7Fh	Bank 1	FFh
Bank 2		Bank 3	
File Address		File Address	
Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	101h	OPTION_REG	181h
PCL	102h	PCL	182h
STATUS	103h	STATUS	183h
FSR	104h	FSR	184h
PORTB	105h		185h
TRISB	106h		186h
TRISC	107h		187h
	108h		188h
	109h		189h
PCLATH	10Ah	PCLATH	18Ah
INTCON	10Bh	INTCON	18Bh
EECON1	10Ch	EECON1	18Ch
EECON2	10Dh	EECON2	18Dh
Reserved <sup>(2)</sup>	10Eh	Reserved <sup>(2)</sup>	18Eh
Reserved <sup>(2)</sup>	10Fh	Reserved <sup>(2)</sup>	18Fh
	110h		190h
			1A0h
		accesses 20h-7Fh	
			16Fh
			170h
			17Fh
		accesses A0h - FFh	
			1EFh
			1F0h
			1FFh

**Figura 3-9 Distribución de la memoria RAM en cuatro bancos con 368 bytes útiles**

### **3.5 Instrucciones para programar un PIC**

Con lo anterior es suficiente para comenzar a ver la estructura general de un programa para un PIC, para esto es necesario 35 instrucciones básicas, las cuales se manipularan de acuerdo a las necesidades que se tengan para realizar un programa que haga que el PIC realice una tarea especifica, ver *apéndice D*, donde se muestras las 35 instrucciones con sus respectivos ejemplos.

#### **3.5.1 Instrucciones del código**

Cada instrucción de PIC16FXXX es una palabra de 14 bits dividida en un OPCODE que especifica el tipo de instrucción y uno o más operándos adicionales que especifican la operación de la instrucción. Las instrucciones de PIC16F8XX se muestran en un resumen en la Tabla 3-2 siguiente, existen instrucciones orientadas a bytes, orientadas a bits, literales y operaciones de control. La tabla 3-3 muestra el formato general de las instrucciones.

<b>Campo</b>	<b>Descripción</b>
f	Resgitros de direccion de archivo (0x00 a 0x7F)
W	Registro de trabajo W (acumulador)
b	Direccion de bit dentro de un registro de archivo de 8-bit
K	Campo de literal, nivel o un dato constante
x	No importa localizacion (= 0 o 1) Al ensamblar el codigo se genera con x =0
d	Selección de destino; d = 0 ; guarda el resultado en W. d = 1; guarda el resultado en el registro f Por defecto d = 1
PC	Contador de programa
TO	Tiempo fuera de un bit (Time-out)
PD	Tiempo bajado de un bit (Time-down)

### Tabla 3-2 Descripción de campos de OPCODE

Como ya se menciono las instrucciones estas, agrupadas en tres categorías básicas.

Operaciones orientadas a bytes

Operaciones orientadas a bits

Operaciones a literales o control

En todos los ejemplos, el siguiente formato representa un número hexadecimal. 0xhh donde h significa que es un digito hexadecimal, ver figura 3-10.

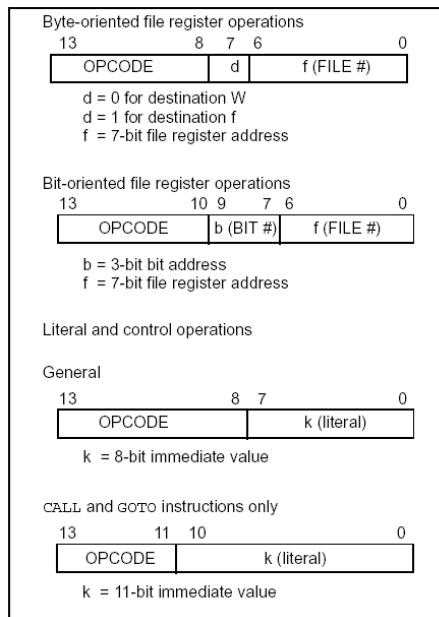


Figura 3-10 Formato general de las instrucciones.

### 3.6 Programación del PIC16F877

Iniciaremos describiendo la estructura necesaria para programar el PIC16F877.

El programa fuente esta compuesto por una sucesión de líneas de programa. Cada línea de programa esta compuesta por 4 campos separados por uno o más espacios o tabulaciones. Estos campos son:

[Etiqueta]	Comando	[Operando(s)]	[;Comentario]
La etiqueta es opcional. El comando puede ser un memónico del conjunto de instrucciones. El operando esta asociado al comando, si no hay comando no hay operando, e inclusive algunos comandos no llevan operando. El comentario es opcional para el compilador aunque es buena práctica considerarlo obligatorio para el programador.			
La etiqueta, es el campo que empieza en la primer posición de la línea. No se pueden insertar espacios o tabulaciones antes de la etiqueta sino será considerado comando. Identifica la línea de programa haciendo que el compilador le asigne un valor automáticamente. Si se trata de una línea cuyo comando es una instrucción de programa del microcontrolador, se le asigna el valor de la dirección de memoria correspondiente a dicha instrucción (location counter). En otros casos se le asigna un valor de una constante, o la dirección de una variable, o será el nombre de una macroinstrucción, etc.			
El comando puede ser un código mnemónico de instrucción del microcontrolador, o una directiva o pseudoinstrucción para el compilador. En el primer caso será directamente traducido a código de maquina, en el segundo caso será interpretado por el compilador y realizara alguna acción en tiempo de compilación como ser asignar un valor a una etiqueta, etc.			
El campo de parámetros puede contener uno o más parámetros separados por comas. Los parámetros dependen de la instrucción o directiva. Pueden ser números o literales que representen constantes o direcciones.			
El campo de comentario debe comenzar con un carácter punto y coma. No necesita tener espacios o tabulaciones separándolo del campo anterior, e incluso puede empezar en la primer posición de la línea. El compilador ignora todo el texto que contenga la línea después de un carácter punto y coma. De esta manera pueden incluirse líneas que contengan solo comentarios, y es muy buena práctica hacer uso y abuso de esta posibilidad para que los programas resulten autodocumentados.			

### 3.6.1 Directivas del programa

En el programa además de las instrucciones propias del lenguaje ensamblador de los microcontroladores PIC presenta una serie de directivas propias de MPLAB y que permiten mejorar la escritura de los programa.

#### Directiva ORG:

Sintaxis: <etiqueta> ORG <expr>

Descripción: Esta directiva especificada en <expr> la dirección de memoria a partir de la que se colocara el código generado. Si se especifica una <etiqueta> se le da el valor de <exp>. Si no hay ningún ORG especificado, la generación del código comienza en la dirección cero.

#### Directiva EQU

Sintaxis: <etiqueta> EQU <expr>

Descripción: Esta directiva permite asignar un valor de <expr> a un identificador <etiqueta>. El resultado puede ser el resultado de una expresión compuesta por otros identificadores y tan compleja como se deseé.

#### Directiva END

Sintaxis: END

#### Descripción:

Esta directiva indica el final del programa y es obligatoria. Si se detecta el fin del fichero y no se ha encontrado la directiva END se produce un error. Todas las líneas posteriores a la línea en la que se encuentra esta directiva, se ignoran y no se ensamblan.

#### Directiva LIST

Sintaxis: LIST <lis\_opción>,.....,<lis\_opción>

#### Descripción:

Esta directiva es la única, es decir, solo puede utilizarse una vez en cada programa y tiene efecto solo sobre el archivo de extensión \*.lst , que es un fichero listable. Además puede ir acompañada de las siguientes opciones, que controlan la estructura del proceso ensamblado o del archivo \*.list, ver tabla 3-3.

Opción	Por defecto	Descripción
b=nnn	8	Espacios de tabulación
c=nnn	132	Fija la anchura de las columnas
f=<format>	INHX8M	Fija el fichero hexadecimal de salida <format> puede ser INHX32, INHX8M, o INHX8S
Free	FIXED	Usa el analizador de formato libre. Suministra la compatibilidad hacia atrás.
Fixed	FIXED	Usa el analizador de formato fijo
Mm=on/off	On	Imprime el mapa de memoria en un fichero tipo listado.
n=nnn	60	Fija las líneas por página.
P=<type>	Ningún tipo	Fija el tipo de procesador.
R=<radix>	Hex	Pone por defecto el RADIX:hex, dec, oct
St=on/off	On	Imprime la tabla de los símbolos en un fichero tipo listado
t=on/off	Off	Corta las líneas de listado (oculta)
w=0 1 2	0	Fija el nivel de mensaje. Ver ERRORLEVEL(nivel de error)
x=on/off	On	Activa o desactiva la expansión de macro

**Tabla 3-3 Opciones de Directiva LIST**

#### Directiva #INCLUDE

Sintaxis:

```
INCLUDE<<include_file>>
INCLUDE"<include_file>"
```

Descripción:

El archivo especificado se lee en como código fuente. El efecto es igual que si el texto entero del archivo INCLUDE se pusiera aquí. Al final del archivo, el código fuente al ensamblar asumirá el archivo fuente original. Se permiten seis niveles de anidamiento. El <include\_file> puede escribirse entre comillas o entre símbolos de mayor que y menor que. Si se especifica totalmente el camino del fichero include, sólo se buscará en dicho. Si no se indica el camino, el orden de la búsqueda es: el directivo activo actual, el directorio de archivo fuente, el directorio ejecutable de MPASM.

Ejemplo

INCLUDE <pic16f877.inc>	;Define el archivo donde están definidos ;todos los registros del pic16f877
INCLUDE " pic16f877.inc"	;También se puede definir de esta forma
INCLUDE "c:\sys\system.inc"	;Define system con su camino
INCLUDE <reg.h>	;Define regs.h

### 3.6.2 Bits de configuración

El PIC16F877 tiene la característica para maximizar la rehabilitación del sistema, minimizando el costo a través de componentes externos, suministrando la salvación de modos de operación. Estos son:

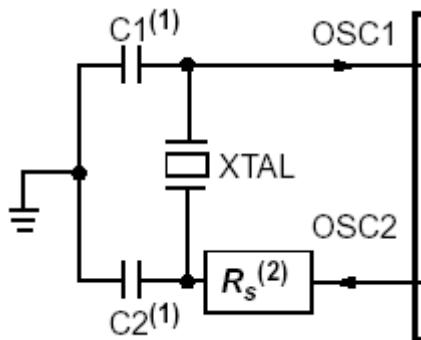
- Selección de oscilador
- RESET
- Watchdog Timer (WDT)
- Power-up Timer (PWRT)
- Brown-out Reset (BOR o BODEN)
- Low Voltage In-Circuit Serial Programming
- Protección de código
- FLASH memoria de programa habilitada para escribir (WRT)
- In-Circuit Debugger

La configuración de bits puede ser programado (leída como '0') o desprogramar (leída como '1'), para seleccionar varias configuraciones de dispositivos, presentadas anteriormente. Para borrar o desprogramar el valor de la palabra de configuración es 3FFFh. Estos bits están en la mapa de localidad de memoria 2007h.

Selección de Oscilador, puede operar cuatro modos diferentes de oscilador, de los cuales se debe de seleccionar uno de los cuatro modos siguientes.

- LP Low Power Crystal
- XT Crystal/Resonator
- HS High Speed Crystal/Resonator
- RC Resistor/Capacitor

En el modo que adaptaremos con un cristal de 20MHz, es el HS debido a que las frecuencias con las que trabaja el resonador son mayores a 8MHz utilizando capacitores cerámicos de 22pF conectados a OSC1 y OSC2 para establecer la oscilación, figura 3-11



**Figura 3-11Conexion del oscilador**

Protección de código, los bits de protección de código, no tienen que ser programados, dentro de la memoria de programa del chip, puede ser leído para propósitos de verificación.

Whatchdog Timer (WDT). El timer del perro guardián esta corriendo libre en la configuración de oscilador RC, el cual no requiere componentes externos. Debido a esto se desactiva ya que para otro tipo de oscilador no es requerido.

Power-up Timer (PWRT). Suministra un tiempo fijo de 72ms después de encender el chip o generar un pulso en el chip con un voltaje que es detectado (en el rango de 1.2 – 1.7V), en el pin MCLR. Es activado.

Brown-out Reset (BOR o BODEN). Cuando se active este bit, el voltaje con que es alimentado el chip cae abajo de 4V por un tiempo de 100us, resetea el dispositivo. Si el voltaje cae dentro de los 100us, el reset no ocurre. Es activado.

Low Voltage In-Circuit Serial Programming (ICSP). Este bit permite al microcontrolador ser programado por vía ICSP usando el voltaje de alimentación del chip, ya que no es activado, no se entra en detalle.

Protección de código, sin habilitar, que el código posiblemente puede ser modificado del que tiene ya grabado en el chip.

FLASH memoria de programa habilitada para escribir (WRT). Cuando le es asignado el bit = 1, la memoria de programa está desprotegida haciendo posible la escritura para el control de EECON, si bit = 0 la memoria de programa está desprotegida no haciendo posible escribir para el control de EECON.

In-Circuit Debugger, cuando el bit DEBUGGER es programado con un '0', se habilita esta función, permitiendo una función simple de depuración cuando se usa MPLAB, con esto es necesario para la programación conexiones al MCLR/Vpp, VDD, GND, RB7 y RB6. Esto es necesario para el tipo de programador con que se cuenta, ya que una de tres para la programación de microcontroladores utiliza esta configuración.

Las siguientes líneas de código son para la programación de bit para el microcontrolador, basado en los temas anteriores, para su configuración apta y satisfactoria.

```
_CONFIG _HS_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON &
_BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_ENABLE_ON & _DEBUG_OFF
```

### **3.6.3 Declaración de variables**

En la programación es necesario variables que contengan información necesaria para su funcionamiento correcto, y debido a que se utilizarán muchas variables se adopta la instrucción

CBLOCK la cual se le indica a partir de cual dirección va iniciar a asignar los valores en este caso 0x20 finalizando con ENC.

```
CBLOCK 0x20
DatoA
DatoB
END
```

### 3.6.4 Inicialización de puertos.

Inicialmente dentro de un programa es necesario la instrucción **org 10** le indica al programador que el programa se inicia en la ubicación 10 de la memoria de programa y que el programa real a partir de ahí.

Es necesario al iniciar cada vez el programa limpiar cada uno de los puertos por posibles datos, que se traslapen, de la con el siguiente código.

```
movlw b'00000000'
movwf PORTA
movlw b'00000000'
movwf PORTB
movlw b'00000000'
movwf PORTC
movlw b'00000000'
movwf PORTD
movlw b'00000000'
movwf PORTE
```

Configuración de puertos cómo entradas o salidas de cada uno de los bits disponibles para cada uno de los puertos. Configurando el TRIS de cada uno de los puertos asignándole el bit=1 hacemos que corresponda a una entrada. Si el bit=0 este corresponde a una salida. De este modo configuraremos cada uno de los puertos con su explicación a la izquierda del código.

```
movlw b'00000000'
```

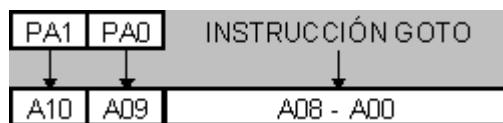
```

movwf TRISA
movlw b'00000000'
movwf TRISB
movlw b'11000000' ; PONE RC6 & RC7 como entradas
movwf TRISC
movlw b'00111111' ; 6 bits de entrada para los limites superior e inferior y home;
; para cada eje.
movwf TRISD
movlw b'11100000' ; Configuracion de puerto E Funcion especial "manejo del
; puerto paralelo"
movwf TRISE ; Bit 4 PSP MODE:
; 1 = Funcion en modo de esclavo para puerto paralelo
; 0 = Funciones para proposito general modo I/O
movlw b'00000111' ; bit 3-0 : 011x Entradas y salidas Digitales
movwf ADCON1 ; puerto A inputs = digital no analogicas

```

### 3.6.5 Direccionamiento de la memoria de programa

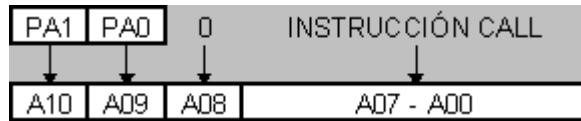
La instrucción GOTO dispone solo de 9 bits en el código de operación para especificar la dirección de destino del salto. Al ejecutar una instrucción GOTO el microprocesador toma los dos bits que restan para completar la dirección de 11 bits, de los bits 5 y 6 de la palabra de estado. Estos últimos son llamados bits de selección de página (PA0 y PA1). El programador deberá asegurarse de que estos dos bits tengan el valor correcto antes de toda instrucción GOTO. Ver figura 3-12



**Figura 3-12 Direccionamiento directo con instrucción GOTO**

Deberá tenerse en cuenta además que es posible avanzar de una página a otra en forma automática cuando el PC se incrementa. Esto ocurre si el programa empieza en una página y sigue en la siguiente. Sin embargo, al incrementarse el PC desde la última posición de una página a la primera de la siguiente, **los bits PA0 y PA1 no se modifican**, y por lo tanto sí se ejecuta una instrucción GOTO, CALL o alguna que actúe sobre el PC, esta producirá un salto a la página anterior, a menos que el programador tenga la precaución de actualizar el valor de dichos bits. Por este motivo es conveniente dividir el programa en módulos o rutinas que estén confinados a una página.

En el caso de la instrucción CALL, el direccionamiento se complica un poco más, ya que la misma solo dispone de 8 bits para especificar la dirección de destino salto. En este caso también se utilizan los mismos bits de selección de página para completar los bits décimo y decimoprimeros de la dirección, pero falta el noveno bit. En estas instrucciones este bit se carga siempre con 0, lo que implica que solo se pueden realizar saltos a subrutina a las mitades inferiores de cada página. En este caso también el programador deberá asegurarse que el estado de los bits PA0 y PA1 sea el correcto al momento de ejecutarse la instrucción. Ver figura 3-13



**Figura 3-13 Direcciónamiento directo con instrucción CALL**

Las instrucciones que operan sobre el PC como registro y alteran su contenido provocando un salto, responden a un mecanismo muy similar al de las instrucciones CALL para la formación de la dirección de destino. En este caso los bits 0 a 7 son el resultado de la instrucción, el bit 8 es 0 y los bits restantes se toman de PA0 y PA1.

Este mecanismo se llama paginado, y a pesar de que representa una complicación bastante molesta para el programador, resulta muy útil ya que permite ampliar la capacidad de direccionamiento de memoria de programa para las instrucciones de salto.

### 3.6.6 Operaciones de comparación

Para las operaciones de comparación usaremos el comando de sustracción.

Tener en cuenta que el comando de sustracción tiene dos bits con aspectos importantes relacionados con el registro STATUS; que nos permitirán saber si la sustracción es positiva, negativa o el resultado es cero.

## Igualdad

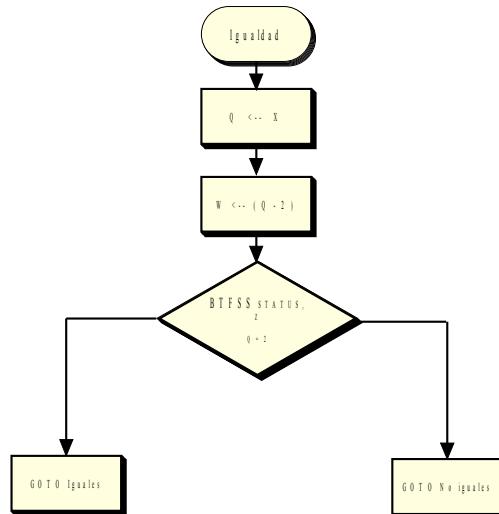
Supongamos que tenemos un variable Q donde recibiremos el dato que compararemos con el valor que tiene el actial registro w que en este cado es 2

```
MOVLW .2  
SUBWF Q, W      ; W = Q - 2  
BTFS $ STATUS, Z  
GOTO No_iguales  
GOTO Iguales
```

Nota las cantidades de la sustracción con guardadas en el registro W y la bandera zero en el registro STATUS es probado, cuando se ejecuta SUBWF o SUBLW.

$$W = Q - W$$

Comparación Igualdad



## Ramificación múltiple

Cuando se tiene que solucionar un diagrama de flujo como el siguiente, en el cual tenemos tres posibles respuestas a una pregunta, se plantean las soluciones aquí presentadas.

Comparando una por una los valores de las diferentes opciones almacenadas en memoria RAM en una variable llamada **OPCION**

```

movlw  Opción1
xorwf  OPCION,0 ;se realiza la verificación del contenido de OPCION con respecto a W
btfsf  STATUS,Z ;Verificando la bandera Z
goto   Acción1
movlw  Opción2
xorwf  OPCION,0 ;se realiza la verificación del contenido de OPCION con respecto a W
btfsf  STATUS,Z ;Verificando la bandera Z
goto   Acción2
movlw  Opción3
xorwf  OPCION,0 ;se realiza la verificación del contenido de OPCION con respecto a W
btfsf  STATUS,Z ;Verificando la bandera Z
goto   Acción3

```

Acción1:

..... ;instrucciones correspondientes a la Acción 1

.....  
.....  
.....  
goto encuentro

Acción2:

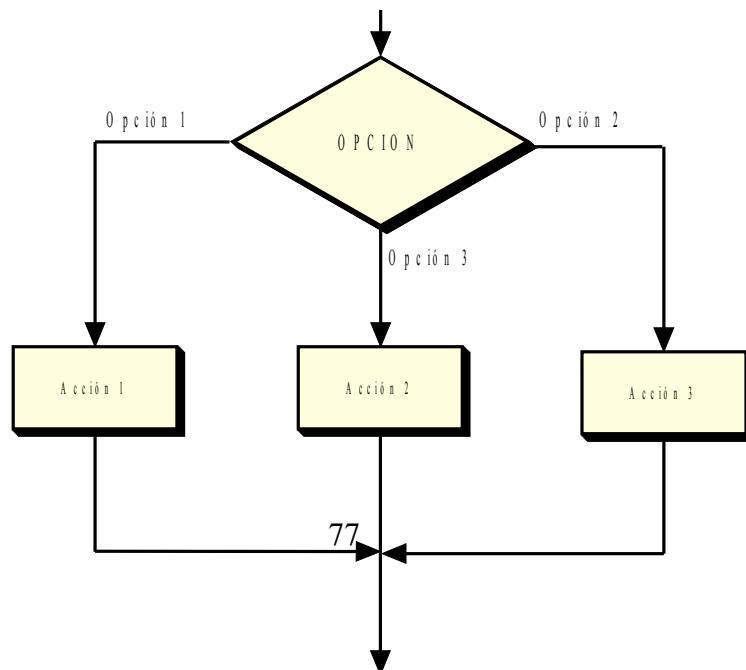
..... ;instrucciones correspondientes a la Acción 2

.....  
.....  
.....  
goto encuentro

Acción3:

..... ;instrucciones correspondientes a la Acción 3

..... R a m i f i c a c i ó n   m u l t i p l e



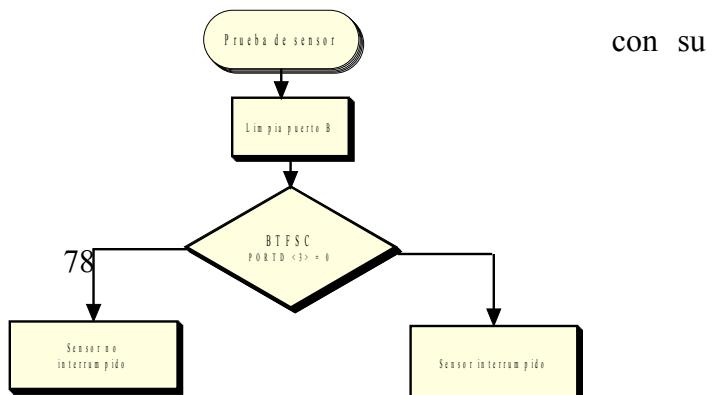
encuentro: ;sitio de encuentro luego de una de las acciones  
..... ;continuación del programa

Aunque este último método es largo, permite que los valores de las diferentes opciones no sean consecutivos entre si.

### 3.6.7 Comparación de bits

Otro de los puntos importantes es el sensado de estado de bits para realizar una determinada función cuando cambia de estados comparando los bits constantemente, mostraremos una rutina respectivo diagrama de flujo para su posterior uso.

Prueba de un sensor en el puerto D bit 3



```
Prueba_sensor
Clrf    portb
Brfsc  portd,3
Goto   Sensor_no_interrumpido
Goto   Sensor_interrumpido
```

### 3.7 Modulación de anchura de pulsos

Los microcontroladores PIC16F87X disponen de dos módulos CCP, llamados CCP1 y CCP2, que son idénticos excepto lo referente a la modalidad de “Disparo especial”, que es utilizado para el modo de captura. Dada esta similitud, la descripción se orienta hacia el modulo CCP1. Estos módulos tienen tres funciones principales:

1. Modo captura: una pareja de registros de un módulo CCPx captura el valor que tiene el TMR1 cuando ocurre un evento especial en la terminal RC2/CCP1 o en la RC1/T1OSI/CCP2
2. Modo de comparación: se compara el valor de 16 bits del TMR1 con otro valor cargado en una pareja de registros de un modulo CCPx y cuando coinciden se produce un evento en la terminal RC2/CCP1 o en la RC1/T1OSI/CCP2
3. Modo modulación de anchura de pulsos (PWM): dentro del intervalo del periodo de un impulso controla la anchura en que la señal vale nivel alto.

El modulo CCP1 utiliza un registro de trabajo de 16 bits que está formado por la concatenación de los registros CCPR1H-CCPR1L (direcciones 16h y 15h). El registro de control del módulo CCP1 es el CCP1CON que ocupa la dirección 17h (Figura 3-14). El modulo CCP2 tiene como registro de trabajo a CCPR2H-CCPR2L (direcciones 1Ch y 1Bh) y como registro de control CCP2CON en la dirección 1Dh. Las parejas de registros son las encargadas de capturar el

valor del TMR1, de comparar el valor que tiene con el TMR1 o, en el modulo PWM, de modular anchura de pulso.

CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS: 17h/1Dh)							
U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0

**Figura 3-14 Asignación de los bits de los registros CCPxCON para los módulos CCP1 y CCP2.**

Todos sus bits son leíbles y escribibles y pasan a 0 cuando se produce un Reset.

#### Configuración de modo de trabajo

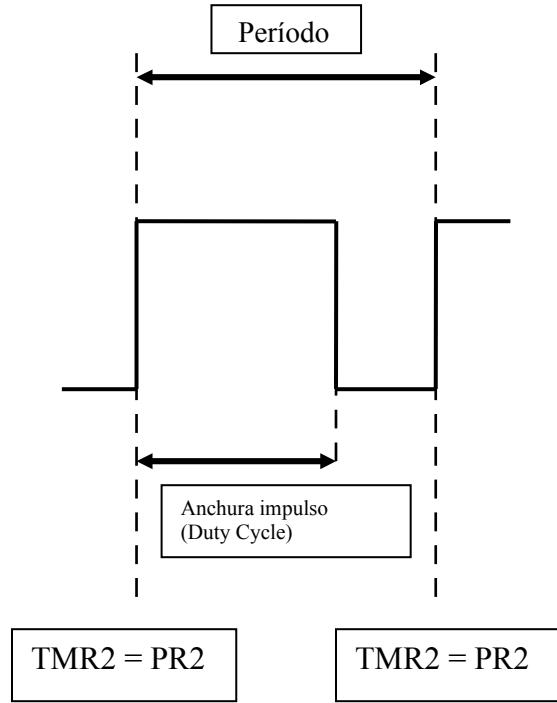
Para la configuración de modo PWM, es necesario poner en alto el bit 3 y 2 del registro con el que se quiera trabajar como se indica a continuación.

#### bit 3-0 CCPxM3:CCPxM0:

11xx = PWM mode

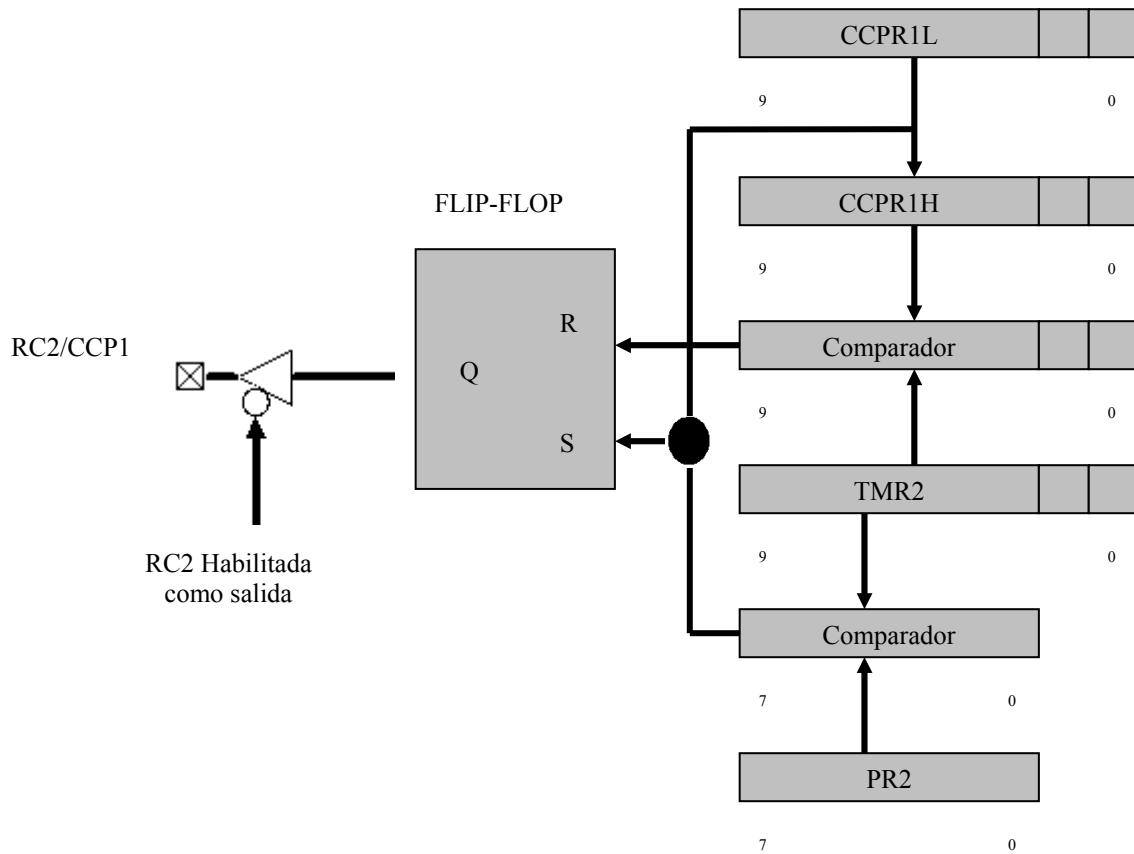
En este modo de trabajo, se consiguen impulsos lógicos cuya anchura del nivel alto es la duración de una variable, que son de enorme aplicación en el control de dispositivos tan populares como los motores y los triacs.

La terminal RC2/CCP1 está configura como salida conmutada entre los niveles lógicos 0 y 1 a intervalos variables de tiempo. Obtendremos un impulso cuyo nivel alto tenga una anchura variable (Duty Cycle) dentro del intervalo del periodo de trabajo. (Figura 3-15)



**Figura 3-15 Salida de PWM**

Para lograr la comutación de la terminal de salida RC2/CCP1 se usa un comparador que pone a 1 (Set) un flip-flop cuando el valor del registro PR2 coincide con la parte alta del TMR2, momento en que el TMR2 toma el valor de 00h. Luego el flip-flop se resetea (se pone en 0) cuando otro comparador detecta la coincidencia del valor existente en CCP1H con la parte alta del TMR2. (Figura 3-16) De esta manera, variando los valores que se cargan en PR2 y en CCPR1L (que luego se traspasa al CCP1H) se varía el intervalo de tiempo en el que la terminal de salida está en 1 y 0.



**Figura 3-16 Estructura interna del módulo CCP1 en modo PWM**

Cuando se trabaja con una presión de 10 bits, los 2 bits CCP1CON<5:4> se concatenan con los 8 bits de CCP1L y, de la misma forma, los 8 bits de más peso del TMR2 se concatenan con los dos bits de menor peso del reloj interno, haciendo que el TMR2 cuente cada Tosc en vez de cada 4\*Tosc

El tiempo que dura el período de la onda depende del valor cargado en PR2, según la fórmula siguiente:

$$\text{Período} = [(PR2) + 1] \bullet 4 \bullet T_{osc} \bullet \text{Valor Preditvisor TMR2}$$

Cuando el valor del TMR2 coincide con el del PR2 suceden tres acontecimientos:

1. Se borra el TMR2
2. La terminal RC2/CCP1 se pone a 1
3. El valor de CCP1L, que es el que determina la anchura del impulso se carga en CCP1H.

