UNIVERSIDAD DE NAVARRA

ESCUELA SUPERIOR DE INGENIEROS INDUSTRIALES

SAN SEBASTIÁN


# CONTROL Y PROGRAMACIÓN DE ROBOTS: APUNTES DE CμRT

*Club de Microrrobótica de TECNUN*
*Profesor: Dr. EMILIO JOSÉ SÁNCHEZ TAPIA*



San Sebastián, Septiembre 2010

tecnun
CAMPUS TECNOLÓGICO DE LA UNIVERSIDAD DE NAVARRA
NAFARROAKO UNIBERTSITATEKO CAMPUS TEKNOLOGIKOA
Escuela Superior de Ingenieros · Ingeniarien Goi Mailako Eskola

# *AGRADECIMIENTOS*

Los apuntes que acabas de descargarte es un resumen-compendio de la información que me parecía de interés para los alumnos que integran el club de microrrobótica de la Escuela Superior de Ingenieros de San Sebastián.

Por ello, y antes de nada quería agradecer a todos aquellos que de forma directa e indirecta han colaborado a la escritura de estos apuntes y, a su publicación, y que por una u otra razón no estén incluidos en los créditos de este documento.

Ahora, el objeto de los mismos es que los alumnos de la universidad donde imparto clases, sean capaces de introducirse en el apasionante mundo de la microrrobótica.

Por último, estimado lector, animarte a que me ayudes a corregir cualquier fallo que veas y, por supuesto, a re-distribuirlos siempre y cuando conserves, en cada una de las copias, el logotipo de TECNUN.

# *INDICE*

Introducción ..... 7
1.1 Descripción general del microcontrolador ..... 7
1.1.1 Resumiendo: ¿Qué es un microcontrolador? ..... 9
1.2 Los microcontroladores PIC ..... 9
1.2.1 ¿Cómo se conecta el microcontrolador PIC16F84A? ..... 10
1.2.2 El PIC 16F87x ..... 12
1.2.3 El PIC 18F4550 ..... 13
Software de Programación del PIC ..... 17
2.1 MPLAB ..... 17
2.2 Grabación del programa en el microcontrolador. ..... 22
2.3 Uso del ICD2 como debuggeador ..... 24
2.4 Diagrama de Conexiones del ICD2 ..... 25
Sensores ..... 27
3.1 LDR (Light dependent resistor) ..... 27
3.2 CNY70 ..... 28
3.3 Bumpers ..... 30
3.4 Ultrasonidos: ..... 31
3.5 Potenciómetros: ..... 32
3.6 ENCODERS ..... 32
3.6.1 GP2D12: Sensor de medida de distancias por infrarrojos ..... 34
Motores ..... 37
4.1 Motores de corriente continua ..... 37
4.1.1 Método de ensayo de motores DC ..... 38
4.2 SERVOS ..... 39
4.2.1 Modificación de servomotores ..... 41
4.3 Motores Paso a Paso ..... 44
4.3.1 1ª secuencia: paso completo ..... 44
4.3.2 2ª secuencia: medio paso ..... 44
Etapas de potencia ..... 47
5.1 Etapas de potencia ..... 47
5.2 Puente-H ..... 48
Puerto Serie ..... 51
6.1 El Puerto serie del PIC ..... 51
6.1.1 Max232 ..... 52
PCB (Printed Circuit Board) ..... 53
7.1 Introducción ..... 53
7.2 EMpleo de PCBs universales ..... 53
7.3 Placas en ácido (PCB): ..... 54
7.3.1 Normas de seguridad ..... 54

7.3.2 Algunas recomendaciones al respecto son:....54
7.4 Consejos para soldar bien ....55
Algunos trucos y algunos errores:....57
Consideraciones mecánicas ....61
9.1 La estructura o chasis....61
9.2 Tracción y dirección ....61
9.2.1 Criterio de Ackerman....62
Anejos ....65
10.1 Manual Rápido del Eagle....65
10.1.1 Creación de un nuevo proyecto....65
10.1.2 Creación del diagrama esquemático del circuito. ....66
10.1.3 Verificar el circuito....67
10.1.4 Crear el PCB ....67
10.1.5 Impresión de la máscara....70
10.2 Patillaje del PIC16F84 ....71
10.3 Manual de referencia del PIC16F877 ....71
10.3.1 Puertos programables de E/S. ....72
10.3.2 Timers ....74
10.3.3 CAPTURE/COMPARE/PWM ....77
10.3.4 Master Synchronous Serial Port....79
10.3.5 Addressable Universal Synchronous Asynchronous Receiver Transmitter (USART). ....80
10.3.6 Analog/Digital Converter....81
10.3.7 Sleep....83
10.3.8 Reset....84
11.1.1 Bits de Configuración ....85
11.1.2 Mapa de direcciones de datos del PIC 16F87....87
11.1.3 Interrupciones en el PIC16F87 ....87
11.2 Resumen de Instrucciones de programación en C. ....89
11.2.1 Ejemplo de entrada analógica, entrada/salida digital y puerto serie....92
11.2.2 Ejemplo de PWM, conversor A/D, y puerto serie ....92
11.2.3 Código de ejemplo de testeo de la tarjeta genérica basada en el PIC 18F4550 (Figura 5) ....94
11.2.4 Código de básico ejemplo de testeo de la tarjeta genérica basada en el PIC 18F4550 (Figura 5)....95
11.3 Juego de instrucciones en ensamblador....96
11.3.1 Descripción del juego de instrucciones....99

# CAPÍTULO 1

# INTRODUCCIÓN

## 1.1 DESCRIPCIÓN GENERAL DEL MICROCONTROLADOR

Un microcontrolador es un pequeño computador en un chip, pero con sus capacidades y recursos muy limitados. Dependiendo de la aplicación, las características que debe reunir el microcontrolador son diferentes. Ante tal diversidad de requerimientos, de acuerdo con las aplicaciones, los fabricantes ofertan una enorme variedad de microcontroladores, desde los más sencillos y baratos destinados a productos básicos, hasta otros muy complejos capaces de controlar avanzados sistemas.

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, actualmente se impone la arquitectura Harvard. La arquitectura de Von Neumann se caracteriza por disponer de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).

La arquitectura Harvard dispone de dos memorias independientes, una que sólo contiene instrucciones, y otra sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias.

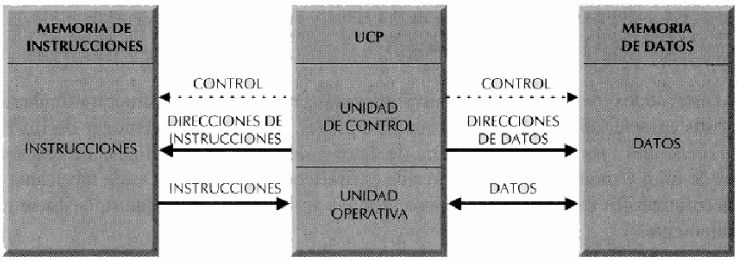



**Figura 1. Estructura Harvard. Relaciones entre las memorias y la CPU.**

Un microcontrolador, en general, consta de cuatro partes: Memoria de programa, Memoria de datos, Procesador y Recursos auxiliares. A continuación se explican brevemente las diferentes partes:

- **Memoria de programa**: Contiene las instrucciones del programa que gobierna la aplicación a la que se destina el microcontrolador. Como siempre se realiza la misma tarea se trata de una memoria no volátil, que

no debe perder la información grabada cuando se desconecta la alimentación.

- **Memoria de datos**: Almacena los datos variables y los resultados temporales, por lo tanto debe permitir lectura y escritura.
- **Procesador**: Es el bloque del computador encargado de interpretar y ejecutar las instrucciones del programa.
- **Recursos auxiliares**: Entre los más conocidos destacan las puertas de entrada y salida digitales; temporizadores; comparadores y capturadores de señales; conversores A/D y D/A; perro guardián y modo de trabajo en reposo.
- **Interrupciones**, etc.

Es conveniente explicar un poco más sobre los recursos auxiliares:

- **Puertas de E/S digitales**: Todos los microcontroladores destinan algunos de sus pines a soportar líneas de E/S digitales. Por lo general, estas líneas se agrupan de ocho en ocho formando Puertas. Las líneas digitales de las Puertas pueden configurarse como Entrada o como Salida cargando un 1 ó un 0 en el bit correspondiente de un registro destinado a su configuración.
- **Temporizadores o "Timers"**: Se emplean para controlar periodos de tiempo (función de temporizador) y para llevar la cuenta de acontecimientos que suceden en el exterior (función de contador). Para la medida de tiempos se carga un registro con el valor adecuado y a continuación dicho valor se va incrementando o decrementando al ritmo de los impulsos de reloj o algún múltiplo hasta que se desborde y llegue a 0, momento en el que se produce una interrupción. Cuando se desean contar acontecimientos que se materializan por cambios de nivel o flancos en alguno de los pines del microcontrolador, el mencionado registro se va incrementando o decrementando al ritmo de dichos impulsos.
- **Conversor A/D (CAD)**: Los microcontroladores que incorporan un Conversor A/D (Analógico/Digital) pueden procesar señales analógicas, tan abundantes en las aplicaciones. Suelen disponer de un multiplexor que permite aplicar a la entrada del CAD diversas señales analógicas desde los pines del circuito integrado.
- **Modulador de anchura de impulsos o PWM**: Son circuitos que proporcionan en su salida impulsos de anchura variable, que se ofrecen al exterior a través de los pines del encapsulado. Son extremadamente útiles para controlar etapas de potencia. Muchos micros tienen este periférico en sustitución del conversod D/A (Digital/Analógico).
- **Puertas de comunicación**: Con objeto de dotar al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, buses de otros microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras normas y protocolos. Algunos modelos, como el PIC 18F4550 usado en este proyecto, disponen de USART, adaptador de comunicación serie síncrona y asíncrona.

### 1.1.1 Resumiendo: ¿Qué es un microcontrolador?

Un microcontrolador es un microprocesador, con periféricos y memoria (ROM/RAM) integrados dentro del mismo chip.

Por eso, en microrrobótica se tiende a usar microcontroladores, porque, aunque son más limitados, para estas tareas sencillas (que, en principio, realizan los microrrobots) son más que suficientes. Además, son mucho más sencillos de programar y de manejar. Al tener muchas funcionalidades encapsuladas en el mismo chip, los circuitos de conexionado externo se simplifican enormemente, y, como consecuencia de ello, se abaratan los costes y el mantenimiento.

## *1.2 LOS MICROCONTROLADORES PIC*

En CMRT hemos utilizado hasta ahora el microcontrolador de Microchip PIC16F84A (de la empresa MICROCHIP) porque es uno de los más sencillos para comenzar, y es muy habitual en los concursos de microrrobótica. Posteriormente usamos el PIC16F877A ya que el 16F84 está muy limitado.

A pesar de todo, cada vez usamos más el PIC 18F4550, pues ofrece mayores prestaciones y la posibilidad de programar tareas de una complejidad superior. No obstante, los conocimientos que se poseean en cualquiera de las familias PIC de Microchip, pueden ser reutilizados y aplicados de manera general sobre cualquiera de las otras familias.

Esto es así, gracias a la política de MICROCHIP de que los chips más potentes mantengan siempre la compatibilidad con sus 'hermanos' menos potentes. Los microcontroladores PIC de Microchip han sido y son tan populares, entre cosas, por lo siguiente:

- Sencillez de manejo: Tienen un juego de instrucciones reducido (RISC); 35 en la gama media.
- Buena información, fácil de conseguir y económica.
- Precio: Su coste es comparativamente inferior al de sus competidores.
- Poseen una elevada velocidad de funcionamiento. Buen promedio de parámetros: Velocidad, consumo, tamaño, alimentación, código compacto, etc.
- Herramientas de desarrollo fáciles y baratas. Muchas herramientas software se pueden recoger libremente a través de Internet desde Microchip (http://www.microchip.com)
- Existe una gran variedad de herramientas hardware que permiten grabar, depurar, borrar y comprobar el comportamiento de los PIC.
- Diseño rápido.
- La gran variedad de modelos de PIC permite elegir el que mejor responde a los requerimientos de la aplicación.

Las características más representativas de los PIC son:

- **Arquitectura**: La arquitectura del procesador sigue el modelo Harvard. En esta arquitectura, la CPU se conecta de forma independiente y con buses distintos con la memoria de instrucciones y con la de datos. Esta

arquitectura permite a la CPU acceder simultáneamente a las dos memorias. Además, propicia numerosas ventajas al funcionamiento del sistema como por ejemplo la segmentación que se explica a continuación.

- **Segmentación**: Se aplica la técnica de segmentación ("pipe-line") en la ejecución de las instrucciones. La segmentación permite al procesador realizar al mismo tiempo la ejecución de una instrucción y la búsqueda del código de la siguiente (Prefetch). De esta forma se puede ejecutar cada instrucción en un ciclo (un ciclo de instrucción equivale a cuatro ciclos de reloj). Las instrucciones de salto ocupan dos ciclos al no conocer la dirección de la siguiente instrucción hasta que no se haya completado la de bifurcación.
- **Formato de las instrucciones**: El formato de todas las instrucciones es de la misma longitud. Las instrucciones de los microcontroladores de la gama baja tienen una longitud de 12 bits (PIC16Cxxx). Los de la gama media tienen 14 bits (PIC16Fxxx), y 16 bits los de la gama alta (PIC17Fxxx) y la gama mejorada (PIC18Fxxx). Esta característica es muy ventajosa en la optimización de la memoria de instrucciones y facilita enormemente la construcción de ensambladores y compiladores.
- **Juego de instrucciones**: Los PIC poseen un procesador RISC (Computador de Juego de Instrucciones Reducido). Los modelos de la gama baja disponen de un repertorio de 33 instrucciones, 35 los de la gama media y casi 60 los de la alta.
- **Arquitectura basada en un "banco de registros"**: Esto significa que todos los objetos del sistema (puertas de E/S, temporizadores, posiciones de memoria, etc.) están implementados físicamente como registros.
- Diversidad de modelos de microcontroladores con prestaciones y recursos diferentes: La gran variedad de modelos de microcontroladores PIC permite que el usuario pueda seleccionar el más conveniente para su proyecto.
- **Herramientas de soporte potentes y económicas**: La empresa Microchip y otras que utilizan los PIC ponen a disposición de los usuarios numerosas herramientas para desarrollar hardware y software. Son muy abundantes los programadores, los simuladores software, los emuladores en tiempo real, Ensambladores, Compiladores C, Intérpretes y Compiladores BASIC.

### 1.2.1 ¿Cómo se conecta el microcontrolador PIC16F84A?

Para tenerlo listo para funcionar, basta con conectarle alimentación, tierra y el circuito oscilador externo que es el que determina la frecuencia de funcionamiento.

El PIC16F84A funciona entre 4 y 10MHz. En CMRT lo hacemos funcionar a 4MHz.

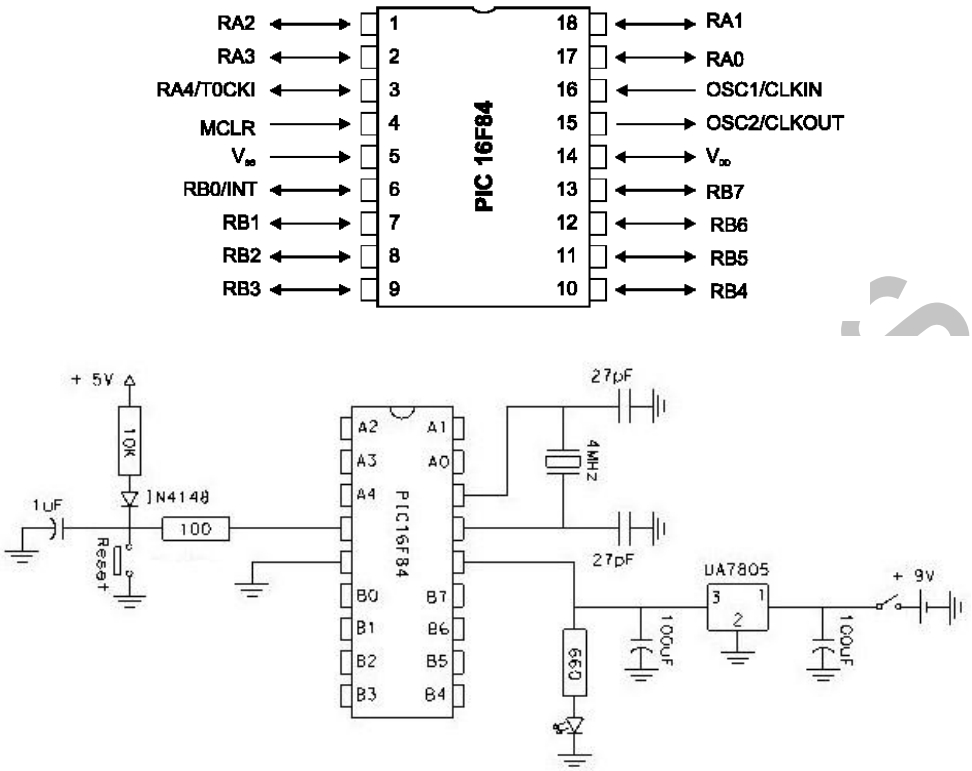



**Figura 2. Conexionado básico del PIC 16F84.**

En la figura se ven (en un chip, el patillaje en sentido contrario a las agujas del reloj, empezando por la superior izquierda, esta parte suele estar marcada por una pequeña muesca en el encapsulado)

- Alimentación:
  1. +5V en el pin 14. Se debe colocar un regulador de tensión que proporcione al micro 5V exactos. Los condensadores son componentes que ayudan al regulador en caso de pequeñas fluctuaciones de tensión.
  2. Se coloca un LED a continuación del regulador con su resistencia de absorción para ver que el micro está encendido. Sin más.
  3. Los 9V son los de la típica pila cuadrada de 9V. Y obviamente un interruptor.
- Tierra:
  1. En el pin número 5.
- Oscilador:
  1. Se coloca en los pines 15 y 16. Esta es la configuración más adecuada. Existen otro tipo de circuitos osciladores menos precisos que no utilizan un cristal de cuarzo como este.
- Botón de RESET:

1. En el PIC16F84A, el pin de reset es la 4. El micro se resetea con un 0 físico, es decir, el pin del reset se pone a tierra. Al resetear el micro, el programa vuelve al comienzo. Las resistencias son indispensables, y el condensador es para evitar los rebotes.

El patillaje restante son todas líneas de entrada/ salida digital (a partir de ahora, diremos líneas de E/S).

## Líneas de E/S: Puerta A y Puerta B

Una puerta es la agrupación de líneas E/S digital. La Puerta A integra 5 líneas E/S, y la Puerta B, 8 líneas.

Son líneas de E/S (es decir, de entrada ó de salida). En algún momento y en algún sitio hay que establecer si es un pin de E/S va a ser una entrada o si por el contrario va a ser una salida.

### 1.2.2 El PIC 16F87x

Las características principales del PIC 16F877 que tiene las siguientes características generales:

- 33 líneas de entrada o salida divididas en 5 puertas, de la A a la E.
- Comunicación serie tipo USART, MSSP e I2C.
- 8 k de Memoria de programa, con palabras de 14 bits. 386 bytes de Memoria RAM de datos y 256 de memoria EEPROM.
- Módulos de captura y comparación y de PWM (CCP).
- Timers, uno de 16 bits y dos de 8, utilizables para diferentes aplicaciones.
- Frecuencia de funcionamiento entre 4 y 20 MHz. En este caso se trabaja con una frecuencia de 4 MHz.
- Un conversor analógico-digital de 10 bits con 8 canales de entrada.
- Voltaje de alimentación entre 2 y 5,5 V en corriente continua.
- Bajo consumo. Menos de 2 mA a 5 V y 5 MHz.

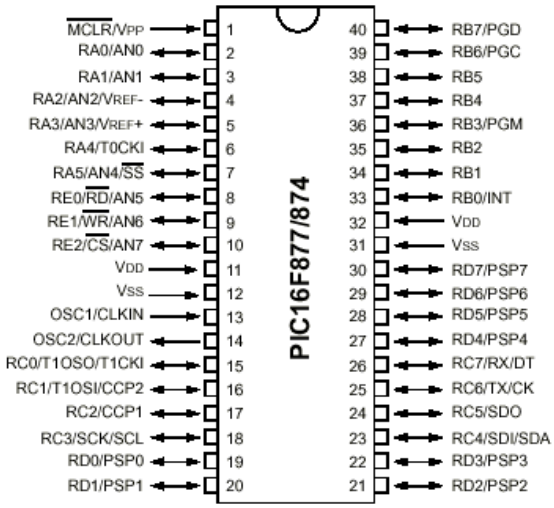



**Figura 3 PIC 16F877.**

En la anterior figura se muestra en diagrama de conexionado del PIC 16F877. Como se observa, el microcontrolador elegido tiene un encapsulado de 40 pines. Para ser un circuito integrado ocupa bastante espacio, cuestión que se ha tenido en cuenta a la hora de diseñar el microrrobot.

Para una mayor referencia de los módulos y arquitectura interna del PIC 16F87x consultar la sección 10.3.

### 1.2.3 El PIC 18F4550

Las características principales del PIC 18F4550 se presentan a continuación:

35 líneas de entrada o salida divididas en 5 puertas, de la A a la E.

- Comunicación serie tipo USART, MSSP, $I^2C$ y USB V.2.0 .
- 32 K de Memoria de programa. 2048 bytes de Memoria RAM de datos y 256 de memoria EEPROM.
- Frecuencia de funcionamiento entre 4 y 48 MHz. En este caso se trabaja con una frecuencia de 48 MHz.
- Un conversor analógico-digital de 10 bits con 13 canales de entrada.
- Posee 4 módulos de Timer o fuentes de reloj internas.
- Módulo de 2 canales para la generación de señales PWM.
- Voltaje de alimentación entre 2 y 5,5 V en corriente continua.
- Bajo consumo. Menos de 2 mA a 5 V y 4 MHz.

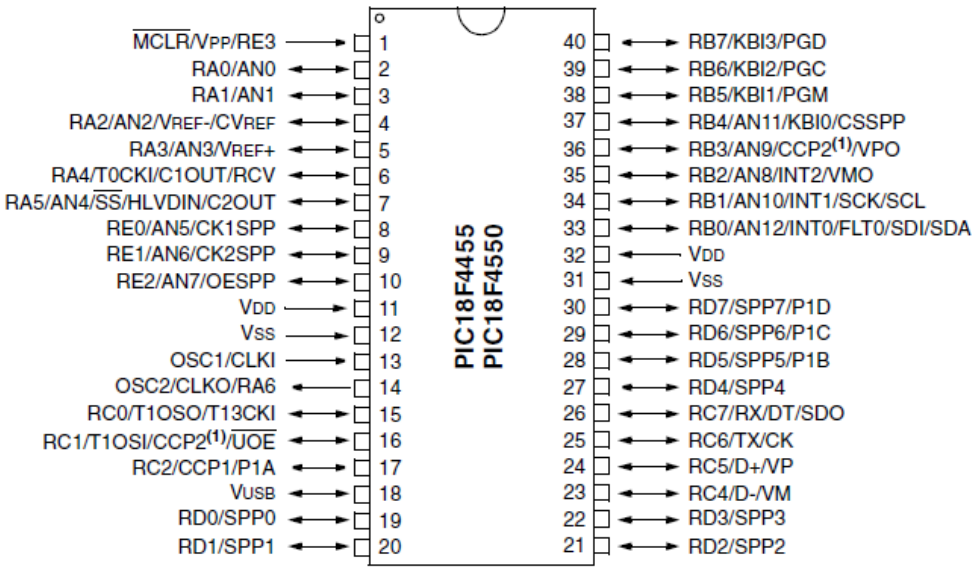



**Figura 4 PIC 18F4550.**

En la Figura 2 se muestra un diagrama de conexionado del PIC 16F4550. Como se observa, el microcontrolador elegido (18F4550) tiene un encapsulado de 40 pines a diferencia del 16F84 que posee tan sólo 18 pines.

A pesar de la notoria diferencia, si observamos con detenimiento los pines de conexionado básico coinciden para ambos encapsulados. De esta manera el diagrama descrito para la conexión del 16F877 es aplicable para el 18F4550, teniendo la precaución de ubicar los pines homólogos entre ambos encapsulados.

La siguiente figura muestra un esquema de interconexionado típico del PIC. El la sección 11.2.3 se muestra un código fuente escrito en C de testeo de la placa.

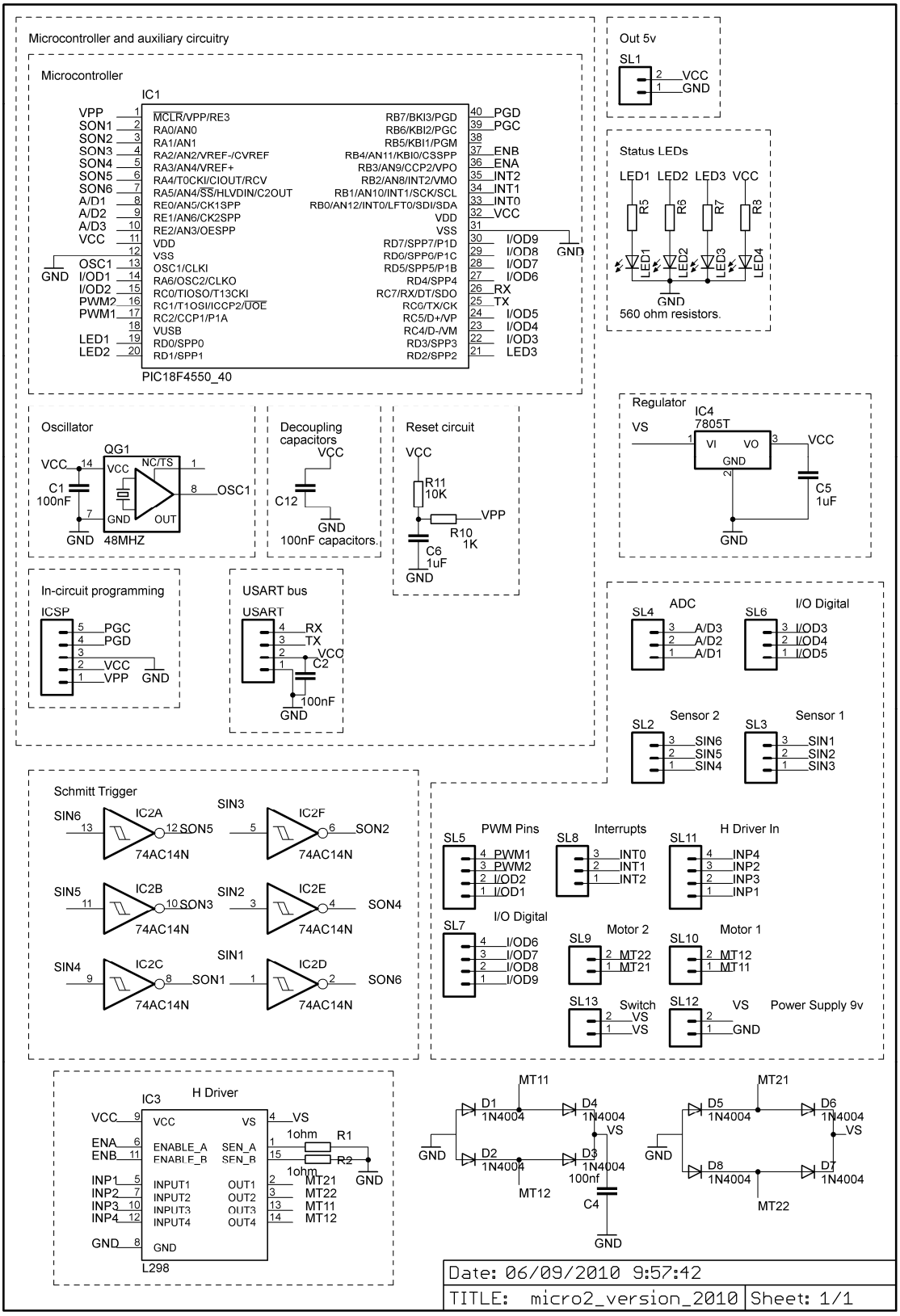



**Figura 5 Interconexionado típico PIC 18F4550.**

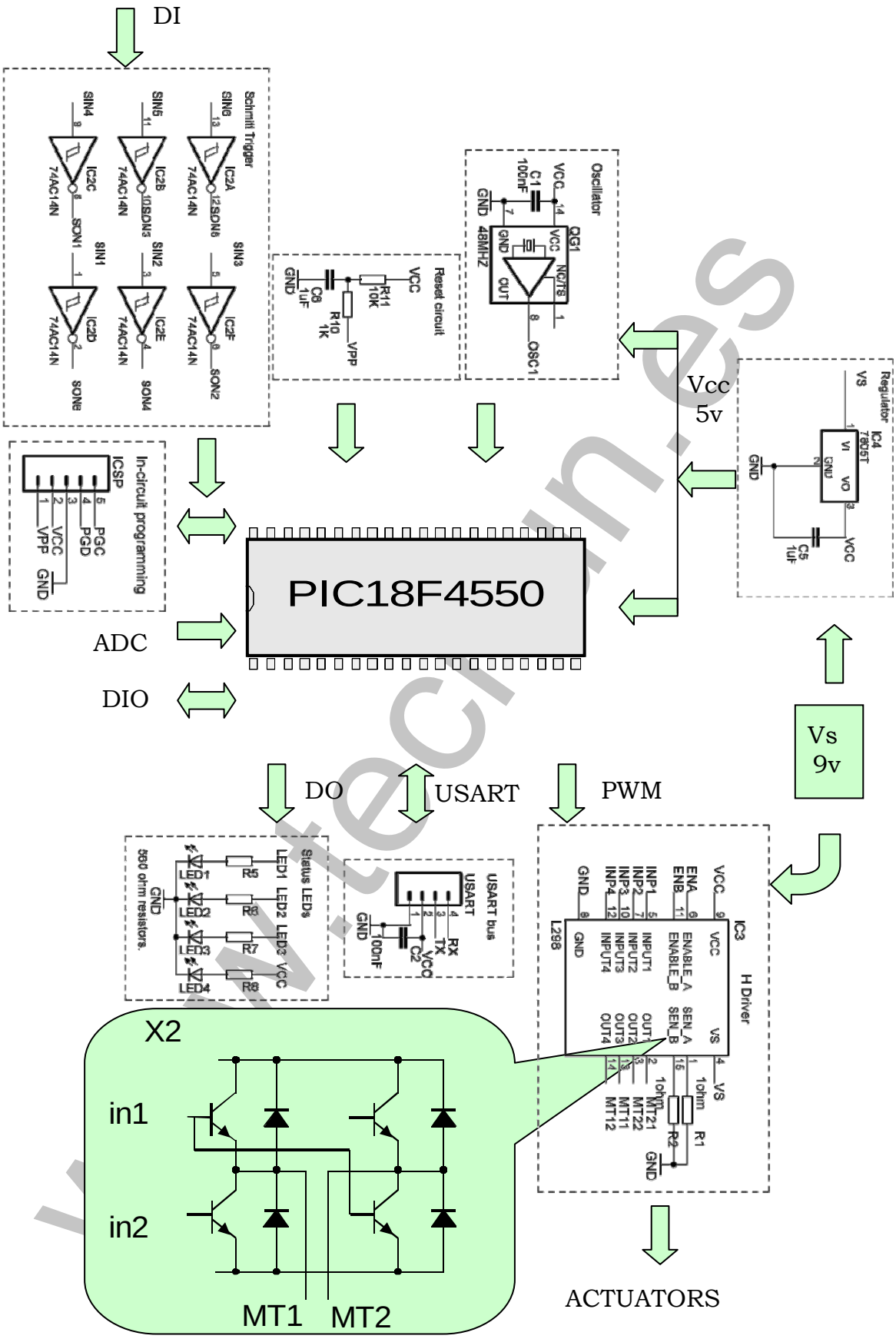



**Figura 6 Diagrama por bloques del interconexionado típico PIC 18F4550.**

# *CAPÍTULO 2*

# *SOFTWARE DE PROGRAMACIÓN DEL PIC*

La programación del PIC puede realizarse en diferentes lenguajes. Sin embargo es recomendable utilizar el lenguaje C. Gracias a un compilador de C la programación se simplifica (programación de alto nivel). Por contra, el rendimiento del código empeora respecto al nemónico (programación de bajo nivel)[1].

