Universidad de Valladolid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

Proyecto Fin de Carrera
Ingeniero en Electrónica

Diseño de un módulo receptor para mandos a distancia (infrarrojos): conexión a un LCD comercial

Autor:
Carlos Arévalo Sillero
Tutor:
Jesús M. Hernández Mangas

Valladolid, 23 de junio de 2005

TÍTULO:	**Diseño de un módulo receptor para mandos a distancia (infrarrojos): conexión a un LCD comercial**
AUTOR:	**Carlos Arévalo Sillero**
TUTOR:	**Jesús M. Hernández Mangas**
DEPARTAMENTO:	**Electricidad y Electrónica**

Miembros del tribunal

PRESIDENTE:	Jesús Arias Álvarez
VOCAL:	Ruth Pinacho Gómez
SECRETARIO:	Jesús M. Hernandez Mangas
FECHA DE LECTURA:	
CALIFICACIÓN:	

Resumen del proyecto

El proyecto está basado en la capacidad de recepción de cualquier señal de infrarrojos proveniente de un mando a distancia comercial, es decir, en la recepción universal. El módulo receptor puede reconocer el código binario asociado a las pulsaciones alfanuméricas de los mandos a distancia.

Este módulo dispone también de una serie de puertos que permiten la comunicación con otros dispositivos. Los puertos son los siguientes:

- Interfaz serie RS-232.
- Interfaz I2C.
- Interfaz paralela.
- Interfaz LCD 16x2.
- Interfaz para actuadores de potencia: relé, triacs.

La aplicación práctica del módulo IR consiste en la particularización del sistema para la definición del contenido de un display LCD comercial de la empresa TECDIS mediante el uso de menús.

Abstract

The project is based on the capacity of reception of all infrared signals that come from a remote control; is based on the universal reception. The receiving module can recognize the binary code of any alfanumerical key of remote cotrols. This module has also ports that allow communication with other devices. This ports are the followings:

- RS-232 port.
- I2C port.
- Parallel port.
- LCD 16x2 port.
- Power actuator port: relé, triacs.

The practical application consists of development of the system for the definition of the content of commercial LCD display of TECDIS by means of the use of menus.

Palabras clave

Infrarrojos, Universal, LCD, Pic16F628, I2C, Paralelo, Rs-232.

Dedicado a mi novia María, a mis padres y a mis hermanos.

Agradecimientos

Me gustaría empezar dando las gracias a mi tutor, Jesús, que me dió la oportunidad de llevar a cabo este proyecto y que me ha guiado y prestado su ayuda para llevarlo a buen puerto.

Quiero también dar mi agradecimiento a mi novia, María, que tantas tardes a pasado junto a mi dándome todo su apoyo.

Por último, doy las gracias a mis compañeros, entre ellos a Iván, que hizo algunas de las fotografías que se incluyen en esta memoria aún disponiendo de muy poco tiempo.

Prólogo

El mundo de los infrarrojos está tremendamente extendido. En todos nuestros hogares disponemos de múltiples mandos a distancia que utilizan señales de infrarrojos y que nos hacen la vida un poco más cómoda. Como demuestra este proyecto, podemos utilizar estos mandos a distancia para el desarrollo de nuevas aplicaciones que nos faciliten determinadas tareas que, de otra manera, serian sensiblemente más complejas.

Este proyecto es un buen ejemplo de aplicación del empleo de señales infrarrojas para todos aquellos que deseen particularizar el uso de este tipo de vías de comunicación para sus propias aplicaciones.

Espero que mi trabajo sirva de ayuda para todo aquel que desee conocer de forma más profunda el universo de la radiación infrarroja.

Carlos Arévalo Sillero

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Objetivo

Se pretende llevar a cabo el diseño y la construcción de un módulo receptor de infrarrojos basado en un microcontrolador. Este módulo será universal por lo que dispondrá de la capacidad de recibir e interpretar señales IR de cualquier mando a distancia comercial.

Se añadirán otras funcionalidades al módulo. La primera de ellas será una interfaz serie doble de entrada salida. Además se incluirán una interfaz I2C, una interfaz paralela, un puerto de seis patillas para conectar un LCD Hitachi 44780 16x2 y una interfaz para actuadores de potencia.

Finalmente se desarrollará el módulo para que sea capaz de definir el contenido de un display LCD comercial de la empresa TECDIS a través del uso de menús. Para ese efecto el sistema memorizará las señales asociadas a cada pulsación alfanumérica del teclado y una vez acabado el aprendizaje realizará las operaciones oportunas a medida que le lleguen señales al receptor IR que coincidan con las almacenadas.

1.2. Motivación

Fundamentalmente con este módulo receptor de infrarrojos se busca facilitar de forma significativa la programación del LCD comercial sin que para tal operación sea necesario conectarlo a un PC. De esta forma, con cualquier mando a distancia del que se disponga, se definirá el mensaje mostrado por el LCD y el modo de visualización en el que se desea que aparezca.

Con el puerto serie, el puerto paralelo y la interfaz I2C se busca añadir potencialidad al sistema para poder desarrollar nuevas aplicaciones basadas en el control con mando a distancia ya que permiten la comunicación entre el módulo principal encargado de recibir

las señales infrarrojas y otros módulos adicionales con unas funciones específicas.

Por último es importante disponer de una interfaz de actuadores de potencia ya que ésta puede ser programada para que a través de relés y triacs posibilite por ejemplo el control de la apertura de una puerta.

1.3. Receptor de Infrarrojos

1.3.1. Transmisión de información por infrarrojos

La señal emitida por un mando a distancia es realmente compleja, pues en ella se unen varios tipos de codificación y modulación. Veamos detenidamente cómo es esta señal.

El medio físico que transporta la señal del mando a distancia es una onda luminosa en el rango de los infrarrojos. Este tipo de luz no es en absoluto peligrosa para la salud, a diferencia de la luz ultravioleta. La luz infrarroja es invisible al ojo humano, pero no a los dispositivos semiconductores. Fuentes de luz infrarroja son: el sol, cualquier bombilla de incandescencia, diodos LED's, etc. Un LED es lo que utilizan los mandos a distancia para emitir la señal.

Figura 1.1: Espectro luminoso

Todos los dispositivos semiconductores son sensibles a la luz (a todo tipo de luz), de hecho este es el principal factor que determina el encapsulado de los chips: Es un plástico completamente opaco, normalmente negro. Un sensor de infrarrojos está construido básicamente por una unión semiconductora recubierta por un cristal que sólo deja pasar la luz infrarroja.

Un sensor de infrarrojos construido de esa forma será capaz de detectar la presencia de cualquier luz infrarroja, independientemente de la fuente que la genere. Los diseñadores de mandos a distancia tuvieron que añadir una característica diferenciadora a la luz emitida desde un mando a distancia para hacerla distinguible del resto de fuentes luminosas. La señal emitida por un mando está modulada a una frecuencia entre 32 y 40 Khz, dependiendo del fabricante y modelo del mando a distancia. En adelante supondremos que la frecuencia del mando a distancia es de 38 Khz, pués suele ser el valor más común. Más del 90 % de todos los mandos utilizan esta frecuencia o una muy próxima.

No hay que olvidar que todo lo que se haga para emitir la señal IR, luego durante la recepción se tendrá que deshacer. Si esta señal se modula con una portadora de 38Khz luego se tendrá que eliminar (filtrar) esa portadora. La demodulación se realiza analógicamente con un sencillo filtro paso banda (eliminar todas las frecuencias que no sean próximas a la frecuencia de emisión) junto con un rectificador/integrador.

Luz infrarroja modulada a una frecuencia de 38 Khz, es el medio de comunicación entre el emisor (mando a distancia) y el receptor (TV, vídeo, etc.). Ahora queda por determinar cómo se transmite la información, esto es, los bits que identifican la tecla del mando a distancia que se ha pulsado.

La forma de codificar la información depende de cada fabricante. Es evidente, dada la forma de modular la señal, que la información se transmite en serie, es decir, un bit detrás de otro, y por tanto sólo es necesaria una línea (de datos) para recibirla.

1.3.2. Receptor de infrarrojos IS1U60

Comenzamos estudiando la parte de la recepción de la señal infrarroja que es sumamente sencilla de conectar a un microcontrolador.

Para la recepción IR vamos a utilizar un dispositivo que unifica en el mismo encapsulado el receptor de luz infrarroja, una lente y toda la lógica necesaria para distinguir señales moduladas a una determinada frecuencia. Concretamente, en nuestro montaje utilizaremos el receptor IS1U60 de Sharp que se activa cuando recibe una luz infrarroja modulada a una frecuencia de 38 kHz (el haz infrarrojo se apaga y enciende 38000 veces por segundo). Esto lo hace compatible con un gran número de mandos a distancia de electrodomésticos.

En el mercado existen otras alternativas tanto de la misma compañía como de otros fabricantes. Sharp también proporciona otros receptores de infrarrojos como por ejemplo los IS1U621, similares al utilizado pero con más rango de recepción (8 metros frente a los 5 de los IS1U60). De otros fabricantes se destacan los PNA4602 o PNA4612 de Panasonic, los LTM-97DS-38 de LiteOn o los SFH5110 de Siemens.

Aunque el patillaje es diferente entre los distintos fabricantes y deberá consultarse en el datasheet correspondiente antes de realizar cualquier conexión, en todos los casos nos vamos a encontrar tres patas: una que conectaremos a Vcc, otra que lo haremos a GND y una tercera, Vout, por la que obtendremos diferentes niveles si se recibe o no la señal infrarroja (en el caso de los Sharp, un nivel alto si no se recibe la señal infrarroja modulada o un nivel bajo si se está recibiendo). Tal y como se muestra en la figura, con el IS1U60 visto de frente, las patas de izquierda a derecha corresponden con Vout, GND

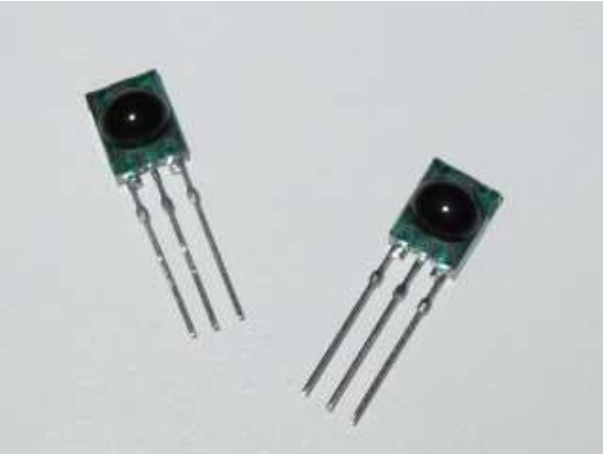


Figura 1.2: Receptor de infrarrojos IS1U60 de SHARP

y Vcc.

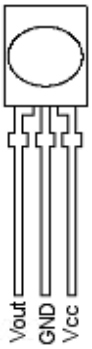


Figura 1.3: IS1U60 visto de frente

Para la conexión, el fabricante recomienda que se utilice un filtro de las conexiones de alimentación mediante una resistencia de 47 ohmios en serie con con Vcc y un condensador de 47 uF entre Vcc y GND tal y como se muestra en la figura 1.4. Si no se incluyen estos elementos los resultados son aproximadamente los mismos, si bien es aconsejable su utilización.

El receptor de infrarrojos necesita una tensión de alimentación entre 4.7 y 5.3 Voltios. Si la tensión baja de 4.7 el receptor deja de funcionar.

La figura 1.5 está obtenida directamente de las hojas de especificaciones del receptor IS1U60, y muestra la estructura interna. La dos flechas de la izquierda representa la luz infrarroja, que es convertida en una señal eléctrica por el led. La señal se amplifica, se le elimina la componente continua, se filtra para dejar pasar sólo frecuencias próximas a

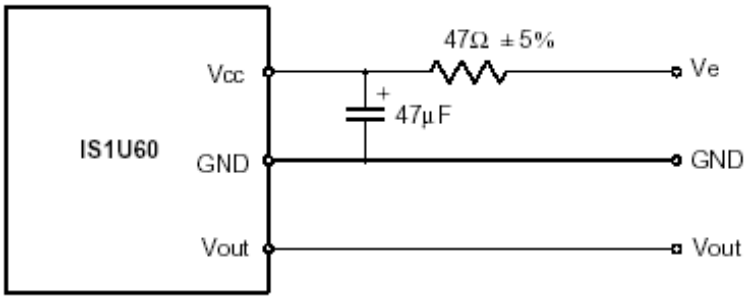


Figura 1.4: Filtro de la alimentación del IS1U60

38Khz, luego se rectifica (demodulador + integrador), y finalmente se convierte en una señal compatible TTL.

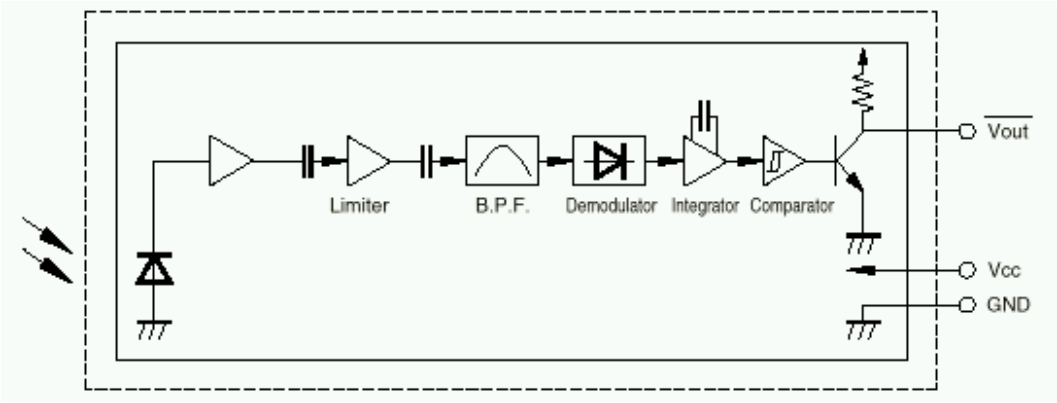


Figura 1.5: Hardware del receptor de infrarrojos

1.3.3. Protocolos y codificaciones de las señales IR

La obtención de un buen algoritmo para el proceso de recepción y reconocimiento de cualquier señal IR procedente de mandos a distancia es una tarea más o menos compleja que hay que iniciar con el estudio del tipo de señales y codificaciones implicadas en este tipo de comunicaciones. Comenzamos por lo tanto haciendo un estudio de este tipo de señales de uso muy común en cualquier hogar.

Tipos de codificación en las señales IR

Las codificaciones habituales empleadas en las señales infrarrojas de los mandos a distancia de electrodomésticos son *Pulse coded* (codificación por pulsos), *Space coded* (codificación de espacios) y *Shitf coded* (codificación por desplazamiento).

Pulse coded

Lo usa Sony en sus mandos de control remoto. La información va codificada en la anchura de los pulsos de la señal (ver figura 1.8).

Space coded

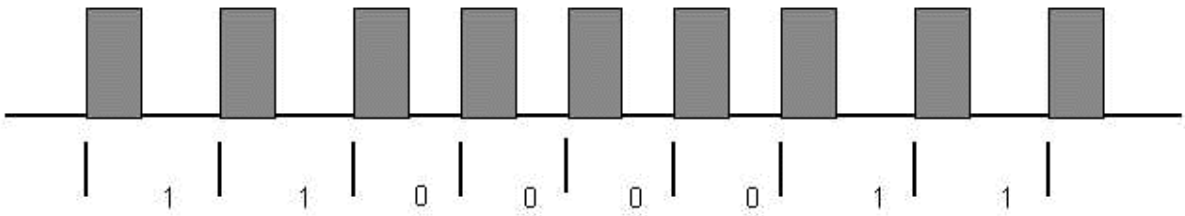


Figura 1.6: Codificación de espacios

Esta codificación varía la anchura de los espacios existentes entre los pulsos para incluir en la señal la información.

Shift coded

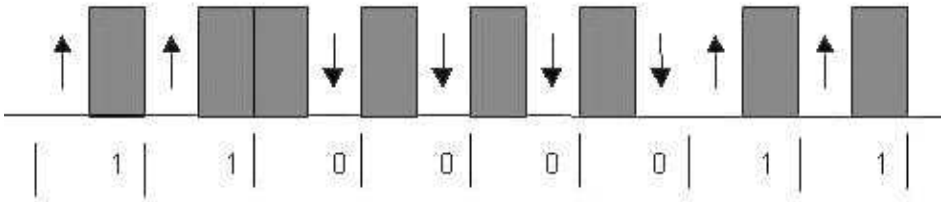


Figura 1.7: Codificación Bi-fase

Tal y como su propio nombre indica, Bi-fase, la información se encuentra incluida en la fase de la señal. En la mitad del tiempo de bit se produce una transición de la señal que dependiendo de en que sentido se produzca se tratará de un bit 1 o bien de un bit 0. A diferencia de lo que ocurre en las codificaciones que envían la información modulando la anchura ya sea del espacio o del pulso, en el código Manchester cada bit de información tiene idéntica duración.

Protocolos empleados por los mandos a distancia de infrarrojos

Aunque no necesitamos conocer los protocolos particulares utilizados por los fabricantes de mandos a distancia para pasar las señales que genera el receptor IR a su salida a un formato binario, vamos a ver un estudio de los que se utilizan porque puede ser conveniente para hacernos una idea de como se emplean las codificaciones vistas en el punto 1.3.3. Este estudio nos va a permitir también conocer el orden de los tiempos manejados en estas señales, detalle que es muy interesante conocer en el proceso de toma de tiempos.

REC-80

Comenzamos viendo REC-80 que también es conocido como RECS-80 y que es un protocolo que emplea la modulación por cambio de la anchura de los pulsos.

Se considera un tiempo mínimo 'T' de 600μs; cada bit de información está codificado mediante un nivel bajo y un nivel alto. Los bits '0' se representan con un pulso de duración 'T' seguido de un espacio de duración '2T', mientras que los bits '1' se codifican como un pulso de tiempo 'T' seguido de un espacio de valor '3T'. La longitud del código es de 48 bits sin tener en cuenta el bit '0' de fin de la transmisión. Mientras la señal está a nivel alto, se está transmitiendo la portadora de 36 KHz y por tanto la radiación infrarroja.

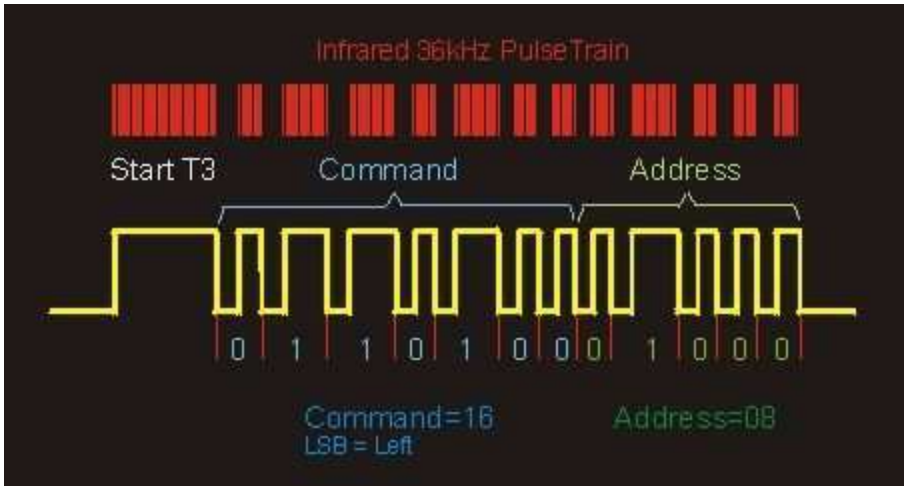


Figura 1.8: Señal REC-80

La trama de bits generada al pulsar una tecla en el mando tiene el siguiente formato:

1. Primero se transmite una cabecera, que consiste en el llamado bit START con una duración a nivel alto de 3T (1800 us).

2. Seguidamente se transmiten 12 bits (comenzando por el LSB) de los cuales los 7 primeros corresponden al código de COMANDO o función de la tecla pulsada, y los 5 restantes a la DIRECCIÓN del dispositivo. En la figura 1.8 el comando es 16h y la dirección 02h. Si se mantiene pulsada la tecla, la trama se repite periódicamente cada 25 ms.

RC-5

Al contrario que REC-80 es un protocolo muy poco habitual que fué desarrollado por Philips. Utiliza la codificación bifase con la particularidad de que una transición de nivel alto a nivel bajo representa un bit '0' y una transición en sentido contrario, es decir, de nivel bajo a nivel alto implica un bit '1'. En el caso de enviar de forma consecutiva bits del mismo valor binario necesita transiciones adicionales al comienzo de cada tiempo de bit para tener el nivel de comienzo que le corresponde. Esta transición adicional no es necesaria si el siguiente bit tiene un valor diferente.

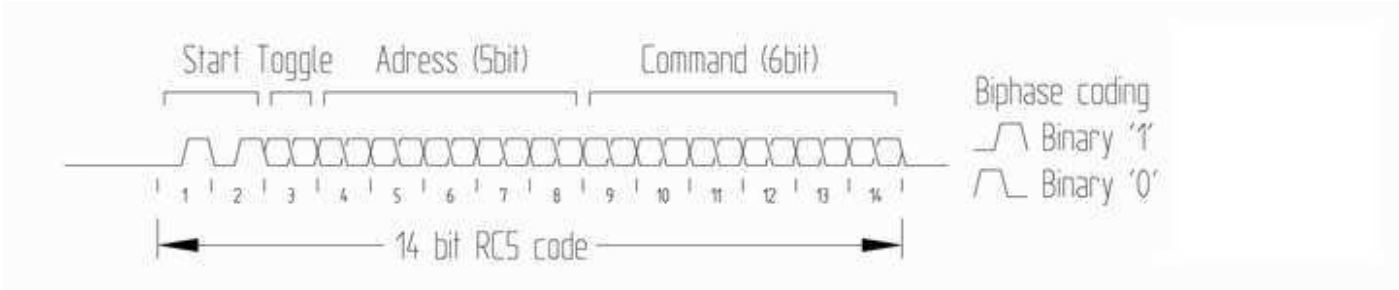


Figura 1.9: Señal RC5

El código RC05 tiene una longitud de 14 bits: Los 2 primeros bits (S1,S2) corresponden al sincronismo de la trama y siempre son '1' y sirven para calibrar el AGC del detector. El siguiente bit (T), es para detectar si la tecla permanece pulsada de forma continuada (CHK). Este bit cambia de estado (Toggle) cada vez que se transmite un nuevo comando. Los siguientes 5 bits (A4...A0) corresponden al campo de DIRECCIÓN que especifica el tipo de dispositivo (TV, CD, VCR, etc). Por último, los 6 bits restantes (C5...C0) representan al COMANDO o función dentro del dispositivo direccionado.

El primer bit transmitido en una secuencia RC5 es el más significativo, el MSB. Cada bit tiene una duración de 1,728 ms y la trama se repite periódicamente cada 130 ms aproximadamente, si se mantiene pulsada la misma tecla. Para la detección del código sería suficiente con detectar el primer flanco de bajada de la señal (primer bit) y muestrear

a partir de este instante a los 4,752ms (2,75 veces el tiempo de bit) para detectar el primer bit de la DIRECCIÓN, el siguiente a 1,728 ms, y así sucesivamente con el resto de bits.

SIRCS

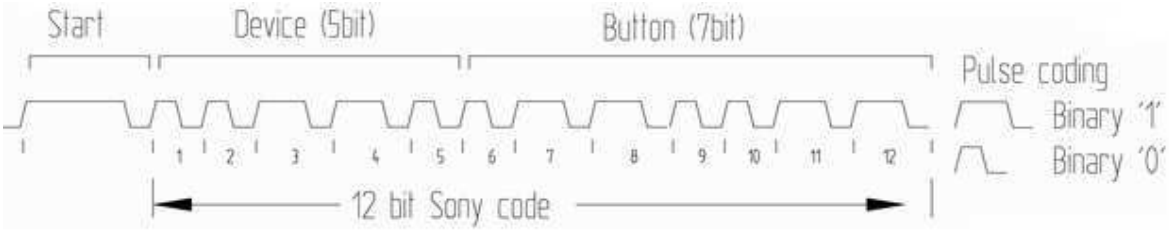


Figura 1.10: SIRCS de Sony

Pertenece a Sony y se basa en la codificación por pulsos. El primer bit transmitido es el menos significativo, el LSB. Los bits de datos pueden ser 12 o 15 pero el estándar señala tan sólo 12. De éstos, 5 identifican al dispositivo al que se dirige la señal y los 7 restantes determinan el botón que fue pulsado. Al contrario de lo que ocurre con la secuencia de bits de Philips la secuencia de comienzo no toma parte en la cuenta del número de bits.

La señal de sincronización de Sony dura 2.4ms y le siguen 0.6ms de espacio. Como la duración del periodo de referencia a partir del cual se construye la señal *Pulse coded* es de 600μs se obtienen la duración mínima y la duración máxima de la señal SIRCS que aparecen en las fórmulas 1.1 y 1.2 respectivamente.

$$3{,}0 + 12 \times 1{,}2 = 17{,}4ms \tag{1.1}$$

$$3{,}0 + 15 \times 1{,}8 = 30ms \tag{1.2}$$

1.3.4. Fabricantes compatibles con el módulo receptor

Los fabricantes de mandos a distancia con los que se ha probado el módulo receptor de infrarrojos y el resultado ha sido satisfactorio son los que se listan a continuación:

- PANASONIC
- TECHNICS
- AIWA
- SONY

- DAEWOO
- SABA
- PIONEER

1.4. Los puertos serie

1.4.1. Introducción a las comunicaciones serie

Las comunicaciones serie se utilizan para enviar datos a través de largas distancias, ya que las comunicaciones en paralelo exigen demasiado cableado para ser operativas.

Los equipos de comunicaciones serie se pueden dividir entre simplex, half-duplex y full-duplex. Una comunicación serie simplex envía información en una sola dirección (p.e. una emisora de radio comercial). Half-duplex significa que los datos pueden ser enviados en ambas direcciones entre dos sistemas, pero en una sola dirección al mismo tiempo. En una transmisión full-duplex cada sistema puede enviar y recibir datos al mismo tiempo.

Hay dos tipos de comunicaciones: síncronas y asíncronas. En una transmisión síncrona los datos son enviados en bloques; el transmisor y el receptor son sincronizados por uno o más caracteres especiales llamados caracteres sync.

El puerto serie del PC es un dispositivo asíncrono. Comenzamos describiendo este tipo de sistemas.

1.4.2. Protocolo serie asíncrono: Norma RS-232

El puerto serie y la comunicación asíncrona

El puerto serie es un dispositivo muy extendido y ya sea uno o dos, con conector grande o pequeño, todos los equipos PC lo incorporan actualmente. Debido a que el estándar del puerto serie se mantiene desde hace muchos años, la institución de normalización americana (EIA) ha escrito la norma RS-232-C que regula el protocolo de la transmisión de datos, el cableado, las señales eléctricas y los conectores en los que debe basarse una conexión RS-232.

La comunicación realizada con el puerto serie es una comunicación asícrona. Para la sincronización de una comunicación se precisa siempre de una línea adicional a través de la cual el emisor y el receptor intercambian la señal del pulso. Pero en la transmisión serie a través de un cable de dos líneas esto no es posible ya que ambas están ocupadas por los datos y la masa. Por este motivo se intercalan antes y después de los datos informaciones

de estado según el protocolo RS-232. Esta información es determinada por el emisor y receptor al estructurar la conexión mediante la correspondiente programación de sus puertos serie. Esta información puede ser la siguiente:

- **Bit de paridad**. Con este bit se pueden descubrir errores en la transmisión. Se puede dar paridad par o impar. En la paridad par, por ejemplo, la palabra de datos a transmitir se completa con el bit de paridad de manera que el número de bits 1 enviados sea par.
- **Bit de parada**. Indica la finalización de la transmisión de una palabra de datos. El protocolo de transmisión de datos permite 1, 1.5 y 2 bits de parada.
- **Bit de inicio**. Cuando el receptor detecta el bit de inicio sabe que la transmisión ha comenzado y es a partir de entonces que debe leer las señales de la línea a distancias concretas de tiempo, en función de la velocidad determinada.

En una transmisión asíncrona, un bit identifica el comienzo y 1 o 2 bits identifican su final; no es necesario ningún carácter de sincronismo. Los bits de datos son enviados al receptor después del bit de start. El bit de menos peso es transmitido primero. Un carácter de datos suele consistir en 7 u 8 bits. Dependiendo de la configuración de la transmisión un bit de paridad es enviado después de cada bit de datos. Finalmente se transmiten 1 o 2 bits de stop.

Funcionamiento del puerto serie RS-232

Estructura de un carácter

En modo asícrono, la primera transición de '1' a '0' es llamada Start Bit, que será seguida, por 5, 6, 7 u 8 bits de datos.

Se puede definir el octavo bit como el bit de paridad (indica si el numero de bits transmitidos es par o impar, para detectar fallos). En ese caso, se transferirá el carácter en 7 bits.

Al final de la transmisión del carácter, la señal debe obligatoriamente regresar a '0', esto se llama el Stop Bit, para atender el próximo Start Bit. Se puede definir 1, 1.5 o 2 Stop bits. Se puede apreciar la forma de un carácter en la figura 1.11

Normalmente, el protocolo utilizado es 8N1 (que significa, 8 bits de datos, sin paridad y con 1 bit de Stop) cuya configuración es la que aparece en la figura 1.12.

Modo de trabajo

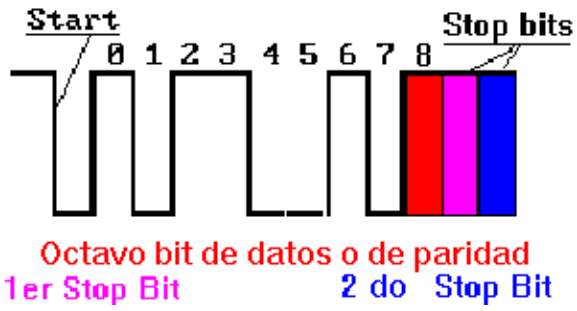


Figura 1.11: Estructura de un carácter

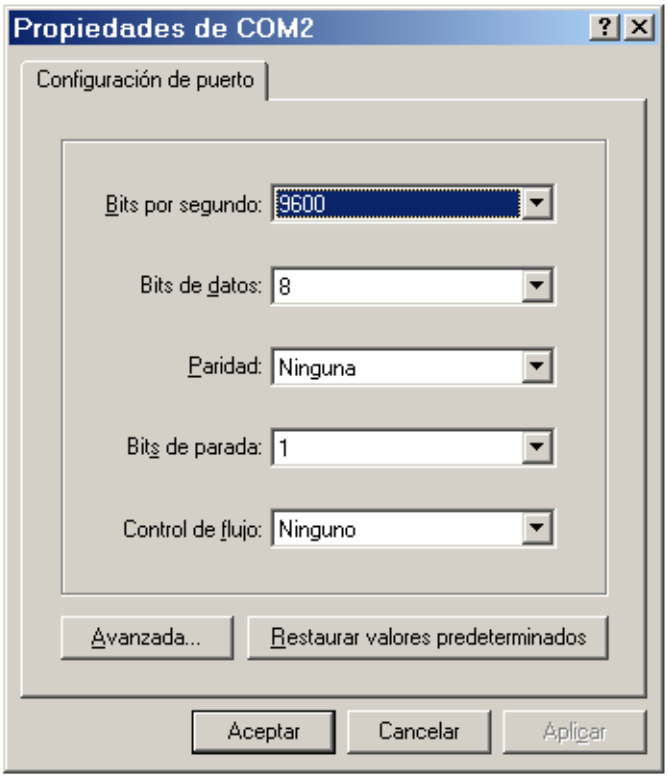


Figura 1.12: Configuración habitual del Hyperterminal

Tanto el aparato a conectar como el ordenador (o el programa terminal) tienen que usar el mismo protocolo serie para comunicarse entre sí. Puesto que el estándar RS-232 no permite indicar en que modo se esta trabajando, es el usuario quien tiene que decidirlo y configurar ambas partes. Los parámetros que hay que configurar son: protocolo serie (8N1), velocidad del puerto serie, y protocolo de control de flujo. Este último puede ser por hardware (el handshaking RTS/CTS) o bien por software (XON/XOFF, el cual no es

muy recomendable ya que no se pueden realizar transferencias binarias). La velocidad del puerto serie no tiene por que ser la misma que la de transmisión de los datos, de hecho debe ser superior. Por ejemplo, para transmisiones de 1200 baudios es recomendable usar 9600, y para 9600 baudios se pueden usar 38400 (o 19200).

Reconstrucción de la señal

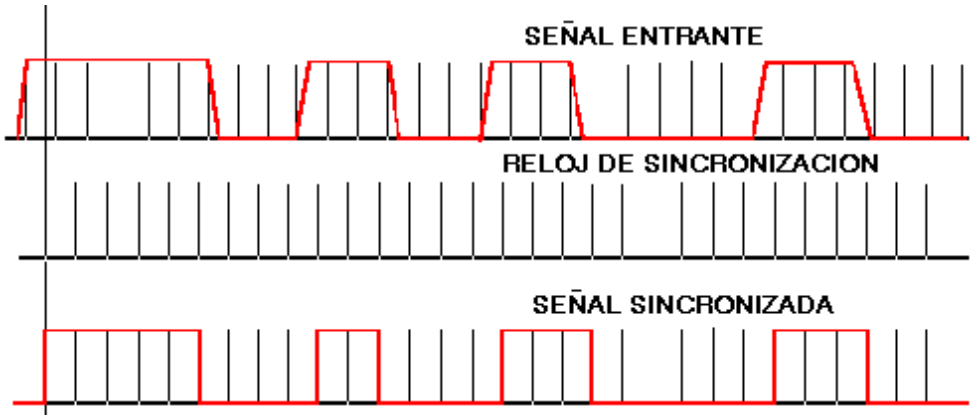


Figura 1.13: Sincronización

Una vez que ha comenzado la transmisión de un dato, los bits tienen que llegar uno detrás de otro a una velocidad constante y en determinados instantes de tiempo. Por eso se dice que el RS-232 es asíncrono por carácter y síncrono por bit. Debido a que se transmite una señal en modo asíncrono sobre una línea, el receptor debe reconstruir ésta tal y como se aprecia en la figura 1.13.

Por este motivo se utiliza un reloj cuya frecuencia es un múltiplo de la frecuencia de emisión (16 o 64 veces más elevada).

Así, se testea la polaridad de la señal entrante a cada tic-tac del reloj. Cuanto más elevada es la frecuencia de los tic-tac , la señal será reproducida más fielmente.

Todo este trabajo de muestreo es producido por los circuitos llamados **UART** (Universal Asynchronous Receiver Transmitter) que aparecen en la figura 1.14.

Conector del puerto RS-232

El estándar especifica 25 pins de señal, y que el conector de DTE debe ser macho y el conector de DCE hembra (El único DCE que tenemos es el LCD de TECDIS; las tres placas de circuito impreso que constituyen los tres módulos de recepción de IR son DTE por lo que llevan un conector macho). El conector más usado es el DB-25, pero muchos

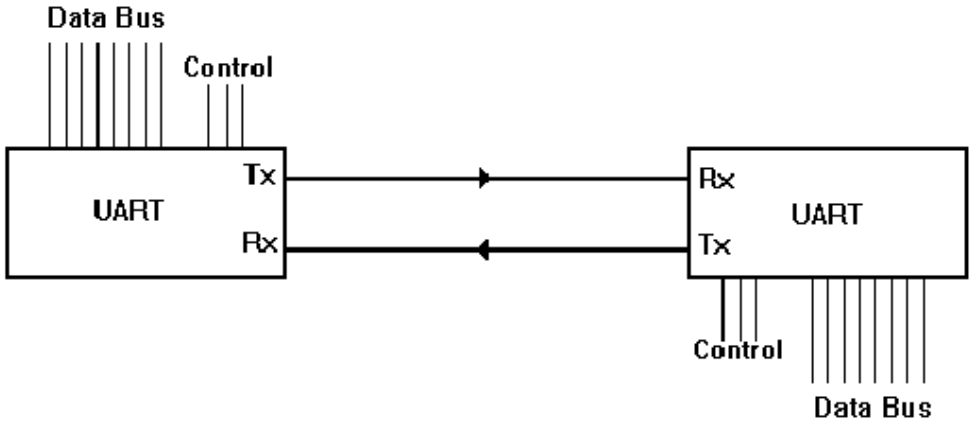


Figura 1.14: UART

de los 25 pins no son necesarios. Por esta razón en muchos puertos serie modernos, como en el que poseen las placas de circuito impreso de este proyecto, se utilizan conectores DB-9 como el de la figura 1.15.

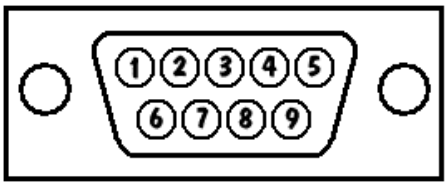


Figura 1.15: Conector DB9 macho

La tabla 1.1 muestra las señales asociadas a cada uno de los pines de los conectores DB9 macho y DB25 macho.

DTR indica que el ordenador está encendido, DSR que el aparato conectado a dicho puerto se encuentra encendido, RTS que el ordenador puede recibir datos (porque no está ocupado), CTS que el aparato conectado puede recibir datos, y DCD detecta que existe una comunicación, detecta la presencia de datos.

Originariamente el puerto serie, en concreto el del PC, estaba ideado para realizar comunicaciones através de un módem. Es por eso que la mayoría de los pines tienen funciones asignadas para controlar el estado y las funciones elementales de un módem (el handshaking). El módulo receptor de infrarrojos solo usa tres de todas las señales disponibles en el puerto serie para la comunicación con el LCD. Estas señales son las siguientes:

- **TxD**: Línea a través de la cual el módulo envía datos.

9-PIN	25-PIN	Tipo	Descripción
1	8	Entrada	DCD (Data Carrier Detect)
2	3	Entrada	RX (Receive Data)
3	2	Salida	TX (Transmit Data)
4	20	Salida	DTR (Data Terminal Ready)
5	7	-	GND
6	6	Entrada	DSR (Data Set Ready)
7	4	Salida	RTS (Request To Send)
8	5	Entrada	CTS (Clear To Send)
9	22		RI (Ring Indicator)

Cuadro 1.1: Pines y sus señales correspondientes

- **RxD**: Cable de recepción de datos.
- **GND**

UART

Un UART es un controlador conectado al bus de un ordenador (sobre la tarjeta madre de una PC, por ejemplo), para hacer oficio de convertidor bidireccional Serie / Paralelo y Paralelo / Serie. Normalmente se utilizan los siguientes modelos de este chip: 8250 (bastante antiguo, con fallos, solo llega a 9600 baudios), 16450 (versión corregida del 8250, llega hasta 115.200 baudios) y 16550A (con buffers de E/S). A partir de la gama Pentium, la circuitería UART de las placa base son todas de alta velocidad, es decir UART 16550A. De hecho, la mayoría de los módems conectables a puerto serie necesitan dicho tipo de UART, incluso algunos juegos para jugar en red a través del puerto serie necesitan de este tipo de puerto serie. Por eso hay veces que un 486 no se comunica con la suficiente velocidad con un PC Pentium... Los portátiles suelen llevar otros chips: 82510 (con búffer especial, emula al 16450) o el 8251 (no es compatible).

Conexión de un microcontrolador al puerto serie RS-232

En la norma RS-232 los voltajes para un nivel lógico alto están entre -3V y -15V mientras que los valores de un nivel lógico bajo se mantendrán entre +3V y +15V. Los niveles de tensión más usados son +12V y -12V.

Para conectar un microcontrolador a un PC o bien a otros dispositivos como el LCD de TECDIS por el puerto serie basta, como se dijo en el apartado 1.4.2, con las señales Tx, Rx y GND. Como tanto el PC como el display utilizan la norma RS232, los niveles de

tensión de los pines entán comprendidos entre +15 y -15 voltios. Los microcontroladores normalmente trabajan con niveles TTL (0-5v). Es necesario, por lo tanto, intercalar un circuito que adapte los niveles tal y como aparece en la figura 1.16.

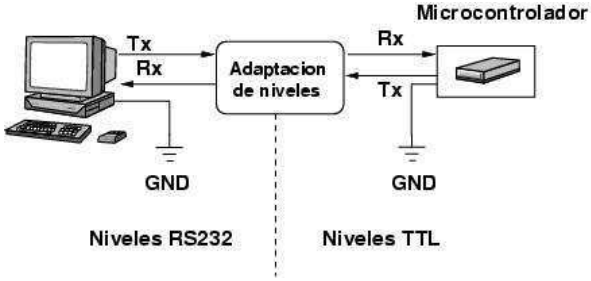


Figura 1.16: Adaptación de niveles de tensión en el puerto serie

Un C.I. que realiza esta adaptación de niveles es el MAX232 de Maxim. Este circuito integrado adapta los niveles de tensión de la norma RS-232 a niveles lógicos TTL y viceversa. Su uso se ha extendido de forma importante puesto que sólo necesita un nivel de tensión de +5V para generar los valores requeridos por el puerto RS-232.

El MAX232 necesita condensadores externos tal y como se aprecia en la figura 1.17. El valor habitual de estos condensadores es de $22\mu F$ aunque se pueden emplear de otros valores parecidos, siempre y cuando no sean menores de $1\mu F$. Una forma de evitar utilizar estos condensadores es el empleo del MAX233, C.I. que es sensiblemente más caro que el MAX232 por lo que no compensa en la mayoría de los casos.

Este circuito integrado lleva internamente 2 conversores de nivel de TTL a rs232 y otros 2 de rs232 a TTL con lo que en total podremos manejar 4 señales del puerto serie del PC; por lo general las más usadas son; TX, RX, RTS, CTS, estas dos últimas son las usadas para el protocolo handshaking pero no es imprescindible su uso como ya se ha comentado.

Está disponible un C.I. similar al MAX232; se trata del MAX3232 para diseños de baja potencia que emplean solo 3V.