El tiempo que la terminal de salida está a nivel alto, que es la anchura del impulso, depende del contenido cargado en CCPR1 y de los bits 5 y 4 del CCP1CON, cuando se trabaja con una precisión de 10 bits

$$Anchuradeimpulso = (CCPR1L : CCP1CON <5:4>) \bullet T_{osc} \bullet Valor\ PredvisorTMR2$$

EL valor CCPR1L:CCP1CON<4:5> puede cargarse en cualquier momento, puesto que el mismo no se traspasa a CCPR1H y se compara hasta que coincidan PR2 con TMR2. En el modo PWM el CCPR1L solo puede ser leído.

Los pasos a seguir para realizar la configuración del modo PWM son los siguientes:

1. Asignar el periodo cargando el oportuno valor PR2.
2. Asignar la anchura del pulso cargando el registro CCPR1L y los dos bits 5 y 4 del CCP1CON.
3. Configurar la línea RC2/CCP1 como salida.
4. Asignar el valor predivisor y activar el TMR2 escribiendo en T2CON.
5. Codificar el Módulo CCP1 en modo PWM.

### **3.7.1 Determinación de frecuencia y configuración de PWM, para el microcontrolador.**

Utilizando una resolución de 10bits con un oscilador de 20MHz de frecuencia, y basándonos en la tabla 3-4 tomada de las hojas de datos del PIC16F87X determinamos la frecuencia de 19KHz para el PWM.

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFFh	0xFFh	0xFFh	0x3Fh	0x1Fh	0x17h
Maximum Resolution (bits)	10	10	10	8	7	5.5

**Tabla 3-4 Frecuencias y resoluciones con cristal de 20MHz**

Configuración de PWM

El PIC corre a 20MHz con un PWM de 19000Hz y suponiendo que requerimos un ciclo útil de 75%, debemos de seguir los siguientes pasos para su configuración optima.

PWM F=19KHz T<sub>pwm</sub>=52.6315us

T<sub>osc</sub> = 1/Fosc = 1/20MHz = 50ps

TMR2 Predivisor = 1

1 Configurar el periodo de PWM escribiendo en el registro PR2.

$$PR2 = (\text{Periodo}/(4 * T_{osc} * \text{TMR2 Predivisor})) - 1$$

$$PR2 = (52.6315\text{us}/(4 * 50\text{ps} * 1)) - 1$$

$$PR2 = 262.1578$$

$$PR2 = 255 = 0xFF$$

Cuando el TMR2 = 255 Finaliza el periodo de PWM

2 Configurando el ciclo util del PWM escribiendo en el registro CCPR1L y los bits CCP1CON <5:4>.

El CCPR1L contiene los 8 bits superiores de los 10bits de valor del ciclo util.

CCP1CON <5:4> contiene los 2 bits inferiores.

WM Duty Cycle = (CCPR1L:CCP1CON<5:4>)\*Tosc\*TMR2 Prescale Value

Ciclo útil de PWM = 75% de periodo

Ciclo útil de PWM = 75% de 52.6ms = 39.4736ms

CCPR1L:CCP1Con<5:4> = PWM Duty Cycle / (Tosc \* TMR2 Predivisor)

CCPR1L:CCP1CON<5:4> = 39.47ms/ (50ps \*1)

CCPR1L:CCP1Con<5:4> = 789.4736 = 789

789 in 10 bit = 1100010101

CCPR1L = 11000101 CCP1Con<5:4> = 01

Escribiendo el valor de 10bits

```
    movlw b'11000101' ; bits 9 - 2  
    movwf CCPR1L  
    bcf CCP1CON,CCP1X ; bit 1  
    bsf CCP1CON,CCP1Y ; bit 0
```

### 3 Configurando el pin CCP1 como salida y limpiando el bit TRSIC<2>

```
    bsf STATUS,RP0  
    movlw b'11111011'  
    andwf TRISC  
    bcf STATUS,RP0
```

### 4 Configurando el valor previsor TMR2 y habilitando el TMR2 para escribir en el T2CON.

El previsor tiene un valor de 1:1

La asignación del previsor TMR2 es con los bits 1 y 0 del T2CON  
para habilitar el TMR2 es necesario poner en el alto el bit 2 del T2CON

```
    movlw b'00000100' ; TMR2 = habilitado, previsor = 1:1  
    movwf T2CON
```

### 5 Configurando el modulo CCP1 en modo PWM

Si hay un valor en el acumulador es necesario hacer una operacion and y una or para  
asegurar la habilitacion del modo PWM en el CCP1

```
    movf CCP1CON,W  
    andlw b'00110000' ; Asegura que los dos bits inferiores del PWM al multiplicarlos por 1  
    iorlw b'00001111' ; y suma esta byte para habilitar el modo PWM  
    movwf CCP1CON
```

De este modo tenemos un PWM de 19000Hz con un ciclo util de 75, para su respectiva  
aplicación.

### 3.8 Comunicación serie asincrona

#### 3.8.1 Generador de Baudios

En el protocolo asíncrono RS-232-C, la frecuencia en baudios (bits por segundo) a la que se realiza la transferencia se debe efectuar a un valor normalizado: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, etc. Para generar esta frecuencia, el USART dispone de un Generador de Frecuencia en Baudios, cuyo valor es controlado por el contenido grabado en el registro SPBRG.

Además del valor X cargado en el registro SPBRG, la frecuencia en baudios del generador del bit BRGH del registro TXSAT<2>. En el caso de que BRGH sea 0 se trabaja a baja velocidad y si BRGH=1 se trabaja en alta velocidad. Según este bit se obtendrá el valor de una constante K necesaria en la determinación de la frecuencia de funcionamiento.

$$\text{Frecuencia en Baudios} = \frac{F_{osc}}{K(X+1)}$$

X es el valor cargado en el registro SPBRG

Si BRGH=0, baja velocidad y K=64

Si BRGH=1, alta velocidad y K=16

BRGH = 1

K = 16

Frecuencia en Baudios = 9600 bps

$$F_{osc} = 20000000 \text{ Hz}$$

X=?

$$X = \frac{F_{osc}}{\text{Frecuencia en Baudios} \bullet K} - 1$$

$$X = \frac{20 \text{ MHz}}{9600 \bullet 16} - 1$$

$$X = 129.2083$$

Si se carga el registro SPBRG con 129 la frecuencia real de trabajo será:

$$\text{Frecuencia en Baudios} = \frac{20\text{MHz}}{16(129+1)}$$

$$\text{Frecuencia en Baudios} = 9615.3846$$

El error que se origina cargando 129 en el SPBRG será:

$$\text{Error} = \frac{9615.3846 - 9600}{9600} = 0.16025\%$$

En la siguiente tabla se muestran los valores que puede tomar el SPBRG con un cristal de 20MHz con la el bit BRGH=1

BAUD RATE (K)	Fosc = 20 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-
1.2	-	-	-
2.4	-	-	-
9.6	9.615	0.16	129
19.2	19.231	0.16	64
28.8	29.070	0.94	42
33.6	33.784	0.55	36
57.6	59.524	3.34	20
HIGH	4.883	-	255
LOW	1250.000	-	0

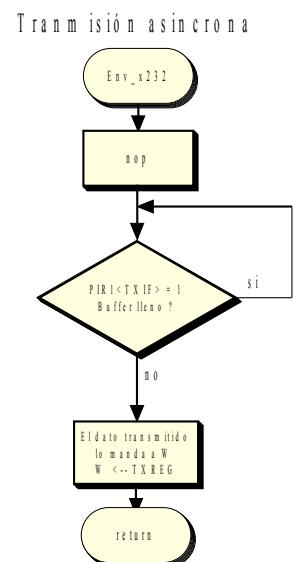
### 3.8.2 Transmisión asíncrona

Con el siguiente código se hace posible, la transmisión de un dato por medio del protocolo RS232, hasta que dicho dato sea transmitido saldrá de la rutina; ver su respectivo diagrama de flujo.

El dato que se debe transmitir por el USART transmisor se deposita en el registro TXREG y a continuación se traspasa al registro de desplazamiento TSR, que va sacando los bits secuencialmente y a la frecuencia establecida. Además, antes de los bits del dato de información incluye un bit de inicio y después de sacar todos los bits añade un bit de parada.

```
; -----
; MANDA EL CARACTER EN W VIA RS232 Y ESPERA HASTA
; QUE TERMINE DE
; ENVIARSE
; -----
```

```
Env_x232    nop
Buferllen    btfss PIR1,TXIF ; PREGUNTA SI EL
                  ; BUFEER xmit ESTA VACIO?
      goto     Buferllen
      movwf   TXREG ; MANDA EL DATO
                  ; ALMACENADO EN W
      return
```



### 3.8.3 Recepción asíncrona

Con el siguiente código se hace posible, la recepción de un dato por medio del protocolo RS232, hasta que dicho dato sea recibido saldrá de la rutina; ver su respectivo diagrama de flujo.

Los datos se reciben en serie, bit a bit, por la patita RC7/RX/DT y se van introduciendo secuencialmente en el registro desplazamiento RSR, que funciona 16 veces más rápida que la de trabajo.

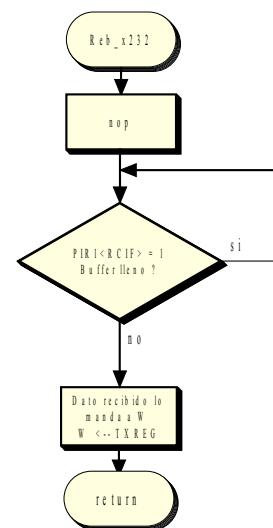
El USART receptor recibe, uno a uno, los bits, elimina los dos de control y los de información una vez que han llenado el registro de desplazamiento RSR los traslada automáticamente al registro RCREG, donde quedan disponibles para su posterior procesamiento.

```

; -----
; RECIBE UN CARACTER DESDE RS232 Y O GUARDA EN W
; -----
; Esta rutina no regresa hasta que un carácter es recibido.
; -----
Reb_x232 nop
    btfss PIR1,RCIF      ; Checa y salta hasta
    recibir un dato
    goto Reb_x232
    movf RCREG,W        ; SALVA EL DATO
    ; EN W
    return

```

Recepción asincrona



### 3.9 MPLAB

EL MPLAB es un “Entorno de Desarrollo Integrado “ (Integrated Development Environment, IDE) que corre en “Windows “, mediante el cual Usted puede desarrollar aplicaciones para los microcontroladores de las familias PIC 16/17.

EL MPLAB le permite a Usted escribir, depurar y optimizar los programas (firmware) de sus diseños con PIC 16/17. EL MPLAB incluye un editor de texto, un simulador y un organizador de proyectos. Además, el MPLAB soporta el emulador PICMASTER y a otras herramientas de desarrollo de Microchip como el PICSTART - Plus.

#### Con el MPLAB Usted puede:

- Depurar sus programas fuente.
- Detectar errores automáticamente en sus programas fuente para editarlos.
- Depurar los programas utilizando puntos de corte (breakpoints) mediante valores de los registros internos.
- Observar el flujo del programa con el simulador MPLAB -SIM, ó seguirlo en tiempo real utilizando el emulador PICMASTER.
- Realizar medidas de tiempo utilizando un cronómetro.
- Mirar variables en las ventanas de observación.
- Encontrar respuestas rápidas a sus preguntas, utilizando la Ayuda en línea del MPLAB.

### **3.9.1 Herramientas de MPLAB**

El organizador de proyectos (Project Manager) es parte fundamental de MPLAB. Sin crear un proyecto Usted no puede realizar depuración simbólica. Con el Organizador de Proyectos (Project manager) puede utilizar las siguientes operaciones:

- Crear un proyecto.
- Agregar un archivo de programa fuente de proyecto.
- Ensamblar o compilar programas fuente.
- Editar programas fuente.
- Reconstruir todos los archivos fuente, o compilar un solo archivo.
- Depurar su programa fuente.

#### **Software ensamblador:**

El software ensamblador que presenta Microchip viene en dos presentaciones, una, para entorno DOS llamado MPASM.EXE y la otra, para entorno Windows llamado MPASMWIN.EXE Las dos presentaciones soportan a TODOS los microcontroladores de la familia PIC de Microchip.

El conjunto de instrucciones de los microcontroladores PIC es en esencia la base del lenguaje ensamblador soportado por este software.

# IV.

## Desarrollo del sistema

### 4.1 Introducción

Los capítulos dos y tres nos dan la información necesaria para el diseño y desarrollo del sistema, debido a la necesidad de realizar un diseño basando esto en los objetivos planteados, es necesario desarrollarlo recabando la información necesario para si mas apta redacción y explicación de cómo se realizo el control por velocidad utilizando LabVIEW y programando el microcontrolador PIC16F877.

### 4.2 Control por velocidad en LabVIEW

#### 4.2.1 Diseño de los Sub VI para el programa de control por velocidad.

El lenguaje de programación G LabVIEW, tiene una gran cantidad de herramientas, por este motivo utilizaremos mayor cantidad de herramientas que nos brinda, como son la construcción de Sub VIs que realizaran determinada función para posteriormente ocuparlos en la implementación del control completo

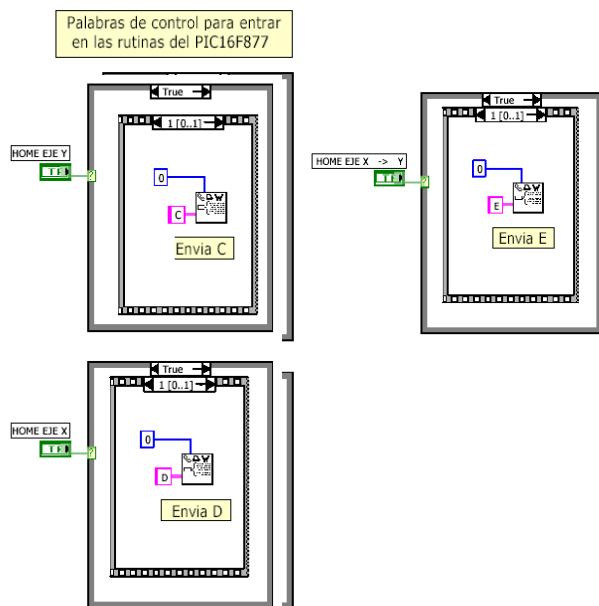
#### 4.2.1.1 Inicialización de rutinas

Tiene como objetivo, tener unos botones para la activación de cada una de las rutinas, para el microcontrolador PIC16F877 las ejecute.

En la figura 4-1 se muestra el diagrama de bloque del Sub VI y el panel frontal en la figura 4-2, llamado inicialización de rutinas con 16F877, que tiene el siguiente icono.



El diagrama de bloques manda un carácter ASCII por medio del puerto serie. En el momento que activamos el respectivo botón que corresponde a la secuencia de estructura el carácter ASCII es transmitido, de este modo nosotros activamos cada una de las rutinas del PIC16F877 programadas previamente.



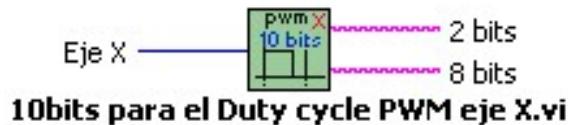
**Figura 4-1 Diagrama de bloques activación de rutinas**



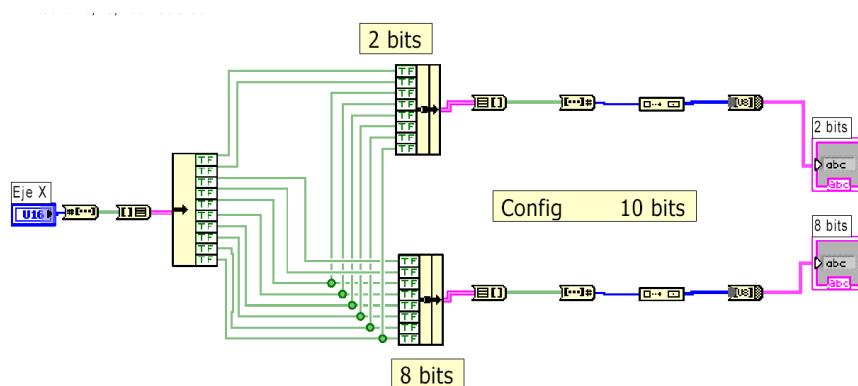
**Figura 4-2 Panel frontal de la activación de rutinas**

#### 4.2.1.2 Selector de 10bits en dos cadenas para configurar el ciclo útil del PWM

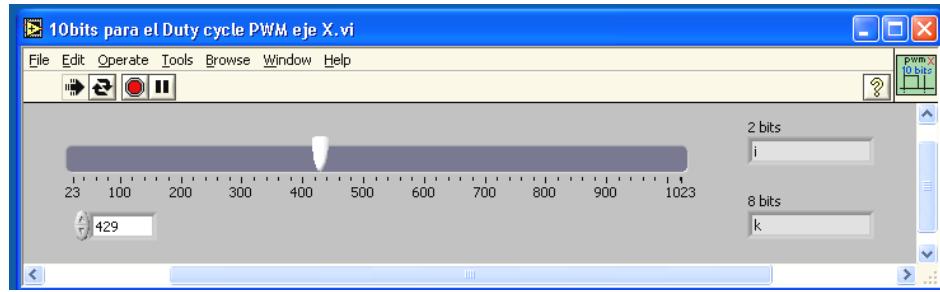
Es necesario para la configuración del ciclo de trabajo del PWM recibir 10 bits, y así tener la resolución completa, el Sub VI llamado 10bits para duty cycle PWM eje X tiene el siguiente icono.



El diagrama de bloques de la figura 4-3, tenemos un selector de 0 a 1023 datos decimales los cuales son convertidos a bits, para posteriormente convertirlos en sus correspondientes cadenas de datos para su transmisión.



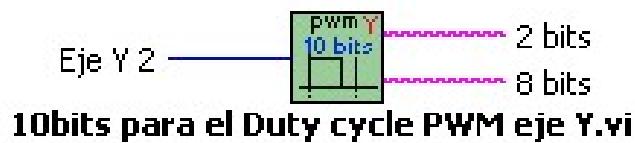
**Figura 4-3 Diagrama de bloques selector de 10bits**



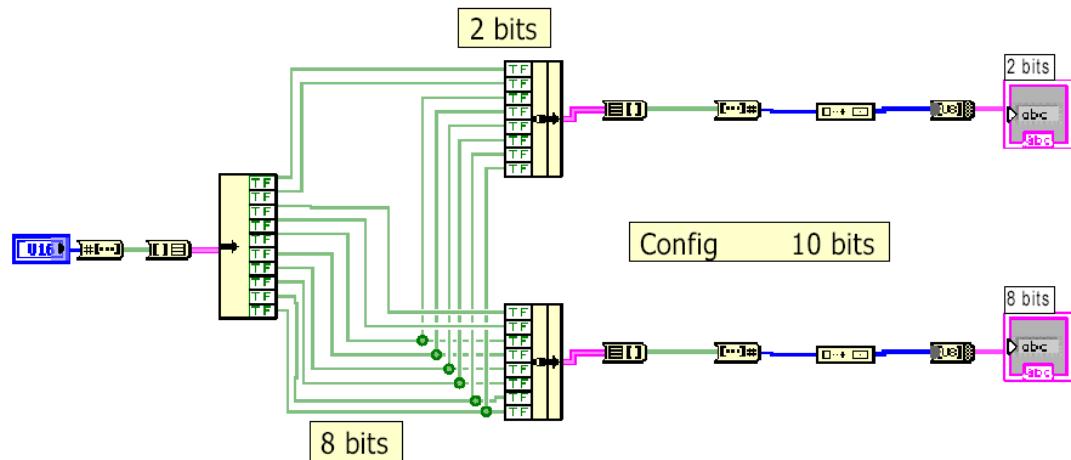
**Figura 4-4 Panel frontal de selector de 10bits para el ciclo útil de PWM**

#### 4.2.1.3 Selector de 10 bits para eje de Y

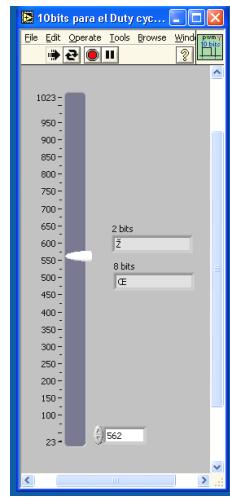
Es un Sub VI similar al anterior, el cambio esta en el selector de números decimales, el cual esta posición vertical, que tiene el siguiente icono.



Presentando el diagrama de bloques en la figura 4-5 y el panel frontal en la figura 4-6



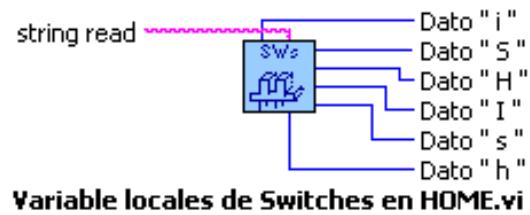
**Figura 4-5 Selector de 10bits para PWM eje Y**



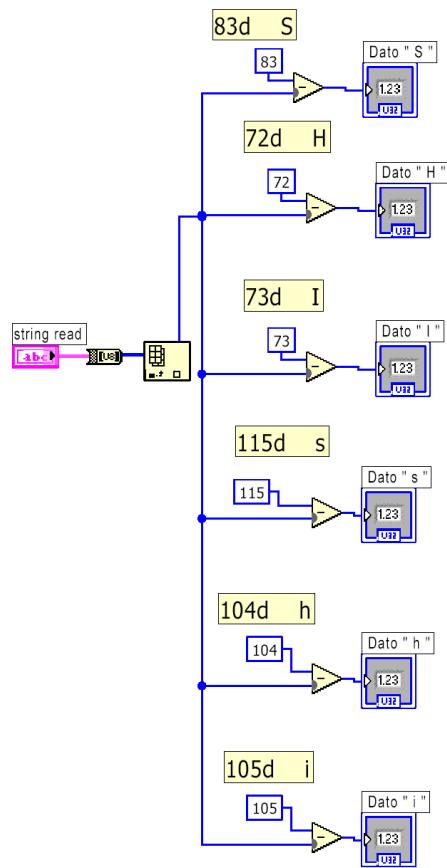
**Figura 4-6 Panel frontal Selector de ciclo útil para PWM eje Y**

#### 4.2.1.4 Indicadores de Switches límite para los dos ejes

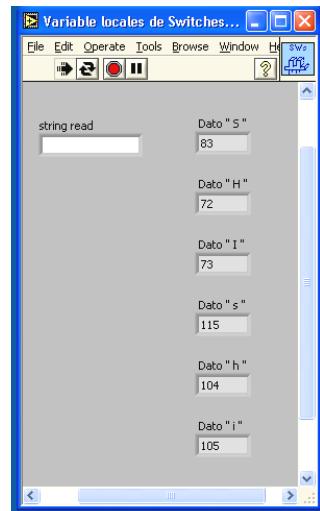
Es necesario sensar el estado de los límites de cada uno de los ejes, por esto se realizo el siguiente Sub VI que tiene el icono que a continuación se muestra.



A partir de la recepción de datos por el puerto serie se compara el dato recibido, de este modo el resultado es un cero. Ver figura 4-7 Diagrama a bloques y figura 4-8 Panel Frontal.



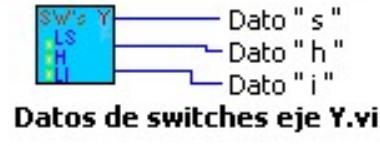
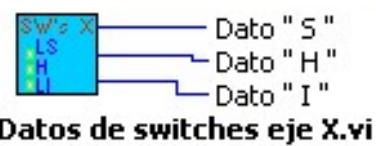
**Figura 4-7 Comparación de datos recibidos para activación de Switches límite**



**Figura 4-8 Panel frontal de datos para activación de Switches límite**

#### 4.2.1.5 Indicadores de Switches limite para cada uno de los ejes.

Los dos siguientes iconos tiene la misma función leer un dato del puerto serie y activar un indicador. Por esta razón explicamos los dos Sub VIs que se presentan a continuación.



Los diagramas de bloques para cada uno de los iconos se presentan en la figura 4-9, donde se reciben datos y envía un cero si el dato recibido es similar a cada una de las restas. El panel frontal para cada uno se muestra en la figura 4-10.

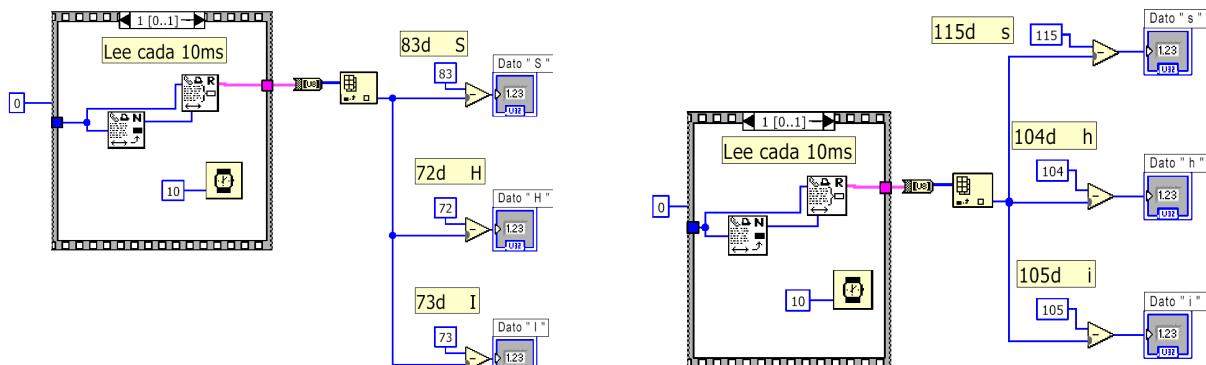


Figura 4-9 Datos de switches X y Y.

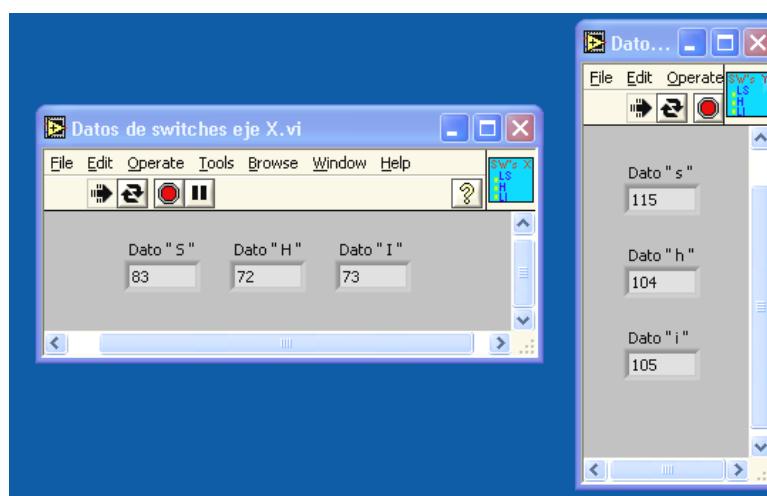
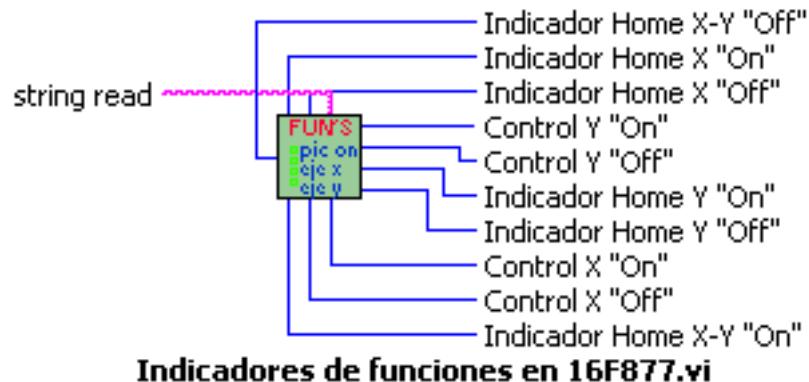


Figura 4-10 Panel frontal de datos de switches X y Y.

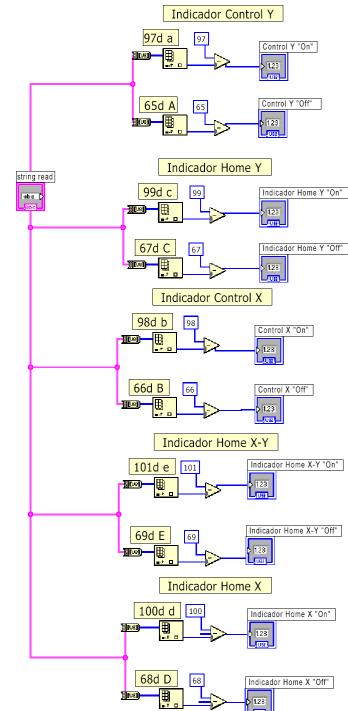
#### 4.2.1.6 Indicadores de funciones en 16F877

El icono que se presenta a continuación define las funciones de este SubVI que es la activación de un indicador, proporcionándonos visualmente la tarea que realiza el microcontrolador PIC16F877.

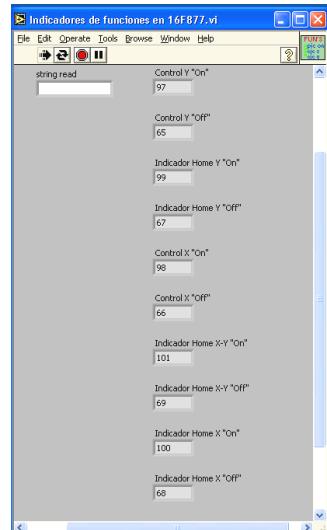


**Indicadores de funciones en 16F877.vi**

El diagrama de bloques se presenta en la figura 4-11, donde se proporciona una lectura de cadena para comparar cada uno de los datos y así activar el correspondiente. Lo que se refiere al panel frontal se muestra en la figura 4-12.



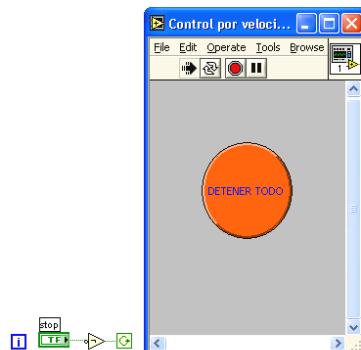
**Figura 4-11 Diagrama de bloque de indicador de funciones en 16F877**



**Figura 4-12 Panel frontal de indicador de funciones en 16F877**

#### 4.2.2 Implementación de los Sub VI para el programa de control por velocidad.

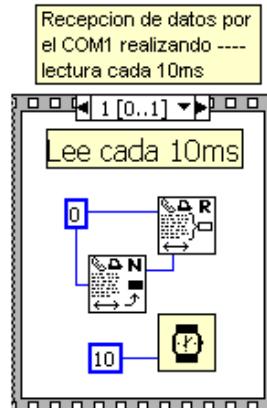
Primeramente crearemos un ciclo while que continua mientras sea verdadero y mantenga la rutina activada hasta que el botón DETENER TODO sea oprimido mandando un falso. Ver figura 1-13



**Figura 4-13 Botón DETENER TODO**

Posteriormente, creamos una la recepción de datos por el COM1

secuencia de estructura para cada 10ms. Figura 4-14

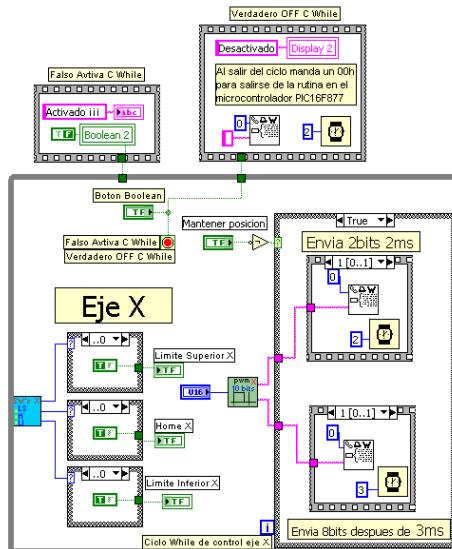


**Figura 4-14 Recepción de datos cada 10ms**

Apoyandonos en la sección 2.5.4, se creara el control de PWM para el eje X y eje Y, utilizando los Sub VIs correspondientes, los cuales se citan en el momento que sean ocupados.