Al usar el nemónico, ensamblador o código máquina se eliminan los grados de libertad que introduce el compilador.

## *2.1 MPLAB*

Es un entorno de programación gratuito suministrado por MICROCHIP. Admite la instalación de plug-ins, como el compilador de C que vamos a emplear. De forma nativa, el MPLAB sólo soporta la programación en nemónico.

Para empezar a programar habrá que crearse un proyecto nuevo, para ello:

–Project wizard ⇒ next
–Device: elegir y luego ⇒ next
- 16F84A
- 16F877
- 18F4550

–Compilador ⇒ CCS C compiler ⇒ next
–Name y directory ⇒ next ⇒ finish
–File new (*.c que vamos a programar)
–Add file

[1] En el caso de la familia 18Fxx, esta afirmación no es del todo correcta ya que su diseño está pensado para ser programado en C con eficiencia.

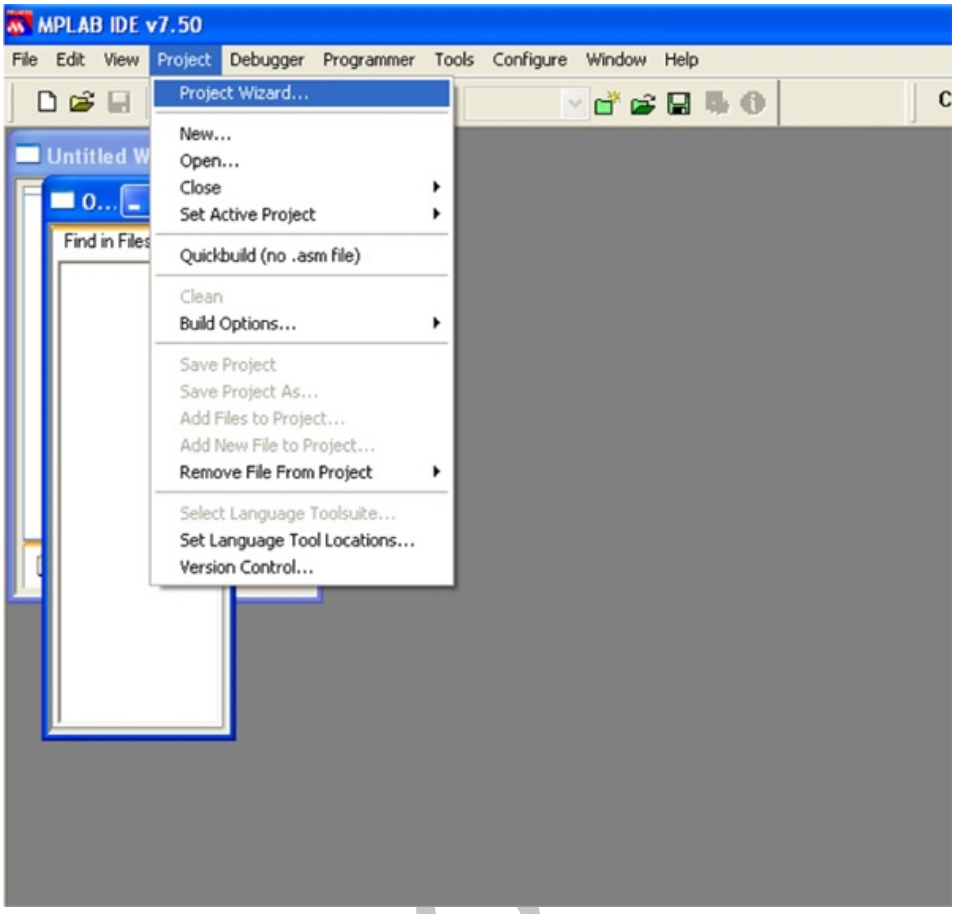



**Figura 7 Inicio del "project wizard" de MPLAB.**

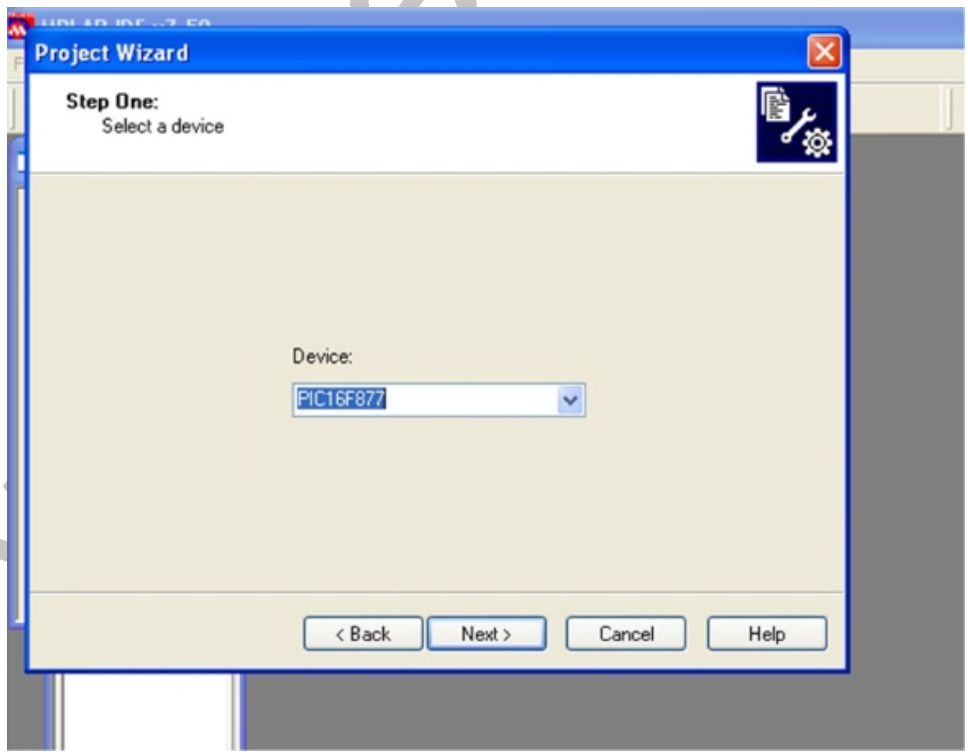



**Figura 8 Selección del dispositivo en el "project wizard" de MPLAB.**

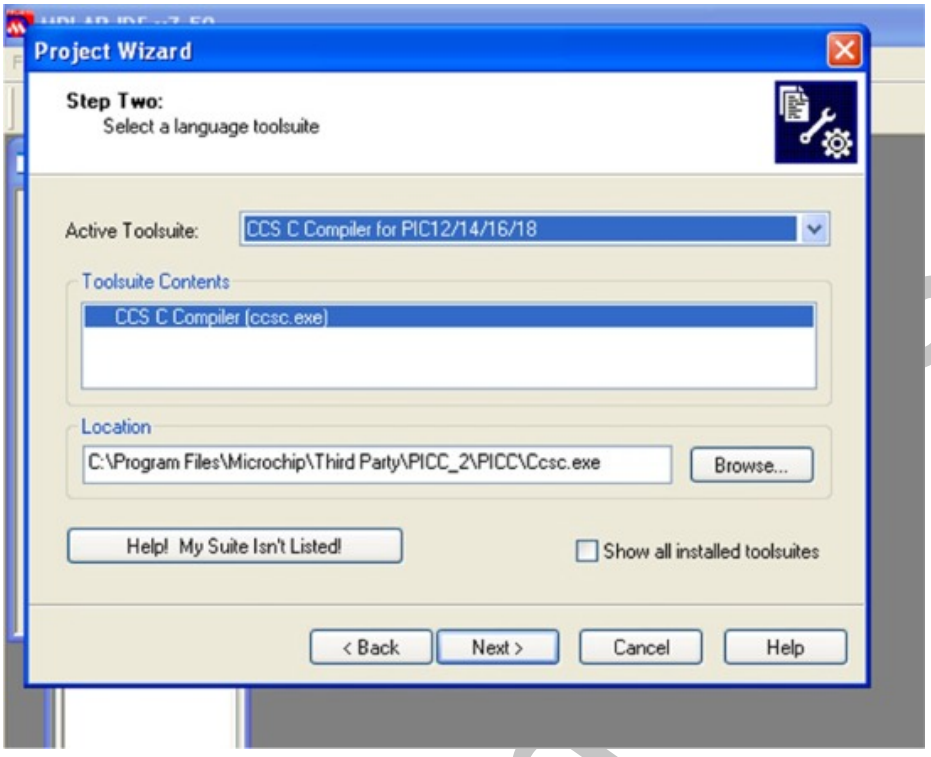



**Figura 9 Selección del compilador de C en el "project wizard" de MPLAB.**

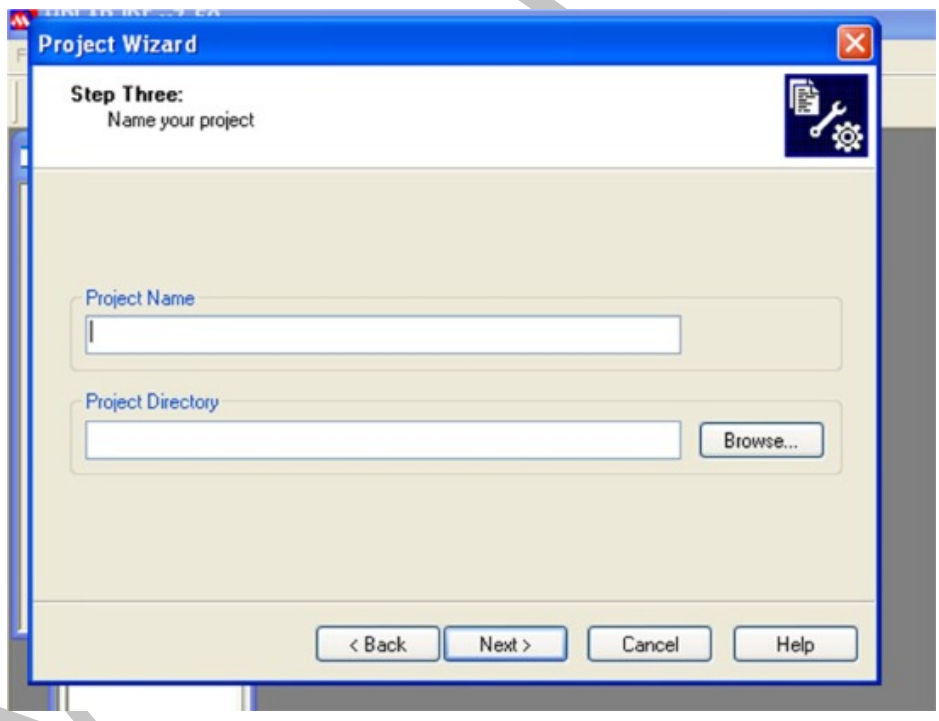



**Figura 10 Definición del nombre del proyecto y directorio de almacenaje en el "project wizard" de MPLAB.**

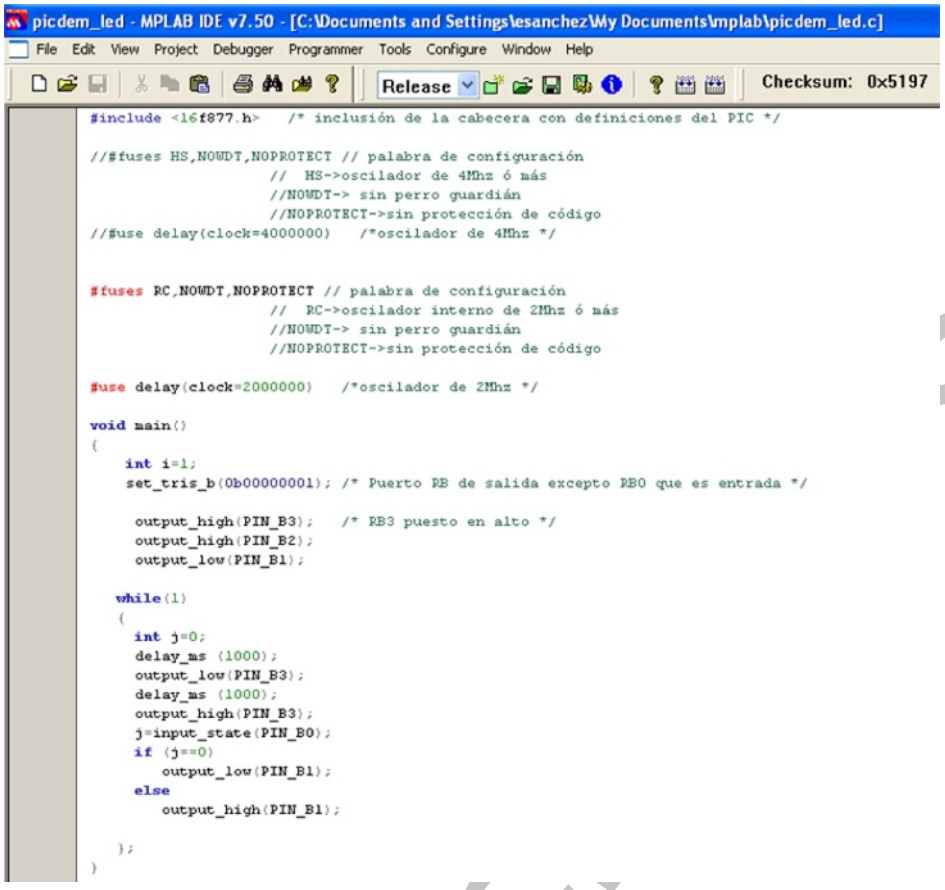



**Figura 11. Pantalla de programación del MPLAB.**

Primero hay que incluir el fichero 16f84.h (o el 18F4550.h o el PIC que se vaya a usar); este fichero define algunas constantes que serán útiles durante el proceso de escritura del código. Lo siguiente a incluir es el valor del oscilador (los Mhz del oscilador contectado), y finalmente hay que definir la posición de memoria de las puertas a y b. Y a partir de aquí ya se puede comenzar programar.

Un resumen de las funciones de C más importantes pueden consultarse en la sección 11.2.

Una vez que se tiene listo el programa, hay que compilar con la opción:

- Project→ Build All

Si el proceso de compilación no da errores, aparecerá una pantalla informando que se ha creado un fichero *.hex. Este fichero es el que contiene el programa que deberá ser grabado en el micro.

Además, con este fichero *.hex, se puede simular el funcionamiento del programa antes de grabarlo sobre el micro:

Se definen unos botones que permiten introducir impulsos en los pines del PIC. Se escoge un botón para cada pin, y también se escoge qué tipo de impulso se va a aplicar: pulso, cero, uno, ó cambio.

- Debugger → Select Tool → MPLAB SIM
- Debugger → Stimulus → New Workbook

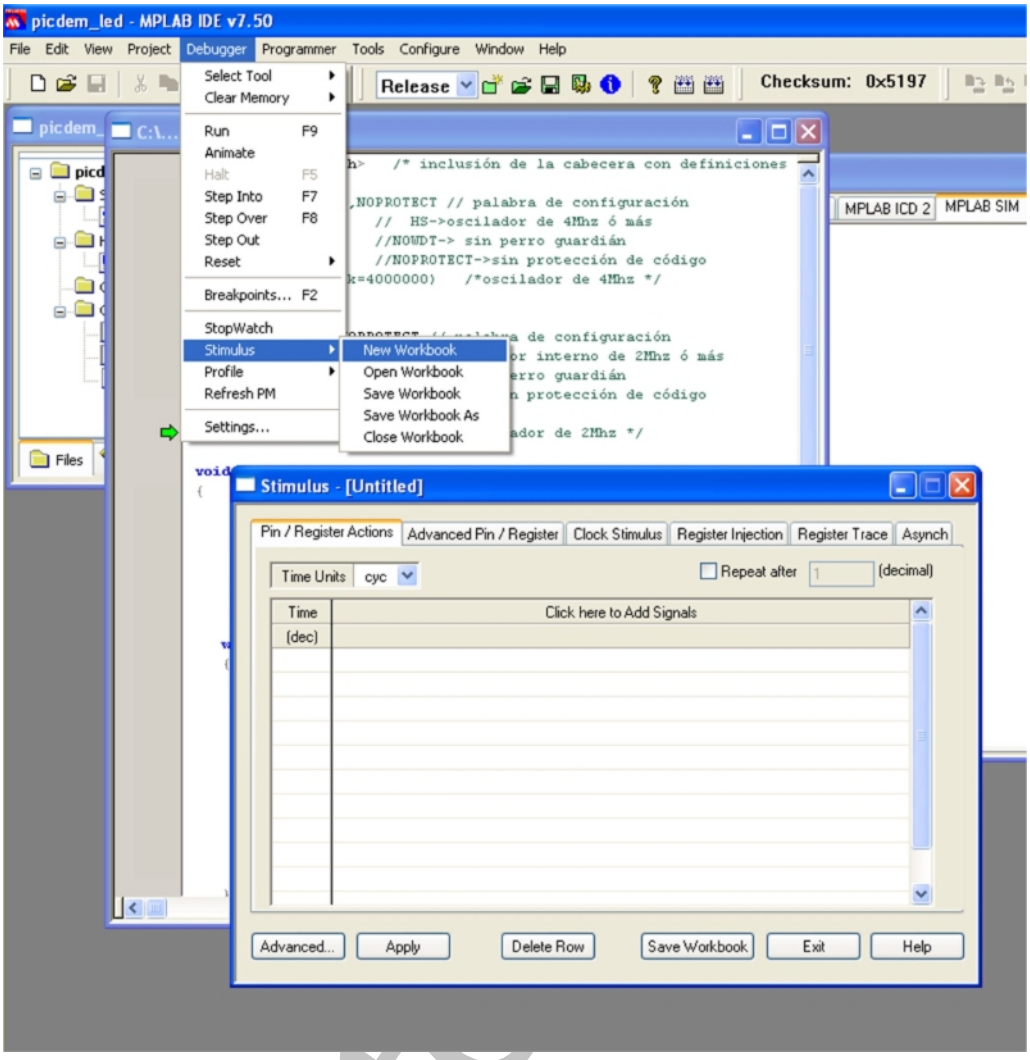



**Figura 12. Configuración de estímulos del simulador del Mplab.**

Una vez que se tiene la ventana mostrada en la Figura 12, se introducen los estímulos deseados. Para ver las señales, del menú 'view' se selecciona lo que se desea ver (EEPROM, Hardware Stack, Locals, Program Memory, etc).

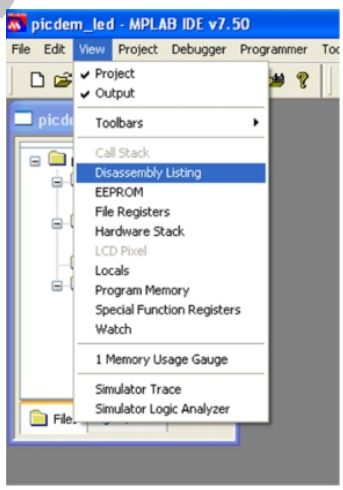



**Figura 13. Ventanas disponibles en el simulador del Mplab.**

Un punto interesante es que podemos ver las variables en decimal, hexadecimal o binario. Muchas veces es más interesante ver en binario (por ejemplo, las salidas o las entradas)

En este punto ya podemos simular el programa:

- Debugger → Run

Las variables cambian a color rojo en el momento en que cambian, y una barra negra irá recorriendo el programa de arriba abajo indicando en qué parte se encuentra. Por supuesto, también existen los habituales break point, step over, etc...

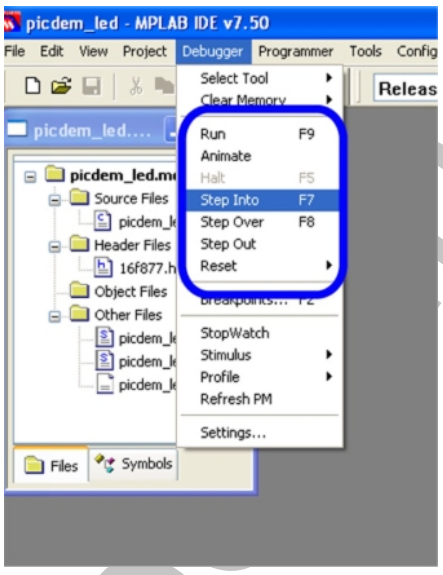



**Figura 14 Control de ejecución del simulador del Mplab.**

## *2.2 GRABACIÓN DEL PROGRAMA EN EL MICROCONTROLADOR.*

Cuando ya se ha generado el fichero *.hex que se desea programar sobre el micro, se procede a la grabación del micro. Para ello se cuenta con un periférico conectado al PC llamado ICD2.


**Figura 15. Programador de PICs ICD2.**

Primero de todo, hay que conectar físicamente el ICD2 al PC por medio de un puerto USB. Posteriormente, declarar en el MPLAB que el programador está conectado y conectarlo a nivel de software .

- Programmer→MPLAB ICD2
- Programmer→connect

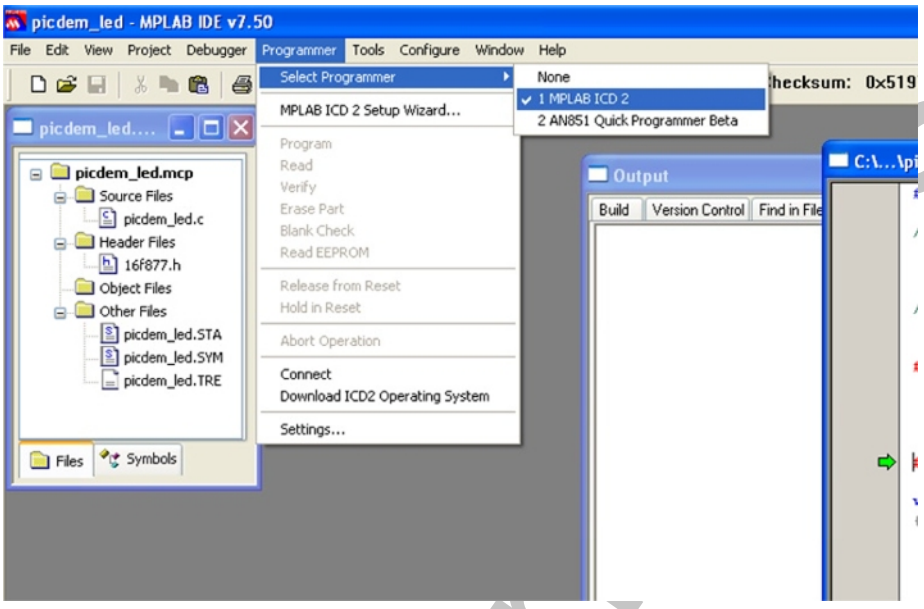



**Figura 16. Declaración del programador ICD2 en el MPLAB.**

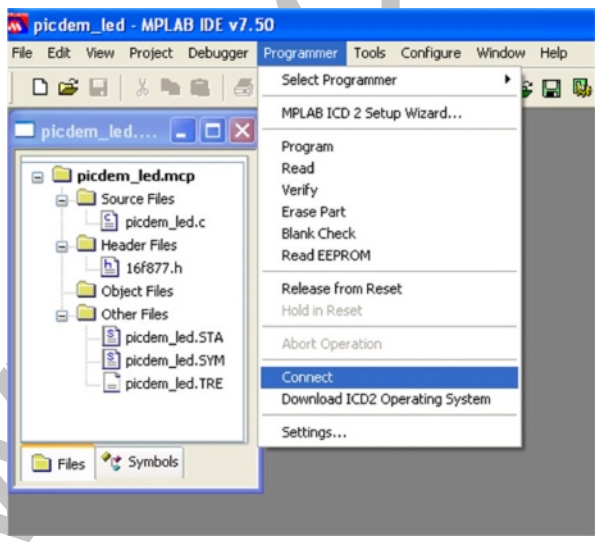



**Figura 17. Conexión del programador ICD2 en el MPLAB.**

Si todo va bien, saldrá una ventana explicando que el programador se encuentra bien conectado y disponible para programar el PIC en cuestión.

Si ha habido algún problema (el programador no está bien conectado o el PIC conectado no se corresponde al que habíamos configurado al crear el proyecto), se emitirá un mensaje de error como el de la figura siguiente:

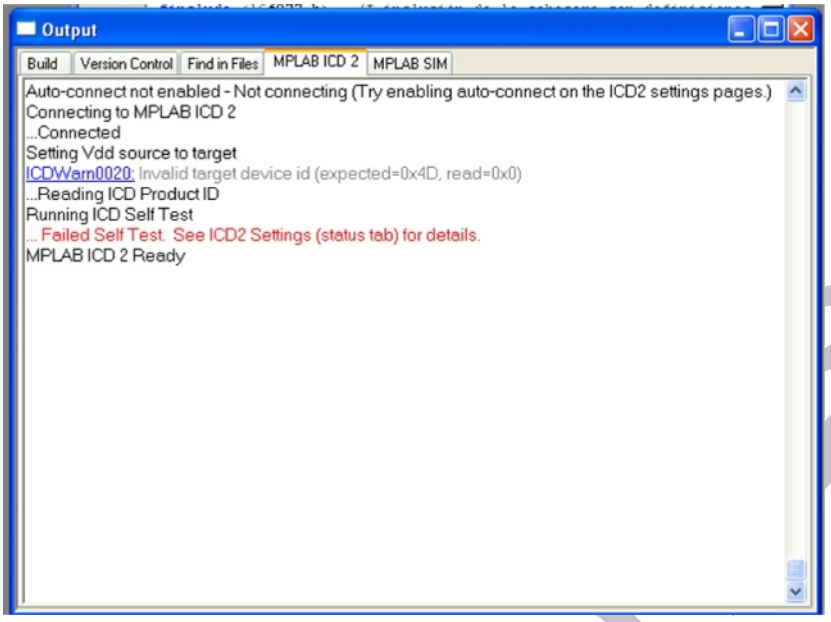



**Figura 18. ICD2 emitiendo un error de conexión con el PIC.**

Una vez resuelto todos los problemas de concexión del ICD2 con el PIC, se podrá descargar el programa con las opciones típicas de cualquier programador:

- Programmer→Program
- Programmer→Read
- Programmer→Verify
- Programmer→Erase Part
- Programmer→Blank Check
- Programmer→Read EEPROM

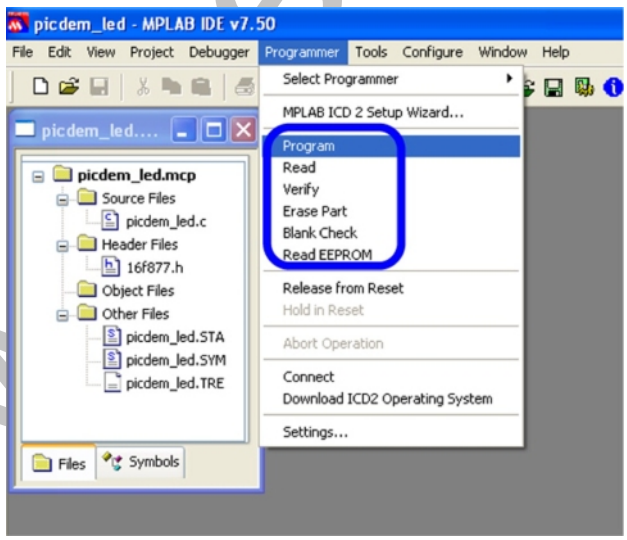



**Figura 19. Opciones del programador ICD2.**

## *2.3 USO DEL ICD2 COMO DEBUGGEADOR*

El ICD2, además de usarse como programador, también se puede usar para "debuggear" (depurar) el programa ejecutándose en el PIC real. Para eso deberemos ir al menú "Debugger" y seleccionar como herramienta el MPLAB ICD 2 en lugar de MPLAB SIM. El manejo es similar.