El cable cruzado o Null Módem

Para comunicar dos dispositivos (ya sean dos PCs, un PC y un circuito, dos circuitos...) lo más común es utilizar el cable de módem nulo. Si se conecta un dispositivo al PC se trata de hacerle creer a éste que lo que tiene en el otro extremo del cable es un módem. Aunque con conectar las lineas TX, RX y GND sería suficiente para que todo funcione, que es lo que hacemos en nuestro caso, es conveniente añadir las conexiones

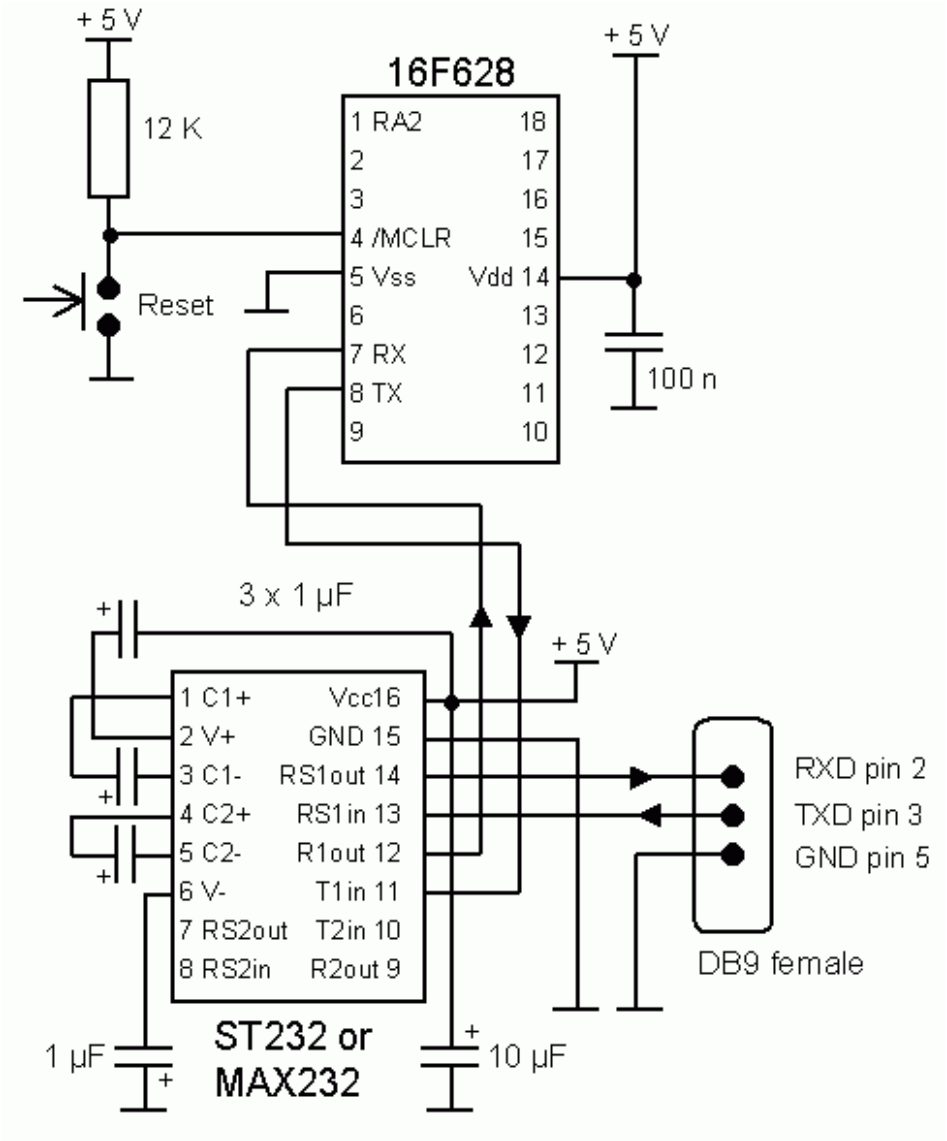


Figura 1.17: Montaje básico con MAX-232

adicionales de DTR, DCD, CTS y RTS si uno de los dispositivos es un PC, porque son las que le hacen saber que tiene un dispositivo conectado y preparado para iniciar una comunicación. El esquema del cable null módem es el de la figura 1.18. La longitud física del cable puede superar tranquilamente los 15 metros, aunque los límites vienen dados por la velocidad de transferencia de las comunicaciones.

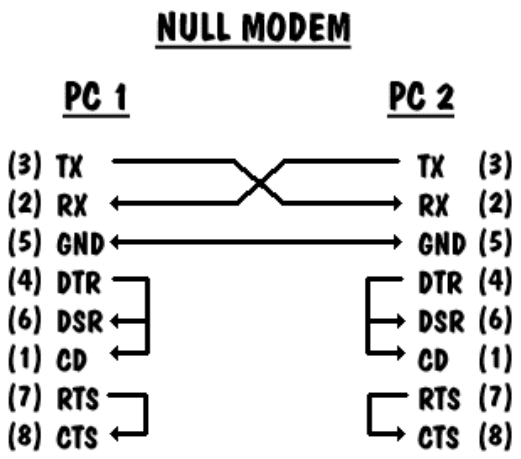


Figura 1.18: Cable null-módem

1.4.3. Protocolo serie síncrono: I2C

Características generales del puerto serie I2C

Diseñado por Philips, este sistema de intercambio de información a través de tan sólo dos cables permite a circuitos integrados y módulos OEM interactuar entre sí a velocidades relativamente lentas.

Como el protocolo I2C requiere muy poco hardware extra, es óptimo para transmitir o recibir pequeñas cantidades de datos.

Las características fundamentales del bus I2C son:

- Se necesitan solamente dos líneas, la de datos (SDA) y la de reloj (SCL).
- Cada dispositivo conectado al bus tiene un código de dirección seleccionable mediante software. Habiendo permanentemente una relación Master/ Slave entre el micro y los dispositivos conectados
- El bus permite la conexión de varios Masters, ya que incluye un detector de colisiones.
- El protocolo de transferencia de datos y direcciones posibilita diseñar sistemas completamente definidos por software.
- Los datos y direcciones se transmiten con palabras de 8 bits.

Las líneas SDA y SCL son del tipo drenador abierto, similares a las de colector abierto pero asociadas a un transistor de efecto de campo (FET). Se deben poner en estado

alto (conectar a la alimentación por medio de resistencias Pull-Up) para construir una estructura de bus tal que se permita conectar en paralelo múltiples entradas y salidas.

Una de las mayores ventajas de este protocolo es que no necesitas una señal de reloj precisa y síncrona. El protocolo aún funciona cuando hay variaciones en la señal del reloj.

Uso extendido del bus I2C

Tanto Philips como como otros fabricantes de dispositivos compatibles con I2C disponen de una amplia gama de circuitos integrados, incluyendo memorias RAM y E2PROM, microcontroladores, puertos de E/S, codificadores DTMF, tranceptores IR, conversores A/D y D/A, relojes de tiempo real, calendarios, etc.

Dado que no siempre se requiere alta velocidad de transferencia de datos este bus es ideal para sistemas donde es necesario manejar información entre muchos dispositivos y, al mismo tiempo, se requiere poco espacio y líneas de circuito impreso. Por ello es común ver dispositivos I2C en video grabadoras, sistemas de seguridad, electrónica automotriz, televisores, equipos de sonido y muchas otras aplicaciones más.

Incluso, y gracias a que el protocolo es lo suficientemente simple, usualmente se ven dispositivos I2C insertados en sistemas microcontrolados que no fueron diseñados con puertos I2C, siendo el protocolo es generado por el firmware.

También hay dispositivos de adaptación que permiten conectar buses originalmente paralelos a sistemas I2C. Tal es el caso del chip PCD 8584 de Philips el cual incorpora bajo su encapsulado todo lo necesario para efectuar dicha tarea.

Hay, además, circuitos integrados cuya única misión es adaptar los niveles presentes en el bus I2C y convertirlos desde y hacia TTL, permitiendo resolver fácil y rápidamente la interconexión de dispositivos de dicha familia con el I2C.

Funcionamiento del puerto serie I2C

Condiciones de START y STOP

Antes de que se establezca un intercambio de datos entre el circuito Master y los Esclavos, el Master debe informar el comienzo de la comunicación (condición de Start): La línea SDA cae a cero mientras SCL permanece en nivel alto. A partir de este momento comienza la transferencia de datos. Una vez finalizada la comunicación se debe informar de esta situación (condición de Stop). La línea SDA pasa a nivel alto mientras SCL permanece en estado alto. Las condiciones de inicio y de parada se pueden apreciar en la figura 1.19.

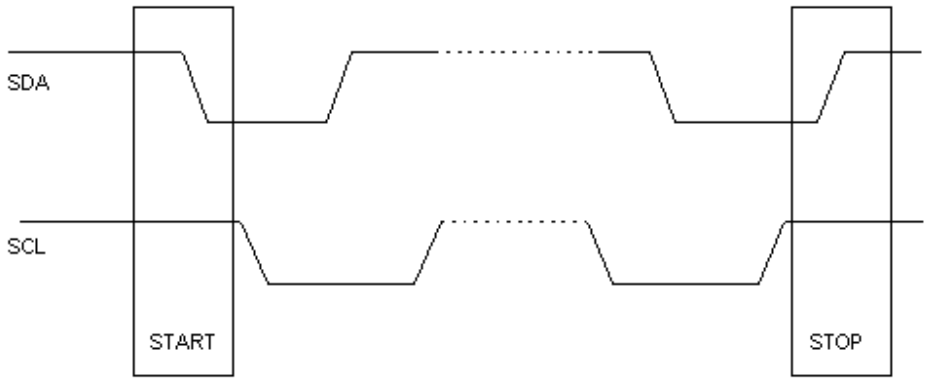


Figura 1.19: Señales de Start y Stop

Transferencia de datos

En primer lugar el Maestro genera la condición de Start. Cada palabra puesta en el bus SDA debe tener 8 bits, la primera palabra transferida contiene la dirección del Esclavo seleccionado. Luego el Master lee el estado de la línea SDA, si vale 0 (impuesto por el esclavo), el proceso de transferencia continúa. Si vale 1, indica que el circuito direccionado no valida la comunicación, entonces, el Maestro genera un bit de stop para liberar el bus I2C. Este acuse de recibo se denomina ACK (acknowledge) y es una parte importante del protocolo I2C. Al final de la transmisión, el Maestro genera la condición de Stop y libera el bus I2C, las líneas SDA y SCL pasan a estado alto.

Descripción del bus I2C

Comenzamos repasando lo que se vió en líneas generales en el apartado 1.4.3 de la introducción.

Para empezar, indicar, que en el bus I2C siempre hay un dispositivo maestro y uno o varios dispositivos esclavo tal y como se comentó anteriormente. El Master es el dispositivo que inicia la transferencia en el bus y genera la señal de Clock mientras que el esclavo (Slave) es el dispositivo direccionado. Los dos cables de este bus se llaman SDA (línea de datos) y SCL (línea de reloj). Las líneas SDA (serial Data) y SCL (serial Clock) son bidireccionales. Cada uno de los dispositivos del bus deben recibir energía en forma independiente (lo mismo que sucede con la comunicación tradicional RS-232). Las dos líneas del bus se encuentran normalmente conectadas vía resistencias ascendentes 4.7K pullup a una tensión de 5V (High). Esto da una conexión eléctrica OR ('o') entre todos los dispositivos. Un dispositivo simplemente extrae una línea a GND cuando quiere

transmitir un 0 o lo deja en estado lógico alto cuando envía un 1. Cuando el bus está libre, ambas líneas están en nivel alto. La transmisión bidireccional serie (8-bits) de datos puede realizarse a 100Kbits/s en el modo standard o 400 Kbits/s en el modo rápido. La cantidad de dispositivos que se pueden conectar al bus está limitada, solamente, por la máxima capacidad permitida: 400 pF.

Cada dispositivo es reconocido por su código (dirección) y puede operar como transmisor o receptor de datos. Además, cada dispositivo puede ser considerado como Master o Slave.

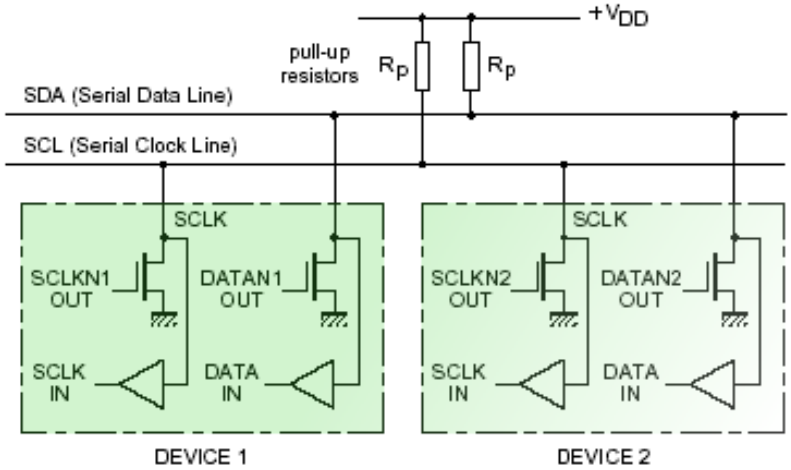


Figura 1.20: Bus I2C

Conexiones posibles del bus I2C con el PIC16F628

Hay dos opciones posibles para la implementación del puerto I2C con el microcontrolador PIC16F628. Estas dos formas de conexionado están condicionadas por la configuración interna de las patillas de los PIC16Fxxx. Si se usa la patilla con drenador abierto (open drain pin) de que dispone este PIC se usa el montaje de la figura 1.21.

Si no se dispone de una patilla con drenador abierto se utiliza este otro montaje, el que aparece en la figura 1.22, que emplea una patilla extra además de las correspondientes al bus I2C.

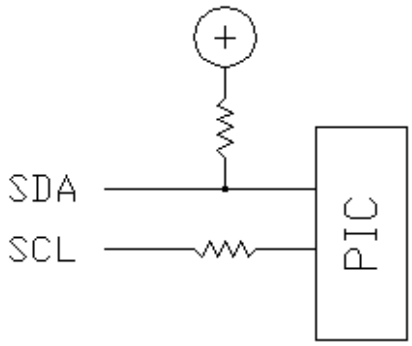


Figura 1.21: Conexionado del bus I2C con un PIC16Fxxx empleando la patilla con drenador abierto RA4

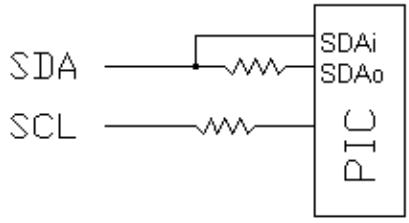


Figura 1.22: Conexionado del bus I2C con un PIC16Fxxx empleando tres patillas

1.5. El puerto paralelo

1.5.1. Generalidades

Con el puerto paralelo que se ha diseñado se transmiten los datos de byte en byte de forma síncrona. El bus paralelo interconecta el dispositivo Maestro y el dispositivo Esclavo permitiendo el envío rápido de los bytes de información, siempre en el mismo sentido; por lo tanto se trata de una comunicación *Simplex*.

Este bus consta de 10 líneas, lo que lo hace poco manejable para largas distancias. Es un serio inconveniente que presentan estos sistemas frente a los puertos serie.

Existe un intercambio de señales de control entre Maestro y Esclavo que sigue unos pasos determinados.

Sólo hay un dispositivo Maestro y otro dispositivo Esclavo, y sus funciones no son

intercambiables. Nuestro sistema será siempre el dispositivo Maestro.

1.5.2. Funcionamiento del puerto paralelo

Los pasos a seguir en una comunicación entre el dispositivo Maestro y el dispositivo Esclavo son mostrados en el diagrama temporal de la figura 1.23.

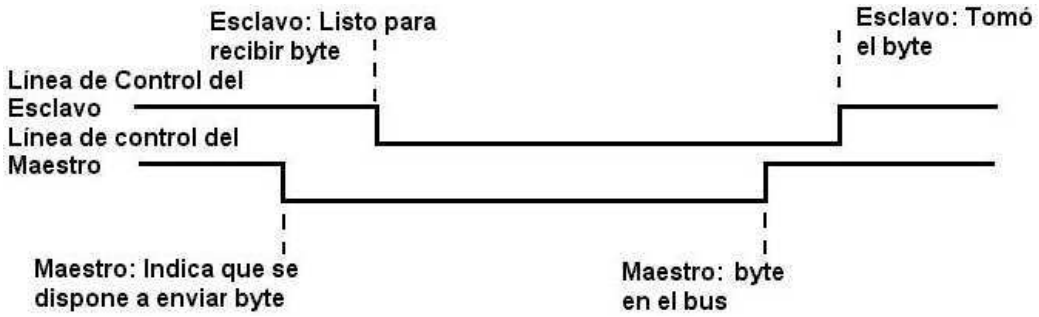


Figura 1.23: Funcionamiento del puerto paralelo

En las siguientes líneas se muestra explicada de forma detallada la secuencia de comunicación:

1. Inicialmente las dos líneas de control poseen un nivel lógico alto, están a '1'. La comunicación comienza cuando el maestro, a través de su línea de control de salida, le indica al esclavo que se dispone a enviar un byte de datos. Para tal finalidad el maestro pone esa línea de control a '0'.

2. Inmediatamente después de detectar el esclavo que su línea de control de entrada está a nivel bajo, cambia el nivel lógico de su línea de control de salida a '0' para informar al maestro de que está listo para recibir el byte.

3. En el momento en el que es maestro detecta la caída de la señal del esclavo pone el dato de 8 bits en el bus, y a continuación coloca de nuevo un '1' en la línea de control que es entrada en el esclavo.

4. El esclavo detecta el cambio en la línea que le indica que el dato está ya preparado en el bus, lee el byte presente en las 8 líneas y poniendo un '1' en el cable de control le dice al maestro que la transmisión finalizó correctamente.

1.5.3. Hardware del puerto paralelo

El circuito asociado al puerto paralelo que interconecta el PIC maestro con el sistema esclavo es sencillo. Se trata únicamente de 10 líneas que unen ambos dispositivos, enlazando entre sí sus puertos. De todas estas líneas, 8 son las encargadas de transportar los datos desde el Maestro hasta el Esclavo. Los otros 2 cables poseen las funciones de control para establecer los pasos implicados en el envío de los datos desde el Maestro hasta el dispositivo esclavo.

1.6. Conexión a LCD alfanumérico 16x2

1.6.1. Generalidades

Existen varios tipos de displays en el mercado, pero el que se va a usar es un modelo basado en el controlador Hitachi HD44780, el cual es bastante común. La mayoría de máquinas que llevan display (alarmas, máquinas de tabaco, controles de acceso...) suelen equipar uno de este tipo.

Los displays se pueden encontrar en varios tamaños aunque el funcionamiento es el mismo. Los nombres de referencia de este tipo de lcd son LM054, LM016L, LM020L, LM041L, LM032L, LM044L, LM243A, LM027, LM23A, LM017L, LM018L, etc ... Lo que se desee mostrar en él y la manera de hacerlo se controla mediante software por lo que la conexión es igual independientemente del tamaño. En la figura 1.24 se puede ver un display de 16x2 (2 líneas y 16 caracteres) que tiene el mismo aspecto que el LCD alfanumérico que se ha utilizado.

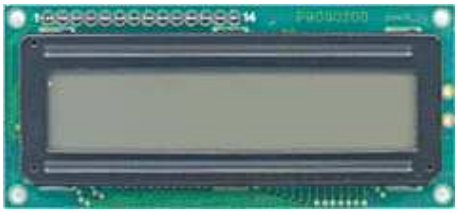


Figura 1.24: LCD alfanumérico 16x2 Hitachi HD44780

1.6.2. Instrucciones

La figura 1.25 muestra las señales asociadas al proceso de escritura de un byte ASCII en la pantalla del LCD. Si se emplean 8 hilos para los datos el byte es enviado de una sóla

vez. Si por el contrario tan sólo se dispone de 4 cables, el byte se envía en dos secuencias, que son acompañadas cada una de ellas por un pulso de reloj 'E'; en la primera se manda la parte alta del dato y en la segunda la parte baja del mismo dato.

La Línea R/S se usa para seleccionar cuando los bytes que viajan desde el microcontrolador al display son dato o bien instrucción. Si esta línea se encuentra en un nivel lógico alto, el byte de la posición actual del cursor del LCD se puede escribir o leer. Con esta patilla a nivel bajo lo que se está enviando al display es una intrucción, o bien se está intentando conocer el estado de la última instrucción que se envió para averiguar si ésta se completó o no.

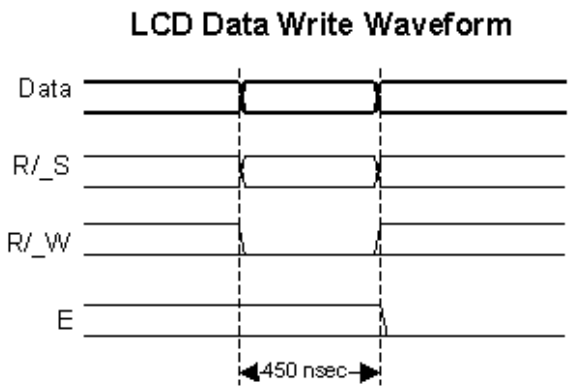


Figura 1.25: Proceso de escritura en el LCD hitachi

La tabla 1.2 muestra el repertorio de instrucciones del LCD Hitachi 44780. A continuación aparece una breve descripción de las intrucciones:

- **Set cursor move direction** (Establecer cursor, desplazar display)
  - ID: Cuando está activado incrementa el cursor después de cada byte que es escrito en el Display.
  - S: Se desplaza la pantalla cuando se escribe un byte
- **Enable display/Cursor** (Habilita display/cursor)
  - D: Encender display(1)/Apagar display(0)
  - C: Activar cursor(1)/Desactivar cursor(0)

R/S	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Instruction/Description
4	5	14	13	12	11	10	9	8	7	Pins
0	0	0	0	0	0	0	0	0	1	Clear Display
0	0	0	0	0	0	0	0	1	*	Return Cursor and LCD to Home Position
0	0	0	0	0	0	0	1	ID	S	Set Cursor Move Direction
0	0	0	0	0	0	1	D	C	B	Enable Display/Cursor
0	0	0	0	0	1	SC	RL	*	*	Move Cursor/Shift Display
0	0	0	0	1	DL	N	F	*	*	Set Interface Length
0	0	0	1	A	A	A	A	A	A	Move Cursor into CGRAM
0	0	1	A	A	A	A	A	A	A	Move Cursor to Display
0	1	BF	*	*	*	*	*	*	*	Poll the "Busy Flag"
1	0	D	D	D	D	D	D	D	D	Write a Character to the Display at the Current Cursor Position
1	1	D	D	D	D	D	D	D	D	Read the Character on the Display at the Current Cursor Position

Cuadro 1.2: Instrucciones del LCD Hitachi 44780

- B: Activar parpadeo del cursor(1)/Desactivar el parpadeo del cursor del cursor(0)

- **Move cursor/Shift display** (Mover cursor/Desplazar display)
  - SC: Activar desplazamiento del display(1)/Desactivar desplazamiento del display(0)
  - RL: Desplazamiento a la derecha(1)/Desplazamiento a la izquierda(0)
- **Set Interface Length** (Establecer número de bits de la interfaz)
  - DL: Longitud de la interfaz de datos 8(1)/4(0)
  - N: Número de líneas del display 1(1)/2(0)
  - F: Establecer fuente 5x10(1)/5x7(0)
- **Poll the "Busy Flag"** (Comprobar el flag de ocupación)
  - BF: Este bit está a '1' cuando el display está ocupado
- **Move cursor to CGRAM/Display**(Mover el cursor a CGRAM/Display)
  - A: Dirección

- **Read/Write ASCII to the display** (Leer/Escribir carácter ASCII en el display)
    - D: Dato

La lectura del los datos del display se usa en aplicaciones que requieren tomar los datos y volver a llevarlos de nuevo al LCD como por ejemplo en el caso de tener implementado un scroll de datos entre líneas. El flag de ocupación puede ser sondeado para determinar se ha terminado de procesar la última instrucción que fué enviada al LCD. En la mayoría de las aplicaciones la línea R/W se pone a cero ya que la lectura de los datos del display no es muy común. Este detalle simplifica los circuitos porque se puede prescindir de conectar este cable al microcontrolador ya que se hace innecesario intercambiar entre modo de lectura y modo de escritura.

El tiempo máximo que invierte el display en cada operación depende de la instrucción de la que se trate. Así en el caso de la instrucción de borrado de la pantalla o el de la instrucción de movimiento del cursor/display hasta el inicio, el tiempo empleado es de 4.1ms. Para ejecutar el resto de comandos basta con 160us. Es recomendable establecer como tiempos los valores máximos puesto que cada display requiere una velocidad diferente. Teniendo en cuenta este punto se pueden evitar muchos problemas.

1.6.3. Inicialización

Modo de 8 bits

Antes de enviar un comando o un dato al módulo LCD es necesario inicializar éste; para el modo de 8 bits se realiza con la siguiente serie de operaciones:

1. Esperar más de 15ms desde el arranque
2. Escribir 0x30 en el LCD y esperar 5ms a que se complete la operación
3. Escribir 0x30 en el LCD y esperar 160us hasta que se acabe de ejecutar la instrucción
4. Escribir de nuevo 0x30 en el LCD y esperar otros 160us o bien sondear el flag de ocupación.
5. Establecer las características de operación del LCD:
    - Escribir **establecer la longitud de la intarfaz**
    - Escribir **0x10 para deshabilitar el display**
    - Escribir **0x01 para borrar el display**

- Escribir **mover cursor desplazar display**
- Escribir **habilitar display/cursor y habilitar display y cursor opcional**

Modo de 4 bits

Para inicializar en modo de 4 bits se realizan las siguientes operaciones:

1. Esperar más de 15ms desde el arranque
2. Escribir 0x30 en el LCD y esperar 5ms a que se complete la operación
3. Escribir 0x30 en el LCd y esperar 160us hasta que se acabe de ejecutar la instrucción
4. Escribir de nuevo 0x30 en el LCD y esperar otros 160us o bien sondear el flag de ocupación.
5. Establecer las características de operación del LCD:
   - Escribir 0x02 para establecer el modo de 4 bits
   - Escribir **establecer longitud de la interfaz** (Requiere el envío en dos grupos de 4 bits)
   - Escribir **0x01/0x00 para deshabilitar el display**
   - Escribir **0x00/0x01 para borrar el display**
   - Escribir **establecer cursor, desplazar display**
   - Escribir **habilita display/cursor y habilita display y cursor opcional**

Una vez la inicialización está completa, el LCD puede recibir las instrucciones o los datos requeridos. Los caracteres son escritos en el display igual que los bytes de control con la salvedad de que la línea R/S se pone a '1'.

Si en la inicialización, se pone a '1' el bit S/C durante el comando de *mover cursor/desplazar display*, después de que cada byte se envíe al LCD, el cursor se colocará en la siguiente posición (puede ser a la derecha o bien a la izquierda). Normalmente, el bit S/C está a '1' con el bit R/L en el comando *mover cursor/desplazar display*, para que de esta manera los caracteres sean escritos de izquierda a derecha.

LCD	Primer carácter	Noveno carácter	Segunda línea	Tercera línea	Cuarta línea
8x1	0	-	-	-	-
16x1	0	0x40	-	-	-
16x1	0	8	0x40	-	-
8x2	0	-	0x40	-	-
10x2	0	8	0x40	-	-
16x2	0	8	0x40	-	-
20x2	0	8	0x40	-	-
24x2	0	8	0x40	-	-
30x2	0	8	0x40	-	-
32x2	0	8	0x40	-	-
40x2	0	8	0x40	-	-
16x4	0	8	0x40	0x20	0x60
20x4	0	8	0x40	0x20	0x60

Cuadro 1.3: Direcciones de acceso de display basados en el chip Hitachi 44780

1.6.4. Localización de una posición en el LCD

Uno de los puntos importantes a la hora de programar utilizando un display es saber como acceder a las diferentes posiciones de los bytes en un LCD y como seleccionar cada una de sus líneas. La tabla 1.3 muestra las direcciones con las que se accede a los bytes de diferentes displays LCD basados en el chip 44780. La primera columna de la tabla indica el número de caracteres por línea de cada LCD.

La mayoría de los displays LCD tienen el 44780 y un chip de apoyo para el control de las operaciones del LCD. El 44780 es responsable de la interfaz externa y provee de suficientes líneas de control para tener 16 caracteres en el LCD. El chip de apoyo ayuda al 44780 para que éste soporte más de 128 caracteres en un LCD. De los display que aparecen en la tabla 1.3 el 8x1 y el 16x1 no tienen el chip de apoyo. Este es el motivo por el cual el noveno carácter en el 16x1 no aparece como dirección 8 y si con la dirección que utilizan los displays de 2 líneas.

1.6.5. Caracteres

El conjunto de caracteres disponible en el 44780 es básicamente el que conforma ASCII. Hay algunos caracteres que en ASCII son de control (de 0x08 a 0x1F) y se pueden mostrar en el display como caracteres Japoneses; y otros como '\' no están disponibles. En la figura 1.26 aparece el repertorio de caracteres del LCD.

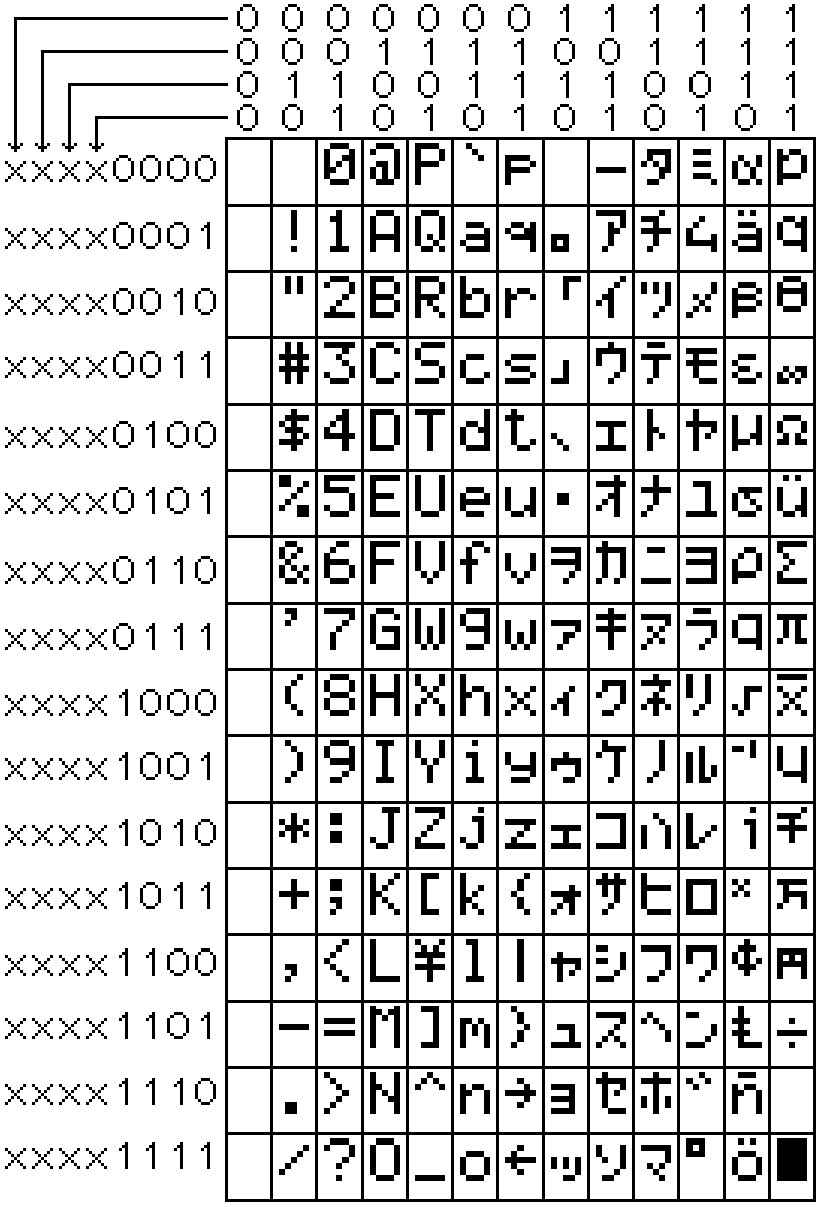


Figura 1.26: Repertorio de caracteres del LCD

1.6.6. Patillaje

Todos los displays basados en el controlador HD44780, disponen de una interfaz de 14 hilos; los retroiluminados llevan además otras dos conexiones para el circuito de iluminación. El patillaje es el siguiente:

1. **Masa**.
2. **Alimentación**. Esta patilla va conectada a 5 v.
3. **Contraste**. Para tener el contraste siempre al máximo, llevar esta patilla a masa directamente, en caso de querer variar el contraste será necesario poner un potenciómetro de 10K entre ésta y masa.
4. **Selección de registro**.
5. **Lectura/escritura**.
6. **Enable**.
7. **D0** (Primera línea de datos).
8. **D1**
9. **D2**
10. **D3**
11. **D4**
12. **D5**
13. **D6**
14. **D7**
15. **Alimentación retroiluminación +** (si el display no es retroiluminado, no se conectará esta patilla)
16. **Alimentación retroiluminación -** (si el display no es retroiluminado, no se conectará esta patilla)

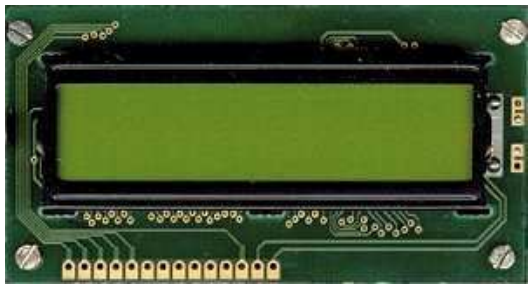


Figura 1.27: Vista superior del LCD 16x2

Figura 1.28: Vista inferior del LCD 16x2

1.6.7. Bus de 8 o de 4 bits

La decisión más importante a la hora de seleccionar el modo de funcionamiento del lcd es la de elegir el modo de envío de los bytes. Se pueden enviar los datos en un bus de 8 bits o bien de 4 en 4 bits. Si se selecciona la primera opción se utilizan todas las líneas de datos D0-D7. En el segundo caso se prescinde de las patillas D0-D3 y la información se envía a través de la parte alta del bus, es decir, los bits viajan en las líneas D4-D7.

Elegir un modo u otro depende de las posibilidades de diseño y de la velocidad requerida por la aplicación. Se usará el modo de 8 bits cuando se disponga en el microcontrolador que se busca conectar al display de un mínimo de 10 patillas libres (8 datos y 2 control) y sea requisito imprescindible la velocidad. Si tan sólo se dispone de 6 patillas libres (4 datos + 2 control) para interconectar con el lcd se utilizará el modo de 4 bits. La señal de reloj 'E' se usa para iniciar la transferencia de datos dentro del LCD.

1.7. PIC 16F628

1.7.1. Qué es un Microcontrolador

Definición

Un microcontrolador es un dispositivo electrónico capaz de llevar a cabo procesos lógicos. Estos procesos o acciones son programados en lenguaje ensamblador por el usuario, y son introducidos en este a través de un programador.

Un microcontrolador es un solo circuito integrado que contiene todos los elementos electrónicos que se utilizaban para hacer funcionar un sistema basado con un microprocesador; es decir contiene en un solo integrado la Unidad de Proceso, la memoria RAM, memoria ROM, puertos de entrada, salidas y otros periféricos, con la consiguiente reducción de espacio.

El microcontrolador es en definitiva un circuito integrado que incluye todos los componentes de un computador. Debido a su reducido tamaño es posible montar el controlador en el propio dispositivo al que gobierna. En este caso el controlador recibe el nombre de controlador empotrado (embedded controller).

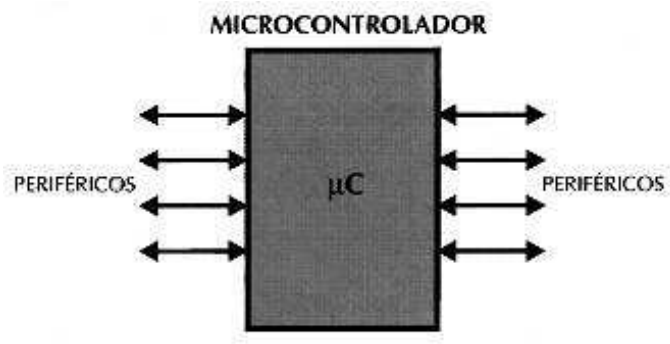


Figura 1.29: Microcontrolador

Ventajas de un microcontrolador frente a un microprocesador

Estas ventajas son reconocidas inmediatamente para aquellas personas que han trabajado con los microprocesadores y después pasaron a trabajar con los microcontroladores. Estas son las diferencias más importantes:

Por ejemplo la configuración mínima básica de un microprocesador estaba constituida por un Micro de 40 Pines, Una memoria RAM de 28 Pines, una memoria ROM de 28 Pines y un decodificador de direcciones de 18 pines; pero un microcontrolador incluye

todo estos elementos en un solo Circuito Integrado por lo que implica una gran ventaja en varios factores: En el circuito impreso por su amplia simplificación de circuitería, el costo para un sistema basado en microcontrolador es mucho menor y, lo mejor de todo, el tiempo de desarrollo de su proyecto electrónico se disminuye considerablemente.

Los microcontroladores hoy día

Los microcontroladores están conquistando el mundo. Están presentes en nuestro trabajo, en nuestra casa y en nuestra vida, en general. Se pueden encontrar controlando el funcionamiento de los ratones y teclados de los computadores, en los teléfonos, en los hornos microondas y los televisores de nuestro hogar. Pero la invasión acaba de comenzar y el nuevo siglo XXI está siendo testigo de la conquista masiva de estos diminutos computadores, que gobernarán la mayor parte de los aparatos que fabricaremos y usaremos los humanos.

Cada vez existen más productos que incorporan un microcontrolador con el fin de aumentar sustancialmente sus prestaciones, reducir su tamaño y coste, mejorar su fiabilidad y disminuir el consumo.

Algunos fabricantes de microcontroladores superan el millón de unidades de un modelo determinado producidas en una semana. Este dato puede dar una idea de la masiva utilización de estos componentes.

Los microcontroladores están siendo empleados en multitud de sistemas presentes en nuestra vida diaria, como pueden ser juguetes, horno microondas, frigoríficos, televisores, computadoras, impresoras, módems, el sistema de arranque de nuestro coche, etc. Y otras aplicaciones como instrumentación electrónica, control de sistemas en una nave espacial, etc. Una aplicación típica podría emplear varios microcontroladores para controlar pequeãs partes del sistema. Estos pequeños controladores podrían comunicarse entre ellos y con un procesador central, probablemente más potente, para compartir la información y coordinar sus acciones, como, de hecho, ocurre ya habitualmente en cualquier PC.

Tipos de arquitecturas de microcontroladores

Arquitectura Von Neumann

La arquitectura tradicional de computadoras y microprocesadores está basada en la arquitectura Von Neumann, en la cual la unidad central de proceso (CPU), está conectada a una memoria única donde se guardan las instrucciones del programa y los datos.

El tamaño de la unidad de datos o instrucciones está fijado por el ancho del bus que comunica la memoria con la CPU. Así un microprocesador de 8 bits con un bus de 8 bits, tendrá que manejar datos e instrucciones de una o más unidades de 8 bits (bytes) de longitud. Si tiene que acceder a una instrucción o dato de más de un byte de longitud, tendrá que realizar más de un acceso a la memoria.

Y el tener un único bus hace que el microprocesador sea más lento en su respuesta, ya que no puede buscar en memoria una nueva instrucción mientras no finalicen las transferencias de datos de la instrucción anterior.

Resumiendo todo lo anterior, las principales limitaciones que nos encontramos con la arquitectura Von Neumann son :

1. La limitación de la longitud de las instrucciones por el bus de datos, que hace que el microprocesador tenga que realizar varios accesos a memoria para buscar instrucciones complejas.

2. La limitación de la velocidad de operación a causa del bus único para datos e instrucciones que no deja acceder simultáneamente a unos y otras, lo cual impide superponer ambos tiempos de acceso.

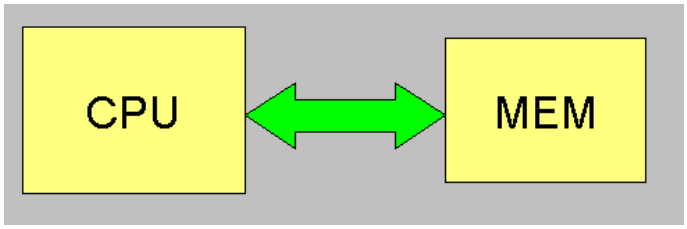


Figura 1.30: Arquitectura Von Neumann

Arquitectura Harvard

La arquitectura Harvard tiene la unidad central de proceso (CPU) conectada a dos memorias (una con las instrucciones y otra con los datos) por medio de dos buses diferentes.

Una de las memorias contiene solamente las instrucciones del programa (Memoria de Programa), y la otra sólo almacena datos (Memoria de Datos).

Ambos buses son totalmente independientes y pueden ser de distintos anchos. Para un procesador de Set de Instrucciones Reducido, o RISC (Reduced Instrucción Set Computer), el set de instrucciones y el bus de memoria de programa pueden diseñarse de tal

manera que todas las instrucciones tengan una sola posición de memoria de programa de longitud.

Además, al ser los buses independientes, la CPU puede acceder a los datos para completar la ejecución de una instrucción, y al mismo tiempo leer la siguiente instrucción a ejecutar.

Ventajas de esta arquitectura:

1. El tamaño de las instrucciones no esta relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa.

2. El tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad en cada operación.

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

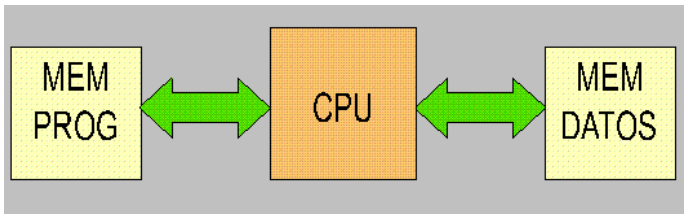


Figura 1.31: Arquitectura Harvard

1.7.2. Estructura y elementos de los Microcontroladores

A continuación se describen los elementos más comunes en todo tipo de microcontroladores y sistemas.

El procesador

Es el elemento más importante del microcontrolador y determina sus principales características, tanto a nivel hardware como software.

Se encarga de direccionar la memoria de instrucciones, recibir el código OP de la instrucción en curso, su decodificación y la ejecución de la operación que implica la instrucción, así como la búsqueda de los operandos y el almacenamiento del resultado.

Existen tres orientaciones en cuanto a la arquitectura y funcionalidad de los procesadores actuales.

- **CISC:** Un gran número de procesadores usados en los microcontroladores están basados en la filosofía CISC (Computadores de Juego de Instrucciones Complejo). Disponen de más de 80 instrucciones máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos ciclos para su ejecución.

  Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.

- **RISC:** Tanto la industria de los computadores comerciales como la de los microcontroladores están decantándose hacia la filosofía RISC (Computadores de Juego de Instrucciones Reducido). En estos procesadores el repertorio de instrucciones máquina es muy reducido y las instrucciones son simples y, generalmente, se ejecutan en un ciclo.

  La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

- **SISC:** En los microcontroladores destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es específico, o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista. Esta filosofía se ha bautizado con el nombre de SISC (Computadores de Juego de Instrucciones Específico).