Dentro del ciclo while de control eje X ocupamos el sub VI  para la selección de bits,

tambien ocupamos el subVI  para los indicadores de los switches limite. De este modo hacemos posible la transmisión de datos para la configuración de ciclo util del eje X y la recepcion de datos para los indicadores de switches limite. Ver figura 4-15.



**Figura 4-15 Diagrama de bloques control de PWM e indicadores de switvhes limite eje X**

Ahora se añade un caso de estructura que hace el diagrama de bloques de la figura 4-15 sea activado, utilizando una resta y al salirse del ciclo while manda un 00h para salir de rutina en el microcontrolador PIC16F877, ver figura 4-16. Hasta el momento se tiene solo el control para el eje X, mostrado en la figura 4-17

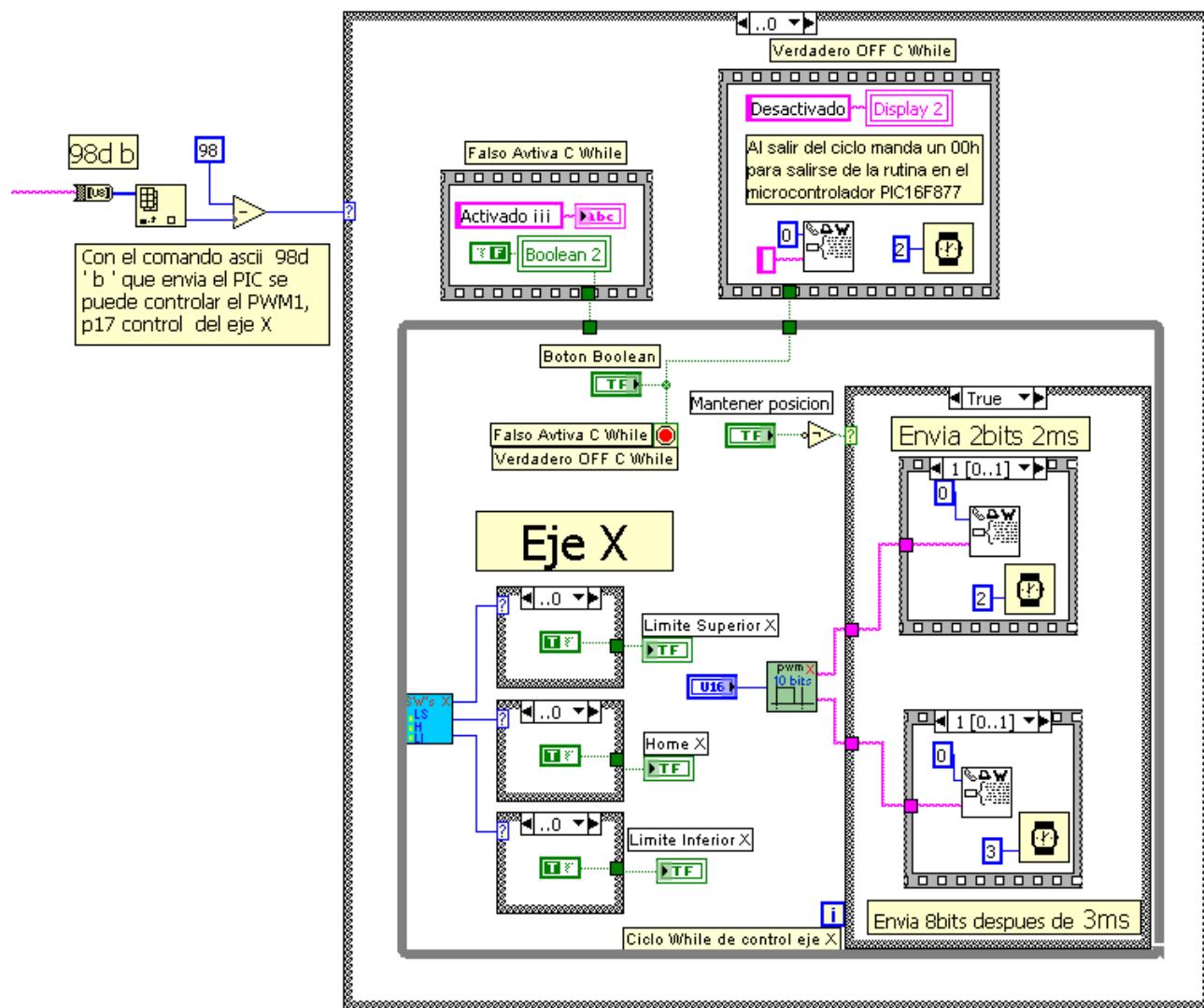
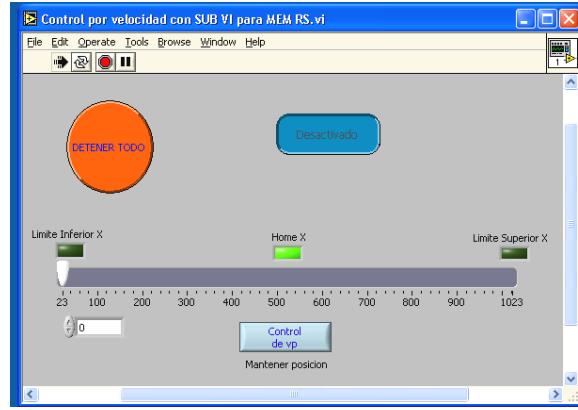
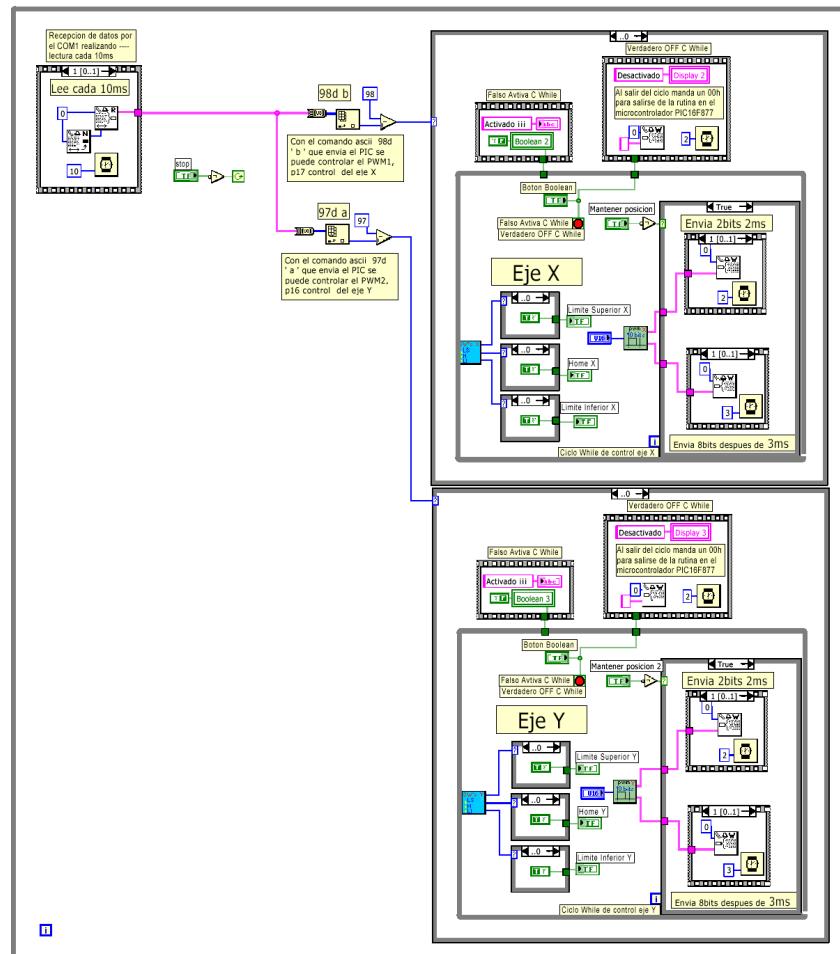


Figura 4-16 Diagrama de bloques del control de eje X

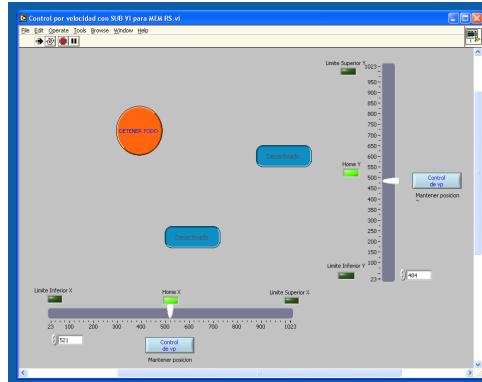


**Figura 4-17 Panel frontal de control del eje X**

Ahora se añadirá el control para el eje Y, el cual es similar, dentro de los cambios en comparación al eje X, están los valores que inicializan los casos de estructura, para el eje X es 98d “b” en ASCII, y para el eje Y es un 97d “a” en ASCII, ver figura 4-18, con su respectivo panel frontal en la figura 4-19.



**Figura 4-18 Control de PWM eje X y Y**

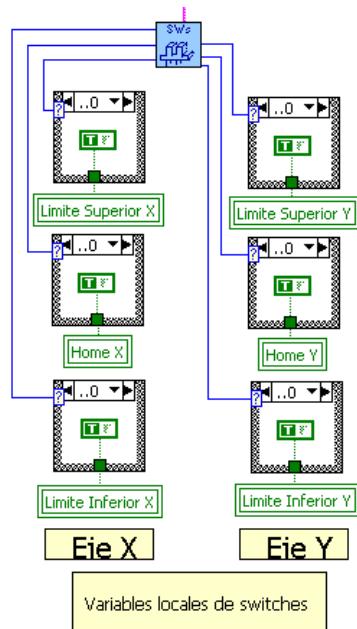


**Figura 4-19 Panel frontal de control de PWM eje X y Y**

Debido a que cada eje tiene sus indicadores de switches límite dentro de casos de estructuras, no es posible sensarlos cuando estamos fuera de la estructura correspondiente, de este modo se utilizan variables locales para los indicadores de cada eje.

La variable local es una copia del mismo objeto, para acceder un valor o actualizar este valor. En otras palabras es que tú puedes leer o escribir en una variable local.

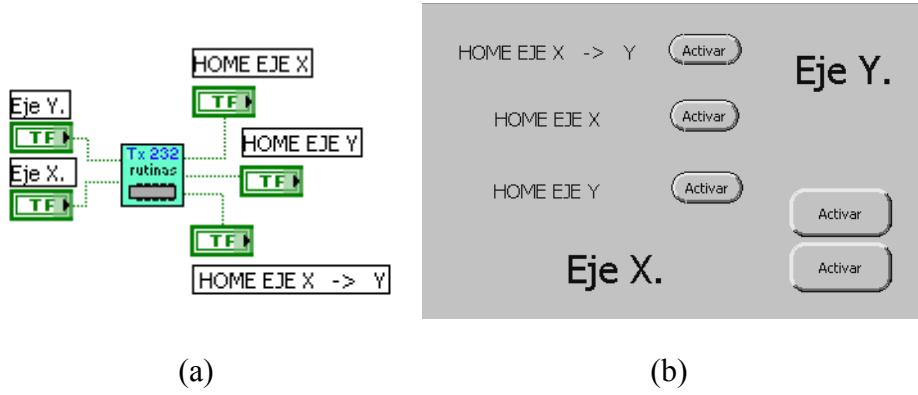
Utilizando el Sub VI de variables locales de switches en Home, realizaremos el sensado de los límites en cualquier momento, dentro de la estructura de caso correspondiente a cada uno de los ejes o fuera de ellos utilizando el siguiente diagrama de bloques, ver figura 4-20.



**Figura 4-20 Diagrama de bloques de variables locales de indicadores de switches**

**límite**

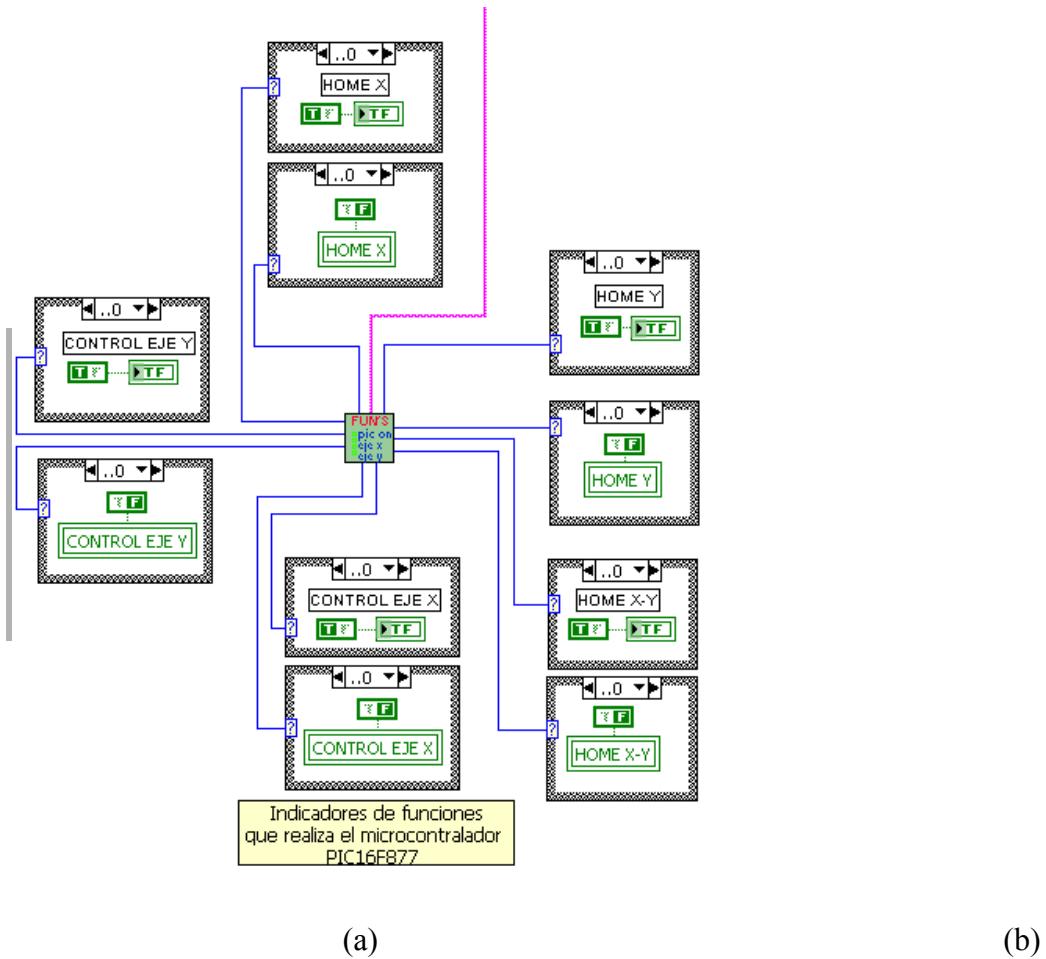
Ahora es necesario añadir el Sub VI  para la inicialización de cada una de las rutinas en el microcontrolador PIC16F877, colocándole los botones necesarios para la activación de cada una de las rutinas, en la figura 4-21 (a) se muestra el diagrama a bloques y la figura 4-21 (b) el panel frontal para la inicialización de cada una de las rutinas.



**Figura 4-21 Inicializadores de rutinas en el microcontrolador PIC16F877**

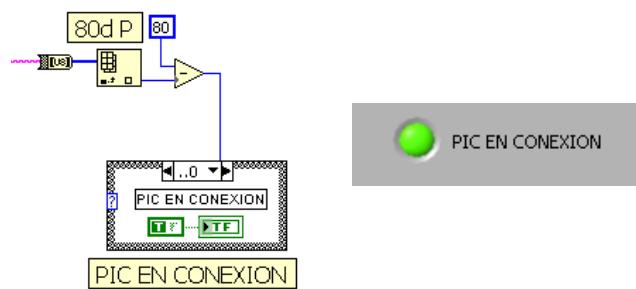
Ahora es necesario tener los indicadores para saber que función esta realizando el

microcontrolador PIC16F877, para eso utilizamos el Sub VI  para activar o desactivar los indicadores, mostrados en la figura 4-22 (b), los cuales están formados por con variables globales y variables locales cada uno de los indicadores como se muestra en el diagrama de bloques de la figura 4-22 (a).



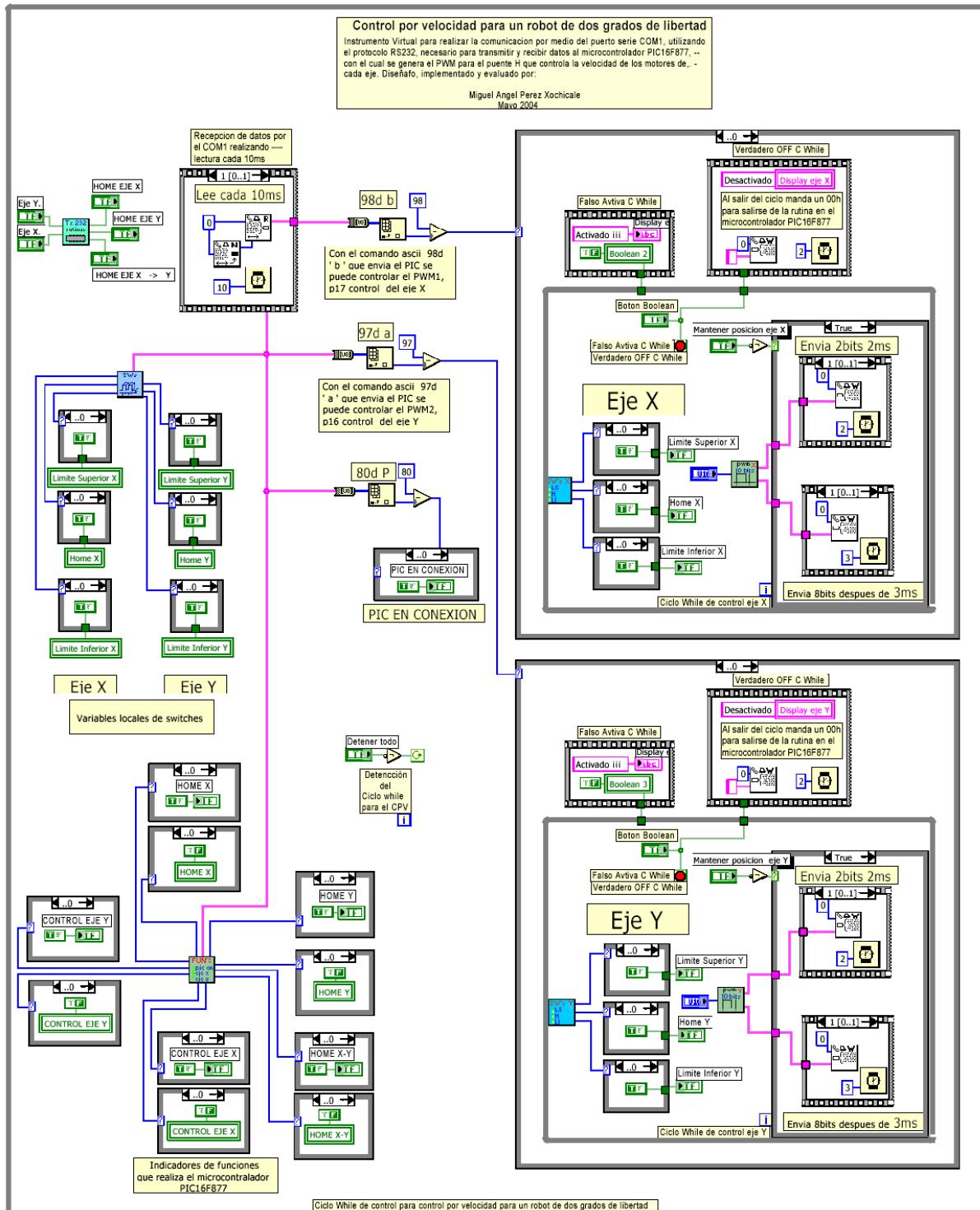
**Figura 4-22 Indicadores de funciones en el microcontrolador PIC16F877**

Posteriormente colocamos un indicador para mostrar que el microcontrolador PIC16F877, esta en condiciones para realizar la comunicación, en la figura 4-23 se muestra el diagrama de bloques utilizado con la apariencia que tienen en el panel frontal



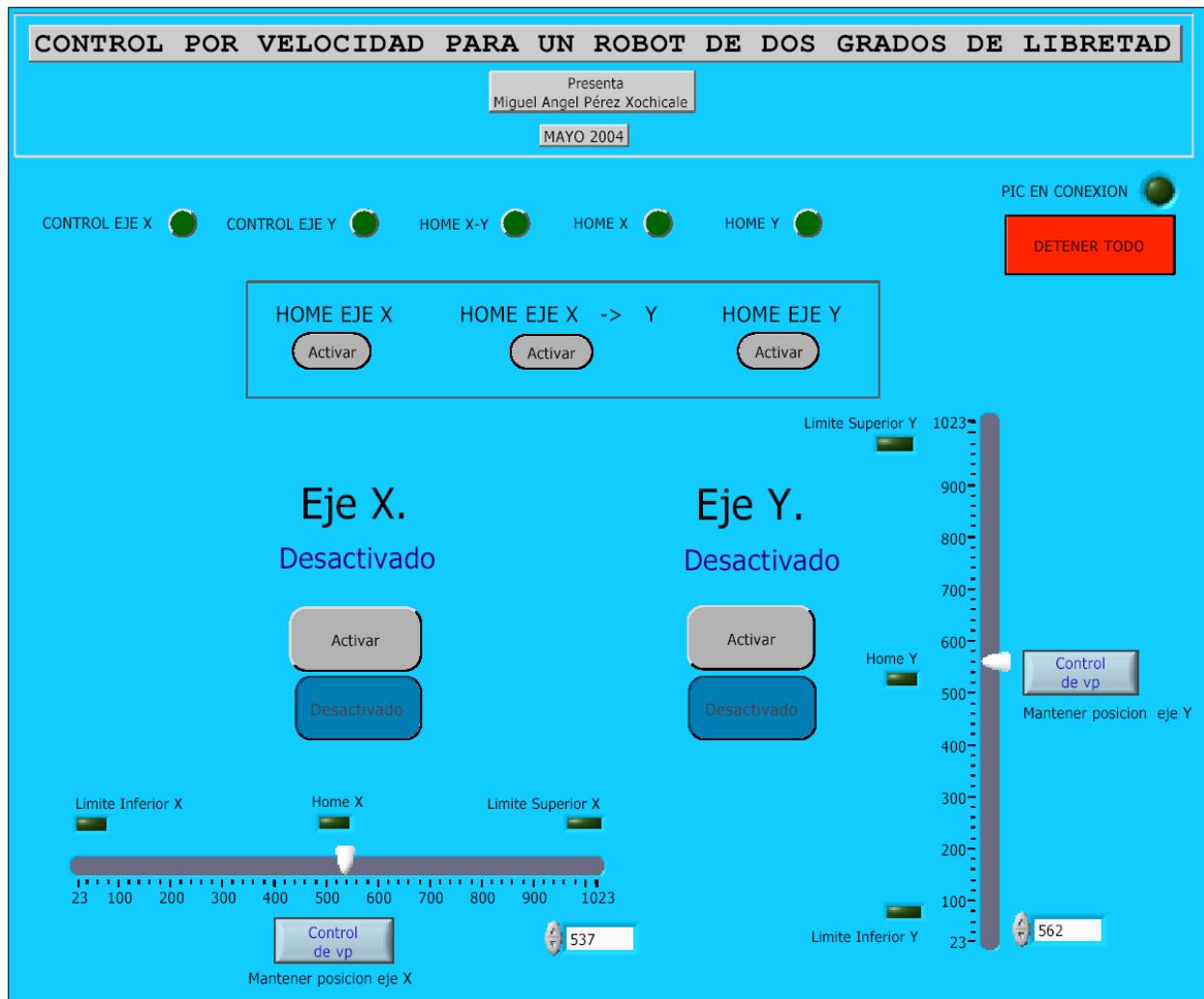
**Figura 4-23 Indicador de PIC en conexión**

Ahora para finalizar la implementación del control por velocidad en LabVIEW presentamos el diagrama de bloques completo, mostrado en la figura 4-24.



**Figura 4-24 Diagrama de bloques de Control por velocidad**

El anterior diagrama de bloques le corresponde el siguiente panel frontal, el cual esta listo para las pruebas de control por velocidad, ver figura 4-25.



**Figura 4-25 Panel frontal Control por velocidad**

## **4.3 Estructura del programa para el control por velocidad con PIC16F877**

### **4.3.1 Configuración del microcontrolador**

En el microcontrolador se utilizó un cristal con una frecuencia de oscilación de 20MHz, debido a que esta frecuencia se obtuvieron los valores de tablas propuestos para el trabajo de PWM.

### **4.3.2 Configuración de comunicación serie.**

Debido a que el Pic cuenta con la funciones necesarias para realizar la trasmisión y recepción serie, al configuración es muy sencilla, solo se debe de tomar en cuenta que la velocidad de comunicación sea la misma con el dispositivo que se piensa comunicarse.

La velocidad adoptada es 19200 bps para el microcontrolador.

### **4.3.3 Diagrama de flujo del programa**

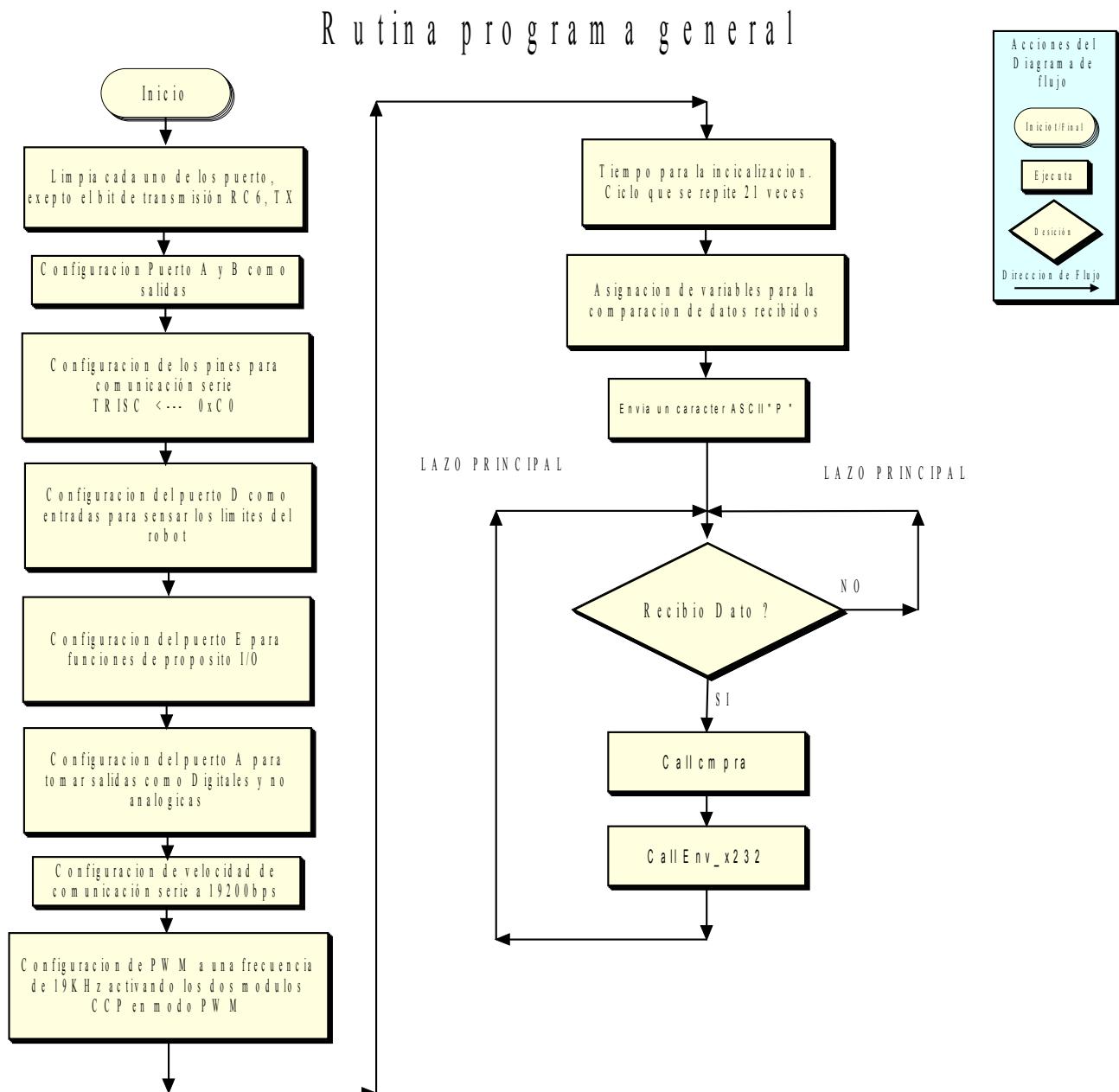
Las señales utilizadas para controlar el ciclo util del PWM son provenientes de programa de control por velocidad hecho en LabVIEW, en el puerto COM1 de la computadora personal.

El pic inicialmente recibe datos los cuales va comparando con otros datos, ya si alguno de estos datos es similar a los estipulados inicia la rutina correspondiente al dato, si ningun dato es similar a los estipulados, repite esta secuencia.

Ya dentro de las rutinas se utilizan diferentes variables para: almacenar bytes para cargar el ciclo útil de PWM ó lectura del estado de los switches limite.

#### 4.3.4 Rutina programa general

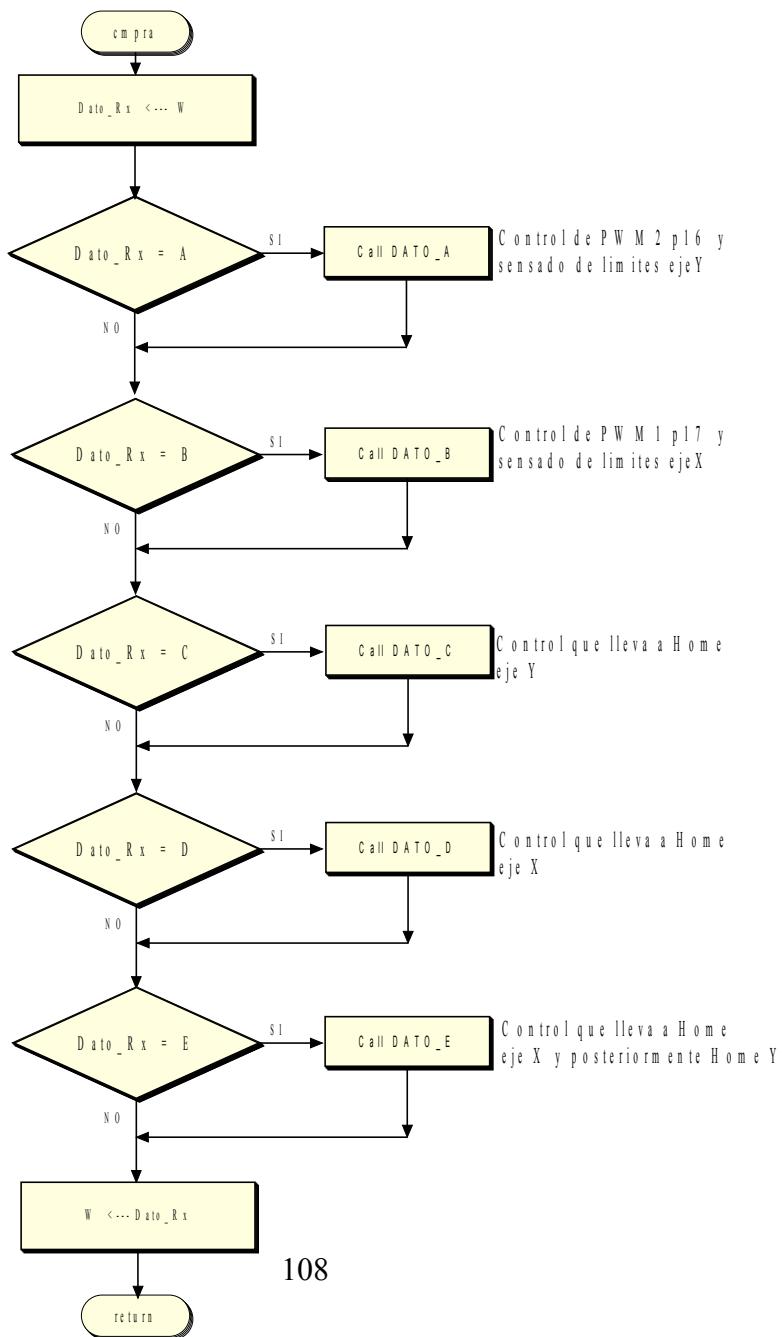
El diagrama de flujo del programa principal se muestra a continuación.