- Debugger → Select Tool → MPLAB ICD 2

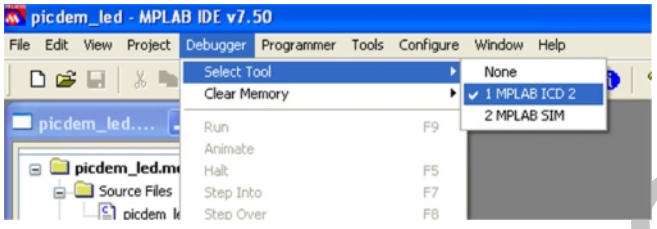



**Figura 20. Usando el ICD2 como "debuggeador".**

## *2.4 DIAGRAMA DE CONEXIONES DEL ICD2*

La siguiente figura muestra el diagrama de conexiones completo que tiene que implementarse para que el ICD2 pueda programar un PIC. Nótese que este diagrama es genérico y vale para cualquier PIC, ya que muestran los nombres de los pines del PIC (nótese que la nomenclatura de los pines es estándar para todos los PICs).

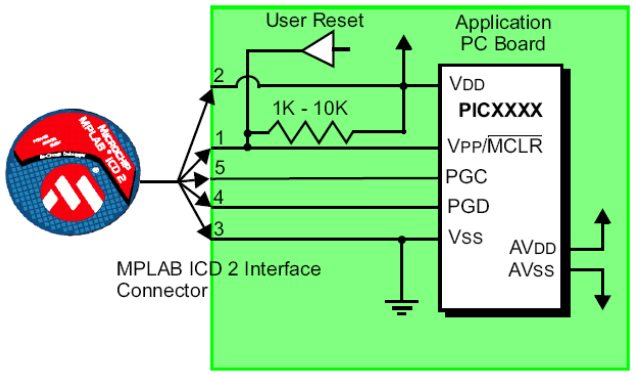



**Figura 21. Diagrama de conexiones PIC-ICD2.**

En el caso de que no se quiera usar el ICD2 como "debuggeador" on-line, sólo como programador, no hace falta que el circuito cuente con un oscilador:

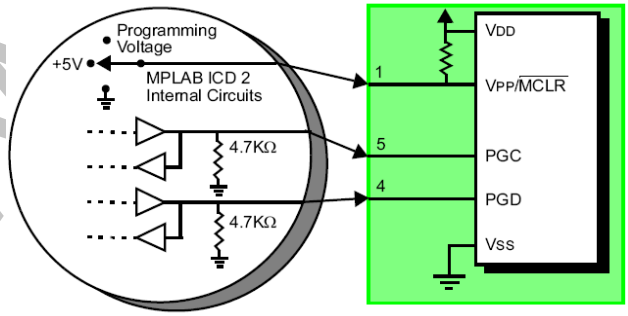



**Figura 22. Diagrama de conexiones PIC-ICD2 para usar el ICD2 sólo como programador.**

En el caso de que se quiera aprovechar la capacidad del ICD2 como "debuggeador", además de conectar el oscilador, se deben cumplir los siguientes requisitos:

- El ICD 2 debe conectarse a los pines VPP, PGC, PGD, VSS y VDD tal y como se muestra en la Figura 22.
- El PIC debe tener conectado correctamente un oscilador
- Si por cualquier razón el PIC no ejecuta instrucciones, el ICD2 no podrá hacer ninguna labor de debug
- Los bits de configuración del PIC tienen que ajustarse correctamente:
  - Oscillator Configuration bits→ lo que corresponda según el oscilador conectado al PIC (RC, XT, etc.),
  - Watchdog Timer deshabilitado
  - Code Protection deshabilitado
  - Dejar un espacio de memoria libre para el programa "debug executive", que ocupa unas 120 palabras de la memoria de programa.

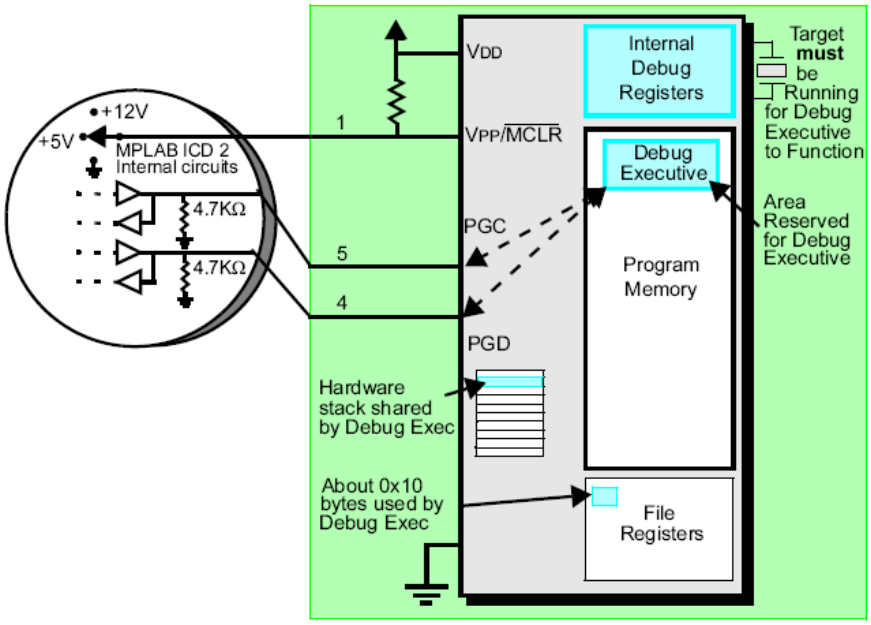



**Figura 23. Diagrama de conexiones PIC-ICD2 para usar el ICD2 como debuggeador.**

# *CAPÍTULO 3*

# *SENSORES*

Todo robot que se precie debe estar dotado de un buen número de sensores para obtener una información precisa del medio que le rodea y así poder cerrar un lazo de control. Existen todo tipo de sensores:

- Sensores de luz (LDR,...)
- Sónar.
- Sensores de infrarojos.
- Sensores de impacto (búmper).
- Sensores de movimiento.
- Cámaras de vídeo
- Brújula electrónica
- Sensores de giro de las ruedas.
- Sensor de desplazamiento 2D (como el mecanismo de un ratón)
- Sensor de nivel de baterías.
- Encoders.
- Y muchos más...

En microrrobótica se pueden emplear todos los sensores que se deseen. Pero los más habituales por ser los más comunes son:

## *3.1 LDR (LIGHT DEPENDENT RESISTOR)*

...o dicho de otra forma, fotorresistencias.

Estos componentes electrónicos varían su resistencia al incidir la luz sobre ellos. Esto permite detectar la luz, y también se puede emplear para detectar grandes contrastes de color (la típica línea blanca sobre fondo negro).

Presentan el inconveniente de que tardan un tiempo en cambiar su valor, y por eso no son del todo adecuados para seguir una línea en el caso de un microrrobot rastreador.

El circuito de conexión típico se encuentra en la Figura 24.

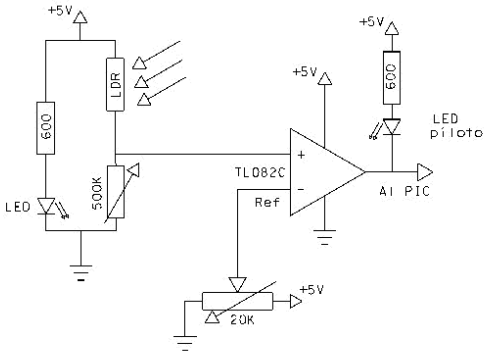



**Figura 24. Esquema de conexión de una LDR.**

En el esquema se puede ver de izquierda a derecha:

- Un Led para iluminar justo debajo de la fotorresistencia para hacerlo más robusto a cambios de luminosidad. Así se pueden tapar los sensores para que no les dé la luz exterior que podría variar.
- Un divisor de tensiones formado por la fotorresistencia y un potenciómetro. Este potenciómetro permite regular la sensibilidad.
- Un comparador con otro potenciómetro como divisor de tensiones. Este también permite regular la sensibilidad. Al tener dos puntos de regulación, permite mayor rango de regulación, pero también es mucho más complicado ajustarlo y a cambio, mucho más fácil que se desajuste. Se comparan las tensiones, y saca una señal igual que la de alimentación en función de cuál sea mayor.
- El Led piloto, que parece una cosa sin importancia, es casi podríamos decir imprescindible. Si no se coloca un Led para ver qué señal proporciona el sensor, será necesario medir constantemente con el voltímetro, lo cual puede resultar bastante engorroso.

Una última observación es el valor de las resistencias. Éste debe ser bien grande para que circule poca intensidad y consuma poco nuestro sensor.

Este es un circuito que opera con una LDR (que proporciona una señal analógica) y lo convierte en digital, que es lo que entiende el PIC16F84. Sin embargo, si lo que se usa es un PIC16F87X (o similar), podría conectarse directamente a una de sus entradas analógicas.

Existen otras muchas maneras de emplear la fotorresistencia: la más frecuente es aprovechar el cambio de la resistencia para disparar un transistor.

### *3.2 CNY70*

El CNY70 es un sensor comercial de infrarrojos por reflexión. Consta de un diodo emisor de infrarrojos y un fototransistor, ambos apuntando en la misma

dirección, de manera que cuando se enfrenta a una superficie suficientemente reflectora, la luz infrarroja refleja y activa el fototransistor.

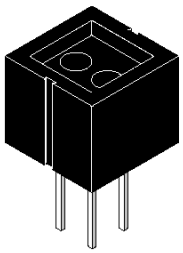


**Figura 25. El CNY70.**

El CNY70 tiene cuatro pines de conexión. Dos de ellos se corresponden con el ánodo y cátodo del diodo emisor, y las otras dos se corresponden con el colector y el emisor del fototransistor. En el esquema siguiente se ve el conexionado de este sensor para un microrrobot que tenga que distinguir un fondo blanco de uno negro:

ATENCIÓN: ¡Esto es un es un esquemático, y los pines no están en este orden en el sensor!

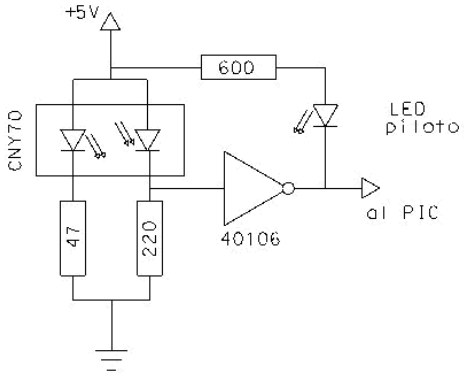



**Figura 26. Esquemático de conexionado del CNY70 al PIC.**

Los pines del sensor de infrarrojos CNY70, mirando a los sensores son:

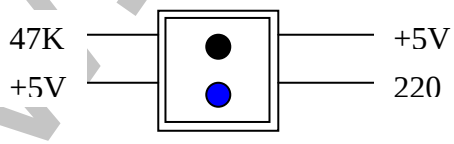



**Figura 27. Pines del CNY70.**

Y mirando desde los pines (*colector* en el lado de las letras blancas):

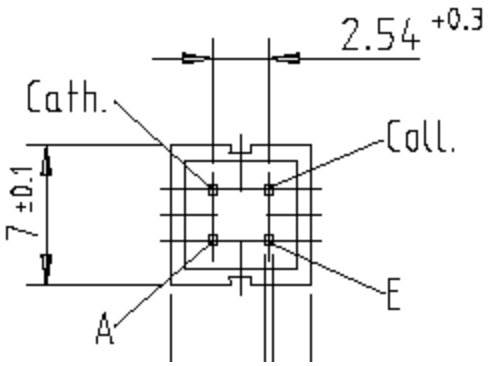



**Figura 28. Layout del CNY70.**

El esquema funciona de la siguiente manera:

- El LED emisor de infrarrojos emite constantemente
- El fototransistor se abrirá cuando refleje suficiente luz (por ejemplo en una superficie clara)
- La entrada del circuito 40106 se pondrá a 5V (un 1 lógico) cuando esto ocurra. Mientras no sea así, permanece a 0V (un 0 lógico)

El 40106 es un Smith-Trigger. Es decir, un integrado que nos permite "pasar a limpio" la señal que nos da el sensor, quitándole ruido y haciendo que quede una señal digital de calidad que pueda ser procesada por el PIC y resto de lógica TTL. Además, invierte la señal (Si el sensor da unos 1.4V, otras veces 1.2V ó 0.9V, al pasar el 40106 tendremos 5V exactos, y si nos da el sensor sus 4.5V, 4.1V nos dará al pasar el Smith-Trigger 0V exactos)

Y no podía faltar el LED piloto, que es completamente imprescindible.

## *3.3 BUMPERS*

...o pulsadores "parachoques". Son sencillamente interruptores que se colocan normalmente en el exterior del microrrobot y se cierran al chocar con algo. Son muy similares a los botones del ratón.

También se suelen poner como sensores de final de carrera (parecido a la luz de la nevera, que se enciende al abrir la puerta gracias al interruptor que suele haber cerca de las bisagras.

Al ser mecánicos, se produce un pequeño chisporroteo transitorio al pulsarlos (rebotes) que da lugar a oscilaciones en la señal. Hay tres maneras diferentes de evitar estos rebotes:

Con un circuito FLIP-FLOP, que permanece activo aunque haya rebotes.

Por programación, determinando un tiempo de espera de unos pocos milisegundos antes de mirar el valor del pin. En ese tiempo, se habrá terminado el transitorio.

Con un circuito anti-rebotes formado por un condensador y unas resistencias. Es el más sencillo, y funciona bien.

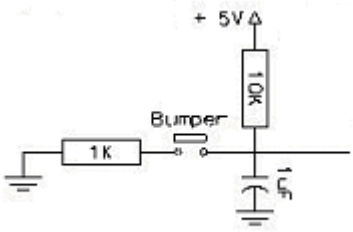



**Figura 29. Conexionado de bumpers.**

Cuando el bumper está abierto, el condensador está cargado y se introduce un nivel alto por la línea de entrada. Al activar el bumper, el condensador se descarga a través de la resistencia de 1k y aplica un nivel lógico bajo. Al desactivar el bumper, el nivel alto no se alcanza hasta que se cargue el condensador con la tensión positiva de alimentación a través de la resistencia de 10k.

### *3.4 ULTRASONIDOS:*

Este sensor consta de un circuito bastante más complicado que los anteriores, pero lo mencionamos porque aparece con mucha frecuencia en las competiciones de microrrobótica, especialmente en las pruebas de sumo y laberinto.

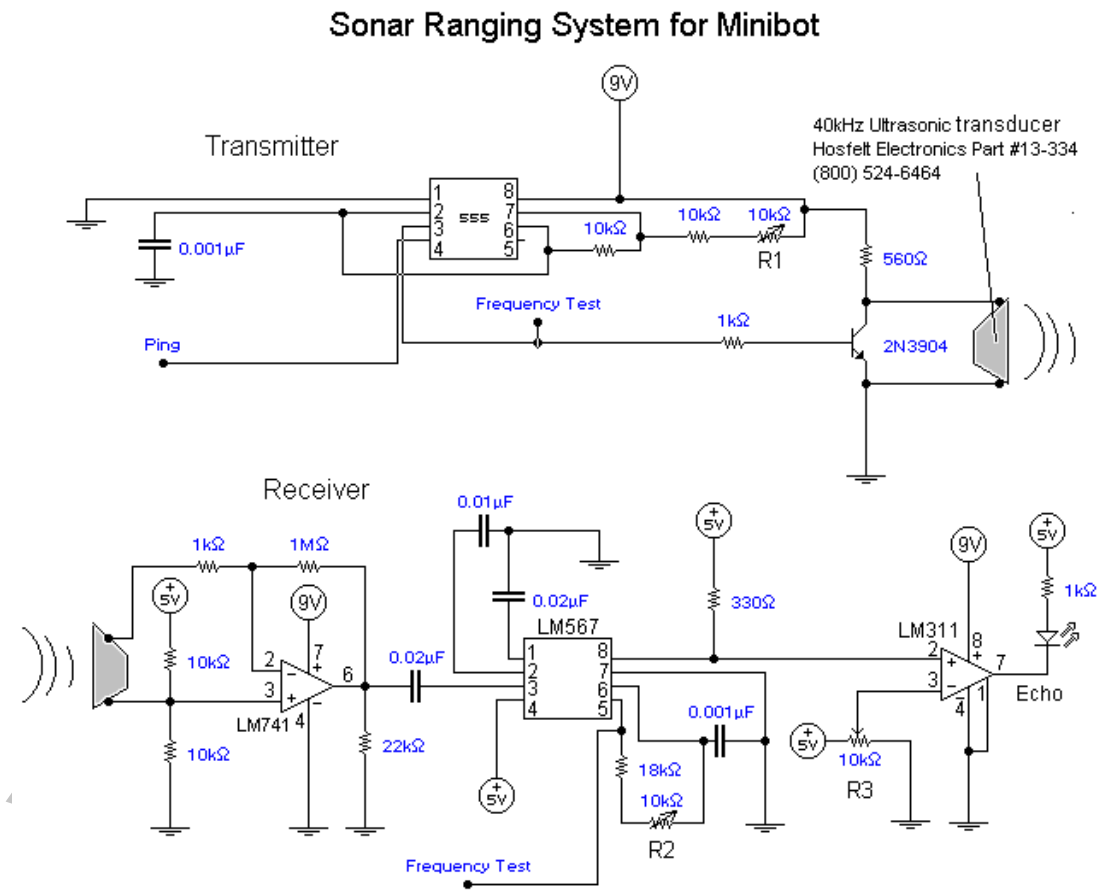



**Figura 30. Conexionado de ultrasonidos.**

El circuito está formado por dos partes independientes:

- El emisor de ultrasonidos:

Consta de un circuito integrado LM555. Su función es la de generar pulsos a una frecuencia fija que depende de los valores de los condensadores y resistencias externas del circuito. En este caso de unos 40kHz.

Esta frecuencia se utiliza para generar tonos en el altavoz, previa amplificación mediante un transistor.

El receptor de ultrasonidos:

Tiene una etapa de amplificación con un operacional.

La señal entonces la toma el LM567, que se trata de un integrado decodificador de tonos. Es capaz de distinguir la frecuencia que se le marque con los condensadores y resistencias externas.

Un comparador al final del circuito determina el nivel de audición y mejora la señal eliminando ruidos. Si el sensor ve algo, el LED se enciende.

Otros circuitos de ultrasonidos son comerciales (se hace saber al personal que muchos participantes en los concursos de microrrobótica llevan estos elementos, bastante fiables por lo que parece).

Existen detectores de presencia (si se coloca algo delante lo ven) y sensores de movimiento (sólo ven algo si ese algo se mueve, como las alarmas).

Hay que decir que ajustar el circuito para que funcione bien tiene su dificultad...

### *3.5 POTENCIÓMETROS:*

En el caso de que el microcontrolador tenga entrada analógica, se podría emplear un convertidor A/D (caso del PIC18F4550) para hacer lecturas directas de valores de tensión. En el caso de un micro pequeño como el PIC16F84, no tiene mucho interés pues no tiene entrada analógica, y un convertidor A/D ocuparía unas entradas digitales (normalmente 8 para el bus de datos y 2 ó 3 para el bus de control), con lo cual, pocas más quedarían para el resto de funciones del microrrobot.

Las entradas analógicas habitualmente se emplean para medir giros, como por ejemplo, giro de ruedas directrices en un vehículo. Los servos de radio control llevan potenciómetros para saber la posición del eje. También existen potenciómetros lineales.

### *3.6 ENCODERS*

Un encoder es un sensor que consiste en un disco con marcas que se van contando para saber cuánto ha girado un eje, en qué sentido lo hace y a qué velocidad. Normalmente es un disco ranurado en su perímetro. Un sensor de luz se activará cada vez que una ranura deje hueco por el que pasar la luz de otro emisor que tiene enfrentado. Así se obtienen una señal cada vez que pasa una ranura. Pueden llegar a tener más de doscientas ranuras.


**Figura 31. Principio de funcionamiento del encoder.**

En microrrobótica se pueden construir unos encoders rudimentarios utilizando sensores de infrarrojos CNY70 reflejando sobre discos con franjas blancas y negras.

Los discos pueden construirse imprimiendo en un papel un dibujo similar a este:

**Figura 32. Disco de encoder casero.**

Los discos, una vez cortados, se pegan por la parte interna de las ruedas intentando que sus centros coincidan con los ejes de las ruedas.

El problema para implementar encoders es ir contando los pulsos sin perder la cuenta, al tiempo que el micro hace otra cosa. Esto se consigue bien por programación mediante excepciones, o bien con un contador de encoders, que cuenta los pulsos y directamente pasa el número al micro cuando este lo requiere. Un ejemplo de contador de encoders es el HTCL2016 ó el HTCL2020.

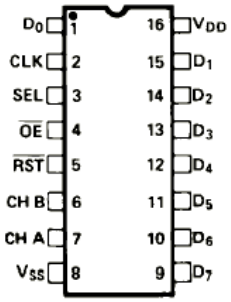



**Figura 33. HTCL2020: contador de encoder.**

Si bien podría usarse alguno de los temporizadores/contadores del PIC, sabiendo que los contadores del PIC sólo cuentan en un sentido, es decir, el valor de cuenta

sólo se incrementa y no puede decrementarse. Este inconveniente puede obviarse si el encoder sólo va a girar en un único sentido.

### 3.6.1 GP2D12: Sensor de medida de distancias por infrarrojos

¿Cómo funciona el autofocus de una cámara de fotos? Con uno similar a estos.

La familia de sensores Sharp GP2Dxx es una de las más utilizadas tanto en lo que viene a denominarse robótica móvil casera como en el ámbito de investigación debido principalmente a su facilidad de integración y su bajo coste (unos 15 Euros). En la Figura puede verse una imagen de un GP2D12.

**Figura 34. El sensor de infrarrojos GP2D12.**

Otro dispositivo de esta familia bastante utilizado es el GP2D02. La principal diferencia entre ambos es que los GP2D02 ofrecen una salida digital (veo/no veo) sin posibilidad de medir distancias, mientras que los GP2D12 dan una salida analógica entre 0 y 3 voltios dependiendo de la distancia a la que se encuentre el objeto.

También existen modelos conocidos como GP2Dx5 que dan una salida a nivel alto cuando la medida sobrepasa determinado umbral y que funcionan como optointerruptores. En las páginas de Sharp[2] puede encontrarse información adicional al respecto.

Tanto unos como otros se basan en el *principio de triangulación* para realizar las medidas. El elemento a la izquierda del sensor según vemos la Figura es un led infrarrojo que emite un haz que será rebotado por el objeto y posteriormente recogido por el elemento situado a la derecha. Este último se conoce como PSD (*Position Sensing Device*, Dispositivo de Percepción de Posición) y puede entenderse como una lente situada sobre un array de células sensibles a la luz infrarroja. Dependiendo del ángulo de incidencia del haz rebotado en la lente, se activa una u otra célula del array lo que permite estimar la distancia a la que se encuentra el objeto.

El conexionado de los GP2D12 con un microcontrolador es sumamente sencillo requiriendo solamente una entrada del conversor analógico-digital a la que se conectará el pin de salida del sensor (el de más a la izquierda visto de frente según se muestra en la Figura). Los otros dos pines corresponden, respectivamente, con GND y con Vcc, la tensión de alimentación, que deberá ser próxima a los 5 voltios. Se recomienda el uso de una tensión regulada (por ejemplo, mediante un 7805) para no introducir ruido en las medidas debido a que la tensión caiga por debajo de su umbral de funcionamiento. Así mismo, se recomienda utilizar un condensador de 22 uF entre Vcc y GND para reducir el ruido en la alimentación debida a la corriente requerida por el led emisor. También podría utilizase un condensador entre la señal de salida Vo

[2] http://www.sharp.co.jp/ecg/opto/products/osd/qr10-01.html

y Vcc o GND (filtro paso bajo) pero se disminuye considerablemente la dinámica del sensor.

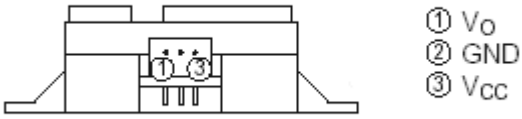



**Figura 35. Conector de los GP2D12.**

El rango de medida se encuentra entre los 10 y 80 cm. Si se baja de los 10cm umbral, el sensor no funciona correctamente. Dada esta característica, es bastante habitual colocar el sensor de forma que nunca puedan darse medidas menores a los 10 cm, dejando, por ejemplo, que esa distancia sea cubierta por el chasis del robot.

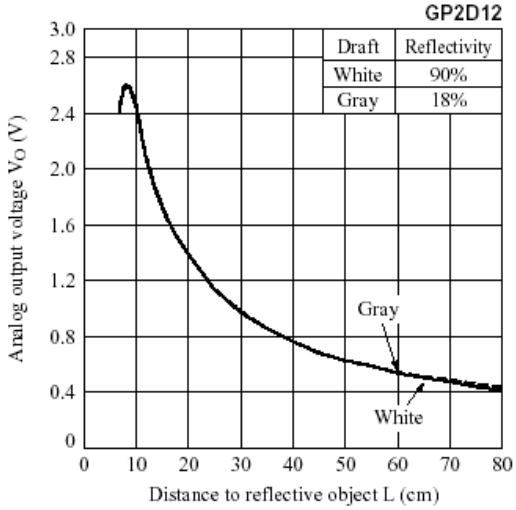



**Figura 36. Curva característica del GP2D12.**

# *CAPÍTULO 4*

# *MOTORES*

## *4.1 MOTORES DE CORRIENTE CONTINUA*

Como dice el nombre, estos motores son de corriente continua. Esto significa que para que giren lo único que se debe hacer es conectarlos a una fuente de alimentación (o una pila). El motor sólo tiene dos terminales; si invertimos la alimentación el motor girará en sentido contrario. También debéis saber que si aumentamos o disminuimos la tensión el motor girará más o menos rápido.

El principio físico de funcionamiento no es muy complejo. Por un lado se tiene un campo magnético fijo en el estator (la parte de fuera y estática del motor). Este campo magnético se consigue gracias a un imán permanente. Por el otro lado, se tiene otro campo magnético asociado al rotor (la parte giratoria). El motor se moverá para intentar alinear los campos magnéticos del rotor y del estator, tal y como sucederían si fueran dos imanes no alineados.

La característica fundamental de los motes DC es cómo se consigue generar el campo magnético del rotor: en este caso tenemos bobinas alimentadas por unas escobillas. Estas escobillas sólo energizan a la(s) bobina(s) adecuadas para que el campo generado en el rotor esté a 90º del campo del estator. El propio movimiento del rotor provoca la conmutación de la alimentación de las bobinas del rotor de forma que sólo esté(n) energizada(s) la(s) bobina(s) que generen un campo magnético en el rotor a 90º del del estator. De esta forma, se consigue un movimiento continuo.

**Figura 37 Foto de un motor DC Común.**

### 4.1.1 Método de ensayo de motores DC

## Introducción

Cuando estamos trabajando con microrobots, es muy común emplear motores de corriente continua, por eso durante la construcción de un microrobot a menudo se recurre al uso de motores de continua reciclados de otras aplicaciones sobre los cuales no tenemos ninguna clase de información.

Esto supone en muchos casos que los motores que hemos aplicado para la realización de ciertas tareas, no sean los adecuados, porque sus características no son las apropiadas para el cumplimiento de dichas tareas.

A continuación se va a describir un método sencillo mediante el cual podremos obtener algo fundamental en todo motor, su curva característica de par/velocidad.

## Material necesario

A la hora de dimensionar un motor, necesitaremos los siguientes aparatos, algunos pueden ser difíciles de encontrar:

- Taladro normal de 400/600W de potencia
- Tacómetro
- Cinta aislante blanca
- Polímetro
- Motor de continua a dimensionar

En caso de no poseer el tacómetro, haría falta un motor del cual se supieran con seguridad las revoluciones por minuto (RPM)

## Procedimiento a seguir

En primer lugar deberemos introducir el eje de salida del motor que deseamos dimensionar en el porta brocas del taladro, como si tratara de una broca, a continuación colocaremos un trozo de cinta aislante blanca por la parte exterior del porta brocas del taladro.

Una vez realizado esto, encenderemos el taladro y sujetando el motor con la mano, para evitar que gire la carcasa (nos interesa que gire el rotor del motor), mediremos la tensión inducida entre las dos bornas del motor usando el polímetro y la velocidad de giro del motor usando el tacómetro. Para ello debemos apuntar con el tacómetro al porta brocas del taladro, así, el tacómetro lanza un haz de luz sobre el porta brocas y recibe un pulso cada vez que pasa la cinta aislante blanca. Con esto conseguiremos la tensión inducida en el motor cuando este gira a una determinada velocidad.

Una vez obtenidos estos datos, a continuación debemos medir la resistencia interna de los bobinados de nuestro motor. Para ello, usaremos el polímetro.