Memoria

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, tipo ROM, y se destina a contener el programa de instrucciones que gobierna la aplicación. Otra parte de memoria será tipo RAM, volátil, y se destina a guardar las variables y los datos.

Hay dos peculiaridades que diferencian a los microcontroladores de los computadores personales:

1. No existen sistemas de almacenamiento masivo como disco duro o disquetes.

2. Como el microcontrolador sólo se destina a una tarea en la memoria ROM, sólo hay que almacenar un único programa de trabajo.

La RAM en estos dispositivos es de poca capacidad pues sólo debe contener las variables y los cambios de información que se produzcan en el transcurso del programa. Por otra parte, como sólo existe un programa activo, no se requiere guardar una copia del mismo en la RAM pues se ejecuta directamente desde la ROM.

Los usuarios de computadores personales están habituados a manejar Megabytes de memoria, pero, los diseñadores con microcontroladores trabajan con capacidades de ROM comprendidas entre 512 bytes y 8 k bytes y de RAM comprendidas entre 20 y 512 bytes.

Según el tipo de memoria ROM que dispongan los microcontroladores, la aplicación y utilización de los mismos es diferente. Se describen las cinco versiones de memoria no volátil que se pueden encontrar en los microcontroladores del mercado.

ROM con máscara

Es una memoria no volátil de sólo lectura cuyo contenido se graba durante la fabricación del chip. Teniendo una idea de cómo se fabrican los circuitos integrados, se sabe de donde viene el nombre. Estos se fabrican en obleas que contienen varias decenas de chips. Estas obleas se fabrican a partir de procesos fotoquímicos, donde se impregnan capas de silicio y óxido de silicio, y según convenga, se erosionan al exponerlos a la luz. Como no todos los puntos han de ser erosionados, se sitúa entre la luz y la oblea una mascara con agujeros, de manera que donde deba incidir la luz, esta pasará. Con varios procesos similares pero más complicados se consigue fabricar los transistores y diodos micrométricos que componen un chip.

El elevado coste del diseño de la máscara sólo hace aconsejable el empleo de los microcontroladores con este tipo de memoria cuando se precisan cantidades superiores a varios miles de unidades.

OTP

El microcontrolador contiene una memoria no volátil de sólo lectura programable una sola vez por el usuario. OTP (One Time Programmable). Es el usuario quien puede escribir el programa en el chip mediante un sencillo grabador controlado por un programa desde un PC.

La versión OTP es recomendable cuando es muy corto el ciclo de diseño del producto, o bien, en la construcción de prototipos y series muy pequeñas.

Tanto en este tipo de memoria como en la EPROM, se suele usar la encriptación mediante fusibles para proteger el código contenido.

EPROM

Los microcontroladores que disponen de memoria EPROM (Erasable Programmable Read OnIy Memory) pueden borrarse y grabarse muchas veces. La grabación se realiza, como en el caso de los OTP, con un grabador gobernado desde un PC. Si, posteriormente, se desea borrar el contenido, disponen de una ventana de cristal en su superficie por la que se somete a la EPROM a rayos ultravioleta durante varios minutos. Las cápsulas son de material cerámico y son más caros que los microcontroladores con memoria OTP que están hechos con material plástico.

EEPROM

Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programmable Read OnIy Memory). Tanto la programación como el borrado, se realizan eléctricamente desde el propio grabador y bajo el control programado de un PC. Es muy cómoda y rápida la operación de grabado y la de borrado. No disponen de ventana de cristal en la superficie.

Los microcontroladores dotados de memoria EEPROM una vez instalados en el circuito, pueden grabarse y borrarse cuantas veces se quiera sin ser retirados de dicho circuito. Para ello se usan grabadores en circuito que confieren una gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo.

El número de veces que puede grabarse y borrarse una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua. Son muy idóneos para la enseñanza y la Ingeniería de diseño.

Se va extendiendo en los fabricantes la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables para guardar y modificar cómodamente una serie de parámetros que adecuan el dispositivo a las condiciones del entorno.

Este tipo de memoria es relativamente lenta.

FLASH

Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar. Funciona como una ROM y una RAM pero consume menos y es más pequeña.

A diferencia de la ROM, la memoria FLASH es programable en el circuito. Es más rápida y de mayor densidad que la EEPROM.

La alternativa FLASH está recomendada frente a la EEPROM cuando se precisa gran cantidad de memoria de programa no volátil. Es más veloz y tolera más ciclos de escritura/borrado.

Las memorias EEPROM y FLASH son muy útiles al permitir que los microcontroladores que las incorporan puedan ser reprogramados en circuito, es decir, sin tener que

sacar el circuito integrado de la tarjeta. Así, un dispositivo con este tipo de memoria incorporado al control del motor de un automóvil permite que pueda modificarse el programa durante la rutina de mantenimiento periódico, compensando los desgastes y otros factores tales como la compresión, la instalación de nuevas piezas, etc. La reprogramación del microcontrolador puede convertirse en una labor rutinaria dentro de la puesta a punto.

Puertas de Entrada y Salida

Las puertas de Entrada y Salida (E/S) permiten comunicar al procesador con el mundo exterior, a través de interfaces, o con otros dispositivos. Estas puertas, también llamadas puertos, son la principal utilidad de las patillas de un microprocesador.

Según los controladores de periféricos que posea cada modelo de microcontrolador, las líneas de E/S se destinan a proporcionar el soporte a las señales de entrada, salida y control.

Reloj principal

Todos los microcontroladores disponen de un circuito oscilador que genera una onda cuadrada de alta frecuencia, que configura los impulsos de reloj usados en la sincronización de todas las operaciones del sistema. Esta señal del reloj es el motor del sistema y la que hace que el programa y los contadores avancen.

Generalmente, el circuito de reloj está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes exteriores para seleccionar y estabilizar la frecuencia de trabajo. Dichos componentes suelen consistir en un cristal de cuarzo junto a elementos pasivos o bien un resonador cerámico o una red R-C.

Aumentar la frecuencia de reloj supone disminuir el tiempo en que se ejecutan las instrucciones pero lleva aparejado un incremento del consumo de energía y de calor generado.

1.7.3. Recursos especiales

Cada fabricante oferta numerosas versiones de una arquitectura básica de microcontrolador. En algunas amplía las capacidades de las memorias, en otras incorpora nuevos recursos, en otras reduce las prestaciones al mínimo para aplicaciones muy simples, etc. La labor del diseñador es encontrar el modelo mínimo que satisfaga todos los requerimientos de su aplicación. De esta forma, minimizará el coste, el hardware y el software.

Los principales recursos específicos que incorporan los microcontroladores son:

- Temporizadores o Timers.
- Perro guardián o Watchdog.
- Protección ante fallo de alimentación o Brownout.
- Estado de reposo o de bajo consumo (Sleep mode).
- Conversor A/D (Analógico → Digital).
- Conversor D/A (Digital → Analógico).
- Comparador analógico.
- Modulador de anchura de impulsos o PWM (Pulse Wide Modulation).
- Puertas de E/S digitales.
- Puertas de comunicación.

A continuación se ve con un poco más de detalle cada uno de ellos.

Temporizadores o Timers

Se emplean para controlar periodos de tiempo (temporizadores) y para llevar la cuenta de acontecimientos que suceden en el exterior (contadores).

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 un aviso.

Cuando se desean contar acontecimientos que se materializan por cambios de nivel o flancos en alguna de las patitas del microcontrolador, el mencionado registro se va incrementando o decrementando al ritmo de dichos impulsos.

Perro guardián o Watchdog

Cuando el computador personal se bloquea por un fallo del software u otra causa, se pulsa el botón del reset y se reinicia el sistema. Pero un microcontrolador funciona sin el control de un supervisor y de forma continuada las 24 horas del día. El Perro Guardián consiste en un contador que, cuando llega al máximo, provoca un reset automáticamente en el sistema.

Se debe diseñar el programa de trabajo que controla la tarea de forma que resetee al Perro Guardián de vez en cuando antes de que provoque el reset. Si falla el programa o

se bloquea (si cae en bucle infinito), no se refrescará al Perro guardián y, al completar su temporización, provocará el reset del sistema.

Protección ante fallo de alimentación o Brownout

Se trata de un circuito que resetea al microcontrolador cuando el voltaje de alimentación (VDD) es inferior a un voltaje mínimo (brownout). Mientras el voltaje de alimentación sea inferior al de brownout el dispositivo se mantiene reseteado, comenzando a funcionar normalmente cuando sobrepasa dicho valor. Esto es muy útil para evitar datos erróneos por transiciones y ruidos en la línea de alimentación

Estado de reposo o de bajo consumo

Son abundantes las situaciones reales de trabajo en que el microcontrolador debe esperar, sin hacer nada, a que se produzca algún acontecimiento externo que le ponga de nuevo en funcionamiento. Para ahorrar energía, (factor clave en los aparatos portátiles), los microcontroladores disponen de una instrucción especial (SLEEP en los PIC), que les pasa al estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal y se congelan sus circuitos asociados, quedando sumido en un profundo sueño el microcontrolador. Al activarse una interrupción ocasionada por el acontecimiento esperado, el microcontrolador se despierta y reanuda su trabajo. Para hacernos una idea, esta función es parecida a la opción de Suspender en el menú para apagar el equipo (en aquellos PCs con administración avanzada de energía)

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 las patillas del circuito integrado.

Conversor D/A (CDA)

Transforma los datos digitales obtenidos del procesamiento del computador en su correspondiente señal analógica que saca al exterior por una de las patillas del chip. Existen muchos circuitos que trabajan con señales analógicas.

Comparador analógico

Algunos modelos de microcontroladores disponen internamente de un Amplificador Operacional que actúa como comparador entre una señal fija de referencia y otra variable que se aplica por una de las patitas de la cápsula. La salida del comparador proporciona un nivel lógico 1 o 0 según una señal sea mayor o menor que la otra.

También hay modelos de microcontroladores con un módulo de tensión de referencia que proporciona diversas tensiones de referencia que se pueden aplicar en los comparadores.

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 las patitas del encapsulado.

Puertos digitales de E/S

Todos los microcontroladores destinan parte de su patillaje a soportar líneas de E/S digitales. Por lo general, estas líneas se agrupan de ocho en ocho formando Puertos.

Las líneas digitales de las Puertos pueden configurarse como Entrada o como Salida cargando un 1 o un 0 en el bit correspondiente de un registro destinado a su configuración.

Puertas de comunicación

Con objeto de dotar al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, otros buses de microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras normas y protocolos. Algunos modelos disponen de recursos que permiten directamente esta tarea, entre los que destacan:

- UART, adaptador de comunicación serie asíncrona (Ej: Puerto Serie).
- USART, adaptador de comunicación serie síncrona y asíncrona.
- Puerta paralela esclava para poder conectarse con los buses de otros microprocesadores.
- USB (Universal Serial Bus), que es un moderno bus serie para los PC.
- Bus I2C.

- CAN (Controller Area Network), para permitir la adaptación con redes de conexionado multiplexado desarrollado conjuntamente por Bosch e Intel para el cableado de dispositivos en automóviles. En EE.UU. se usa el J185O.

Tanto el I2C en televisores, como el Bus CAN en automóviles, fueron diseñados para simplificar la circuitería que supone un bus paralelo de 8 líneas dentro de un televisor, así como para librar de la carga que supone una cantidad ingente de cables en un vehículo.

1.7.4. La familia PIC

En los últimos tiempos esta familia de microcontroladores ha revolucionado el mundo de las aplicaciones electrónicas. Tienen un don especial con el cual han fascinado a programadores y desarrolladores. Quizá sea por su facilidad de uso, programación e integración.

Es probable que en un futuro próximo otra familia de microcontroladores le arrebate ese don. Hay que tener en cuenta que para las aplicaciones más habituales (casi un 90 %) la elección de una versión adecuada de PIC es la mejor solución; sin embargo, dado su carácter general, otras familias de microcontroladores son más eficaces en aplicaciones específicas, especialmente si en ellas predomina una característica concreta, que puede estar muy desarrollada en otra familia.

Esta familia, desarrollada por la casa Microchip, se divide en cuatro gamas, gamas enana, baja, media y alta. Las principales diferencias entre estas gamas radica en el número de instrucciones y su longitud, el número de puertos y funciones, lo cual se refleja en el encapsulado, la complejidad interna y de programación, y en el número de aplicaciones.

En las próximas líneas se describen brevemente las cualidades de esta familia.

Gama baja o gama enana, de 8 patillas

Se trata de un grupo de PIC de reciente aparición que ha acaparado la atención del mercado. Su principal característica es su reducido tamaño, al disponer todos sus componentes de 8 patitas. Se alimentan con un voltaje de corriente continua comprendido entre 2,5 V y 5,5 V, y consumen menos de 2 mA cuando trabajan a 5 V y 4 MHz. El formato de sus instrucciones puede ser de 12 o de 14 bits y su repertorio es de 33 o 35 instrucciones, respectivamente. En la figura 1.32 se muestra el diagrama de conexionado de uno de estos PIC.

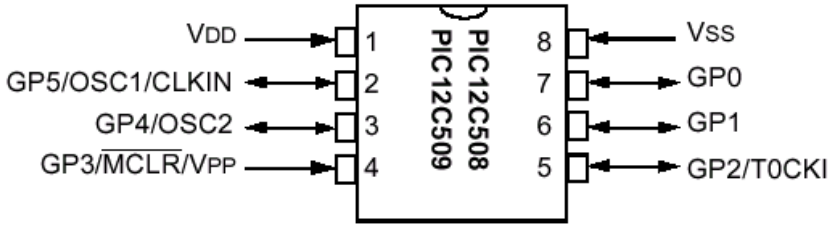


Figura 1.32: Diagrama de conexiones de los PIC12Cxxx de la gama enana.

Aunque los PIC enanos sólo tienen 8 patillas, pueden destinar hasta 6 como líneas de E/S para los periféricos porque disponen de un oscilador interno R-C, lo cual es una de su principales características.

Los modelos 12C5xx pertenecen a la gama baja, siendo el tamaño de las instrucciones de 12 bits; mientras que los 12C6xx son de la gama media y sus instrucciones tienen 14 bits. Los modelos 12F6xx poseen memoria Flash para el programa y EEPROM para los datos.

Gama baja o básica: PIC16C5X con instrucciones de 12 bits

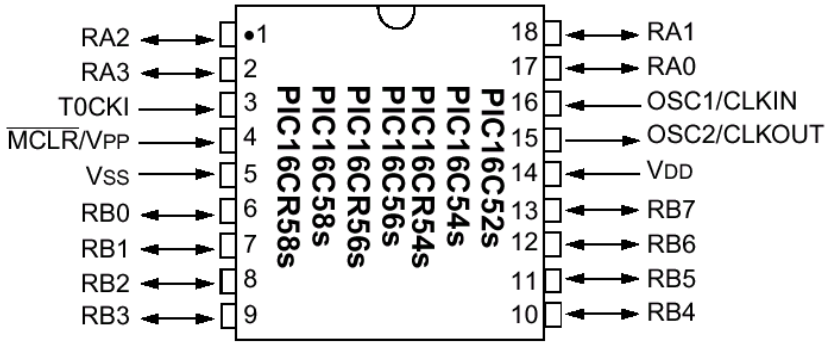


Figura 1.33: Diagrama de patillas de los PIC de la gama baja que responden a la nomenclatura PIC16C54/56.

Se trata de una serie de PIC de recursos limitados, pero con una de la mejores relaciones coste/prestaciones. Sus versiones están encapsuladas con 18 y 28 patitas y pueden alimentarse a partir de una tensión de 2,5 V, lo que les hace ideales en las aplicaciones que funcionan con pilas teniendo en cuenta su bajo consumo (menos de 2 mA a 5 V y 4 MHz). Tienen un repertorio de 33 instrucciones cuyo formato consta de 12 bits. No admiten ningún tipo de interrupción y la Pila sólo dispone de dos niveles. En la figura 1.33 se muestra el diagrama de conexionado de uno de estos PIC.

Al igual que todos los miembros de la familia PIC16/17, los componentes de la gama baja se caracterizan por poseer los siguientes recursos: Sistema Power On Reset, Perro guardián (Watchdog o WDT), Código de protección, Sep, etc. Sus principales desventajas o limitaciones son que la pila sólo tiene dos niveles y que no admiten interrupciones. En la figura 1.34 se presentan las principales características de los modelos de esta subfamilia.

MODELO	MEMORIA PROGRAMA (x12 BITS) EPROM ROM	MEMORIA DATOS (bytes)	FRECUENCIA MÁXIMA	LÍNEAS E/S	TEMPORIZADORES	PATITAS
PIC16C52	384	25	4 MHz	4	TMR0 + WDT	18
PIC16C54	512	25	20 MHz	12	TMR0 + WDT	18
PIC16C54A	512	25	20 MHz	12	TMR0 + WDT	18
PIC16CR54A	512	25	20 MHz	12	TMR0 + WDT	18
PIC16C55	512	24	20 MHz	20	TMR0 + WDT	28
PIC16C56	1 K	25	20 MHz	12	TMR0 + WDT	18
PIC16C57	2 K	72	20 MHz	20	TMR0 + WDT	28
PIC16CR57B	2 K	72	20 MHz	20	TMR0 + WDT	28
PIC16C58A	2 K	73	20 MHz	12	TMR0 + WDT	18
PIC16CR58A	2 K	73	20 MHz	12	TMR0 + WDT	18

Figura 1.34: Características de los modelos PIC16C(R)5X de la gama baja

Gama media. PIC16CXXX con instrucciones de 14 bits

Es la gama más variada y completa de los PIC. Abarca modelos con encapsulado desde 18 patitas hasta 68, cubriendo varias opciones que integran abundantes periféricos. Dentro de esta gama se halla el fabuloso PIC16F84 y sus variantes.

Encuadrado en la gama media también se halla la versión PIC14C000, que soporta el diseño de controladores inteligentes para cargadores de baterías, pilas pequeñas, fuentes de alimentación ininterrumpibles y cualquier sistema de adquisición y procesamiento de señales que requiera gestión de la energía de alimentación. Los PIC 14C000 admiten cualquier tecnología de las baterías como Li-Ion, NiMH, NiCd, Ph y Zinc.

El temporizador TMR1 que hay en esta gama tiene un circuito oscilador que puede trabajar asíncronamente y que puede incrementarse aunque el microcontrolador se halle en el modo de reposo (sleep), posibilitando la implementación de un reloj en tiempo real. Las líneas de E/S presentan una carga pull-up activada por software.

Gama alta: PIC17CXXX con instrucciones de 16 bits

Se alcanzan las 58 instrucciones de 16 bits en el repertorio y sus modelos disponen de un sistema de gestión de interrupciones vectorizadas muy potente. También incluyen variados controladores de periféricos, puertas de comunicación serie y paralelo con elementos externos, un multiplicador hardware de gran velocidad y mayores capacidades de memoria, que alcanza los 8 k palabras en la memoria de instrucciones y 454 bytes en la memoria de datos.

Quizás la característica más destacable de los componentes de esta gama es su arquitectura abierta, que consiste en la posibilidad de ampliación del microcontrolador con elementos externos. Para este fin, las patitas sacan al exterior las líneas de los buses de datos, direcciones y control, a las que se conectan memorias o controladores de periféricos. Esta facultad obliga a estos componentes a tener un elevado número de patitas comprendido entre 40 y 44. Esta filosofía de construcción del sistema es la que se empleaba en los microprocesadores y no suele ser una práctica habitual cuando se emplean microcontroladores.

1.7.5. El lenguaje ensamblador

Identificación de elementos del programa en ensamblador

Cada instrucción de un PIC está formada por una palabra de 14 bits y a su vez está dividida e un tipo de código denominado OPCODE, que especifica el tipo de instrucción, y uno o más operandos que además especifican la operación de la instrucción. Todo ello forma un mnemónico o instrucción.

Estos OPCODES o instrucciones se componen de los siguientes tipos de datos. Estos son abreviaturas usados en el lenguaje ensamblador. Sólo los 6 primeros componen estos OPCODES.

Las instrucciones se componen del código de la instrucción y de algunos operandos, que son los datos con los que la instrucción en sí, deberá hacer operaciones dentro del microcontrolador. Estos operandos son:

- **f** Registro de direcciones de registros(file register address) (0x00-0x7F)
- **w** Registro de trabajo (Working Register)
- **b** Dirección de un bit dentro de un registro de 8 bits (0-7)
- **l ó k** Literal

- **d** Bit de destino
- **x** Los bits que estén representados por este tipo de dato no tienen ninguna función y su valor lo define el compilador

Éstos vienen explicados detalladamente a continuación.

f (file register)

Este carácter se usa para definir registros de cualquier tipo. Cualquier instrucción que contenga este campo, contendrá la dirección de un registro, no su contenido. Un registro puede variar entre las direcciones 00h y 7Fh. En el caso de los registros especiales en vez de la dirección podremos poner directamente el nombre del registro.

Ej:

en vez de:

```
BSF     STATUS,5
```

se pone:

```
BSF     STATUS,RP0
```

l o k (literal)

Este valor será almacenado en la propia instrucción en tiempo de ensamblado, esto significa que son los valores que se introducen en las instrucciones para que trabaje con ellos (independientemente de los datos que se puedan almacenar o contener en la EEPROM de datos). El valor que se puede introducir dentro de un literal está comprendido entre 0 y 255, ya q es el máximo que puede representar un byte. No se debe olvidar que este valor debe ser introducido en hexadecimal. Siendo así, el valor que puede almacenar l está comprendido entre 0 y FF.

d (destiny bit)

Donde aparezca esta letra, es necesario especificar donde se alamacenará el resultado de una instrucción, en w o en un registro. Puesto que esto no es un lenguaje de alto nivel, no se puede almacenar el resultado de una operación sobre una tercera variable o registro, así que este deberá ser almacenado en el registro origen (sobrescribiéndose), o en el acumulador. Esto se define a través de dos valores:

- **1**: El resultado se almacenará en f

- **0**: El resultado se almacenará en W

label o etiquetas

Las etiquetas se sitúan a la izquierda de las instrucciones y sirven para agrupar fragmentos de código. Estos fragmentos pueden ser de dos tipos:

- El primer tipo no es un fragmento tal cual, si no que es un punto del programa al que podremos saltar de manera incondicional a través de la instrucción adecuada
- El segundo tipo es denominado subrutina. Éste empieza con una etiqueta y acaba con la instrucción RETURN o RETLW

Directrices del ensamblador

Las instrucciones más importantes que podemos manejar son las que proporciona el fabricante de un microprocesador para su producto. Pero existen otras genéricas para gran cantidad de microcontroladores, que no son para el manejo del PIC, si no del ensamblador. Estos comandos generalmente se usan para simplificar la tarea de programar, y reciben el nombre de directrices.

A continuación se exponen las más relevantes.

Directriz EQU

El nombre de esta instrucción viene de la palabra *equal*, o lo que es lo mismo, "igual". Sirve para igualar la posición de cualquier registro a un nombre personalizado que le hayamos dado nosotros. Si el nombre es más descriptivo que una simple dirección, la tarea de programar se hará mucho más sencilla. También podemos asignar un nombre a una instrucción que repitamos varias veces a lo largo del algoritmo, de manera que sea mucho más sencillo el tener que programarlo. A estos nombre que asignamos mediante esta directriz se les denomina constantes, ya que el registro al que apuntan no variará durante el programa.

Ej:

```
temp    EQU   12
DATO    EQU   22
Bank_1  EQU   BSF  STATUS,RP0
```

Esto es lo que se ha hecho para crear las abreviaturas de los registros antes expuestas. Estas están contenidas en un fichero que se incluye al principio de código y del cual se hablará más tarde.

No siempre es necesario que con esta directriz se igualen posiciones de memoria a las etiquetas, ya que podremos poner nombres a datos. Por ejemplo, se puede calcular la frecuencia de máquina a partir de la frecuencia de reloj con la finalidad de emplearla para hacer otros cálculos de la manera que se describe a continuación:

```
clockrate EQU .4000000      ; frecuencia del cristal

fclk      EQU  clockrate/4 ; frecuencia del reloj interno
```

Además de esto, se pueden igualar a las etiquetas cualquier otro tipo de valores usados, como, por ejemplo, el cero y el 1 en el bit de destino:

```
W EQU 0
F EQU 1
```

Generalmente este ejemplo no tendrá que realizarse, siempre que se incluya el fichero correspondiente al PIC con el que se esté trabajando.

Directriz ORG

Esta directriz dice al ensamblador a partir de que posición de memoria se situarán las siguientes instrucciones. Un ejemplo de su uso es el siguiente:

Ej:

Inicia el programa en la posición cero;

```
    ORG    0x00
```

se salta el vector de interrupción;

```
    ORG  0x00     ; El programa comienza en la direccion 0 y
    GOTO inicio   ; salta a la direccion 5
```

el vector de interrupción está situado en la posición 4;

```
    ORG    0x05
```

inicio xxx...

Directriz INCLUDE

Esta instrucción indica qué archivos deberán tomarse en cuenta a la hora de compilar el código. Normalmente se usa para incluir el archivo de PIC que el ensamblador tiene entre sus archivos, con el cual el compilador será capaz de reconocer todos los registros especiales y sus bits. Su uso recuerda al #include del lenguaje C. Esta línea debe colocarse al principio, y tiene la siguiente sintaxis:

```
include "P16F628.INC"
```

En ciertas ocasiones gran cantidad errores son debidos a que el nombre del archivo puesto entre comillas no coincide con el que tiene el ensamblador en sus registros internos. Para suprimirlos basta con poner en la cabecera el nombre de este archivo.

Podemos crear archivos con funciones, definiciones y subrutinas que usemos a menudo en nuestro código, y que para evitarnos tener que copiarlas cada vez, bastará con incluir el archivo.

El archivo P16F628 contiene, como ya se ha comentado anteriormente, las definiciones de los registros, bits y bits de configuración (también llamados fuses). Su inclusión en nuestro programa no es obligatoria, y podemos suprimir esta directriz, pero a cambio tendremos que redefinir los nombres de los registros que usemos o bien llamarlos por su posición de memoria.

Directriz LIST

Este comando sirve para que el compilador tenga en cuenta sobre qué procesador se está trabajando. Este comando debe estar en todo proyecto, situado debajo del *include*, con la siguiente sintaxis.

```
LIST    P=16F628
```

Directriz END

Al igual que las dos anteriores, ésta debe ir incluida una sola vez en todo el programa. En concreto, ésta debe situarse al final, para indicar al ensamblador que el programa ha finalizado. Siempre debe estar presente, aunque el flujo de nuestro programa acabe en un bucle.

Directriz #DEFINE

Define se usa para crear pequeñas macros. Con estas macros podremos poner nombres a pequeños fragmentos de código que nos facilitarán la realización y comprensión del algoritmo. Por ejemplo, podremos poner nombres a bits.

```
#define CERO STATUS,2
```

Así, en vez de tener que llamar al bit por un número y un registro, podremos usar directamente la palabra CERO. Hemos de tener en cuenta que la definición del ejemplo ya está hecha en el archivo P16F84.inc. Otro ejemplo muy práctico es el de poner nombre a un fragmento de código usado frecuentemente. Este fragmento de código, puede ser por ejemplo, el que conmuta entre los dos bancos.

```
    BSF OPTION,RP0
    BCF OPTION,RP0
```

Como cambiamos varias veces de banco a lo largo de un algoritmo, puede resultar más práctico ponerle un nombre:

```
#define BANCO1  BSF  OPTION,RP0
#define BANCO0  BCF  OPTION,RP0
```

De este modo bastará con poner BANCO1 o BANCO0 para conmutar entre los dos bancos de memoria. Una cosa a tener en cuenta es que con la directriz *INCLUDE*, podemos prescindir del carácter almohadilla (#), pero en el caso de la directriz *DEFINE*, no.

Directriz TITLE

Esta directriz no sirve de mucho, pero será útil para aquellos que quieran que el compilador tenga en cuenta el título que le ha puesto a su código. Tiene la siguiente sintaxis:

```
    TITLE           Nombre del codigo
```

Este nombre aparecerá en los archivos .lst (listados) que cree el compilador.

Directriz MACRO

A diferencia de la instrucción *define*, esta directriz permite crear macros más extensas que *#define*, que nos evitarán tener que ejecutar reiteradamente fragmentos de código idénticos. Cuando una macro es invocada, ésta es copiada por el ensamblador en el lugar de la invocación dentro del código fuente. La macro se declara con la directriz *MACRO*, y termina con la directriz *ENDM*.

Ej:

```
EN_STROBE
        MACRO
        SET_EN
        nop
        CLEAR_EN
        ENDM
```

ahora lo invocamos;

```
        bcf     PORTB,LCD_DB7

        EN_STROBE

        movlw   2
        call    msRetardo               ; Espera 2ms.
```

Estructura de un programa en ensamblador

A continuación se estudia la estructura básica de un algoritmo en código ensamblador. Esta estructura contiene los siguientes elementos o partes que deben ser codificadas:

- Comentario descriptivo del programa (opcional, pero recomendable)
- Definir el microcontrolador que se usará (con las directrices *LIST* e *INCLUDE*).
- Introducir las opciones de compilación (opcional)
- Establecer las constantes que se usarán (con la directriz *EQU*).
- Reservar espacios de memoria (directriz *RES*) (si es necesario)
- Configurar los puertos
- Desarrollar el programa
- Poner comentarios

Esa estructura se muestra en la figura 1.35.

Vista la estructura general, se pasa a ver la posición de los elementos del código por columnas. Estas se dividen en cuatro:

- Columna 1: Etiqueta. Las etiquetas se rigen por las siguientes normas:
    - Debe situarse en la primera columna

```
;*************************************************
; ESTE PROGRAMA ENCIENDE DOS LEDS DESDE EL CENTRO
; DEL PUERTO B Y LOS ROTA HASTA LOS EXTREMOS CADA
; VEZ QUE EL MICRO TIENE UN CERO EN EL PIN RA4
;*************************************************

        LIST P=16F84

STATUS  EQU     3
PORTA   EQU     5
TRISA   EQU     5
PORTB   EQU     6
TRISB   EQU     6

RPO     EQU     5

        ORG 0XC
CONTA1  RES     1
CONTA2  RES     1
CONTA3  RES     1

        ORG 0X00
INICIO  BSF     STATUS,RPO      ; ENTRA EN EL BANCO 1
        MOVLW   0X1F            ; MUEVE 1Fh A W
        MOVWF   TRISA           ; CONFIGURA EL PUERTO A COMO ENTRADA
        MOVLW   0X00            ; MUEVE 00 A W
        MOVWF   TRISB           ; CONFIGURA EL PUERTO B COMO SALIDA
        BCF     STATUS,RPO      ; SALE DEL BANCO 1
        CLRF    PORTB           ; LIMPIA EL PUERTO B

OTRA    BTFSC   PORTA,4         ; PREGUNTA SI EL PIN 4 DEL PUERTO A ES
CERO
        GOTO    OTRA            ; SALTA A LA ETIQUETA OTRA
        MOVLW   0X18            ; MUEVE 18h A W
        MOVWF   PORTB           ; CARGA W EN EL PUERTO B
        CALL    RETARDO         ; LLAMA A LA ETIQUETA RETARDO
        BTFSC   PORTA,4         ; PREGUNTA SI EL PIN 4 DEL PUERTO A ES
CERO
        GOTO    OTRA1           ; SALTA A LA ETIQUETA OTRA1
```


Figura 1.35: Esquema de un programa

- Debe contener únicamente caracteres alfanuméricos
- El máximo de caracteres es de 31

- Columna 2: Operación. En esta columna se situarán las instrucciones
- Columna 3: Son los registros (f, l o k , b y w) donde se almacenarán los resultados y con los que se operará

- Columna 4: Comentario. Aquí se situará cualquier comentario personalizado que deseemos. Estos son útiles para saber qué hace un programa sin tener que descifrar el código entero. El compilador (ensamblador) ignorará todo texto más allá del carácter punto y coma ';'. Estos comentarios generalmente se sitúan en la cuarta columna para describir la acción de una línea de código, pero pueden situarse en cualquier parte de programa para describir cualquier otro evento, siempre que estén después del carácter ';' (semicolon en inglés).

Se aprecia más detenidamente cuál es su posición en la figura 1.36

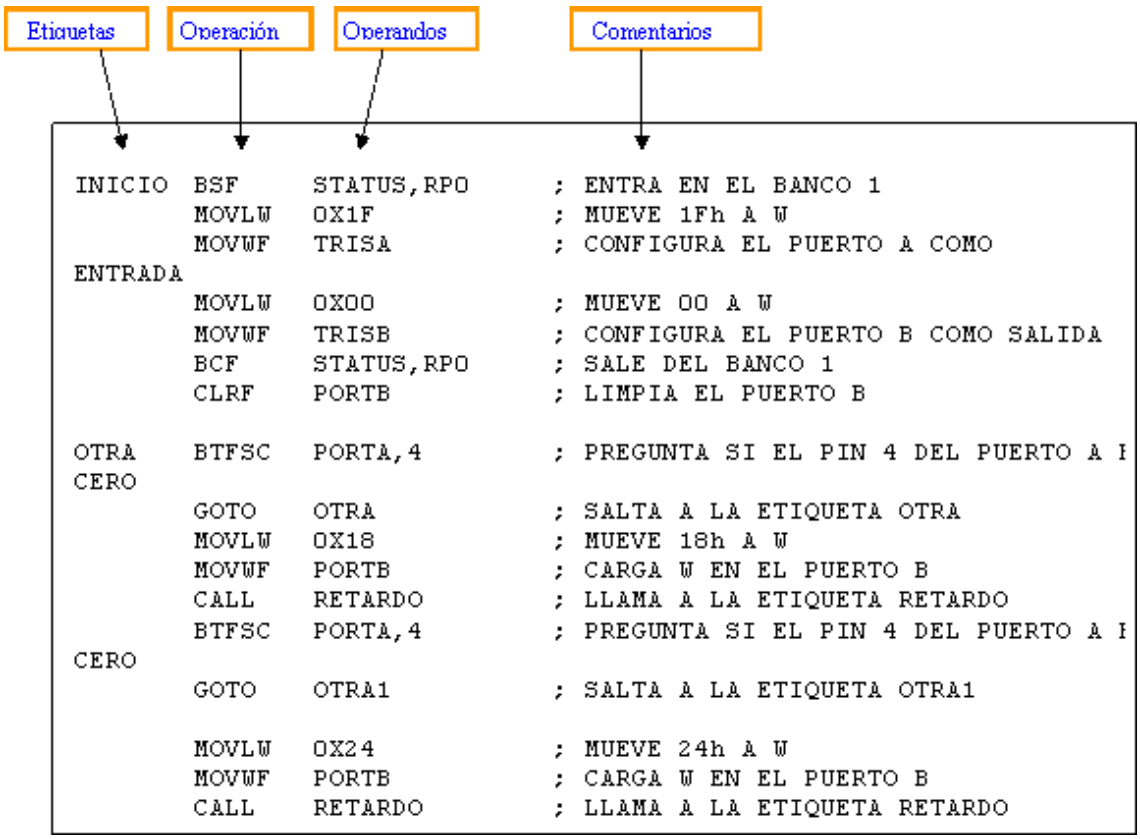


Figura 1.36: Esquema de un programa por columnas

Normalmente las columnas son separadas por una tabulación. El espacio mínimo entre dos columnas es de un carácter, que puede ser un espacio en vez de una tabulación.

1.7.6. El PIC16F628

Descripción general del PIC16F628

El PIC16F628 de Microchip es un potente microcontrolador CMOS FLASH de 8 bits de arquitectura RISC capaz de operar con frecuencias de reloj hasta de 20 MHz (ciclos de instrucción de apenas 200 ns), fácil de programar (sólo 35 instrucciones para aprender) y disponible en cápsulas DIP y SOIC de 18 pines.

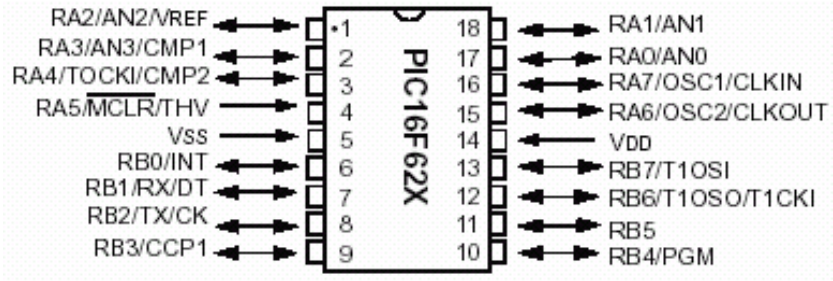


Figura 1.37: Presentación en cápsula DIP-18 y distribución de pines del PIC16f628

Posee internamente un oscilador de 4 MHz y un circuito de P.O.R. (Power-On Reset) que eliminan la necesidad de componentes externos y expanden a 16 el número de pines que pueden ser utilizados como líneas I/O de propósito general.

Adicionalmente, el PIC16F628 proporciona una memoria de datos EEPROM de 128x8 , una memoria de programa FLASH de 2048x14, una memoria de datos RAM de propósito general de 224x8, un módulo CCP (captura/comparación/PWM), un USART, 2 comparadores analógicos, una referencia de voltaje programable y tres temporizadores. Estas y otras características lo hacen ideal en aplicaciones automotrices, industriales, y de electrónica de consumo, así como en equipos e instrumentos programables de todo tipo.

Descripción de los pines

La distribución de pines del PIC16F628, figura 1.37, es idéntica a la del PIC16F627, excepto que este último posee una memoria de programa FLASH de 1024x14. También es idéntica a la del PIC16F84, con la diferencia de que en el PIC16F628 se puede disponer de tres líneas I/O adicionales para el puerto A (RA7, RA6, RA5) y algunos pines I/O están multiplexados con una función alterna para los diversos dispositivos periféricos que soporta el chip. Por ejemplo, RB1 funciona también como la línea de recepción del USART (RX). En general, cuando un periférico está habilitado, esa línea no puede ser utilizada como un pin I/O de propósito general.

Organización de la memoria

El PIC16F628 posee un contador de programa de 13 bits, capaz de direccionar un espacio de memoria de 8Kx14. Sin embargo, únicamente los primeros 2Kx14, desde 0000h hasta 07FFh, están implementados. Los vectores de reset e interrupción están en las direcciones 0000h y 0004h, respectivamente. La pila (stack) es de 8 niveles, lo cual significa que puede soportar hasta 8 direcciones de retorno de subrutina. El PIC16F627 y el PIC16F84 tienen la misma organización, excepto que únicamente están implementados los primeros 1Kx14, desde 0000h hasta 03FFh.

El PIC16F628 y el PIC16F627 poseen un espacio de memoria RAM de datos de 512x8, dividido en 4 bancos de 128 bytes cada uno (ver figura 1.38). Sin embargo, sólo están implementados 330 bytes, correspondiendo 224 al área de los registros de propósito general (GPR) y 36 al área de los registros de función especial (SFR)

Los restantes 70 bytes implementados son espejos de algunos SFR de uso frecuente, así como de los últimos 16 GPR del banco 0. Por ejemplo, las posiciones 0Bh, 8Bh, 10Bh y 18Bh corresponden al registro INTCON, de modo que una operación hecha en cualquiera de ellos, se refleja automáticamente en los otros. Se dice, entonces, que las posiciones 8Bh, 10Bh y 18Bh están mapeadas en la posición 0Bh. Esta característica agiliza el acceso a estos registros, puesto que no siempre es necesario especificar el banco donde se encuentran. La selección del banco de ubicación de un SFR o un GPR particular se hace mediante los bits 6 (RP1) y 5 (RP0) del registro STATUS.

Después análisis general de los PIC en la sección 1.7.6 del apartado de introducción vamos a ver una completa descripción del PIC16F628, elemento que se encarga de poner la inteligencia al circuito.

1.7.7. Formatos del PIC16F628

El PIC16F628 puede presentarse en varios formatos:

- En encapsulado DIL (Dual In Line o Doble En Línea), es el encapsulado tradicional, de toda la vida, y también el más grande y manejable.
- En encapsulado SOIC, para montaje superficial SMD, una tecnología de mayor integración que ocupa muy poco espacio, pero con un proceso de soldadura más difícil.

El formato empleado en los montajes es el SMD, de forma que el tamaño de las placas resulta bastante ajustado.

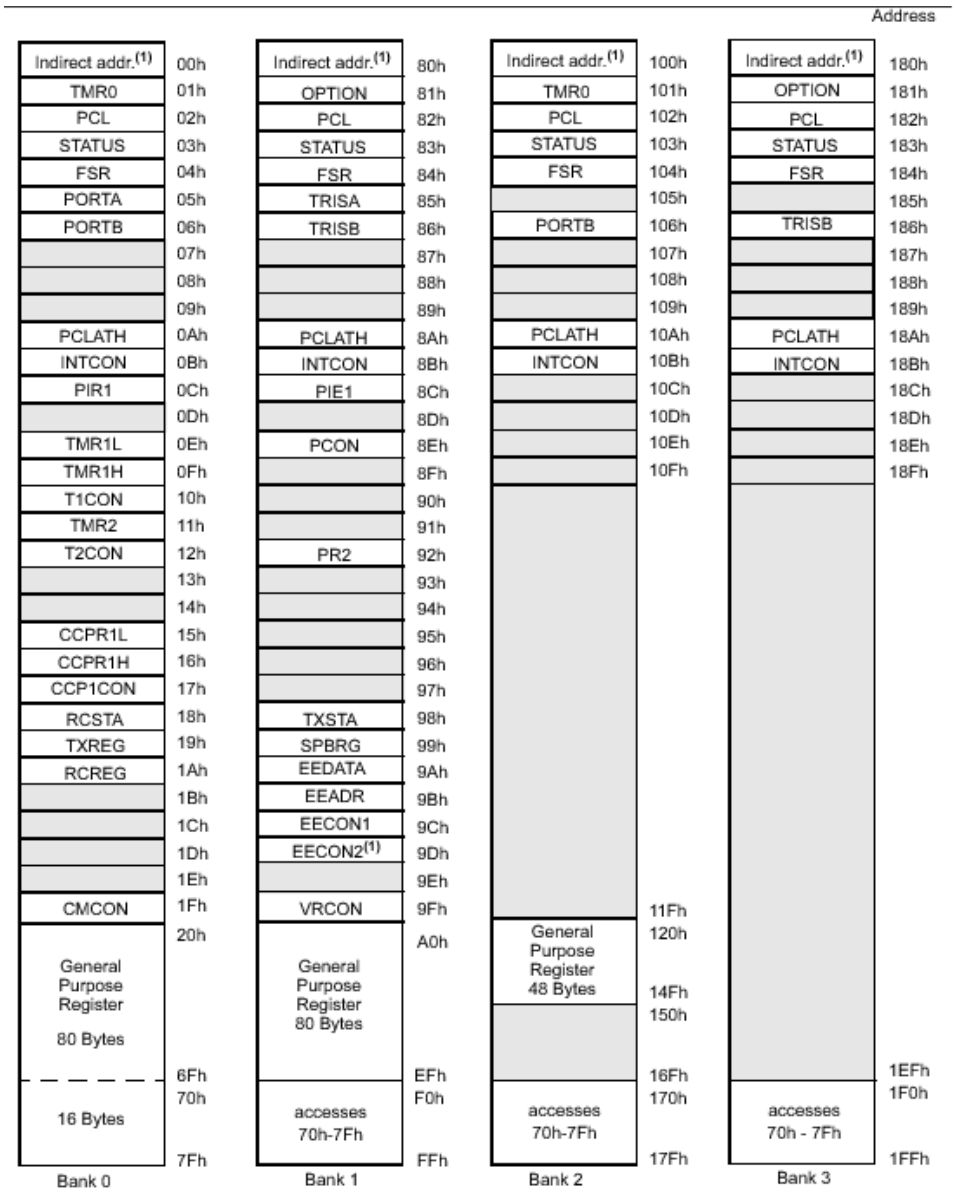


Figura 1.38: RAM de datos del PIC16f628

1.7.8. Palabra de configuración

El PIC16f628 posee un área especial para la palabra de configuración en la memoria de programa. Esta palabra está mapeada en memoria en la localización 2007h. Esta dirección está fuera del espacio de memoria de programa de usuario. Esta palabra sólo puede ser accedida durante la programación.