#### 4.3.5 Rutina de comparación

Si en el programa general entra al lazo principal y recibe un dato llama a la rutina comparación, la cual dependiendo al valor que reciba llamará otra rutina para ejecutar una tarea específica, en caso de que ninguno corresponda al dato recibido se sale de la rutina y envia el dato que recibió antes de entrar a la rutina

Rutina programa general



#### **4.3.6 Subrutinas de comparación**

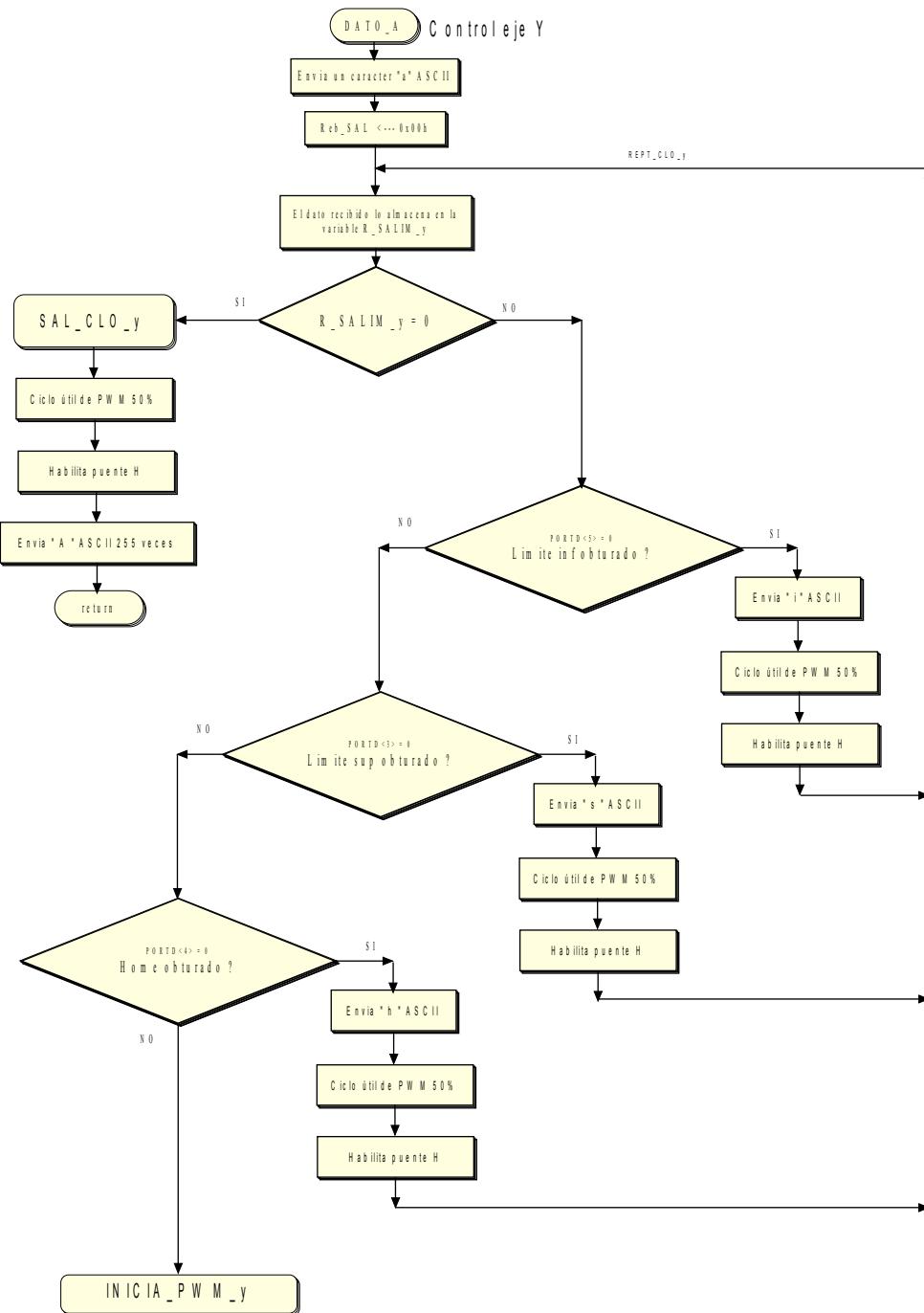
##### **4.3.6.1 Rutina DATO\_A**

Esta rutina es para sensar los límites del eje Y y el control del ciclo útil de PWM.

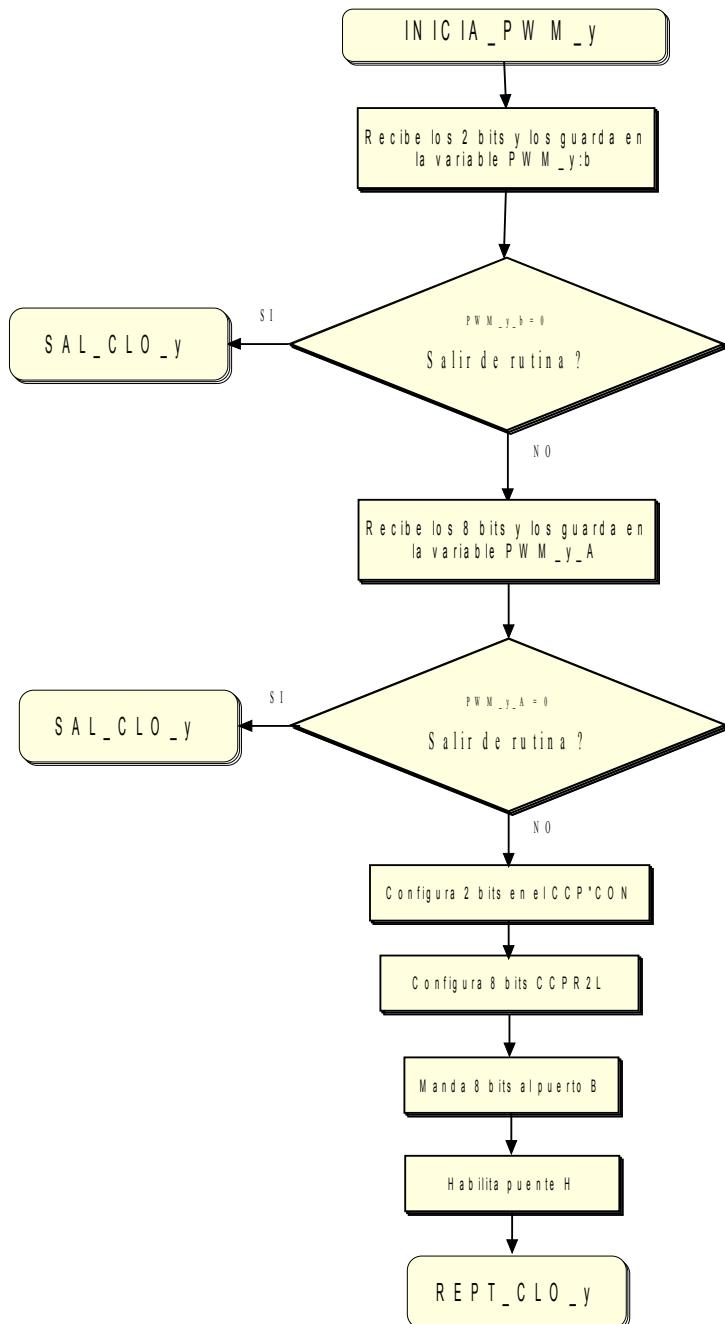
Si son interrumpidos el límite superior, límite inferior o home transmiten un dato por RS232 y configurando el PWM al 50% de ciclo útil mientras es interrumpido de lo contrario la rutina sigue corriendo, al mandar el dato debido a que fue interrumpido un límite, el programa para el control en LabVIEW lo lee y activa un indicador.

Por otro lado para el control de ciclo útil de PWM, al momento de entrar a esta rutina manda un carácter indicando que esta ya controlando el PWM2 del microcontrolador, en este momento inicia trasmitiendo el programa de control en LabVIEW datos para configurar los 10bits de resolución.

## Rutina DATO\_A



## Continuación de Rutina DATO\_A



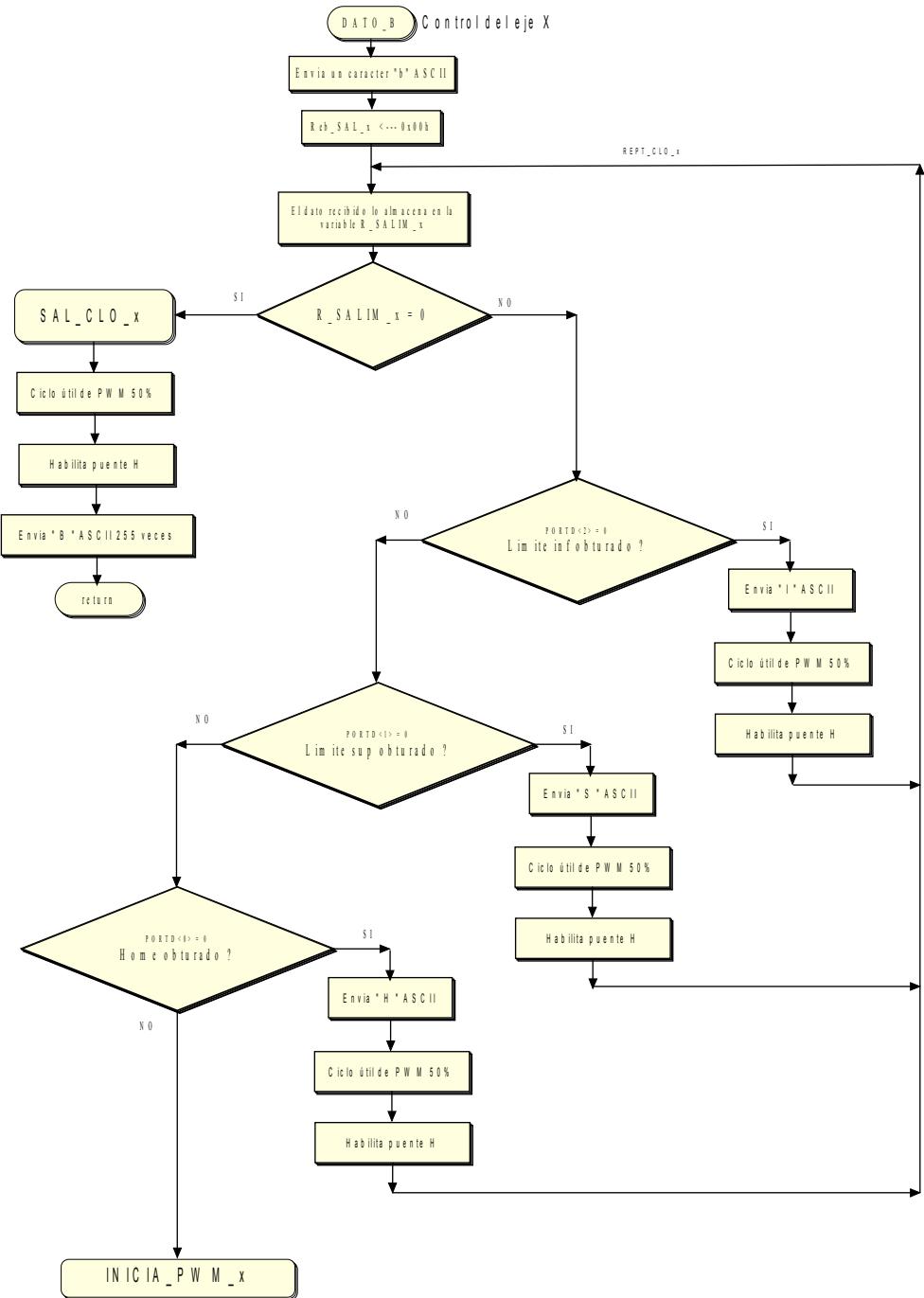
#### **4.3.6.2 Rutina DATO\_B**

Esta rutina es similar a la anterior, el cambio reside en los pines donde se conecta los switches y el registro para controlar el PW1, con esto sensa los limites del eje X y el control del ciclo útil de PWM.

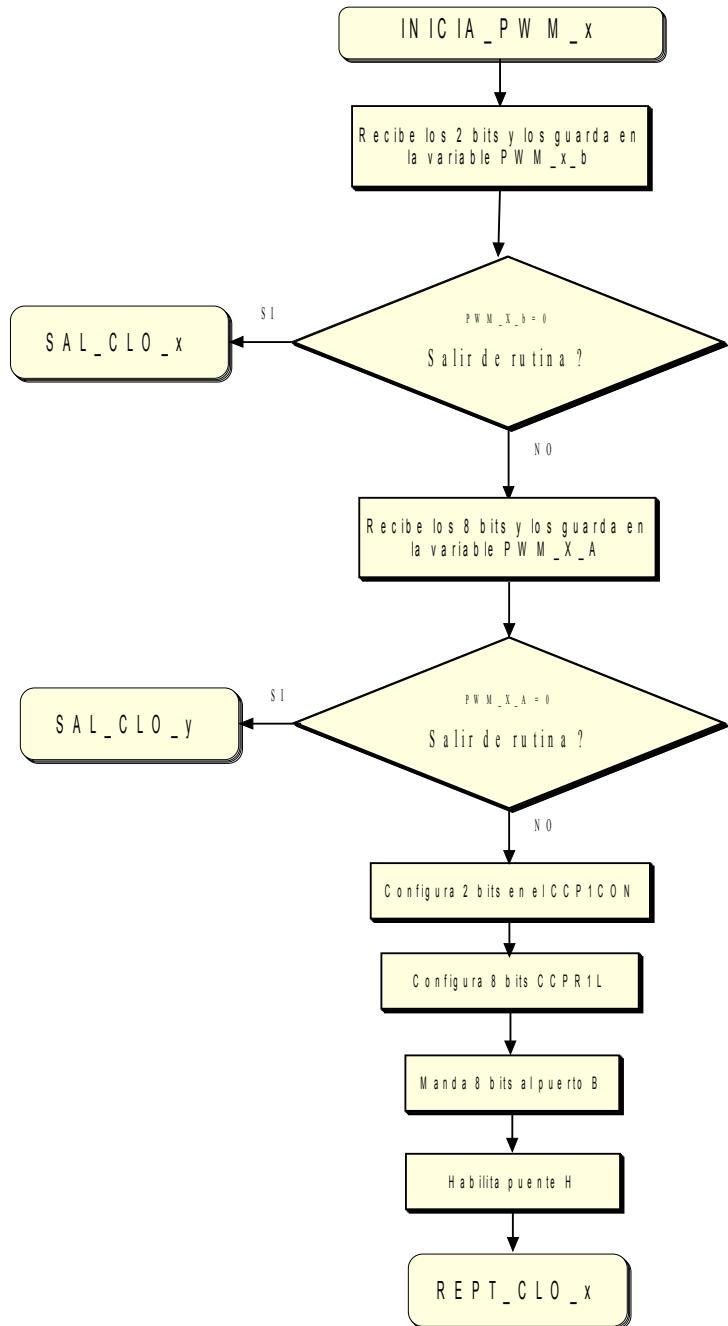
Si son interrumpidos el límite superior, límite inferior o home transmiten un dato por RS232 y configurando el PWM al 50% de ciclo útil mientras es interrumpido de lo contrario la rutina sigue corriendo, al mandar el dato debido a que fue interrumpido un límite, el programa para el control en LabVIEW lo lee y activa un indicador.

Por otro lado para el control de ciclo útil de PWM, al momento de entrar a esta rutina manda un carácter indicando que esta ya controlando el PWM1 del microcontrolador, en este momento inicia trasmitiendo el programa de control en LabVIEW datos para configurar los 10bits de resolución.

## Rutina DATO\_B

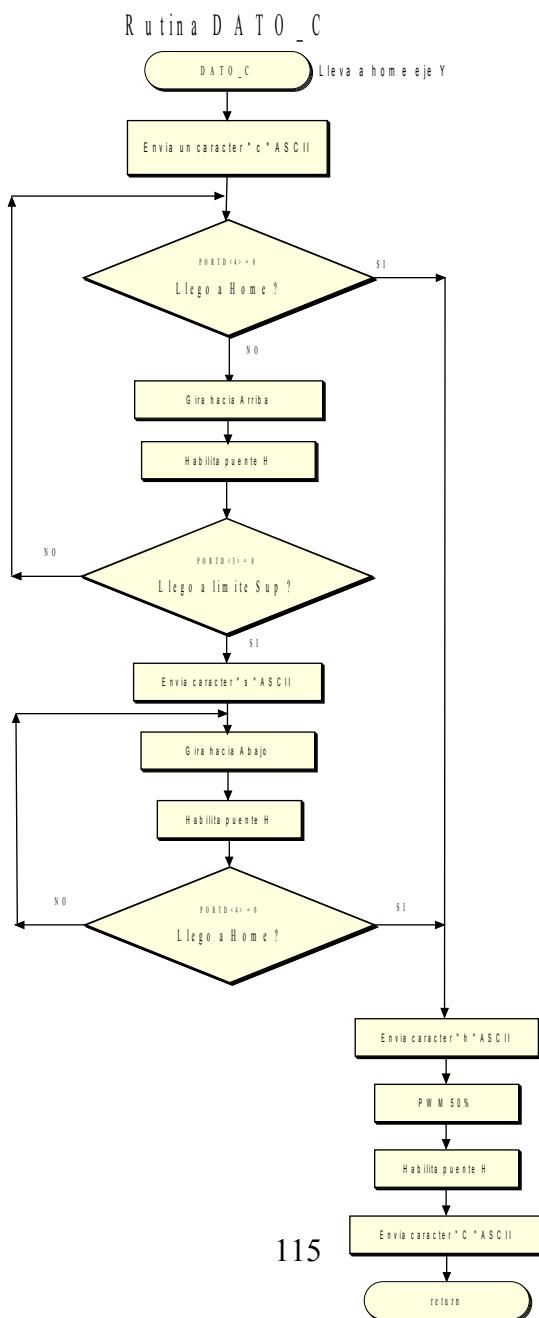


## Continuación de Rutina DATO\_B



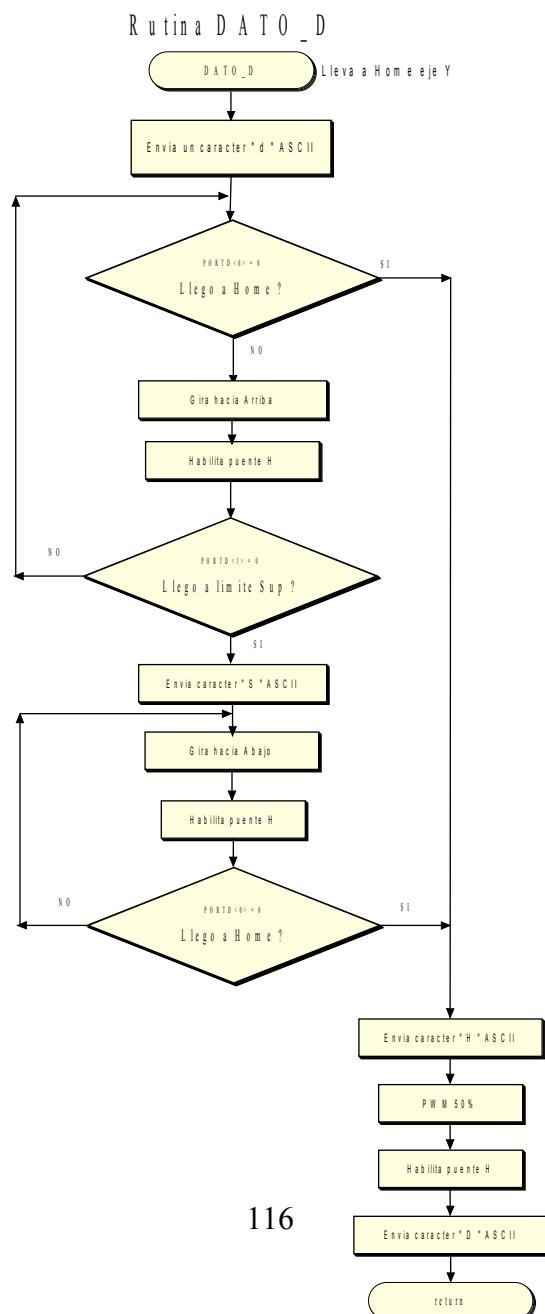
#### 4.3.6.3 Rutina DATO\_C

En esta rutina a partir del estado de los switches que determinan los límites, llevara a la posición home el eje Y, sensandos inicialmente si pasa por home si es así, el ciclo útil se configura a un 50% del PWM, si no sensa este switch, se configura el PWM2 con un 62.5% de ciclo útil para que llegue al límite superior, cuando sensa el límite superior, configura el PWM2 con un ciclo útil de 37.55% para que gire hacia abajo y llegue a la posición home, ver diagrama de flujo para un mayor entendimiento.



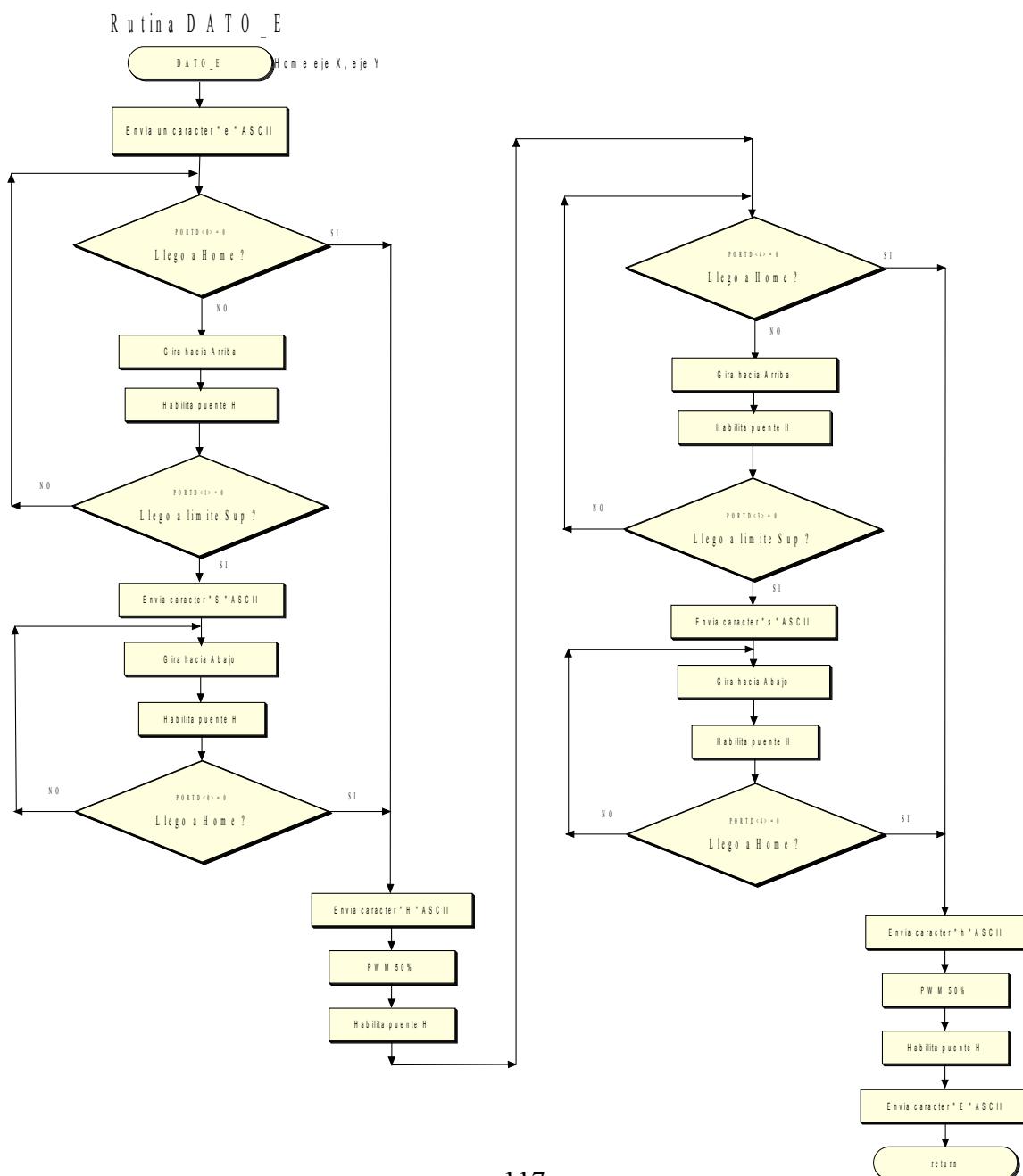
#### 4.3.6.4 Rutina DATO\_D

Similar a la anterior solo que ahora sensa los switches que determinan los límites, lleva a la posición home el eje X, sensandos inicialmente si pasa por home si es así, el ciclo útil se configura a un 50% del PWM, si no sensa este switch, se configura el PWM1 con un 62.5% de ciclo útil para que llegue al límite superior, cuando sensa el límite superior, configura el PWM1 con un ciclo útil de 37.5% para que gire hacia abajo y llegue a la posición home, ver diagrama de flujo para un mayor entendimiento.



#### 4.3.6.5 Rutina DATO\_E

En esta rutina conjunta las dos anteriores primeramente sensa switches para posicionarse en home en el eje X, posteriormente sensa los switches para posicionarse en posición home eje Y, de esta manera los dos ejes se encuentra en su posición home cuando es ejecutada, ver diagrama de flujo a continuación.



#### 4.3.7 Conclusiones

La elaboración del programa fue un complicado debido, principalmente, a los tiempos de comunicación, a la sincronización entre el microcontrolador y LabVIEW. Las subrutinas implementadas en el programa fueron dos principales, de las cuales surgieron cinco para el control de los dos ejes, teniendo funciones diferentes, el nombre de cada una de estas informa mucho sobre el papel que desarrolla dentro del programa. Se cumplieron las metas planteadas previas a la elaboración del control por velocidad, obteniendo un circuito capaz de controlar y sensar los límites del robot para cada uno de los ejes. Gracias al programa para el control en LabVIEW, es posible detectar cada una de las rutinas en las que el PIC ejecuta.

#### 4.3.8 Listado de programa

El programa se puede consultar en el Apéndice E

### 4.4 Diagramas de conexión del sistema.

A continuación se presenta cada una de las partes que conforman los dispositivos necesarios para lograr la comunicación serie, el sensado de límite y la activación del puente H con sus respectivas señales de PWM para cada uno de sus ejes.

#### 4.4.1 Diagrama para lograr la comunicación serie

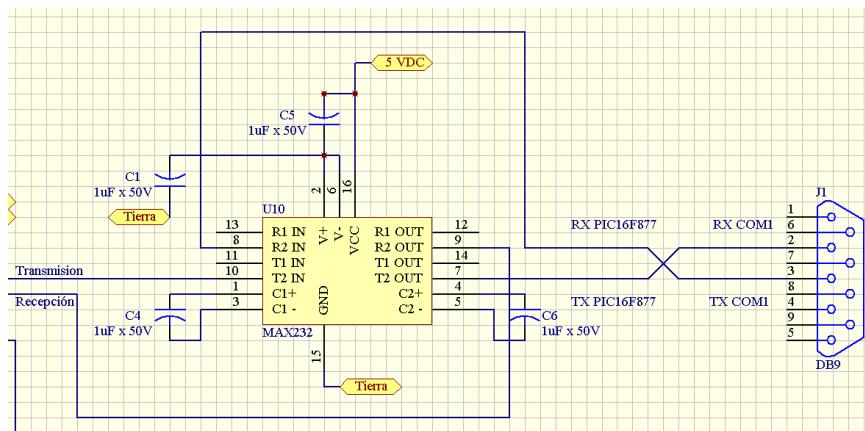
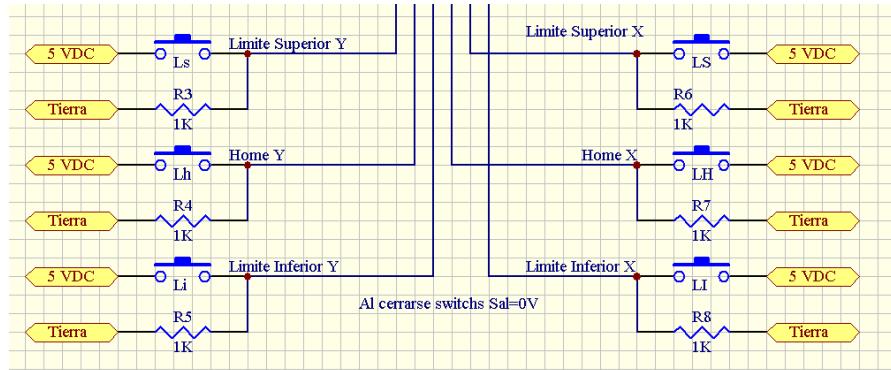


Figura 4-26 MAX232 Comunicación serie

#### 4.4.2 Diagrama que simula los Switches limite

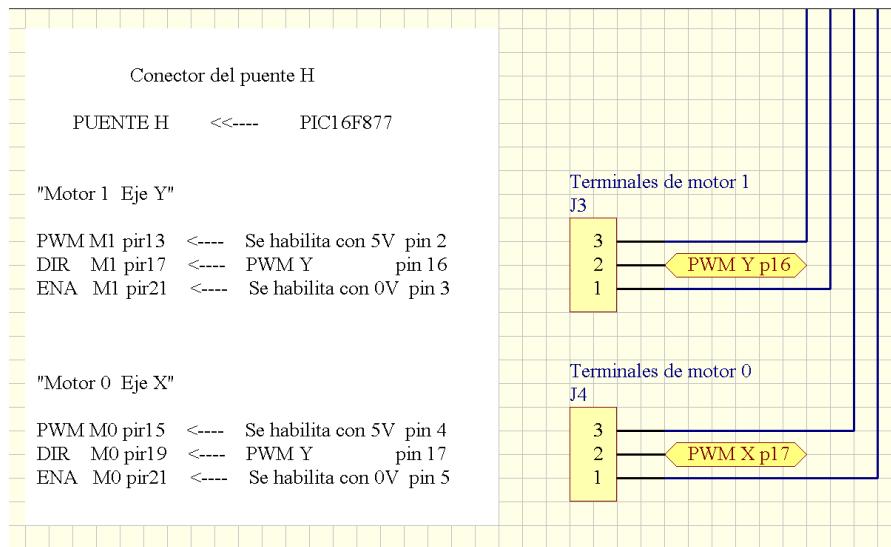
Cuando un switch es interrumpido el voltaje de salida es de 0V, si no es así el voltaje es 5V.



**Figura 4-27** Switches limite para los dos ejes

#### 4.4.3 Diagrama de conexión con el puente H

Es necesario hacer cuidadosamente la conexión entre el microcontrolador y el puente H ya que las señales deben corresponder para su óptimo funcionamiento.



**Figura 4-28** Conexión de puente H

#### 4.4.4 Diagrama completo

Las secciones anteriores se interconectan para tener el diagrama completo para el control por velocidad para un robot de dos grados de libertad.