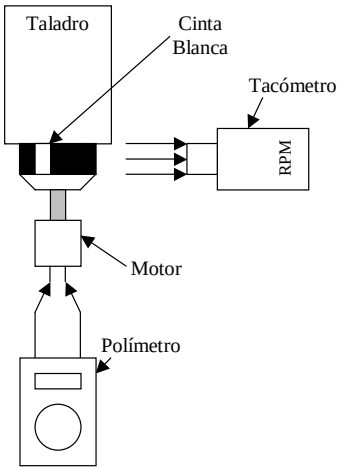



**Figura 38 Ensayo de un motor DC Común.**

## Cálculos a realizar

Con los datos obtenidos del ensayo, obtendremos el valor de ($k_\phi$) mediante la expresión:

$$K_\phi = \frac{E}{\Omega}$$

Una vez obtenida dicha constante, se sustituye en la ecuación que nos dará la curva par/velocidad del motor, que en este caso debe ser una línea recta:

$$T = \frac{V}{R} K_\phi - \frac{K_\phi^2}{R} \Omega$$

Siendo:

- **T**: Par entregado por el motor [N/m]
- **V**: Voltaje aplicado a las bornas del motor
- **Ω**: Velocidad de giro del motor [rad/s]

Hay que destacar que la ecuación obtenida es para un comportamiento totalmente ideal del motor, en el que se ha despreciado el rozamiento producido por las escobillas del motor. Realmente el motor trabajará en torno al 25% de las revoluciones máximas obtenidas idealmente.

Por último solo queda dibujar la curva de par/velocidad y con el dato aproximado de la velocidad real de salida en vacío del motor, sabremos cual es el par aproximado entregado por el motor, trabajando éste en vacío.

### *4.2 SERVOS*

Los servos de modelismo son los que se usan en aviones, coches y vehículos a escala a control remoto. Tienen la ventaja de que llevan un pequeño circuito que hace que su control sea sencillo (de lazo abierto, que ya diremos luego lo que es).

Este tipo de motor es bueno para definir el ángulo de giro que se desea; por ejemplo, tú quieres que el servo gire 30 grados. Pues se envía la referencia (a través de su señal correspondiente), y el servo se pone a 30 grados. Esto con un motor DC convencional, no se puede conseguir así de fácil (hay que hacer el famoso lazo cerrado de control).

Este es el diagrama de un servomotor típico para modelismo:

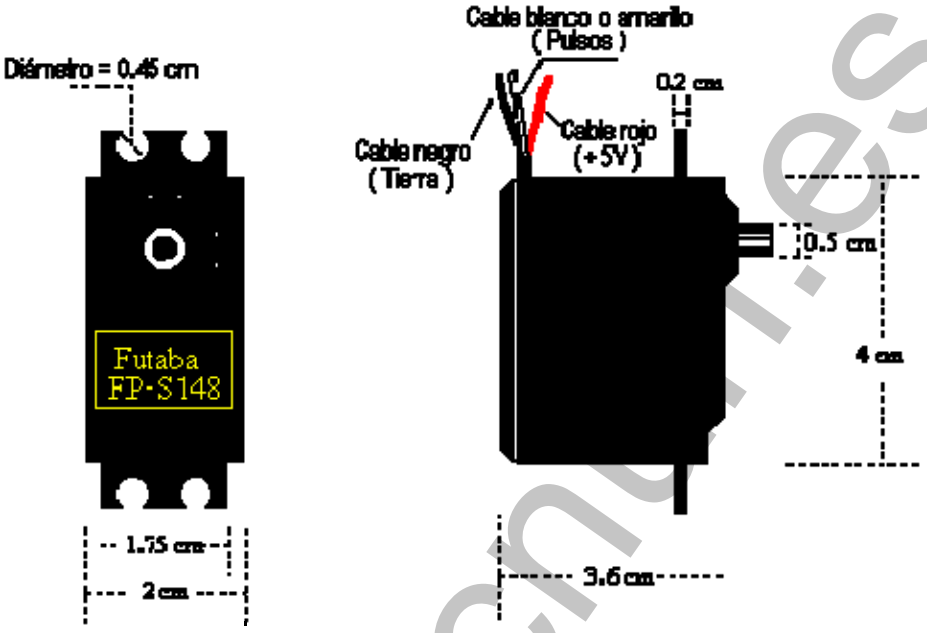



**Figura 39. Servomotor.**

Un servomotor de estos es básicamente un motor eléctrico que sólo se puede mover en un ángulo de aproximadamente 180 grados (no dan vueltas completas como los motores normales). Ver que solo tiene TRES cables que salen de su cajita. El rojo es de alimentación de voltaje (+5V), el negro es de tierra y el cable blanco (a veces amarillo) es el cable por el cuál se le pide al servomotor a qué posición debe ir (entre 0 grados a 180); en ese cable se debe meter un tren de pulsos.

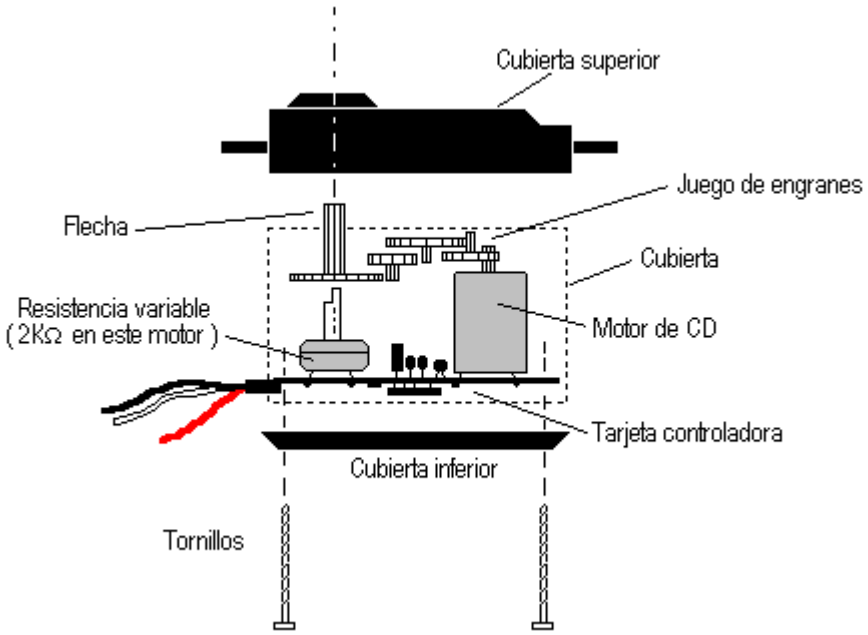



**Figura 40. Partes de un servo.**

La señal de pulsos controla al servo de la siguiente forma:

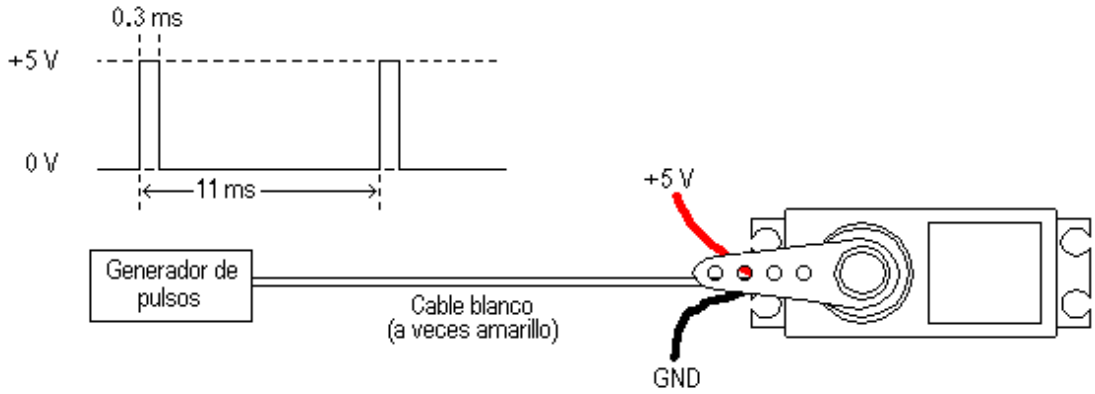



**Figura 41. Diagrama de tiempos de control de un servo.**

Teneis que saber que el intervalo de pulsos se mantiene constante, y que lo que hace cambiar la posición del servo es el ancho del pulso de entrada. Responden bien a pulsos desde 50 Hz hasta aproximadamente 100 Hz. Para cada tipo de servo que se desee controlar, se deberá realizar una prueba preliminar para encontrar exactamente el período y la duración de los pulsos que mejor le funcionen. Un osciloscopio y un generador de señales facilitan mucho las cosas (y lo mejor sigue siendo mirar en la hoja de características).

### 4.2.1 Modificación de servomotores

Los servomotores se pueden usar como vienen de fábrica para posicionar el eje en un ángulo concreto, que es su función y para lo que están diseñados, o modificarlos y conseguir otras funciones. Dependiendo de las modificaciones que se efectúen se conseguirán diferentes tipos de motor o de actuaciones.

A continuación se muestra cuales pueden ser las posibilidades:

- Sin modificación: El servo actúa en su función nominal, para lo que está diseñado. Al aplicar un tren de pulsos determinado, se posiciona con un ángulo concreto.
- Quitando el tope de un engranaje y la placa de control: En este caso queda un motor de corriente continua común con una caja reductora.
- Quitando el tope de un engranaje y el potenciómetro y manteniendo la placa de control: Queda un motor de giro completo, controlándose la velocidad y el sentido mediante señales PWM.

El trucaje de cada servo se ha realizado siguiendo los siguientes pasos:

- Lo primero que se ha hecho es abrir la caja posterior del servo desenroscando los cuatro tornillos.

- Se ha extraído la tapa superior del servo que encierra la caja reductora compuesta por engranajes rectos.

- Valiéndose de una cuchilla, se ha cortado el tope mecánico del engranaje de salida del servo para dejar girar libremente y sin limitaciones al eje en los 360º.

- Se ha sustituido el potenciómetro original del servo de 5K por uno del mismo valor multivuelta vertical, en la posición análoga

a la del original. Para poder hacerlo, se han soldado tres cables a las patas del potenciómetro y a la placa.

- Para terminar, se ha practicado un agujero de 3 mm de diámetro a la altura a la que ha quedado el tornillo de ajuste del potenciómetro, para poder ajustarlo y actuar sobre él desde el exterior.

Para que el servo se comporte como se espera, es necesario un ajuste del potenciómetro multivuelta. Para ello se ha conectado a un circuito que da la señal requerida en uno no trucado para posicionarlo en el centro y se ha variado el valor del potenciómetro hasta que el eje del servomotor trucado se ha detenido. De este modo, se consigue que la analogía posición-velocidad entre un servo trucado y uno no trucado, tengan el mismo valor central, que corresponde en el primero en la posición media y en el segundo con velocidad nula. Para que dos servos trucados obtengan la misma velocidad ante la misma señal de control, se requiere que el ajuste se realice con idéntica señal para los dos.

Al trucar un servo, en realidad lo que se hace es abrir el lazo de control dejando el potenciómetro con un valor fijo. El controlador espera que el valor leído de la resta entre la señal de entrada y la del potenciómetro vaya reduciéndose, pero como el potenciómetro tiene un valor constante, el resultado de la resta no cambia. Con este trucaje se consigue "engañar" a la tarjeta controladora para que constantemente mande al motor "la orden" de girar.

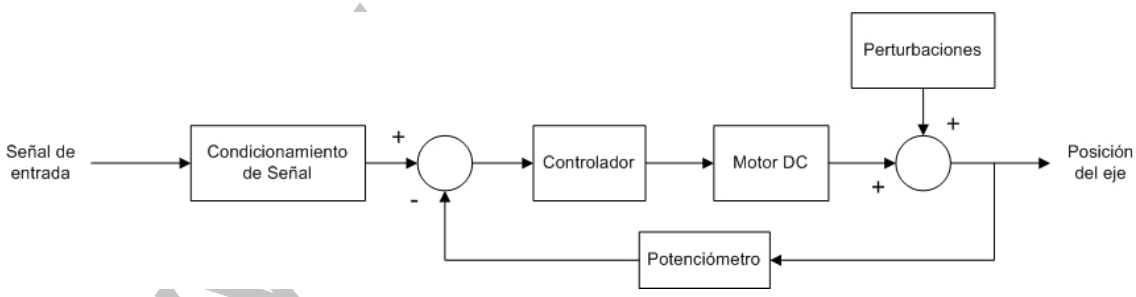



**Figura 42. Diagrama de bloques de la controladora de servomotor.**

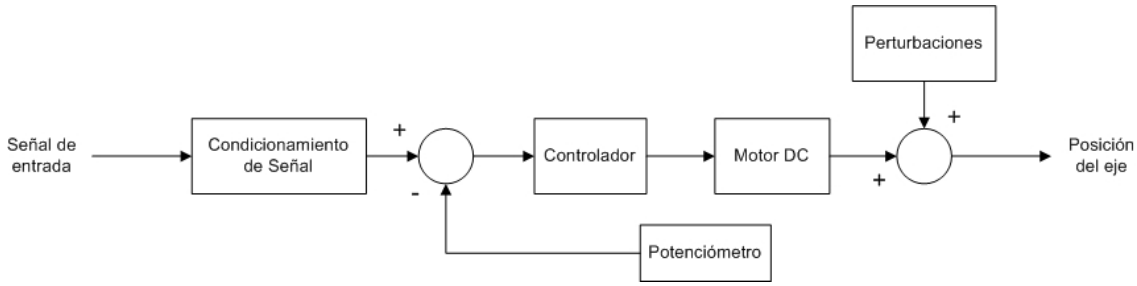



**Figura 43. Diagrama de bloques de la controladora de servomotor trucado**

Tras estas modificaciones, el servomotor se controla en velocidad mediante una señal PWM con un periodo alrededor de 20 ms. El servo estará parado cuando el ancho de pulso sea 1.5 ms y girará a velocidad máxima cuando sea 0.5 o 2.5 ms, dependiendo del sentido de giro.

## *4.3 MOTORES PASO A PASO*

Simplificando el motor lo podemos esquematizar mediante 4 bobinas. Cuando alimentamos una bobina se va a incidir un campo magnético el cual conseguirá "mover " el rotor (es decir, el "eje" del motor).

Va a haber 2 secuencias básicas. A partir del paso 4 la secuencia es repetida desde el paso 1.

### 4.3.1 1ª secuencia: paso completo

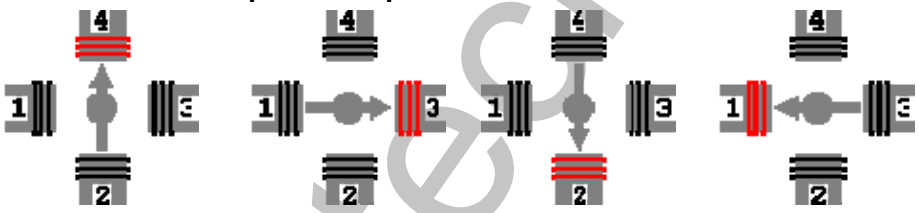


**Figura 44. Secuencia de un motor PAP de paso completo.**

Cada bobina es alimentada una vez. Esta secuencia produce el movimiento más suave y es el que consume menos potencia.

### 4.3.2 2ª secuencia: medio paso

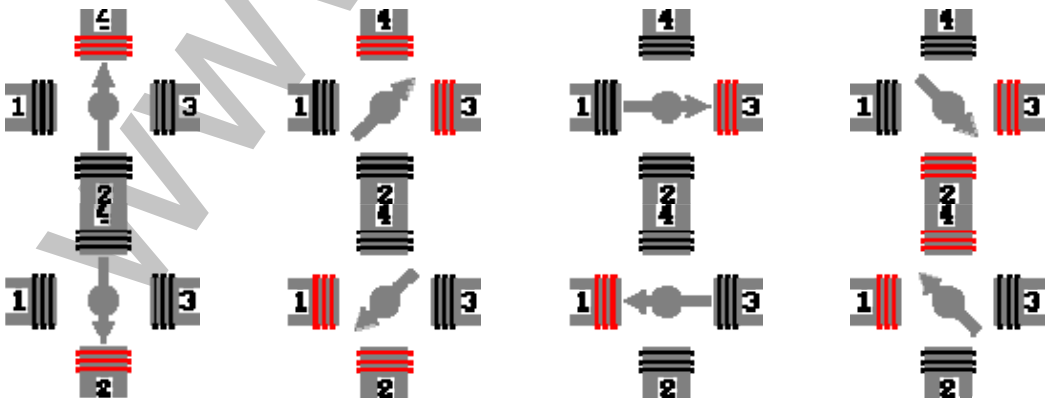


**Figura 45. Secuencia de un motor PAP de medio paso.**

Cada turno son alimentadas una ó dos bobinas.

El movimiento es más suave pero conseguimos un par mucho mayor cuando tenemos dos bobinas energizadas.

Las conexiones del motor PAP son de este modo:


(NOTA: Coil = Bobina.)

**Figura 46. Pinamen de un motor PAP.**

Para la etapa de potencia se puede utilizar el L297 y L298, diseñados para controlar motores paso a paso, tanto uni como bipolares, con un voltaje máximo de 35v. y hasta 3 A. de consumo por bobinado. Hay que tener en cuenta que las etapas de potencia hay que elegirlas según las características del motor .

**Figura 47. Foto de un motor PAP.**

# *CAPÍTULO 5*

# *ETAPAS DE POTENCIA*

## *5.1 ETAPAS DE POTENCIA*

Vamos a empezar por el primer problema que se tiene cuando se quiere controlar **desde un circuito digital** un dispositivo electromecánico: **¿Cómo conectar mi dispositivo electromecánico?** Pues el más importante consejo es: **NUNCA** lo conectes directamente a la salida digital de tu circuito. Por dos razones:

- **Razón 1:** Un circuito digital tradicional generalmente no tiene la capacidad de corriente necesaria para hacer que un motor eléctrico de vueltas. Si conectas directamente un motor pequeño, lo más probable es que tu circuito se sobrecaliente y se queme en unos segundos. La manera más sencilla de manejar un elemento electromecánico pequeño con un circuito digital es utilizando un TRANSISTOR como interruptor. Así tu circuito digital solo activa y desactiva el transistor (eso sí se puede) y el transistor es el que activa y desactiva el motor.
- **Razón 2:** Casi todos los dispositivos electromecánicos (aunque sean pequeños) son muy inductivos. ¿Qué significa eso? Significa que **no permiten ser apagados de golpe**. Es decir, cuando tú desconectas un motor eléctrico que está funcionando, el motor (debido a que es un dispositivo inductivo) trata todavía de mantener por una fracción de segundo la corriente circulando a través de él. Y durante este pequeñísimo tiempo puede generarse una chispa en la parte del circuito que realizó la desconexión. Esta chispa puede muy fácilmente dañar circuitos electrónicos.

Según el tamaño del motor y según la corriente que esté utilizando, esta chispa puede o no ser visible, pero siempre existe **a menos que se coloque en paralelo con el motor un diodo de protección**. Este diodo tiene como finalidad servir de "desahogo" para esta corriente residual que aparece después de que se apaga el motor. Así que, muy en resumen, este es el circuito que necesitamos para encender y apagar un motor eléctrico pequeño de corriente directa desde un circuito digital:

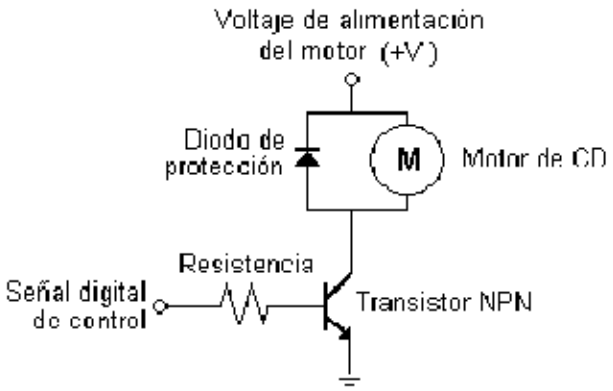



**Figura 48. Esquema de una etapa de potencia simple.**

## *5.2 PUENTE-H*

Los puentes-H (llamados "H BRIDGES" en inglés) son circuitos que permiten controlar motores eléctricos **de corriente continua** en dos direcciones desde un circuito digital (TTL, CMOS, el puerto de una computadora, desde un microcontrolador, etc...). Se les llama "Puentes H" porque precisamente su forma recuerda (muy vagamente)a una letra "H".

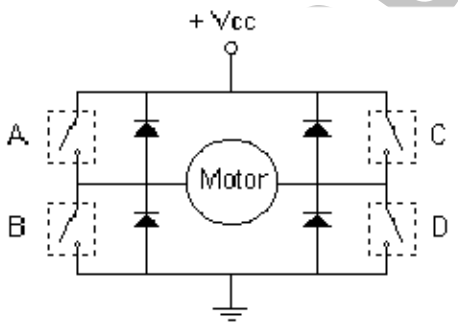



**Figura 49. Esquema de un puente H.**

Estos interruptores (A, B, C y D) pueden ser de transistores bipolares (como el de arriba), de mosfets, de jfets.

Este circuito, además de permitir el arranque y parada del motor, controla el sentido de giro.

Si se cierran solamente los contactos **A y D** la corriente circulará en un sentido a través del motor (o del relevador o de cualquier sistema que esté conectado ahí en medio), y si se cierran solamente los contactos **B y C** la corriente circulará en sentido contrario. De preferencia nunca cierres los contactos A y B al mismo tiempo (tampoco C y D) porque podrías fundir un fusible en alguna parte. Observa también que un puente H necesita de cuatro diodos de protección para el motor.

El siguiente esquemático muestra cómo se conectaría un puente-H. Una cosa importante a señalar es que suele ser interesante tener dos pilas o fuentes de alimentación: una para alimentar la electrónica de control (PIC y demás) y otra que suministre la corriente de los motores.

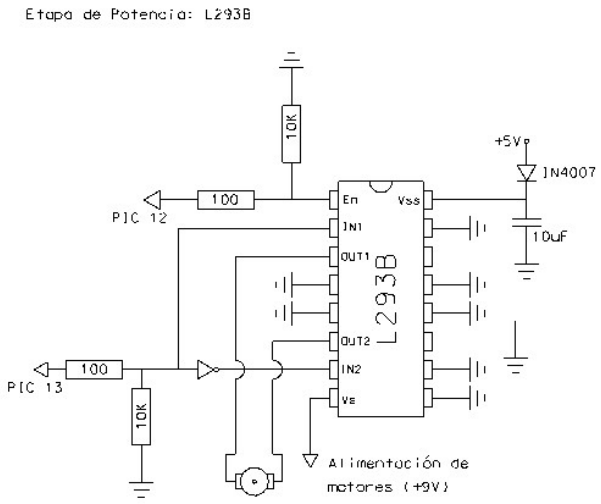



**Figura 50. Etapa de potencia basada en un L293.**

El PIC puede controlar el puente L293B mediante las señales que envía por salidas digitales que están conectadas al ENABLE del puente y a las entradas IN1 e IN2, de manera que el sentido de giro se le asigna a una sola salida y la velocidad a la otra si ésta se conecta a una de las salidas PWM del PIC16F87x. En realidad, este puente se diseñó para dejar el ENABLE habilitado y jugar con las entradas IN1 e IN2, pero esto supone relizar una modulación PWM en dos pines distintos del PIC, lo cual complicaría bastante la programación.

Si el PIC no tuviera salidas PWM, éstas podrían intentarse simular por software. Es decir: el PWM que se hace en el micro se consigue programando bucles. Y resulta difícil conseguir un bucle que proporcione distintas frecuencias. En este caso, la carga de cómputo que le pedimos al PIC es mucho mayor, dejando pocos recursos para hacer otras cosas.

# *CAPÍTULO 6*

# *PUERTO SERIE*

## *6.1 EL PUERTO SERIE DEL PIC*

El PIC16F87x y el PIC18F4550 (no así el 16F84) tienen una interfaz de puerto serie sencilla de utilizar. A través del puerto serie del PIC podrían conectarse consolas de RS232 e incluso un PC para comunicarnos con el PIC.

Aprovechando la conexión desde el PC podemos realizar un programa en C, Visual Basic, Java, LabView para completar las capacidades del software grabado en el PIC o simplemente si usamos el HyperTerminal, podemos usar el PC como dispositivo entrada-salida del programa en el PIC.

En la siguiente figura se representa la parte del esquemático básica para hacer funcionar el puerto serie de un PIC, donde se pueden distinguir las diferentes partes que lo rodean: el oscilador, la circuitería de reset, las entradas de los dos sensores, las salidas PWM con sus señales de control (SELECT) y las dos líneas de comunicación serie (TX y RX).

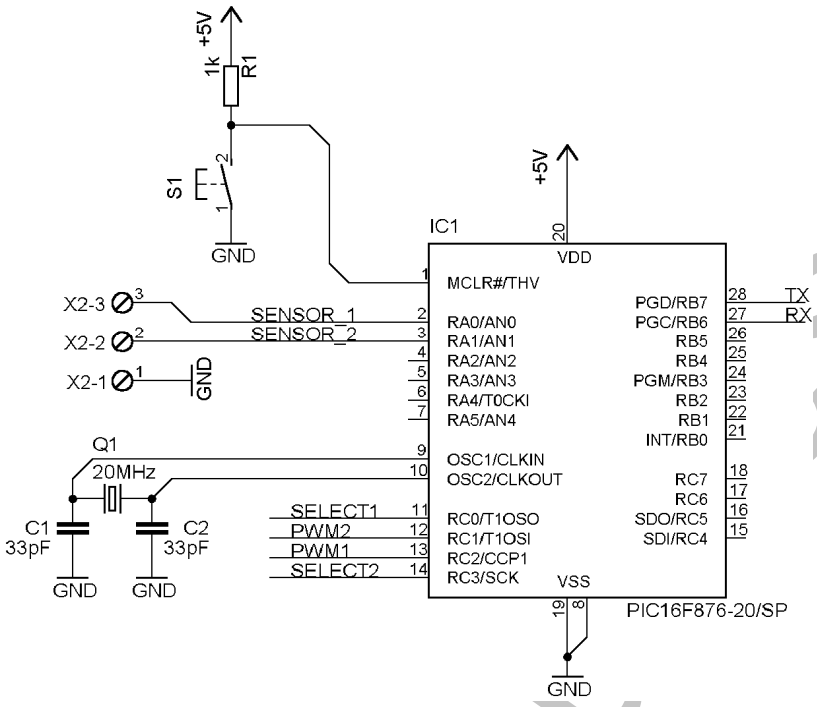



**Figura 51. Circuito básico del PIC16F87: los puertos usados para comunicación serie son el pin 28 para el TX y para el 27 el RX.**

### 6.1.1 Max232

El problema que ahora se plantea es que los pines 27 (Rx) y 28 (Tx) trabajan con tensiones TTL (0-5V) que no son compatibles con los niveles de tensiones de la norma RS232. Se necesita un convertidor TTL/RS232.

El convertidor TTL / RS232 tiene como objetivo realizar un enlace a través del puerto serie del PC, de manera que se puedan enviar y recibir datos de la tarjeta de control. Está basado en el circuito integrado MAX232 de la casa Maxim.

La siguiente figura muestra en detalle el esquema utilizado para la conversión de las señales TTL en señales compatibles RS232:

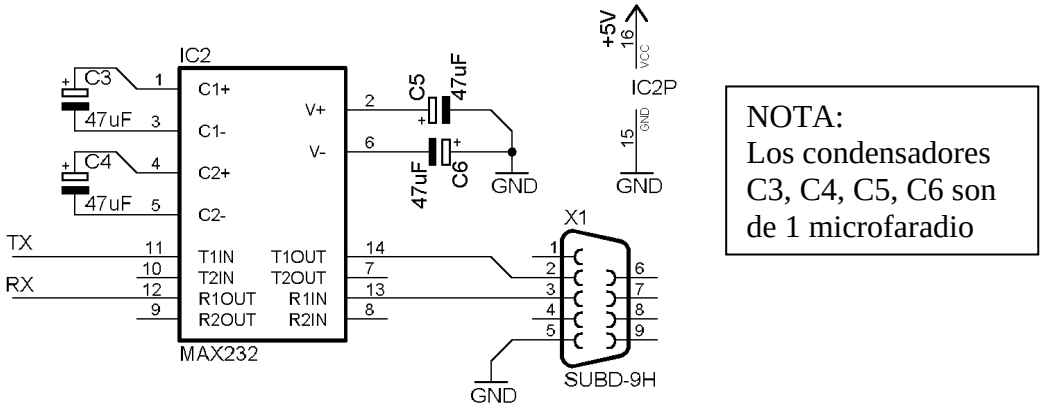



**Figura 52. Conexionado del Max232.**

# CAPÍTULO 7

# PCB (PRINTED CIRCUIT BOARD)

## 7.1 INTRODUCCIÓN

Ahora es el turno de hablar sobre la típica placa perforada donde se insertan los componentes electrónicos.

Existen dos opciones de 'atacar' el problema. La primera es ir a Electrosón y comprar las PCBs universales. La segunda es diseñar nuestra propia PCB y construirla con el proceso de 'ataque' por ácido.

## 7.2 EMPLEO DE PCBS UNIVERSALES

En un primer momento puede parecer la solución 'más fácil y más rápida'. Pero si no se hace con mucho cuidado puede ser el método más frustrante. El mejor consejo que puede darse es que hay que hacerlo con mucho orden.