Los requerimientos de sistema del PIC asignados mediante la palabra de configuración son los que siguen:

- **CP1,CP0**: Bits de protección de código.
  - **11**: Protección de código de la memoria de programa deshabilitada
  - **10**: Protección de código de 0400h-07FFh
  - **01**: Protección de código de 0200h-07FFh
  - **00**: Protección de código de 0000h-07FFh
- **CPD**: Bit de protección de código de datos
  - **1**: Protección de código de la memoria de datos deshabilitada
  - **0**: Protección de código de la memoria de datos
- **LVP**: Habilitación de la programación de bajo voltaje
  - **1**: El pin RB4/PGM tiene la función PGM; habilitación de la programación de bajo voltaje
  - **0**: RB4/PGM es una E/S digital; se programa a través de MCLR HV
- **BODEN**: Bit de habilitación de reset si se detecta una caída de tensión por debajo de los 4 voltios
  - **1**: Reset BOD habilitado
  - **0**: Reset BOD deshabilitado
- **MCLRE**: Selección de la función del pin RA5/MCLR
  - **1**: La función del pin RA5/MCLR es MCLR
  - **0**: La función del pin RA5/MCLR es la de entrada digital, MCLR se conecta internamente a VDD
- **PWRTE**: Bit de habilitación de retardo para el correcto arranque
  - **1**: PWRT deshabilitado
  - **0**: PWRT habilitado
- **WDTE**: Bit de habilitación del temporizador perro guardián
  - **1**: WDT habilitado
  - **0**: WDT deshabilitado

- **FOSC2,FOSC1,FOSC0**: Selección del tipo de oscilador
  - **111**: ER(Resistencia externa) oscilador: CLKOUT en el pin RA6/OSC2/CLKOUT, Resistencia en el pin RA7/OSC1/CLKIN
  - **110**: ER(Resistencia externa) oscilador: Función de E/S en RA6/OSC2/CLKOUT, Resistencia en el pin RA7/OSC1/CLKIN
  - **101**: INTRC(Oscilador interno de 4MHz) oscilador: CLKOUT en RA6/OSC2/CLK OUT, E/S en RA7/OSC1/CLKIN
  - **100**: INTRC(Oscilador interno de 4MHz) oscilador: E/S en RA6/OSC2/CLKOUT, E/S en RA7/OSC1/CLKIN
  - **011**: EC(Reloj externo): E/S en el pin RA6/OSC2/CLKOUT, CLKIN en RA7/OSC1/CLKIN
  - **010**: Oscilador HS: Cristal de alta velocidad/resonador conectado a RA6/OSC2/CL KOUT y RA7/OSC1/CLKIN
  - **001**: Oscilador XT: Cristal/resonador conectado RA6/OSC2/CLKOUT y RA7/OS C1/CLKIN
  - **000**: Oscilador LP: Cristal de baja potencia conectado a RA6/OSC2/CLKOUT y RA7/OSC1/CLKIN

1.7.9. Osciladores

Los osciladores se agrupan en cuatro tipos.

Elemento oscilador externo. (FOSC2-0: 000/001/010)

El circuito de la figura 1.39 es el que generalmente se usa. Xtal es un elemento oscilador de cristal. En lugar de utilizar este esquema también se emplean resonadores que incluyen en el conjunto el cristal y los condensadores.

Reloj externo (FOSC2-0: 011)

En este caso la señal de reloj del PIC es la salida del circuito oscilador externo. De esta forma RA6/OSC2/CLKOUT (pin15) se puede usar como un puerto de E/S.

Circuito oscilador interno (FOSC2-0: 100/101)

La frecuencia de la señal de reloj interna está fijada en 4Mhz. Este esquema se usa cuando es necesario emplear los pines RA6 y RA7 como puertos de E/S.

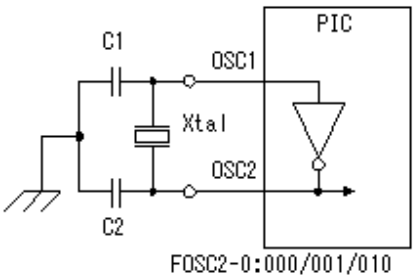


Figura 1.39: Circuito oscilador basado en cristal de cuarzo

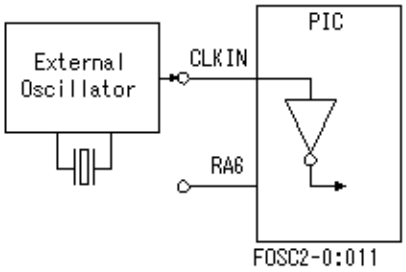


Figura 1.40: Reloj externo

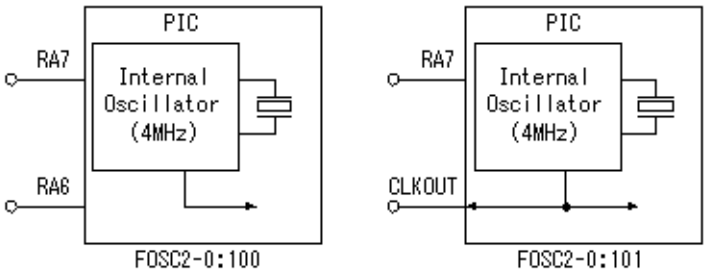


Figura 1.41: Reloj interno

Ajuste de la frecuencia del oscilador interno con una resistencia externa (FOSC2-0: 110/111)

En el esquema de la figura 1.42 se conecta una resistencia en la patilla OSC del PIC para realizar el ajuste de la frecuencia interna de reloj. Por lo tanto RA7 no se puede usar como un puerto de E/S.

En la tabla 1.4 se muestran una serie de valores de resistencias, asociadas a las frecuencias de reloj que se consiguen.

Resistencia	Frecuencia
0	10.4MHz
1K	10MHz
10K	7.4MHz
20K	5.3MHz
47K	3MHz
100K	1.6MHz
220K	800kHz
470K	300kHz
1M	200kHz

Cuadro 1.4: Resistencias y frecuencias de reloj asociadas

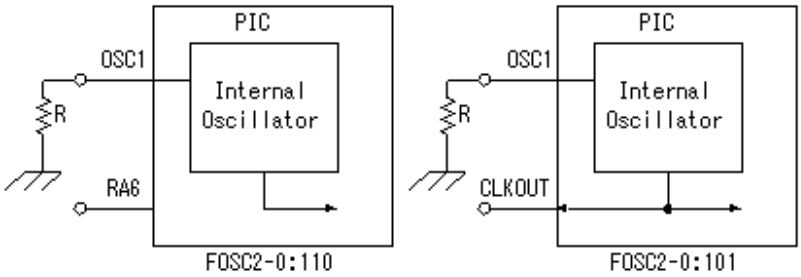


Figura 1.42: Resistencia externa para el ajuste del reloj interno

1.7.10. Registros de función específica (SFR)

STATUS (Registro de estado)

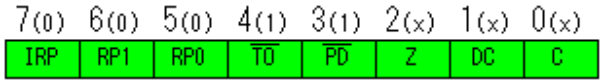


Figura 1.43: Registro STATUS

- **IRP**: Los bits IRP y MSB del registro FSR especifican el banco de RAM si el direccionamiento es indirecto, mientras que los 7 bits menos significativos se encargan de determinar la dirección dentro de ese banco.

- **RP1,RP0**: En el caso de direccionamiento directo estos bits permiten elegir el banco de memoria.

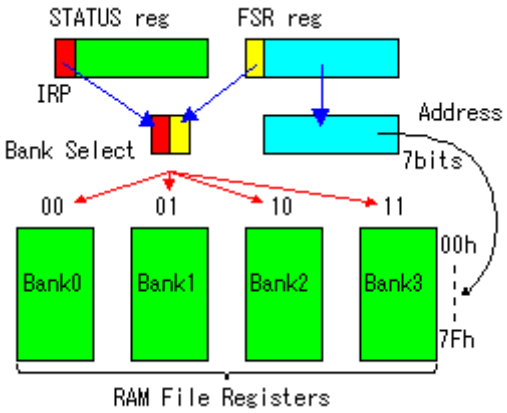


Figura 1.44: Función del bit IRP

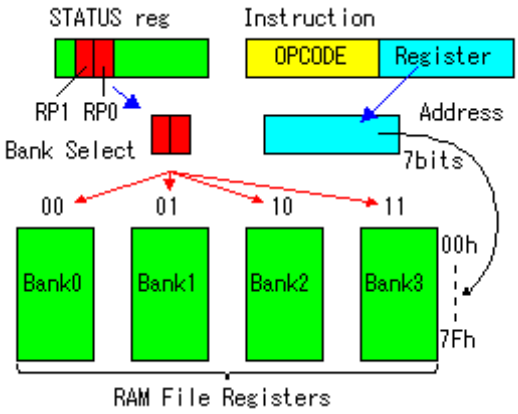


Figura 1.45: Uso de los bits RP1,RP0

- **TO(inv)**: Bit de condición de expiración de la cuenta del WDT. (No se puede escribir).
  - 1: Después de un arranque, una instrucción CLRWDT o una instrucción SLEEP.
  - 0: El WDT acabó su cuenta
- **PD(inv)**: Indica condición de modo consumo. (No se puede escribir).
  - 1: Después de un arranque o de una instrucción CLRWDT.
  - 0: Después de la instrucción SLEEP.
- **Z**: Bit de cero.

RP1	RP0	Banco
0	0	0
0	1	1
1	0	2
1	1	3

Cuadro 1.5: Selección de banco en el direccionamiento directo

- 1: El resultado de una operación lógica o aritmética fué cero.
- 0: El resultado de una operación lógica o aritmética no fué cero.

- **DC**: Bit de carry de dígito (para ADDWF y ADDLW).
  - 1: Se produjo un carry del 4 bit de menor orden del resultado.
  - 0: No se produjo un carry del 4 bit de menor orden del resultado.
- **C**: Bit de carry (para ADDWF y ADDLW).
  - 1: Se produjo carry en el bit más significativo del resultado.
  - 0: No se produjo carry en el bit más significativo del resultado.

PCLATH (Parte alta del contador de programa)

Este registro se encarga de almacenar los 5 bits más significativos del contador de programa.

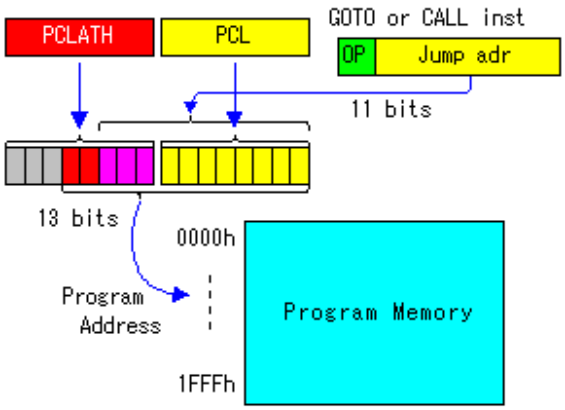


Figura 1.46: Registro PCLATH

PCL (Parte baja del contador de programa)

PCL guarda los 8 bits de menor peso del contador de programa.

En el caso de las intrucciones *GOTO* o *CALL*, el operando de la instrucción determina los 11 bits más bajos de la dirección de programa. En esta situación sólo 2 bits, los más significativos de esta dirección, vienen especificados por PCLATH.

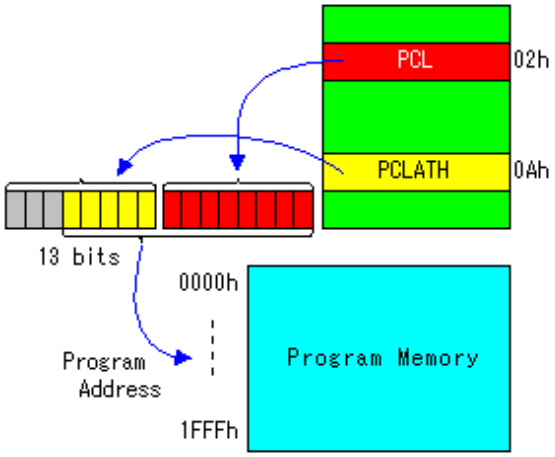


Figura 1.47: Registro PCL

FSR (Registro de direccionamiento indirecto)

FSR es el registro que almacena la dirección a la que se desea acceder mediante el método de direccionamiento indirecto.

El bit más alto de FSR es el bit más bajo de la especificación del banco de memoria, siendo necesarios los 7 bits restantes para apuntar a una dirección concreta dentro del banco. El bit más significativo de la especificación del banco viene determinada por IRP; este bit pertenece al registro STATUS (ver sección 1.7.10).

INDF

INDF es el encargado de personalizar la dirección de memoria a la que se accede después de dar el valor correspondiente a FSR e IRP para realizar el direccionamiento indirecto. No es un registro físico.

El contenido de INDF se convierte en el mismo que el del registro de la dirección establecida por FSR. El valor del byte presente en esta dirección puede ser reescrito reescribiendo, o leído leyendo, el contenido de este registro.

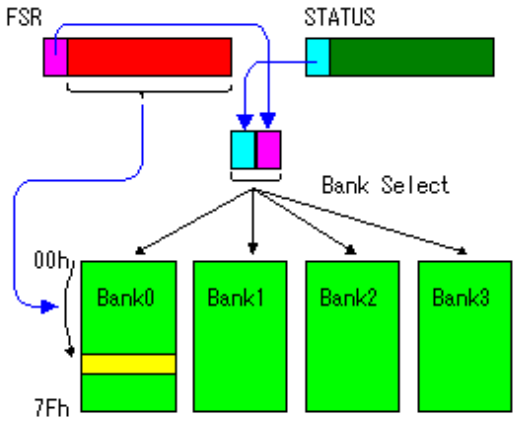


Figura 1.48: Registro FSR

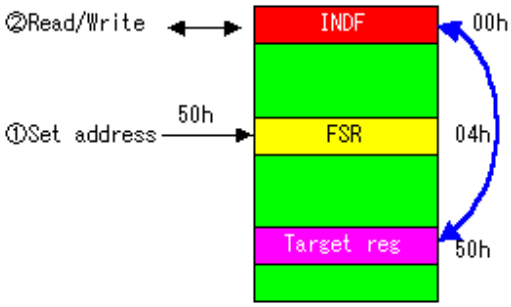


Figura 1.49: Registro INDF

PORTn (registro de puerto)/TRISn (registro de configuración de puerto)

El registro PORTn almacena el dato de salida o de entrada de cada bit dependiendo de la configuración que tenga cada patilla. A través de TRISn se configura cada uno de los bits de los puertos, de modo que se selecciona si cada uno de ellos es entrada o por el contrario salida. El PIC16F628 posee dos puertos de E/S: PORTA y PORTB (TRISA es el registro de configuración del puerto A y TRISB el registro de configuración del puerto B).

Para que una patilla del PIC sea salida se le da el valor '0' al bit correspondiente del registro TRISn. En caso contrario, cuando se desea que el pin sea una entrada, se le proporciona el valor '1'.

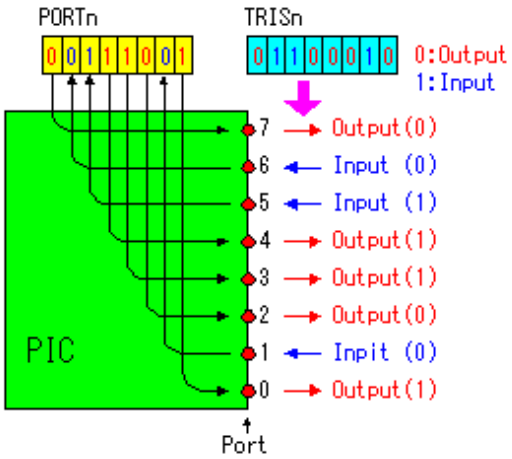


Figura 1.50: Registro PORTn

PIR1 (Registro de interrupción de periféricos)

- **EEIF** (Flag de interrupción de operación de escritura de la EEPROM).
  - 1: La operación de escritura se completó (debe ser borrado por software)
  - 0: La operación de escritura no se completó o aún no ha comenzado
- **CMIF** (Flag de interrupción de comparación)
  - 1: La salida del comparador ha cambiado
  - 0: La salida del comparador no ha cambiado
- **RCIF** (Flag de interrupción de recepción del USART)
  - 1: El búffer de recepción está lleno
  - 0: El búffer de recepción está vacío
- **TXIF** (Flag de interrupción de transmisión del USART)
  - 1: El búffer de transmisión está lleno
  - 0: El búffer de transmisión está vacío
- **CCP1IF** (Flag de interrupción de CCP1)
  - Modo captura

    - 1: Se produjo una captura del registro TMR1 (debe ser borrado por software)
    - 0: No hubo captura del registro TMR1
  - Modo comparación
    - 1: Ocurrió una coincidencia en la comparación con TMR1 (debe ser borrado por software)
    - 0: No ocurrió una coincidencia en la comparación con TMR1

- **TMR2IF** (Flag de interrupción por coincidencia entre TMR2 y PR2)
  - 1: Coincidencia de TMR2 y PR2 (debe ser borrado por software)
  - 0: No hubo coincidencia de TMR2 y PR2
- **TMR1IF** (Flag de interrupción por desbordamiento de TMR1)
  - TMR1 desbordó (debe ser borrado por software)
  - TMR1 no desbordó

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(0)
EEIF	CMIF	RCIF	TXIF	-	CCP1IF	TMR2IF	TMR1IF

Figura 1.51: Registro PIR1

PIE1 (Registro de habilitación de interrupciones de periféricos)

- **EEIE** (bit de habilitación por finalización de la escritura EE)
  - 1: Habilita la interrupción por finalización de la escritura EE
  - 0: Deshabilita la interrupción por finalización de la escritura EE
- **CMIE** (bit de habilitación de interrupción por comparación)
  - 1: Habilita la interrupción producida por el comparador
  - 0: Deshabilita la interrupción producida por el comparador
- **RCIE** (bit de habilitación de interrupción por recepción del USART)

  - 1: Habilita la interrupción por recepción del USART
  - 0: Deshabilita la interrupción por recepción del USART

- **TXIE** (bit de habilitación de interrupción por transmisión del USART)
  - 1: Habilita la interrupción por transmisión del USART
  - 0: Deshabilita la interrupción por transmisión del USART

- **CCP1IE** (bit de habilitación de interrupción de CCP1)
  - 1: Habilita la interrupción de CCP1
  - 0: Deshabilita la interrupción de CCP1

- **TMR2IE** (bit de habilitación de interrupción por coincidencia de TMR2 y PR2)
  - 1: Habilita la interrupción por coincidencia de TMR2 y PR2
  - 0: Deshabilita la interrupción por coincidencia de TMR2 y PR2

- **TMR1IE** (bit de habilitación de interrupción por desbordamiento de TMR1)
  - 1: Habilita la interrupción por desbordamiento del TMR1
  - 0: Deshabilita la interrupción por desbordamiento del TMR1

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(0)
EEIE	CMIE	RCIE	TXIE	-	CCP1IE	TMR2IE	TMR1IE

Figura 1.52: Registro PIE1

PCON

- **OSCF**: Frecuencia oscilador INTRC/ER
  - 1: 4Mhz
  - 0: 37Khz

- **POR**: Bit de estado del Power-ON Reset
  - 1: No se produjo un Power-On Reset

- 0: Se produjo un Power-On Reset (Debe ponerse a '1' por software después del Power-On Reset)

- **BOD**: Bit de estado de Brown-Out (caída de la alimentación)
  - 1: No se produjo un reset por Brown-Out
  - 0: Se produjo un reset por Brown-Out

U-0	U-0	U-0	U-0	R/W-1	U-0	R/W-q	R/W-q
—	—	—	—	OSCF	—	$\overline{POR}$	$\overline{BOD}$
bit 7							bit 0

Figura 1.53: Registro PCON

CMCON (Registro de control del comparador)

- **C2OUT** (Comparador de 2 salidas)
  - C2INV = 0
    - 1: $V_{IN+} > V_{IN-}$
    - 0: $V_{IN+} < V_{IN-}$
  - C2INV = 1
    - 1: $V_{IN+} < V_{IN-}$
    - 0: $V_{IN+} > V_{IN-}$

- **C1OUT** (Comparador de 1 salida)
  - C1INV = 0
    - 1: $V_{IN+} > V_{IN-}$
    - 0: $V_{IN+} < V_{IN-}$
  - C1INV = 1
    - 1: $V_{IN+} < V_{IN-}$
    - 0: $V_{IN+} > V_{IN-}$

- **C2INV** (Comparador de 2 salidas con inversión)

- 1: Salida invertida
- 0: Salida no invertida

- **C1INV** (Comparador de 1 salida con inversión)
  - 1: Salida invertida
  - 0: Salida no invertida

- **CIS** (Interruptor de entrada al comparador)
  - CM2:CM0 = 001
    - 1: C1 V_{IN-} conectado a RA3
    - 0: C1 V_{IN-} conectado a RA0
  - CM2:CM0 = 010
    - 1
      - C1 V_{IN-} conectado a RA3
      - C2 V_{IN-} conectado a RA2
    - 0
      - C1 V_{IN-} conectado a RA0
      - C2 V_{IN-} conectado a RA1

- **CM2-0** (Modo comparador)

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(0)
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0

Figura 1.54: Registro CMCON

OPTION_REG (Registro de opciones)

- **PBPU(inv)** (bit de habilitación de pull-up del PORTB)
  - 1: PORTB pull-ups deshabilitadas
  - 0: PORTB pull-ups habilitadas

- **INTEDG** (bit de de selección del flanco que produce la interrupción)

- 1: Interrupción en el flanco de subida de RB0/INT
- 0: Interrupción en el flanco de bajada de RB0/INT

- **T0CS** (bit de selección del flanco de reloj del TMR0)
  - 1: Transiciones en el pin RA4/T0CKI
  - 0: Reloj del ciclo de instrucción interno (CLKOUT)
- **T0SE** (bit del flanco de la fuente del TMR0)
  - 1: Incremento en la transición de alta a baja del pin RA4/T0CKI
  - 0: Incremento en la transición de baja a alta del pin RA4/T0CKI
- **PSA** (bit de asignación de la preescala)
  - 1: Preescala asignada al WDT
  - 0: Preescala asignada al TMR0
- **PS2,PS1,PS0** (bits de selección del valor de la preescala)

PS2	PS1	PS0	TMR0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

7(1)	6(1)	5(1)	4(1)	3(1)	2(1)	1(1)	0(1)
$\overline{\text{PBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Figura 1.55: Registro OPTION_REG

VRCON (Registro de control de la tensión de referencia)

- **VREN** (Habilitación de VREF)
  - 1: Circuito VREF habilitado
  - 0: Circuito VREF deshabilitado
- **VRON** (Habilitación de salida de VREF)
  - 1: VREF conectada al pin RA2
  - 0: VREF desconectada del pin RA2
- **VRR** (Selección de rango de VREF)
  - 1: Nivel bajo
  - 0: nivel alto
- **VR3-0** (Selección del valor de VREF)

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(0)
VREN	VROE	VRR	-	VR3	VR2	VR1	VR0

Figura 1.56: Registro VRCON

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(1)	0(0)
CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D

Figura 1.57: Registro TXSTA

TXSTA

- **CSRC**: Bit de selección de reloj fuente
  - Modo asíncrono
    - -
  - Modo síncrono
    - 1: Modo Maestro (Reloj generado internamente desde BRG)

    - 0: Modo Esclavo (Reloj proveniente de fuente externa)

- **TX9**: Bit de habilitación de la transmisión de 9 bits
  - 1: Transmisión de 9 bits
  - 0: Transmisión de 8 bits
- **TXEN**: Bit de habilitación de la transmisión (Los bits SREN/CREN del registro RCSTA sobreescriben TXEN en modo síncrono)
  - 1: Habilita la transmisión
  - 0: Deshabilita la transmisión
- **SYNC**: Bit de selección de modo del USART
  - 1: Modo síncrono
  - 0: Modo asínrono
- **BRGH**: Bit de selección de la velocidad
  - Modo asíncrono
    - 1: Alta velocidad
    - 0: Baja velocidad
  - Modo síncrono
    - -
- **TRMT**: Bit de estado del registro de desplazamiento de la transmisión (Sólo lectura)
  - 1: TSR vacío
  - 0: TSR lleno
- **TX9D**: Noveno bit del dato transmitido. Puede ser el bit de paridad

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(x)
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

Figura 1.58: Registro RCSTA

RCSTA

- **SPEN**: Bit de habilitación del puerto serie
  - 1: Puerto serie habilitado
  - 0: Puerto serie deshabilitado
- **RX9**: Bit de habilitación de la recepción de 9 bits
  - 1: Recepción de 9 bits
  - 0: Recepción de 8 bits
- **SREN**: Bit de habilitación de recepción simple (Este bit se pone a cero cuando se completa la recepción)
  - 1: Modo asíncrono
    - -
  - 0: Modo sínrono Maestro
    - 1: Habilita la recepción simple
    - 0: Deshabilita la recepción simple
  - 0: Modo sínrono Esclavo
    - -
- **CREN**: Bit de habilitación de recepción contínua
  - Modo asíncrono
    - 1: Habilita la recepción contínua
    - 0: Deshabilita la recepción contínua
  - Modo síncrono
    - 1: Habilita la recepción contínua hasta que el bit CREN se borra
    - 0 : Deshabilita la recepción contínua

- **ADDEN**: Bit de habilitación de detección de dirección (Modo asíncrono de 9 bits)
  - 1: Habilita la detección de dirección. Habilita la interrupción y la carga del búffer de recepción cuando el bit RSR< 8 > está a '1'.
  - 0: Deshabilita la detección de dirección. Todos los bytes son recibidos y el noveno bit se puede usar como bit de paridad
- **FERR**: Bit de error de trama (Sólo lectura)
  - 1: Error de trama (Puede ser actualizado leyendo el registro RCREG y recibiendo el siguiente byte válido)
  - 0: No hay error de trama
- **OERR**: Bit de error de sobrecarga
  - 1: Error de sobrecarga (Se puede borrar poniendo el bit CREN a '0')
  - 0: No hay error de sobrecarga
- **RX9D**: Noveno bit del dato recibido (Sólo lectura)

TXREG y RCREG

Los registros ***TXREG*** y ***RCREG*** son los búferes de transmisión y recepción del USART respectivamente.

SPBRG

El establecimiento de la velocidad de señalización viene dado por este registro. En el modo asíncrono existe un error en la velocidad de señalización que dependerá de la frecuencia de reloj empleada; este aspecto debe ser tenido en cuenta a la hora de elegir el reloj apropiado.

Los diferentes valores de velocidades en función del contenido del registro SPBRG y de la frecuencia del oscilador se presentan en las tablas 1.6 - 1.14.

CCP1CON (Registro de control CCP1)

- **CCP1X,CCP1Y**: Bits menos significativos de PWM
- **CCP1M3,CCP1M2,CCP1M1,CCP1M0**: Bits de selección de modo CCP1

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	0.300	0	207
1.2	1.202	0.17	51
2.4	2.404	0.17	25
9.6	8.929	6.99	6
19.2	20.833	8.51	2
28.8	31.250	8.51	1
36.6	-	-	-
57.6	62.500	8.51	0
ALTA	0.244	-	255
BAJA	62.500	-	0

Cuadro 1.6: BRGH = 0 y Fosc = 4Mhz

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	1.202	0.17	129
2.4	2.404	0.17	64
9.6	9.766	1.73	15
19.2	19.531	1.72	17
28.8	31.250	8.51	4
36.6	31.250	6.99	4
57.6	52.083	9.58	2
ALTA	0.610	-	255
BAJA	158.250	-	0

Cuadro 1.7: BRGH = 0 y Fosc = 10Mhz

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	1.221	1.75	255
2.4	2.404	0.17	129
9.6	9.766	1.73	31
19.2	19.531	1.72	15
28.8	31.250	8.51	9
36.6	34.722	3.34	8
57.6	62.500	8.51	4
ALTA	1.221	-	255
BAJA	312.500	-	0

Cuadro 1.8: BRGH = 0 y Fosc = 20Mhz

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	1.202	0.17	207
2.4	2.404	0.17	103
9.6	9.615	0.16	6
19.2	19.231	0.16	2
28.8	27.798	3.55	8
36.6	35.714	6.29	6
57.6	62.500	8.51	3
ALTA	0.977	-	255
BAJA	250.000	-	0

Cuadro 1.9: BRGH = 1 y Fosc = 4Mhz

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	-	-	-
2.4	2.441	1.71	255
9.6	9.615	0.16	64
19.2	19.531	1.72	31
28.8	28.409	1.36	21
36.6	32.895	2.10	18
57.6	56.818	1.36	10
ALTA	2.441	-	255
BAJA	625.000	-	0

Cuadro 1.10: BRGH = 1 y Fosc = 10Mhz

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	-	-	-
2.4	-	-	-
9.6	9.615	0.16	129
19.2	19.231	0.16	64
28.8	29.070	0.94	42
36.6	33.784	0.55	36
57.6	59.524	3.34	20
ALTA	4.883	-	255
BAJA	1250.000	-	0

Cuadro 1.11: BRGH = 1 y Fosc = 20Mhz

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	-	-	-
2.4	-	-	-
9.6	9.615	0.16	103
19.2	19.231	0.16	51
28.8	28.571	0.79	34
36.6	33.333	0.79	29
57.6	58.824	2.12	16

Cuadro 1.12: Modo síncrono y Fosc = 4Mhz

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	-	-	-
2.4	-	-	-
9.6	-	-	-
19.2	19.231	0.16	129
28.8	29.070	0.94	85
36.6	33.784	0.55	73
57.6	58.140	0.94	42

Cuadro 1.13: Modo síncrono y Fosc = 10Mhz

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(0)
-	-	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0

Figura 1.59: Registro CCP1CON

CCPR1L,CCPR1H (Registro CCP1)

Este registro se usa para los datos manejados en las operaciones CCP. Su uso depende del modo de funcionamiento.

- **Modo captura**. Almacena el contenido del Temporizador1 en el momento de la captura.
- **Modo comparación**. Almacena el dato que se compara con el Temporizador1.
- **Modo PWM**. CCPR1L almacena un dato con el que se obtiene el ciclo de trabajo. CCPR1H se usa para el procesamiento interno y es un registro de sólo lectura.

Velocidad señalización (K-bps)	Velocidad señalización	Error (%)	SPBRG (Decimal)
0.3	-	-	-
1.2	-	-	-
2.4	-	-	-
9.6	-	-	-
19.2	-	-	-
28.8	28.902	0.35	172
36.6	33.557	0.13	148
57.6	57.471	0.22	86

Cuadro 1.14: Modo síncrono y Fosc = 20Mhz

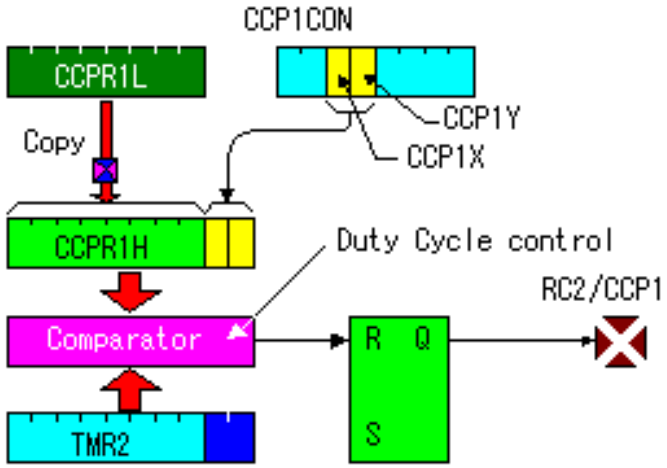


Figura 1.60: Bits CCP1X,CCP1Y

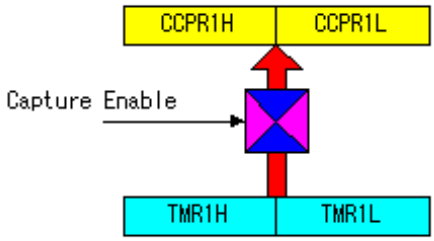


Figura 1.61: Modo captura

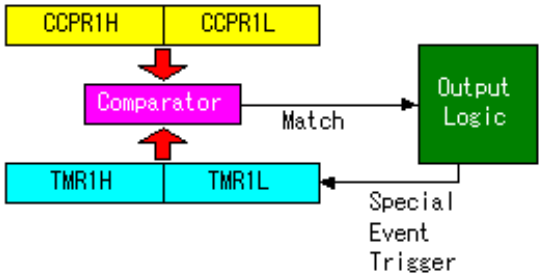


Figura 1.62: Modo comparación

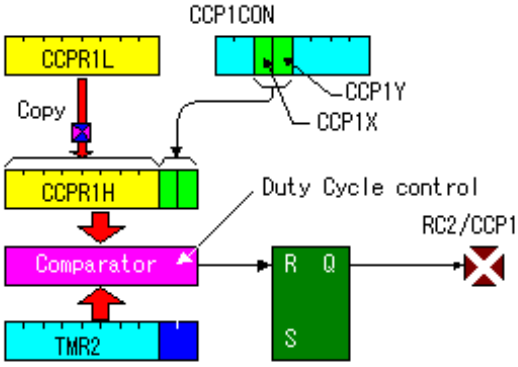


Figura 1.63: Modo PWM

PR2 (Registro de periodo de PWM)

Este registro establece el periodo de la señal PWM. El periodo del pulso de salida PWM depende de la comparación del contenido de PR2 y el valor del Temporizador2.

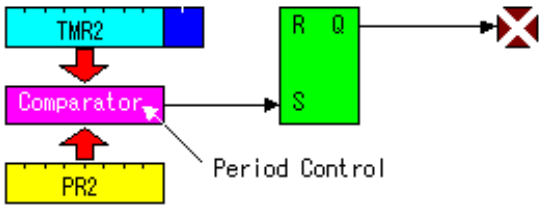


Figura 1.64: Registro PR2

INTCON (Registro de control de interrupciones)

- **GIE**: Bit de habilitación de interrupciones globales
  - 1: Habilita las interrupciones no enmascarables
  - 0: Deshabilita todas las interrupciones
- **PEIE**: Bit de habilitación de interrupciones de periféricos
  - 1: Habilita todas las interrupciones de periféricos no enmascarables
  - 0: Deshabilita todas las interrupciones de periféricos
- **T0IE**: Bit de habilitación de interrupción por desbordamiento del TMR0
  - 1: Habilita la interrupción del TMR0
  - 0: Deshabilita la interrupción del TMR0
- **INTE**: Bit de habilitación de interrupciones en la patilla RB0/INT
  - 1: Habilita las interrupciones de la patilla RB0/INT
  - 0: Deshabilita las interrupciones de la patilla RB0/INT
- **RBIE**: Bit de habilitación de interrupción por cambios en el puerto B
  - 1: Habilita las interrupciones por cambios en el puerto B
  - 0: Deshabilita las interrupciones por cambios en el puerto B
- **T0IF**: Bit del flag de interrupción por desbordamiento del TMR0
  - 1: TMR0 desbordó (Debe ser borrado por software)

- 0: TMR0 no desbordó

- **INTF**: Bit flag por interrupción en la patilla RB0/INT
  - 1: Se produjo una interrupción en la patilla RB0/INT (Debe ser borrado por software)
  - 0: No se produjo una interrupción en la patilla RB0/INT
- **RBIF**: Bit flag de interrupción por cambios en el puerto B
  - 1: Se produjo un cambio en al menos una de las patillas RB7-RB4 (Debe ser borrado por software)
  - 0: No se produjo un cambio en ninguna de las patillas RB7-RB4

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(x)
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Figura 1.65: Registro INTCON

TMR0 (Registro contador del Temporizador 0)

Es un registro que almacena la cuenta del Temporizador 0. Puede almacenar un máximo de 255 cuentas.

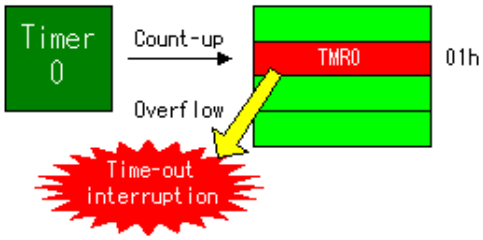


Figura 1.66: Registro TMR0

TMR1L, TMR1H (Registros de cuenta del temporizador 1)

Este registro almacena la cuenta del temporizador 1. Puede almacenar una cuenta de valor máximo 65535.

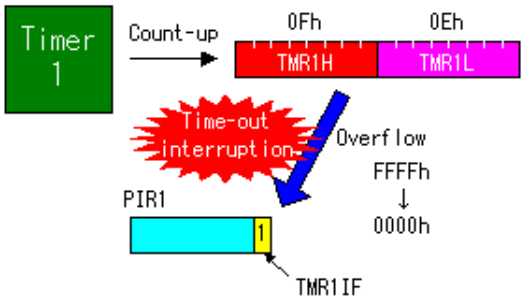


Figura 1.67: Registro TMR1L, TMR1H

T1CKPS1	T1CKPS0	Preescala
0	0	1:1
0	1	1:2
1	0	1:4
1	1	1:8

Cuadro 1.15: Preescala según los valores de T1CKPS1,T1CKPS0

T1CON (Registro de control del temporizador 1)

- **T1CKPS1,T1CKPS0**: Bits de selección de preescala de reloj de entrada del temporizador 1.
- **T1OSCEN**: Bit de control de habilitación del oscilador del temporizador 1
  - 1: Oscilador habilitado
  - 0: Oscilador deshabilitado
- **T1SYNC**: Bit de control de sincronización de la entrada de reloj externa (Este bit se ignora si TMR1CS=0)
  - 1: No sincroniza con la entrada de reloj externo
  - 0: Sincroniza con la entrada de reloj externo
- **TMR1CS**: Bit de selección de la fuente de reloj del temporizador 1
  - 1: Reloj externo procedente del pin RC0/T1OSO/T1CKI (en el flanco de subida)
  - 0: Reloj interno (Fosc/4)

- **TMR1ON**: Bit de habilitación del temporizador 1
  - 1: Habilita el temporizador
  - 0: Para el temporizador

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(0)
-	-	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON

Figura 1.68: Registro T1CON

TMR2 (Registro de cuenta del Temporizador 2)

Este registro puede almacenar un valor de 255 cuentas.

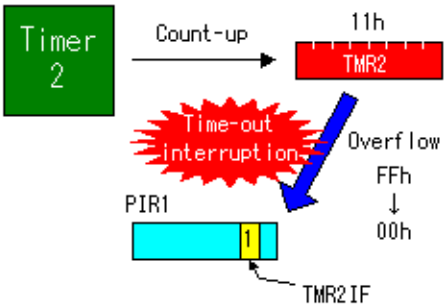


Figura 1.69: Registro TMR2

T2CON (Registro de control del Temporizador 2)

7(0)	6(0)	5(0)	4(0)	3(0)	2(0)	1(0)	0(0)
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Figura 1.70: Registro T2CON

- **TOUTPS3,TOUTPS2, TOUTPS1, TOUTPS0**: Bits de selección de postescala del Temporizador 2

TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	Postescala
0	0	0	0	1:1
0	0	0	1	1:2
0	0	1	0	1:3
0	0	1	1	1:4
0	1	0	0	1:5
0	1	0	1	1:6
0	1	1	0	1:7
0	1	1	1	1:8
1	0	0	0	1:9
1	0	0	1	1:10
1	0	1	0	1:11
1	0	1	1	1:12
1	1	0	0	1:13
1	1	0	1	1:14
1	1	1	0	1:15
1	1	1	1	1:16

Cuadro 1.16: Postescala del Temporizador 2

- **TMR2ON**: Bit de habilitación del Temporizador 2
  - 1: Temporizador 2 habilitado
  - 0: Temporizador 2 deshabilitado
- **T2CKPS1,T2CKPS0**: Bits de selección de la preescala del reloj del Temporizador 2

T2CKPS1	T2CKPS0	Preescala
0	0	1:1
0	1	1:4
1	0	1:16
1	1	1:16

EECON1 (Registro de control de la EEPROM)

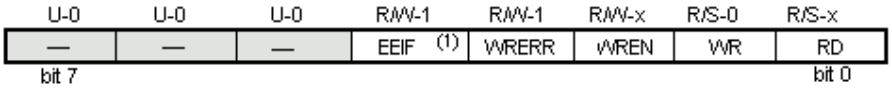


Figura 1.71: Registro EECON1

Seguidamente se describen los bit de control de registro EECON1.

- **WRERR**: Flag de error de escritura en la EEPROM.
  - 1: El proceso de lectura se ha producido prematuramente (se ha producido un Reset por MCLR o un WDT durante el proceso).
  - 0: Se ha producido el proceso de escritura con éxito.
- **WREN**: bit de habilitación de escritura.
  - 1: Permite inicializar el ciclo de escritura.
  - 0: Inhibe la escritura.
- **WR**: bit de inicio de escritura.
  - 1: Cuando se le pone a 1 comienza el ciclo de escritura de la memoria no volátil. (El bit se pone de nuevo a cero por hardware cuando la escritura se completa).
  - 0: Toma este valor cuando completa el ciclo de escritura de la memoria no volátil.
- **RD**: bit de inicio de lectura
  - 1: Cuando se le pone a 1 se inicia un ciclo de lectura. El bit RD se pone a cero por hardware.
  - 0: No ha comenzado el ciclo de lectura de la memoria no volátil.