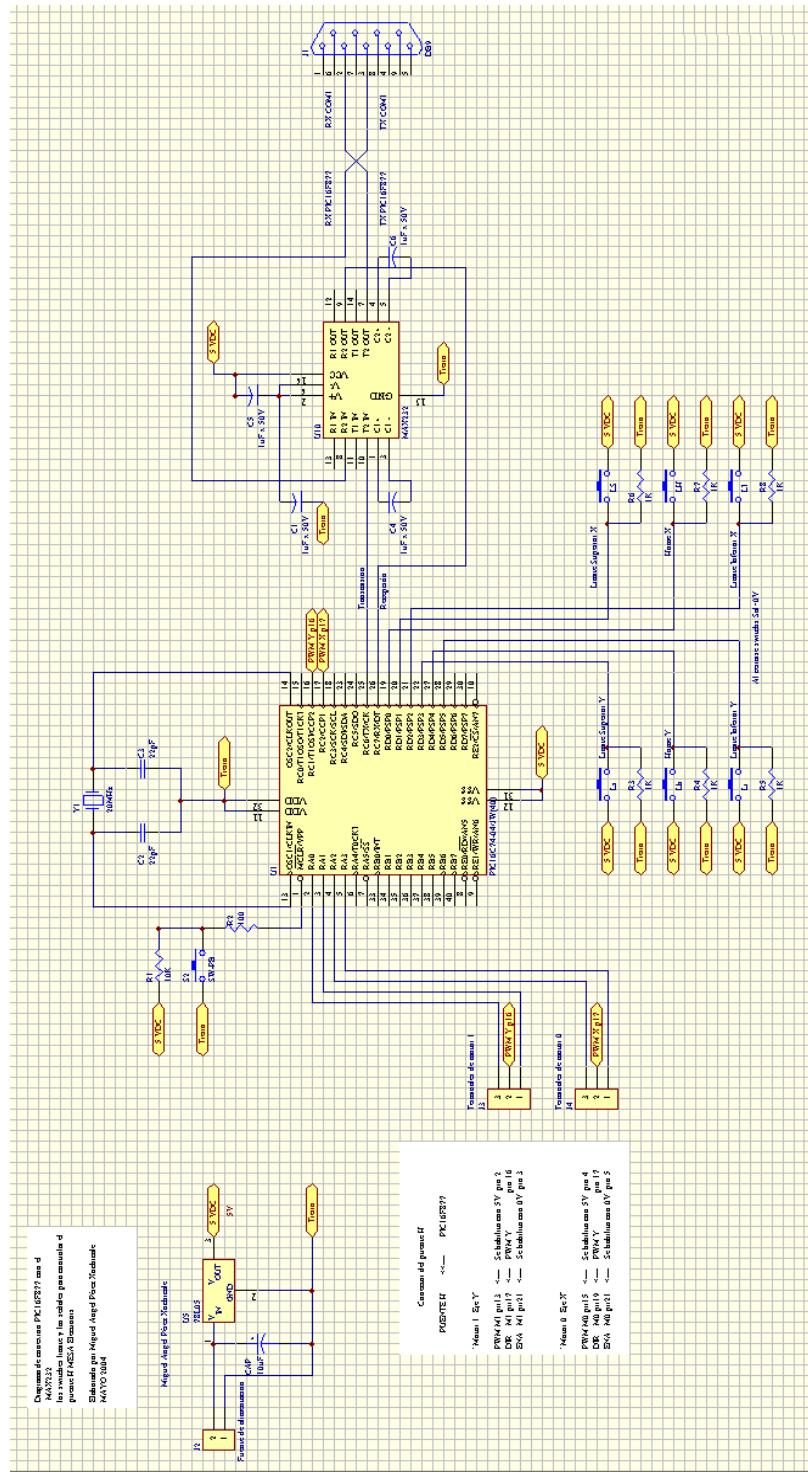


Figura 4-29 Diagrama de conexión del sistema

V.

# Pruebas y resultados

## 5.1 Introducción

El proyecto cumplió con el objetivo planteado, es decir, logró el control por velocidad a través de comunicación serie entre un microcontrolador PIC16F877 y el programa en LabVIEW, así se pobra instalar en la institución para su apto desempeño.

La necesidad contar con un control por velocidad fue satisfecha, se elaboró el programa en LabVIEW, y el programa para el microcontrolador y así se tuvo completo control sobre cada uno de los ejes, el proceso de sensado de límites, los indicadores de funciones, la transmisión de datos, y su constante operación de entrar y salir de cada una de las rutinas, para así poder tener un mejor conocimiento del funcionamiento del sistema.

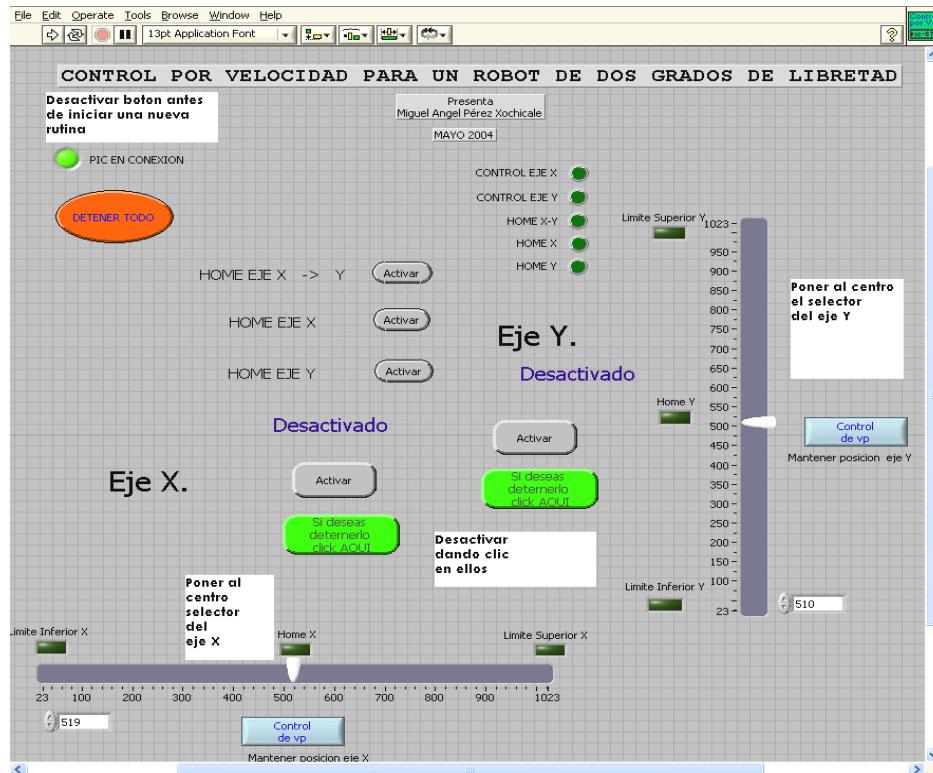
A lo largo del proyecto se tuvieron problemas, el primer problema que se presentó fue la configuración apta para la comunicación entre el microcontrolador y el programa de LabVIEW, debido a la velocidad de comunicación que no coincidía, otro problema que se tuvo es la sincronización del PIC con el programa de LabVIEW al configurar el ciclo útil del PWM, ya que

se realiza una rutina que sincroniza al PIC, con el programa, no se esta seguro que la sincronización sea la adecuada y satisfactoria pues no se cuenta con un osciloscopio para, tratar de hacer una apta sincronización.

En la elaboración de hardware, se tuvo la problemática de manejar diferentes tipos de señales TTL y RS232, así que fue necesario utilizar dispositivos especiales que permiten el cambio de un tipo de señal a otro; para ello se uso el componente MAX232, el cual es un componente CMOS, esto dificulto la elaboración del hardware, ya que su manejo debía ser cuidadoso y siempre existía la posibilidad de que el componente estuviera dañado sin que lo supiéramos, por eso se tuvo que verificar el circuito cada vez que se conecta al COM1 de la computadora personal.

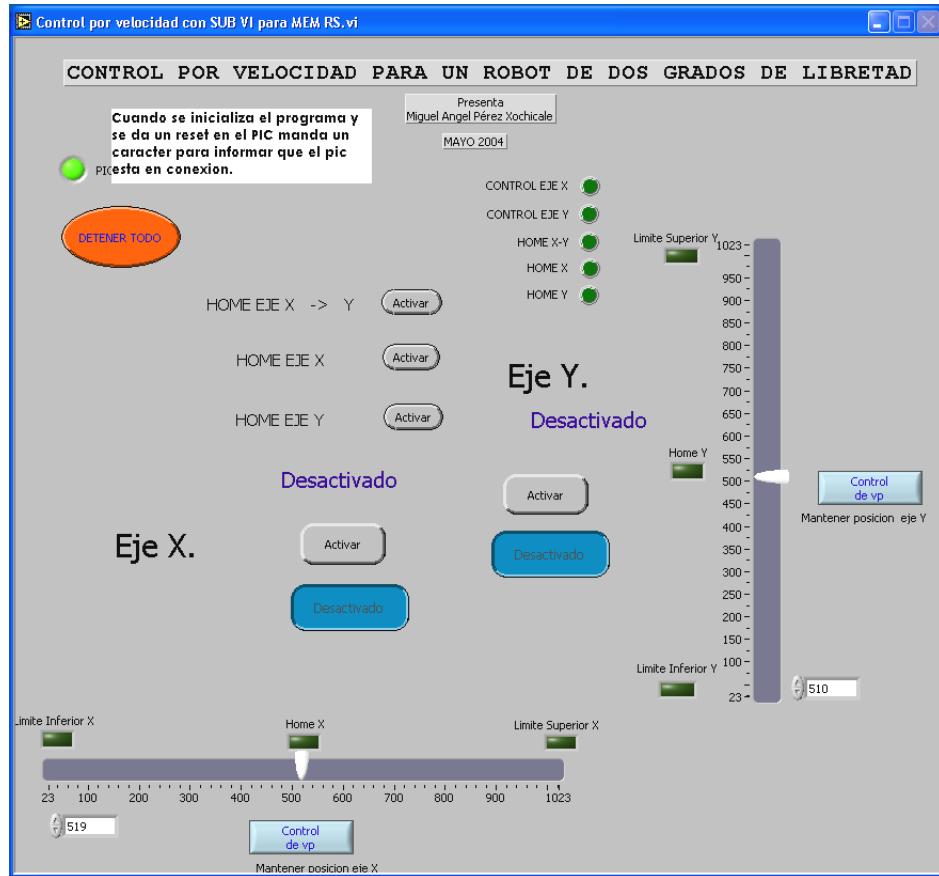
## 5.2 Comunicación entre dispositivos para el control por velocidad

Siempre antes de que corramos el programa en LabVIEW, es necesario colocar los selectores de ciclo útil al centro de cada uno, posteriormente, desactivar los botones de color verde, para que estén desactivados en cuando se inicie el correr el programa, ver figura 5-1.



**Figura**

### 5-1Inicializacion del programa para control por velocidad



**Figura 5-2 PIC EN CONEXIÓN**

### 5.2.1 Activación del control eje X

Cuando se da clic en el botón de activar para el eje X, este manda un dato por el COM1 hacia el PIC, y entra a la rutina del DATO\_B para que el microcontrolador transmita un dato necesario para la activación del indicador que muestra que función esta realizando el microcontrolador, en este momento se puede controlar el ciclo útil del PWM para el motor del eje X variando el selector, ver figura 5-3(a) ó oprimiendo el botón control de vp, mantenemos un PWM de 50% de ciclo útil, ver figura 5-3(b).

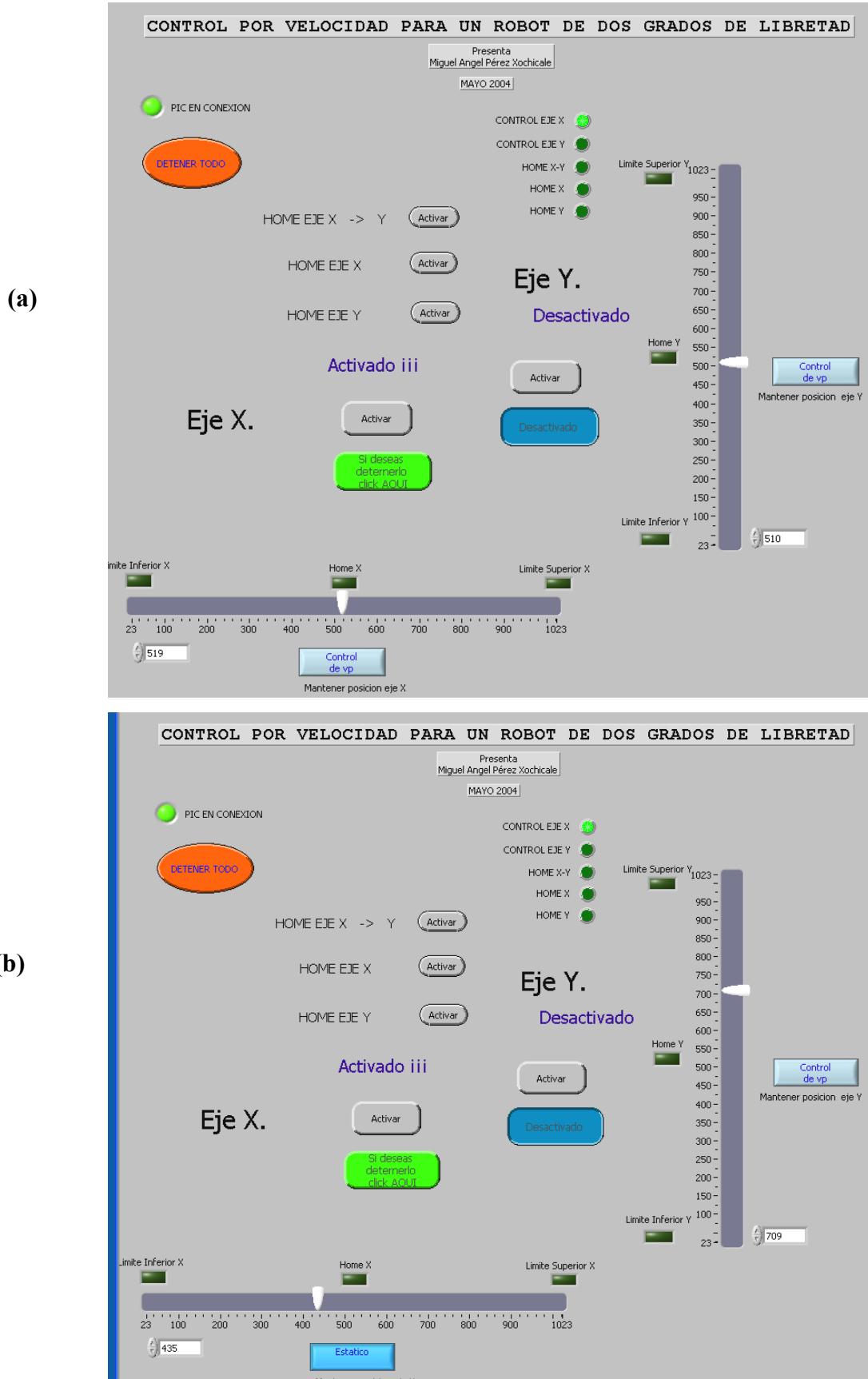
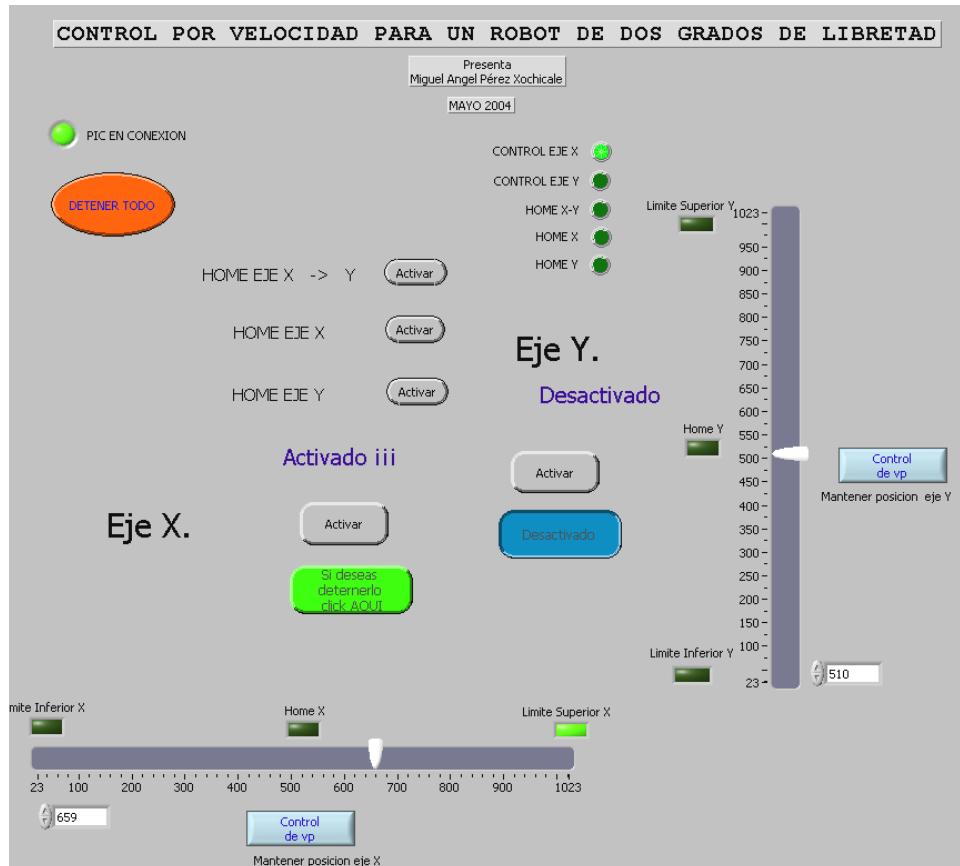


Figura 5-3 Activación del control eje X

### 5.2.1.1 Indicadores de límites

Siempre se están sensando los switches de el eje X, como en la figura siguiente si el robot esta girando hacia la derecha y detecta la interrupción del switch limite superior X, es activado proporcionando en este momento un ciclo útil de 50 % para mantenerse sin movimiento, ver figura 5-4



**Figura 5-4 Límite superior X interrumpido**

Ahora es necesario desactivarlo dando clic en el botón verde, y así desactivar también el indicador de que ninguna de las rutinas en el microcontrolador están activas.

### 5.2.2 Activación del control eje Y

Cuando se da clic en el botón de activar para el eje Y, este manda un dato por el COM1 hacia el PIC, y entra a la rutina del DATO\_A para que el microcontrolador transmita un dato necesario para la activación del indicador que muestra que función esta realizando el microcontrolador, en este momento se puede controlar el ciclo útil del PWM para el motor del eje Y, variando el selector o oprimiendo el botón control de vp, mantenemos un PWM de 50% de ciclo útil, ver figura 5-5.

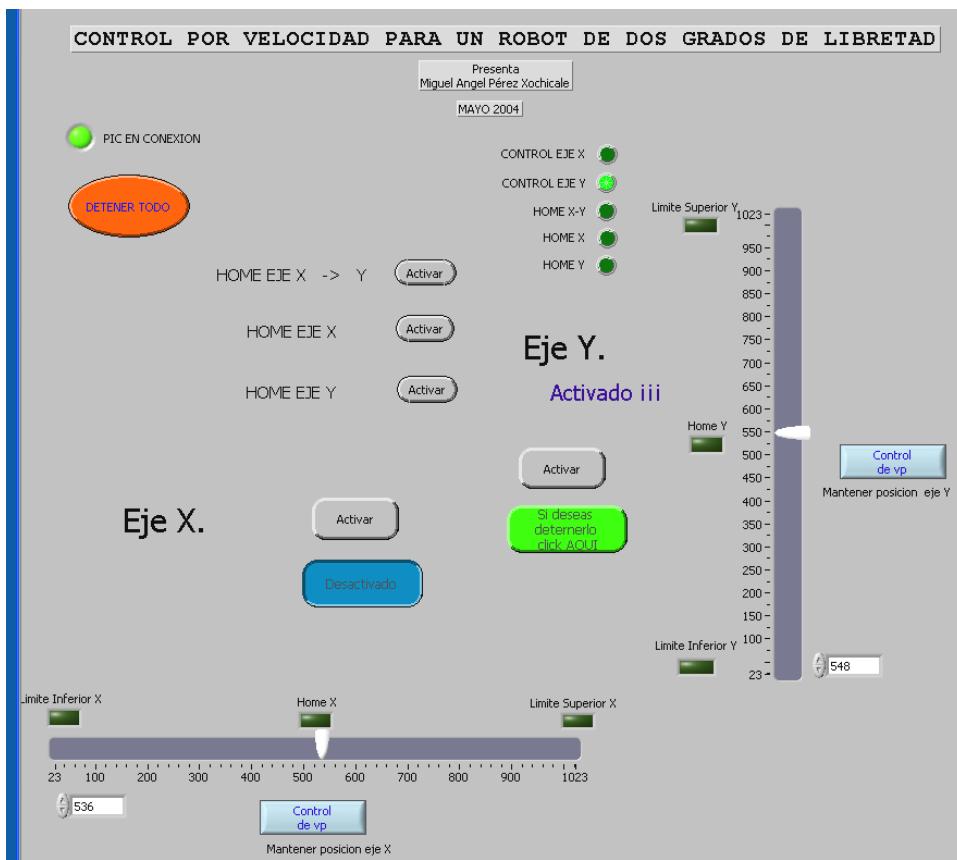


Figura 5-5 Activación del control eje Y

### 5.2.2.1 Indicadores de límites

Siempre se están sensando los switches de el eje Y, como en la figura siguiente si el robot esta girando hacia la derecha y detecta la interrupción del switch limite superior Y, es activado proporcionando en este momento un ciclo útil de 50 % para mantenerse sin movimiento, ver figura 5-6

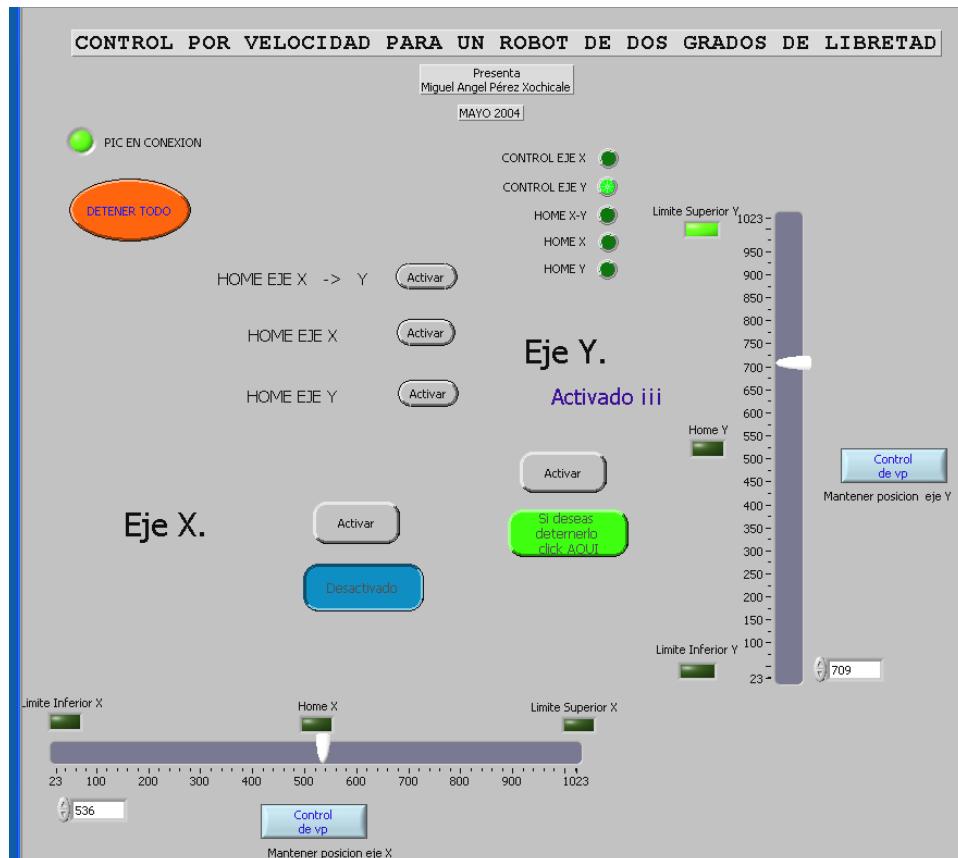
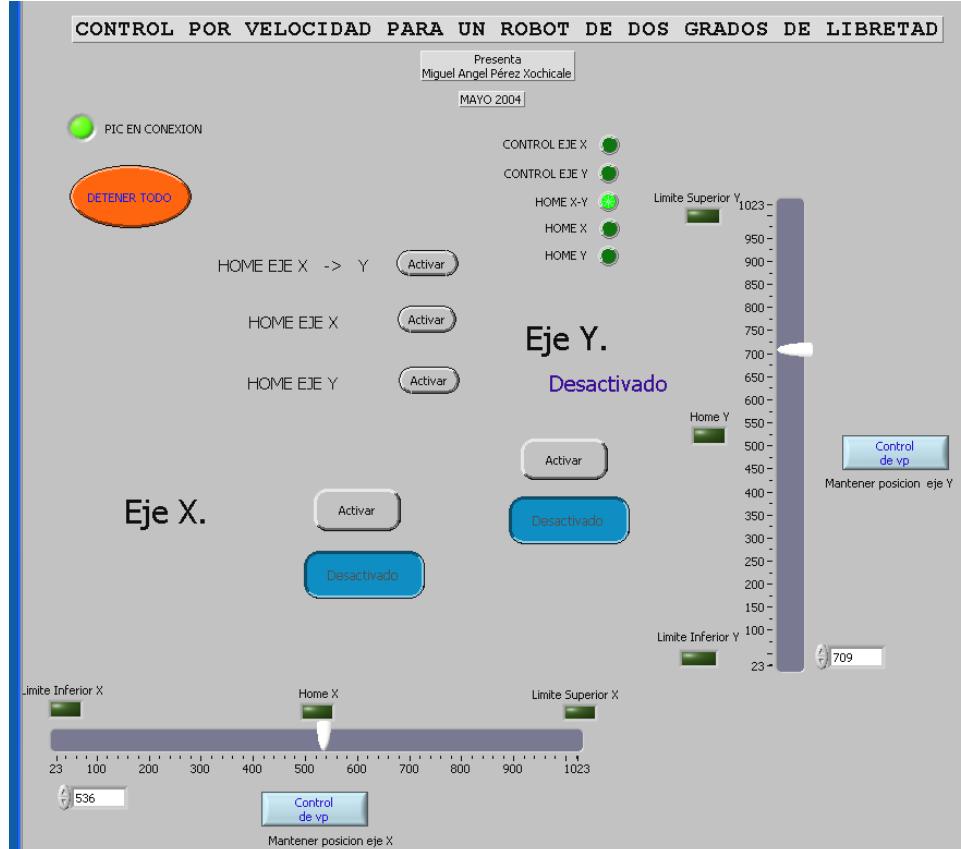


Figura 5-6 Límite superior Y interrumpido

### 5.2.3 Activación de Home X – Y

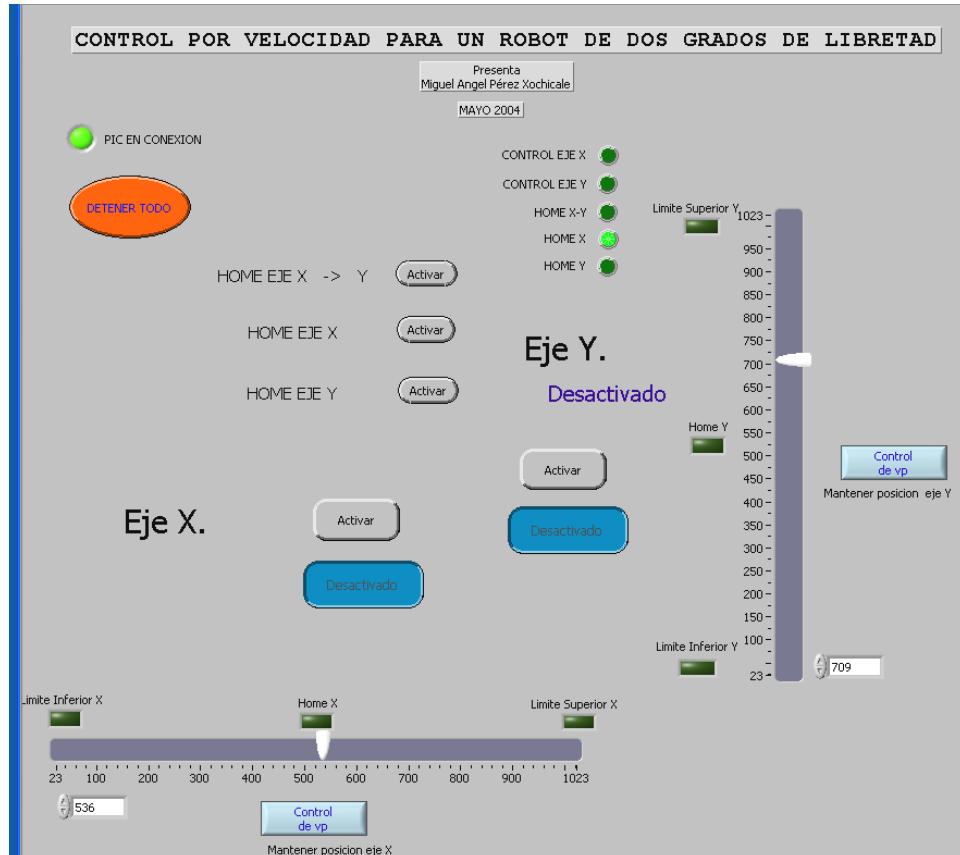
Cuando es activada esta rutina dentro del microcontrolador, manda un dato para activar indicador que esta en esta rutina, y como se vio en el capitulo IV, en los diagramas de flujo, pues aquí entra a la rutina DATO\_E, cuando llega a home del eje Y manda un dato para desactivar el indicador de HOME X-Y, entonces sale de la rutina y se va al lazo principal, ver figura 5-7.



**Figura 5-7 Activación de HOME X-Y**

#### 5.2.4 Activación de Home X

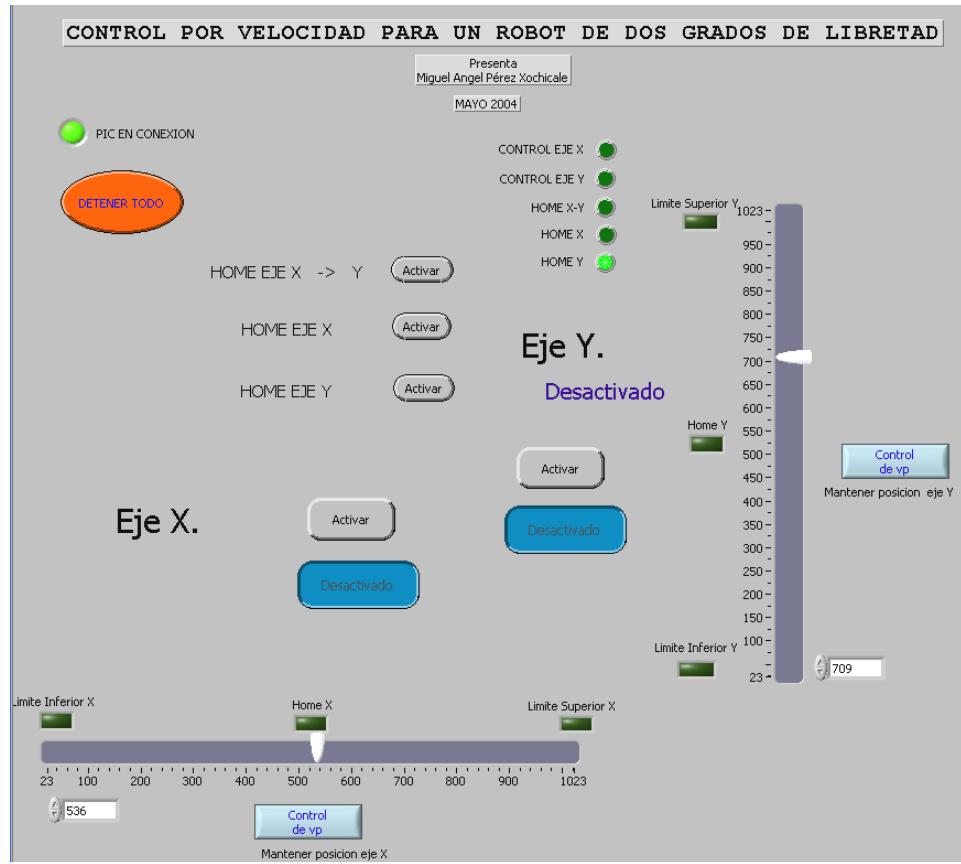
Cuando es activada esta rutina dentro del microcontrolador, manda un dato para activar indicador que esta en esta rutina, y como se vio en el capítulo IV, en los diagramas de flujo, pues aquí entra a la rutina DATO\_C, cuando llega a home del eje X manda un dato para desactivar el indicador de HOME Y, entonces sale de la rutina y se va al lazo principal, ver figura 5-8.