Antes de nada, hay que dibujar el circuito en papel cuadriculado (o en el ordenador) antes de soldar nada. De esta manera se prevé hueco para todos los componentes y se pueden trazar las pistas minimizando el tamaño y el número de cables que se pueden enredar, romper, cortocircuitar...además, queda constancia por escrito del circuito, en la que se puede ver qué es cada cosa y cada pista.

Si hay que cruzar pistas, los puentes se pueden hacer con los trozos sobrantes de los pines de los componentes. (Para que encajen perfectamente, se cuentan los agujeros que se quieren "saltar" y se introduce lo que será el puente en el borde de la placa a una distancia igual al número de agujeros que se han contado. Al estar en el borde, lo podemos doblar con gran facilidad y nos quedará una grapa perfecta y a medida para poner donde queramos)

Si el puente que hay que echar es muy largo, un cable rígido aislado es más apropiado que uno trenzado, porque lo podemos dejar doblado al ras de la placa, y no cuelga. Si el cable es para conectar la placa a otros elementos susceptibles de moverse, es más recomendable el trenzado, al ser más flexible y no romperse cuando se manipula.

Si se van a soldar cables trenzados a la placa, es muy interesante cubrir sus extremos con estaño. Esto evitará que se deshilachen y que se rompan.

Para que los componentes se sujeten al dar la vuelta a la placa, abrimos un poco sus pines.

La mayoría de los componentes encajan en los agujeros sin forzar sus patas. Ponerlos bien pegados a la placa evitará que se doblen de un lado para otro y acaben rompiéndose.

## *7.3 PLACAS EN ÁCIDO (PCB):*

Partiendo del esquema del circuito se dibujan las pistas con la ayuda de un programa de ordenador (Protel, Eagle...).

Ahora se construye la máscara: el layout, generado por el programa de CAD, debe imprimirse, a la máxima calidad posible, en transparencia para pasarlo a la placa. Si la impresora que se tiene no es de muy buena calidad, usar dos transparencias suele ser un truquillo que funciona bastante bien.

El procedimiento de ataque con ácido es el siguiente:

- Se compra una placa PCB para atacar de las denominadas 'positivas'. Esto es, la máscara que se usa es 'positiva' a diferencia de las placas de revelado negativo.
- Se quita la protección de la placa fotosensible en un cuarto oscuro (usad la luz roja) para que no se vele.
- Se coloca la transparencia sobre la placa por el lado sensible a la luz (se sujeta con cinta adhesiva)
- Se mete en la insoladora durante unos tres minutos.
- Se revela la placa en una disolución de: 1 parte de *Revelador* por 5 de agua .
- Una vez bien revelado (es decir, se ven que aparecen las pistas), se aclara con agua.
- Una vez aclarado, ya se puede encender la luz normal.
- Atacar entonces con ácido hasta que se coma todo salvo las pistas con: 1 parte de *Atacador Rápido S* por 5 partes de *Atacador Rápido L*.
- Aclarar entonces con agua, y limpiar con alcohol.
- Tanto el revelador como el ácido pueden utilizarse varias veces, pero con el uso pierden sus propiedades, por lo que hay que cambiarlos por soluciones nuevas.

### 7.3.1 Normas de seguridad

Los productos utilizados en el proceso de fabricación de la PCB son peligrosos, por lo cual debéis extremar la atención:

- Usad siempre guantes desechables
- No rascarse los ojos, ni meterse las manos en la boca (sí no os riáis) cuando estéis trabajando con estos productos
- Después de revelar las placas, lavaros las manos con agua abundante
- Los productos residuales deben verterse en los bidones a tal fin: uno de ellos es para el revelador y otro es para el ácido.

### 7.3.2 Algunas recomendaciones al respecto son:

- No hacer las pistas demasiado finas porque el ácido podría comérselas: es más, intentar hacerlas lo más gruesas posibles e intentar que el ácido tenga que comer sólo lo imprescindible.

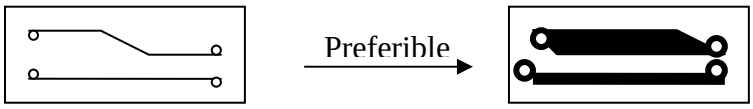



**Figura 53. Grosor recomendado de pistas en los PCBs.**

- Cambiar de posición la placa en la insoladora para que la luz incida más uniformemente (sobre todo si la placa es grande)
- Marcar los agujeros con un punzón antes de hacerlos.

## *7.4 CONSEJOS PARA SOLDAR BIEN*

- Aplicación correcta de la gota de estaño: la gota de estaño debe quedar uniforme, brillante y formando un cono en el pad abrazando completamente la patilla del componente

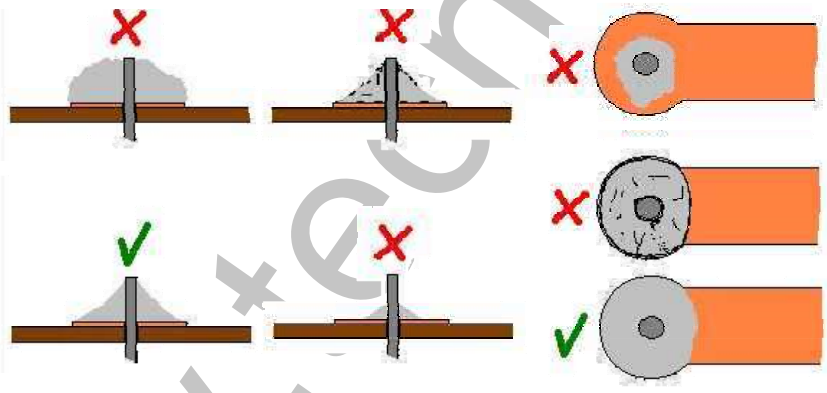


**Figura 54. Aplicación correcta de la gota de estaño.**

- Procedimiento de soldadura con el estañador:
  - Verificar que el componente y la superficie a estañar estén limpias.
  - Asegurar bien el área de trabajo, inmovilizando si es posible el PCB y el componente a soldar.
  - Limpiar el estañador con la esponja humedecida.
  - Colocar un poco de estaño en la punta del estañador.
  - Introducir el componente en el orificio del PCB.
  - Calentar la pata del componente y la superficie de cobre del pad.
  - Acercar estaño y dejar que se funda y fluya libremente bordeando completamente la patita del componente formado la típica estructura cónica.
  - Retirar rápidamente el estañador y bajo ningún concepto mover las superficies a estañar hasta que el estaño se haya solidificado bien. En caso que se muevan, el estaño adquirirá un tono mate (poco brillante) dando lugar a una soldadura fría que no trae más que problemas.

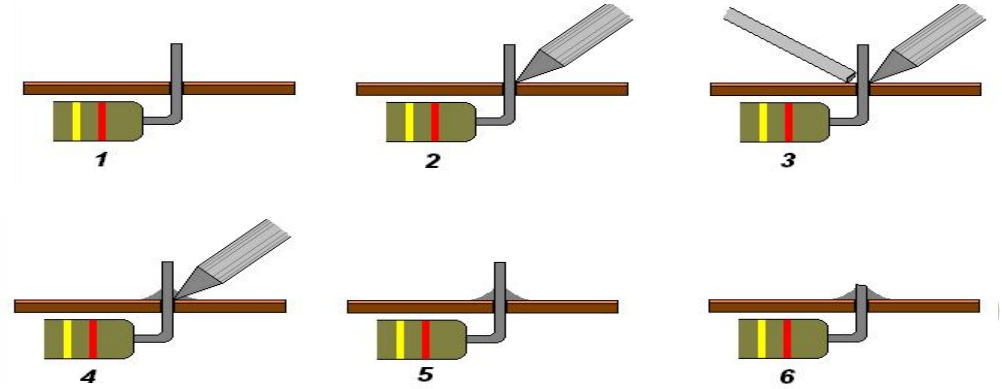



**Figura 55. Pasos a seguir par soldar bien.**

El orden recomendado para montar los componentes es:

- Zócalos y conectores
- Resistencias
- Condensadores y diodos
- Circuitos integrados y SMDs (Surface Mounted Device).

# *CAPÍTULO 8*

# *ALGUNOS TRUCOS Y ALGUNOS* ERRORES:

Estas cosillas conviene saberlas si se va a enredar en un laboratorio con circuitillos, motores, multímetros...

- *Fuentes de alimentación con voltaje e intensidad regulables:*

Normalmente se fija un voltaje para el circuito, y si todo va bien, circulará la corriente que deba. Si todo va mal, se producirá un cortocircuito, o algún componente estará al revés y se quemará. Estas fuentes permiten limitar la corriente y evitar estas 'barbacoas'.

Cuando el circuito pide más corriente de la que le da la fuente de alimentación (que no le da más porque la limitamos), se enciende un piloto rojo que nos advierte. Además, en el voltímetro que llevan se puede ver cómo baja el voltaje. El gran craso error es intentar subir el voltaje con el piloto de la intensidad encendido. Por más vueltas que le demos a la ruedita, el voltaje no subirá, y el circuito seguirá soportando la sobrecarga. Lo único que se debería tocar es el interruptor de apagado ó bien reducir la intensidad con su ruedita.

Los motores están formados por bobinas que al arrancar y en los cambios de sentido absorben mucha intensidad. A veces, parece que no funciona el robot porque la corriente de los motores está limitada.

- *Fuentes de alimentación regulables y no regulables. Pilas y baterías también:*

Si colocamos para un mismo circuito distintas alimentaciones por el motivo que sea (por ejemplo, lógica a 5V y motores a 9V), hay que referenciarlo todo a la misma tierra. Es decir: conectar las tierras de ambos. Esto es importante, porque si no, el circuito no funcionará.

- *Componentes digitales:*

Los pines al aire no son nada buenas, porque no se sabe si está a tierra o a cualquier otro voltaje. La respuesta del circuito es imprevisible...

¡¡Qué bien ya tenemos montada una antenita lista para captar nuestra emisora de radio favorita!!

- *Selección del TRIS*

Se recomienda el definir la configuración inicial de las líneas E/S, a través de set_tris_x, aunque éstas no se usen como E/S digitales. Por ejemplo, las líneas que se vayan usar como entradas analógicas se deberían configurar como entradas. Si se usa la línea serie, la pata RX hay que configurarla como salida, y TX, como entrada.

- *Circuitos integrados:*

...y todos aquellos susceptibles de estropearse con el calor, hay que evitar soldarlos directamente. Por eso están los zócalos.

- *Código de colores de los cables:*

Su importancia no conoce límites. Evita errores garrafales del tipo "¡oops!, conecté la polaridad al revés...". Los más importantes son: 'Cero Voltios' siempre de color **NEGRO**. Alimentación siempre de color **ROJO**. Otro color típico es blanco para señal (cuando es única).

- *Trabajo en grupo:*

Un error frecuente al trabajar en grupo es descubrir un error cometido por *otro*, corregirlo y más tarde descubrir que *no* era un error.

- *Papel:*

Pensar las cosas en papel, ahorra tiempo y facilita comprender qué ocurre cuando un circuito no funciona. Tener el esquemático delante es muy interesante. Saber qué señal proporcionan los sensores, también lo es (en el rastreador, ¿qué nivel lógico proporcionan los sensores al ver color negro? ¿con qué color se enciende el led?

- *Más papel:*

...tener las hojas de características delante ayuda mucho a comprender los circuitos y a no cometer errores de conexionado.

- *Zócalos:*

Cuando se va a meter y sacar un componente varias veces, es muy fácil (terriblemente fácil) que se rompa alguno de sus pines. Para evitarlo, lo metemos en un zócalo. De esta manera, si se rompe algún pin es del zócalo, se cambia y ya está. Así que no haya nadie que no le ponga un zócalo al PIC.

- *Cable plano:*

Cerrar el conector hembra sin el cable es un error. Luego no se puede abrir... Y cuanto más corto, mejor. Y nunca más de 5 metros...

- *Condensadores:*

Algunos tienen polaridad: son los condensadores electrolíticos. Parecen pequeñas pilas. La pata positiva es la más larga. Corren rumores de que si se conectan al revés, explotan.

Tántalo: naranjas o de color mostaza. La pata positiva viene marcada con una señal. A veces son tan (...) que la marcan con una rayita y parece un signo menos. No os dejeis engañar. (Parece ser, además, que tienen muy buen comportamiento a altas frecuencias).

- *Sentido de los Leds:*

Generalmente solemos poner leds para ver si está encendido o apagado el sensor (siempre que sea señal digital, es decir, 1 ó 0). Y eso suele tener sus dificultades, sobre todo si se hacen las cosas a última hora y rápido.

Fijaros por ejemplo en los sensores de línea que tenéis; en el esquema siguiente: ¿están bien conectados los LEDs?

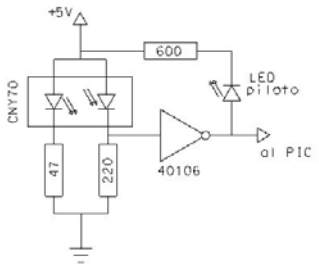



**Figura 56 Conexión errónea de LEDs**

Bueno, pues uno puede tener diferente dudas para colocarlo; siempre hay que ver en qué lado está el sitio más positivo y una vez situado pensar de donde a donde puede circular la intensidad. La pata positiva es la *más* larga. Y cuando se cortan los pines, se sabrá su polaridad porque tiene la cápsula con un lado plano donde está el pin negativo.

- *Signo de la señal de los sensores:*

Todos los sensores digitales (1 ó 0) dan un resultado dependiendo de su estado. Para saber este valor (p.ej. si los sensores de línea ven negro), nos podemos fijar en la hoja de características (que es buena idea), o podemos hacer algunas pruebas (peor). De la primera forma veremos en qué disposición está cada componente y así no meter la entrada de tensión por la entrada de tierra y ese tipo de cosas que se suelen hacer.

De todas formas, para el caso de los sensores de infrarrojos, conviene ver el esquema del diodo de infrarrojos y del fototransistor. Es lógico que si el diodo emite (para esa configuración, siempre emite), el fototransistor recibirá el rayo si este rebota en algún sitio; y eso lo hace si la superficie es blanca. Una vez que se recibe la luz, el colector del fototransistor se conectará a los 5 V. Pero ojo que luego hay una puerta inversora que cambiará el signo (de 1 a 0 y de 0 a 1).

Esto parece simple, pero a la hora de programar, dará problemillas si quereis hacer comparaciones del tipo AND, OR…

- *Polímetros:*

Muy útiles en general, especialmente para medir a la salida de los sensores y disipar sospechas sobre su buen funcionamiento.

Algunas veces (pocas) los polímetros fallan y miden cosas incomprensibles, por eso conviene tener dos encima de la mesa. La causa principal es la pila del chisme.

- *Programación:*

Aunque sea algo que ya sabéis, antes de programar siempre conviene hacerse un esquemilla (aunque sea mental) de lo que se quiere conseguir y cómo se quiere conseguir. Para ello se puede hacer un pequeño **diagrama de flujo**.

- *No funciona el programador:*

Seguramente sí funcione, pero tu maña para cambiar cosas de la configuración de un programa son sorprendentes y las has cambiado sin más. Así que revisa la configuración (en nuestro caso de ICD2, Ic-Prog o WinLV).

De todas formas, todavía puede pasar (poquísimas veces) que la configuración sea la correcta, así que conviene que probéis con otro PIC, por si acaso se ha quemado.

Cada vez que se grabe un programa nuevo, es conveniente borrar el viejo (aunque no necesario) ya que la probabilidad de fallo de grabación es menor.

Verificar el programa recién grabado, hay veces que el programador da el mensaje de succesfully programmed, y nos está mintiendo como un bellaco.

Por último, no es conveniente ejecutar otra aplicación mientras se está grabando el PIC o 'distraer' al ordenador con otras aplicaciones ya abiertas.

- *Salidas en colector abierto:*

Acordaros que las salidas en colector abierto no pueden suministrar niveles altos de tensión si no se les colocan resistencias pull-up.

- *Oscilador del PIC:*

Tanto el cuarzo como los condensadores asociados al oscilador deben montarse lo más cerca posible al PIC. Además tienen que estar bien, pero que bien soldados. Uno no sabe muy bien cómo se pueden comportar las pistas ante señales de tan alta frecuencia (y si no pregunta a los de telecos!!).

- *Debug del programa:*

Muchas veces es difícil hacer una traza de errores en un programa del micro si no se puede hacer 'printfs' para ver lo que está pasando dentro del PIC. Por ello es muy recomendable el dejar alguna línea de salida digital libre, de forma que podamos activar desde el programa dicha línea. Así podemos comprobar con un osciloscopio cuándo el PIC atraviesa una instrucción específica. Este truco nos puede servir también para medir tiempos de ejecución.

- *Resistencias en serie:*

Pueden ser para limitar la corriente de entrada/salida de los puertos digitales del PIC o pueden ser útiles para adaptar la impedancia de las entradas analógicas del PIC. Ante la duda, suele ser un valor muy socorrido el de 10k.

- *Todo va mal:*

En ese caso, lo mejor es irse a casa, porque si no se empieza a desvariar completamente.

De todas formas, cuando todavía queda fuerza de voluntad para no irse, conviene empezar repasando estos pocos errores, porque seguro que te has equivocado en alguno de ellos.

# CAPÍTULO 9

# CONSIDERACIONES MECÁNICAS

Cuando se quiere diseñar un microrrobot, lo primero que se debe tener claro es la aplicación para la que se diseña, en este caso, viene definida en los objetivos del proyecto. Una vez decidido para qué servirá, se debe diseñar la estructura mecánica, sobre la que estarán el resto de componentes que lo conforman.

## 9.1 LA ESTRUCTURA O CHASIS

Para la construcción de un microrrobot se pueden utilizar muchos tipos de estructuras, que dependerán de la función que se desea que realice.

Se pueden usar diversos materiales: Plástico, metal o madera, por citar algunos ejemplos. Algunos microrrobots están realizados con estructuras a base de juegos educacionales de construcción como los conocidos Lego o Mecano, muy interesantes dada su flexibilidad, pero a costa de falta de durabilidad ya que se desmontan fácilmente. Otras opciones 'más durables' son los mecanizados en alumnio, acero y metacrilato.

## 9.2 TRACCIÓN Y DIRECCIÓN

A la hora de diseñar la tracción de un microrrobot utilizando ruedas, se puede pensar en varias opciones:

- Estructura formada por dos ruedas de tracción independiente y una rueda loca para darle estabilidad.

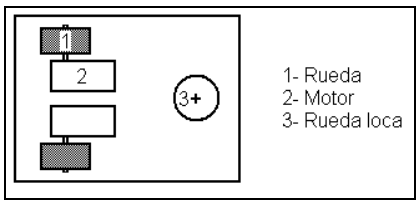



**Figura 57. Arquitectura cinemática de eje motriz-directriz más rueda loca.**

- Arquitectura tipo triciclo, formada por dos ruedas de tracción y una de dirección independientes.

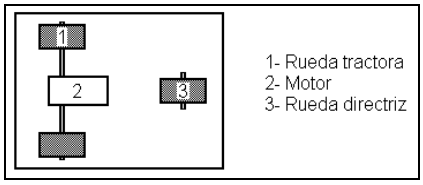



**Figura 58. Arquitectura cinemática eje motriz-eje directriz tipo triciclo.**

- Estructura similar a los coches tradicionales con dos ruedas tractoras con control de la dirección y dos ruedas sin tracción.

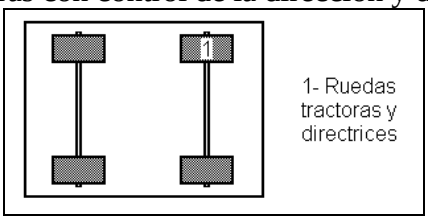



**Figura 59. Arquitectura cinemática eje motriz-eje directriz tipo coche tradicional.**

En el caso de ejes con dos ruedas tractoras, hay que fijarse que, en las curvas, las ruedas interiores giran menos revoluciones que las ruedas exteriores. Por ello es interesante montar estos ejes con una etapa diferencial o por lo que si se quiere evitar que patinen las ruedas.

En el caso de ejes con dos ruedas directrices, existe un problema similar en las curvas. Entonces, y para evitar patinazos y mejorar comportamiento en la toma de curvas, es necesario cumplir con el criterio de Ackerman: la rueda delantera interior debe girar un ángulo ligeramente superior a la exterior.

### 9.2.1 Criterio de Ackerman

El criterio de Ackerman establece una relación entre los ángulos que gira cada rueda. Para un vehículo que circule en curva a una velocidad constante baja. Esta relación asegura la rodadura de todas las ruedas, sin que se produzca deslizamiento, en caso de que la velocidad fuera alta o hubiera aceleraciones aparecerían unas fuerzas de inercia considerables que provocarían fuerzas laterales en los ejes anterior y posterior, y por tanto influirían los ángulos de deriva de los neumáticos (el ángulo de deriva es el ángulo formado entre la dirección real en la que se mueve el vehículo y la dirección apuntada por las ruedas, y depende de la rigidez de los neumáticos y de los esfuerzos a los que estén sometidos).

La relación que establece la geometría de dirección de Ackerman puede deducirse a partir de consideraciones geométricas de la siguiente figura.

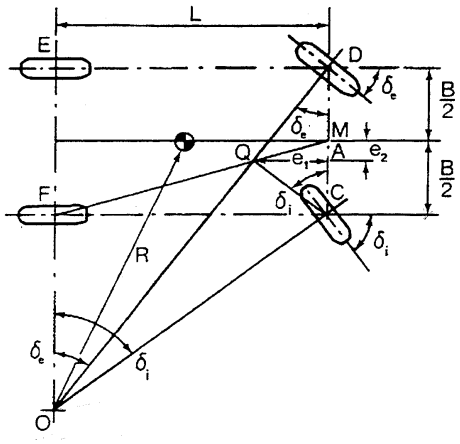



**Figura 60. Criterio de de Ackerman.**

La relación que se obtiene es la siguiente:

$$Co\tan(\delta_e) - Co\tan(\delta_i) = \frac{B}{L}$$

Existe una relación geométrica, que si se cumple, hace que también se cumpla la anterior ecuación. Esta relación se ha representado en la Figura 60 y consiste en que si el punto Q, en el que se cortan las rectas trazadas desde D y C con ángulos $\delta_e$ y $\delta_i$ respectivamente, también pertenece a la recta que une F con el punto M (punto medio del eje anterior), en este caso, se demuestra que se cumple también la ecuación.

# CAPÍTULO 10

# ANEJOS

## 10.1 MANUAL RÁPIDO DEL EAGLE

El editor EAGLE es una sencilla herramienta para diseñar circuitos impresos (PCB's). El nombre EAGLE es un acrónimo (Easily Applicable Graphical Layout Editor).

El programa consta de dos módulos principales:

- Editor de diagramas esquemáticos (*Schematic Editor*): usándolo se introduce el esquemático del cual queremos elaborar la PCB.
- Editor de circuito impreso (*Layout Editor*) que incluye un 'Autouter': a partir del esquemático implementado en el editor de esquemáticos, se genera el PCB.

La versión de evaluación del software permite crear circuitos impresos de hasta dos caras y con área máxima de 100 x 80 mm, suficiente para una gran variedad de circuitos sencillos. La versión de evaluación del programa, así como un tutorial, se pueden descargar de manera gratuita de http://www.cadsoftusa.com

### 10.1.1 Creación de un nuevo proyecto.

Al ejecutar el EAGLE aparece el Panel de control (*Control Panel*). En la columna izquierda hay un árbol de opciones desde donde se accede a todos los módulos del programa, mientras que en la parte derecha hay una descripción de la selección dentro del árbol.

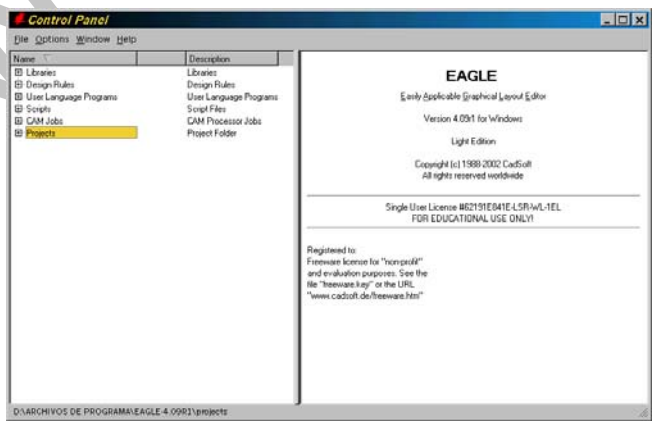



**Figura 61. Panel de control del Eagle.**

Para crear un nuevo proyecto, haga click con el botón derecho en projects y luego click en new project, y escriba el nombre del nuevo proyecto.

### 10.1.2 Creación del diagrama esquemático del circuito.

Para crear el diagrama esquemático del circuito:

- File ⇒ new ⇒ Schematic.
- Se abrirá la ventana del editor de diagramas esquemáticos.

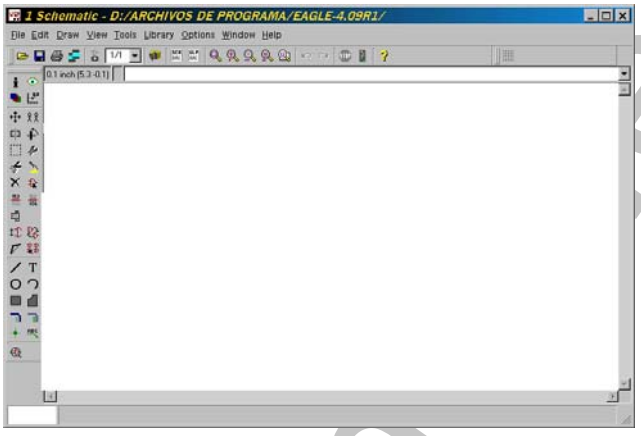



**Figura 62. Editor de esquemáticos.**

- File ⇒ save: entonces se guarda y se le da nombre al nuevo esquema

Ahora hay que comenzar a dibujar el esquema:

- Pinchar el botón *ADD* de la barra de herramientas del lado izquierdo o escribir ADD en la barra superior de entrada de comandos.
- Aparecerá el cuadro de diálogo ADD que muestra todas las librerías de componentes instaladas.
- Se puede explorar las librerías para buscar el componente deseado o usar la opción *search* para buscar todos los componentes de una misma familia. Por ejemplo si desea agregar un buffer 74LS244, se escribe en el campo *search*: 74*244*. En el resultado de la búsqueda aparecerán todos los componentes relacionados.
- Selecciona el componente deseado ⇒ *OK*.
- Finalmente se ubica el componente en el lugar deseado y click para fijarlo.
- Se puede seguir poniendo más unidades del mismo componente o presione ESC para terminar.

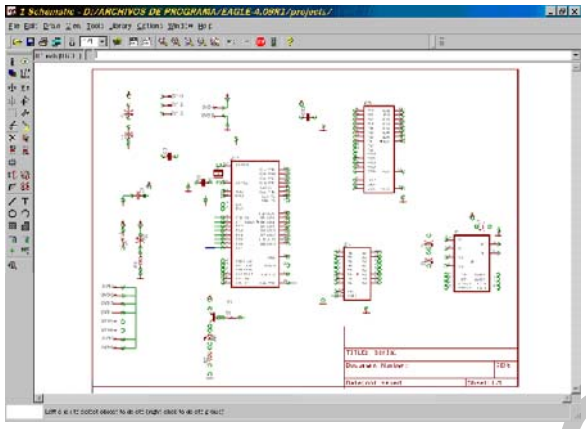


**Figura 63. Ubicación de componentes.**

A continuación hay que hacer las conexiones entre los componentes:

- Pinchar el botón *NET* de la barra de herramientas del lado izquierdo o escribir NET en la barra superior de entrada de comandos. (No usar nunca la opción *WIRE*)
- Para conectar dos hilos: *JUNCTION*.
- Para crear un bus ⇒ BUS + <nombre del bus>
- Para poner etiquetas en el diagrama a las conexiones o a los buses ⇒ *LABEL*.

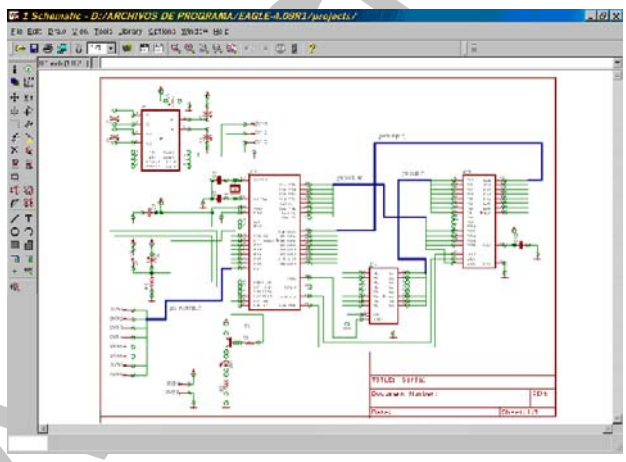


**Figura 64. Esquema listo.**

### 10.1.3 Verificar el circuito

Después de dibujar el circuito en el editor esquemático, se puede utilizar el *Electrical Rule Check (ERC)* para verificar las conexiones:

- Chequear circuito ⇒ *ERC*
- Como resultado se emite un informe con todos los errores y *warnings* detectados.

### 10.1.4 Crear el PCB

Una vez que el circuito esquemático está terminado, se carga el editor de layout:

– Comando *BOARD*

Se abrirá entonces la ventana del editor de circuito impreso, con todos los componentes alineados junto un rectángulo blanco. Todas las conexiones o redes del esquemático se muestran en el editor como líneas finas (*airwires*).