EECON2 (Registro de control de la EEPROM)

El registro EECON2, no está implementado físicamente y sólo se utiliza para la operación de lectura, es decir, antes de iniciar la escritura de un dato en la memoria, se debe escribir en el registro EECON2 primero el dato 55h y posteriormente el dato AAh.

EEDATA y EEADR

En el registro EEADR se almacena la dirección de la localidad de memoria que se desea acceder y en EEDATA es colocado un dato de 8 bits, el cual se encuentra guardado en esa posición. Tanto cuando se escribe como cuando es lee una localidad de memoria, estos dos registros siempre son utilizados.

1.7.11. Comunicación asíncrona con el Pic16f628 (USART)

USART y modos de comunicación

USART es el puerto de comunicación con el que se encuentra equipado el Pic16f628. USART corresponde a las iniciales de Universal Synchronous Asynchronous Receiver Transmitter (Receptor Transmisor Síncrono Asíncrono Universal).

El USART se puede configurar con los siguientes modos de comunicación:

- **Asíncrono** (Full dúplex)
- **Síncrono Maestro** (Half dúplex)
- **Síncrono Esclavo** (Half dúplex)

Generalmente, en las comunicaciones con el USART, un bloque de datos está compuesto por 8 bits. El módulo USART posee la capacidad de establecer varias comunicaciones utilizando una detección de direcciones de 9 bits.

Modo de comunicación asíncrono

En el modo de comunicación asíncrono del USART, el Pic16f628 dispone de un puerto de recepción que se usa para tomar datos y de un puerto de transmisión para enviar información, por lo que es posible recibir y enviar a la vez (Full dúplex). La patilla RB2/TX/CK (pin 8) es la de transmisión, mientras que la RB1/RX/DT (pin 7)es a través de la cual se produce la recepción.

Para poder usar las patillas de TX y RX, figura 1.72, es necesario señalar como entrada (1) RB1 y como salida (0) RB2 en el registro TRISB.

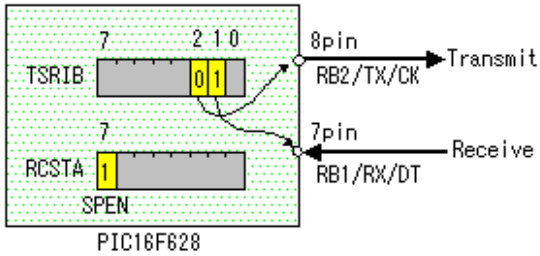


Figura 1.72: Patillas asociadas al USART en el Pic16f628

Para poder usar las patillas RB1 y RB2 como puerto del USART, el bit SPEN del registro RCSTA debe estar a '1'. RCSTA es el registro de control y estado de la recepción, y el bit SPEN es el que habilita el puerto serie.

Modo	Baja velocidad (BRGH=0)	Alta velocidad (BRGH=1)
Asíncrono (SYNC=0)	Fosc/(64(X + 1))	Fosc/(16(X + 1))
Síncrono (SYNC=1)	Fosc/(4(X + 1))	Fosc/(4(X + 1))

Cuadro 1.17: Cálculo de la frecuencia de reloj del PIC y de las velocidades de comunicación del USART

La designación del modo asícrono se realiza poniendo a '0' el bit SYNC registro TXSTA.

Selección de la velocidad de comunicación

La velocidad de intercambio de datos está controlada por BRG (Baud Rate Generator). BRG se usa tanto en las comunicaciones asíncronas como en las síncronas del USART, y está controlado por el registro SPBRG. En el caso del modo asíncrono, el bit BRGH del registro TXSTA se usa también para el control de la velocidad. Cuando se escribe un valor en el registro SPBRG, el temporizador BRG se inicializa.

La frecuencia del PIC y la velocidad de comunicación se calculan con las fórmulas de la tabla 1.17, donde Fosc es la frecuencia de reloj del PIC y X es el valor del registro SPBRG; varía entre 0 y 255.

Operación de transmisión

Cuando se realiza una transmisión asíncrona con el USART, se pone el bit SPEN del registro RCSTA a '1' de forma que RB2 pasa a ser el puerto de transmisión. En el momento en el que se carga un dato que se va a transmitir en el registro TXREG, el hardware lo manda al registro TSR (Registro de desplazamiento de transmisión) para su envío finalmente por el puerto TX.

La transmisión del dato es llevada a cabo poniendo el bit TXEN del registro TXSTA a '1'. El primer transmitido es el menos significativo (LSB). Cuando el contenido del registro TXREG es enviado al TRS, el bit TXIF del registro PIR1 toma el valor '1' y da lugar a una interrupción. El bit TX1E del registro PIE1 debe ponerse a '1'. Esta interrupción indica que el contenido del registro TXREG se ha enviado al TSR posibilitando la carga de un nuevo dato en TXREG. Los datos se envían continuamente si al detectar esta interrupción se colocan los siguientes datos en TXREG uno tras otro dando lugar al diagrama temporal de la figura 1.73. El bit TXIF no puede ser borrado por sotfware, ya que únicamente se pone a '0' cuando un dato es cargado en TXREG.

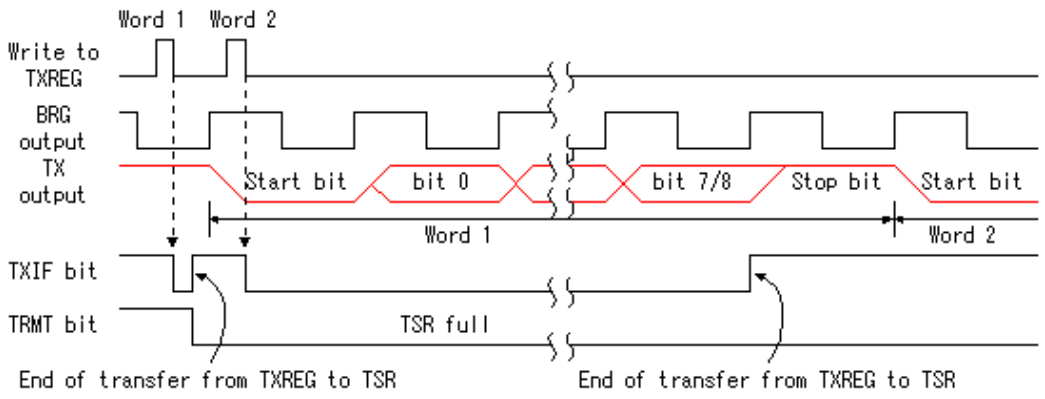


Figura 1.73: Diagrama de tiempos del envío de bytes uno detrás de otro

El bit TRTM del registro TXSRA se pone a '1' cuando el dato del registro TSR es enviado. Este bit no produce interrupción. Por lo tanto, para conocer que el registro TSR está vacío es necesario sondear el bit TRMT de forma periódica. Debido a que el registro TSR no está mapeado en la memoria de datos, no se puede ni leer ni escribir en él de forma directa.

La paridad no está soportada por el hardware pero puede obtenerse si se implementa por software. En este caso el noveno bit del dato es el de paridad y se almacena en en el bit TX9D del registro TXSTA. Es necesario habilitar el bit de paridad poniendo el bit TX9 a '1'. TX9D debe adquirir su valor antes de que el dato sea enviado a TXREG.

La transmisión comienza tan pronto como se carga el dato en TXREG. Cuando se borra el bit TXEN al transmitir un dato, el transmisor se inicializa y el pin RB2/TX/CK cambia a estado de alta impedancia.

De forma resumida los pasos a seguir en una transmisión asíncrona son:

1. Inicializar el registro SPBRG para obtener la velocidad apropiada. Si lo que se desea es alta tasa de velocida el bit BRGH se configura con el valor lógico '1'.

2. Habilitar el puerto serie asíncrono borrando el bit SYNC de registro TXSTA y poniendo a '1' el bit SPEN del registro RCSTA.

3. Si se desea interrupción poner a '1' el bit de habilitación TXIE del registro PIE1.

4. Si se quiere una transmisión de 9 bits, es necesario que el bit TX9 del registro TXSTA pase a '1'.

5. Habilitar la transmisión poniendo el bit TXEN del registro TXSTA a '1'. En este

punto, se puede colocar el dato en el registro TXREG y el bit del registro PIR1 pasa a '1'.

6. Si fué seleccionada la transmisión de 9 bits el noveno bit se tiene que colocar en TX9D.

7. Cargar el dato en el registro TXREG. Comienza en este momento la transmisión.

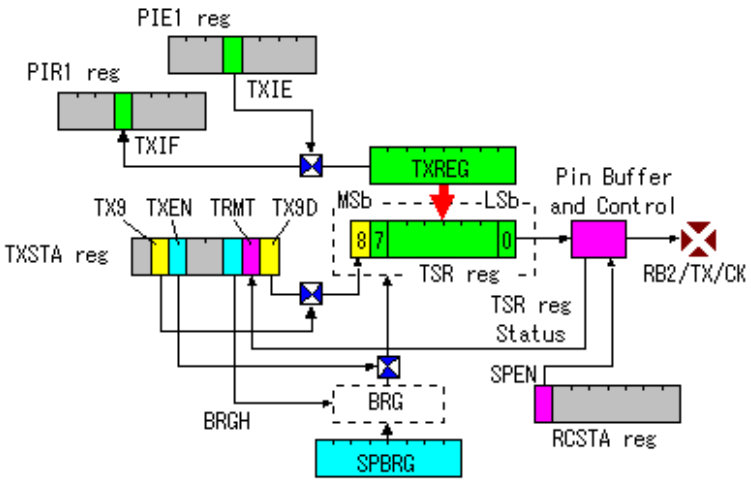


Figura 1.74: Proceso de transmisión con el USART

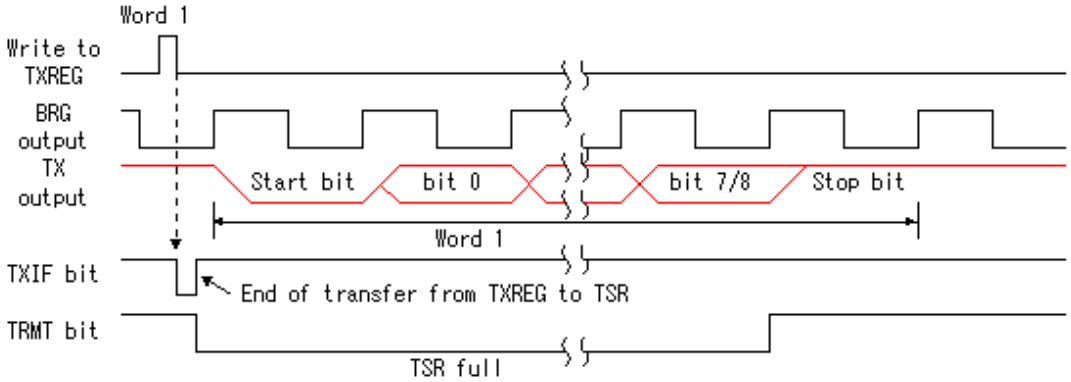


Figura 1.75: Diagrama de tiempos del envío de una palabra

Operación de recepción

Para realizar una recepción asíncrona con el USART, se le da el valor lógico '1' al bit SPEN del registro RCSTA para que la patilla RB1 sea el puerto de recepción.

El dato recibido por el puerto de recepción es muestreado 3 veces para averiguar donde hay nivel alto y donde nivel bajo de señal. El dato es almacenado en el registro RSR deacuerdo con la velocidad especificada por el registro SPBRG y el bit BRGH del registro TXSTA. Cuando se detecta un bit de STOP, el contenido del registro RSR se transfiere a RCREG.

El bit RCIF del registro PIR1 se pone a '1' cuando el dato es almacenado en RCREG y se produce la interrupción. Para que esta interrupción ocurra se ha de poner el bit RCIE del registro PIE1 a '1'. RCREG está formado por 2 búferes FIFO (First In First Out) y puede almacenar 2 datos. Esta es una medida de protección para evitar que el retardo en la lectura de los datos por software pueda producir la pérdida de alguno de ellos. El bit RCIF es únicamente de lectura y es borrado por el hardware cuando todos los registros RCREG son leídos. Si ambos registros RCREG están llenos porque no se han leído aún, cuando se ha completado la recepción con el registro RSR, el bit OERR de RCSTA se pone a '1' y se señala un error de sobrecarga. El dato que fué almacenado en RSR se pierde, y la operación de recepción no se realiza. Esta situación borra el bit CREN de RCSTA y se continúa normalmente de nuevo. Mediante esta operación el bit OERR se borra.

El bit FERR de RCSTA se pone a '1' cuando se detectan errores de trama en RSR. Los bits RX9D y FERR se sobreescriben cada vez que se recibe un dato en una FIFO. El bit FERR se debe comprobar antes de leer el contenido del registro RCREG. Cuando se recibe una trama válida después de una errónea la información de FERR desaparece.

Se resume a continuación la secuencia de una recepción asńcrona:

1. Inicializar el registro SPBRG para conseguir la tasa de bps deseada. Si se busca una tasa alta es necesario poner el bit BRGH de TXSTA a '1'.

2. Habilitar el puerto serie asíncrono borrando el bit SYNC del registro TXSTA y poniendo a '1' el bit SPEN de RCSTA.

3. Si se quieren interrupciones poner a '1' el bit de habilitación RCIE de PIE1.

4. Para una recepción de 9 bits se configura el bit RX9 de RCSTA poniéndolo a '1'.

5. Habilitar la recepción poniendo a '1' el bit CREN de RCSTA.

6. El bit RCIF de PIR1 se pondrá a '1' cuando la recepción se complete y se generará una interrupción si se habilitó esta opción.

7. Leer el registro RCSTA para tomar el noveno bit (si está habilitado) y para determinar si ocurrió algún error durante la recepción.

8. Leer los 8 bit correspondientes al dato leyendo el registro RCREG.

9. Si ocurrió algún error, borrar la señalización del error poniendo a '0' el bit CREN.

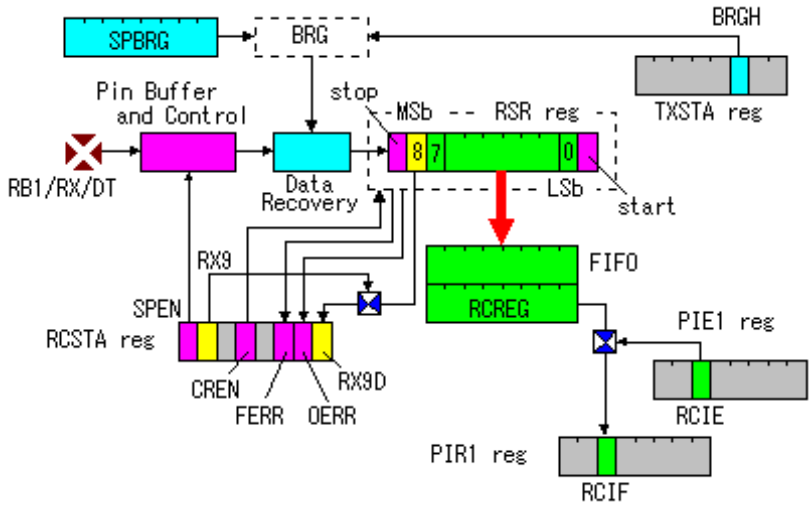


Figura 1.76: Proceso de recepción con el USART

Detección de direcciones

El noveno bit se puede usar para la detección de direcciones. El proceso requerido para esta operación es el que se muestra en el diagrama de la figura 1.78. Esta operación se lleva a cabo en el caso de la transferencia de 9 bits, para lo cual es necesario poner a '1' los bits RX9 y ADDEN que pertenecen al registro RCSTA. De esta forma, sólo cuando el bit noveno posee el valor lógico '1', el dato recibido se almacena en RCREG.

1.7.12. Temporizadores del PIC16F628

Temporizador 0

Las especificaciones del temporizador 0 del Pic16F628 son las mismas que las del Pic16F84. Posee las siguientes características:

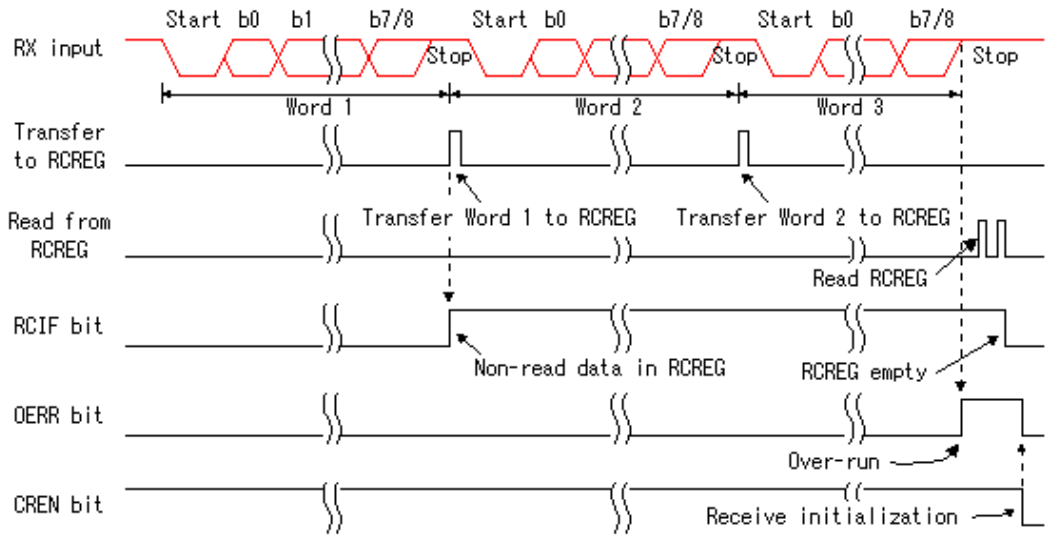


Figura 1.77: Diagrama de tiempos de la recepción

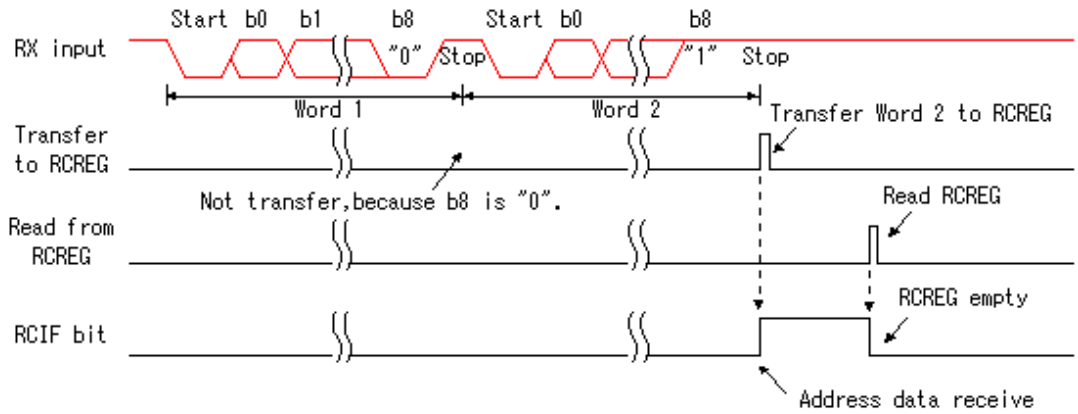


Figura 1.78: Diagrama de tiempos en una detección de dirección

- Contador/temporizador de 8 bits (TMR0).
- El contador se puede leer y escribir.
- Preescala programable de 8 bits.
- Selección de reloj interno o externo
- Produce interrupción cuando la cuenta pasa de FFh a 00h.
- Selección del flanco de subida o de bajada de la señal de reloj externa.

La figura 1.79 muestra el diagrama de bloques del temporizador 0 (TMR0) y el temporizador perro guardián (WDT). Cada registro de los que aparece debajo de la figura 1.79 interviene en la configuración del temporizador.

La preescala puede ser asignada al TMR0 o bien al WDT, pero a este último como postescala. En la figura 1.79 se muestra conectada al TMR0. El bit PSA (asignación de preescala) del registro OPTION_REG determina a cuál de estos dos elementos modifica la cuenta. La preescala es un contador programable que se configura con los bits PS0, PS1, PS2 de OPTION_REG.

El temporizador no se puede leer o escribir. Una modificación de TMR0 borra la preescala, y lo mismo ocurre si se borra el temporizador perro guardián con CLRWDT cuando éste está asignado al WDT.

El TMR0 es un contador binario de 8 bits con una cuenta mínima de valor 256. Cuando este contador pasa de 255 (FFh) a 0 (00h) se produce una interrupción que da lugar a que el bit T0IF de INTCON tome el valor '1'. El hardware está diseñado para que cuando los bits GIE y TOIE de INTCON estén a '1', el contador de programa, como consecuencia de la interrupción, se carge con el valor 04h para que la rutina de atención correspondiente realice las operaciones oportunas. Una vez atendida la interrupción el bit TIF debe ponerse a '0' por software. La interrupción producida por el TMR0 no puede despertar al μC del estado SLEEP de bajo consumo.

El empleo de la preescala se debe a que habitualmente una cuenta de valor 256 se queda corta, porque por ejemplo, en el caso de utilizar una frecuencia de reloj interno de 20Mhz la frecuencia de entrada del contador es 5Mhz (FOSC/4). Como el periodo de esta señal de reloj es 200ns (1/5Mhz = 0.2μs) se obtiene que una vuelta completa del TMR0 se corresponde con un valor temporal de 51.2μs (0.2$\mu s \times 256$), siendo éste un tiempo muy pequeño.

Con la preescala asignada al TMR0 se divide la señal de entrada por 2, 4, 8, 16, 32, 64, 128, o 256. Por ejemplo, cuando la preescala se configura para dividir por 2, hay un pulso de salida por cada 2 pulsos que entran, y si la preescala es 256, habrá 1 pulso de salida por cada 256 pulsos entrantes. En el ejemplo anterior, con una la preescala de 256, el TMR0 acaba su cuenta en unos 13ms (51.2μsx256).

La entrada al temporizador TMR0 puede ser un reloj externo o interno. Para usar un reloj externo , el bit T0CS del registro OPTION_REG y el bit 4 del registro TRISA deben estar a '1'. Para seleccionar el flanco del pulso de reloj se modifica el bit T0SE de OPTION_REG. Borrando este bit se selecciona el flanco de subida y poniéndolo a '1' el flanco de bajada.

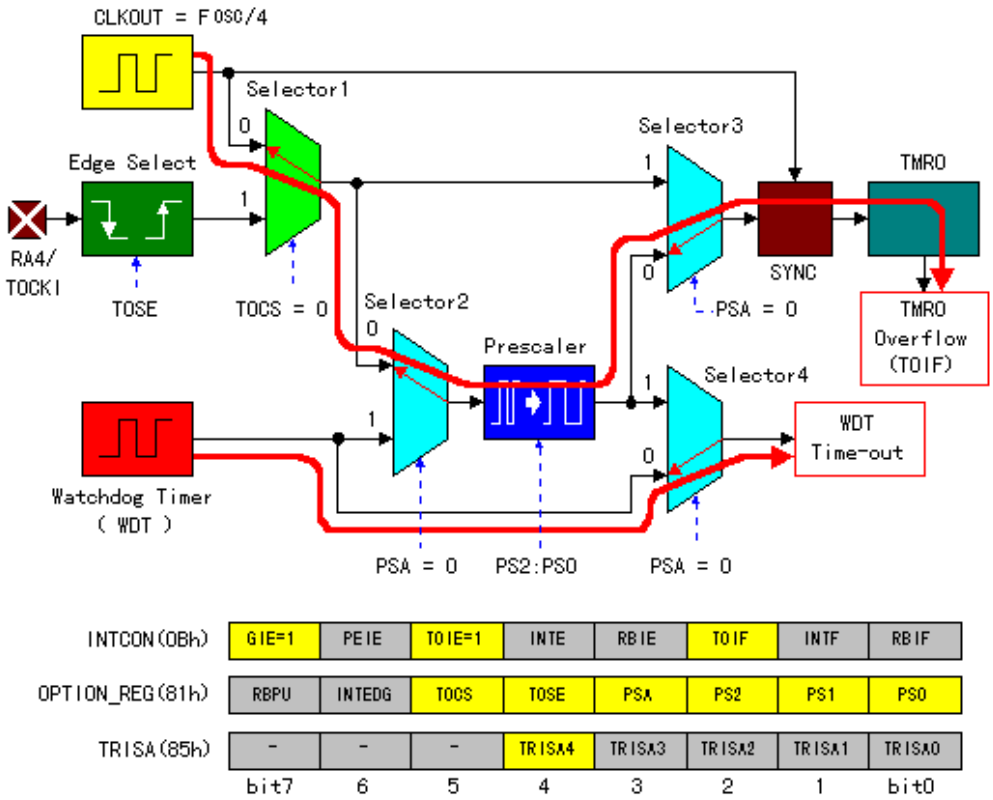


Figura 1.79: Temporizador 0 y temporizador perro guardián

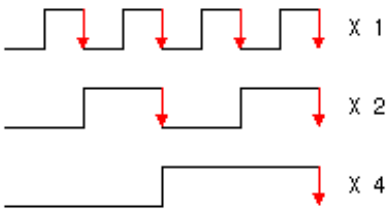


Figura 1.80: Efecto de la preescala

El oscilador del temporizador perro guardián es independiente del reloj de la CPU, y tiene un time-out de unos 18ms.

La función del WDT es la de evitar errores de operación del software, como por ejemplo la ejecución de instrucciones que no forman parte del programa o los bucle infinito, en los que se ejecuta el mismo fragmento de código de forma repetida. Por lo tanto la presencia del WDT no es para nada imprescindible porque si existen errores de diseño en el software, es el propio programador el que se encargará de detectarlos, y un

contínuo reset del sistema se lo podría impedir. La presencia o no del WDT se decide en la palabra de configuración.

Para evitar que acabe la cuenta del WDT se resetea de forma periódica vía software usando la instrucción CLRWDT. Si el temporizador no se resetea, cuando expira su cuenta, se fuerza un reset de la CPU de forma inmediata. La preescala puede aumentar el valor del tiempo de cuenta del WDT. En el caso de asignar la preescala al WDT, ésta podría tener 8 valores, que son 1, 2, 4, 8, 16, 32, 64, o 128. Por ejemplo, si la preescala es 128, se consigue un tiempo de cuenta de unos 2 segundos (18ms×128 = 2,304ms).

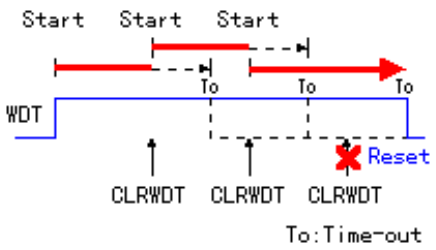


Figura 1.81: Actualización del WDT

Temporizador 1

El módulo temporizador 1 es un temporizador/contador de 16 bits formado por 2 registros de 8 bits (TMR1H and TMR1L). Al igual que el temporizador 0 tiene una preescala con distintos valores seleccionables. La interrupción por desbordamiento puede activarse o desactivarse poniendo a '1' o borrando el bit de habilitación de interrupción TMR1IE del registro PIE. La interrupción se produce cuando la cuenta pasa de FFFFh a 0000h.

El modo de operación de este módulo, para que funcione como temporizador o trabaje como contador, se decide con el bit de selección de reloj (bit TMR1CS del registro T1CON). Si este bit, el TMR1CS, se borra funciona como temporizador, y en caso contrario, cuando está a '1' se convierte en contador.

Modo temporizador

En modo temporizador, el valor del Temporizador1 se incrementa en cada ciclo de instrucción (FOSC/4). Este temporizador dispone de una entrada interna de reset que puede ser generado por los módulos CCP (ver sección 1.7.13). Poniendo a '1' el bit

T1OSCEN de T1CON, se habilita una conexión de un oscilador de cristal en los pines 12 y 13.

Modo contador

Cuando este módulo se encuentra en modo contador, se incrementa la cuenta en cada flanco de subida del reloj de entrada. Si el bit T1OSCEN de T1CON está a '0' se incrementa la cuenta con el reloj proveniente de la patilla 12 (RB6/T1OSO/T1CKI). En el caso en el que este mismo bit esté a '1' el reloj que provoca el incremento de la cuenta con cada uno de sus flancos de subida es el que entra por el pin 13 (RB7/T1OSI(). Para que la cuenta de flancos de subida pueda comenzar es necesario que ocurra en primer lugar un flanco de bajada inicial. El primer flanco de subida posterior a este inicio se ignora. Se puede sincronizar el reloj externo con el interno borrando el bit T1SYNC de T1CON. Esta sincronización se lleva a cabo después del bloque correspondiente a la preescala como se aprecia en la figura 1.82. Este temporizador se usa en el modo de *Captura* y *Comparación* (ver sección 1.7.13), pero siempre y cuando el contador no se encuentre en modo contador asíncrono.

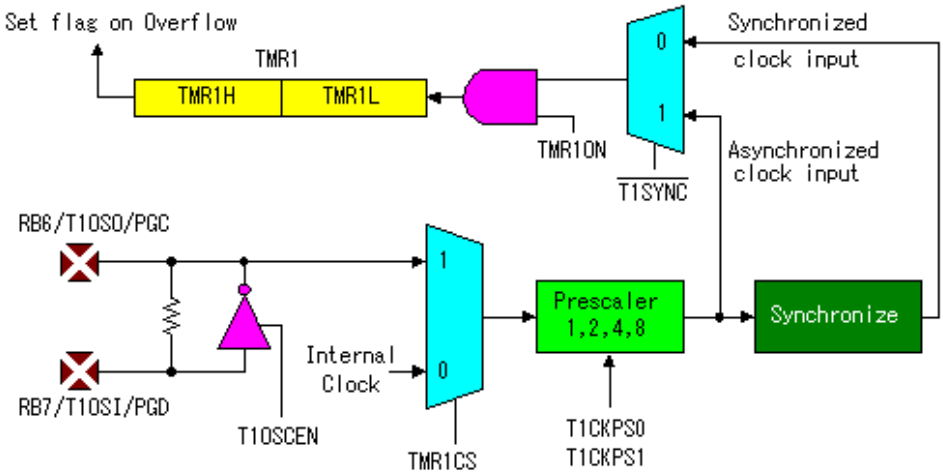


Figura 1.82: Temporizador 1

Temporizador 2

El Temporizador2 es un temporizador de 8 bits con preescala y postescala, que tiene un registro de periodo de 8 bits (PR2). Los valores de preescala para el reloj de entrada que se pueden seleccionar configurando los bits T2CKPS1 y T2CKPS0 del registro T2CON son: 1:1, 1:4 y 1:16.

La salida match del TMR2, que se puede ver en la figura 1.83, pasa a través de una postescala que añade un escalado de valores de entre 1:1 y 1:16 para después generar una interrupción señalada en el bit TMRIF de PIR.

El Temporizador2 incrementa su cuenta desde 00h hasta que el valor de ésta coincide con el que almacena PR2; en el siguiente ciclo vuelve otra vez a 00h. PR2 parte con un valor inicial FFh en el reset.

Este temporizador se puede desconectar borrando el bit TMR2ON de T2CON para lograr un ahorro de consumo.

El temporizador2 se usa en PWM (ver sección 1.7.13) para controlar el periodo de los pulsos.

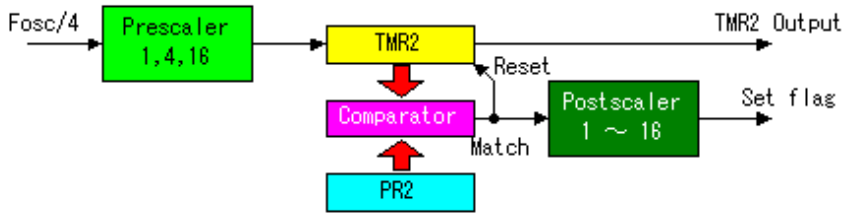


Figura 1.83: Temporizador 2

1.7.13. Funcionalidad CCP del Pic16f628

Generalidades

CCP se corresponde con las iniciales de Captura/Comparación/PWM (Modulación de la anchura del pulso).

- **Captura**. La función de captura toma los 16 bits del registro Temporizador1 cuando ocurre un evento en el pin RB3/CCP1 (ver figura 1.84). Esta operación se puede usar para medir el periodo de una señal.

- **Comparación**. Esta función compara constantemente los 16 bits del Temporizador1 con el valor del registro CCPR1 (ver figura 1.85).

- **PWM**. Con este modo de trabajo, se consiguen impulsos lógicos cuya anchura del nivel alto es de duración variable, que son de enorme aplicación en el control de dispositivos tan populares como los motores y triacs. (Duty). PWM usa el temporizador2 (ver figura 1.86).

CCP1M3-0	Evento
0000	CCP off (resetea el módulo CCP1)
0100	Cada flanco de bajada
0101	Cada flanco de subida
0110	Cada 4 flancos de subida
0111	Cada 16 flancos de subida

Cuadro 1.18: Selección de eventos

El registro CCP1 está compuesto de 2 registros de 8 bits cada uno: CCPR1L es la parte baja, mientras que CCPR1H conforma la parte alta. El control de las operaciones del registro CCP1 lo lleva a cabo otro registro, el CCP1CON.

Como resultado de la comparación se puede generar un disparo que produce un reset en el Temporizador1.

Función de captura

En el modo Captura, la patilla RB3/CCP1 debe estar configurada como entrada estableciendo tal opción en el registro TRISB. Si este pin se configura como salida, una escritura en el puerto puede ocasionar una condición de captura.

Un evento se selecciona mediante unos bits de control, CCP1M3:CCP1M0, que están contenidos en el registro CCP1CON. Los eventos se definen como se muestra en la tabla 1.18.

Cuando se lleva a cabo la captura, el bit de flag de petición de interrupción del registro PIR1 se pone a '1'. Este flag de interrupción debe ser borrado por software. Si se produce otra captura antes de que el valor almacenado en el registro CCPR1 sea leído, la captura anterior se pierde.

El Temporizador1 debe estar configurado como temporizador o en modo contador sincronizado para poder usar la función de captura, pero en modo contador sincronizado, podría no realizarse la operación de captura. Cuando el modo de captura está cambiado, se podría producir una falsa interrupción por captura. Para evitar este problema el bit CCP1IE del registro PIE se debe mantener a '0' y se debe borrar el bit CCP1F después de cada cambio en el modo de operación.

Función de comparación

Cuando se produce una coincidencia, después de una comparación, se realiza la operación que indiquen los bits CCP1M3:CCP1M0 del registro CCP1CON (ver tabla 1.19).

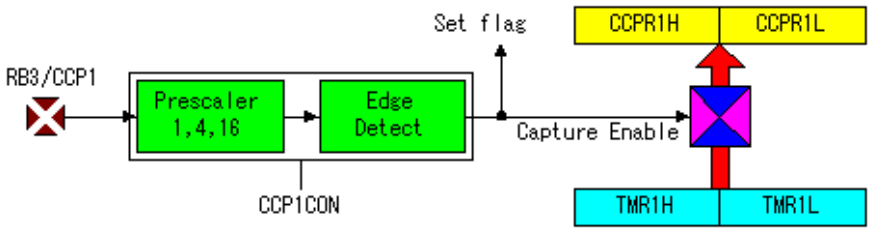


Figura 1.84: Captura

CCP1M3-0	Operación
1000	Salida a '1' (CCP1IF está a '1')
1001	Salida a '0' (CCP1IF está a '1')
1010	Mantiene hasta que desaparece la igualdad (CCP1IF está a '1')
1011	Disparo evento especial (CCP1IF está a '1'); mantiene hasta que desaparece la igualdad En caso de CCP1, inicializa TMR1

Cuadro 1.19: Selección de operaciones

En modo comparación es necesario que la patilla RB3/CCP1 esté configurada como salida. El Temporizador1 debe estar en modo temporizador o en modo contador sincronizado; en este último modo la operación de comparación podría estar inactiva.

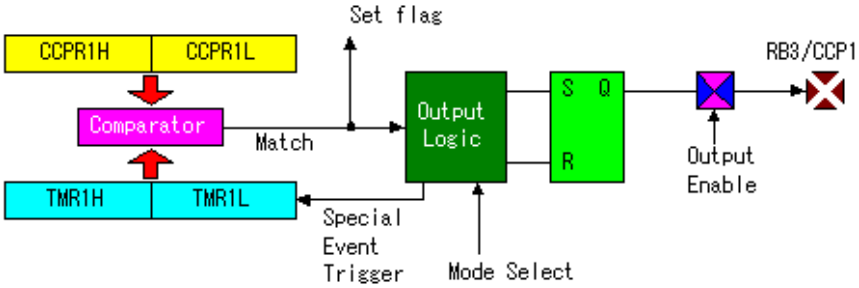


Figura 1.85: Comparación

PWM

En el modo PWM (Modulación de la anchura de pulsos), la patilla CCP1 da lugar a una salida PWM de más de 10 bits de resolución. Para las operaciones PWM es necesario

que los bits CCP1M3 y CCP1M2 que pertenecen a CCP1CON estén a '1'.

Es imprescindible que el pin RB3/CCP1 se configure como salida. El valor del registro PR2, el periodo de oscilación del reloj y la preescala del Temporizador2 especifican el periodo de PWM mientras que el ciclo de trabajo (tiempo que la salida permanece a '1') viene especificado por el registro CCPR1L, el reloj y la preescala del Temporizador2. CCPR1L contiene los 8 bits más significativos que se completan con los LSB que se encuentran en CCP1X:CCP1Y.

El ciclo de trabajo más corto posible se corresponde con el periodo del reloj del oscilador. Por ejemplo, en el caso de emplear un oscilador de 10Mhz, tiene un valor de 0.1μs. La resolución máxima de este ciclo de trabajo es 1024. Si ocurre que el ciclo de trabajo es mayor que el periodo PWM, la patilla CCP1 no se pone a '0'; en caso contrario, si el valor del ciclo de trabajo dado por CCPR1L+CCP1X+CCP1Y es cero, la patilla CCP1 no pasa a '1' en ningún momento.

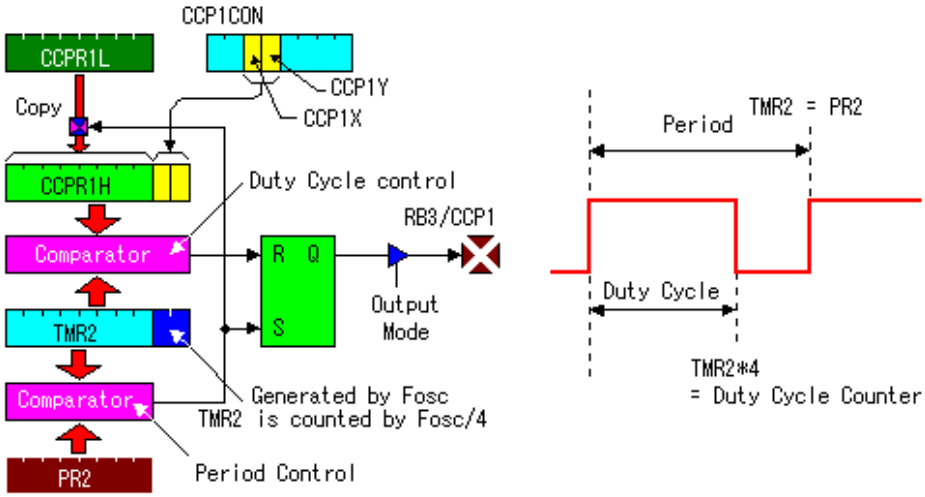


Figura 1.86: Estructura interna del módulo CCP1 cuando funciona en modo PWM y señal PWM

Ejemplo de aplicación PWM

Se desea programar la patita RB3/CCP1 de un PIC16F628 para que trabaje en modo PWM obteniendo la mayor frecuencia posible de esta señal. El circuito se muestra en la 1.87. Cuando el microcontrolador tiene voltaje de alimentación el led conectado en el pin RB2 debe de encenderse. Cuando el botón 1 esta presionado el ancho de pulso debe incrementarse poco a poco teniendo como máximo 100 % de modulación. Cuando

el botón 2 es presionado el ancho de pulso debe disminuir poco a poco teniendo como mínimo 0 % de modulación.

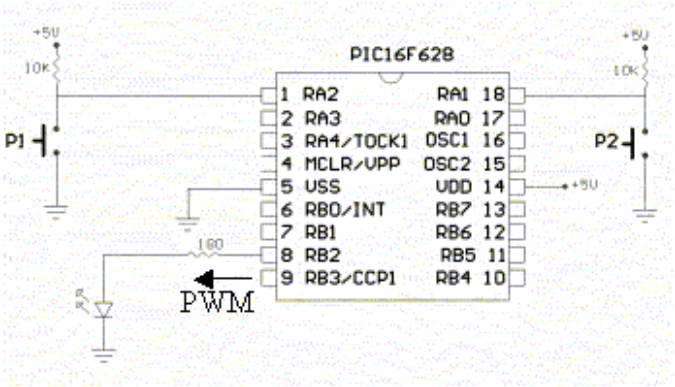


Figura 1.87: Circuito de pruebas para PWM

Los diagramas de flujo del programa son mostrados en las figuras 1.88 y 1.89.

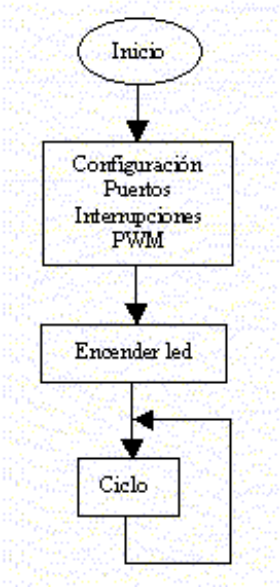


Figura 1.88: Programa principal de configuración para PWM

1.7.14. Módulo de comparación analógica

Generalidades

El Pic16f628 posee un comparador de voltaje analógico, que se muestra en la figura 1.90. Para esta función dispone de dos circuitos, con la capacidad de comparar los voltajes

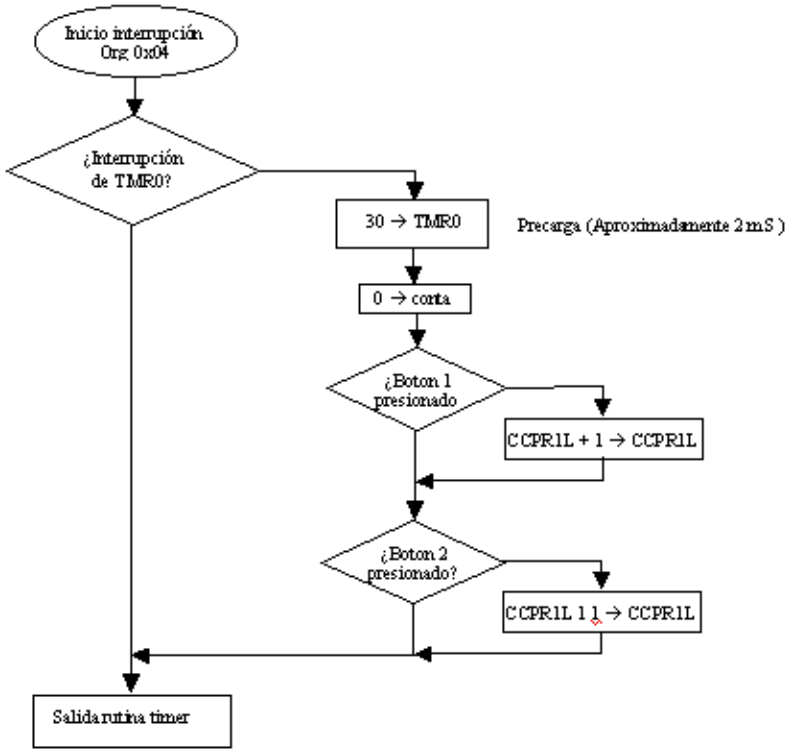


Figura 1.89: Interrupción de TMR0 cada 60 ms para PWM

de los pines RA0-RA3 entre sí, y estos voltajes con el de referencia.