**Figura 5-8 Activación HOME X**

### 5.2.5 Activación de Home Y

Cuando es activada esta rutina dentro del microcontrolador, manda un dato para activar indicador que está en esta rutina, y como se vio en el capítulo IV, en los diagramas de flujo, pues aquí entra a la rutina DATO\_D, cuando llega a home del eje Y manda un dato para desactivar el indicador de HOME Y, entonces sale de la rutina y se va al lazo principal, ver figura 5-9.



**Figura 5-9 Activación HOME Y**

# VII.

## Conclusiones y recomendaciones

### Conclusiones

Durante el desarrollo del proyecto presente se logró alcanzar cada uno de los objetivos particulares satisfactoriamente realizando la programación del microcontrolador, necesaria para realizar las rutinas correspondientes con la interfaz que se realizó con LabVIEW. En LabVIEW se crearon los SubVIs necesarios para la aplicación del control por velocidad evaluando y probando dando como resultado satisfactorio

El control implementado con las herramientas anteriores es muy versátil y no requiere de dispositivos de una gran inversión económica, además de su manejo facilita la colocación en un espacio pequeño en comparación a la tarjeta que se manejaba anteriormente, ya que solo es un circuito integrado, en el cual se conectarán las diferentes señales de entrada o de salida, por otro lado la comunicación por el protocolo RS-232-C nos permite la conexión con un cable entre PIC y PC de hasta 50 FT (15.24 mts) y contemplar velocidades de transmisión hechas estándar de: 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600 y 19200 bit/seg. Las cuales pueden ser configuradas por el usuario dentro de la programación del PIC. Que puede ser conectado a cualquier PC que cuente con puerto serie y sea posible instalar LabVIEW.

Se concluye remarcando la importancia que la elaboración del proyecto tuvo, ya que LabVIEW, tiene mucho más aplicaciones que la presentada, que tiene un gran potencial en relación a la comunicaciones y sus múltiples módulos y tarjetas con las que puede interactuar, al

igual que los microcontroladores PIC de la serie 18, que contiene un conversor analógico digital con resolución de 10bit, múltiples conexiones de interfases RS-232, comparadores de arriba de 256 bytes, con voltajes de operación para el chip de menos de 2.0 volts, con un oscilador de 40MHz para aplicaciones de alto rendimiento.

En el microcontrolador se debe tomar en cuenta los tiempos en que realiza la transmisión y recepción de datos serie, ya que Labview se debe sincronizar satisfactoriamente para un funcionamiento entre hardware y software. Con el manejo de LabVIEW es necesario tener bien afirmados su conocimiento acerca del manejo de ciclos while, ciclos de secuencias, casos, etc., ya que la programación de la interfase con LabVIEW se basa mucho en las funciones de programación anteriormente citadas, también es necesario tener bien comprendidos las funciones para el puerto serie, conversión de palabras de control a bits, números decimales y caracteres tipo ASCII.

### **Perspectivas**

En el momento de la implementación con el robot necesariamente debe verificar los pines del microcontrolador, lo que respecta a la conexión con el puente H, se debe realizar correctamente ya que este cuenta con tres terminales, una de las cuales es donde va la señal de PWM, la cual estará activada en alto, otra terminal para el control de sentido, donde conectaremos la salida de PWM del PIC, y la habilitación del motor, que esta negado y se le manda un bit bajo, y los swith de cada uno de los límites y la posición home, que corresponda a cada eje. Una posible mejora del sistema que se realiza en dicho proyecto es implementar memoria que almacenen la posición del robot a través de encoder, con lo cual se podrá, tener un sistema retroalimentado, con señales de posición de error, velocidad de respuesta, con la implementación un control más eficaz. Algunas otras sugerencias de mejora podrían surgir conforme el usuario vaya utilizando la aplicación.

# **Apéndice A.**

## Hoja técnica del PIC16F87X



# PIC16F87X

## 28/40-Pin 8-Bit CMOS FLASH Microcontrollers

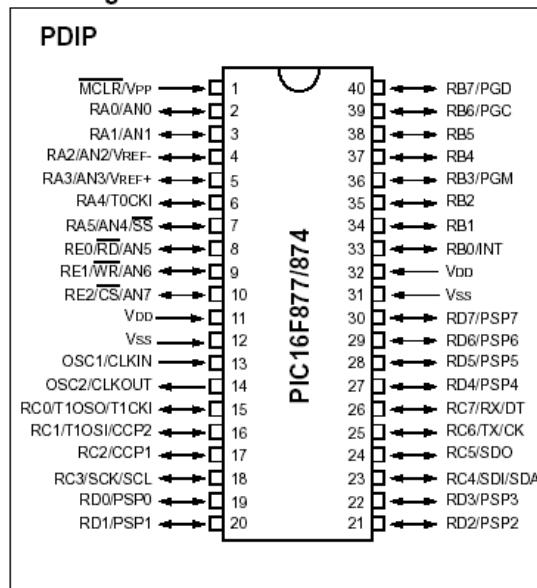
### Devices Included in this Data Sheet:

- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

### Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input  
DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,  
Up to 368 x 8 bytes of Data Memory (RAM)  
Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and  
Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

### Pin Diagram



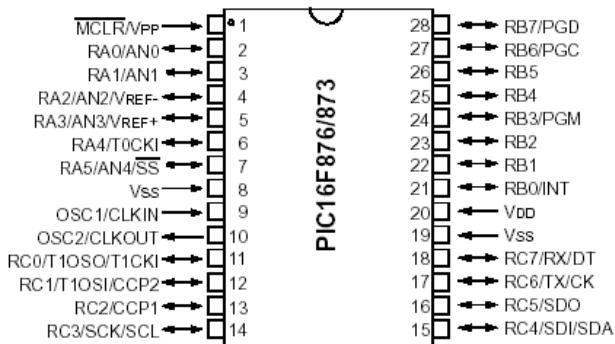
### Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I<sup>2</sup>C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

# PIC16F87X

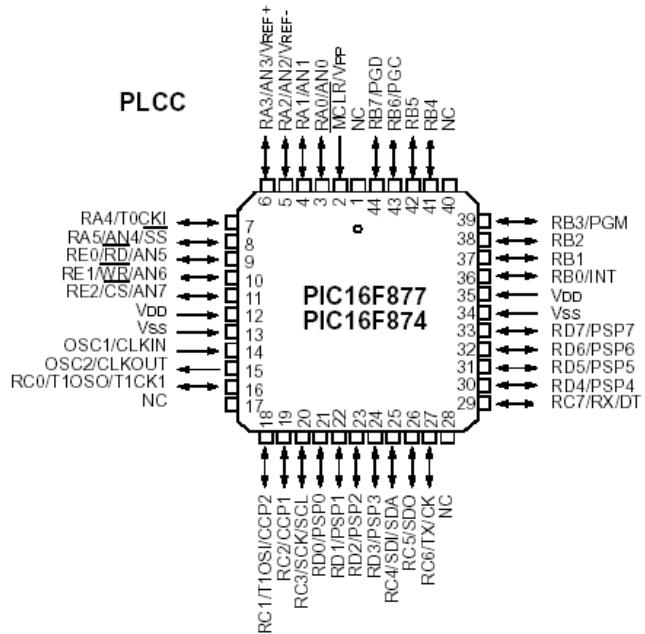
## Pin Diagrams

PDIP, SOIC



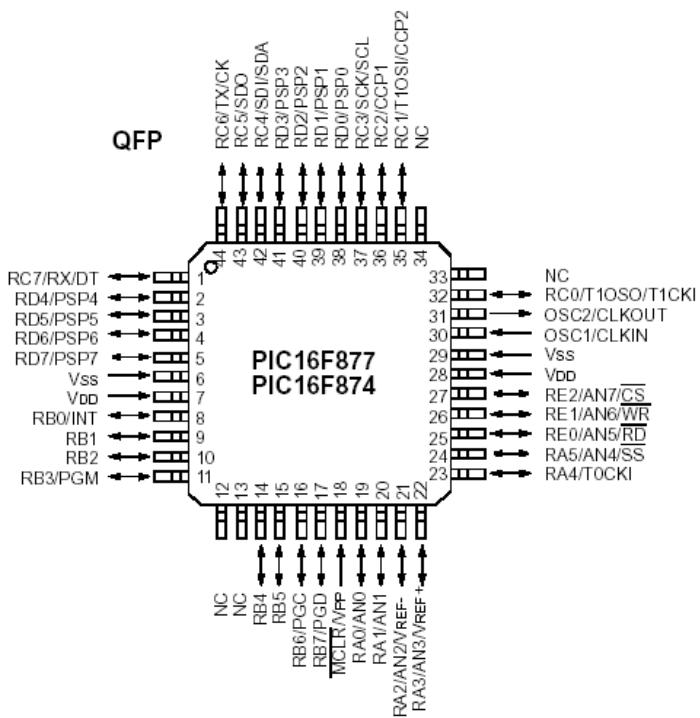
PIC16F876/873

PLCC



PIC16F877  
PIC16F874

QFP



# PIC16F87X

---

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz			
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

# PIC16F87X

TABLE 1-1: PIC16F873 AND PIC16F876 PINOUT DESCRIPTION

Pin Name	DIP Pin#	SOIC Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	9	9	I	ST/CMOS <sup>(3)</sup>	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	10	10	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, the OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/VPP	1	1	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
RA0/AN0	2	2	I/O	TTL	PORTA is a bi-directional I/O port. RA0 can also be analog input0.
RA1/AN1	3	3	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	6	I/O	ST	RA4 can also be the clock input to the Timer0 module. Output is open drain type.
RA5/SS/AN4	7	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
RB0/INT	21	21	I/O	TTL/ST <sup>(1)</sup>	PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs. RB0 can also be the external interrupt pin.
RB1	22	22	I/O	TTL	
RB2	23	23	I/O	TTL	
RB3/PGM	24	24	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	25	25	I/O	TTL	Interrupt-on-change pin.
RB5	26	26	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	27	27	I/O	TTL/ST <sup>(2)</sup>	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	28	28	I/O	TTL/ST <sup>(2)</sup>	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
RC0/T1OSO/T1CKI	11	11	I/O	ST	PORTC is a bi-directional I/O port. RC0 can also be the Timer1 oscillator output or Timer1 clock input.
RC1/T1OSI/CCP2	12	12	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	13	13	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	14	14	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I <sup>2</sup> C modes.
RC4/SDI/SDA	15	15	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I <sup>2</sup> C mode).
RC5/SDO	16	16	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	17	17	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	18	18	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
Vss	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
Vdd	20	20	P	—	Positive supply for logic and I/O pins.

Legend: I = input   O = output   I/O = input/output   P = power

— = Not used

TTL = TTL input

ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

# PIC16F87X

---

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS <sup>(4)</sup>	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/VPP	1	2	18	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
RA0/AN0	2	3	19	I/O	TTL	PORTA is a bi-directional I/O port. RA0 can also be analog input0.
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer/counter. Output is open drain type.
RA5/SS/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
RB0/INT	33	36	8	I/O	TTL/ST <sup>(1)</sup>	PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs. RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	41	14	I/O	TTL	Interrupt-on-change pin.
RB5	38	42	15	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	39	43	16	I/O	TTL/ST <sup>(2)</sup>	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST <sup>(2)</sup>	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.

Legend: I = input    O = output  
— = Not used

I/O = input/output  
TTL = TTL input

P = power  
ST = Schmitt Trigger input

**Note 1:** This buffer is a Schmitt Trigger input when configured as an external interrupt.

**2:** This buffer is a Schmitt Trigger input when used in Serial Programming mode.

**3:** This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).

**4:** This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

# PIC16F87X

---

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION (CONTINUED)

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	PORTC is a bi-directional I/O port. RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I <sup>2</sup> C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I <sup>2</sup> C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
						PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	I/O	ST/TTL <sup>(3)</sup>	
RD1/PSP1	20	22	39	I/O	ST/TTL <sup>(3)</sup>	
RD2/PSP2	21	23	40	I/O	ST/TTL <sup>(3)</sup>	
RD3/PSP3	22	24	41	I/O	ST/TTL <sup>(3)</sup>	
RD4/PSP4	27	30	2	I/O	ST/TTL <sup>(3)</sup>	
RD5/PSP5	28	31	3	I/O	ST/TTL <sup>(3)</sup>	
RD6/PSP6	29	32	4	I/O	ST/TTL <sup>(3)</sup>	
RD7/PSP7	30	33	5	I/O	ST/TTL <sup>(3)</sup>	
						PORTE is a bi-directional I/O port.
RE0/RD/AN5	8	9	25	I/O	ST/TTL <sup>(3)</sup>	RE0 can also be read control for the parallel slave port, or analog input5.
RE1/WR/AN6	9	10	26	I/O	ST/TTL <sup>(3)</sup>	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/CS/AN7	10	11	27	I/O	ST/TTL <sup>(3)</sup>	RE2 can also be select control for the parallel slave port, or analog input7.
Vss	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
Vdd	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28, 40	12,13, 33,34		—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input

O = output

— = Not used

I/O = input/output

TTL = TTL input

P = power

ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).

4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

**FIGURE 2-3:** PIC16F877/876 REGISTER FILE MAP

File Address	File Address	File Address	File Address
Indirect addr.(*)	Indirect addr.(*)	Indirect addr.(*)	Indirect addr.(*)
00h TMR0	01h OPTION_REG	80h TMR0	100h OPTION_REG
02h PCL	02h PCL	81h PCL	101h PCL
03h STATUS	03h STATUS	82h STATUS	102h STATUS
04h FSR	04h FSR	83h FSR	103h FSR
PORTA	05h TRISA	84h PORTB	104h PORTB
PORTB	06h TRISB	85h TRISC	105h TRISB
PORTC	07h TRISC	86h	106h
PORTD <sup>(1)</sup>	08h TRISD <sup>(1)</sup>	87h	107h
PORTE <sup>(1)</sup>	09h TRISE <sup>(1)</sup>	88h	108h
PCLATH	0Ah PCLATH	89h	109h
INTCON	0Bh INTCON	8Ah PCLATH	10Ah PCLATH
PIR1	0Ch PIE1	8Bh INTCON	10Bh INTCON
PIR2	0Dh PIE2	8Ch EEDATA	10Ch EECON1
TMR1L	0Eh PCON	8Dh EEADR	10Dh EECON2
TMR1H	0Fh	8Eh EEDATH	10Eh Reserved <sup>(2)</sup>
T1CON	10h	8Fh EEADRH	10Fh Reserved <sup>(2)</sup>
TMR2	11h SSPCON2	90h	110h
T2CON	12h PR2	91h	111h
SSPBUF	13h SSPADD	92h	112h
SSPCON	14h SSPSTAT	93h	113h
CCPR1L	15h	94h	114h
CCPR1H	16h	95h	115h
CCP1CON	17h	96h	116h
RCSTA	18h TXSTA	97h General Purpose Register	117h General Purpose Register
TXREG	19h SPBRG	98h 16 Bytes	118h 16 Bytes
RCREG	1Ah	99h	119h 16 Bytes
CCPR2L	1Bh	9Ah	11Ah
CCPR2H	1Ch	9Bh	11Bh
CCP2CON	1Dh	9Ch	11Ch
ADRESH	1Eh ADRESL	9Dh	11Dh
ADC0N0	1Fh ADCON1	9Eh	11Eh
	20h	9Fh	11Fh
General Purpose Register 96 Bytes		A0h	120h
	General Purpose Register 80 Bytes		General Purpose Register 80 Bytes
	accesses 70h-7Fh	EFh	16Fh
		F0h	170h
		FFh	17Fh
Bank 0	Bank 1	Bank 2	Bank 3
Unimplemented data memory locations, read as '0'.			
* Not a physical register.			
<b>Note 1:</b> These registers are not implemented on the PIC16F876.			
<b>2:</b> These registers are reserved, maintain these registers clear.			

# PIC16F87X

FIGURE 2-4: PIC16F874/873 REGISTER FILE MAP

File Address	File Address	File Address	File Address
Indirect addr. <sup>(*)</sup>	00h	Indirect addr. <sup>(*)</sup>	100h
TMR0	01h	OPTION_REG	101h
PCL	02h	PCL	102h
STATUS	03h	STATUS	103h
FSR	04h	FSR	104h
PORTA	05h	TRISA	105h
PORTB	06h	TRISB	106h
PORTC	07h	TRISC	107h
PORTD <sup>(1)</sup>	08h	TRISD <sup>(1)</sup>	108h
PORTE <sup>(1)</sup>	09h	TRISE <sup>(1)</sup>	109h
PCLATH	0Ah	PCLATH	10Ah
INTCON	0Bh	INTCON	10Bh
PIR1	0Ch	PIE1	10Ch
PIR2	0Dh	PIE2	10Dh
TMR1L	0Eh	PCON	10Eh
TMR1H	0Fh		10Fh
T1CON	10h		110h
TMR2	11h	SSPCON2	
T2CON	12h	PR2	
SSPBUF	13h	SSPADD	
SSPCON	14h	SSPSTAT	
CCPR1L	15h		
CCPR1H	16h		
CCP1CON	17h		
RCSTA	18h	TXSTA	
TXREG	19h	SPBRG	
RCREG	1Ah		
CCPR2L	1Bh		
CCPR2H	1Ch		
CCP2CON	1Dh		
ADRESH	1Eh	ADRESL	
ADCON0	1Fh	ADCON1	
	20h		
General Purpose Register		General Purpose Register	
96 Bytes		96 Bytes	
Bank 0	7Fh	Bank 1	FFh
		accesses 20h-7Fh	
		16Fh	
		170h	
		17Fh	
			1A0h
			accesses A0h - FFh
			1EFh
			1F0h
			1FFh

 Unimplemented data memory locations, read as '0'.

\* Not a physical register.

**Note 1:** These registers are not implemented on the PIC16F873.

**2:** These registers are reserved, maintain these registers clear.

## 2.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. A list of these registers is given in Table 2-1.

The Special Function Registers can be classified into two sets: core (CPU) and peripheral. Those registers associated with the core functions are described in detail in this section. Those related to the operation of the peripheral features are described in detail in the peripheral features section.

**TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:						
<b>Bank 0</b>																	
00h <sup>(3)</sup>	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27						
01h	TMR0	Timer0 Module Register								xxxx xxxx	47						
02h <sup>(3)</sup>	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26						
03h <sup>(3)</sup>	STATUS	IRP	RP1	RP0	T0	PD	Z	DC	C	0001 1xxx	18						
04h <sup>(3)</sup>	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27						
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	29						
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31						
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	33						
08h <sup>(4)</sup>	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	35						
09h <sup>(4)</sup>	PORTE	—	—	—	—	RE2	RE1	RE0	----	-xxx	36						
0Ah <sup>(1,3)</sup>	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter						--0 0000	26					
0Bh <sup>(3)</sup>	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20						
0Ch	PIR1	PSPIF <sup>(3)</sup>	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	22						
0Dh	PIR2	—	(5)	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0	24						
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52						
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52						
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	51						
11h	TMR2	Timer2 Module Register								0000 0000	55						
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	55						
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	70, 73						
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	67						
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	57						
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	57						
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	58						
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	96						
19h	TXREG	USART Transmit Data Register								0000 0000	99						
1Ah	RCREG	USART Receive Data Register								0000 0000	101						
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	57						
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	57						
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	58						
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	116						
1Fh	ADC0N0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON	0000 00-0	111						

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.

Shaded locations are unimplemented, read as '0'.

**Note 1:** The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

**2:** Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

**3:** These registers can be addressed from any bank.

**4:** PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

**5:** PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

# PIC16F87X

---

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:						
<b>Bank 1</b>																	
80h <sup>(3)</sup>	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27						
81h	OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	19						
82h <sup>(3)</sup>	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26						
83h <sup>(3)</sup>	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	18						
84h <sup>(3)</sup>	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27						
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	29						
86h	TRISB	PORTB Data Direction Register								1111 1111	31						
87h	TRISC	PORTC Data Direction Register								1111 1111	33						
88h <sup>(4)</sup>	TRISD	PORTD Data Direction Register								1111 1111	35						
89h <sup>(4)</sup>	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits				0000 -111	37					
8Ah <sup>(1,3)</sup>	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter						---0 0000	26					
8Bh <sup>(3)</sup>	INTCON	GIE	PEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20						
8Ch	PIE1	PSPIE <sup>(2)</sup>	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	21						
8Dh	PIE2	—	(5)	—	EEIE	BCLIE	—	—	CCP2IE	-r-0 0--0	23						
8Eh	PCON	—	—	—	—	—	—	POR	BOR	---- --qq	25						
8Fh	—	Unimplemented								—	—						
90h	—	Unimplemented								—	—						
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	68						
92h	PR2	Timer2 Period Register								1111 1111	55						
93h	SSPADD	Synchronous Serial Port (I <sup>2</sup> C mode) Address Register								0000 0000	73, 74						
94h	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF	0000 0000	66						
95h	—	Unimplemented								—	—						
96h	—	Unimplemented								—	—						
97h	—	Unimplemented								—	—						
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	95						
99h	SPBRG	Baud Rate Generator Register								0000 0000	97						
9Ah	—	Unimplemented								—	—						
9Bh	—	Unimplemented								—	—						
9Ch	—	Unimplemented								—	—						
9Dh	—	Unimplemented								—	—						
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	116						
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000	112						

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.

Shaded locations are unimplemented, read as '0'.

**Note 1:** The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

**2:** Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

**3:** These registers can be addressed from any bank.

**4:** PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

**5:** PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
<b>Bank 2</b>											
100h <sup>(3)</sup>	INDF									0000 0000	27
101h	TMR0									xxxx xxxx	47
102h <sup>(3)</sup>	PCL									0000 0000	26
103h <sup>(3)</sup>	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	18
104h <sup>(3)</sup>	FSR									xxxx xxxx	27
105h	—									—	—
106h	PORTB									xxxx xxxx	31
107h	—									—	—
108h	—									—	—
109h	—									—	—
10Ah <sup>(1,3)</sup>	PCLATH	—	—	—						---0 0000	26
10Bh <sup>(3)</sup>	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
10Ch	EEDATA									xxxx xxxx	41
10Dh	EEADR									xxxx xxxx	41
10Eh	EEDATH	—	—							xxxx xxxx	41
10Fh	EEADRH	—	—	—						xxxx xxxx	41
<b>Bank 3</b>											
180h <sup>(3)</sup>	INDF									0000 0000	27
181h	OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	19
182h <sup>(3)</sup>	PCL									0000 0000	26
183h <sup>(3)</sup>	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	18
184h <sup>(3)</sup>	FSR									xxxx xxxx	27
185h	—									—	—
186h	TRISB									1111 1111	31
187h	—									—	—
188h	—									—	—
189h	—									—	—
18Ah <sup>(1,3)</sup>	PCLATH	—	—	—						---0 0000	26
18Bh <sup>(3)</sup>	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	41, 42
18Dh	EECON2									-----	41
18Eh	—									0000 0000	—
18Fh	—									0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.  
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

# **Apéndice B**

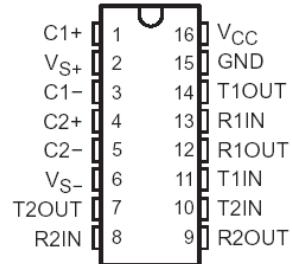
## **Hoja técnica del MAX232**

# MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047L – FEBRUARY 1989 – REVISED MARCH 2004

- Meets or Exceeds TIA/EIA-232-F and ITU Recommendation V.28
- Operates From a Single 5-V Power Supply With 1.0- $\mu$ F Charge-Pump Capacitors
- Operates Up To 120 kbit/s
- Two Drivers and Two Receivers
- $\pm 30$ -V Input Levels
- Low Supply Current . . . 8 mA Typical
- ESD Protection Exceeds JESD 22
  - 2000-V Human-Body Model (A114-A)
- Upgrade With Improved ESD (15-kV HBM) and 0.1- $\mu$ F Charge-Pump Capacitors is Available With the MAX202
- Applications
  - TIA/EIA-232-F, Battery-Powered Systems, Terminals, Modems, and Computers

MAX232 . . . D, DW, N, OR NS PACKAGE  
MAX232I . . . D, DW, OR N PACKAGE  
(TOP VIEW)



## description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply TIA/EIA-232-F voltage levels from a single 5-V supply. Each receiver converts TIA/EIA-232-F inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V, a typical hysteresis of 0.5 V, and can accept  $\pm 30$ -V inputs. Each driver converts TTL/CMOS input levels into TIA/EIA-232-F levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

## ORDERING INFORMATION

TA	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
$0^{\circ}\text{C}$ to $70^{\circ}\text{C}$	PDIP (N)	Tube of 25	MAX232N	MAX232N
	SOIC (D)	Tube of 40	MAX232D	
		Reel of 2500	MAX232DR	MAX232
	SOIC (DW)	Tube of 40	MAX232DW	
		Reel of 2000	MAX232DWR	MAX232
$-40^{\circ}\text{C}$ to $85^{\circ}\text{C}$	SOP (NS)	Reel of 2000	MAX232NSR	MAX232
	PDIP (N)	Tube of 25	MAX232IN	MAX232IN
		Tube of 40	MAX232ID	
	SOIC (D)	Reel of 2500	MAX232IDR	MAX232I
		Tube of 40	MAX232IDW	
		Reel of 2000	MAX232IDWR	MAX232I

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at [www.ti.com/sc/package](http://www.ti.com/sc/package).



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

LinASIC is a trademark of Texas Instruments.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.



Copyright © 2004, Texas Instruments Incorporated

# MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047L – FEBRUARY 1989 – REVISED MARCH 2004

## Function Tables

### EACH DRIVER

INPUT TIN	OUTPUT TOUT
L	H
H	L

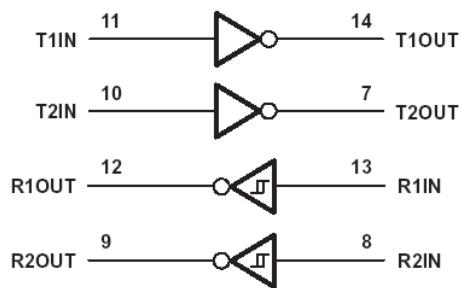
H = high level, L = low level

### EACH RECEIVER

INPUT RIN	OUTPUT ROUT
L	H
H	L

H = high level, L = low level

## logic diagram (positive logic)



# MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047L – FEBRUARY 1989 – REVISED MARCH 2004

## absolute maximum ratings over operating free-air temperature range (unless otherwise noted)<sup>†</sup>

Input supply voltage range, $V_{CC}$ (see Note 1) .....	-0.3 V to 6 V
Positive output supply voltage range, $V_{S+}$ .....	$V_{CC}$ – 0.3 V to 15 V
Negative output supply voltage range, $V_{S-}$ .....	-0.3 V to -15 V
Input voltage range, $V_I$ : Driver .....	-0.3 V to $V_{CC} + 0.3$ V
Receiver .....	$\pm 30$ V
Output voltage range, $V_O$ : T1OUT, T2OUT .....	$V_{S-} - 0.3$ V to $V_{S+} + 0.3$ V
R1OUT, R2OUT .....	-0.3 V to $V_{CC} + 0.3$ V
Short-circuit duration: T1OUT, T2OUT .....	Unlimited
Package thermal impedance, $\theta_{JA}$ (see Notes 2 and 3): D package .....	73°C/W
DW package .....	57°C/W
N package .....	67°C/W
NS package .....	64°C/W
Operating virtual junction temperature, $T_J$ .....	150°C
Storage temperature range, $T_{STG}$ .....	-65°C to 150°C

<sup>†</sup> Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. All voltages are with respect to network GND.

2. Maximum power dissipation is a function of  $T_J(\max)$ ,  $\theta_{JA}$ , and  $T_A$ . The maximum allowable power dissipation at any allowable ambient temperature is  $P_D = (T_J(\max) - T_A)/\theta_{JA}$ . Operating at the absolute maximum  $T_J$  of 150°C can affect reliability.
3. The package thermal impedance is calculated in accordance with JESD 51-7.

## recommended operating conditions

		MIN	NOM	MAX	UNIT
$V_{CC}$	Supply voltage	4.5	5	5.5	V
$V_{IH}$	High-level input voltage (T1IN, T2IN)	2			V
$V_{IL}$	Low-level input voltage (T1IN, T2IN)			0.8	V
R1IN, R2IN	Receiver input voltage			$\pm 30$	V
$T_A$	Operating free-air temperature	MAX232	0	70	°C
		MAX232I	-40	85	

## electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Note 4 and Figure 4)

PARAMETER	TEST CONDITIONS	MIN	TYP <sup>‡</sup>	MAX	UNIT
$I_{CC}$ Supply current	$V_{CC} = 5.5$ V, All outputs open, $T_A = 25^\circ$ C	8	10		mA

<sup>‡</sup> All typical values are at  $V_{CC} = 5$  V and  $T_A = 25^\circ$ C.

NOTE 4: Test conditions are C1–C4 = 1  $\mu$ F at  $V_{CC} = 5$  V  $\pm 0.5$  V.

# MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047L – FEBRUARY 1989 – REVISED MARCH 2004

## DRIVER SECTION

**electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 4)**

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
$V_{OH}$	High-level output voltage	$T_1OUT, T_2OUT$	$R_L = 3 k\Omega$ to GND	5	7	V
$V_{OL}$	Low-level output voltage‡	$T_1OUT, T_2OUT$	$R_L = 3 k\Omega$ to GND	-7	-5	V
$r_o$	Output resistance	$T_1OUT, T_2OUT$	$V_{S+} = V_{S-} = 0$ , $V_O = \pm 2 V$	300		$\Omega$
$I_{OS}^§$	Short-circuit output current	$T_1OUT, T_2OUT$	$V_{CC} = 5.5 V$ , $V_O = 0$		$\pm 10$	mA
$I_{IS}$	Short-circuit input current	$T_1IN, T_2IN$	$V_I = 0$		200	$\mu A$

† All typical values are at  $V_{CC} = 5 V$ ,  $T_A = 25^\circ C$ .

‡ The algebraic convention, in which the least-positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.

§ Not more than one output should be shorted at a time.

NOTE 4: Test conditions are  $C_1-C_4 = 1 \mu F$  at  $V_{CC} = 5 V \pm 0.5 V$ .

**switching characteristics,  $V_{CC} = 5 V$ ,  $T_A = 25^\circ C$  (see Note 4)**

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
SR	Driver slew rate	$R_L = 3 k\Omega$ to $7 k\Omega$ , See Figure 2		30		$V/\mu s$
SR(t)	Driver transition region slew rate	See Figure 3		3		$V/\mu s$
	Data rate	One TOUT switching		120		kbit/s

NOTE 4: Test conditions are  $C_1-C_4 = 1 \mu F$  at  $V_{CC} = 5 V \pm 0.5 V$ .

## RECEIVER SECTION

**electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 4)**

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT	
$V_{OH}$	High-level output voltage	$R_{1OUT}, R_{2OUT}$	$I_{OH} = -1 mA$	3.5		V	
$V_{OL}$	Low-level output voltage‡	$R_{1OUT}, R_{2OUT}$	$I_{OL} = 3.2 mA$		0.4	V	
$V_{IT+}$	Receiver positive-going input threshold voltage	$R_{1IN}, R_{2IN}$	$V_{CC} = 5 V$ , $T_A = 25^\circ C$		1.7	2.4	V
$V_{IT-}$	Receiver negative-going input threshold voltage	$R_{1IN}, R_{2IN}$	$V_{CC} = 5 V$ , $T_A = 25^\circ C$	0.8	1.2		V
$V_{hys}$	Input hysteresis voltage	$R_{1IN}, R_{2IN}$	$V_{CC} = 5 V$	0.2	0.5	1	V
$r_i$	Receiver input resistance	$R_{1IN}, R_{2IN}$	$V_{CC} = 5 V$ , $T_A = 25^\circ C$	3	5	7	$k\Omega$

† All typical values are at  $V_{CC} = 5 V$ ,  $T_A = 25^\circ C$ .