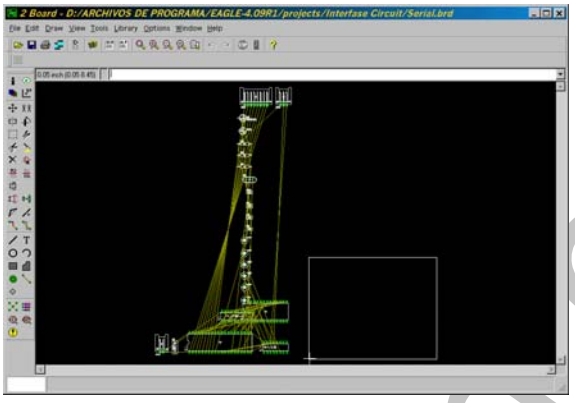


**Figura 65. Layout editor.**

Ahora hay que situar todos los componentes dentro del rectángulo blanco, que define los límites de la PCB que se va a fabricar. Si la versión del eagle es demo, no se podrá ampliar la extensión del rectángulo por defecto.

– Mover componentes ⇒ *MOVE*

– Se puede rotar el componente que está siendo movido ⇒ click botón derecho del ratón

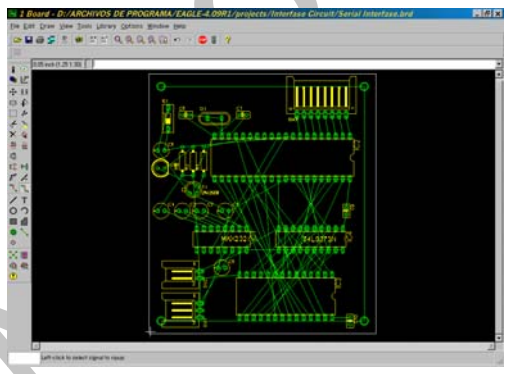


**Figura 66. Componentes dentro de los límites del PCB.**

El siguiente paso es 'rutear las pistas'. Pero antes de intentar 'rutear', es preferible con el *DRC* editor ajustar los tamaños de las pistas, pads, distancias entre las pistas, etc. Las unidades están en <mil.> que son milésimas de pulgada.

Una vez hechos los ajustes, ahora sí que hay que 'rutear'. Para ello hay dos métodos:

– Manual ⇒ comando *ROUTE* ⇒ click en una línea ⇒ click botón derecho ajusta el ángulo.

– Automático ⇒ comando *AUTO* ⇒ Aparecerá un cuadro de diálogo con los parámetros que por omisión usa el Autorouter. Se puede escoger la orientación preferida para las pistas en cada una de las dos caras del circuito (*Top* y *Bottom*). ⇒ aceptar para que el Autorouter comience a

acomodar las pistas del circuito. En circuitos muy complejos puede ser que el Autorouter no consiga completar el 100% de las pistas y que queden algunas líneas sin 'rutear'; en ese caso es necesario 'rutear' manualmente las líneas restantes.

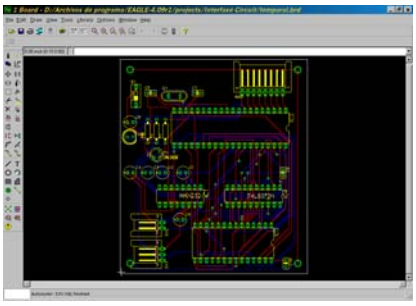


**Figura 67. PCB 'ruteado'.**

- Si se quiere 'desrutear' una pista ⇒ comando *RIPUP* ⇒ click en la pista que se desea 'desrutear' y cuando se desea finalizar al *ESC*.
- Si se quiere 'desrutear' todo ⇒ comando *RIPUP* ⇒ click en la luz verde del semáforo.

El último paso es llenar con polígonos las áreas que no han sido ocupadas por pistas para facilitar el proceso de atacado de la PCB:

- *POLYGON* ⇒ En la barra de herramientas superior puede cambiar los parámetros del comando como separación entre pistas y el polígono, patrón de llenado, etc.
- Para cubrir todas las áreas libres de la PCB, de una sola vez, hay hacer click en la parte superior izquierda del rectángulo límite de la PCB y describir un rectángulo que cubra toda la PCGB haciendo haciendo clicks en los sucesivos vértices terminar de nuevo en el vértice se comenzó.
- Finalmente, pulsar *RATSNEST*.
- El diseño está completo.

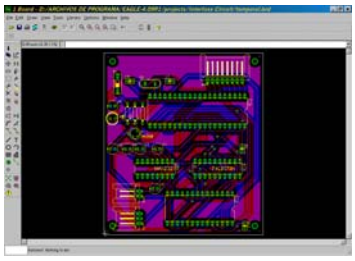


**Figura 68. Diseño del PCB terminado.**

NOTA: La cara en la que se está dibujando el polígono, depende de la capa (*Layer*) que está seleccionada en la barra de herramientas superior.

### 10.1.5 Impresión de la máscara

Una vez que el diseño está completo, se debe realizar la máscara que se usará para insolar el PCB. Para ello basta con imprimir el diseño desde el *layout* editor. Antes de mandar imprimir, con el comando *DISPLAY* hay que desactivar todas las capas que no deseamos sean impresas. Se debe conseguir que se queden activas las siguientes capas: *Top* ó *Bottom* (según la cara del diseño que se va a imprimir), *Pads*, *Vías* y *Dimension*. Las opciones se activan y desactivan haciendo click en el número de la opción. Además es conveniente quitar las opciones de color para que la impresión se realice en blanco y negro. En este caso el diseño se muestra en blanco y negro.

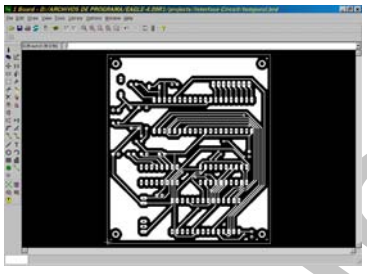


**Figura 69. Preparación para imprimir la máscara.**

Ya sólo queda imprimir, a la mayor resolución posible (mín. 600dpi), la máscara que se utilizará para el atacado en ácido. Si la impresora no puede imprimir a calidad alta o tiene poca tinta, un truco fácil es imprimir dos máscaras, y, a la hora de insolar, superponerlas.

## 10.2 PATILLAJE DEL PIC16F84

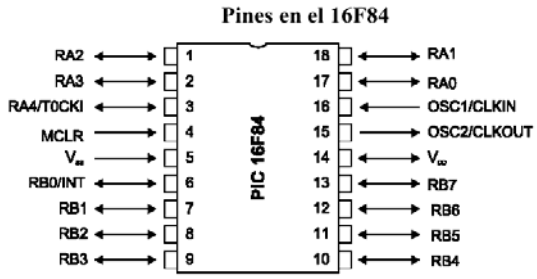



**Figura 70. Patillaje del 16F84.**

## 10.3 MANUAL DE REFERENCIA DEL PIC16F877

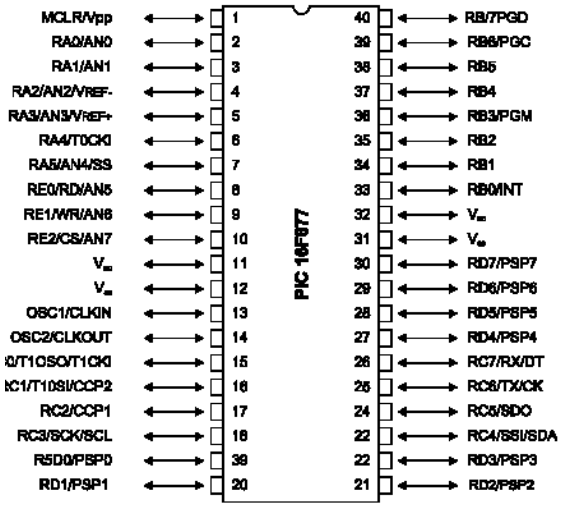



**Figura 71. Patillaje del 16F877.**

Los periféricos que tiene el PIC16F877 son:

- Puertos programables de E/S
- Timers/Counters
- Puertos de captura/comparación de datos
- Moduladores de ancho de pulso (PWM)
- Conversor Analógico/Digital de 10 bits
- Puerto serie síncrono
- USART
- Parallel Slave Port

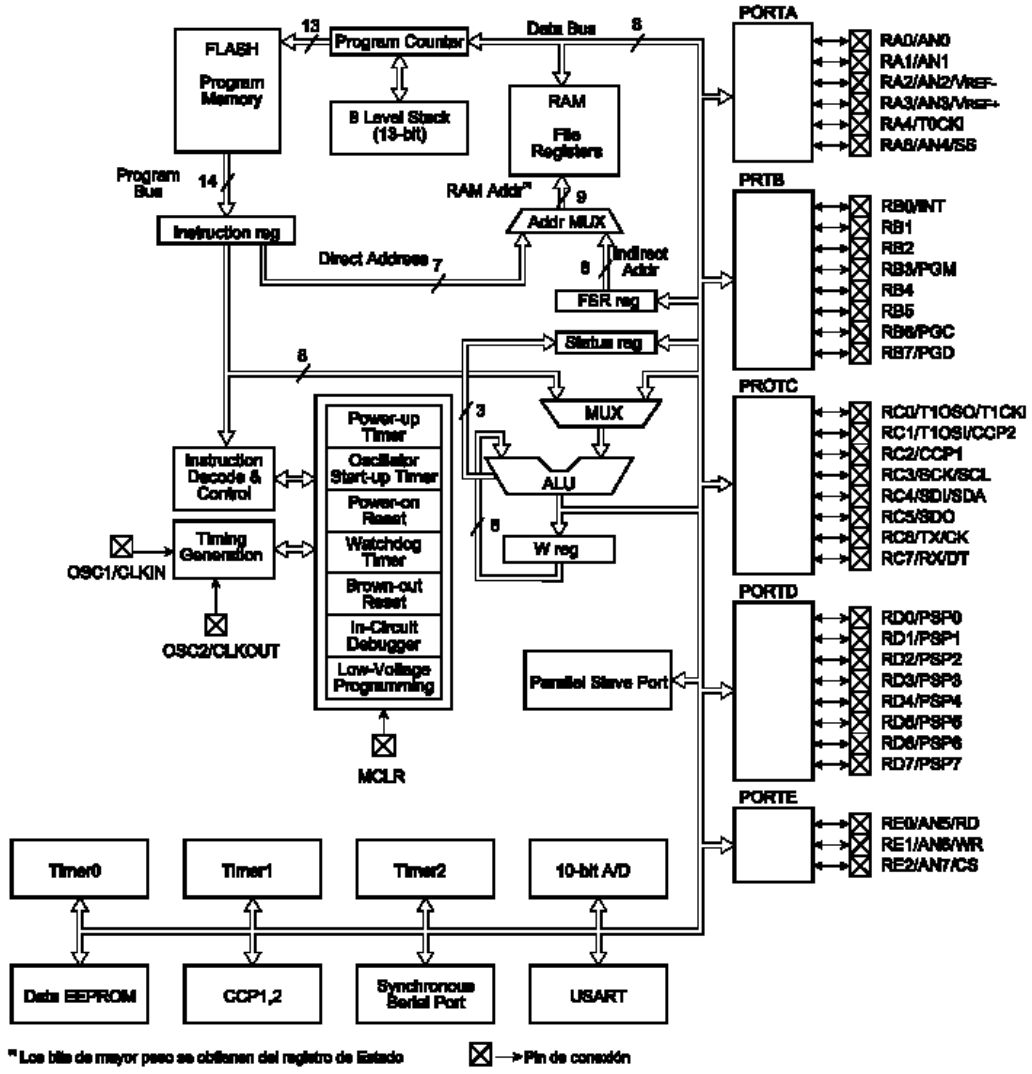



**Figura 72. Arquitectura del PIC 16F87x.**

### 10.3.1 Puertos programables de E/S.

Pueden disponerse hasta de 33 pines bits de E/S. Sin embargo y debido a que los pines del PIC pueden tener otras funciones, el número de E/S puede ser bastante inferior. Las principales características pueden resumirse en:

- Programables como entradas o salidas individualmente.
- Corriente máxima de cada línea ⇒ 25 mA (pueden alimentar un LED directamente). No obstante la corriente total en los puertos A, B y E no puede superar los 200 mA. y en los puertos C y D otros 200 mA.
- Entradas tipo TTL o ST (Schmitt Trigger).
- Resistencias Pull-up (habilitables por programa) en el puerto B

Cada puerto de E/S cuenta dos registros TRISX y PORTX. El registro TRISX tiene asociado un bit a cada línea del puerto: si el bit se pone a 1, la línea funciona como entrada, y si se pone a 0, la línea funcionará como salida:

- 1⇒input
- 0⇒output

El segundo registro, PORTX, nos permite acceder al puerto. Si se escribe en este registro, se modifica el estado de cada línea que se haya programado como salida. En modo lectura, nos informa del estado de las líneas (E/S).

No es conveniente realizar una lectura del estado del puerto inmediatamente después de haber realizado una operación de escritura sobre el mismo.

Las líneas de E/S están agrupadas en 5 puertos:

- Puerto A: 6 bits ( pines RA0-RA5),

  El puerto A tiene todas sus salidas Totem pole, excepto la RA4 que es del tipo Open collector (open drain de forma más exacta). RA4 puede funcionar como entrada de reloj externo del TMR0. RA0-RA3 y RA5 están multiplexados con el conversor A/D.

- Puerto B: 8 bits (pines RB0-RB7)

  Los pines del puerto RB3, RB6 y RB7 están multiplexados con el módulo de programación del PIC.

  Otra característica del puerto B es que dispone de resistencias Pull-up programables, es decir que pueden activarse por software cuando la línea correspondiente funciona como entrada. La activación se realiza con el bit RBPU (bit 7) del registro OPTION_REG (direcciones 81h y 181h).

  Además la línea RB0 puede funcionar como entrada de petición de interrupción por flanco de subida o flanco de bajada. Para ello se debe activar el bit INTE (bit 4) del registro INTCON (direcciones 0Bh, 8Bh, 10Bh y 18Bh). Con el bit INTDEG (bit 6) de OPTION_REG (direcciones 81h y 181h) se selecciona si se activa la interrupción con flanco de subida (1) o de bajada (0). Con el bit INTF (bit 1) del registro INTCON (direcciones 0Bh, 8Bh, 10Bh y 18Bh) puede verse si RB0 ha generado una interrupción.

  Los bits RB4 a RB7 también pueden generar una interrupción especial llamanda ‘Interrupt-on-change’, es decir, se solicita una interrupción cada cada vez que se detecta un cambio de estado en cualquiera de los bits mencionados. Este tipo de interrupción es bastante usado en teclados. Esta interrupción está controlada por los bits RBIF y el RBIE del INTCON.

- Puerto C: 8 bits (RC0-RC7)

  Este puerto tiene asociados otros periféricos que se resumen en:

  - RC0: Salida del oscilador o entrada del reloj del TMR1.
  - RC1: Entrada del oscilador del TMR1 o entrada del Capture2 o salida del Capture2 o salida del PWM2
  - RC2: Entrada del Capture1 o salida del Capture1 o salida del PWM1
  - RC3: Entrada/salida de reloj para modo síncrona del módulo SPI, entrada/salida de reloj para modo síncrona del módulo $I^2C$

- RC4: Entrada de datos en modo SPI, entrada/salida de datos en modo $I^2C$
- RC5: Salida de datos en modo SPI
- RC6: Salida asíncrona del USART (RS232)
- RC7: Entrada asíncrona del USART (RS232)

- Puerto D: 8 bits (RD0-RD7)

  La principal característica del puerto D es que tiene dos modos de funcionamiento: o como un puerto adicional normal del PIC de 8 bits o como bus de datos del puerto paralelo esclavo.

- Puerto E (3 bits):

  Las funciones multiplexadas que presenta el puerto E son las siguientes:

  - RE0: bit de control de lectura del puerto paralelo esclavo (RD, activo en bajo) o entrada analógica 5
  - RE1: bit de control de escritura del puerto paralelo esclavo (WR, activo en bajo) o entrada analógica 6
  - RE2: bit de control de selección del puerto paralelo esclavo (CD, activo en bajo) o entrada analógica 7.

Los puertos A y B cuando se configuran como entradas, trabajan con niveles TTL (salvo la línea RA4), y los puertos C, D y E como Schmitt Trigger (también RA4)[3].

### 10.3.2 Timers

El PIC tiene 3 temporizadores, numerados del 0 al 2, y un Watch Dog Timer.

## Timer0

Timer0 es un temporizador/contador de 8 bits de resolución. Puede ser leído y escrito a través del registro TMR0 (direcciones 01h y 101h). La entrada de reloj del temporizador puede seleccionarse que sea interna o externa:

- Interna: Frecuencia de reloj CPU dividida por 4.
- Externa: Entrada RA4/T0CKI.

Para configurar la entrada del reloj debe modificarse el bit T0CS (bit 5) del OPTION_REG (direcciones 81h y 181h), de forma que si se escribe un cero, se selecciona el reloj interno (modo temporizador), y si se escribe un 1, se selecciona el reloj externo (modo contador).

Si se usa un reloj externo, la frecuencia máxima de entrada estará limitada a la mitad de la frecuencia de la entrada de reloj de la CPU.

La función de reloj externo puede ser interesante para contar pulsos/eventos a través de la entrada RA4/T0CKI. En este caso se puede seleccionar si se activa la entrada por flanco de subida o de bajada, escribiendo en el bit T0SE (bit 5) del

[3] Las entradas Schmitt trigger permiten cambios lentos en las variaciones de los niveles lógicos

OPTION_REG (direcciones 81h y 181h). Un 0 provocaría la detección del flanco de subida, y con un 1 se detectaría el flanco de bajada.

El Timer0 admite un *preescaler* o circuito divisor de frecuencia. Para ello se emplea el bit PSA (bit 3) del OPTION_REG (direcciones 81h y 181h). Este preescaler está compartido por el Watchdog, es decir, si se pone el bit a 0, se asigna el preescaler al Timer0, y en el caso de que se escriba un 1 en ese bit, el preescaler será asignado al Watchdog. Para ajustar el factor de división del preescaler se disponen de tres bits: PS2, PS1 y PS0, (bits 2, 1 y 0) del OPTION_REG (direcciones 81h y 181h) que introducen los factores de división mostrados en la siguiente tabla:

| Bits PS | Timer0 | Watch dog |
|---|---|---|
| 000 | 1:2 | 1:1 |
| 001 | 1:4 | 1:2 |
| 010 | 1:8 | 1:4 |
| 011 | 1:16 | 1:8 |
| 100 | 1:32 | 1:16 |
| 101 | 1:64 | 1:32 |
| 110 | 1:128 | 1:64 |
| 111 | 1:256 | 1:128 |

**Figura 73. Configuración del factor de división del preescaler del Timer0 y del Watchdog.**

Si se quiere que se genere una interrupción cada vez que se produzca un desbordamiento/overflow del contador (el valor de contaje pasa de FFh a 00h), se debe poner a uno el bit T0IE (bit 5) del registro INTCON (0Bh y 8Bh). A través del bit T0IF (bit 2) del registro INTCON (0Bh y 8Bh) puede comprobarse si el timer0 ha generado una interrupción.

## Timer1

El Timer1 es un temporizador/contador de 16 bits. El valor de contaje se accede a través de los registros TMR1H (0Fh) y TMR1L (0Eh).

Tal y como sucede ya con el Timer0. la entrada de reloj del temporizador puede ser interna o externa:

- Interna: Frecuencia de reloj CPU dividida por 4.
- Externa: Entrada RC1/T1OSICCP2 y RC0/T1OSO/T1CKI.

La configuración de la entrada se realiza a través del bit TMR1CS (bit 1) del registro T1CON (10h):

- 0⇒reloj interno.
- 1⇒reloj externo (frec. Max. Es la mitad de la frec. Del reloj del PIC), y la activación del timer se produce en el flanco de subida si y sólo si se ha producido antes un flanco de bajada.

Además se dispone de un bit que permite activar y desactivar el Timer: bit TMR1ON (bit 0) del registro T1CON (10h).

Nuevamente, este temporizador dispone de un preescaler no compartido. Este preescaler está controlado por los bits T1CKPS0 (bit 4) y T1CKPS1 (bit 5) del registro T1CON (10h). Los factores de división vienen dados por la siguiente tabla:

| Bits T1CKPS | Timer1 |
|---|---|
| 00 | 1:1 |
| 01 | 1:2 |
| 10 | 1:4 |
| 11 | 1:8 |

**Figura 74. Configuración del factor de división del preescaler del Timer1.**

Para activar las interrupciones en caso de desbordamiento hay que poner a uno el bit TMR1IE (bit 0) del registro PIE1 (8Ch). Además también hay que activar los bits que habilitan las interrupciones de los periféricos (bit PEIE, bit 6 del registro INTCON, direcc. 0Bh y 8Bh), y el general de activación de las interrupciones (bit GIE, bit 7 del registro INTCON, direcc. 0Bh y 8Bh). A través del bit TMR1IF (bit 0) del registro PIR1 (0Ch) puede comprobarse si el timer1 ha generado una interrupción.

También se puede conectar un oscilador de cristal de cuarzo en los pines T1OSI y T1OSO. Entonces hay que poner a uno el bit T1OSCEN (bit 3) del registro T1CON (10h). Este modo está pensado para conectarle un oscilador de 32Khz y así tener una base de tiempos para programar un reloj de tiempo real que mida segundos con alta precisión. La siguiente tabla muestra los valores de los 2 condensadores que deben conectarse dependiendo de la frecuencia del cristal:

| Frecuencia (Khz) | capacidad (pF) |
|---|---|
| 32 | 33 |
| 100 | 15 |
| 200 | 15 |

**Figura 75. Capacidades que deben asociarse al cristal del Timer1.**

## Timer2

El Timer2, a pesar de ser un temporizador de 8 bits, el funcionamiento interno es similar al Timer1, con las siguientes salvedades:

El valor de cuenta se lee en el registro TMR2 (11h) y el valor máximo de cuenta se almacena en el registro PR2 (92h). Es decir, cuando TMR2 alcanza el valor de PR2, se produce un overflow y TMR2 pasa a 00h.

La entrada de reloj del temporizador es interna, concretamente la frecuencia de reloj CPU dividida por 4.

El bit para activar/desactivar el temporizador es el TMR2ON (bit 2) del registro T2CON (12h).

El valor del preescaler está controlado por los bits T2CKPS0 (bit 0) y T2CKPS1 (bit 1) del registro T2CON (12h). Los divisores vienen dados por la siguiente tabla:

| Bits T2CKPS | Timer2 |
|---|---|
| 00 | 1:1 |
| 01 | 1:4 |
| 1x | 1:16 |

**Figura 76. Configuración del factor de división del preescaler del Timer2.**

Además de preescaler, el TMR2 tiene un postescaler que se activa con TOUTPS0-TOUTPS3 (bits 3 a 6) del registro T2CON (12h). Los valores del

postescaler van desde 1:1 a 1:16 que van codificados de la misma forma que los bits del preescaler.

Finalmente existe un bit de activación de interrupción en caso de que se produzca un overflow, bit TMR2IE (bit 1 de registro PIE1, 8Ch). Nuevamente y tal como sucedía con el Timer1, también hay que activar los bits que habilitan las interrupciones de los periféricos (bit PEIE, bit 6 del registro INTCON, direcc. 0Bh y 8Bh), y el general de activación de las interrupciones (bit GIE, bit 7 del registro INTCON, direcc. 0Bh y 8Bh). A través del bit TMR2IF (bit 1) del registro PIR1 (0Ch) puede comprobarse si el timer2 ha generado una interrupción.

Este contador/temporizador puede usarse como base de tiempo para el módulo de captura/comparación/PWM y la puerta asíncrona (SSP).

## Watch dog Timer

Este temporizador reinicia la CPU si se llega a producir un overflow. Gracias a este mecanismo, si la ejecución del micro se queda 'colgada' en un punto, automáticamente el micro se resetea y comienza a ejecutar otra vez el programa. Así, el microcontrolador puede recuperarse después de un error de ejecución/ o pérdida del control del programa.

Para activar el Watchdog, hay que modificar los bits de configuración del microcontrolador. Éstos bits sólo se pueden modificar en el momento en el que se descarga el programa en el chip.

El tiempo que el Watchdog tarda en alcanzar el overflow, con un Preescaler 1:1, puede oscilas entre 7 y 33 ms, con un valor típico de 18 mS. Esta variabilidad se debe a la imprecisión del oscilador interno que depende de la temperatura y la tensión de alimentación. Este intervalo puede variarse con el preescaler mencionado en el apartado del Timer1.

### 10.3.3 CAPTURE/COMPARE/PWM

El PIC 16F87x dispone de dos módulos llamados CCP: CCP1 y CCP2. Ambos presentan el mismo funcionamiento con la salvedad de que el periférico CCP2 podrá ser programado para que también reciba como valor de entrada una conversión proveniente del el módulo A/D.

Tanto el CCP1 como el CCP2 trabajan con una resolución de 16bit:

- Módulo CCP1: CCPR1H (16h) y CCPR1L (15h).
- Módulo CCP2: CCPR2H (1Ch) y CCPR2L (1Bh).

Además de los registros de datos, también existe otro de configuración:

- Módulo CCP1: CCP1CON (17h)
- Módulo CCP2: CCP2CON (1Dh)

Presentan tres modos de trabajos denominados captura, comparación y PWM. Estos modos se configuran a través de los bits CCPxM0 a CCPxM3 del registro CCPxCON (17h, para CCP1, ó 1Eh, para CCP2):

| CCPxM3:CCPxM0 | Modo de trabajo |
|---|---|
| 0000 | módulo CCPx desactivado (por defecto) |
| 0100 | Modo captura por flanco ascendente |
| 0101 | Modo captura por flanco descendente |
| 0110 | Modo captura cada 4 flancos ascendentes |
| 0111 | Modo captura cada 16 flancos ascendentes |
| 1000 | Modo comparación, línea RCx/CCPx se pone a 1 si comparación OK y CCPxIF se pone a 1 |
| 1001 | Modo comparación, línea RCx/CCPx se pone a 0 si comparación OK y CCPxIF se pone a 1 |
| 1010 | Modo comparación, línea RCx/CCPx inalterada si comparación OK y CCPxIF se pone a 1 |
| 1011 | Modo comparación, si comparación OK y CCPxIF se pone a 1 y comienza una conversión A/D si este módulo está activo |
| 11xx | Modo PWM |

**Figura 77. Configuración del factor de división del preescaler del Timer2.**

## Modo captura

En el modo captura, el módulo CCPx captura el contenido del Timer1 cuando se produce una transición de niveles de señales en las RC1/T1OSI/CCP2 y RC2/CCP1. Pueden programarse para detectarse tanto flancos de subida como de bajada. Este modo de trabajo puede resultar extremadamente útil para calcular tiempos entre dos sucesos de forma muy precisa. Si además las interrupciones están activas, se generará una por cada captura realizada.

## Modo comparación

En este modo, el módulo CCPx comparará el contenido de los registros de 16 bits del propio módulo con el valor de contaje del Timer1. En el momento en el cual se produzca una coincidencia, se generará una interrupción (si éstas están activas). Además podrá programarse para que las líneas RC1/T1OSI/CCP2 ó RC2/CCP1 se pongan a nivel alto, bajo o no cambien.

A continuación se reseteará el Timer1. Y en el caso del CCP2, simultáneamente se podrá lanzar una conversión A/D (en este último caso, se observa que fácilmente pueden programarse muestreos A/D periódicos)

## Modo PWM

En este caso se produce una salida PWM en el puerto CCPx, con una resolución de 10 bits:

- 8 bits del registro CCPRxL (15h, para CCP1 ó 1Bh, para CCP2)
- 2 bits de los bits CCPxX (bit 5) y CCPxY (bit 4) del registro CCPxCON (17h, para CCP1, ó 1Eh, para CCP2).

La frecuencia de la señal PWM se establece por medio del Timer2 el duty-cycle se controla por medio de los registros del CCP que se esté utilizando.

El período del PWM viene dado por la siguiente fórmula:

$$T_{PWM}=[(PR2)+1]*4*T_{osc}*Prescaler\_TMR2$$

El valor del duty-cycle es:

$Duty_{PWM}$=[(CCPRxL:CCPxCON<5:4>]*4*$T_{osc}$*Prescaler_TMR2

Para configurar correctamente el modo PWM hay que seguir los siguientes pasos:

- Escribir el período del PWM en el registro PR2 (92h).
- Definir el duty-cycle en CCPRxL (15h, para CCP1 ó 1Bh, para CCP2) y en CCPxCON (bits 5 y 4 en la dirección 17h para CCP1, y bits 5 y 4 en la dirección 1Eh para CCP2).
- Configurar la línea de E/S como salida actuando sobre el registro TRIS correspondiente.
- Definir el valor del preescaler y habilitar el timer2
- Configurar el módulo CCPx para la operación en PWM.

### 10.3.4 Master Synchronous Serial Port

Este módulo implementa un interfaz de comunicaciones en serie. Existen multitud de dispositivos que permiten este tipo de comunicación, tales como memorias EEPROM, conversores A/D, sensores de diferentes tipos (temperatura, posición, distancia, ...), displays, otros microcontroladores, PCs...

El módulo puede operar en dos modos de funcionamiento: SPI e $I^2C$.

## SPI: Serial Peripheral Interface

Este interface suele ser utilizado para comunicar dos dispositivos entre si, uno se configura como master y el otro como esclavo. En este interface se definen las líneas:

- Serial Data Out (SDO): pin (RC5/SDO).
- Serial Data In (SDI): pin (RC4/SDI/SDA).
- Serial Clock (SCK): pin (RC3/SCK/SCL/LVDIN).

Adicionalmente se puede usar un cuarto pin en modo esclavo.