La función de comparación del voltaje analógico se controla a través del contenido del registro CMCON.

Es posible provocar una interrupción cuando cualquiera de las salidas del comparador sufre un cambio. Para que esto sea posible es necesario poner a '1' el bit CMIE de PIE1 además de los bits PEIE y GIE del registro OPTION_REG. Cuando ocurre la interrupción se activa el bit CMIF de PIR1, bit que después de la atención de la interrupción es necesario borrar.

Control de salida

La salida del comparador está conectada a los bits C1OUT y C2OUT del registro CMCON. El contenido de estos dos bits dependerá por lo tanto del resultado de la comparación de las entradas. Configurando los bits C1INV y C2INV se puede condicionar el contenido de C1OUT y C2OUT. En la tabla 1.20 se muestra un ejemplo para CMP1 (en el caso de CMP2 sería lo mismo).

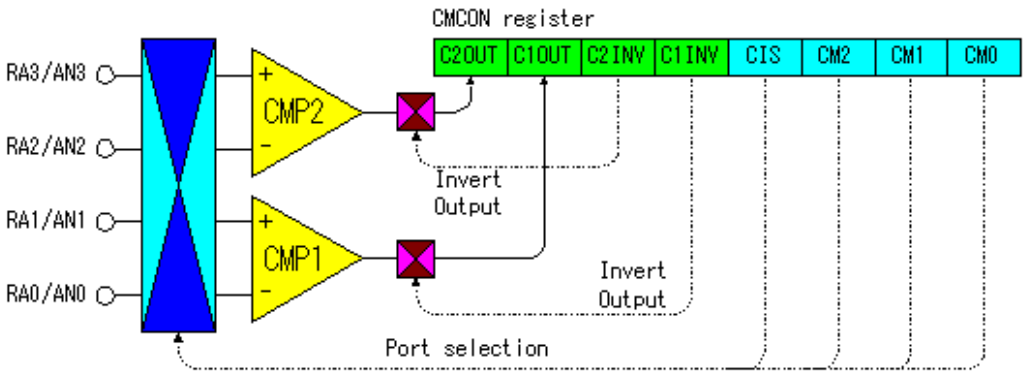


Figura 1.90: Módulo de comparación

Condición de entrada	C1INV	C1OUT
$\mathbf{VIN}_+ > VIN_-$	0	1
$\mathbf{VIN}_+ > VIN_-$	1	0
$\mathbf{VIN}_+ < VIN_-$	0	0
$\mathbf{VIN}_+ < VIN_-$	1	1

Cuadro 1.20: Relación entre C1INV y C1OUT

Control de entrada

La entrada al comparador se puede controlar con los bits CM0-CM2 y CIS del registro CMCON, que inicialmente están a '0'; en esta situación el puerto RA0-RA3 es el puerto analógico de AN0-AN3.

Es necesario poner a '1' CM0-CM2 durante el proceso de inicialización cuando no se pretenda usar el comparador.

CIS se corresponde con las iniciales de Comparator Input Switch (Interruptor de entrada del comparador). Este interruptor se usa cuando se pretende comparar más de una señal analógica. Cuando CM2-CM0 es '010', se puede comparar un máximo de 4 señales analógicas con el voltaje de referencia mediante 2 comparadores, modificando CIS.

Voltaje de referencia

La función que crea el voltaje de referencia se encuentra en un circuito independiente. Cuando CM2-CM0 es '010', el voltaje creado con este circuito se aplica al terminal positivo de entrada de cada comparador.

VR3-0	Nivel bajo	Nivel alto
0000	0.00	1.25
0001	0.21	1.41
0010	0.42	1.56
0011	0.63	1.72
0100	0.83	1.88
0101	1.04	2.03
0110	1.25	2.19
0111	1.46	2.34
1000	1.67	2.50
1001	1.88	2.66
1010	2.08	2.81
1011	2.29	2.97
1100	2.50	3.13
1101	2.71	3.28
1110	2.92	3.44
1111	3.13	3.59

Cuadro 1.21: Tensiones de referencia para Vdd = 5 V

Configurando VRCON se puede llevar el voltaje de referencia a la patilla RA2.

El voltaje de referencia se designa mediante 4 bits (VR3-VR0) incluidos en VRCON. Se puede especificar 16 niveles diferentes (0 a 15). Hay 2 niveles que son de nivel bajo y nivel alto; en el caso de ser de nivel alto el valor máximo es de unos 3.6V. El rango de tensiones se puede especificar modificando el bit VRR de VRCON.

Para realizar el cálculo de la tensión de referencia se utilizan las fórmulas 1.3 y 1.4.

- **VRR = 1**(Nivel bajo)

$$Vref = (VR < 3 : 0 > /24) \times Vdd \tag{1.3}$$

- **VRR = 0**(Nivel alto)

$$Vref = (1/4 + VR < 3 : 0 > /32) \times Vdd \tag{1.4}$$

Los valores de la tensiones obtenidas para un valor Vdd = 5V se muestran en la tabla 1.21.

1.7.15. Lectura y escritura en la memoria EEPROM

Características generales

En los PICs, no es posible tener acceso a la memoria EEPROM de una forma directa como en el caso de la memoria de uso común (FSR). Para poder leer o grabar datos en la memoria EEPROM es necesario utilizar cuatro registros especiales y de esta manera manejar de forma indirecta los datos. Los registros son los siguientes:

- EECON1
- EECON2
- EEDATA
- EEADR

Los PIC16F628 cuentan 128 bytes de memoria EEPROM comenzando en la dirección 0x00 hasta 0x7F. Cuando un nuevo byte es escrito en una localidad de memoria, este es sobre escrito de forma automática sobre el viejo dato. Cada localidad de memoria EEPROM pude ser escrita alrededor de 1,000,000 de veces sin sufrir daño alguno.

El tiempo que se necesita para grabar un dato no tiene un tiempo exacto, dado que esta determinado por un chip timer interno destinado especialmente a esta actividad. Además, este timer esta afectado directamente por la temperatura ambiente y del voltaje de alimentación del microcontrolador. Mediante el registro EEADR se pueden direccionar como máximo 256 bytes de EEPROM pero solo los 128 primeros son implementados físicamente.

EECON1 es un registro de control, el cual solo tiene implementados los 4 bits bajos y los otros 4 bits restantes no son implementados, lo que hace que si se desean leer estos bits, estos arrojaran el valor de 0.

Fijando mediante software en '1' los bits de control RD y WR se inicia la lectura y escritura, respectivamente. Estos bits una vez puestos en marcha serán puestos a '0' vía hardware al finalizar la lectura o escritura. El propósito de no poder borrar por medio de software estos bits es para prevenir la escritura y lectura de datos prematuros.

El bit WREN al ser fijado a 1 da permiso para habilitar la memoria EEPROM. Si un *Power Up* ocurre, el bit WREN será puesto a cero automáticamente. El bit WRERR pasa automáticamente a 1 cuando la operación de escritura es interrumpida por un reset de MCLR o un reset de desborde del timer del WDT durante una operación normal del microcontrolador. Si un reset sucede exactamente cuando el proceso de escritura

está teniendo lugar, el usuario programador puede revisar el estado de la bandera WRERR y volver a escribir el dato en la misma localidad de memoria. El dato tiene que volver a ser colocado en los registros EEDATA y EEADR. La bandera de interrupción EEIF pasa a '1' al finalizar la escritura. Ésta tiene que ser borrada por software.

EECON2 no es un registro implementado físicamente por lo que si se desea leer solo se obtendrán en todas sus localidades ceros. El registro EECON2 es utilizado exclusivamente para la secuencia de escritura en la EEPROM.

Lectura de la memoria EEPROM

Para leer de la memoria EEPROM han de seguirse los siguientes pasos:

- Escritura de la dirección que hay que leer en el registro EEADR
- Poner a 1 el bit RD del registro EECON, para habilitar la lectura
- Lectura del dato leído y espera a que termine la operación
- El dato está disponible en el registro EEDATA

Veamos un ejemplo práctico:

```
LECTURA BCF     STATUS,RP0   ; Selecciona banco 0
        MOVLW   MEM1         ; Direccion a leer de
        MOVWF   EEADR        ; la EEPROM
        BSF     STATUS,RP0   ; Selecciona banco 1
        BSF     EECON1,RD    ; Activar lectura
ESPERA  BTFSC   EECON1,RD    ; Espera final de lectura
        GOTO    ESPERA       ; A que baje la bandera
        BCF     STATUS,RP0   ; Selecciona banco 0
        MOVF    EEDATA,W     ; W se carga con el valor
                             ; leido en eeprom
```

Como la memoria EEPROM es bastante lenta, es importante esperar a que este ciclo de lectura termine, aunque algunas veces se omita. Pero es aun más importante es esta espera en el ciclo de escritura, ya que la EEPROM puede tardar en ser escrita hasta 10 ms.

Escritura de la memoria EEPROM

El proceso de escritura es aún más complejo ya que deberemos hacer todo lo anterior y además escribir un código especial de protección. Estos pasos los vemos en las siguientes líneas:

- Cargar en EEADR la dirección de la posición a escribir
- Cargar en el registro EEDATA el valor a grabar
- Ejecutar la siguiente secuencia que inicia la escritura de cada byte y además sirve de protección frente a errores eventuales. Esta secuencia siempre es la misma y ha de ejecutarse siempre

```
MOVLW 55H
MOVWF EECON2                ; Escribe 55h
MOVWF AAH
MOVWF EECON2                ; Escribe AAh
BSF   EECON1,WR             ; Coloca a 1 el bit de escritura
```

- Esta última instrucción inicia el proceso de escritura. Cuando se termina, el bit EEIF está a 1 y, si ha sido habilitada la interrupción de EEPROM haciendo uso del bit EEIE del registro INTCON, esta interrupción se genera
- Mediante software es necesario poner a cero el bit EEIF

Veamos un ejemplo de escritura típico:

```
ESCRITURA BCF   STATUS, RP0 ; Selecciona el banco 0
          MOVLW MEN1
          MOVWF EEADR       ; Escribe la direccion en EEADR
          MOVLW DATO1
          MOVWF EEDATA      ; Se escribe el dato en EEDATA
          BSF   STATUS, RP0 ; Selecciona el banco 1
          BSF   EECON1, WREN; Permiso de escritura

          ;Comienzo Secuencia de escritura
          MOVLW 0x55
          MOVWF EECON2      ; Se escribe el dato 55h en eecon2
          MOVLW 0xAA        ;
          MOVWF EECON2      ; Se escribe AA h en eecon2
          BSF   EECON1, WR  ; Comienza la escritura
ESPERA    BTFSC EECON1, WR  ; Espera a que termine la escritura
          GOTO  ESPERA
          BCF   STATUS, R0  ; Selecciona el banco 0
```

La escritura no se iniciará si la secuencia de introducir 55 y AA en el EECON2, y habilitar el bit WR no es correcta.

Es recomendable que se deshabiliten las interrupciones durante este proceso, mediante las siguientes instrucciones, con el fin de evitar errores no deseados:

```
     BCF INTCON, GIE                ; Deshabilita interrupcion
     BSF INTCON, GIE                ; Habilita interrupcion
```

Para evitar errores, también es recomendable que el bit WREN esté desactivado durante todo el programa excepto en el momento de escritura. Debemos tener en cuenta que este bit no se pone a cero automáticamente mediante hardware. Una vez iniciado el ciclo de escritura, si ponemos a cero WREN, esto no afectará a este ciclo En este caso el bit WR estará inhibido y no se podrá poner a uno.

Después del ciclo el bit WR es puesto a cero por hardware y el EEIF es puesto a uno (si EEIE lo está). Si EEIE, está habilitado, EEIF debe ser puesto a cero por software.

Verificación de la escritura

Dependiendo de la aplicación, la experiencia en programación dice que los datos escritos en la EEPROM deben ser verificados comparándolos con el dato que se acaba de escribir. Esto debe usarse en aplicaciones en las que un bit de la EEPROM sufre ciclos de lectura/escritura hasta rozar el límite de las especificaciones.

Generalmente el fallo de escritura en un bit de la EEPROM será un bit que se escribe como un 0 lógico pero devuelve un 1 debido a la pérdida de ese bit.

Un ejemplo de verificación del dato escrito es el siguiente:

```
     BCF    STATUS, RP0       ; Nos situamos en el banco 0
     :                        ; Aqui va otro codigo
     :
     MOVF   EEDATA, W         ; Debemos estar en el banco 0
     BSF    STATUS, RP0       ; Cambiamos al banco 1
READ BSF    EECON1, RD        ; Leemos el dato q se guarda en
     BCF    STATUS, RP0       ; EEDATA, y cambiamos a banco 0

     ; Son los valores escritos (en W) y los leidos (en EEDATA)
     ; los mismos??

     SUBWF  EEDATA, W         ; Restamos ambos valores
     BTFSS  STATUS, Z         ; Si la operacion es cero, son iguales
     GOTO   WRITE_ERR         ; Si es asi, saltamos a WRITE_ERR
     :                        ; Si no, seguimos con el programa
```

1.7.16. Set de instrucciones del PIC16F628

El PIC16F628 pertenece a la gama media y es de tipo RISC; esto quiere decir que tiene un juego de instrucciones reducido, en concreto de 35. Estas 35 instrucciones o

nemónicos (del inglés mnemonics y a su vez proveniente del juego de palabras: Nem On Icks) serán la base de funcionamiento del PIC.

Las instrucciones fundamentalmente se dividen en tres tipos. Esta división viene dada por el tipo de datos con los que trabajan:

- Instrucciones orientadas a los bytes (byte-oriented operations)
- Instrucciones orientadas a los bits (bit-oriented operations)
- Operaciones con literales y de control (literal and control operations)

Instrucciones orientadas al manejo de bytes (Registros)

Estas instrucciones pueden ser de simple o doble operando de origen. El primer operando de origen será siempre el registro seleccionado en la instrucción, el segundo, en caso de existir, será el registro W. El destino, es decir donde se guardará el resultado, será el registro seleccionado o el W, según se seleccione con un bit de la instrucción.

A continuación se detalla este tipo de instrucciones.

ADDWF

- **Acción**: Suma el contenido del acumulador y el registro dado, y el resultado lo guarda en d.
- **Sintaxis**: **ADDWF f,d**.
- **Funcionamiento**: Add W to file register (Añade W al registro).
- **Hexadecimal**: 07 ff.
- **Bits (OPCODE)**: 00 0111 dfff ffff.
- **Operación**: d = W + f (d puede ser W o f).
- **Descripción**: Esta instrucción suma el contenido de un registro específico al contenido de W donde f puede ser un registro cualquiera con un determinado valor.
- **Comentarios**: Bit d:
    - Si vale 1, el resultado se guarda en el registro f.
    - Si vale 0, el resultado se guarda en el acumulador W.

- **Registro STATUS**: Modifica los bits Z, DC y C.
  - Z vale 1 si el resultado de la operación es 0.
  - DC vale 1 si el resultado de la operación es un número superior a 15.
  - C vale 1 si el resultado de la operación es positivo o el bit 7 del registro que contiene el resultado vale 0. En caso contrario C vale 0 (resultado negativo).
- **Ejemplo**: Tomamos como valores iniciales W = 5 y DATO = 10, donde dato es un registro cualquiera.

```
ADDWF DATO       ; DATO = 15 y W = 5.
ADDWF DATO, 1    ; DATO = 15 y W = 5.
ADDWF DATO, 0    ; W = 15 y DATO = 10.
ADDWF DATO, W    ; W = 15 y DATO = 10.
```

- **Ciclos de máquina**: 1

ANDWF

- **Acción**: Realiza la operación AND entre un registro y W.
- **Sintaxis**: **ANDWF f,d**.
- **Funcionamiento**: AND W with f.
- **Hexadecimal**: 05 ff.
- **Bits (OPCODE)**: 00 0101 dfff ffff.
- **Operación**: d = W AND f (d puede ser W o f).
- **Descripción**: Esta instrucción realiza la operación lógica AND entre el acumulador y el registro f; el resultado se guarda dependiendo del valor de d. Si éste se omite, el valor por defecto es 1 y se guarda en f.
- **Comentarios**: La operación AND es una de las operaciones básicas del álgebra de Boole. La operación lógica es:

  $S = f \cdot W$

  Esta instrucción realiza esta operación para cada uno de los 8 bits de los dos registros, dos a dos, guardando el resultado en el registro correspondiente.

- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**:

```
    ADDWF REG1,1
```

  Antes de la instrucción:

  W = 0x17

  REG1 = 0xC2

  Después de la instrucción:

  W = 0x17

  REG1 = 0x02

- **Ciclos de máquina**: 1

CLRF

- **Acción**: Borra un registro.
- **Sintaxis**: **CLRF f**.
- **Funcionamiento**: Clear file register.
- **Hexadecimal**: 01 8f.
- **Bits (OPCODE)**: 00 0001 1fff ffff.
- **Operación**: F = 0.
- **Descripción**: Esta instrucción borra un registro específico, poniendo sus bits a cero.
- **Registro STATUS**: Modifica el bit Z y lo pone a 1 (ya que el resultado de la operación es 0).
- **Ejemplo**: Tenemos un registro que se llama dato y que vale 3F. Ponemos:

```
    CLRF    dato
```

Ahora dato vale 00.

- **Ciclos de máquina**: 1

CLRW

- **Acción**: Borra el acumulador.
- **Sintaxis**: **CLRW**.
- **Funcionamiento**: Clear W.
- **Hexadecimal**: 01 8f.
- **Bits (OPCODE)**: 00 0001 0xxx xxxx.
- **Operación**: W = 0.
- **Descripción**: Esta instrucción borra el registro W solamente.
- **Comentarios**: Donde pone xxx en la instrucción en hexadecimal, significa que no importa qué valor puede contener
- **Registro STATUS**: Modifica el bit Z y lo pone a 1 (ya que el resultado de la operación es 0)
- **Ejemplo**: Tenemos el acumulador cargado con el valor 3F. Ponemos:

```
    CLRW
```

Ahora W vale 00.

- **Ciclos de máquina**: 1

COMF

- **Acción**: Complementa el registro F.

- **Sintaxis**: **COMF f,d**.
- **Funcionamiento**: Complement f.
- **Hexadecimal**: 09 ff.
- **Bits (OPCODE)**: 00 1001 dfff ffff.
- **Operación**: d = NOT f (d puede ser W o f).
- **Descripción**: Esta instrucción complementa un registro, es decir, los ceros los convierte en unos, y los unos en ceros.
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**: Supongamos que tenemos un registro f denominado regist = 00111011; cuando es aplicada la instrucción tenemos que los 0 cambian a valores 1 y los valores 1 cambian a 0 obteniéndose un registro invertido. El resultado será regist = 11000100.
- **Ciclos de máquina**: 1

DECF

- **Acción**: Decrementa el registro f.
- **Sintaxis**: **DECF f,d**.
- **Funcionamiento**: Decrement f.
- **Hexadecimal**: 03 ff.
- **Bits (OPCODE)**: 00 0011 dfff ffff.
- **Operación**: d = f 1 (d puede ser W o f).
- **Descripción**: Esta instrucción decrementa en una sola unidad el registro 'f'.
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.

- **Ejemplo**: Nuestro registro se llama regist = 5; cuando se aplica la instrucción DECF f,0 el resultado será W=4.

  Por el contrario, si aplicamos la instrucción DECF f,1 el resultado será regist = 4.

- **Ciclos de máquina**: 1

DECFSZ

- **Acción**: Decrementa el registro f, y si el resultado es cero, se salta una instrucción.
- **Sintaxis**: **DECFSZ f,d**.
- **Funcionamiento**: Decrement f, skip if 0.
- **Hexadecimal**: 0B ff.
- **Bits (OPCODE)**: 00 1011 dfff ffff.
- **Operación**: d = f 1, si d = 0 SALTA (d puede ser W o f).
- **Descripción**: Esta instrucción decrementa el contenido del registro direccionado por el parámetro f, y si el resultado es 0 salta la instrucción siguiente. Si no, sigue con su curso habitual.
- **Comentarios**: Aquí nos enfrentamos ante la primera instrucción que plantea una condición, y que modifica el curso del PC.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
DECFSZ   VALOR, W
INSTRUCCION 1
INSTRUCCION 2
```

Si el contenido del registro VALOR al decrementarlo es igual a 0, se guarda el resultado en el acumulador y sigue con la INSTRUCCION2, saltándose la INSTRUCCION1.

Si el resultado que guardamos en W no es 0, sigue con la INSTRUCCION1 y después con la INSTRUCCION2 (no se salta la inmediata siguiente).

- **Ciclos de máquina**: 1 (2) Si tiene que saltar ocupa dos ciclos.

INCF

- **Acción**: Suma una unidad al registro f.
- **Sintaxis**: **INCF f,d**.
- **Funcionamiento**: Increment f.
- **Hexadecimal**: 0A ff.
- **Bits (OPCODE)**: 00 1010 dfff ffff.
- **Operación**: d = f + 1 (d puede ser W o f).
- **Descripción**: Esta instrucción incrementa en una sola unidad el registro f.
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**: Si tenemos un registro llamado DIA = 7.

  Aplicando la instrucción:

```
    INCF DIA, 0
```

  tendremos W = 8 y DIA = 7.

  Si aplicamos esta otra:

```
    INCF DIA, 1
```

  tendremos DIA = 8.

- **Ciclos de máquina**: 1

INCFSZ

- **Acción**: Incrementa en 1 a f, y si f = 0 salta la siguiente instrucción.

- **Sintaxis**: **INCFSZ f,d**.
- **Funcionamiento**: Increment f, Skip if 0.
- **Hexadecimal**: 0F ff.
- **Bits (OPCODE)**: 00 1111 dfff ffff.
- **Operación**: d = f + 1, si d = 0 SALTA (d puede ser W o f).
- **Descripción**: Esta instrucción incrementa en una sola unidad el registro 'f', en la cual si el resultado d es igual a cero, entonces salta la instrucción siguiente.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**: Ejecutamos las siguientes instrucciones:

```
INCFSZ   VALOR, W        ; el resultado se almacenara en W
INSTRUCCION 1            ; salta aqui si W no es 0
INSTRUCCION 2            ; salta aqui si W es 0
```

  Si el contenido del registro VALOR es igual a 0, al incrementarlo en una unidad, se guarda el resultado en el acumulador y sigue con la INSTRUCCION2, saltándose la INSTRUCCION1.

  Si el resultado que guardamos en W no es 0, sigue con la INSTRUCCION1 y después con la INSTRUCCION2 (no se salta la inmediata siguiente).

- **Ciclos de máquina**: 1 (2) Si tiene que saltar ocupa dos ciclos.

IORWF

- **Acción**: Operación lógica OR entre el acumulador y un registro.
- **Sintaxis**: **IORWF f,d**.
- **Funcionamiento**: Inclusive Or W with F.
- **Hexadecimal**: 04 ff.
- **Bits (OPCODE)**: 00 0100 dfff ffff.
- **Operación**: d = W OR f (d puede ser W o f).

- **Descripción**: Esta instrucción realiza una operación lógica OR inclusivo entre el acumulador W y el registro direccionado por el parámetro f. El parámetro d determina donde se almacenará el resultado de la operación. Si no se pone nada, el valor por defecto es 1 y se guarda en f.

- **Comentario**: La operación OR inclusivo suele llamarse OR a secas, pero se pone así para diferenciarla de la Suma Exclusiva.

  La operación lógica que describe esta instrucción es esta:

  F + W = S

  Se puede ver que basta con que uno de los dos registros tenga un uno para que la salida sea un uno también.

- **Registro STATUS**: Modifica el bit Z.

  - Z vale 1 si el resultado de la operación es 0.

- **Ejemplo**:

```
IORWF REG1, 0 ;
```

  Antes de la instruccón:

  REG1 = 0x13

  W = 0x91

  Después de la instrucción:

  REG1 = 0x13

  W = 0x93

  Z = 1

- **Ciclos de máquina**: 1 (2) Si tiene que saltar ocupa dos ciclos.

MOVF

- **Acción**: Mueve el contenido de un registro al acumulador o al propio registro.

- **Sintaxis**: **MOVF f,d**.

- **Funcionamiento**: Move f.
- **Hexadecimal**: 08 ff.
- **Bits (OPCODE)**: 00 1000 dfff ffff.
- **Operación**: d = f (d puede ser W o f).
- **Descripción**: Esta instrucción mueve el contenido del registro f en el mismo registro f o en W. D determina el destino del resultado. Si no se pone nada, el valor por defecto es 1 y se guarda en f.
- **Comentario**: Se suele usar para mover datos al acumulador. El hecho de que se pueda mover sobre sí mismo no es otro que para mirar el resultado en el registro STATUS.
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**: Tenemos el registro EDAD = 38

```
MOVF    EDAD, 0        ; hace que W = 38.
MOVF    EDAD, 1        ; hace que EDAD = 38.
MOVF    EDAD, W        ; hace que W = 38.
MOVF    EDAD           ; hace que EDAD = 38.
```

- **Ciclos de máquina**: 1.

MOVWF

- **Acción**: Mueve el acumulador al registro f.
- **Sintaxis**: **MOVWF f**.
- **Funcionamiento**: Move W to f.
- **Hexadecimal**: 00 ff.
- **Bits (OPCODE)**: 00 0000 1fff ffff.
- **Operación**: f = W.

- **Descripción**: Esta instrucción copia el contenido del acumulador W en el registro direccionado por el parámetro f.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**: Si queremos escribir el valor 10H en el registro TMR0, que está situado en la dirección 01h, tendremos que cargar primero el valor en el acumulador y después copiarlo al registro.

```
    MOVWF    10H            ; cargar el valor 10H en el acumulador.
    MOVWF    01H            ; copia el acumulador en la direccion 01H.
```

O escrito de otra manera:

```
    MOVWF    10H             ; cargar el valor 10H en el acumulador.
    MOVWF    TMR0            ; copia el acumulador en el registro TMR0.
```

- **Ciclos de máquina**: 1.

NOP

- **Acción**: No opera.
- **Sintaxis**: **NOP**.
- **Funcionamiento**: No Operation.
- **Hexadecimal**: 00 00.
- **Bits (OPCODE)**: 00 0000 0xx0 0000.
- **Operación**: Ninguna.
- **Descripción**: Esta instrucción no reliza ninguna ejecución, pero sirve para gastar un ciclo de máquina, equivalente a 4 de reloj.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**: Si usamos un cristal de cuarzo de 4 Mhz. en el oscilador, podremos obtener un retardo igual a un microsegundo por cada instrucción NOP que insertemos en el código del programa:

```
RETARDO    NOP
           NOP
           NOP
           RETURN
```

Cada vez que llamemos a la subrutina RETARDO, obtendremos 3 microsegundos de demora.

- **Ciclos de máquina**: 1

RLF

- **Acción**: Rota a la izquierda el registro f.
- **Sintaxis**: **RLF f,d**.
- **Funcionamiento**: Rotate Left through Carry f.
- **Hexadecimal**: 0D ff.
- **Bits (OPCODE)**: 00 1101 dfff ffff.
- **Operación**: $d\epsilon[0,1]$ (d puede ser W o f).
- **Descripción**: Esta instrucción rota a la izquierda todos los bits del registro direccionado por el parámetro f pasando por el bit CARRY del registro STATUS (desde los bits menos significativos a los más significativos).

  Es como si multiplicáramos por dos el contenido del registro.

  El bit D7 pasa al CARRY del registro STATUS, el contenido del CARRY pasa al D0, el D0 al D1, etc.
- **Registro STATUS**: Modifica el bit C (CARRY).
- **Ejemplo**: Tenemos el registro VALOR = 00000001 y aplicamos la instrucción

```
    RLF VALOR
```

entonces el resultado será VALOR = 00000010 y el bit C = 0.

Si tenemos el registro VALOR = 10000000 y aplicamos la instrucción

```
RLF VALOR
```

el resultado será VALOR = 00000000 y el bit C = 1.

- **Ciclos de máquina**: 1.

RRF

- **Acción**: Rota a la derecha el registro f.
- **Sintaxis**: **RRF f,d**.
- **Funcionamiento**: Rotate Right through Carry f.
- **Hexadecimal**: 0C ff.
- **Bits (OPCODE)**: 00 1100 dfff ffff.
- **Operación**: $d = f \gg 1$ (d puede ser W o f).
- **Descripción**: Esta instrucción rota a la derecha todos los bits del registro direccionado por el parámetro f pasando por el bit CARRY del registro STATUS (desde los bits más significativos a los menos significativos).

  Es como si dividiéramos por dos el contenido del registro.

  El bit C del registro STATUS pasa al D7, el D0 pasa al bit C, el D1 al D0, etc.

  El bit d determina el destino del resultado. Si no se pone nada, el valor por defecto es 1 y se guarda en f.
- **Registro STATUS**: Modifica el bit C (CARRY).
- **Ejemplo**: Si tenemos el registro VALOR = 00000001 y aplicamos la instrucción

```
RRF VALOR
```

entonces el resultado será VALOR = 00000000 y el bit C = 1.

Si tenemos el registro VALOR = 10000000 y aplicamos la instrucción

```
    RRF VALOR
```

el resultado será VALOR = 01000000 y el bit C = 0.

- **Ciclos de máquina**: 1.

SUBWF

- **Acción**: Resta el contenido del registro W el registro f.
- **Sintaxis**: **SUBWF f,d**.
- **Funcionamiento**: Subtract W from f.
- **Hexadecimal**: 02 ff.
- **Bits (OPCODE)**: 00 0010 dfff ffff.
- **Operación**: d = f W (d puede ser W o f).
- **Descripción**: Esta instrucción resta el valor contenido en el acumulador W del valor contenido en el registro direccionado por el parámetro f. El parámetro determina el destino. Si no se pone nada el valor por defecto será 1 y se almacenará en f.
- **Registro STATUS**: Modifica los bits Z, DC y C.
  - Z vale 1 si el resultado de la operación es 0.
  - DC vale 1 si el resultado de la operación es un número superior a 15.

  C vale 1 si el resultado de la operación es positivo o el bit 7 del registro que contiene el resultado vale 0. En caso contrario C vale 0 (resultado negativo).
- **Ejemplo**: Según sean los valores de W y el registro DATO, si aplicamos

```
    SUBWF DATO
```

obtendremos diferentes resultados en el bit CARRY.

Si DATO = 3 y W = 2; el resultado será DATO = 1 y C = 1.

Si DATO = 2 y W = 2; el resultado será DATO = 0 y C = 1.

Si DATO = 1 y W = 2; el resultado será DATO = FFH y C = 0.

Vemos que C = 1 porque el resultado es positivo y C = 0 cuando el resultado es negativo.

- **Ciclos de máquina**: 1.

SWAPF

- **Acción**: Invierte los dos nibbles que forman un byte dentro de un registro.
- **Sintaxis**: **SWAPF f,d**.
- **Funcionamiento**: Swap nibbles in f.
- **Hexadecimal**: 0E ff.
- **Bits (OPCODE)**: 00 1110 dfff ffff.
- **Operación**: f = 0123 SWAP 4567 de f.
- **Descripción**: Esta instrucción intercambia el valor de los 4 bits más significativos (D7-D4) contenidos en el registro f, con los 4 bits menos significativos (D3-D0) del mismo. El parámetro d determina el destino. Si no se pone nada, el valor por defecto es 1 y se guarda en f.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**: Tenemos VALOR = 00001111 y W = 00000000. Aplicamos la instrucción SWAPF

```
SWAPF VALOR          ; VALOR = 11110000B y W = 00000000B.
SWAPF VALOR, W       ; VALOR = 00001111B y W = 11110000B.
```

Con la primera instrucción modificamos el valor del registro DATO, y en la segunda instrucción modificamos el valor del acumulador sin que varíe el registro DATO.

- **Ciclos de máquina**: 1.

XORWF

- **Acción**: Operación lógica OR-Exclusiva.
- **Sintaxis**: **XORWF f,d**.
- **Funcionamiento**: Exclusive OR W with f.
- **Hexadecimal**: 06 ff.
- **Bits (OPCODE)**: 00 0110 dfff ffff.
- **Operación**: d = W OR f.
- **Descripción**: Esta instrucción efectúa la operación lógica XOR (OR exclusivo) entre el valor contenido en el acumulador W y el valor contenido en el registro direccionado por el parámetro f. El parámetro d determina el destino. Si no se pone nada el valor por defecto es 1 y se guarda en f.
- **Comentario**: La salida únicamente se pondrá a nivel alto cuando las dos entradas sean distintas. Esto es útil cuando tenemos una suma lógica en la que 1 + 1 es 10 y nos llevamos 1.

  Esta operación algebraicamente se expresa así:

  $S = f + W$
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**:

```
    XORWF REG1, 1
```

Antes de la instrucción:

REG1 = 0xAF

W = 0xB5

Después de la instrucción:

REG1 = 0x1A

W = 0xB5

- **Ciclos de máquina**: 1.

Instrucciones orientadas al manejo de Bits

BCF

- **Acción**: Pone a cero el bit b del registro f.
- **Sintaxis**: **BCF f,b**.
- **Funcionamiento**: Bit Clear f.
- **Hexadecimal**: 1b ff.
- **Bits (OPCODE)**: 01 00bb bfff ffff.
- **Operación**: F(b) = 0.
- **Descripción**: Esta instrucción pone a cero un bit que hayamos elegido de un registro determinado.
- **Comentario**: No debemos olvidar que en la numeración de los bits también se tiene en cuenta el 0. Si tratamos con un registro especial, podemos poner el nombre del bit correspondiente.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
BCF     PORTA,    RA4   ; pone a 0 el bit RA4 del registro PORTA
BCF     PORTA,    4     ; igual, si no conocemos en nombre del
                        ; bit
```

Si en el PORTA tenemos como valor inicial 11111111, después de aplicar el ejemplo anterior, PORTA = 11101111.

- **Ciclos de máquina**: 1.

BSF

- **Acción**: Pone a uno el bit b del registro f.
- **Sintaxis**: **BSF f,b**.
- **Funcionamiento**: Bit Set f.
- **Hexadecimal**: 1b ff.
- **Bits (OPCODE)**: 01 01bb bfff ffff.
- **Operación**: F(b) = 1.
- **Descripción**: Esta instrucción pone a uno un bit que hayamos elegido de un registro determinado.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
    BSF     PORTA,    RA0   ; pone a 1 el bit RA0 del registro PORTA
    BSF     PORTA,    0     ; igual, si no conocemos en nombre del
                            ; bit
```

  Si en el PORTA tenemos como valor inicial 00000000, después de aplicar el ejemplo anterior, PORTA = 00000001.

- **Ciclos de máquina**: 1.

BTFSC

- **Acción**: Comprueba un bit b del registro f, y salta la instrucción siguiente si este es cero.
- **Sintaxis**: **BTFSC f,b**.
- **Funcionamiento**: Bit Test, Skip if Clear.
- **Hexadecimal**: 1b ff.
- **Bits (OPCODE)**: 01 10bb bfff ffff.

- **Operación**: F(b) = 0? SI, salta una instrucción.
- **Descripción**: Esta instrucción comprueba el valor del bit b en el registro f, y si b = 0 entonces se salta la siguiente instrucción. Si b = 1 no salta y sigue con su ejecución normal.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
BTFSC PORTA,2
INSTRUCCION 1
INSTRUCCION 2
```

  Si en PORTA tenemos como valor inicial 11111011, el programa continúa con la instrucción 2, saltándose la instrucción 1

  Si en PORTA tenemos el valor 00000100, el programa sigue con la instrucción 1 y después la instrucción 2

- **Ciclos de máquina**: 1 (2) Si tiene que saltar ocupa dos ciclos.

BTFSS

- **Acción**: Comprueba un bit b del registro f, y salta la instrucción siguiente si éste es uno.
- **Sintaxis**: **BTFSC f,b**.
- **Funcionamiento**: Bit Test, Skip if Set.
- **Hexadecimal**: 1b ff.
- **Bits (OPCODE)**: 01 11bb bfff ffff.
- **Operación**: F(b) = 1? SI, salta una instrucción.
- **Descripción**: Esta instrucción comprueba el valor del bit b en el registro f, y si b = 1 entonces se salta la siguiente instrucción. Si b = 0 no salta y sigue con su ejecución normal.
- **Registro STATUS**: No modifica ningún bit de estado.

- **Ejemplo**:

```
BTFSS PORTB, 7
INSTRUCCION 1
INSTRUCCION 2
```

  Si en PORTB tenemos como valor inicial 10000000, el programa continúa con la instrucción 2, saltándose la instrucción 1. Si tenemos el valor 01111111, el programa sigue con la instrucción 1 y después la instrucción 2.

- **Ciclos de máquina**: 1 (2) Si tiene que saltar ocupa dos ciclos.

Operaciones con literales y de control

ADDLW

- **Acción**: Suma a W un literal.
- **Sintaxis**: **ADDLW**.
- **Funcionamiento**: Add literal to W.
- **Hexadecimal**: 3E kk.
- **Bits (OPCODE)**: 11 111x kkkk kkkk.
- **Operación**: W = W + k.
- **Descripción**: Esta instrucción suma un valor de un literal al contenido del registro W y lo guarda en W.
- **Cometarios**: Es igual que su homólogo manejando registros.
- **Registro STATUS**: Modifica los bits Z, DC y C.
  - Z vale 1 si el resultado de la operación es 0.
  - DC vale 1 si el resultado de la operación es un número superior a 15.
  - C vale 1 si el resultado de la operación es positivo o el bit 7 del registro que contiene el resultado vale 0. En caso contrario C vale 0 (resultado negativo).

- **Ejemplo**:

```
MOVLW   3   ; carga el acumulador W con el valor 3.
ADDLW   1   ; suma 1 al acumulador.
```

Al final el acumulador tendrá el valor 4.

- **Ciclos de máquina**: 1.

ANDLW

- **Acción**: Realiza la operación AND entre un literal y W.
- **Sintaxis**: **ANDLW k**.
- **Funcionamiento**: AND W with k.
- **Hexadecimal**: 39 ff.
- **Bits (OPCODE)**: 11 1001 kkkk kkkk.
- **Operación**: W = W AND k.
- **Descripción**: Esta instrucción realiza una operación lógica AND entre el contenido de W y k. El resultado se guarda siempre en el acumulador W.
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**: Si cargamos el acumulador con el número binario 10101010 y hacemos un AND con el binario 11110000, nos quedará el resultado de la operación en el acumulador W.

```
MOVLW 10101010
ANDLW 11110000
```

El resultado de la operación queda en W = 10100000.

- **Ciclos de máquina**: 1.

CALL

- **Acción**: Llama a una subrutina en la dirección k.
- **Sintaxis**: **CALL k**.
- **Funcionamiento**: Call subroutine.
- **Hexadecimal**: 2k kk.
- **Bits (OPCODE)**: 10 0kkk kkkk kkkk.
- **Operación**: CALL $\mapsto$ k...RETURN $\mapsto$ PC+1.
- **Descripción**: Esta instrucción llama a un grupo de instrucciones (subrutina) que comienzan en la dirección k, donde k puede ser un valor numérico o una etiqueta. Siempre termina con la instrucción de retorno (RETURN o RETLW).

  Definición de subrutina: son un grupo de instrucciones que forman un programa dentro del programa principal y que se ejecutan cuando las llama el programa principal.

  Utilidad: sirven para utilizarlas varias veces en cualquier parte del programa, sin necesidad de tener que copiar las mismas instrucciones, con el consiguiente ahorro de memoria.

  Funcionamiento: cuando un programa ejecuta una instrucción CALL, guarda en el stack el valor del registro PC+1 (PC = Program Counter) de manera que al regresar de la subrutina continúa con la instrucción siguiente recuperándola del stack, ejecutando la instrucción de retorno RETURN o RETLW.

  Limitaciones: en el PIC16F84 tenemos disponibles 8 niveles de stack, por lo que el número máximo de CALL reentrantes (instrucciones CALL que contengan otra instrucción CALL) queda limitado a 8.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

  PRINCIPAL: etiqueta que identifica una dirección de memoria.

  RETARDO: etiqueta que identifica el comienzo de una subrutina.

  BUCLE: etiqueta que identifica una dirección de memoria.

```
PRINCIPAL CALL      RETARDO
          BTFSC     PORTB,    RB0
          GOTO      PRINCIPAL
           *
           *
           *
RETARDO   CLRF      CONTADOR BUCLE
          DECFSZ    CONTADOR, 1
          GOTO      BUCLE
          RETURN
```

En este listado vemos que la subrutina RETARDO salta a un grupo de instrucciones que forman un bucle y cuando éste termina regresa para seguir con la instrucción siguiente al salto (BTFSC...).

- **Ciclos de máquina**: 2.

CLRWDT

- **Acción**: Pone el temporizador WDT a cero.
- **Sintaxis**: **CLRWDT**.
- **Funcionamiento**: Clear WatchDog Timer.
- **Hexadecimal**: 00 64.
- **Bits (OPCODE)**: 00 0000 0110 0100.
- **Operación**: WDT = 0.
- **Descripción**: Esta instrucción se utiliza cuando programamos el PIC con la opción Watch Dog habilitada. Para evitar el reset del PIC, el programa debe contener cíclicamente la instrucción CLRWDT para ponerlo a cero. Si no se pone a cero a tiempo, el WDT interpretará que se ha bloqueado el programa y ejecutará un reset para desbloquearlo.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
Bucle       CLRWDT
              *
              *
              *
            GOTO     Bucle
```

- **Ciclos de máquina**: 1.

GOTO

- **Acción**: Salto incondicional a k.
- **Sintaxis**: **GOTO k**.
- **Funcionamiento**: Go to address (label).
- **Hexadecimal**: 28 kk.
- **Bits (OPCODE)**: 10 1kkk kkkk kkkk.
- **Operación**: Salto $\mapsto$ k.
- **Descripción**: Esta instrucción ejecuta un salto del programa a la dirección k. El parámetro k puede ser un valor numérico o una etiqueta.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
         INSTRUCCION 1
         GOTO ABAJO
         INSTRUCCION 3
         INSTRUCCION 4
         INSTRUCCION 5
ABAJO    INSTRUCCION 6
```

  Primero se ejecuta la instrucción 1, después GOTO y continúa con la instrucción 6 saltándose las instrucciones 3, 4 y 5.