‡ The algebraic convention, in which the least-positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.

NOTE 4: Test conditions are  $C_1-C_4 = 1 \mu F$  at  $V_{CC} = 5 V \pm 0.5 V$ .

**switching characteristics,  $V_{CC} = 5 V$ ,  $T_A = 25^\circ C$  (see Note 4 and Figure 1)**

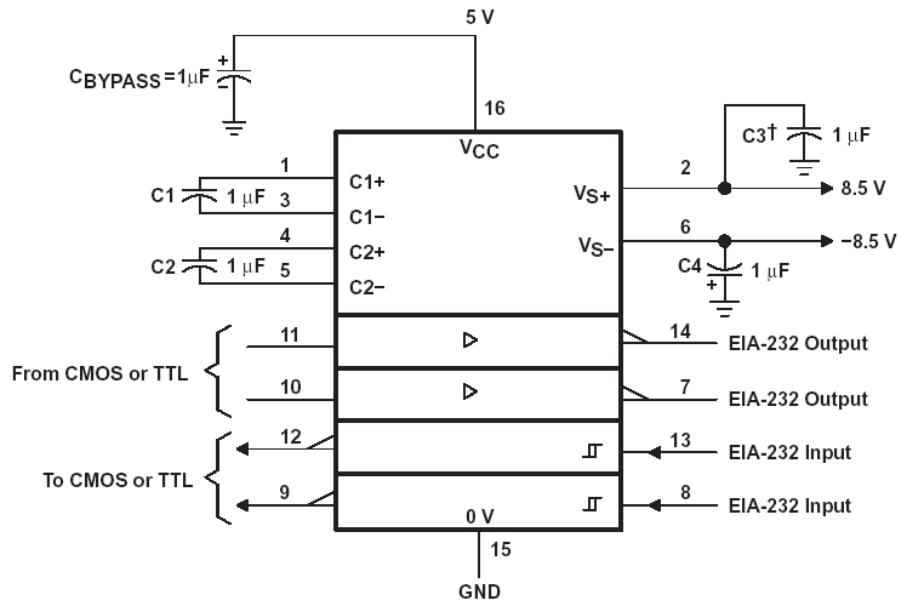
PARAMETER		TYP	UNIT
$t_{PLH(R)}$	Receiver propagation delay time, low- to high-level output	500	ns
$t_{PHL(R)}$	Receiver propagation delay time, high- to low-level output	500	ns

NOTE 4: Test conditions are  $C_1-C_4 = 1 \mu F$  at  $V_{CC} = 5 V \pm 0.5 V$ .

# MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047L – FEBRUARY 1989 – REVISED MARCH 2004

## APPLICATION INFORMATION



† C3 can be connected to V<sub>CC</sub> or GND.

NOTES: A. Resistor values shown are nominal.

B. Nonpolarized ceramic capacitors are acceptable. If polarized tantalum or electrolytic capacitors are used, they should be connected as shown. In addition to the 1- $\mu$ F capacitors shown, the MAX202 can operate with 0.1- $\mu$ F capacitors.

**Figure 4. Typical Operating Circuit**

# **Apéndice C**

## Grabador de PICs

# Programador PP2.

Por José Manuel García

## Introducción.

Se han presentado en artículos anteriores otros programadores muy simples aunque limitados. Presentamos ahora un programador semi-profesional, capaz de programar una cantidad ingente de dispositivos actuales y futuros, gracias a sus posibilidades de ampliación, desde memorias EEPROM de varias familias hasta microcontroladores de diversos fabricantes y, en general, casi cualquier dispositivo programable mediante un protocolo serie. Y todo ello, manteniendo la compatibilidad con el programador en el que está basado, el ProPic Programmer, por lo que se puede utilizar con la mayor parte del software de programación de dispositivos existente. Teniendo en cuenta esto, su precio es razonable, unos 15 euros con zócalos estándar y menos de 50 euros con ZIF (zócalo de fuerza de inserción nula).

Además del zócalo incluido en el programador, que permite programar los dispositivos más habituales, se ha añadido un conector a través del cuál, mediante adaptadores muy simples, se pueden programar otros dispositivos menos utilizados, o en encapsulados diferentes, por ejemplo dispositivos para montaje superficial. A través de este conector, también se pueden programar dispositivos en el propio circuito (ICSP), y será la vía para adaptarse a otros dispositivos programables en serie que aparezcan en el futuro. La tabla 1 muestra los dispositivos programables actualmente (los que aparecen en negrita los he probado personalmente).

Dispositivo:	Zócalo	ICSP	Adapt.
PICs: <b>12C508, 12C508A, 12C509, 12C509A, 12CE518, 12CE519, 12C671, 12C672, 12CE673, 12CE674, 16C61, 16C62A, 12C62B, 16C63, 12C63A, 16C64A, 16C65A, 12C65B, 16C66, 16C67, 16C71, 16C72, 12C72A, 16C73A, 16C73B, 16C74A, 16C74B, 16C76, 16C77, 16F83, 16C84, 16F84, 16F84A, 16C433, 16C505, 16C620, 16C621, 16C622, 16C622A, 16F627, 16F628, 16C715, 16F870, 16F871, 16F872, 16F873, 16F874, 16F876, 16F877, 16C923, 16C924</b>	SI	SI	NO

## El circuito.

El esquema del programador aparece en la figura 1. Su diseño se basa en el ProPic Programmer, y utiliza las mismas señales del puerto paralelo que él para comunicarse con el PC. Sin embargo, no se puede decir que sea absolutamente compatible, ya que algunas señales están invertidas respecto al original. El motivo es adaptarse a los componentes disponibles, como luego se verá, y no supone un gran problema, ya que la mayor parte de los programas existentes para grabación de chips permiten invertir las señales a voluntad (entre ellos el que yo recomiendo, ICProg). Por otro lado, se ha corregido un fallo de diseño que hacía imposible que el ProPic Programmer funcionara correctamente (al menos el esquema disponible de dicho programador).

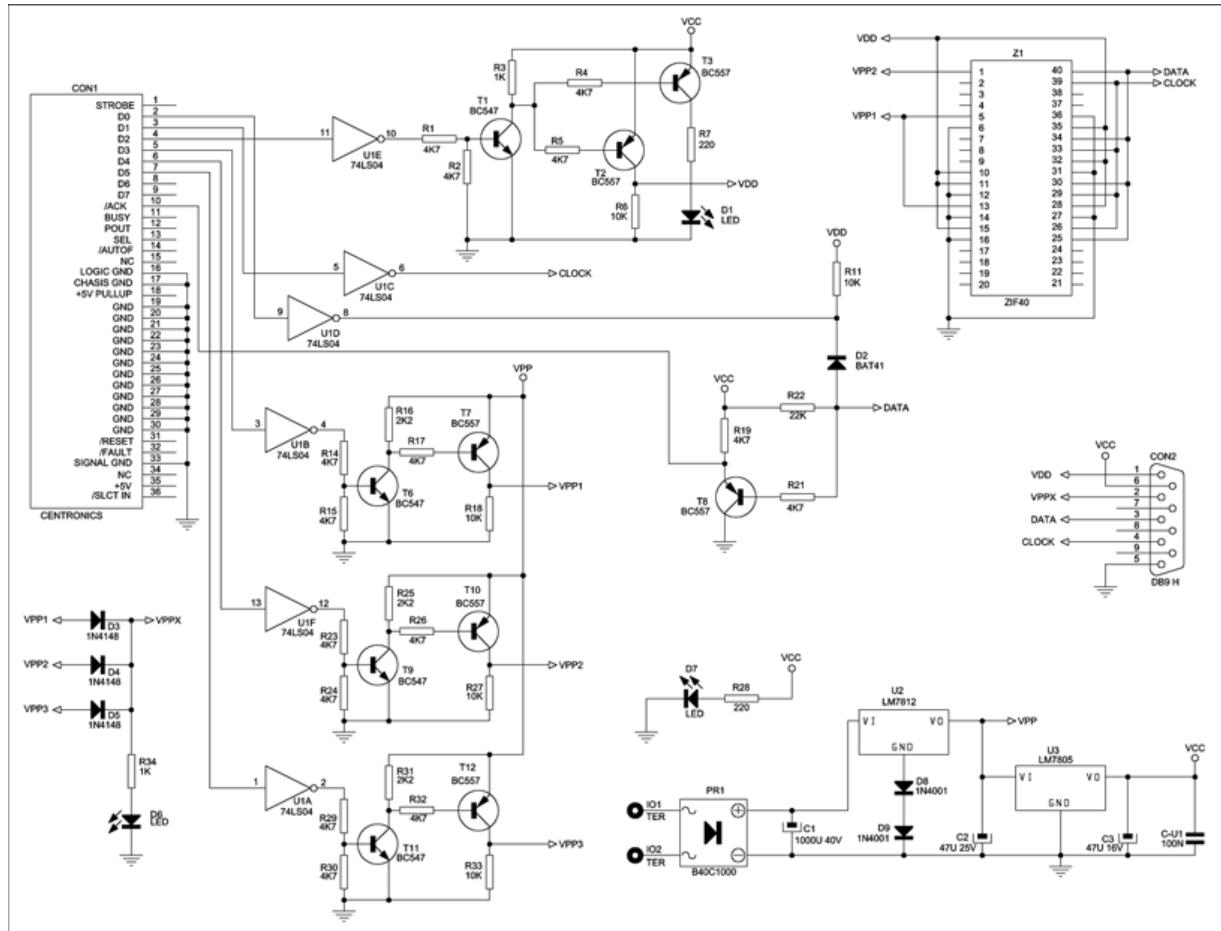


Figura 1

Las señales del puerto paralelo utilizadas son D0 para enviar la señal DATA al PIC (o el dispositivo que se está programando), D1 para la señal CLOCK, D2 para activar la tensión de alimentación del PIC (Vdd) y ACK para recibir la señal DATA enviada por el PIC. D3, D4 y D5 activan la tensión de programación en distintas patillas del zócalo. Esto es necesario para evitar que llegue una tensión tan alta (13V) a una patilla de un PIC que no está preparada para

soportarla (en otros programadores se obvia este problema, presuponiendo que el PIC aguantará esa tensión).

Los inversores U1A a U1F tienen por objeto aislar el circuito del PC, para evitar que éste pueda dañarse por error. Originalmente, en lugar de inversores se utilizaron buffers no inversores de tipo 74LS34, pero la dificultad de encontrar dicho chip me decidió a substituirlo. Como contrapartida, todas las señales están invertidas, por lo que habrá que configurar el software de grabación teniendo en cuenta este detalle, como luego veremos. Quien encuentre un 74LS34 puede ponerlo en el lugar de U1, en cuyo caso el programador será totalmente compatible con el ProPic Programmer, y por tanto no habrá que invertir ninguna señal en la configuración del software.

La conmutación de la alimentación del PIC se consigue mediante los transistores T1 y T2 y los componentes adyacentes. T3 se encarga de activar el LED que indica que el PIC está alimentado. T8 y los componentes que lo rodean sirven para acoplar las señales DATA IN y DATA OUT del puerto paralelo a la señal DATA IN/OUT del PIC. Vpp1, Vpp2 y Vpp3 se conmutan mediante los bloques construidos en torno a T6-T7, T9-T10 y T11-T12 respectivamente. Cuando se activa cualquiera de estas señales, gracias a los diodos D3, D4 y D5 se activa también la señal Vppx y se enciende el LED D6, indicador de que la tensión de programación está activada.

La fuente de alimentación es muy simple. La tensión que viene del transformador se rectifica y se filtra mediante PR1 y C1. U2 estabiliza esta tensión a unos +13V para obtener la tensión de programación, Vpp, ya que su patilla de masa está a aproximadamente +1V respecto a la masa del circuito, gracias a D8 y D9. A partir de esta tensión y mediante U3 obtenemos Vcc (+5V).

El conector CON2, cuya función será permitir tanto la programación de dispositivos en el propio circuito (ICSP) como el acoplamiento de adaptadores para algunos dispositivos, utiliza 6 de sus pines para las siguientes señales:

- PIN 1 Vdd: Tensión de alimentación para el PIC, controlada por la señal D2 del puerto paralelo, para dar o no alimentación al PIC según se necesite.
- PIN 2 Vppx: Tensión de programación del PIC, que será activada por cualquiera de las señales D3, D4 ó D5 del puerto paralelo. Esto garantiza que, sea cual sea el dispositivo que se programe, el software activará la tensión de programación.
- PIN 3 Data: Señal DATA I/O del PIC. Para enviar datos al PIC se utiliza la señal D0, y para recibir datos la señal ACK del puerto paralelo.
- PIN 4 Clock: Señal de reloj para el PIC, controlada por la señal D1 del puerto paralelo.
- PIN 5 Masa: Señal Vss del PIC, correspondiente a 0V.
- PIN 6 Vcc: Alimentación para futuras ampliaciones y adaptadores. Vcc es +5V siempre que el programador esté encendido.

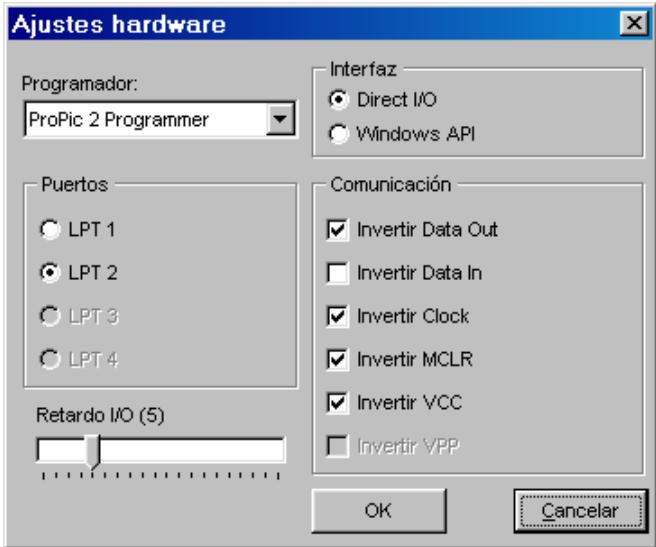
## Utilización.

El programador PP2 está pensado como un dispositivo de uso intensivo. Puede estar encendido y funcionando 24 horas al día sin problemas, y por tanto estará conectado siempre a un puerto paralelo del PC. Sin embargo, si sólo se dispone de un puerto paralelo, habrá que conectarlo y desconectarlo con frecuencia, para lo cual es bueno conocer ciertos detalles. El puerto paralelo no está preparado para conectar y desconectar dispositivos “en caliente”, es decir, que en teoría habría que apagar el PC y el dispositivo cada vez que se vaya a conectar o desconectar. Sin embargo, apagar el PC cada vez es un trastorno, así que el programador se ha diseñado de forma que un par de segundos después de apagarlo no quede ningún tipo de tensión residual en las patillas del conector paralelo. Por otro lado, al utilizar lógica TTL-LS no es previsible que se produzca ningún daño en él si recibe señales estando apagado. En definitiva, que si esperamos unos segundos después de apagar el programador, podemos conectarlo o desconectarlo del puerto paralelo sin problemas. Eso sí, si se conecta o desconecta estando encendido, se pueden producir daños en el puerto paralelo del PC, así que habrá que ser muy prudente en este sentido.

Por otro lado, por motivos parecidos, nunca se debe insertar o extraer un dispositivo programable estando presentes en el programador las tensiones de alimentación o de programación para dicho dispositivo. Por ello se incluyeron en el montaje los LEDs D1 y D6, indicadores respectivamente de que está activada la tensión de alimentación o la de programación. **NUNCA** se debe insertar o extraer un dispositivo programable en el zócalo mientras uno de estos LEDs esté encendido. Tampoco se debe encender o apagar el programador con un dispositivo programable pinchado en el zócalo, ya que en el encendido o apagado se pueden producir señales espúreas que podrían incluso borrar el contenido de la memoria del dispositivo.

Por tanto, la situación correcta para insertar o extraer un dispositivo programable es con el LED D7 (POWER ON) encendido y los LEDs D1 (Vdd) y D6 (Vpp) apagados. Esta situación sólo se consigue con el programador encendido y con el software de grabación correctamente configurado y funcionando.

Si bien cada cual puede utilizar el software de grabación que mejor se adapte a sus necesidades, yo recomiendo ICPROG 1.4, que es freeware y funciona perfectamente (de hecho, no me ha fallado una sola vez con este programador). Se puede bajar de [su website](#). Una vez instalado, en el menú SETTINGS – OPTIONS, en la sección LANGUAGE elegimos ESPAÑOL. A continuación, en el menú AJUSTES escogemos TIPO HARDWARE (se puede hacer directamente pulsando F3) y aparece la ventana de la figura 9. Todas las opciones deben quedar como en la figura 9, salvo el puerto, en el que habrá que marcar el que hayamos utilizado, pulsando OK para finalizar. Si se ha utilizado para U1 el 74LS34 en lugar del 74LS04, deberán dejar todas las señales sin invertir, es decir, todas las casillas en blanco.



**Figura 9**

Podemos verificar, al menos en parte, el funcionamiento del programador, gracias a una utilidad que incluye el programa. Seleccionando en el menú AJUSTES la opción PRUEBA HARDWARE, aparece la pantalla de la figura 10. Si el programador está conectado al puerto paralelo y encendido, y todo está correcto, al marcar la casilla “Habilitar VCC” debería encenderse el LED D1 (Vdd) del programador. Ahora, al marcar la casilla “Habilitar MCLR” debería encenderse el LED D6 (Vpp) del programador, y la casilla “Data In” debe adoptar el mismo estado que marquemos en la casilla “Habilitar Data Out”. No podemos verificar el funcionamiento de la señal Clock si no es usando un voltímetro para medir en el programador, pero si funcionan las demás señales, podemos suponer que esta funcionará también.

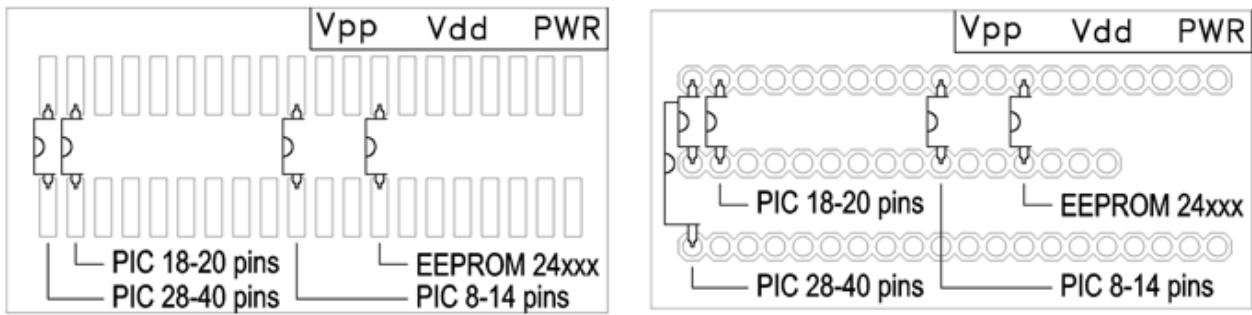


**Figura 10**

Existe una prestación en este software que permite ver la posición en la que habría que insertar el PIC en el zócalo, pero sólo es válida para el ProPic Programmer original, así que no

hagáis caso de ella. Sólo queda seleccionar el modelo de PIC que vamos a grabar o leer, y llevar a cabo las acciones deseadas. El programa trabaja igual con ficheros .BIN ó .HEX. Para grabar un PIC, abrimos el fichero mediante el menú ARCHIVO y seleccionamos PROGRAMAR TODO en el menú COMANDO (o pulsamos F5). Para leer un PIC, seleccionamos LEER TODO en el menú COMANDO (o pulsamos F8) y luego podemos salvarlo a un fichero mediante el menú ARCHIVO. En la página de ICPROG hay instrucciones sobre el programa.

Por último, quiero recalcar la importancia de insertar el dispositivo programable en la posición correcta, dada la tendencia de los transistores T7, T10 y T12 a incinerarse cuando no se hace así. En la figura 11 aparecen las plantillas que indican la posición correcta para cada dispositivo, en el zócalo ZIF, o en el zócalo estándar.



**Figura 11**

# Apéndice D

## Descripción de instrucciones para programar un PIC

### **ADDFW Suma W y f**

Sintaxis: [label] ADDWF f,d

Operandos: d=0 ó d=1; 0 <= f <= 31

Operación: (W) + (f) --> (dest)

Flags afectados: C, DC, Z

Descripción: Suma el contenido del registro W y el registro "f". Si "d" es 0, el resultado se almacena en el registro W. Si "d" es 1 el resultado se almacena en el registro "f".

Ejemplo:

ADDFW REG,0

Antes de la instrucción:

W= 0x17

REF= 0xC2

Después de la instrucción:

W= 0xD9

REG= 0xC2

### **ANDWF W AND f**

Sintaxis: [label] ANDWF f,d

Operandos: d=0 ó d=1; 0 <= f <= 31

Operación: (W) AND (f) --> (dest)

Flags afectados: Z

Descripción: Realiza la operación lógica AND entre el registro W y el registro "f". Si "d" es 0 el resultado se almacena en el registro W. Si "d" es 1, el resultado se almacena en el registro "f".

Ejemplo:

ANDWF REG,1

Antes de la instrucción:

W=0x17

REG= 0xC2

Después de la instrucción:

W=0x17

REG= 0x02

### **ANDLW W AND literal**

Sintaxis: [label] ANDWL k

Operandos: 0 <= k <= 255

Operación: (W) AND (k) --> (W)

Flags afectados: Z

Descripción: Realiza la operación lógica AND entre el registro W y la constante "k". El resultado se almacena en el registro W.

Ejemplo:

ANDLW 0x5F

Antes de la instrucción:

W= 0xA3

Después de la instrucción:

W= 0x03

### **BCF Borra un bit**

Sintaxis: [label] BCF f,b

Operandos:  $0 \leq f \leq 31$ ;  $0 \leq b \leq 7$ ;

Operación:  $0 \rightarrow (f < b >)$

Flags afectados: Ninguno

Descripción: Borra el bit "b" del registro "f".

Ejemplo:

BCF REG,7

Antes de la instrucción:

REG = 0x0A

Después de la instrucción:

REG = 0x47

### **BSF Activa un bit**

Sintaxis: [label] BSF f,b

Operandos:  $0 \leq f \leq 31$ ;  $0 \leq b \leq 7$ ;

Operación:  $1 \rightarrow (f < b >)$

Flags afectados: Ninguno

Descripción: Activa el bit "b" del registro "f".

Ejemplo:

BSF REG,0x0A

Antes de la instrucción:

REG = 0x0A

Después de la instrucción:

REG = 0x8A

### **BTFS C Test de bit y salto**

Sintaxis: [label] BTFS C f,b

Operandos:  $0 \leq f \leq 31$ ;  $0 \leq b \leq 7$ ;

Operación: Salto si  $(f < b >) = 0$

Flags afectados: Ninguno

Descripción: Si el bit "b" del registro "f" es 0, se salta una instrucción y se continúa con la ejecución.

Ejemplo:

COM BTFS REG,1

FALSE GOTO PROCESA\_X

TRUE ° PC= Dirección (CÓM\_)

Antes de la instrucción:

SI REG <1> = 0

PC= Dirección (TRUE)

Después de la instrucción:

SI REG <1> = 1

PC= Dirección (FALSE)

### **BTFSS Test de bit y salto**

Sintaxis: [label] BTFSS f,b

Operandos: 0 <= f <= 31; 0 <= b <= 7;

Operación: Salto si ( $f < b$ ) = 1

Flags afectados: Ninguno

Descripción: Si el bit "b" del registro "f" es 1, se salta una instrucción y se continúa con la ejecución.

Ejemplo:

COM BTFSS REG,6

FALSE GOTO PROCESA\_X

TRUE °

PC= Dirección (COM\_)

Antes de la instrucción:

SI REG <6> = 0

PC= Dirección (FALSE)

Después de la instrucción:

SI REG <6> = 1

PC= Dirección (TRUE)

### **CALL Salto a subrutina**

Sintaxis: [label] CALL k

Operandos: 0 <= k <= 255

Operación: (PC) + 1 --> stack; k --> PC

Flags afectados: Ninguno

Descripción: Salto a subrutina. La dirección de retorno se guarda en el stack. La constante "k" de 8 bits forma la dirección de salto y se carga en los bits <7:0> del PC.

Los bits <10:9> del PC se cargan con los bits <6:5> del registro "STATUS". PC <8> se pone a 0.

Ejemplo:

ORG CALL DESTINO

Antes de la instrucción:

PC = ORG

Después de la instrucción:

PC = DESTINO

### **CLRF Borra un registro**

Sintaxis: [label] CLRF f

Operandos: 0 <= f <= 32

Operación: 00f --> (f); 1 --> Z

Flags afectados: Z

Descripción: Se borra el contenido del registro "f" y el flag Z de estado se activa.

Ejemplo:

CLRF REG

Antes de la instrucción:

REG = 0x5A

Después de la instrucción:

REG = 0x00

Z = 1

### **CLRW Borra el registro W**

Sintaxis: [label] CLRW

Operandos: Ninguno

Operación: 00h --> W; 1 --> Z

Flags afectados: Z

Descripción: El registro de trabajo "W" se carga con 00h. El flag de Z se activa.

Ejemplo:

CLRW

Antes de la instrucción:

W = 0x5A

Después de la instrucción:

W = 0x00

Z = 1

### **CLRWDT Borra el "watchdog"**

Sintaxis: [label] CLRWDT

Operandos: Ninguno

Operación: 00h --> WDT; 1 --> TO; 1 --> PD;

Flags afectados: TO, PD

Descripción: Esta instrucción borra tanto el "watchdog" como el prescaler. Los bits TO y PD del registro de estado se ponen a "1".

Ejemplo:

CLRWDT

Después de la instrucción:

Contador WDT = 0

Prescaler WDT = 0

Bit de estado TO = 1

Bit de estado PD = 1

### **COMF Complementa f**

Sintaxis: label COMF f,d

Operandos: 0 <= f <= 31; d = 0 ó d = 1

Operación: (f) --> (dest)

Flags afectados: Z

Descripción: El contenido del registro "f" se complementa. Si "d" es 0 el resultado "R" se almacena en el registro W. Si "d" es 1, el resultado "R" se almacena en el registro "f".

### **DECF Decremento de f**

Sintaxis: [label] DECF f,d

Operandos: 0 <= f <= 31; d = 0 ó d = 1

Operación: (f) - 1 --> (dest)

Flags afectados: Z

Descripción: Se decrementa en una unidad el contenido del registro "f". Si "d" es 0, el resultado se almacena en W. Si "d" es 1, el resultado se almacena en "f".

Ejemplo:

DEC CONT, 1

Antes de la instrucción:

CONT = 0x01

Z = 0

Después de la instrucción:

CONT = 0x00

Z = 1

### **DECFSZ Decremento y salto**

Sintaxis: [label] DECFSZ f,d

Operandos: 0 <= f <= 32; d = 0 ó d = 1

Operación: (f) - 1 --> d; Salto si R = 0

Flags afectados: Ninguno

Descripción: El contenido del registro "f" se decrementa. Si "d" es 0, el resultado "R" se coloca en el registro W. Si "d" es 1, el resultado "R" se coloca en el registro "f". Si R = 0, se salta la siguiente instrucción y se continúa con la ejecución.

Ejemplo:

COM\_DECFSZ

REG,O

PC = Dirección (COM\_)

GOTO\_NO\_ES\_0

REG = REG - 1

Antes de la instrucción:

SI REG = 0

PC = Dirección CONTINUA

Después de la instrucción:

SI REG != 0

PC = Dirección (COM\_ + 1)

### **GOTO Salto incondicional**

Sintaxis: [label] GOTO k

Operandos: 0 <= k <= 511

Operación: k --> PC --> <8:0>

Flags afectados: Ninguno

Descripción: Se trata de un salto incondicional. Los 9 bits de la constante "k" que forman la instrucción, se cargan en los bits <8:0> del PC y forman la dirección de salto. Los bits <10:9> del PC se cargan con los bits <6:5> del registro de estado.

Ejemplo:

ORG GOTO DESTINO

Antes de la instrucción:

PC = 0

Después de la instrucción:

PC = DESTINO

### **INCF Incremento de f**

Sintaxis: [label] INCF f,d

Operandos:  $0 \leq k \leq 31$ ;  $d = 0$  ó  $d = 1$

Operación:  $(f) + 1 \rightarrow (\text{dest})$

Flags afectados: Z

Descripción: Incrementa en una unidad el contenido del registro "f". Si "d" es 0, el resultado se almacena en W. Si "d" es 1, el resultado se almacena en "f".

Ejemplo:

INCF CONT,1

Antes de la instrucción:

CONT = 0xFF

Z = 0

Después de la instrucción:

CONT = 0x00

Z = 1

### **INCFSZ Incremento de f, si es 0 salta**

Sintaxis: [label] INCFSZ f,d

Operandos:  $0 \leq f \leq 31$ ;  $d = 0$  ó  $d = 1$

Operación:  $(f) + 1 \rightarrow (\text{dest})$ ; Salto si R = 0

Flags afectados: Ninguno

Descripción: Incrementa en una unidad el contenido del registro "f". Si "d" es 0, el resultado se almacena en W. Si "d" es 1, el resultado se almacena en "f". Si R=0, se salta la siguiente instrucción se continúa con la ejecución.

Ejemplo:

COM\_ INCFSZ

REG,1

GOTO\_NO\_ES\_0 PC = Dirección (COM\_)

CONTINUA CONT + 1

Antes de la instrucción:

SI CNT = 0

PC = Dirección CONTINUA

Después de la instrucción:

SI REG != 0

PC = Dirección (COM\_ + 1)

### **IORLW W OR literal**

Sintaxis: [label] IORLW k

Operandos:  $0 \leq f \leq 255$

Operación:  $(W).OR.(k) \rightarrow (W)$

Flags afectados: Z

Descripción: Se realiza la función lógica OR entre el registro W y la constante "k". El resultado se almacena en el registro W.

Ejemplo:

IORLW Ox35

Antes de la instrucción:

W = 0x9A

Después de la instrucción:

W = 0xBF

### **IOWR OR entre W y f**

Sintaxis: [label] IOWR f,d

Operandos: 0 <= f <= 31; d = 0 ó d = 1

Operación: (W).OR.(f) --> (dest)

Flags afectados: Z

Descripción: Realiza la operación lógica OR entre el registro W y el registro "f". Si "d" es 0 el resultado se almacena en el registro W. Si "d" es 1, el resultado se almacena en el registro "f".

Ejemplo:

IOWR REG,0

Antes de la instrucción:

RESULTADO = 0x13

W = 0x91

Después de la instrucción:

RESULTADO = 0x13

W = 0x93

Z = 0

### **MOVF Mover a f**

Sintaxis: [label] MOVF f,d

Operandos: 0 <= f <= 31; d = 0 ó d = 1

Operación: (f) --> (dest)

Flags afectados: Z

Descripción: El contenido del registro "f" se mueve al destino "d". Si "d" es 0, el destino es el registro w. Si "d" es 1, el destino es el propio registro "f". Esta posibilidad permite verificar dicho registro ya que el flag Z queda afectado.

Ejemplo:

MOVF REG,0

Después de la instrucción:

W = REG

### **MOVLW Carga un literal en W**

Sintaxis: label MOVLW k

Operandos: 0 <= f <= 255

Operación: (k) --> (W)

Flags afectados: Ninguno

Descripción: El registro W se carga con el valor de 8 bits expresado mediante la literal "k".

Ejemplo:

MOVLW 0x5A

Después de la instrucción:

W = 0x5A

### **MOVWF Mover W a f**

Sintaxis: label MOVWF f

Operandos: 0 <= f <= 31

Operación: (W) --> (dest)

Flags afectados: Ninguno

Descripción: Mueve el contenido del registro W al registro "f".

Ejemplo:

MOVWF REG

Antes de la instrucción:

REG = 0xFF

W = 0x4F

Después de la instrucción:

REG = 0x4F

W = 0x4F

### **NOP No operar**

Sintaxis: [label] NOP

Operandos: Ninguno

Operación: No operar

Flags afectados: Ninguno

Descripción: No realiza operación alguna. Consumo un ciclo de instrucción.