- Slave Select (SS): pin (RA5/SS/AN4)

Permite velocidades de hasta 8 Mbps con reloj a 20 MHz. El master inicia las transferencias activando la línea SCK. Entonces el esclavo recibe y transmite información cuando detecta que la línea SCK se activa. Cada vez que recibe un byte, en modo esclavo, se puede activar la interrupción correspondiente.

Este módulo usa los registros:

- SSPCON (14h): configuración/control.
- SSPSTAT (94h): estado.
- SSPBUF (13h): buffer de transmisión/recepción.
- SSPSR (no accesible): registro de desplazamiento.

Cuando llega un byte completo, se activa el bit SSPIF (bit 3 del registro PIR1) y se genera una interrupción (si está activa).

## $I^2C$: Inter-Integrated Circuit

Permite la interconexión de múltiples dispositivos formando un bus. A cada dispositivo se le asigna una dirección y los paquetes de datos que se envían por el bus llevan direccionamiento para identificar el dispositivo emisor/receptor. En general se configura un dispositivo como master y el resto como esclavos, no obstante, se permiten buses multi-master, en este caso se dispone de mecanismos de detección de colisiones y gestión del Bus.

En este interface se definen las líneas:

- Serial Data (SDA): pin (RC4/SDI/SDA).
- Serial Clock (SCL): pin (RC3/SCK/SCL/LVDIN).

Se pueden alcanzar velocidades de 1 Mbps, y conectar hasta 128 o 1.024 dispositivos en función del tipo de direccionamiento utilizado.

Este módulo usa los registros:

- SSPCON (14h): configuración/control.
- SSPCON2 (91h): configuración/control.
- SSPSTAT (94h): estado.
- SSPBUF (13h): buffer de transmisión/recepción.
- SSPSR (no accesible): registro de desplazamiento.

### 10.3.5 Addressable Universal Synchronous Asynchronous Receiver Transmitter (USART).

Este módulo implementa el popular interface de comunicaciones serie que tienen incluido gran cantidad de ordenadores. Permite comunicación serie entre dos dispositivos, y en algunos modos de funcionamiento, permite la conexión de más de dos dispositivos incluyendo 9 bits de direccionamiento (512 dispositivos).

A la USART también se le conoce como Serial Communications Interface (SCI). Se puede configurar como:

- Asíncrono full-duplex.
- Síncrono-Master half-duplex.
- Síncrono-Slave half-duplex.

Para que sea posible la comunicación por el protocolo RS232, es necesario conectar un MAX232 entre el UART del PIC y la línea serie. La función de este chip (el MAX232) es adecuar los niveles de tensión para que sean compatibles con la norma RS232.

Los registros asociados al puerto serie son:

- TXSTA (98h): estado y control de transmisión.
- RCSTA (18h): estado y control de recepción.
- SPBRG (99h): configuración del generador de baudios.
- TXREG (19h): buffer de transmisión.

- RXREG (1Ah): buffer de recepción.

### 10.3.6 Analog/Digital Converter.

El PIC 16F87x dispone de 8 entradas analógicas (AN0-AN7), permitiéndose así la conexión de entradas analógicas para convertirlas en valores discretos. El conversor realiza una operación se muestreo y retención (sample and hold) y la conversión a digital se realiza por aproximaciones sucesivas.

Debido al elevado precio de los módulos de conversión analógico/digital, todas las entradas comparten un único conversor. A pesar de todo, el PIC puede programarse para realizar conversiones desde las distintas entradas/canales gracias a un multiplexor. El multiplexor se controla a través de los bits CHS0 (bit 3) a CHS2 (bit 5) del registro ADCON0 (1Fh):

| CHS2:CHS0 | Canal seleccionado |
|---|---|
| 000 | canal 0 (pin RA0/AN0) |
| 001 | canal 1 (pin RA1/AN1) |
| 010 | canal 2 (pin RA2/AN2/ $V_{ref}$-) |
| 011 | canal 3 (pin RA3/AN3/$V_{ref}$+) |
| 100 | canal 4 (pin RA5/AN4) |
| 101 | canal 5 (pin RE0/AN5) |
| 110 | canal 6 (pin RE1/AN6) |
| 111 | canal 7 (pin RE2/AN7) |

**Figura 78. Multiplexado del canal analógico (ADCON1).**

Las tensiones de referencia para el módulo conversor y el número de canales activos para la conversión se configuran por software: bits PCFG0 (bit 0) a PCFG3 (bit 3) del registro ADCON1 (9Fh):

| PCFG3:PCFG0 | AN7(1) RE2 | AN6(1) RE1 | AN5(1) RE0 | AN4 RA5 | AN3 RA3 | AN2 RA2 | AN1 RA1 | AN0 RA0 | VREF+ | VREF- | CHAN / REFS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | A | A | A | A | A | A | A | A | VDD | VSS | 8/0 |
| 0001 | A | A | A | A | VREF+ | A | A | A | RA3 | VSS | 7/1 |
| 0010 | D | D | D | A | A | A | A | A | VDD | VSS | 5/0 |
| 0011 | D | D | D | A | VREF+ | A | A | A | RA3 | VSS | 4/1 |
| 0100 | D | D | D | D | A | D | A | A | VDD | VSS | 3/0 |
| 0101 | D | D | D | D | VREF+ | D | A | A | RA3 | VSS | 2/1 |
| 011x | D | D | D | D | D | D | D | D | VDD | VSS | 0/0 |
| 1000 | A | A | A | A | VREF+ | VREF- | A | A | RA3 | RA2 | 6/2 |
| 1001 | D | D | A | A | A | A | A | A | VDD | VSS | 6/0 |
| 1010 | D | D | A | A | VREF+ | A | A | A | RA3 | VSS | 5/1 |
| 1011 | D | D | A | A | VREF+ | VREF- | A | A | RA3 | RA2 | 4/2 |
| 1100 | D | D | D | A | VREF+ | VREF- | A | A | RA3 | RA2 | 3/2 |
| 1101 | D | D | D | D | VREF+ | VREF- | A | A | RA3 | RA2 | 2/2 |
| 1110 | D | D | D | D | D | D | D | A | VDD | VSS | 1/0 |
| 1111 | D | D | D | D | VREF+ | VREF- | D | A | RA3 | RA2 | 1/2 |

| U-0 | U-0 | R/W-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| ADFM | — | — | — | PCFG3 | PCFG2 | PCFG1 | PCFG0 |
| bit7 | | | | | | | bit0 |

**Figura 79. Configuración de las entradas analógicas a través del registro ADCON1.**[4]

[4] Es recomendable seleccionar como entrada, a través de TRISx, las líneas que se desean utilizar en el conversor A/D.

La frecuencia de muestreo viene dada por la frecuencia del oscilador principal del PIC o bien por un oscilador interno RC del PIC. En el caso de encontrarse seleccionado el oscilador principal, además puede programarse un preescaler para que el muestreo sea más lento. Los bits encargados de estas funciones de selección del reloj son el ADSC2 (bit 6) del registro ADCON1 (9Fh), y ADSC1 y ADSC0 (bits 7 y 6) del registro ADCON0 (1Fh). La siguiente tabla resume las opciones de selección:

| ADCS2:ADCS0 | Canal seleccionado |
|---|---|
| 000 | $F_{osc}/2$ |
| 001 | $F_{osc}/8$ |
| 010 | $F_{osc}/32$ |
| 011 | oscilador interno (conversión A/D funciona en estado SLEEP) |
| 100 | $F_{osc}/4$ |
| 101 | $F_{osc}/16$ |
| 110 | $F_{osc}/64$ |
| 111 | oscilador interno (conversión A/D funciona en estado SLEEP) |

**Figura 80. Selección del reloj para muestreado A/D.**

Una vez configurado el módulo del conversor, el muestreado no comenzará hasta que no se active el bit ADCON (bit 0) del registro ADCON0 (1Fh). Cada vez que se realice una muestra, de activará el bit G0/DONE (bit 3) del registro ADCON0 (1Fh). Para que se pueda producir la siguiente muestra no hay que olvidarse poner el bit G0/DONE a 1.

El valor de conversión se encuentra en los registros ARDESH (bits altos, 1Eh) y ARDESL (bits bajos, 9Eh).

Se puede programar para que el conversor interrumpa cada vez que finalice una conversión. Para ello, hay que activar los dos bits de control de interrupción del módulo ADIE (bit 6 de registro PIE1, 8Ch) y ADIF (bit 6 del registro PIR1, 0Ch). Nuevamente y tal como sucedía con los demás módulos, también hay que activar los bits que habilitan las interrupciones de los periféricos (bit PEIE, bit 6 del registro INTCON, direcc. 0Bh y 8Bh), y el general de activación de las interrupciones (bit GIE, bit 7 del registro INTCON, direcc. 0Bh y 8Bh).

En resumen, la configuración y lectura del conversor A/D debe realizarse como sigue:

- Configurar el convertidor A/D:
  - configurar los canales y las referencias de tensión deseadas (ADCON1).
  - Seleccionar el reloj y el canal de entrada a muestrear (ADCON0).
- Activar (si se quiere) las interrupciones A/D:
  - bit ADIE de PIE1.
  - bits GIE y PEIE de INTCON.
- Activar el módulo (ADON).
- Esperar a la carga del condensador del hold (típ. 20 μs). Esta espera debe realizarse cada vez que se haga un cambio de canal o haya un cambio brusco en la tensión de entrada.

- Iniciar la conversión (activar el bit GO/GONE).
- Esperar el tiempo requerido para la adquisición
  - O bien encuestando a GO/GONE hasta que se resetee.
  - O bien esperando a la interrupión (si se ha programado)
- Leer el valor de conversión de los registros ARDESH:ADRESL
- Resetear ADIF (si están programadas las interrupciones)
- Repetir el proceso desde el punto 6. Si se quiere cambiar de canal no hay que olvidarse de multiplexar el conversor.

### 10.3.7 Sleep

El PIC tiene un modo de funcionamiento de muy bajo consumo llamado SLEEP. Cuando le damos al PIC la orden de SLEEP se desconecta el circuito de excitación del cristal de cuarzo y hace que todos los periféricos que dependan de él se detengan, incluida la CPU.

En este modo el consumo del chip es del orden del µA (ahorrándose baterías!!).

Para entrar en modo SLEEP hay que ejecutar la instrucción SLEEP y el microcontrolador se despertará al producirse un reset o una interrupción.

Cuando se hace SLEEP el pic hace prefetch de la siguiente instrucción por lo que al despertarse por interrupción se ejecutará la siguiente instrucción del SLEEP y a continuación se saltará a la dirección de atención a interrupción siempre y cuando el bit GIE de INTCON esté activo. No hace falta activar GIE para que se despierte por interrupción, pero si hace falta tener activado el bit de habilitación de la interrupción del periférico que tienen que despertar al micro.

Los periféricos que pueden provocar interrupción mientras el PIC está en sleep son:

- Lectura o escritura del PSP (Puerto Paralelo eSclavo).
- El TMR1 en modo contador asíncrono (excitado externamente).
- Captura del CCP (empleando el oscilador RC como base de tiempos)
- Eventos especiales de disparo del TMR1 cuando se usa con reloj externo.
- El bit start/stop (SSP).
- El SSP en transmisión o recepción en modo esclavo (SPI/I2C).
- RX o TX de la USART en modo esclavo.
- El convertidor A/D cuando usa el oscilador RC interno.
- Al finalizar una operación de escritura en la EEPROM.
- El Watchdog (ya que usa un reloj RC interno).

Los siguientes eventos también provocan la salida del modo SLEEP: reset del WDT, reset externo mediante MCLR, interrupciones externas en RB0/INT y los cambios de estado de las patas RB7:RB4.

### 10.3.8 Reset

En el PIC puede provocarse un reset de seis formas diferentes:

- Power-On Reset (POR): encendido normal.
- MCLR: reset durante funcionamiento normal.
- MCLR: reset en estado de SLEEP.
- WDT reset durante funcionamiento normal.
- WDT reset en estado de SLEEP.
- Brown-Out Reset (BOR): reset por caída de tensión de alimentación.

Es posible determinar el motivo del reset mediante software. Para ello el programador dispone de los bits POR y BOR del registro PCON (83h) y de los bits /TO (bit 4) y /PD (bit 3) del registro STATUS (03h, ó 83h, ó 103h, ó 183h). La siguiente tabla muestra el estado de estos bits tras cada uno de los posibles resets del PIC:

| POR | BOR | /TO | /PD | |
|---|---|---|---|---|
| 0 | x | 1 | 1 | Encendido normal |
| 0 | x | 0 | x | Ilegal |
| 0 | x | x | 0 | Ilegal |
| 1 | 0 | 1 | 1 | Bajada de tensión |
| 1 | 1 | 0 | 1 | WDT Reset |
| 1 | 1 | 0 | 0 | WDT Reset en estado de SLEEP |
| 1 | 1 | u | u | Activación de MCLR reset |
| 1 | 1 | 1 | 0 | MCLR reset o interrupción durante SLEEP |

(x= indefinido, u=sin cambio)
(POR y BOR son W/R; TO y PD son solo R)

**Figura 81 Tipos de resets.**

## MCLR Master clear

Es el pin de reset que se activa con 0V. Si no se desea reset externo bastará con llevar este pin a $V_{DD}$ a través de una resistencia.

## POR Power-on reset

Este reset se genera al encender el microcontrolador. Este reset se genera cuando la tensión de alimentación supera cierto umbral situado entre los límites de 1.2V y 1.7V Algunos microcontroladores no disponen de este sistema de reset, por lo que necesitan de hardware externo que lo provoque. Un ejemplo de este caso es el famoso i8051 que típicamente suele ir acompañado del MAX691 que controla el reset.

## BOR Brown-Out Reset

Este reset se provoca cuando la tensión de alimentación cae de un cierto umbral $V_{BOR}$ (aprox. 4V) durante un tiempo $T_{BOR}$ (aprox. 100us). Una típica fuente de estos resets es el arranque de un motor. El pico de corriente que genera un motor al arrancar puede hacer que caiga la tensión de una mala fuente (p.e. las pilas). Una buena idea es no arrancar varios motores al mismo tiempo.

### 11.1.1 Bits de Configuración

Existe entre la dirección 2000h y 2007h de memoria de programa un conjunto de bits de configuración del PIC. Esta dirección está por encima de la memoria de programa de usuario (000h-1FFFh), por lo que no se puede acceder más que en el momento de la programación del PIC.

Las 7 primeras palabras (0x2000 a 0x2006) se denominan ID Location, y sirven para almacenar un nº de serie, o un nº de versión de software instalada, etc.

La siguiente palabra (0x2007) , es la palabra de configuración, sus bits tiene los siguiente significados:

| - | - | CP1 | CP0 | DEBUG | - | WRT | CPD | LVP | BODEN | CP1 | CP0 | /PWRTE | WDTE | FOSC1 | FOSC0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- Bits 13-12 y 5-4: CP1:CP0. Están relacionados con varios niveles de protección del software:
  - 11⇒ no protegido.
  - 10⇒ protegido de 1F00h a 1FFFh.
  - 01⇒ protegido de 1000h a 1FFFh.
  - 00⇒ protegido de 0000h a 1FFFh.
- Bit 11: DEBUG. Si vale 0 se activa el modo de depuración en circuito, entonces RB6 y RB7 quedan reservadas para el debugger.
- Bit 9: WRT. Permite (1), o no (0), escribir en la memoria Flash de programa
- Bit 8. CPD Protección de los datos de la EEPROM. (1=no protegido, 0=protegido)
- Bit 7: LVP. Permite la grabación a baja tensión:
  - 1⇒programación a baja tensión habilitada a través de RB3/PGM.
  - 0⇒RB3 tiene función E/S digital y la programación es por MCLR con tensión alta.
- Bit 6: Si es 1, habilita el control del Brown-out.
- Bit 3. PWRT . Permite activar (0) o desactivar (1) el Power-up Timer.
- Bit 2: WDTE. Permite activar (1) o desactivar (0) el Watch Dog.
- Bit 1- 0: FOSC1:FOSC0. Selección de tipo de oscilador usado:
  - LP (Low Power Cristal) ⇒00 (el típico cristal de cuarzo, frec. entre 32 Khz y 200Khz)
  - XT (Cristal/Resonator) ⇒01 (el típico cristal de cuarzo de 200Khz a 4Mhz)
  - HS (High Speed Cristal/Resonator)⇒10 (el típico crystal de cuarzo de 4Mhz o más)
  - RC (Resistor-Capacitor)⇒11 (oscilador basado en un circuito RC, bajo coste, pero poco preciso en frecuencia)

| Modo | Frecuencia | Valor de capacidad (pF) |
|---|---|---|
| LP | 32 khz | 33 |
| | 200 khz | 15 |
| XT | 200 khz | 47-68 |
| | 1 Mhz | 15 |
| | 4 Mhz | 15 |
| HS | 4 Mhz | 15 |
| | 8 Mhz | 15-33 |
| | 20 Mhz | 15-33 |

**Figura 82. Valor de las capacidades del oscilador principal del PIC.**

### 11.1.2 Mapa de direcciones de datos del PIC 16F87

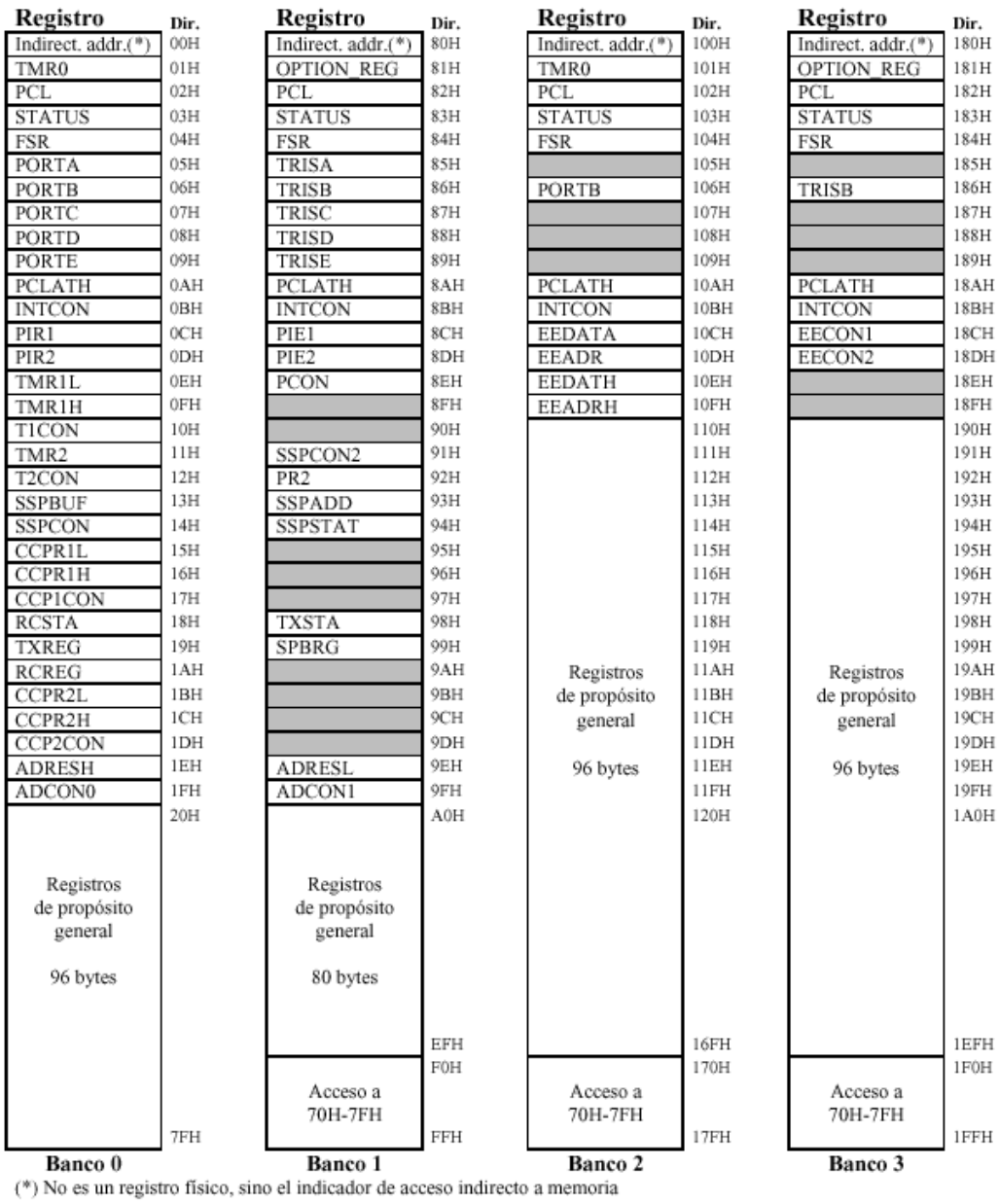



**Figura 83. Mapa de datos del PIC 16F87.**

### 11.1.3 Interrupciones en el PIC16F87

Los módulos del PIC tienen capacidad de generar hasta 15 tipos diferentes de interrupciones, todas ellas confluyen a la misma ISR cuya dirección de inicio es la 04h. Es por ello que una vez que se produce la interrupción, una de las primeras funciones que debe hacer la ISR es determinar qué dispositivo a provocado la interrupción.

Para la gestión de las interrupciones, el PIC cuenta con 5 registros:

- INTCON (0Bh, ó 8Bh, ó 10Bh, ó 18Bh): que controla

- la habilitación global de todas las interrupciones (GIE, bit 7, y PEIE, bit 6),
- la habilitación específica del timer0 (TMR0IE, bit 5), del pin RB0 (INTE, bit 4) y de los cambios de las entradas RB7:RB4 (RBIE, bit 3).
- los flags de interrupción activa del timer0 (T0IF, bit 2), RB0 (INTF, bit 1) y de los cambios de las entradas RB7:RB4 (RBIF, bit 0).

| GIE | PEIE | T0IE | INTE | RBIE | xxxIE | Significado |
|---|---|---|---|---|---|---|
| 0 | X | X | X | X | X | Todas las interrupciones deshabilitadas |
| 1 | 0 | X | X | X | X | Deshabilitadas las interrupciones de los periféricos internos salvo Timer 0 |
| 1 | 1 | X | X | X | X | Permitidas las interrupciones de los periféricos internos. Hay un bit adicional para cada periférico. |
| 1 | X | 0 | X | X | X | Deshabilitada Int. Timer 0 |
| 1 | X | 1 | X | X | X | Habilitada Int. Timer 0 |
| 1 | X | X | 0 | X | X | Deshabilitada Int. externa |
| 1 | X | X | 1 | X | X | Habilitada Int. externa |
| 1 | X | X | X | 0 | X | Deshabilitada Int. cambio líneas RB4,..,RB7 |
| 1 | X | X | X | 1 | X | Habilitada Int. cambio líneas RB4,..,RB7 |

**Figura 84. Habilitación de las interrupciones el registro INTCOM (0Bh, ó 8Bh, ó 10Bh, ó 18Bh).**

- PIE1 (8Ch) y PIE2(8Dh): habilitación de interrupciones del resto de módulos.
- PIR1 (0Ch) y PIR2 (0Dh): estado de la interrupción de los periféricos gestionada desde PIE1 y PIE2.

| Bit | Reg. | bit | Activar interrupción si | Flag | Reg. | bit |
|---|---|---|---|---|---|---|
| TMR1IE | PIE1 | 0 | Overflow en el Timer 1 | TMR1IF | PIR1 | 0 |
| TMR2IE | PIE1 | 1 | Overflow en el Timer 2 | TMR2IF | PIR1 | 1 |
| CCP1IE | PIE1 | 2 | Captura o Comparación en CCP1 | CCP1IF | PIR1 | 2 |
| SSPIE | PIE1 | 3 | Byte recibido o transmitido por el puerto serie síncrono | SSPIF | PIR1 | 3 |
| TXIE | PIE1 | 4 | Byte transmitido por la USART | TXIF | PIR1 | 4 |
| RCIE | PIE1 | 5 | Byte recibido por la USART | RCIF | PIR1 | 5 |
| ADIE | PIE1 | 6 | Finalizada conversión por el conversor A/D | ADIF | PIR1 | 6 |
| PSPIE | PIE1 | 7 | Byte recibido o transmitido por el puerto paralelo esclavo | PSPIF | PIR1 | 7 |
| CCP2IE | PIE2 | 0 | Captura o Comparación en CCP2 | CCP2IF | PIR2 | 0 |
| BCLIE | PIE2 | 3 | Colisión en el bus $I^2C$ | BCLIF | PIR2 | 3 |
| EEIE | PIE2 | 4 | Fin de operación de escritura en la EEPROM | EEIF | PIR2 | 4 |

**Figura 85. Habilitación de las interrupciones de los distintos módulos del PIC (registros PIE1, 8Ch,y PIE2, 8Dh) y detección del dispositivo que ha generado la interrupción (registros PIR1, 0Ch, y PIR2, 0Dh).**

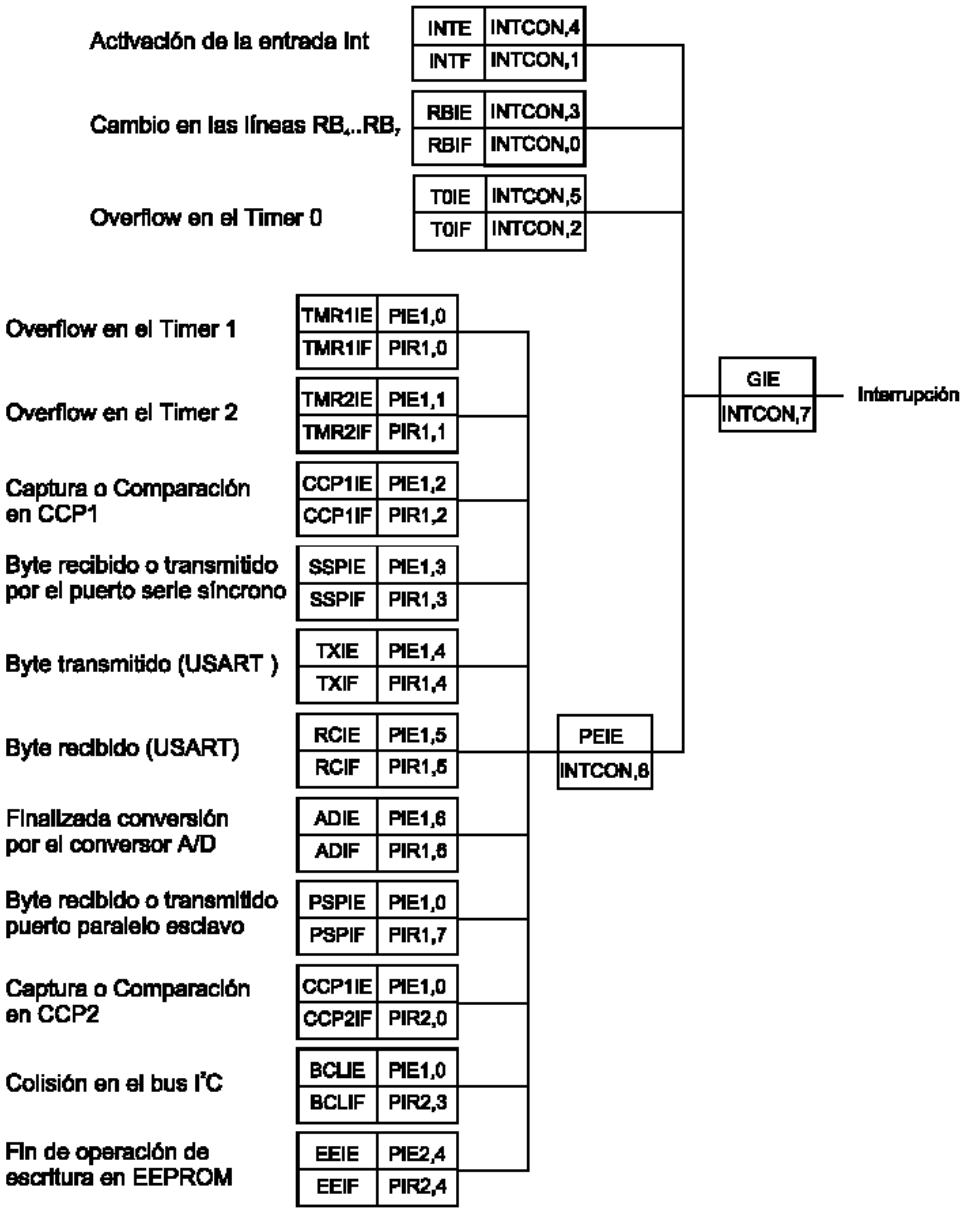



**Figura 86. Resumen de la arquitectura del sistema de interrupciones del PIC 16F87.**

## *11.2 RESUMEN DE INSTRUCCIONES DE PROGRAMACIÓN EN C.*

Para poder trabajar con un lenguaje de alto nivel, como C usado en el presente proyecto, se requiere un compilador que "traduzca" los algoritmos a un lenguaje que el PIC o el simulador pueda interpretar. Esta es la función del programa PIC C Compiler de CCS (Custom Computer Services INC.), el cual puede ser llamado desde MPLAB para compilar los programas directamente desde éste.

El resumen de las funciones más importantes son:

- Preprocesador:
  - #include <16f84.h> #include <16f877.h> : define las constantes específicas del dispositivo.

  - #fuses HS,NOWDT,NOPROTECT: Actualiza la palabra (2 bytes) de configuración como:
    - HS ⇒ oscilador del tipo HS
    - NOWDT⇒ sin Watchdog
    - NOPROTEC⇒no se proteja el código contra lectura una vez grabado en el PIC.
  - #use delay(clock=4000000) : fija el valor del oscilador (en nuestro caso son 4MHz)
  - #byte puertaa (o b) = 0x05 (ó 0x06) : coloca una etiqueta para el byte que está en la dirección correspondiente.
  - #use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, PARITY=N, BITS=8,BRGH1OK): Programa el PIC para poder usar el puerto serie:
    - velocidad⇒ 9600 baudios (velocidad estándard)
    - se usa el pin RC6 como línea de transmisión (no hay que olvidar configurar con set_tris_C la línea 6 como salida)
    - se usa el pin RC7 como línea de recepción (no hay que olvidar configurar con set_tris_C la línea 7 como entrada)
    - PARITY⇒ None, Even, Odd
    - BITS⇒ número de bits, valores posibles: 5-9
    - BRGH1OK⇒cuando haya problemas, puede dar la solución
  - #asm....#endasm: el código que esté entre las dos etiquetas debe escribirse en ensamblador
  - #int_xxx: etiqueta que la función siguiente es una ISR que debe atender a la interrupción procedente del dispositivo XXX. Ejemplos de los flags que podemos encontrar son:
    - INT_AD (conversión A/D completa), INT_ADOF (A/D time out), INT_CPP1, INT_CPP2, INT_EXT (int. de RB0), INT_RB (cambio de RB7:RB4), INT_RDA (recibido dato por RS232), INT_RTCC, INT_TIMERx (x=0,1,2), INT_DEFAULT, INT_GLOBAL
- Control del tiempo:
  - delay_us(x), delay_ms(x): Hace que el programa se pare el tiempo deseado en microsegundos o en milisegundos.
  - delay_cycles(n_ciclos): espera n_ciclos de cpu
- Funciones matemáticas: abs(x), fabs(x), labs(x), acos(x), asin(x), atan(x), atan2(x,y), ceil(x), cos(x), cosh(x), sin(x), sinh(x), tan (x), tanh(x), sqrt(x), exp(x) [potencia de e], pow(base, exponente), floor(x), fmod (x,y) [resto del cociente x/y], log(x) [log neperiano], log10(x)...
- Funciones con strings: strcat (), strcmp(),strcpy(), strlen()...
- Conversiones de tipos:
  - atof, atoi, atol, atoi32
- Entradas/salidas digitales:
  - set_tris_x (0bXXXXXXXX): Con esta función se elige el modo deseado para cada línea (E/S). La x es para definir la puerta a ó b y las X son o 1 ó 0; los 1 son entradas, y los 0 salidas. La X de la derecha es para el bit menos significativo.
  - input(PIN_XZ): Nos devuelve el valor del pin correspondiente, en ese momento. La X es para el tipo de puerta, y la Z para el bit que se quiere mirar.

  - input_x(): Nos devuelve el estado de un puerto entero (x= a, b, c, d, e)
  - output_high(PIN_XZ): Coloca a 1 el pin correspondiente a X y Z.
  - output_low(PIN_XZ): Adivinad.
  - output_x(valor): (x= a, b, c, d, e) Adivinad también.
- Manipulación de bits:
  - bit_clear (dirección, n_bit): Pone a cero el bit <n_bit> de dirección
  - bit_set (dirección, n_bit): Pone a uno el bit <n_bit> de dirección
  - bit_test(dirección, n_bit): Devuelve 0 ó 1, según sea el valor del bit <n_bit> de dirección
- Control de las interrupciones:
  - enable_interrupts(nivel): habilita las interrupciones del nivel señalado.
  - disable_interrupts(nivel): deshabilita las interrupciones del nivel señalado.
- Temporizadores:
  - get_timerx(): devuelve el valor de contaje del timer x (x=0,1,2).
  - set_timerx(valor): establece el valor máximo de cuenta (x=0,1,2).
  - setup_counters (estado del contador, estado del preescaler): configura el timer0 y el Watchdog.
  - setup_timer_x(modo): configura los temporizadores (x=0,1)
  - setup_timer_2(modo, periodo, postescaler): ídem pero con el timer 2
- Funciones del módulo conversor A/D
  - setup_adc_ports (valor). Configura los pines que se usarán cono entradas analógicas y como entradas de referencia.
  - setup_adc (modo): Configura el reloj de muestreo del conversor A/D .
  - set_adc_channel (canal): Selecciona el canal usado en la próxima llamada a la función read_adc ().
  - read_adc (). Devuelve el valor de salida del conversor de A/D.
- Funciones del módulo de comunicaciones RS232:
  - printf(string): envía un string por el puerto serie.
  - kbhit(): comprueba si hay un carácter en el buffer de recepción.
  - getc(): espera por un carácter y cuando llega, lo devuelve como valor de retorno.
  - gets(string): lee caracteres del puerto serie hasta que le llega un <return> (ASCII 13).
  - putc(carácter): envía un carácter por el RS232.
  - puts(string): envía un string por el RS232.
  - set_uart_speed(baudios): configura la velocidad del puerto RS232.
- Funciones de control de las salidas PWM
  - set_pwm1_duty(valor)/set_pwm2_duty(valor): establece el duty cycle para la salida CCP1 y CCP2 respectivamente.
  - setup_ccp1(modo)/ setup_ccp2(modo): configura el modo de funcionamiento de los módulos CCP1 y CCP2:
    - CCP_OFF,
    - modo de captura: CCP_CAPTURE_FE, CCP_ CAPTURE_RE, CCP_CAPTURE_DIV_4, CCP_CAPTURE_DIV_16
    - modo de comparación: CCP_COMPARE_SET_ON_MATCH, CCP_COMPARE_CLR_ON_MATCH, CCP_COMPARE_INT, CCP_COMPARE_RESET_TIMER
    - modo PWM: CCP_PWM

### 11.2.1 Ejemplo de entrada analógica, entrada/salida digital y puerto serie

```
#include <16f876.h>   /* inclusión de la cabecera con definiciones
del PIC */
#include <stdlib.h>     /* inclusión de la cabecera para funciones
I/O

#fuses HS,NOWDT,NOPROTECT /* palabra de configuración
                             HS->oscilador de 4Mhz ó más
                             NOWDT-> sin perro guardián
                             NOPROTECT->sin protección de código */
#use delay(clock=4000000)   /*oscilador de 4Mhz */
/*puerto serie a 9600 baudios, transmitir por pin RC6, recibir por
pin RC7 */
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BRGH1OK)


  float adc_ch0;
  float value_ch0;


void main() {
      setup_adc_ports(RA0_ANALOG);/* config, RA0 como entrada
analógica */
      setup_adc(ADC_CLOCK_DIV_32);  /*config. del reloj de muestreo
del A/D*/
      set_adc_channel(0);                  /* Selección del
canal analógico 0 */

      set_tris_a(0b00000001);  /* Puerto RA0 entrada, resto salidas
*/
      set_tris_c(0b10000000);        /* Puerto RC7 entrada, resto
salidas */
      set_tris_b(0b00000000); /* Puerto RB de salida */

      output_high(PIN_B7);    /* RB7 puesto en alto */


      do {

      set_adc_channel(0);   /*selección del canal 0 del A/D*/

      delay_us(10);    /*esperar 10 microseg para la conversión A/D */
      adc_ch0 = (float)Read_ADC(); /*lectura del canal 0*/
      value_ch0 = (float)5*adc_ch0/255; /*paso de cuentas del
conversor a voltios*/

printf("\n\r %f      :  %2x",value_ch0,value_ch0); /*envío por el
puerto serie la
medida */

     } while(1);
}
```

### 11.2.2 Ejemplo de PWM, conversor A/D, y puerto serie

```
#include <16F877.h> /*selección del PIC 16F877*/
#fuses HS,NOWDT,NOPROTECT,NOLVP /* configuración:
oscilador de más de 4Mhz, no perro guardián, no proteger el código,
programación a alta tensión */
```

```
#use delay(clock=20000000)  /*oscilador a 20 Mhz*/

/*puerto serie a 9600 baudios, transmitir por pin RC6, recibir por
pin RC7 */
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BRGH1OK)

void main() {
   char seleccion;
   int16 valor;
  int8 period=127;

   printf("\r\nFrecuencia:\r\n");
   printf("    1) 39.1 khz\r\n");
   printf("    2) 9.8 khz\r\n");
   printf("    3) 2.4 khz\r\n");

   do {
     seleccion=getc();
   } while((seleccion<'1')||(seleccion>'3'));


   setup_ccp1(CCP_PWM);   // configurar el módulo CCP1 como PWM

         /*   La frecuencia del PWM se calcula como
(1/clock)*4*t2div*(period+1)
            En este ejemplo clock=20000000 y period=127
            Entonces tenemos:
     (1/20000000)*4*1*128 =  25.6 us o 39.1 khz
                 (1/20000000)*4*4*128 = 102.4 us o 9.8 khz
                 (1/20000000)*4*16*128= 409.6 us o 2.4 khz  */

   switch(seleccion) {
     case '1' : setup_timer_2(T2_DIV_BY_1, period, 1);
                break;
     case '2' : setup_timer_2(T2_DIV_BY_4, period, 1);
                break;
     case '3' : setup_timer_2(T2_DIV_BY_16, period, 1);
                break;
   }

  printf("%c\r\n",seleccion);

  setup_adc_ports(RA0_ANALOG);
  setup_adc(adc_clock_internal); /*en este caso se elige como de
reloj de muestreo el
interno del PIC */
  set_adc_channel( 0 );  /*se selecciona el canal 0*/

  while(1) {
    set_pwm1_duty(valor);
              /* se establece el duty-cycle como el valor recogido
en la entrada analógica
       se puede calcular el duty-cycle (en segundos)
como:valor*(1/clock)*t2div*/
  }

}
```

### 11.2.3 Código de ejemplo de testeo de la tarjeta genérica basada en el PIC 18F4550 (Figura 5)

```
// CODIGO CONTROL DE MOTORES MICROROBOT

//DEFINICION DE VARIABLES E INICIALIZACION DE PARAMETROS DEL SISTEMA

#include <18F4550.h>
#device adc=10
//#fuses EC_IO
#DEVICE HIGH_INTS=TRUE
#use delay (clock = 48000000)
#use rs232 (baud=9600,xmit=PIN_C6, rcv=PIN_C7)
//#byte PORTB = 0xf81
//#byte PORTA = 0xf80
//#byte PORTD = 0xf83
int16 w=0;
int16 w1=0;
long duty=0;
long duty2=0;
int vec[5];
int vec2[5];
int scan;
int tmp;
int tipo;
int a=0;
int b=0;
int c=0;
int d=0;


int i=0;
long j=0;
long valor[4];
float leer[6];
int8 cruce=0;

void main ()
 {
SET_TRIS_A( 0x3F );
duty=240;
duty2=384;
setup_timer_2(T2_DIV_BY_4,192,2);
setup_ccp1(CCP_PWM);
setup_ccp2(CCP_PWM);
set_pwm1_duty (duty);
set_pwm2_duty (duty2);

////////////////////////////////

while(1){
output_high(PIN_D0);
output_high(PIN_D1);
output_high(PIN_D2);
output_high(PIN_D3);
output_high(PIN_D4);
output_high(PIN_D5);
output_low(PIN_D6);
output_high(PIN_D7);

output_high(PIN_B0);
```

```
output_high(PIN_B1);
output_high(PIN_B2);
output_high(PIN_B3);
output_high(PIN_B4);

output_low(PIN_D4);
output_HIGH(PIN_D6);

output_high(PIN_C0);
output_high(PIN_C4);
output_high(PIN_C5);
output_high(PIN_C6);
output_high(PIN_C7);

}

}
```

### 11.2.4 Código de básico ejemplo de testeo de la tarjeta genérica basada en el PIC 18F4550 (Figura 5)

```
//DEFINICION DE VARIABLES E INICIALIZACION DE PARAMETROS DEL SISTEMA

#include <18F4550.h>
#fuses EC_IO,CPUDIV1              // Definicion de los parámetros de
reloj y sistema.

// #DEVICE HIGH_INTS=TRUE                          // Manejo de
prioridades de las interrupciones.

#use delay (clock = 20000000)                      //Indicación de la
frecuencia del reloj.
// #use rs232 (baud=9600,xmit=PIN_C6, rcv=PIN_C7)  //Descomentar si
se utiliza puerto serie.

//#byte PORTB = 0xf81                              // Definición de
los puertos A y B.
//#byte PORTA = 0xf80                              // No es
obligatorio



// DATOS DE PINES DE SALIDA
//LED 0 = RDO  //LED 1 = RD1  //LED 2 = RD2


void main ()
 {


SET_TRIS_B( 0x00 );
SET_TRIS_A( 0x3F );


output_low(PIN_D0);                                // Inicio los LEDS
en apagado.
output_low(PIN_D1);
output_low(PIN_D2);
```

```
while(1){                                          // Código para
rotar los LEDS cada 200 ms
output_high(PIN_D0);
output_low(PIN_D2);
delay_ms(200);
output_low(PIN_D0);
output_high(PIN_D1);
delay_ms(200);
output_low(PIN_D1);
output_high(PIN_D2);
delay_ms(200);

}



}
```

## *11.3 JUEGO DE INSTRUCCIONES EN ENSAMBLADOR*

El PIC 16F87 tiene un juego de instrucciones de 14 bits, divididas en OPCODE (que especifica la instrucción) y uno o más operandos específicos de cada instrucción. Las instrucciones se clasifican en 5 categorías: orientadas a bytes, orientadas a bits, operaciones de control y con constantes (literales), instrucciones generales y llamadas call y goto.

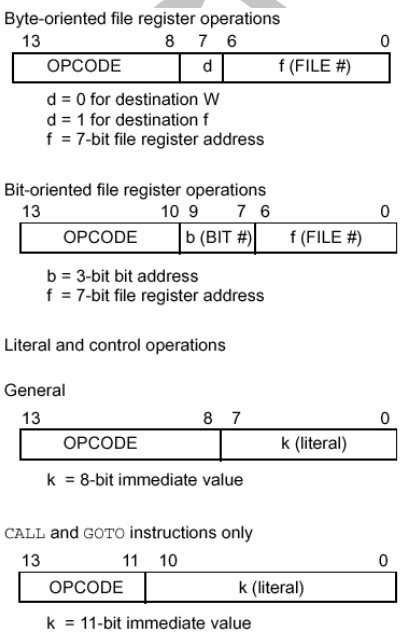



**Figura 87. Tipos de instrucciones del PIC 16F87.**

Para las instrucciones orientadas al byte: ‘f’ representa un registro y ‘d’ indica:

- si d=0, el resultado se guarde en el registro W, y
- si d=1, el resultado se guarda en ‘f’.

Para las instrucciones orientadas al bit, ‘b’ es el puntero que selecciona, a través de su valor numérico, el bit afectado por la instrucción. Y ‘f’ representa la dirección del registro que contiene el bit que será modificado.

Para las instrucciones sobre constantes y control, ‘k’ representa una constante de 8 o 16 bits.

Todas las instrucciones se ejecutan en un ciclo de reloj, excepto cuando hay que ejecutar una instrucción de salto y con ello se debe modificar el contador de programa. Un ciclo de reloj son cuatro períodos del oscilador.

| Mnemonic, Operands | | Description | Cycles | 14-Bit Opcode MSb LSb | Status Affected | Notes |
|---|---|---|---|---|---|---|
| **BYTE-ORIENTED FILE REGISTER OPERATIONS** | | | | | | |
| ADDWF | f, d | Add W and f | 1 | 00 0111 dfff ffff | C,DC,Z | 1,2 |
| ANDWF | f, d | AND W with f | 1 | 00 0101 dfff ffff | Z | 1,2 |
| CLRF | f | Clear f | 1 | 00 0001 lfff ffff | Z | 2 |
| CLRW | - | Clear W | 1 | 00 0001 0xxx xxxx | Z | |
| COMF | f, d | Complement f | 1 | 00 1001 dfff ffff | Z | 1,2 |
| DECF | f, d | Decrement f | 1 | 00 0011 dfff ffff | Z | 1,2 |
| DECFSZ | f, d | Decrement f, Skip if 0 | 1(2) | 00 1011 dfff ffff | | 1,2,3 |
| INCF | f, d | Increment f | 1 | 00 1010 dfff ffff | Z | 1,2 |
| INCFSZ | f, d | Increment f, Skip if 0 | 1(2) | 00 1111 dfff ffff | | 1,2,3 |
| IORWF | f, d | Inclusive OR W with f | 1 | 00 0100 dfff ffff | Z | 1,2 |
| MOVF | f, d | Move f | 1 | 00 1000 dfff ffff | Z | 1,2 |
| MOVWF | f | Move W to f | 1 | 00 0000 lfff ffff | | |
| NOP | - | No Operation | 1 | 00 0000 0xx0 0000 | | |
| RLF | f, d | Rotate Left f through Carry | 1 | 00 1101 dfff ffff | C | 1,2 |
| RRF | f, d | Rotate Right f through Carry | 1 | 00 1100 dfff ffff | C | 1,2 |
| SUBWF | f, d | Subtract W from f | 1 | 00 0010 dfff ffff | C,DC,Z | 1,2 |
| SWAPF | f, d | Swap nibbles in f | 1 | 00 1110 dfff ffff | | 1,2 |
| XORWF | f, d | Exclusive OR W with f | 1 | 00 0110 dfff ffff | Z | 1,2 |
| **BIT-ORIENTED FILE REGISTER OPERATIONS** | | | | | | |
| BCF | f, b | Bit Clear f | 1 | 01 00bb bfff ffff | | 1,2 |
| BSF | f, b | Bit Set f | 1 | 01 01bb bfff ffff | | 1,2 |
| BTFSC | f, b | Bit Test f, Skip if Clear | 1 (2) | 01 10bb bfff ffff | | 3 |
| BTFSS | f, b | Bit Test f, Skip if Set | 1 (2) | 01 11bb bfff ffff | | 3 |
| **LITERAL AND CONTROL OPERATIONS** | | | | | | |
| ADDLW | k | Add literal and W | 1 | 11 111x kkkk kkkk | C,DC,Z | |
| ANDLW | k | AND literal with W | 1 | 11 1001 kkkk kkkk | Z | |
| CALL | k | Call subroutine | 2 | 10 0kkk kkkk kkkk | | |
| CLRWDT | - | Clear Watchdog Timer | 1 | 00 0000 0110 0100 | $\overline{TO}$,$\overline{PD}$ | |
| GOTO | k | Go to address | 2 | 10 1kkk kkkk kkkk | | |
| IORLW | k | Inclusive OR literal with W | 1 | 11 1000 kkkk kkkk | Z | |
| MOVLW | k | Move literal to W | 1 | 11 00xx kkkk kkkk | | |
| RETFIE | - | Return from interrupt | 2 | 00 0000 0000 1001 | | |
| RETLW | k | Return with literal in W | 2 | 11 01xx kkkk kkkk | | |
| RETURN | - | Return from Subroutine | 2 | 00 0000 0000 1000 | | |
| SLEEP | - | Go into standby mode | 1 | 00 0000 0110 0011 | $\overline{TO}$,$\overline{PD}$ | |
| SUBLW | k | Subtract W from literal | 1 | 11 110x kkkk kkkk | C,DC,Z | |
| XORLW | k | Exclusive OR literal with W | 1 | 11 1010 kkkk kkkk | Z | |

**Note 1:** When an I/O register is modified as a function of itself ( e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

**2:** If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

**3:** If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

| | |
|---|---|
| PC | contador de programa (13bits) |
| TOS | cima de pila |
| WDT | watchdog |
| W | registro de trabajo del PIC |

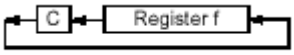
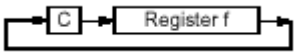
| | |
|---|---|
| TO | bit ‘Time Out’ del registro STATUS |
| PD | bit ‘Power Down’ del registro STATUS |

**Figura 88. Tabla resumen del juego de instrucciones del PIC 16F87.**

### 11.3.1 Descripción del juego de instrucciones

| | |
|---|---|
| **ADDLW Add Literal and W**<br>Syntax: *[label*] ADDLW k<br>Operands: 0<=k<=255<br>Operation: (W) + k →(W)<br>Status Affected: C, DC, Z<br>Description: The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register. | **ADDWF Add W and f**<br>Syntax: *[label*] ADDWF f,d<br>Operands: 0<=f<=127<br>d ∈[0,1]<br>Operation: (W) + (f) → (destination)<br>Status Affected: C, DC, Z<br>Description: Add the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'. |
| **ANDLW AND Literal with W**<br>Syntax: *[label*] ANDLW k<br>Operands: 0<=k<=255<br>Operation: (W) <AND> (k) → (W)<br>Status Affected: Z<br>Description: The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register. | **ANDWF AND W with f**<br>Syntax: *[label*] ANDWF f,d<br>Operands: 0 <=f<=127<br>d ∈[0,1 ]<br>Operation: (W) <AND> (f) →.(destination)<br>Status Affected: Z<br>Description: AND the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'. |
| **BCF Bit Clear f**<br>Syntax: *[label*] BCF f,b<br>Operands: 0<=f<=127<br>0<=b<=7<br>Operation: 0→ (f<b>)<br>Status Affected: None<br>Description: Bit 'b' in register 'f' is cleared. | **BSF Bit Set f**<br>Syntax: *[label*] BSF f,b<br>Operands: 0 <=f<=127<br>0 <=b<=7<br>Operation: 1 → (f<b>)<br>Status Affected: None<br>Description: Bit 'b' in register 'f' is set. |
| **BTFSS Bit Test f, Skip if Set**<br>Syntax: *[label*] BTFSS f,b<br>Operands: 0 <=f <=127<br>0<=b<=7<br>Operation: skip if (f<b>) = 1<br>Status Affected: None<br>Description: If bit 'b' in register 'f' is '0', the next instruction is executed. If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a 2TCY instruction. | **BTFSC Bit Test, Skip if Clear**<br>Syntax: *[label*] BTFSC f,b<br>Operands: 0 <=f<=127<br>0 <=b<=7<br>Operation: skip if (f<b>) = 0<br>Status Affected: None<br>Description: If bit 'b' in register 'f' is '1', the next instruction is executed. If bit 'b', in register 'f', is '0', the next instruction is discarded, and a NOP is executed instead, making this a 2TCY instruction. |
| **CALL Call Subroutine**<br>Syntax: [ *label* ] CALL k<br>Operands: 0 <=k< =2047<br>Operation: (PC)+ 1→TOS,<br>k→.PC<10:0>,<br>(PCLATH<4:3>)→PC<12:11><br>Status Affected: None<br>Description: Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two-cycle instruction. | **CLRF Clear f**<br>Syntax: *[label*] CLRF f<br>Operands: 0 <=f< =127<br>Operation: 00h → (f)<br>1→Z<br>Status Affected: Z<br>Description: The contents of register 'f' are cleared and the Z bit is set. |
| **CLRW Clear W**<br>Syntax: [ *label* ] CLRW<br>Operands: None<br>Operation: 00h → (W)<br>1→Z<br>Status Affected: Z<br>Description: W register is cleared. Zero bit (Z) is set. | **CLRWDT Clear Watchdog Timer**<br>Syntax: [ *label* ] CLRWDT<br>Operands: None<br>Operation: 00h→WDT<br>0→WDT prescaler,<br>1→TO<br>1→PD<br>Status Affected: TO, PD<br>Description: CLRWDT instruction resets the |

| | |
|---|---|
| | Watchdog Timer. It also resets the prescaler of the WDT. Status bits TO and PD are set. |
| **COMF Complement f**<br>Syntax: [ *label* ] COMF f,d<br>Operands: 0 <=f<=127<br>d ∈ [0,1]<br>Operation: (f) .(destination)<br>Status Affected: Z<br>Description: The contents of register 'f' are complemented. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f'. | **DECF Decrement f**<br>Syntax: *[label*] DECF f,d<br>Operands: 0 <=f<=127<br>d∈ [0,1]<br>Operation: (f) - 1 → (destination)<br>Status Affected: Z<br>Description: Decrement register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'. |
| **DECFSZ Decrement f, Skip if 0**<br>Syntax: [ *label* ] DECFSZ f,d<br>Operands: 0 <=f <=127<br>d∈ [0,1]<br>Operation: (f) - 1→ (destination);<br>skip if result = 0<br>Status Affected: None<br>Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, then a `NOP` is executed instead making it a 2TCY instruction. | **GOTO Unconditional Branch**<br>Syntax: [ *label* ] GOTO k<br>Operands: 0 <=k< =2047<br>Operation: k →PC<10:0><br>PCLATH<4:3>→PC<12:11><br>Status Affected: None<br>Description: `GOTO` is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. `GOTO` is a two-cycle instruction. |
| **INCF Increment f**<br>Syntax: [ *label* ] INCF f,d<br>Operands: 0 <=f<=127<br>d ∈ [0,1]<br>Operation: (f) + 1→ (destination)<br>Status Affected: Z<br>Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. | **INCFSZ Increment f, Skip if 0**<br>Syntax: [ *label* ] INCFSZ f,d<br>Operands: 0 <=f<=127<br>d ∈ [0,1]<br>Operation: (f) + 1→ (destination),<br>skip if result = 0<br>Status Affected: None<br>Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, a `NOP` is executed instead, making it a 2TCY instruction. |
| **IORLW Inclusive OR Literal with W**<br>Syntax: [ *label* ] IORLW k<br>Operands: 0 <=k<=255<br>Operation: (W) <OR> k→(W)<br>Status Affected: Z<br>Description: The contents of the W register are OR'ed with the eight bit literal 'k'. The result is placed in the W register. | **IORWF Inclusive OR W with f**<br>Syntax: [ *label* ] IORWF f,d<br>Operands: 0<=f<=127<br>d∈ [0,1]<br>Operation: (W) <OR> (f) →.(destination)<br>Status Affected: Z<br>Description: Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'. |
| **MOVF Move f**<br>Syntax: [ *label* ] MOVF f,d<br>Operands: 0 <=f <=127<br>d ∈ [0,1]<br>Operation: (f) →(destination)<br>Status Affected: Z<br>Description: The contents of register f are moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register, since status flag Z is affected. | **MOVLW Move Literal to W**<br>Syntax: [ *label* ] MOVLW k<br>Operands: 0 <=k <=255<br>Operation: k → (W)<br>Status Affected: None<br>Description: The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's. |

| | |
|---|---|
| **MOVWF Move W to f**<br>Syntax: [ *label* ] MOVWF f<br>Operands: 0 <=f <=127<br>Operation: (W) → (f)<br>Status Affected: None<br>Description: Move data from W register to register 'f'. | **NOP No Operation**<br>Syntax: [ *label* ] NOP<br>Operands: None<br>Operation: No operation<br>Status Affected: None<br>Description: No operation. |
| **RETFIE Return from Interrupt**<br>Syntax: [ *label* ] RETFIE<br>Operands: None<br>Operation: TOS →PC,<br>1 →GIE<br>Status Affected: None | **RETLW Return with Literal in W**<br>Syntax: [ *label* ] RETLW k<br>Operands: 0 <=k <=255<br>Operation: k → (W);<br>TOS →PC<br>Status Affected: None<br>Description: The W register is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two-cycle instruction. |
| **RLF Rotate Left f through Carry**<br>Syntax: [ *label* ] RLF f,d<br>Operands: 0 <=f <=127<br>d ∈ [0,1]<br>Operation: See description below<br>Status Affected: C<br>Description: The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is stored back in register 'f'.<br>  | **RETURN Return from Subroutine**<br>Syntax: [ *label* ] RETURN<br>Operands: None<br>Operation: TOS → PC<br>Status Affected: None<br>Description: Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two-cycle instruction. |
| **RRF Rotate Right f through Carry**<br>Syntax: [ *label* ] RRF f,d<br>Operands: 0 <=f <=127<br>d ∈ [0,1]<br>Operation: See description below<br>Status Affected: C<br>Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.<br>  | **SLEEP**<br>Syntax: [ *label* ] SLEEP<br>Operands: None<br>Operation: 00h →WDT,<br>0 → WDT prescaler,<br>1 →/TO,<br>0 →/PD<br>Status Affected: /TO, /PD<br>Description: The power-down status bit, /PD is cleared. Time-out status bit, /TO is set. Watchdog Timer and its prescaler are cleared. The processor is put into SLEEP mode with the oscillator stopped. |
| **SUBLW Subtract W from Literal**<br>Syntax: [ *label* ] SUBLW k<br>Operands: 0 <=k <=255<br>Operation: k - (W) → (W)<br>Status Affected: C, DC, Z<br>Description: The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register. | **SUBWF Subtract W from f**<br>Syntax: [ *label* ] SUBWF f,d<br>Operands: 0 <=f <=127<br>d ∈ [0,1]<br>Operation: (f) - (W) → (destination)<br>Status Affected: C, DC, Z<br>Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'. |
| **SWAPF Swap Nibbles in f**<br>Syntax: [ *label* ] SWAPF f,d<br>Operands: 0 <=f <=127<br>d ∈ [0,1]<br>Operation: (f<3:0>) → (destination<7:4>),<br>(f<7:4>) → (destination<3:0>)<br>Status Affected: None<br>Description: The upper and lower nibbles of | **XORLW Exclusive OR Literal with W**<br>Syntax: *[label*] XORLW k<br>Operands: 0 <=k <=255<br>Operation: (W) <XOR> k→.(W)<br>Status Affected: Z<br>Description: The contents of the W register are XOR'ed with the eight-bit literal 'k'. The result is placed in the W register. |

| | |
|---|---|
| register ’f’ are exchanged. If ’d’ is 0, the result is placed in the W register. If ’d’ is 1, the result is placed in register ’f’. | |
| **XORWF Exclusive OR W with f**<br>Syntax: *[label*] XORWF f,d<br>Operands: 0 <=f <=127<br>d ∈ [0,1]<br>Operation: (W) <XOR> (f) →.(destination)<br>Status Affected: Z<br>Description: Exclusive OR the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'. | |