Ejemplo:

NOP

### **RETLW Retorno, carga W**

Sintaxis: [label] RETLW k

Operandos: 0 <= f <= 255

Operación: (k) --> (W); TOS --> PC

Flags afectados: Ninguno

Descripción: El registro W se carga con los 8 bits de la constante k. El PC se carga con el contenido de la cima stack (TOS): dirección de retorno. Esta instrucción consume dos ciclos.

Ejemplo:

MOVLW CTE ;dependiendo la cte es el número de lugares que va a bajar PC en la tabla

CALL TABLA ;contiene el offset

;de la tabla.

TABLA ;Ahora W tiene el valor de la tabla

ADDWF PC ,1 ;W offset, si w tiene un 3 entonces va a bajar tres lugares el PC

RETLW K1 ;comienza tabla

RETLW K2

RETLW Kn ;Fin de la tabla

Antes de la instrucción:

PC PC

Después de la instrucción CALL:

PC + 1 TOS

TABLA PC

Después de la instrucción RETLW:

TOS PC

Ki w

### **RLF Rota f a la izquierda**

Sintaxis: [label] RLF f,d

Operandos:  $0 \leq f \leq 31$ ; d = 0 ó d = 1

Operación: Rotación a la izquierda de f

Flags afectados: C

Descripción: El contenido del registro "f" se rota una posición a la izquierda. El bit de más peso pasa al carry y el carry se introduce por el bit de menos peso de "f". Si "d" es 0, el resultado se coloca en el registro W. Si "d" es 1, el resultado queda en el registro "f".

Ejemplo:

RLF REG1,0

Antes de la instrucción:

REG1 = 11100110

C = 0

Después de la instrucción:

REG1 = 11100110

W = 11001100

C = 1

### **RRF Rota f a la derecha**

Sintaxis: [label] RRF f,d

Operandos:  $0 \leq f \leq 31$ ; d = 0 ó d = 1

Operación: Rotación a la derecha

Flags afectados: C

Descripción: El contenido del registro "f" se rota a la derecha. El bit de menos peso de "f" pasa al flag carry. El carry se coloca en el bit de más peso de "f". Si "d" es 0, el resultado se almacena en el registro W. Si "d" es 1, el resultado se almacena en "f".

Ejemplo:

RRF REG1

Antes de la instrucción:

REG1 = 11100110

C = 0

Después de la instrucción:

REG1 = 11100110

W = 01110011

C = 0

## **SLEEP Pasa a estado de reposo**

Sintaxis: [label] SLEEP

Operandos: Ninguno

Operación: ooh --> WDT; 0 --> WDT prescaler; 1 --> TO; 0 --> PD

Flags afectados: TO, PD, GPWUF

Descripción: Al salir, activa el bit de estado TO y borra el PD.

El WDT y el prescaler se borran.

Al entrar en el modo SLEEP, se detiene el oscilador.

Ejemplo: SLEEP

## **SUBWF Resta f - W**

Sintaxis: [label] SUBWF f,d

Operandos: 0 <= f <= 32; d = 0 ó d = 1

Operación: (f) - (W) --> (dest)

Flags afectados: C, DC, Z

Descripción: Resta, por el método del complemento a 2, el contenido del registro "f" menos el contenido del registro W. Si "d" es 0, el resultado se almacena en el registro W. Si "d" es 1, el resultado se almacena en el registro "f". Si el resultado es negativo, el carry se pone a 0.

Ejemplo:

SUBWF REG,1

Antes de la instrucción:

REG = 1

W = 2

Después de la instrucción:

REG = 0xFF

W = 0xFF

## **SWAPF Intercambio de f**

Sintaxis: [label] SWAPF f,d

Operandos: 0 <= f <= 31; d = 0 ó d = 1

Operación: (f<3:0>) --> (dest.<7:4>  
(f<7:4>) --> (dest.<3:0>)

Flags afectados: Ninguno

Descripción: Los cuatro bits de más peso del registro "f" se intercambian con los cuatro bits de menor peso del mismo registro. Si "d" es 0, el resultado se coloca en el registro W. Si "d" es 1, el resultado queda en el registro "f".

Ejemplo:

SWAPF REG,0

Antes de la instrucción:

REG = 0x5A

Después de la instrucción:

REG = 0x5A

W = 0xA5

### **XORLW W XOR literal**

Sintaxis: [label] XORLW k

Operandos:  $0 \leq k \leq 255$

Operación: (W).XOR.K  $\rightarrow$  (W)

Flags afectados: Z

Descripción: Realiza la función lógica OR exclusiva (EOR) entre el contenido del registro W y la constante "k" de 8 bits. El resultado se almacena en el registro W.

Ejemplo:

XORLW 0xAF

Antes de la instrucción:

W = 0xB5

Después de la instrucción:

W = 0x1A

### **XORWF W XOR f**

Sintaxis: [label] XORWF f,d

Operandos:  $0 \leq f \leq 31$ ; d = 0 ó d = 1

Operación: (W).XOR.(f)  $\rightarrow$  (dest.)

Flags afectados: Z

Descripción: Realiza la función lógica OR exclusiva (EOR) entre el contenido del registro W y el registro "f". Si "d" es 0, el resultado se almacena en el registro W. Si "d" es 1 el resultado se almacena en el registro "f".

Ejemplo:

XORWF REG,1

Antes de la instrucción:

REG = 0xAF

W = 0xB5

Después de la instrucción:

REG = 0x1A

W = 0xB5

# **Apéndice E**

## Listado de programa

```

;=====
;*****Nombre de archivo: Control_x_vel.asm *****
;   Fecha:          10.05.2004      *
;   Autor:          Miguel Angel Pérez Xochicale    *
;   Compilador:     MPLAB IDE 6.30      *
;*****list p=16f877      ; Tipo de processor

errorlevel 1, -(305)
;Sin advertencias del limite de la página del programa
ERRORLEVEL -302
;Sin warnings de no bank

#include "p16f877.inc"

__CONFIG _HS_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_ENABLE_ON & _DEBUG_OFF

;Declara variable con direccion iniciando en la 0x20
CBLOCK 0x20
dataL

;Variables para la compracion de datos recibidos
Dato_Rx
D_cmpra_A
D_cmpra_B
D_cmpra_C
D_cmpra_D
D_cmpra_E
D_cmpra_F

;Variable para DATO_A "Entra con A
Reb_SAL_y
R_SALIM_y
NPWM_b_y
PWM_y_b
PWM_y_A

Env_O_255

;Variable para DATO_B "Entra con B
Reb_SAL_x
R_SALIM_x
NPWM_b_x
PWM_X_b
PWM_X_A

Env_T_255

;Variable para DATO_C "Entra con C
Env_h_255
Env_s_255

Env_y_255

;Variable para DATO_D "Entra con D
Env_s_255
Env_H_255

Env_x_255

;Variable para DATO_E "Entra con E
Env_S_255XY
Env_H_255XY
Env_h_255XY
Env_s_255XY

Env_m_255

;Variables para retardo 10ms
PDe10
PDe11

;Variables para retardo 500ms
P500
P501
P502

ENDC

```

```

org 0           ;Inicia en = 0000H
    NOP          ;Para compatibilidad de un bootloader
    NOP
    clrf STATUS   ;Banco 0
    goto inicio
org 10

; -----
; INICIALIZACION DE PUERTOS
; -----
inicio

    ;Limpia cada uno de los puertos excepto el RC6 que es el bit de transmision Tx
    banksel PORTA ; banco 0
    movlw b'00000000'
    movwf PORTA
    movlw b'00000000'
    movwf PORTB
    movlw b'01000000'
    movwf PORTC
    movlw b'00000000'
    movwf PORTD
    movlw b'00000000'
    movwf PORTE

    ;Configuracion de puertos como entradas o salidas.
    banksel TRISA ; banco 1
    movlw b'00000000'
    movwf TRISA
    movlw b'00000000'
    movwf TRISB
    movlw b'11000000' ; PONE RC6 & RC7 como entradas
    movwf TRISC
    movlw b'00111111' ; 6 bits de entrada para los limites superior e inferior y home; para cada eje.
    movwf TRISD
    movlw b'11100000' ;Configuracion de puerto E Funcion especial "manejo del puerto paralelo"
    movwf TRISE ;Bit 4 PSP MODE:
                                ;      1 = Funcion en modo de esclavo para puerto
paralelo
                                ;      0 = Funciones para proposito general modo
I/O

    movlw b'00000111' ;bit 3-0 : 011x Entradas y salidas Digitales
    movwf ADCON1 ; puerto A inputs = digital no analogicas

; -----
; DETERMINACION DE VELOCIDAD DE BAUDIOS PARA COMUNICACION CON PC
; -----
; Bits por segundo 19200
; Bits de datos      8
; Paridad            Ninguna
; Stop bits          1
; Flow control       None
;
    movlw d'64'          ; 19200 bps @ 20 Mhz Fosc +0.16 err
    movwf SPBRG
    movlw b'00100100'    ; Baud Values with BRGH = 1
    movwf TXSTA ; Habilita Transmision Asincrona, colocando brgh en alto
    bcf STATUS,RPO ; Banco 0
    movlw b'10010000'    ; Habilita Recepcion Asincrona
    movwf RCSTA

; -----
; Caracteristicas de PWM F=19KHz P=52.6315us
; -----
;
; Tosc = 1/Fosc = 1/20MHz = 50ps
; Caracterizando el periodo de PWM
; pins 16 and 17 son los 2 canales de PWM
;

MOVF CCP1CON,W ;CCP1 en modo PWM
ANDLW 0xF0
IORLW 0x0C
MOVWF CCP1CON

MOVF CCP2CON,W ;CCP2 en modo PWM
ANDLW 0xF0
IORLW 0x0C
MOVWF CCP2CON

;PR2 = (52.6315us/4*50ps*1)) - 1
;PR2 = 262.1578
;PR2 = 255 = 0xFF

MOVLW d'255'
BANKSEL PR2
MOVWF PR2
BANKSEL TMR2

MOVF T2CON,W ;Predivisor a 1:1

```

```

        ANDLW  0xF8          ; PWM
        IORLW  0x00
        MOVWF  T2CON

        MOVF   T2CON,W       ;Predivisor 1:1
        ANDLW  0x07
        IORLW  0x00
        MOVWF  T2CON

                                                ;Configurando PWM a 0
        CLRF  CCP1RL
        CLRF  CCP1R2L
        bcf    CCP1CON,CCP1X      ;Configurando bit 0
        bcf    CCP1CON,CCP1Y      ;Configurando bit 1

        BSF    T2CON, TMR2ON     ;Habilita Timer T2CON<3> = 1
; -----



; ----- ; SUMINISTRANDO UN TIEMPO PARA LA INICIALIZACION ; -----
        clrf  dataL
settle decfsz dataL,F
        goto settle
        movf RCREG,W
        movf RCREG,W
        movf RCREG,W           ; buffer de la cadena recibida

; ----- ;Palabras de control asignadas a las siguientes variables para la comparacion de datos ; -----
        movlw   0x41          ;En ASCII es una A
        movwf  D_cmpra_A
        movlw   0x42          ;En ASCII es una B
        movwf  D_cmpra_B
        movlw   0x43          ;En ASCII es una C
        movwf  D_cmpra_C
        movlw   0x44          ;En ASCII es una D
        movwf  D_cmpra_D
        movlw   0x45          ;En ASCII es una E
        movwf  D_cmpra_E
        movlw   0x46          ;En ASCII es una F
        movwf  D_cmpra_F

; *****
; -----
; ----- *** ; LAZO PRINCIPAL ***
; ----- *** ; ****
; ----- *** ; ****
; ----- *** ; ****
        laxxo  call    Reb_x232      ; ESPERA UN CARACTER
        call    cmpra
        *** ; ;
        call    Env_x232      ; MANDA UN CARACTER
        goto   laxxo
        *** ; ;
; *****
; ***** ; RECIBE UN CARACTER DESDE RS232 Y O GUARDA EN W ; -----
; ----- ; Esta rutina no regresa hasta que un caracter es recibido. ; -----
Reb_x232 nop
        btfss PIR1,RCIF      ; Checa y salta hasta recibir un dato
        goto Reb_x232
        movf RCREG,W         ; SALVA EL DATO EN W
        return
; ----- ; MANDA EL CARACTER EN W VIA RS232 Y ESPERA HASTA QUE TERMINE DE ENVIRSE ; -----
; ----- Env_x232    nop
Buferllen    btfss  PIR1,TXIF      ; PREGUNTA SI EL BUFEER xmit ESTA VACIO?
        goto  Buferllen
        movwf TXREG          ; MANDA EL DATO ALMACENADO EN W
        return

```

```

; -----
; COMPARACION DE DATOS
; -----
;Se realiza con una resta si es igual brinca al dato correspondiente
;si no lo es sigue comparando hasta terminar la rutina compara.

cmpra
    movwf Dato_Rx          ;Mueve el contenido del registro W a Dato_Rx

    ; DATO RECIBIDO = A
    movf Dato_Rx,0          ;Mueve el contenido de Dato_Rx al registro W
    subwf D_cmpria_A,W      ;Resta/compara con numero A
    btfsc STATUS,Z           ;Si los datos son iguales el bit Z de STATUS es igual a 1 (Z=1) ??
    goto DATO_A              ;Si

    ; DAT1 RECIBIDO = B
    movf Dato_Rx,0
    subwf D_cmpria_B,W
    btfsc STATUS,Z
    goto DATO_B

    ; DAT2 RECIBIDO = C
    movf Dato_Rx,0
    subwf D_cmpria_C,W
    btfsc STATUS,Z
    goto DATO_C

    ; DAT3 RECIBIDO = D
    movf Dato_Rx,0
    subwf D_cmpria_D,W
    btfsc STATUS,Z
    goto DATO_D

    ; DAT4 RECIBIDO = E
    movf Dato_Rx,0
    subwf D_cmpria_E,W
    btfsc STATUS,Z
    goto DATO_E

    ; DAT5 RECIBIDO = F
    movf Dato_Rx,0
    subwf D_cmpria_F,W
    btfsc STATUS,Z
    goto DATO_F

    ; Si no es ningun dato que coincide con las funciones
    ; regresa con el valor que recibio al entrar a este ciclo

    movf Dato_Rx,0          ;Mueve el contenido de Dato_Rx al registro W
    return

;=====
;=====
;=====

DATO_A   ;Entra con A para el control del CCP2 control de PWM2 - p16

    movlw 'N'
    call Env_x232

    movlw 0x00
    movwf Reb_SAL_y          ;En ASCII 0
                                ;Palabra de control para salida del CICLO DAT6

REPT_CLO_y:
    call Reb_x232
    movwf R_SALIM_y
    subwf Reb_SAL_y,W
    btfsc STATUS,Z           ;Son iguales (Z=1) ??
    GOTO SAL_CLO_y
    GOTO PROB_LI_y

PROB_LI_y
    BTFSC PORTD,5
    GOTO PROB_LS_y
    GOTO INT_LI_y

PROB_LS_y
    BTFSC PORTD,3
    GOTO PROB_H_y
    GOTO INT_LS_y

PROB_H_y
    BTFSC PORTD,4
    GOTO INICIA_PWM_y
    GOTO INT_H_y

INT_LI_y
    movlw 'i'
    call Env_x232
    movlw 0x80
    movwf CCPR2L
    bsf PORTA,0               ; PWM con 50% de ciclo util para que se mantenga en Home Eje Y
    bcf PORTA,1               ;Habilita bit0 del puerto A (p2) para el motor del eje Y
    GOTO REPT_CLO_y           ;Habilita bit1 del puerto A (p3) para control de Puente H eje Y

INT_LS_y
    movlw 's'
    call Env_x232
    movlw 0x80
                                ; PWM con 50% de ciclo util para que se mantenga en Home Eje Y

```

```

        movwf    CCPR2L          ;Habilita bit0 del puerto A (p2) para el motor del eje Y
        bsf      PORTA,0
        bcf      PORTA,1
        GOTO    REPT_CLO_Y
INT_H_Y
        movlw    'h'
        call    Env_x232
        movlw    0x80
        movwf    CCPR2L          ;Habilita bit0 del puerto A (p2) para el motor del eje Y
        bsf      PORTA,0
        bcf      PORTA,1
        GOTO    REPT_CLO_Y

INICIA_PWM_Y:
        call    Reb_x232          ; PWM 8 bits altos
        movwf    PWM_Y_b
        subwf   Reb_SAL_y,W       ; NPWM - NPbajD = W
        btfsr   STATUS,Z          ;Son iguales (Z=1) ??
        GOTO    SAL_CLO_Y

        call    Reb_x232          ; PWM 2 bits bajos
        movwf    PWM_Y_A
        subwf   Reb_SAL_y,W       ; NPWM - NPbajD = W
        btfsr   STATUS,Z          ;Son iguales (Z=1) ??
        GOTO    SAL_CLO_Y

        movf    PWM_Y_b,0          ;Mueve el contenido de NPWM a registro W
        andlw   0x03
        movwf    NPWM_b_Y          ;w*00000011=W
        rlf     NPWM_b_Y,1          ;Rota a la izquierda y el resultado lo guarda en NPbajW
        rlf     NPWM_b_Y,1          ;Rota a la izquierda y el resultado lo guarda en NPbajW
        rlf     NPWM_b_Y,1          ;Rota a la izquierda y el resultado lo guarda en NPbajW
        rlf     NPWM_b_Y,0          ;Rota a la izquierda y el resultado lo guarda en W
        andlw   0x30
        iorlw   0x0C
        MOVWF   CCP2CON            ;w=00xx1100           x-es es valor cargado

        movf    PWM_Y_A,0          ;Mueve el contenido de NPbajD al registro W
        CCP2R2L
        movwf    PORTB
        bsf      PORTA,0
        bcf      PORTA,1          ;Habilita el Motor 2 del puente H
        bcf      PORTA,1          ;Habilita el puente H
        GOTO    REPT_CLO_Y

SAL_CLO_Y ;Antes de salir de esta rutina se configura el CCPR1L al 50% para el PWM
        movlw    0x80
        movwf    CCPR2L
        bcf      CCP2CON,CCP2X
        bcf      CCP2CON,CCP2Y
        bsf      PORTA,0
        bcf      PORTA,1          ;Habilita bit0 (p2) del puerto A para el motor del eje X
        bcf      PORTA,1          ;Habilita bit1 (p3) del puerto A para control de Puente H eje
        X

        movlw    .255
Lasal_A_O
        movwf    Env_O_255
        movlw    'O'
        call    Env_x232
        decfsz Env_O_255,1
        goto    Lasal_A_O

RETURN
=====
=====
=====
DATA_B ;Entra con B para el control del CCP1 control de PWM1 - p17
        movlw    'E'
        call    Env_x232
        movlw    0x00
        movwf    Reb_SAL_x          ;En ASCII 0
        movwf    Reb_SAL_x          ;Palabra de control para salida del CICLO DAT1

REPT_CLO_X:
        call    Reb_x232
        movwf    R_SALIM_x
        subwf   Reb_SAL_x,W
        btfsr   STATUS,Z          ;Son iguales (Z=1) ??
        GOTO    SAL_CLO_X
        GOTO    PROB_LIX

PROB_LIX
        BTFSC   PORTD,2
        GOTO    PROB_Lsx
        GOTO    INT_LIX
PROB_Lsx
        BTFSC   PORTD,1
        GOTO    PROB_Hx
        GOTO    INT_Lsx
PROB_Hx

```

```

        BTFSC    PORTD,0
        GOTO     INICIA_PWM_X
        GOTO     INT_HX

INT_LIX
        movlw    'I'
        call    Env_x232
        movlw    0x80
        movwf    CCPR1L
        bsf     PORTA,2
        bcf     PORTA,3
        GOTO    REPT_CLO_X

INT_LSX
        movlw    'S'
        call    Env_x232
        movlw    0x80
        movwf    CCPR1L
        bsf     PORTA,2
        bcf     PORTA,3
        GOTO    REPT_CLO_X

INT_HX
        movlw    'H'
        call    Env_x232
        movlw    0x80
        movwf    CCPR1L
        bsf     PORTA,2
        bcf     PORTA,3
        GOTO    REPT_CLO_X

INICIA_PWM_X:
;Checa si el dato recibido es 0 para salirse del ciclo de lo contrario permanece ejecutandolo
        call    Reb_x232      ;!
        movwf  PWM_X_b       ;PWM 8 bits altos
        subwf  Reb_SAL_x,W   ;NPWM - NPbajD = W
        btfsc STATUS,Z        ;Son iguales (Z=1) ??
        goto   SAL_CLO_X

        call    Reb_x232      ;!
        movwf  PWM_X_A       ;PWM 2 bits bajos
        subwf  Reb_SAL_x,W   ;NPWM - NPbajD = W
        btfsc STATUS,Z        ;Son iguales (Z=1) ??
        goto   SAL_CLO_X

;Recibe desde el serie el dato para los dos bit CCP1X y CCP1Y
;carga los bits y termina con el valor que inicialmente recibio

        movf   PWM_X_b,0      ;Mueve el contenido de NPWM a registro W
        andlw  0x03           ;w*00000011=W
        movwf  NPWM_b_X
        ;PWM b=0 b=1
        rlf    NPWM_b_X,1      ;Rota a la izquierda y el resultado lo guarda en NPbajW
        rlf    NPWM_b_X,1      ;Rota a la izquierda y el resultado lo guarda en NPbajW
        rlf    NPWM_b_X,1      ;Rota a la izquierda y el resultado lo guarda en NPbajW
        rlf    NPWM_b_X,0      ;Rota a la izquierda y el resultado lo guarda en W
        andlw  0x30           ;w*00110000=W
        iorlw  0x0C           ;w*00001100=W
MOVWF  CCP1CON
;Datos para EL registro CCP1CON

        movf   PWM_X_A,0      ;Mueve el contenido de NPbajD al registro W
        movwf  CCPR1L
        movwf  PORTB

        bsf    PORTA,2
        bcf    PORTA,3

        goto   REPT_CLO_X

SAL_CLO_X ;Antes de salir de esta rutina se configura el CCPR1L al 50% para el PWM
        movlw  0x80
        movwf  CCPR1L
        bcf   CCP1CON,CCP1X
        bcf   CCP1CON,CCP1Y

        bsf    PORTA,2
        bcf    PORTA,3
;Habilita bit2 (p4) del puerto A para el motor del eje X
;Habilita bit3 (p5) del puerto A para control de Puente H eje X

X
        movlw   .255
        movwf  Env_T_255
        movlw   'T'
        call   Env_x232
        decfsz Env_T_255,1
        goto   Lasal_B_T

RETURN
=====
=====
```

```

;=====
;===== DATO_C ;Entra con C para el control que lleva a Home el eje Y =====
;===== ;Eje Y ;LS - p22 ;H - p27 ;LI - p28 =====
;===== movlw '1' call Env_x232 =====
;===== ChecknevAr1 ;nop =====
;===== btfsc PORTD,4 ;Bit 4 del puertoD (p27) Home Eje Y
;===== goto PWMarrib1
;===== goto LlegaH =====
;===== PWMarrib1 movlw 0xA0 ; PWM con 62.5% de ciclo util para que gire hacia arriba Eje Y
;===== eje y movwf CCPR2L
;===== bsf PORTA,0 ;Habilita bit0 del puerto A (p2) para el motor del
;===== Puente H eje Y bcf PORTA,1 ;Habilita bit1 del puerto A (p3) para control de
;===== Laso_s movwf Env_s_255 movlw .255
;===== movlw 's' call Env_x232
;===== decfsz Env_s_255,1
;===== goto Laso_s =====
;===== ChecknevAb1 ;nop =====
;===== btfsc PORTD,3 ;Bit 1 del puertoD (p22) LSup Eje Y
;===== goto ChecknevAb
;===== eje y movlw 0x60 ; PWM con 37.5% de ciclo util para que gire hacia abajo Eje Y
;===== Puente H eje Y movwf CCPR2L
;===== bsf PORTA,0 ;Habilita bit0 del puerto A (p2) para el motor del
;===== LlegaH movlw .255
;===== Laso_h movwf Env_h_255 movlw .255
;===== movlw 'h' call Env_x232
;===== decfsz Env_h_255,1
;===== goto Laso_h =====
;===== eje y movlw 0x80 ; PWM con 50% de ciclo util para que se mantenga en Home Eje Y
;===== Puente H eje Y movwf CCPR2L
;===== bsf PORTA,0 ;Habilita bit0 del puerto A (p2) para el motor del
;===== bcf PORTA,1 ;Habilita bit1 del puerto A (p3) para control de
;===== Lasal_h_y movwf Env_y_255 movlw .255
;===== movlw 'y' call Env_x232
;===== decfsz Env_y_255,1
;===== goto Lasal_h_y =====
;===== RETURN =====
;===== ====== DATO_D ;Entra con D para el control lleva a Home el eje X =====
;===== ;Eje X ;LS - p19 ;H - p20 ;LI - p21 =====
;===== movlw 'L' call Env_x232 =====
;===== ChecknevArl1 ;nop =====
;===== btfsc PORTD,0 ;Bit 4 del puertoD (p27) Home Eje X
;===== goto PWMarrib1
;===== goto LlegaH1 =====
;===== PWMarrib1 movlw 0xA0 ; PWM con 62.5% de ciclo util para que gire hacia arriba Eje X
;===== eje x movwf CCPR1L
;===== bsf PORTA,2 ;Habilita bit4 del puerto A (p4) para el motor del

```

```

Puente H eje X          bcf           PORTA,3           ;Habilita bit5 del puerto A (p5) para control de
                        btfsc          PORTD,1 ;Bit 1 del puertoD (p22)      LSup Eje X
                        goto           ChecknevAr1
                        movlw          .255
Laso_S                 movwf          Env_S_255
                        movlw          'S'
                        call            Env_x232
                        decfsz         Env_S_255,1
                        goto           Laso_S
ChecknevAb1             movlw          0x60           ; PWM con 37.5% de ciclo util para que gire hacia abajo Eje X
                        movwf          CCPR1L
                        bsf             PORTA,2           ;Habilita bit4 del puerto A (p4) para el motor del
eje X                  bcf             PORTA,3           ;Habilita bit5 del puerto A (p5) para control de
Puente H eje X          btfsc          PORTD,0 ;Bit 1 del puertoD (p27)      Home Eje X
                        goto           ChecknevAb1
LlegaH1                movwf          Env_H_255
                        movlw          'H'
                        call            Env_x232
                        decfsz         Env_H_255,1
                        goto           Laso_H
                        movlw          0x80           ; PWM con 50% de ciclo util para que se mantenga en Home Eje X
                        movwf          CCPR1L
                        bsf             PORTA,2           ;Habilita bit2 del puerto A (p4) para el motor del
eje X                  bcf             PORTA,3           ;Habilita bit3 del puerto A (p5) para control de
Puente H eje X          movwf          Env_x_255
                        movlw          'x'
                        call            Env_x232
                        decfsz         Env_x_255,1
                        goto           Lasal_C_x
                        RETURN
;=====
;=====
;=====

DATO_E                 ;Entra con E para el control que lleva a Home eje X y posteriormente eje Y
                        ;Poniendo eje X y eje Y en HOME

                        ; X                         ; Y
                        ;LS - p19                   ;LS - p22
                        ;H - p20                    ;H - p27
                        ;LI - p21                   ;LI - p28

                        movlw          'M'
                        call            Env_x232
                        ;-----;
                        ;Eje X          PWM1 - p17

ChecknevAr1x            nop
                        btfsc          PORTD,0 ;Bit 0 del puertoD (p19)      Home Eje X
                        goto           PWMarribix
                        goto           LlegaH1x
PWMarribix              movlw          0xA0           ; PWM con 62.5% de ciclo util para que gire hacia arriba Eje X
                        movwf          CCPR1L
                        bsf             PORTA,2           ;Habilita bit2 del puerto A (p4) para el motor del
eje X                  bcf             PORTA,3           ;Habilita bit3 del puerto A (p5) para control de
Puente H eje X          btfsc          PORTD,1 ;Bit 1 del puertoD (p20)      LSup Eje X
                        goto           ChecknevAr1x
                        movlw          .255
Laso_SXY                movwf          Env_S_255XY
                        movlw          'S'
                        call            Env_x232
                        decfsz         Env_S_255XY,1
                        goto           Laso_SXY
ChecknevAb1x              movlw          0x60           ; PWM con 37.5% de ciclo util para que gire hacia abajo Eje X
                        movwf          CCPR1L
                        bsf             PORTA,2           ;Habilita bit2 del puerto A (p4) para el motor del
eje X                  bcf             PORTA,3           ;Habilita bit3 del puerto A (p5) para control de
Puente H eje X          btfsc          PORTD,0 ;Bit 0 del puertoD (p19)      Home Eje X
                        goto           ChecknevAb1x
LlegaH1x

```

```

        movwf   Env_H_255XY .255
Laso_HXY movlw   'H'
                call    Env_x232
                decfsz Env_H_255XY,1
                goto   Laso_HXY

                movlw   0x80 ;PWM con 50% de ciclo util para que se mantenga en Home Eje X
                movwf   CCPR1L
                bsf     PORTA,2 ;Habilita bit2 del puerto A (p4) para el motor del
eje X          bcf    PORTA,3 ;Habilita bit3 del puerto A (p5) para control de
Puente H eje X

;-----;Eje Y PWM2 - p16
ChecknevAry    nop
                btfsc  PORTD,4 ;Bit 4 del puertoD (p27) Home Eje Y
                goto   PWMMarriby
                goto   LlegaHy

PWMMarriby movlw  0xA0 ;PWM con 62.5% de ciclo util para que gire hacia arriba Eje Y
eje y          movwf  CCPR2L
                bsf    PORTA,0 ;Habilita bit0 del puerto A (p2) para el motor del
                bcf    PORTA,1 ;Habilita bit1 del puerto A (p3) para control de
Puente H eje Y

                btfsc  PORTD,3 ;Bit 3 del puertoD (p22) LSup Eje Y
                goto   ChecknevAry

                movlw   Env_s_255XY .255
Laso_sXY      movlw   's'
                call    Env_x232
                decfsz Env_s_255XY,1
                goto   Laso_sXY

ChecknevAby    movlw   0x60 ;PWM con 37.5% de ciclo util para que gire hacia abajo Eje Y
eje y          movwf  CCPR2L
                bsf    PORTA,0 ;Habilita bit0 del puerto A (p2) para el motor del
                bcf    PORTA,1 ;Habilita bit1 del puerto A (p3) para control de
Puente H eje Y

                btfsc  PORTD,4 ;Bit 4 del puertoD (p27) Home Eje Y
LlegaHy
                goto   ChecknevAby

                movlw   Env_h_255XY .255
Laso_hXY      movlw   'h'
                call    Env_x232
                decfsz Env_h_255XY,1
                goto   Laso_hXY

                movlw   0x80 ;PWM con 50% de ciclo util para que se mantenga en Home Eje Y
                movwf   CCPR2L
                bsf    PORTA,0 ;Habilita bit2 del puerto A (p2) para el motor del
eje y          bcf    PORTA,1 ;Habilita bit3 del puerto A (p3) para control de
Puente H eje Y

                movwf   Env_m_255 .255
Lasal_E_m      movlw   'm'
                call    Env_x232
                decfsz Env_m_255,1
                goto   Lasal_E_m

RETURN
;=====
;
```

END

# Referencias

LabVIEW Programming, Data Acquisition and análisis

Jeffrey Y. Beyon

Prentice Hall Hispanoamericana, S.A.,Mexico.

LabVIEW Básico I Introducción, Manual de curso

Software del curso versión 7.0

Edición Agosto 2003

Numero de parte 322307C-01

Microcontroladores <>PIC>>

Diseño practico de aplicaciones.

Jose Angulo Usategui

Susana Romero Yesa

Ignacio Angulo Martinez

Ed. McGraw Hill

Segunda Edicion.

Microcontroladores <>PIC>>

Diseño practico de aplicaciones.

Segunda parte: PIC16F87X

Jose Angulo Usategui

Susana Romero Yesa

Ignacio Angulo Martinez

Ed. McGraw Hill

Paginas de internet relacionadas con la programación del microcontrolador.

<http://www.piclist.com/techref/piclist/index.htm>

[http://www.todopic.com.ar/asm/16F87X\\_ejemplos/](http://www.todopic.com.ar/asm/16F87X_ejemplos/)

<http://www.todopic.com.ar/apuntes/>

Página de internet relacionada con la programación en LabVIEW.

<http://www.labviewgi.cjb.net/>