- **Ciclos de máquina**: 2.

IORLW

- **Acción**: Operación lógica OR entre el acumulador y un literal.
- **Sintaxis**: **IORWF f,d**.
- **Funcionamiento**: Inclusive OR W with l.
- **Operación**: W = W OR l.
- **Descripción**: Esta instrucción realiza una operación lógica OR inclusivo entre el acumulador W y un literal. El resultado siempre se guarda en el acumulador.
- **Comentarios**: La operación OR inclusivo suele llamarse OR a secas, pero se pone así para diferenciarla de la Suma Exclusiva. La operación lógica que describe esta instrucción es esta:

  F + W = S

  Se puede ver que basta con que uno de los dos registros tenga un uno para que la salida sea un uno también.
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**:

```
    IORLW 0x35
```

  Antes de la instrucción:

  W = 0x9A

  Después de la instrucción:

  W = 0xBF

  Z = 0
- **Ciclos de máquina**: 2.

MOVLW

- **Acción**: Copia el contenido de un literal al acumulador.

- **Sintaxis**: **MOVLW f**.
- **Funcionamiento**: Move literal to W.
- **Hexadecimal**: 30 kk.
- **Bits (OPCODE)**: 11 00xx kkkk kkkk.
- **Operación**: W = f.
- **Descripción**: Esta instrucción asigna al acumulador W el valor del literal k, el cual debe estar comprendido entre 0 y 255.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**: Si tenemos el acumulador a cero o con cualquier valor, y queremos que contenga el que le asignemos nosotros directamente entonces usaremos esta instrucción: W = 0.

  Valor a asignar = 100.

  Instrucción:

```
    MOVLW 100
```

  El acumulador valdrá 100 (W = 100).

  Con distinto valor de partida del acumulador:

  W = 225.

```
    MOVLW 100
```

  El acumulador valdrá 100 (W = 100).

- **Ciclos de máquina**: 1.

RETFIE

- **Acción**: Retorno de la llamada a una subrutina.
- **Sintaxis**: **RETFIE**.

- **Funcionamiento**: Return From Interrupt.
- **Hexadecimal**: 00 09.
- **Bits (OPCODE)**: 00 0000 0000 1001.
- **Operación**: FIN INTERRUPCIÓN.
- **Descripción**: Esta instrucción devuelve el control al programa principal después de ejecutarse una subrutina de gestión de una interrupción.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
        ORG     00H
BUCLE   GOTO    BUCLE ; bucle infinito.
        ORG     04H   ; vector de interrupcion.
        RETFIE        ; retorna de la interrupcion
```

  Este código de programa ejecuta un bucle infinito. Si habilitamos una de las interrupciones del 16F84, en cuanto ésta se produzca pasará el control al programa situado en la dirección 04h y la instrucción RETFIE regresa de la interrupción.

  Al ejecutarse una interrupción, el bit GIE del registro INTCON se pone a 0 y así evita que otra interrupción se produzca mientras ya está con una en marcha.

  Con la instrucción RETFIE ponemos de nuevo el bit GIE a 1 para así atender de nuevo a futuras interrupciones.

- **Ciclos de máquina**: 2.

RETLW

- **Acción**: Retorno de subrutina y carga literal k en el acumulador.
- **Sintaxis**: **RETLW**.
- **Funcionamiento**: Return with Literal in W.
- **Hexadecimal**: 34 kk.
- **Bits (OPCODE)**: 11 01xx kkkk kkkk.

- **Operación**: RETORNO con W = k.
- **Descripción**: Esta instrucción retorna de una subrutina al programa principal, cargando el acumulador W con el literal k.

  Es la última instrucción que forma una subrutina, al igual que RETURN, con la diferencia que carga en W el valor de k.
- **Comentarios**: Nos será muy útil al programar con TABLAS.
- **Registro STATUS**: No modifica ningún bit de estado.
- **Ejemplo**:

```
            CALL      SUBRUT1    ; llama a Subrut1.
            MOVWF     DATO 1     ; carga W en Dato1.
            CALL      SUBRUT2    ; llama a Subrut2.
            MOVWF     DATO2      ; carga W en Dato2.
             *
             *
SUBRUT1     RETLW     0A         ; carga W = 0A y retorna.

SUBRUT2     RETLW     0B         ; carga W = 0B y retorna.
```

- **Ciclos de máquina**: 2.

RETURN

- **Acción**: Retorno de una subrutina.
- **Sintaxis**: **RETURN**.
- **Funcionamiento**: Return from subroutine.
- **Hexadecimal**: 00 08.
- **Bits (OPCODE)**: 00 0000 0000 1000.
- **Operación**: RETORNO.
- **Descripción**: Esta instrucción retorna de una subrutina al programa principal en la instrucción siguiente a la llamada de la subrutina, tomando el valor almacenado en el stack para continuar.

  Es la última instrucción que forma una subrutina (al igual que RETLW).

- **Comentarios**: El procedimiento es siempre el mismo. Así, tenemos que crear la subrrutina y darle el nombre para poder ser llamada; al final de la subrrutina se debe escribir la instrucción denominada RETURN. Entonces podemos concluir que una subrrutina esta constituida por un conjunto de instrucciones demarcadas por un nombre que se encuentra al inicio y la instrucción RETURN que se encuentra al final demarcando el final de la subrrutina.

  Estos mismos pasos debemos seguirlos para la instrucción RETLW

- **Registro STATUS**: No modifica ningún bit de estado.

- **Ejemplo**:

```
          CALL  COMPARA        ; llama a Compara.
          INSTRUCCION1         ; vuelve aqui cuando se
          INSTRUCCION2         ; ejecuta return
                *
                *
COMPARA   INSTRUCCION R1
          INSTRUCCION R2
          RETURN
```

  Aquí llamamos a la subrutina COMPARA, se ejecutan las instrucciones R1 y R2 y con el RETURN regresa a la instrucción siguiente al CALL y ejecuta las instrucciones 1, 2 y sigue con el programa.

- **Ciclos de máquina**: 2.

SLEEP

- **Acción**: Paso a modo de bajo consumo.

- **Sintaxis**: **SLEEP**.

- **Funcionamiento**: Go into Standby Mode.

- **Hexadecimal**: 00 63.

- **Bits (OPCODE)**: 00 0000 0110 0011.

- **Operación**: EN ESPERA.

- **Descripción**: Esta instrucción detiene la ejecución del programa, deja el PIC en modo suspendido y el consumo de energía es mínimo.

  No ejecuta ninguna instrucción hasta que sea nuevamente reinicializado (reset) o surja una interrupción.

  Durante este modo, el contador del Watch Dog sigue trabajando, y si lo tenemos activado el PIC se reseteará por este medio.

- **Registro STATUS**: No modifica ningún bit de estado.

- **Ciclos de máquina**: 1.

SUBLW

- **Acción**: Resta al literal k el valor del acumulador.

- **Sintaxis**: **SUBLW k**.

- **Funcionamiento**: Substract W from Literal.

- **Hexadecimal**: 3C kk.

- **Bits (OPCODE)**: 11 110x kkkk kkkk.

- **Operación**: W = k - W.

- **Descripción**: Esta instrucción resta al literal k el valor almacenado en W y el resultado se guarda en el acumulador.

- **Registro STATUS**: Modifica los bits Z, DC y C.

  - Z vale 1 si el resultado de la operación es 0.
  - DC vale 1 si el resultado de la operación es un número superior a 15.
  - C vale 1 si el resultado de la operación es positivo o el bit 7 del registro que contiene el resultado vale 0. En caso contrario C vale 0 (resultado negativo).

- **Ejemplo**:

```
    MOVLW   10      ; carga el acumulador W con el valor 10.
    SUBLW   15      ; resta a 15 el valor del acumulador.
```

Al final el acumulador tendrá el valor W = 5.

- **Ciclos de máquina**: 1.

XORLW

- **Acción**: Operación lógica OR exclusivo entre el acumulador y el literal k.
- **Sintaxis**: **XORLW k**.
- **Funcionamiento**: Exclusive OR Literal with W.
- **Hexadecimal**: 3A kk.
- **Bits (OPCODE)**: 11 1010 kkkk kkkk.
- **Operación**: W = W XOR k.
- **Descripción**: Esta instrucción realiza un OR exclusivo entre el contenido del acumulador W y el valor del literal k. El resultado se guarda siempre en el acumulador (recuerda que k es un literal, no un registro).
- **Comentarios**: La salida únicamente se pondrá a nivel alto cuando las dos entradas sean distintas. Esto es útil cuando tenemos una suma lógica en la que 1 + 1 es 10 y nos llevamos 1.

  Esta operación algebraicamente se expresa así:

  S = f + W
- **Registro STATUS**: Modifica el bit Z.
  - Z vale 1 si el resultado de la operación es 0.
- **Ejemplo**:

```
    XORLW 0xAF
```

  Antes de la instrucción:

  W = 0xB5

  Después de la instrucción:

  W = 0x1A
- **Ciclos de máquina**: 1.

Capítulo 2

Hardware

2.1. Introducción

El sistema hardware consta de dos placas funcionalmente diferentes, que se denominan TIPO1 y TIPO2. La lista de funcionalidades ofrecidas en total por ambos circuitos en conjunto es la siguiente:

- Interfaz IR (Infrarrojos).
- Interfaz serie RS-232 (Permite la comunicación con el LCD de TECDIS o con el PC).
- Interfaz serie I2C.
- Interfaz paralela.
- Interfaz LCD 16x2.
- Interfaz actuadores de potencia: relé, triacs.

En las secciones 2.2 y 2.3 se estudian con detalle las placas TIPO1 y TIPO2 respectivamente.

En los siguientes apartados se pasa a analizar con todo detalle la inclusión de los elementos hardware que forman parte tanto de la placa TIPO1 en los dos montajes posibles como de la placa TIPO2. El receptor de IR, el PIC16F628, la circuitería asociada al puerto serie RS-232 y las conexiónes de estos elementos son comunes a los tres circuitos.

2.2. Placa TIPO 1

2.2.1. Funcionalidades

La placa TIPO1 tiene una característica que no posee la placa TIPO2. Esta placa permite dos montajes diferentes dependiendo de cuáles sean las funcionalidades buscadas, es decir, hay dos circuitos diferentes que se montan sobre la misma placa de circuito impreso. Se puede ver su diagrama de bloques en la figura 2.1.

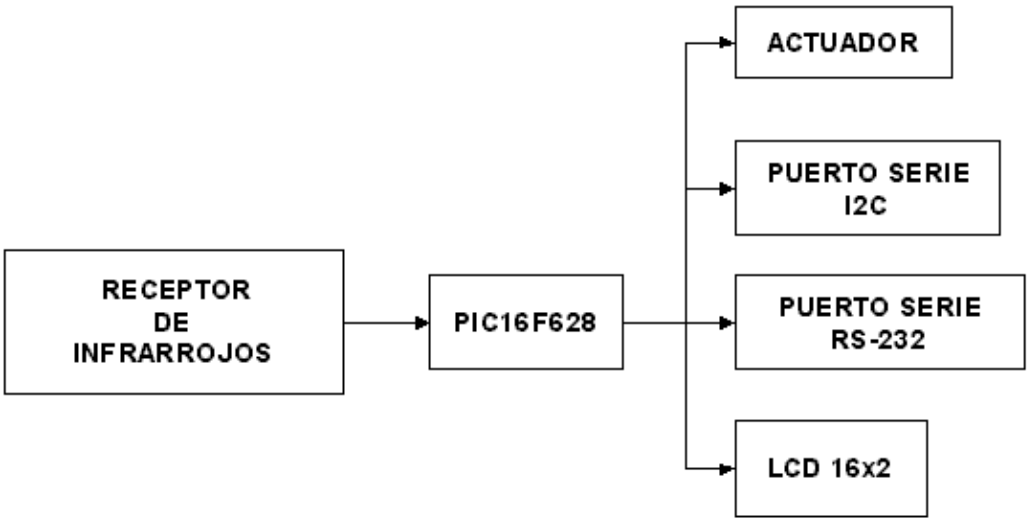


Figura 2.1: Diagrama de bloques de la placa TIPO 1

De los dos montajes posibles, el que menos componentes requiere es el correspondiente al circuito que permite la definición del contenido del display LCD de TECDIS; se trata de la aplicación práctica que será estudiada en el capítulo 4.

2.2.2. Circuito con puerto serie I2C, puerto para conectar LCD 16x2 y puerto para actuadores

El otro circuito ligado a la placa TIPO1, requiere de una programación no menos sencilla de su elemento controlador, que es como ya sabemos el PIC16F628, porque aunque no permite la programación del LCD de TECDIS añade otras funcionalidades de gran utilidad. Se puede resumir el funcionamiento de este montaje diciendo que se limita a recibir las señales de infrarrojos, y que una vez interpretadas, envía los códigos binarios asociados a estas señales a través de sus puertos. Vemos detallados a continuación todos los elementos que conforman este circuito:

Puerto receptor de infrarrojos IS1U60

La patilla de la señal de salida del receptor de IR IS1U60 va conectada a la patilla RB0 del puerto B del PIC16F628. El hecho de realizar la conexión entre estas patillas se debe a que de esta forma el receptor, cada vez que es estimulado con luz infrarroja, provoca en el PIC una interrupción para que éste reciba y trate la señal de tensión correspondiente.

En la alimentación al circuito IS1U60 se coloca un filtro formado por un condensador electrolítico de $47\mu F$ y una resistencia de 47Ω entre Vcc y la patilla de alimentación positiva del receptor.

Puerto serie RS-232

Gracias al puerto serie RS-232, el circuito que nos ocupa, permite la comunicación del módulo receptor de IR con el PC. Esto puede ser muy útil para el control de programas, como por ejemplo el reproductor de música, mediante el empleo de un mando a distancia.

Se utiliza el montaje básico empleado en cualquier puerto serie de un microcontrolador. Este montaje es el que se describió en el apartado 1.4.2. Como el PIC16F628 dispone de un USART interno la conexión con el conversor de niveles MAX232 se realiza exactamente igual que en el caso de la figura 1.17 de la sección que se ha mencionado anteriormente, la 1.4.2. La patilla RB2 del micro es la de transmisión del USART y se conecta con la patilla de recepción de señales TTL del MAX232 T1IN; RB1, que es el pin encargado de la recepción, está conectado con T1OUT.

El C.I. MAX232 empleado es del tipo SMD; requiere soldadura superficial permitiendo así un ahorro considerable de espacio al ir situado éste en la cara de soldadura y no requerir orificios para sus patillas. Los chips SMD tienen buenas propiedades mecánicas y eléctricas pero son más complicados de soldar.

Puerto serie I2C

Proporciona la posibilidad de comunicación con otros dispositivos esclavos conectados a este puerto.

Gracias a que el PIC16F628 dispone de una patilla RA4 de tipo drenador abierto, I2C solo necesita 2 de las patillas de este PIC para su implementación. Será la línea SDA la que vaya conectada con el pin drenador abierto; para poder colocar un '1' en esta línea hay un camino de corriente hasta Vcc a través de una resistencia de $5{,}6K\Omega$. La utilización de la patilla con drenador abierto como elemento de unión a la línea de datos SDA posibilita la conexión en paralelo de varios esclavos, pero manteniendo siempre como maestro a nuestro sistema. El reloj es proporcionado al bus a través de la patilla RA3.

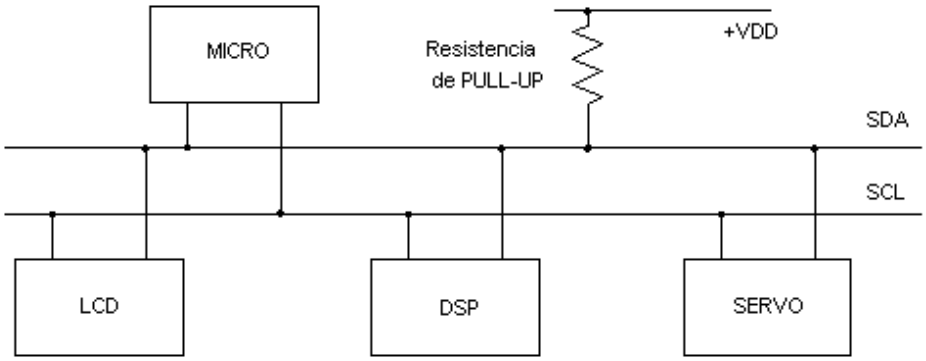


Figura 2.2: Bus I2C

Puerto de comunicación con LCD 16x2

Para la elección de la interfaz del LCD 16x2 han sido determinante las limitaciones impuestas por el escaso número de pines disponibles del PIC. Por este motivo se utiliza un bus de datos de 4 bits. Este bus de datos, **D4-D7** va conectado a la parte alta del puerto B del microcontrolador. Las patillas de datos **D0-D3** se conectan a Vss.

La patilla de habilitación del LCD, **E**, está unida a la patilla RB3 del LCD y la de selección de comando o dato (**RS**) a RA1.

La línea de escritura/lectura se pone a '0' ya que el display se utilizará sólo para escribir; en ningún caso será necesario leer el contenido del mismo.

Otra línea, que también se conecta a 0V es la de contraste, para conseguir un contraste máximo.

Actuadores

Son dos pines que podrían utilizarse como actuadores de potencia; se trata de las patillas del puerto A del PIC RA0 y RA2. El pin RA0 es el que se utiliza como botón de selección del modo de funcionamiento en la aplicación práctica. Esto se verá más adelante.

PIC16F628

El PIC interpreta las señales recibidas desde el puerto de infrarrojos por la patilla RB0 y envía el código binario correspondiente a través de los puertos I2C y RS-232. Además muestra los bits de la señal en formato hexadecimal en el LCD 16x2.

Se utiliza un chip de tipo SMD, como ocurría en el caso del MAX232, para reducir el tamaño de la placa. El aspecto de la placa con los chips SMD se observa en la fotografía de la figura 2.3.

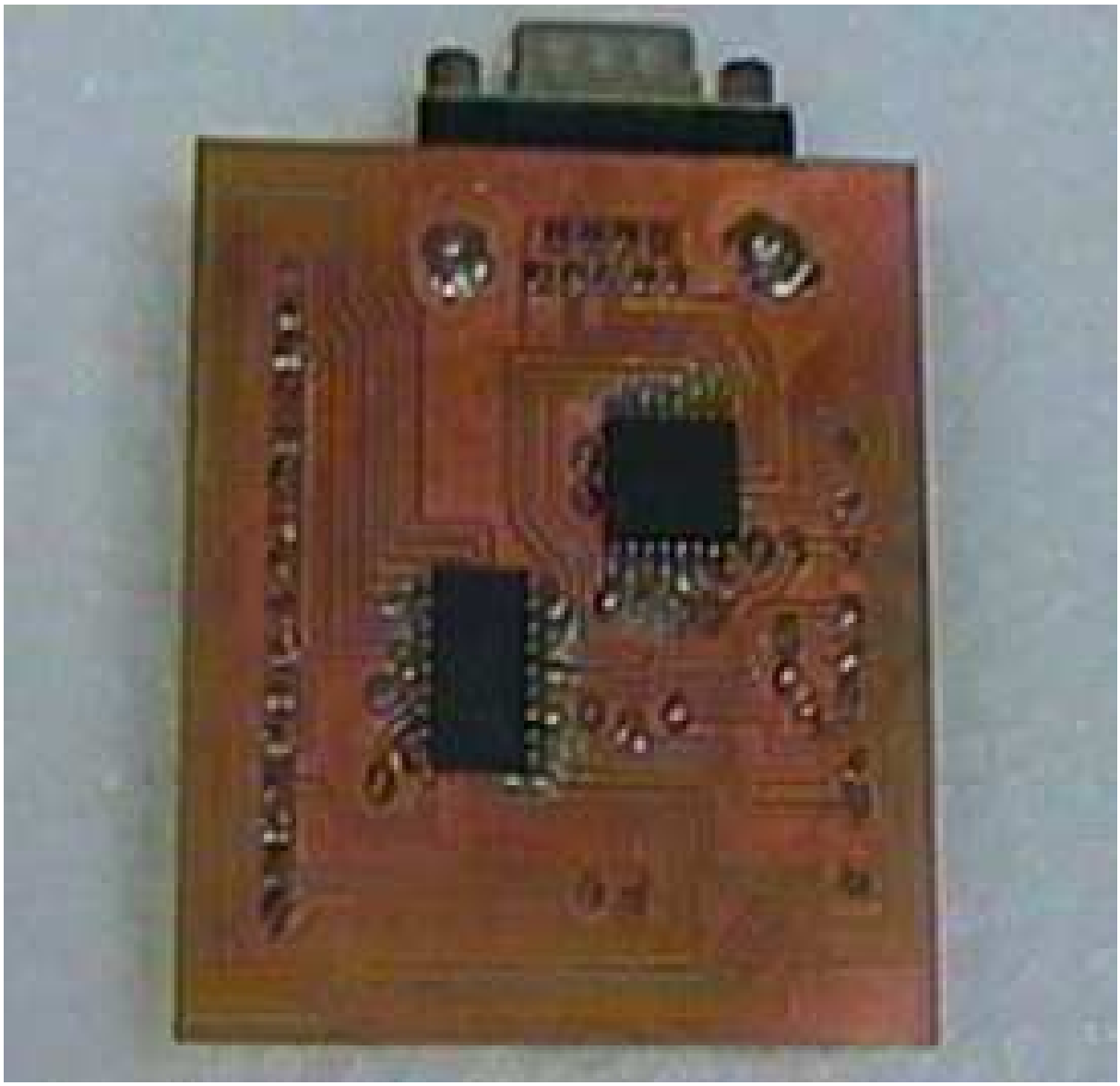


Figura 2.3: Vista inferior de la placa TIPO1

El esquema eléctrico en el que se pueden apreciar todas las conexiones es el de la figura 2.4; en la fotografía de la figura 2.5 aparece el circuito correspondiente al segundo montaje de la placa TIPO1.

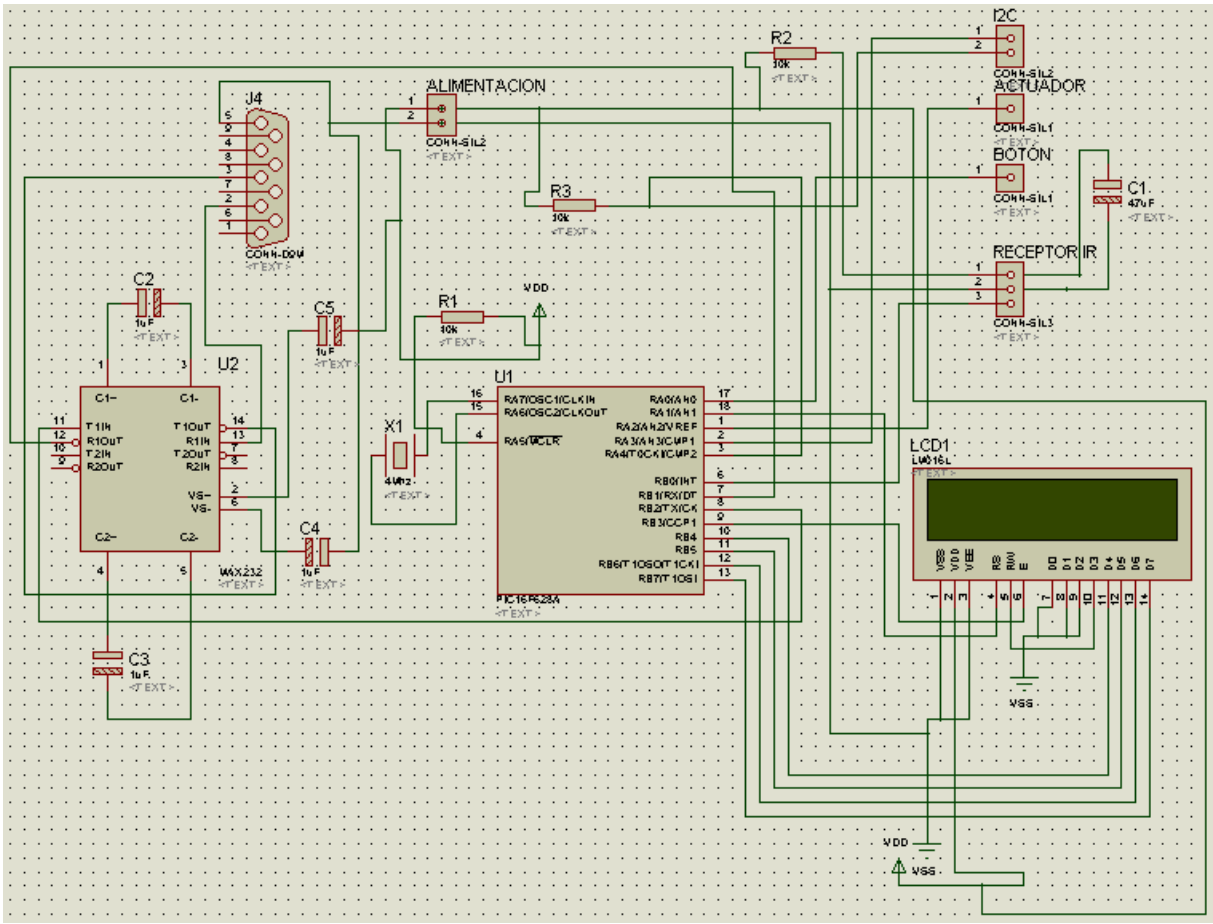


Figura 2.4: Esquema eléctrico de la placa TIPO1

2.3. Placa TIPO 2

2.3.1. Funcionalidades

La placa TIPO2 es la de menor tamaño de los dos tipos existentes. La funcionalidad exclusiva de esta placa y que marca la diferencia con el TIPO1 es que dispone de un puerto paralelo. El diagrama de bloques de este circuito es el que se presenta en la figura 2.6.

Vamos a ver con más profundidad los detalles hardware de este circuito en las siguientes líneas.

Puerto receptor de infrarrojos IS1U60

Al igual que en la placa TIPO1, la patilla de la señal de salida del receptor de IR IS1U60 va conectada a la patilla RB0 del puerto B del PIC16F628 para provocar en éste

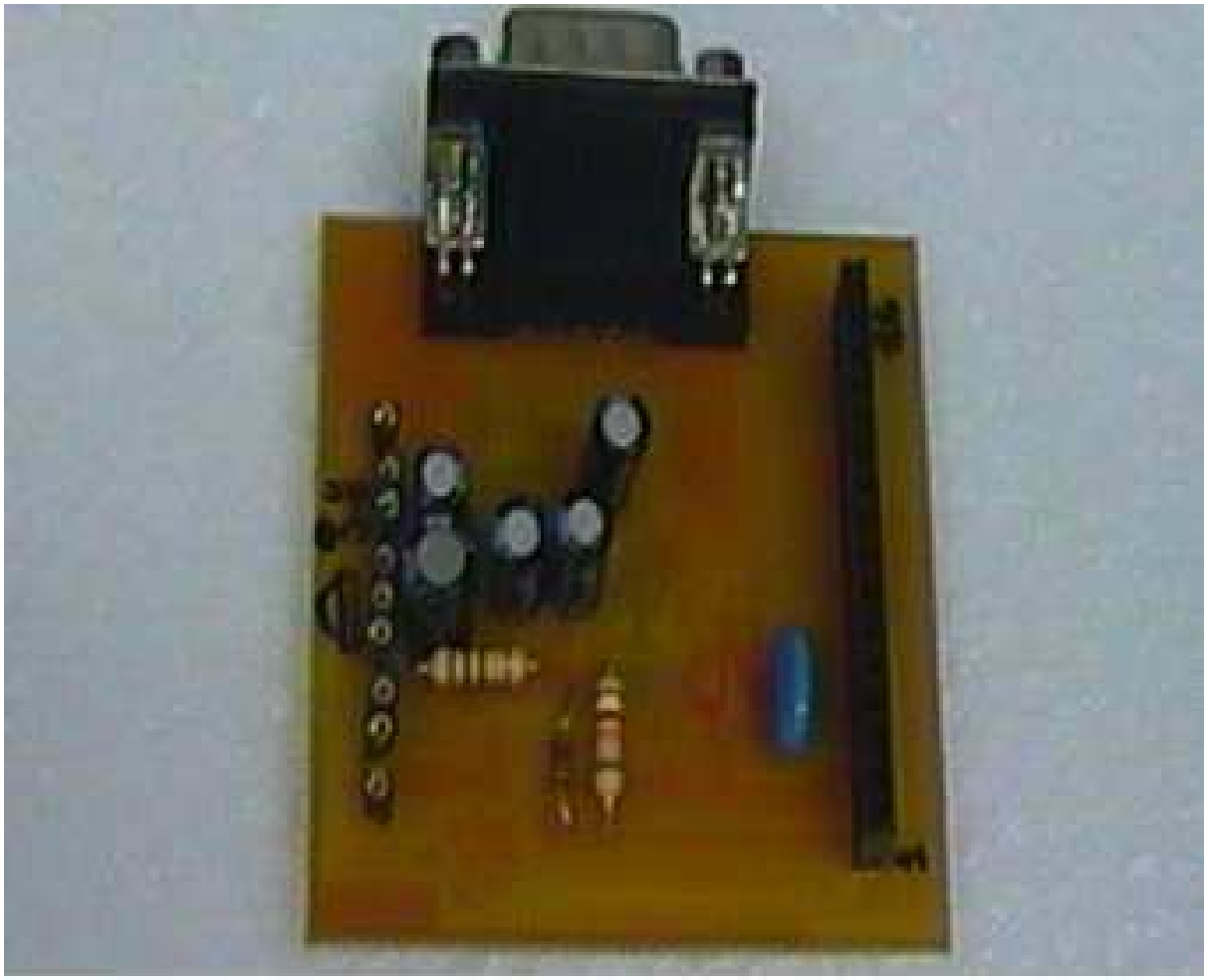


Figura 2.5: Fotografía de la placa TIPO1 con el montaje que contiene el puerto serie I2C

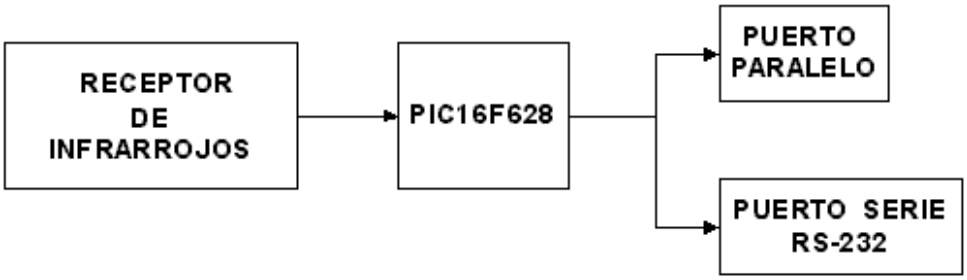


Figura 2.6: Diagrama de bloques de la placa TIPO 2

una interrupción cada vez que llega una nueva señal.

Puerto serie RS-232

Todo el hardware relacionado con este puerto serie es exactamente el mismo que el que se vió en la sección 2.2.2, motivo por el cual aquí no es necesario añadir nada más.

Puerto serie Paralelo

Para facilitar el manejo de datos por parte del programa correspondiente al puerto paralelo se han diseñado las conexiones de una manera particular. La parte alta del dato de 8 bits es propocionada por la parte baja del puerto A del microcontrolador, es decir, las patillas RA0-RA3 conforman la parte alta del bus de datos. La parte baja de este bus, los 4 bits de menos peso, se corresponden con la parte alta del puerto B (RB4-RB7).

Como el puerto paralelo necesita 2 líneas adicionales que se encargen de las tareas de control de la comunicación, se utilizan las patillas RA4 y RB3. Es la patilla RA4, la seleccionada como señal de salida de control proveniente del PIC. No hay que olvidar que esta patilla es de tipo drenador abierto y necesita por lo tanto una resistencia de PULL-UP para poder poner un '1'. La resistencia de PULL-UP utilizada para conectar RA4 con Vcc tiene un valor de 5,6$K\Omega$. El pin RB3 actuará como entrada de la señal de control proveniente del circuito esclavo.

El sistema esclavo empleado para conectar en el puerto paralelo es un PIC16F84A, pero se podía haber empleado otro dispositivo diferente. Este PIC recibe los datos del puerto paralelo y los muestra en un LCD 16x2. Como el número de patillas no es suficiente para que las conexiones del LCD y las del puerto paralelo sean independientes en el PIC16F84A, se colocan en las líneas compartidas resistencias de 100Ω situadas entre los pines de conexión del puerto paralelo del módulo IR y los cables de las conexiones del LCD. De esta forma se evitan posibles cortocircuitos accidentales aunque durante las operaciones de manejo del LCD por parte del PIC16F84A, el PIC16F628 coloca todas la líneas del puerto paralelo en alta impedancia para evitar que esto suceda.

PIC16F628

En la placa TIPO2 también se utiliza un PIC16F628 de tipo SMD. Es gracias al empleo de este tipo de chips, el motivo por el cual se conseguido un considerable reducido tamaño.

En la figura 2.7 aparece el circuito de la placa TIPO2 conectado a un circuito esclavo controlado por un PIC16F84; el circuito maestro manda los códigos de las señales IR al esclavo por el puerto paralelo que los une, para que éste último los muestre en hexadecimal en el LCD 16x2 que tiene conectado.

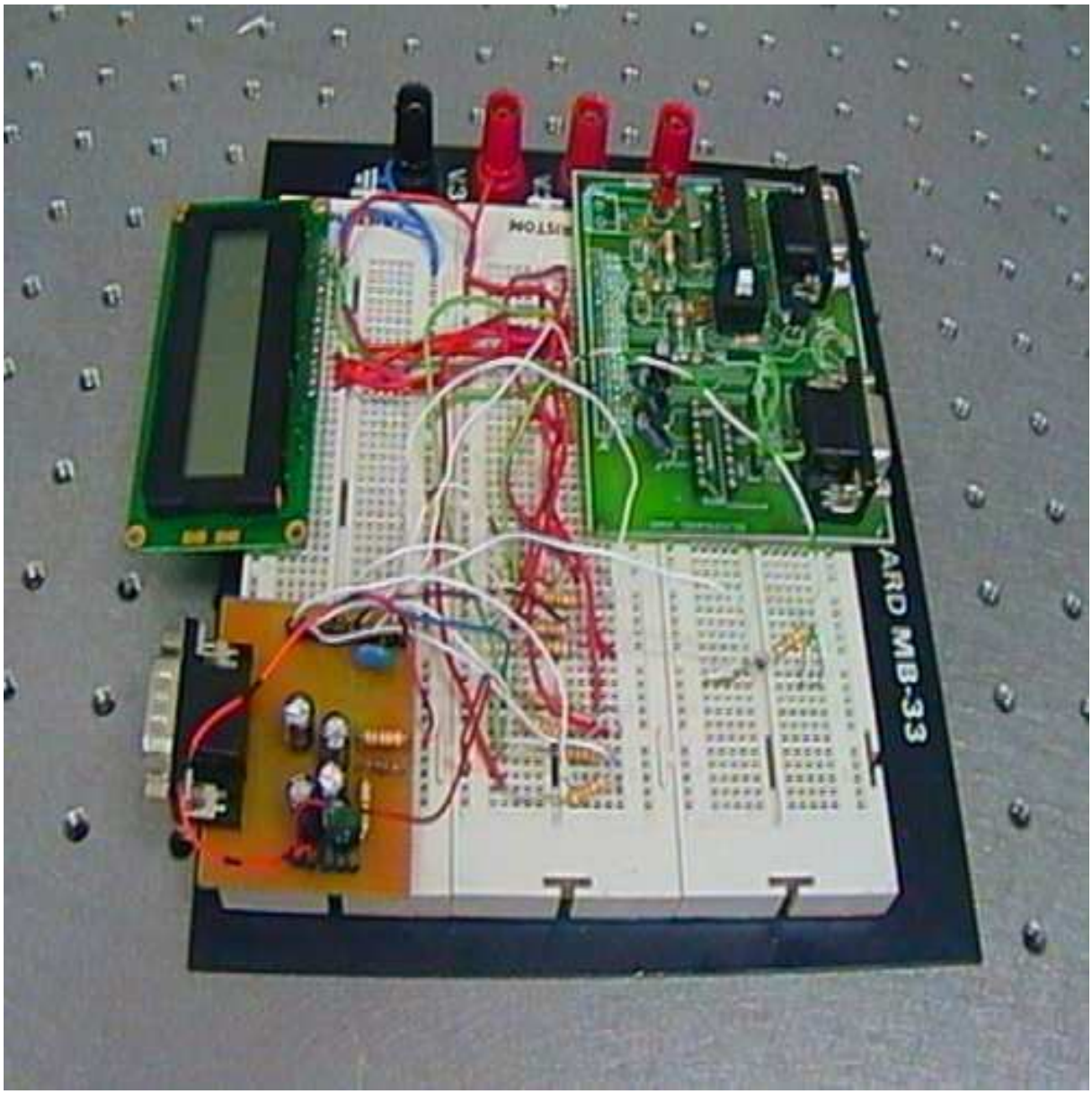


Figura 2.7: Placa TIPO2 conectada a un esclavo con el puerto paralelo

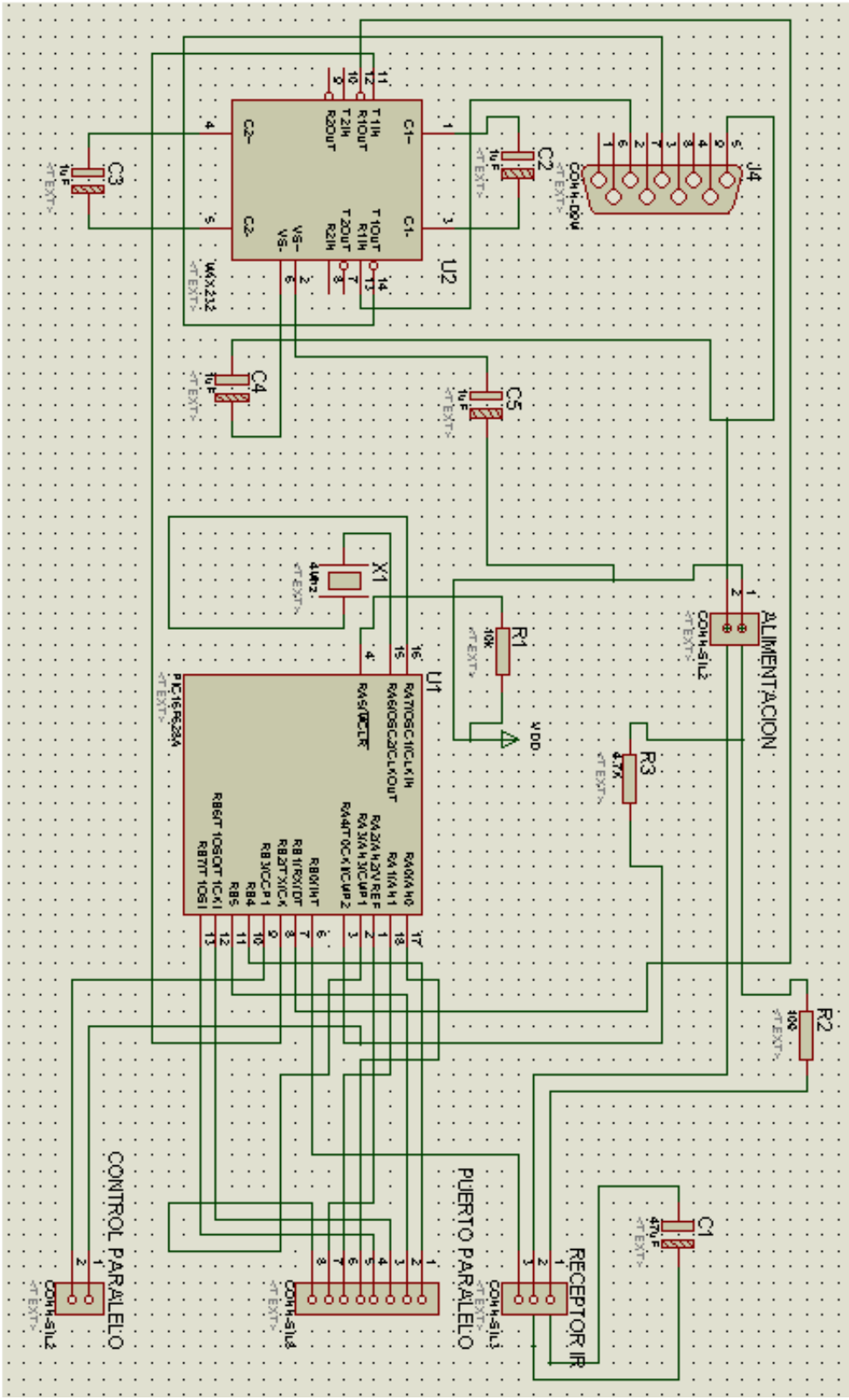


Figura 2.8: Esquema eléctrico de la placa TIPO2

Capítulo 3

Software

3.1. Software de recepción de la señal

En proceso de recepción de la señal se puede resumir en estos 3 pasos:

- Medida de los tiempos asociados a la señal proveniente del receptor IR en la patilla RB0 del PIC.
- Reconocimiento del tipo de codificación de la señal.
- Conversión a bits.

Cada uno de estos pasos lo vamos a estudiar de forma detallada.

El diagrama de flujo de la figura 3.1 describe el proceso a seguir para convertir la señal obtenida después de la recepción de la señal de infrarrojos a una secuencia de bits.

3.1.1. Toma de tiempos

El proceso de almacenamiento de las medidas temporales asociadas a la señal comienza cuando a la entrada del PIC cae la tensión a 0V (El valor de la tensión de salida del receptor IS1U60 en ausencia de señal es de 5V). Con esta bajada de la tensión en la entrada de la patilla RB0 del PIC se provoca una interrupción que obliga al contador de programa a saltar a la dirección 0x004 para realizar la correspondiente atención de la interrupción. A partir de este momento se van midiendo los tiempos que la señal permanece a nivel bajo y luego a nivel alto de forma sucesiva.

Para realizar las mediciones temporales se emplea el temporizador TMR0 con una preescala de valor 1:8 de forma que se pueden tomar medidas de valor máximo igual a $2048\mu s$ aunque los tiempos existentes en este tipo de señales son sensiblemente menores.

Si la señal viene precedida de una cabecera con tiempos de una duración superior al valor máximo de cuenta del TMR0 (2048μs) ésta se salta porque son tiempos que no es necesario tener en cuenta. Para pasar por alto esta cabecera se espera hasta la aparición de un nuevo '0' en la señal de entrada.

Para el almacenamiento de los tiempos se utilizan los registros de propósito general de la RAM y los valores son almacenados mediante direccionamiento indirecto, es decir, usando los registros INDF y FSR .

La toma de tiempos finaliza según dos situaciones diferentes:

- Si se trata del primer '1' recibido de la señal y ya ha superado la cuenta máxima establecida del TMR0, que es 2048μs como ya sabemos, pasa a considerarse cabecera. En la parte de código encargada del tratamiento de la cabecera se establece un valor de la preescala del TMR0 de 1:64. De esta forma el TMR0 puede realizar una cuenta máxima de 16384μs. El hecho de aumentar el tiempo de cuenta permite saber si la duración del '1' es excesiva, cosa que indicaría que el final de la señal ya ha ocurrido. Esta situación sería propia de una señal de indicación de mantenimiento de tecla pulsada.

- Si se ha realizado la toma de más de una medida de señal '0', en el momento en el que el '1' en proceso de medición supere un valor de 2048μs se considera el fin de la señal. Evidentemente, este último '1' que no se puede medir no se tiene en cuenta.

Vemos directamente en las siguientes líneas el código correspondiente a la toma de tiempos ya que posee una sencilla estructura:

```
    ; Comienza la toma de tiempos.
    ; El tiempo maximo de referencia para tomar medidas 2048us.
    ; Preescala del TMR0 = 8 (cada cuenta seran 8us).

    movlw   Direccion_ceros          ; Primera posicion de los ceros.
    movwf   FSR_TEMP1
    movlw   Direccion_unos           ; Primera posicion de los unos.
    movwf   FSR_TEMP2

otro_tiempo:
    bcf     INTCON,T0IF              ; Borra el T0IF.
    clrf    TMR0                     ; Borra el temporizador.
    call    PREESCALA_1              ; Valor de la preescala 8.

cambia_a_uno:
    btfsc   INTCON,T0IF              ; Comprueba si desbordo.
    goto    es_cabecera              ; Hay un nivel bajo muy largo ->
                                     ; cabecera.
    btfss   PORTB,0                  ; Comprueba la entrada.
    goto    cambia_a_uno             ; Repite mientras es cero.
```

```
    movfw   FSR_TEMP1
    movwf   FSR                         ; Inicializacion de FSR: apunta
                                        ; a la direccion de los ceros.
    movfw   TMR0                        ; Guarda el valor del TMR0.
    movwf   INDF
    incf    nummedidascero              ; Una medida cero mas.
    incf    FSR                         ; Apunta a la siguiente direccion.
    movfw   FSR
    movwf   FSR_TEMP1                   ; Vuelve a guardar el FSR.

    bcf     INTCON,T0IF                 ; Borra el T0IF.
    clrf    TMR0                        ; Borra el temporizador.
    call    PREESCALA_1                 ; Preescala 8.

cambia_a_cero:
    btfsc   INTCON,T0IF                 ; Comprueba si desbordo.
    goto    Comprueba_cab_fin
    btfsc   PORTB,0                     ; Comprueba la entrada.
    goto    cambia_a_cero               ; Repite mientras es cero.
    movfw   FSR_TEMP2
    movwf   FSR                         ; Inicializacion de FSR: apunta
                                        ; a la direccion de los unos.
    movfw   TMR0
    movwf   INDF                        ; La direccion apuntada por FSR
                                        ; pasa a contener el valor
                                        ; temporal de un uno.
    incf    nummedidasuno               ; Un uno mas.
    incf    FSR                         ; Apunta a la siguiente direccion.
    movfw   FSR
    movwf   FSR_TEMP2                   ; Se vuelve a guardar el FSR.
    goto    otro_tiempo

Comprueba_cab_fin:
    movlw   1
    subwf   nummedidascero,0            ; numceros - 1.
    btfsc   STATUS,Z
    goto    es_cabecera2                ; Un solo cero, estamos en cabecera.
    goto    fin_de_trama

es_cabecera:

   ; Espera a que pasen el '0' y el '1' de la cabecera.

cab_pasa_a_uno:
   btfss   PORTB,0                      ; Comprueba la entrada.
   goto    cab_pasa_a_uno

   bcf     INTCON,T0IF                  ; Borra el T0IF.
   clrf    TMR0                         ; Borra el temporizador.
   call    PREESCALA_2                  ; Preescala 64 para comprobar
                                        ; si la senal se acabo o no.
cab_sigue_a_uno:
   btfsc   INTCON,T0IF
   goto    fin_de_trama                 ; Si desborda se acabo la trama.
   btfsc   PORTB,0                      ; Se comprueba la entrada hasta
                                        ; que cambia a cero.
   goto    cab_sigue_a_uno
   goto    otro_tiempo                  ; Despues de evitar la cabecera
                                        ; toma medidas de los tiempos.
```

```
es_cabecera2:
   decf    nummedidascero
   decf    FSR_TEMP1                  ; Vuelve a apuntar a la primera
   ; posicion de los ceros.
   bcf     INTCON,TOIF                ; Borra el TOIF.
   clrf    TMR0                       ; Borra el temporizador.

   call    PREESCALA_2

cab_pasa_a_cero:
   btfsc   INTCON,TOIF
   goto    fin_de_trama               ; Si desborda se acabo la trama.
   btfsc   PORTB,0                    ; Comprueba la entrada.
   goto    cab_pasa_a_cero
   goto    otro_tiempo

fin_de_trama:
```

3.1.2. Reconocimiento del tipo de codificación de la señal y conversión a bits de la señal

El siguiente paso necesario para obtener la señal binaria buscada es el de reconocimiento de la codificación empleada en la señal. Durante este proceso se averigua si la señal es *Space Coded*, *Pulse Coded* o bien *Shift Coded*. El algoritmo requerido para reconocer una señal que sea de un tipo de entre los dos primeros mencionados (señales que llevan la información en la modulación del tiempo) es prácticamente el mismo; sin embargo una señal *Shift Coded* posee una estructura totalmente diferente al llevar la información en la fase.

Reconocimiento de Pulse Coded y Space Coded

Estos dos tipos de codificación tienen un punto en común. Ambas tienen o bien todas las medidas '0' o por el contrario todas las medidas '1' iguales exceptuando, por supuesto, la cabecera si la hay (la cabecera no se tiene en cuenta en la toma de tiempos). Es la señal de codificación por espacios la que cumple que todos sus ceros son iguales y la de codificación por pulsos la que posee todos los unos de la misma magnitud. Esto es así porque no hay que olvidar que el receptor IS1U60 proporciona una señal de tensión con niveles invertidos.

Aunque se ha dicho, según corresponda, que todos los unos o los ceros son iguales, esto no es realmente así ya que existen inexactitudes en todos los valores temporales. Un motivo capaz de distorsionar en cierta medida una señal IR es el rebote. Debido a estas inexactitudes es necesario tener en cuenta ciertas tolerancias a la hora de reconocer el

tipo de codificación y en el momento de la conversión a bits. El valor de la tolerancia que establece el límite entre considerar una determinada medida como tiempo de referencia (el valor teórico es 'T') o bien como tiempo extendido consecuencia de la modulación (valores típicos teóricos '2T' y '3T') es la mitad del valor temporal mínimo obtenido de entre todas las mediciones.

Una vez reconocida la codificación se obtiene la señal binaria que corresponda. En esta operación se ponen en juego los tiempos mínimos obtenidos para el cálculo de las tolerancias.

Si es una señal *Space Coded*:

- Bit '1' si: $MEDIDAUNO > tolerancia+$ MEDIDAUNOPEQUEÑA
- Bit '0' si: $MEDIDAUNO \leq tolerancia+$ MEDIDAUNOPEQUEÑA

Si se trata de una señal *Pulse Coded*:

- Bit '1' si: $MEDIDACERO > tolerancia+$ MEDIDACEROPEQUEÑA
- Bit '0' si: $MEDIDACERO \leq tolerancia+$ MEDIDACEROPEQUEÑA

Reconocimiento de Shift Coded

En el caso de tratarse de una señal que transporta la información en la fase es posible reconocerla porque presisamente no cumple los requisitos que si seguían las anteriores codificaciones. No tiene todos los '1' ni todos los '0' de valor parecido. Siempre hay ceros y unos de longitudes diferentes salvo en un caso. Este caso será aquel en el que toda la señal estuviese compuesta por ceros o bien por unos, situación en la que no podría direfenciarse de una señal de las anteriores. Este situación evidentemente no se va a dar nunca.

Para obtener la señal binaria se parte, como en las codificaciones anteriores, de los valores '1' y '0' mínimo que serán las referencias. El proceso de reconocimiento de los bits aumenta, en este caso, de forma considerable su complejidad. Inicialmente se tienen en cuenta el primer tiempo '0' y el primer tiempo '1'.

- Si el bit anterior fué '1' (El primer bit de la trama siempres es '1'):
    - Bit '1': Si después de restarle a la anterior medida '1' el '1' mínimo queda un tiempo restante menor que el propio '1' mínimo.
    - Bit '0': Si depués de restarle al anterior '1' el '1' mínimo queda un tiempo mayor o igual que el tiempo mínimo '1'.

- Si el bit anterior fué '0':
  - Bit '0': Si después de restarle a la anterior medida '0' el '0' mínimo queda un tiempo restante menor que el propio '0' mínimo.
  - Bit '1': Si depués de restarle al anterior '0' el '0' mínimo queda un tiempo mayor o igual que el tiempo mínimo '0'.

3.1.3. Software RS-232

Introducción

El desarrollo del software encargado de implementar las tareas de envío de señales deacuerdo con el protocolo RS-232 se ve enormemente reducido gracias a que el PIC16F628 posee un USART.

Programación del USART del PIC16F628

El display de TECDIS posee unos microinterruptores de configuración que están situados en una posición que habilita la comunicación a través de su puerto serie a una velocidad de 9600 Baudios. Por lo tanto para nuestras comunicaciones a través del puerto serie RS-232, el usart se configura a 9600 Baudios, sin paridad y con un bit de stop.

Como $BRGH = 1$ (Alta velocidad) la fórmula que se emplea para la obtención del valor de configuración de SPBRG es $Fosc/(16(X+1))$.

Los cálculos realizados para obtener el valor que es necesario cargar en el registro SPBRG se muestran seguidamente.

$$TasadeBaudiosdeseada = Fosc/(16(X+1)) \tag{3.1}$$

$$9600 = 4000000/(16(X+1)) \tag{3.2}$$

$$X = (4000000/(9600 \cdot 16)) - 1 = 25{,}041(0x19) \tag{3.3}$$

Por lo tanto en el registro SPBRG se carga el valor 0x19. De esta forma se configura el USART para que realice las tareas de comunicación a 9600bps.

El código que realiza esta tarea de configuración inicial del USART es el siguiente:

```
INI_RS232:
    bsf     STATUS,RP0          ; Banco 1.


; Establece la velocidad de comunicacion con el PC.
;-----------------------------------------------------------

;Baudios = 9600, Sin paridad, 1 Bit de Stop.

    movlw   0x19                ; 0x19=9600 bps.
    movwf   SPBRG
    movlw   b'00100100'         ; brgh = Alto (2).
    movwf   TXSTA               ; Habilita la transmision asincrona.
    bcf     STATUS,RP0          ; Banco 0.
    movlw   b'10010000'         ; Habilita la recepcion asincrona.
    movwf   RCSTA
```

Para el envío de datos a través del puerto serie RS-232 se siguen estos pasos:

1. Se carga el registro TXREG con el byte que se desea enviar.

2. Se comprueba el bit TRMT del registro TXSTA hasta que pasa a '1'. En el momento en el que este bit pasa a nivel alto ya está enviado el byte.

```
send:
   movwf   TXREG                ; Envia el dato de W.
   bsf     STATUS,RP0           ; Banco 1.
AQUII:
   btfss   TXSTA,TRMT           ; (1) La transmision se completa si
                                ; esta a '1'.
   goto    AQUII
   bcf     STATUS,RP0           ; Banco 0.
   return
```

3.1.4. Software I2C

Introducción

En lo que se refiere al puerto I2C, todo el protocolo al completo se ha tenido que implementar mediante código de programa, puesto que el PIC16F628 no dispone de ningún elemento preparado para realizar las tareas de comunicación correspondientes a esta interfaz.

Rutinas del Puerto I2C

La rutina de envío por el puerto I2C se llama *I2C_COMUNICACION* y comienza de la siguiente manera:

```
I2C_COMUNICACION:
    movlw b'00000000'
    movwf DIR_ESCLAVO
    bsf   I2C_PORT,I2C_SDA

    ; Configura el puerto I2C.
    bsf   STATUS,RP0                  ; Banco 1.
    bcf   TRISA,I2C_SCL               ; SCL y SDA son salidas.
    bcf   TRISA,I2C_SDA
    bcf   STATUS,RP0                  ; Banco 0.
    bsf   I2C_PORT,I2C_SDA


mas:
    call  I2C_INICIO                  ; Condicion de comienzo.
    movfw DIR_ESCLAVO                 ; Guarda la direccion del esclavo.
    call  I2C_ENVIA
    call  I2C_ACK_esclavo             ; Espera el noveno pulso: ACK.
    btfss STATUS,Z                    ; Comprueba Z y si es 1 hubo ACK.
    goto  N_ACK                       ; Esclavo no respondio.
    bsf   STATUS,RP1                  ; Banco 2.
    movfw byte1
    bcf   STATUS,RP1                  ; Banco 0.
    call  I2C_ENVIA                   ; Envia el byte por I2C.
    call  I2C_ACK_esclavo             ; Espera el ACK del esclavo.
    btfss STATUS,Z                    ; Comprueba Z y si es 1 hubo ACK.
    goto  N_ACK
```

Los pasos seguidos son:

1. La subrrutina *I2C_INICIO* produce la condición de inicio.
2. *I2C_ENVIA* envía el dato.
3. Se espera el ACK procedente del esclavo.
   - Si el esclavo responde se procede al envío de otro dato.
   - Si el esclavo no responde sale de la subrrutina de envío.

Después de enviar los 8 bytes de datos que representan a la señal de infrarrojos recibida, viene la condición de STOP y sale de la subrrutina:

```
N_ACK:
        call     I2C_PARADA    ; Condicion de STOP
        return
```

Vemos las rutinas que incluye la función *I2C_COMUNICACION*:

I2C_INICIO

Esta subrrutina se encarga de generar la condicón de inicio o de START.

La línea de datos (SDA) toma un estado bajo mientras que la línea de reloj (SCL) permanece alta.

```
I2C_INICIO:
    bsf    STATUS,RP0
    bcf    TRISA,I2C_SDA
    bcf    STATUS,RP0

    bsf    I2C_PORT,I2C_SDA
    bsf    I2C_PORT,I2C_SCL            ; INICIO I2C: Cambia SDA de
    call   I2C_Retardo                 ; 1 a 0 cuando SCL = 1.
    bcf    I2C_PORT,I2C_SDA
    call   I2C_Retardo
    bcf    I2C_PORT,I2C_SCL
    return
```

I2C_ENVIA

Cada bit enviado es válido cuando SCL está a '1' por lo que hay que colocar el dato en SDA antes de poner a '1' l línea SCL.

```
    movwf  I2C_TEMP                    ; Envia por I2C el byte de W.

    bsf    STATUS,RP0
    bcf    TRISA,I2C_SDA
    bcf    STATUS,RP0

    bcf    I2C_PORT,I2C_SDA
    btfsc  I2C_TEMP,7
    bsf    I2C_PORT,I2C_SDA
    bsf    I2C_PORT,I2C_SCL
    call   I2C_Retardo
    bcf    I2C_PORT,I2C_SCL
```

I2C_ACK_esclavo

Para recibir la respuesta del esclavo, SDA se pone como entrada en el noveno pulso de reloj. Existe ACK si el esclavo pone a '0' la línea SDA durante el noveno pulso mencionado.

```
I2C_ACK_esclavo:
    bsf    STATUS,RP0
    bsf    TRISA,I2C_SDA
    bcf    STATUS,RP0
```

```
    call   I2C_Retardo            ; ACK del esclavo I2C:
    bsf    I2C_PORT,I2C_SCL       ; SDA a 1 por el maestro.
    call   I2C_Retardo            ; SDA=1 si esclavo no responde.
    bsf    STATUS,Z               ; SDA=0 si esclavo responde.
    btfsc  I2C_PORT,I2C_SDA
    bcf    STATUS,Z               ; STATUS,Z=1 si respondio.
    bcf    I2C_PORT,I2C_SCL       ; STATUS,Z=0 si no respondio.
    return
```

I2C_PARADA

Finalmente el PIC envía la condición de STOP. Con SCL a '1' SDA pasa de nivel bajo a nivel alto.

```
I2C_PARADA:
    bsf    STATUS,RP0
    bcf    TRISA,I2C_SDA
    bcf    STATUS,RP0

    bcf    I2C_PORT,I2C_SDA
    bsf    I2C_PORT,I2C_SCL       ; PARADA I2C: Cambia SDA de
    call   I2C_Retardo            ; 0 a 1 cuando SCL = 1.
    bsf    I2C_PORT,I2C_SDA
    return
```

3.1.5. Software del puerto paralelo

Inicialmente las dos líneas de control, tanto la que controla el dispositivo Maestro (RA4) como la controlada por el Esclavo (RA3), está nivel alto. El envío de un dato comienza cuando nuestro sistema pone un '0' en la línea conectada a RA4.

```
    bcf      PORTA,4          ; Pone a cero RB3.

no_pasa:
    btfsc    INTCON,T0IF
    goto     Esclavo_No_Resp
    btfsc    PORTB,3
    goto     no_pasa
                              ; RA0-RA3 Y RB7-RB4 pasan a ser salidas.
    bsf      STATUS,RP0       ; Banco 1.

    movlw    b'00000000'      ; Todo salidas.
    movwf    TRISA            ; Configura PORTA como salidas.
    movlw    B'00001011'
    movwf    TRISB            ; Configura RB0 y RB1 como entradas.

    bcf      STATUS,RP0       ; Banco 0.

    ; Pone el dato en el bus.
    movfw    caracteraux      ; Contiene el caracter que se envia.
```

```
    call     ENVIA_DATOII     ; Envia el dato por el puerto paralelo.
    bsf      PORTA,4          ; Indica que el dato esta listo.

espera_espera:
    btfsc    INTCON,TOIF      ; Comprueba si acabo la cuenta del TMR0.
                              ; Si TMR0 termino de contar no se
                              ; obtuvo respuesta del esclavo.
    goto     Esclavo_No_Resp  ; El esclavo no respondio.
    btfss    PORTB,3          ; Espera que el esclavo lea el dato.
    goto     espera_espera

    bsf      APRENDIZAJE_REG,0; Indica que esclavo recibio el dato.
Esclavo_No_Resp:
    bsf      PORTA,4          ; Pone a uno RB3 para no producir
                              ; una nueva int. al esclavo.
```

En el envío de cada byte intervienen el puerto A y el puerto B del PIC. La parte alta del dato se envía por el puerto A (RA0-RA3) y los bits menos significativos por el puerto B (RB4-RB7):

```
ENVIA_DATOII:
    movwf    caracteraux          ; Byte que se envia por el puerto.
    movwf    cuenta               ; Se hace otra copia del byte.
    bcf      STATUS,C             ; Borra el carry.
    rrf      cuenta,f             ; Desplaza cuatro bits a la derecha para
    rrf      cuenta,f             ; colocar la parte alta del dato sobre
    rrf      cuenta,f             ; la parte baja de PORTA.
    rrf      cuenta,f
    btfss    PORTA,4              ; Mantiene el valor de la senal de
    bcf      cuenta,4             ; control del puerto paralelo.
    bsf      cuenta,4
    movfw    cuenta               ; Coloca la parte alta del dato en
    movwf    PORTA                ; el bus.
    movfw    caracteraux          ; Copia el byte que se envia.
    movwf    cuenta
    bcf      STATUS,C             ; Borra el carry.
    rlf      cuenta,f             ; Rota a la izquierda cuatro posiciones.
    rlf      cuenta,f
    rlf      cuenta,f
    rlf      cuenta,f
    movfw    cuenta               ; Coloca la parte baja del dato sobre
    movwf    PORTB                ; la parte alta de PORTB.

    return
```

La rutina de envío de bytes por puerto paralelo *ENVIA_PARALELO_BYTES* tiene una medida de seguridad que evita que el sistema se mantenga en situación de espera permanente de respuesta por parte del esclavo cuando por ejemplo este no está conectado. Se temporiza el tiempo que pasa desde que se solicita una respuesta por parte del Esclavo hasta que se tiene. Si esta respuesta no llega se sale de la subrrutina. Se configura el

TMR0 con una preescala de 1:32 para poder contar un total de 8192μs. De esta forma se puede emplear el puerto serie RS-232.

3.1.6. Software del LCD

En el LCD 16x2 sólo se muestra información en la primera de sus líneas. Basta sólo con una línea porque se muestran señales binarias de tamaño máximo igual a 8 bytes (64 bits).

Cada vez que un byte se manda al LCD éste se inicializa. Bastaría con inicializar una sola vez el display; de esta forma se conseguiría un ahorro de tiempo pero esto tiene un inconveniente: Si es LCD no está conectado en el momento inicial de puesta en marcha del sistema y se coloca con posterioridad en su posición no funcionará. Por esta razón se inicializa el LCD cada vez que se envía un nuevo dato. Para inicializarlo simplemente se sigue el protocolo tal y como indican las especificaciones. Esta subrrutina emplea una Macro que se llama EN_ESTROBE. Esta Macro habilita el LCD durante 1μs.

El bus de datos se configura para que sea de 4 bits.

La secuencia a seguir para pone el LCD en modo 4 bits es:

1. Enviar 0x03 ; esperar 2ms
2. Enviar 0x03 ; esperar 2ms
3. Enviar 0x03 ; esperar 2ms
4. Enviar 0x02 ; esperar 2ms

```
LcdInic:
    bcf     PORTB,LCD_E     ; Deshabilita el LCD
    bcf     PORTA,LCD_RS    ; Pone el LCD en modo comando
    movlw   10
    call    Retardoms
                            ; Envia al LCD la secuencia de reset
    bsf     PORTB,LCD_DB4   ; Envia 0x03
    bsf     PORTB,LCD_DB5
    bcf     PORTB,LCD_DB6
    bcf     PORTB,LCD_DB7
    EN_STROBE
    movlw   2
    call    Retardoms       ; Espera 2ms
    EN_STROBE
    movlw   2
    call    Retardoms
    EN_STROBE
    movlw   2
    call    Retardoms
```

```
    bcf     PORTB,LCD_DB4   ; Envia 0x02
    bsf     PORTB,LCD_DB5
    bcf     PORTB,LCD_DB6
    bcf     PORTB,LCD_DB7
    EN_STROBE
    movlw   2
    call    Retardoms
                            ; Bus de datos de 4 bits
    movlw   0x28
    call    LcdEnviaComando
                            ; Incremento, no desplazamiento
    movlw   0x06
    call    LcdEnviaComando
                            ; Display ON, Cursor OFF, Parpadeo OFF
    movlw   0x0C
    call    LcdEnviaComando
    movlw   0x01
    call    LcdEnviaComando
    return
```

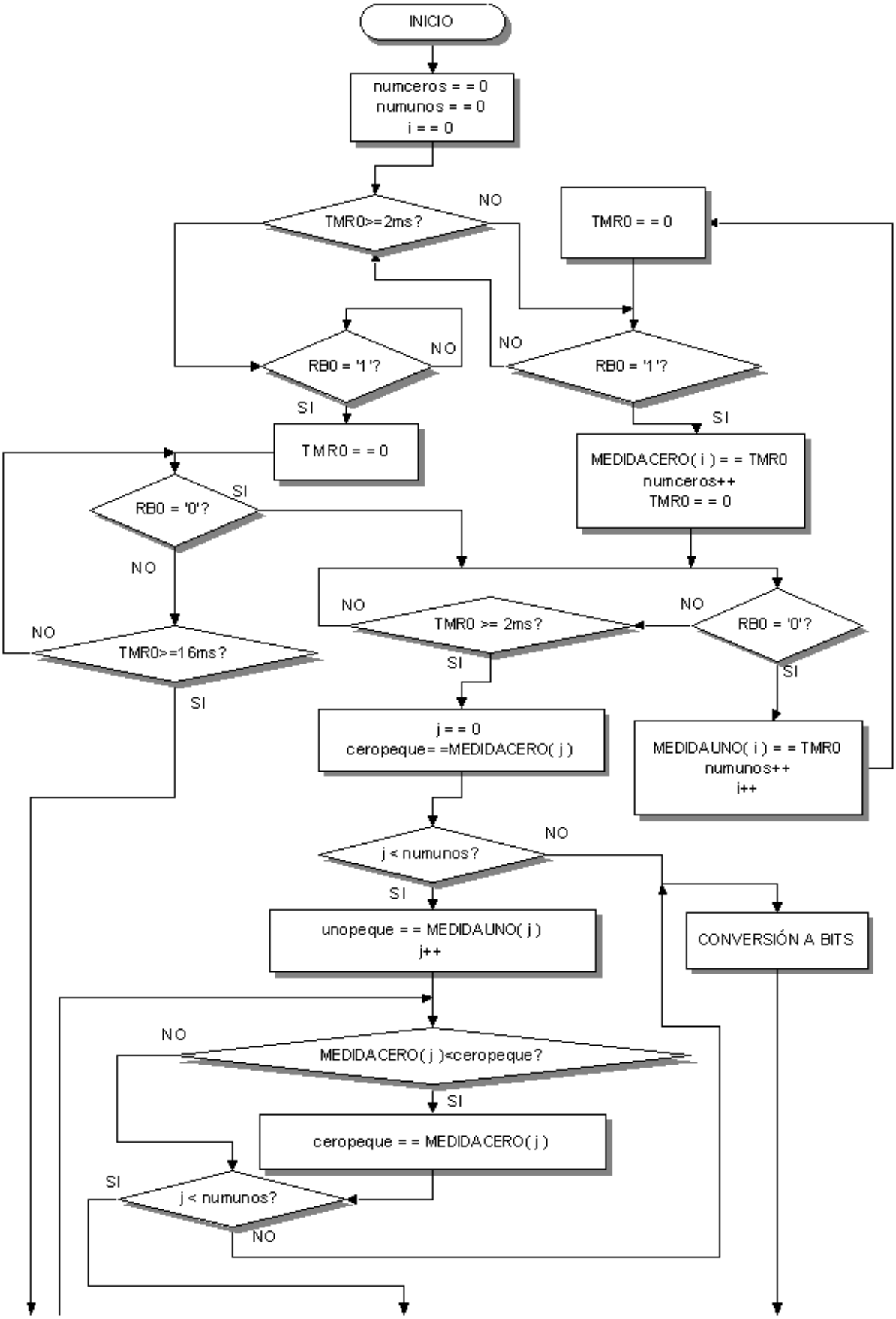
El envío de datos lo realiza la subrrutina LcdEnviaDato. Este segmento de código pone un '1' en RS indicando que se manda un dato y posteriormente la rutina LcdEnviaByte se encarga del resto del trabajo. Esta última rutina envía en primer lugar los 4 bits más significativos y después termina con el envío de los 4 bits de la parte baja del dato.

```
LcdEnviaDato:
    bsf     PORTA,LCD_RS
    call    LcdEnviaByte
    return
```

```
LcdEnviaByte:

    movwf   tmpLcdRegistro ; Salvaguarda el byte que se envia
                ; Envia los 4 bits mas significativos
    bcf     PORTB,LCD_DB4
    bcf     PORTB,LCD_DB5
    bcf     PORTB,LCD_DB6
    bcf     PORTB,LCD_DB7
    btfsc   tmpLcdRegistro,4
    bsf     PORTB,LCD_DB4
    btfsc   tmpLcdRegistro,5
    bsf     PORTB,LCD_DB5
    btfsc   tmpLcdRegistro,6
    bsf     PORTB,LCD_DB6
    btfsc   tmpLcdRegistro,7
    bsf     PORTB,LCD_DB7
    EN_STROBE
    movlw   2
    call    Retardoms
                ; Envia los 4 bits menos significativos
    bcf     PORTB,LCD_DB4
    bcf     PORTB,LCD_DB5
    bcf     PORTB,LCD_DB6
```

```
    bcf      PORTB,LCD_DB7
    btfsc    tmpLcdRegistro,0
    bsf      PORTB,LCD_DB4
    btfsc    tmpLcdRegistro,1
    bsf      PORTB,LCD_DB5
    btfsc    tmpLcdRegistro,2
    bsf      PORTB,LCD_DB6
    btfsc    tmpLcdRegistro,3
    bsf      PORTB,LCD_DB7
    EN_STROBE
    movlw    2
    call     Retardoms
    return
```


INICIO
numceros = = 0
numunos = = 0
i = = 0
TMR0>=2ms?
NO
TMR0 = = 0
RB0 = '1'?
NO
NO
RB0 = '1'?
SI
SI
TMR0 = = 0
MEDIDACERO(i) = = TMR0
numceros++
TMR0 = = 0
RB0 = '0'?
SI
NO
NO
TMR0 >= 2ms?
NO
RB0 = '0'?
NO
TMR0>=16ms?
SI
SI
SI
j = = 0
ceropeque= =MEDIDACERO(j)
MEDIDAUNO(i) = = TMR0
numunos++
i++
j < numunos?
NO
SI
unopeque = = MEDIDAUNO(j)
j++
CONVERSIÓN A BITS
NO
MEDIDACERO(j)<ceropeque?
SI
ceropeque = = MEDIDACERO(j)
SI
j < numunos?
NO

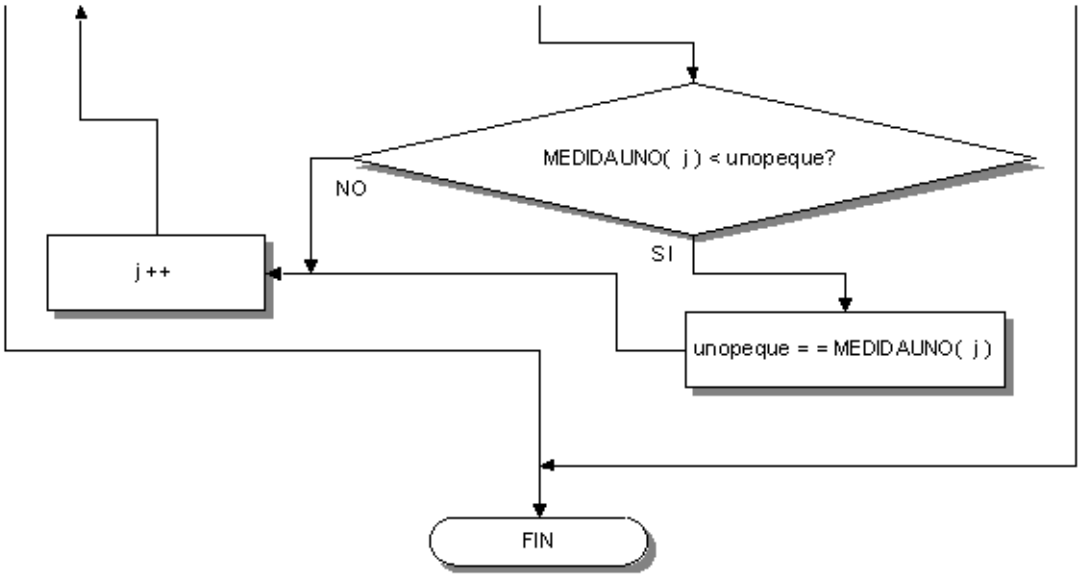


Figura 3.1: Diagrama de flujo de la subrrutina que toma los tiempos de la señal de entrada y decodifica los bits de ésta.

Capítulo 4

Aplicación práctica

4.1. Introducción

4.1.1. Finalidad

Este sistema es el encargado de la definición del contenido del LCD de TECDIS medidante la recepción de las señales de un mando cualquiera. Pero antes de estar preparado para el proceso de programación del LCD, es necesario que el módulo memorice las señales asociadas a las teclas que desee el usuario.

El circuito correspondiente a la aplicación práctica se monta sobre la placa TIPO1.

El diagrama de bloques de la figura 4.1 presenta de forma global los componentes de esta aplicación.

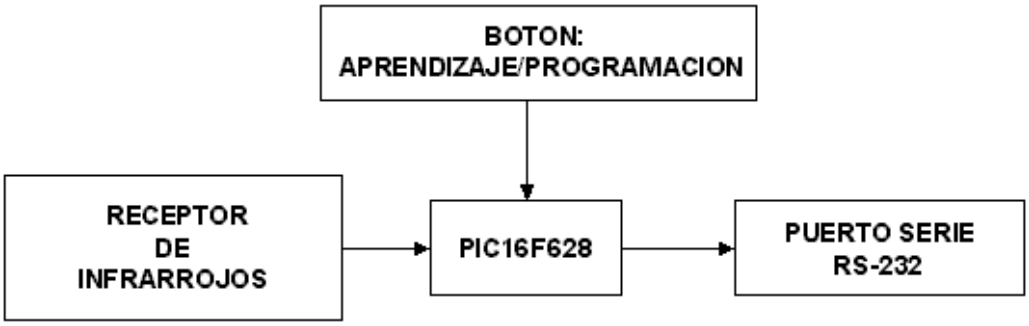


Figura 4.1: Diagrama de bloques de la aplicación práctica

4.1.2. Display LCD comecial de TECDIS

El LCD de TECDIS pertenece a una serie de displays que emplea el protocolo RS-232 para sus comunicaciones. Se trata de la versión 1 de esta serie. La velocidad a la que

circulan los datos por el puerto serie del LCD es configurable mediante unos microinterruptores, que están en una posición que establece una velocidad de 9600bps.

Vamos a ver el formato general que tienen los comandos que emplea este LCD para posteriormente ver en detalle los comandos de la versión 1.

Figura 4.2: Display LCD fabricado por TECDIS

FORMATO GENERAL

El formato general de un comando enviado al display es este:

- **STX-COMANDO-DATOS-ETX-CHK**

Para cada comando recibido el LCD tiene dos respuestas posibles:

- OK: **STX-COMANDO-ACK-DATOS-ETX-CHK**
- NO OK: **STX-COMANDO-NAK-ERR-ETX-CHK**

COMANDOS

1. **Petición de la versión del protocolo de comunicaciones.**

   Pide el número de la versión del protocolo de comunicaciones.

   - **COMANDO 0x20**: STX-0x20-ETX-CHK
   - **RESPUESTA**:
     - STX-0x20-ACK-NVCOM-ETX-CHK (NVCOM = Número de versión (en ASCII)('1'))

    - STX-0x20-NAK-ERR-ETX-CHK

2. **Petición del nombre y de la versión del programa del panel.**

   Pide el nombre y la versión del programa del panel.

   - **COMANDO 0x21**: STX-0x21-ETX-CHK
   - **RESPUESTA**:
     - STX-0x21-ACK-NOMVER-ETX-CHK (NOMVER = Nombre y versión del programa separado por una coma (','))
     - STX-0x21-NAK-ERR-ETX-CHK

3. **Cambiar número de caracteres por línea.**

   Indica al panel el número de caracteres por línea.

   - **COMANDO 0x22**: STX-0x22-0x81-NCARLIN-ETX-CHK (NCARLIN = Número de caracteres por línea (1 byte))
   - **RESPUESTA**:
     - STX-0x22-ACK-ETX-CHK
     - STX-0x22-NAK-ERR-ETX-CHK

4. **Cambiar la velocidad del scroll.**

   Cambia la velocidad del scroll.

   - **COMANDO 0x23**: STX-0x23-0x81-VELSCR-ETX-CHK (VELSCR = Velocidad del scroll 1-5 (1 byte). 1 más rápida, 5 más lenta.)
   - **RESPUESTA**:
     - STX-0x23-ACK-NVCOM-ETX-CHK
     - STX-0x23-NAK-ERR-ETX-CHK

5. **Cambiar retardo de modo alternativo.**

   Cambia el retardo del modo alternativo.

   - **COMANDO 0x24**: STX-0x24-0x81-VELALT-ETX-CHK (VELALT = Retardo modo alternativo 1-40 (1 byte). 1 más rápido, 40 más lento.)

- **RESPUESTA**:
  - STX-0x24-ACK-ETX-CHK
  - STX-0x24-NAK-ERR-ETX-CHK

6. **Leer número de líneas y de caracteres por línea**

   Lee del panel la configuración del número de líneas y el número de caracteres por línea.

   - **COMANDO 0x2A**: STX-0x2A-ETX-CHK
   - **RESPUESTA**:
     - STX-0x2A-ACK-NLIN-NCARLIN-ETX-CHK
       - NLIN = Número de líneas = 0x81 (Se pone a '1' el bit más significativo para indicar que los datos se almacenan en la memoria no volátil)
       - NCARLIN = Número de caracteres por línea (1 byte).
     - STX-0x2A-NAK-ERR-ETX-CHK

7. **Enviar mensaje.**

   Escribe una cadena de texto con las siguientes opciones de visualización:

   - **Normal**: Visualiza el mensaje de forma fija, truncándolo o rellenando con espacios si fuera necesario.
   - **Parapadeo**: Análogo a modo normal, pero con el mensaje parpadeando. Puede ser de toda la línea o del segundo grupo de caracteres. Estos se seleccionan por un puente en el hardware.
   - **Scroll**: El mensaje se visualiza de forma continuada apareciendo por la derecha, realizando un desplazamiento de carácter en carácter y desapareciendo por la izquierda, de forma continuada.
   - **Alternativo**: El mensaje se divide en trozos del ancho del panel, y se visualiza cada uno de forma alternativa.

   - **COMANDO 0x30**: STX-0x30-0x81-OPC-CADENA-ETX-CHK
     - OPC = Opción de visualización del mensaje (1 byte). Si tiene el MSB a 1, el mensaje se graba en la memoria no volátil (Es necesario que siempre esté a '1').

    - 0x80 - Normal
    - 0x81 - Parpadeo de toda la línea
    - 0x82 - Scroll de un carácter
    - 0x84 - Alternativo
    - 0x88 - Parpadeo del segundo grupo de caracteres
  - CADENA = Cadena a visualizar (máximo 120 bytes).

- **RESPUESTA**:
  - STX-0x30-ACK-ETX-CHK
  - STX-0x30-NAK-ERR-ETX-CHK

Códigos de error

- ERR = Código de error
  - 0x80 Comando desconocido
  - 0x81 Número de línea incorrecto
  - 0x82 Opción de visualización incorrecta
  - 0x83 No se ha recibido el ETX
  - 0x84 Checksum incorrecto
  - 0x85 Demasiados caracteres en la cadena de mensaje
  - 0x88 Número de líneas o número de de caracteres por línea incorrecto
  - 0x89 Error al escribir en la memoria no volátil
  - 0x8A Velocidad de scroll incorrecta
  - 0x8B No se ha detectado sensor de temperatura
  - 0x8C No se ha detectado reloj en tiempo real
  - 0x8D Retardo modo alternativo incorrecto
  - 0x8E Tipo de placa de LCDs incorrecta
  - 0x90 Se ha recibido un código de control en los caracteres de una cadena de texto

Códigos de control

- NUL = 0x00
- STX = 0x02
- ETX = 0x03
- ACK = 0x06
- NAK = 0x15

Observaciones

- El checksum (CHK) se calcula como la OR exclusiva de los bytes enviados entre el STX y el ETX, sin incluir éstos.
- Los guiones no están incluidos en el protocolo. Son sólo a efectos de separación de los campos.
- En ningún caso se enviará el byte de control NUL.
- En el comando *escribir texto*, los caracteres permitidos son los del juego de caracteres válido. Si se envía un carácter no válido, el panel lo sustituirá por un espacio en blanco.

  Los caracteres permitidos son: **A**, **B**, **C**, **D**, **E**, **F**, **G**, **H**, **I**, **J**, **K**, **L**, **M**, **N**, **Ñ**, **O**, **P**, **Q**, **R**, **S**, **T**, **U**, **V**, **W**, **X**, **Y**, **Z**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **0**, **”**, _, +,-, \, /,(,), <, >, **’**, **:**, **?**, **¿**, **!**, **¡**, **.**, y **,**.

4.1.3. Teclado alfanumérico

Como teclado para la programación del LCD se utiliza cualquier mando a distancia comercial que emita a una frecuencia de 38Khz. El teclado del mando a distancia se usa como teclado alfanumérico. Un teclado alfanumérico tiene las teclas del alfabeto, los diez dígitos decimales y los signos de puntuación y de acentuación. En nuestro caso son 10 teclas las que engloban las letras del alfabeto (sólo mayúsculas), los dígitos decimales y los signos de acentuación y puntuación de la misma forma que lo hace el teclado de un teléfono móvil.

En cada tecla se incluyen diferentes caracteres distribuidos de la siguiente manera:

- Tecla 0: **Espacio en blanco**, +, **-**, /, \, (,), **¿**,**¡**

- Tecla 1: **1**, **”**, **_**, **’**, **:**, **?**, **¿**, **!**, **¡**, **.**
- Tecla 2: **A**, **B**, **C**, **2**
- Tecla 3: **D**, **E**, **F**, **3**
- Tecla 4: **G**, **H**, **I**, **4**
- Tecla 5: **J**, **K**, **L**, **5**
- Tecla 6: **M**, **N**, **Ñ**, **O**, **6**
- Tecla 7: **P**, **Q**, **R**, **S**, **7**
- Tecla 8: **T**, **U**, **V**, **8**
- Tecla 9: **W**, **X**, **Y**, **Z**, **9**

Además de las teclas 0-9 dispone de otras dos más: ENTER y ESCAPE. Con la primera se aceptan las opciones solicitadas y con la segunda se deshace la última operación.

Las teclas 2 y 8 permiten navegar a través de los menús; la 2 para subir menú y la 8 para bajarlo.

4.2. Hardware

El circuito de la aplicación práctica es como el visto en la sección 2.2 con la salvedad de que en este caso se prescinde de algunos componentes que no son necesarios. Desde el punto de vista hardware se trata de un diseño sencillo ya que únicamente tiene un puerto de comunicaciones, el puerto serie RS-232 que se conecta con el LCD comercial.

La imagen 4.3 muestra la fotografía de la placa TIPO1 con el montaje más sencillo, el que permite la programación del LCD de TECDIS. Este circuito consta de los siguientes elementos:

Puerto receptor de infrarrojos

El receptor empleado es el mismo que en los sistemas de las secciones 2.2 y 2.3, el IS1U60 de Sharp. La patilla Vo del receptor está unida al pin RB0 del PIC. La alimentación dispone también de un filtro (ver apartado 2.2.2).

Puerto serie RS-232

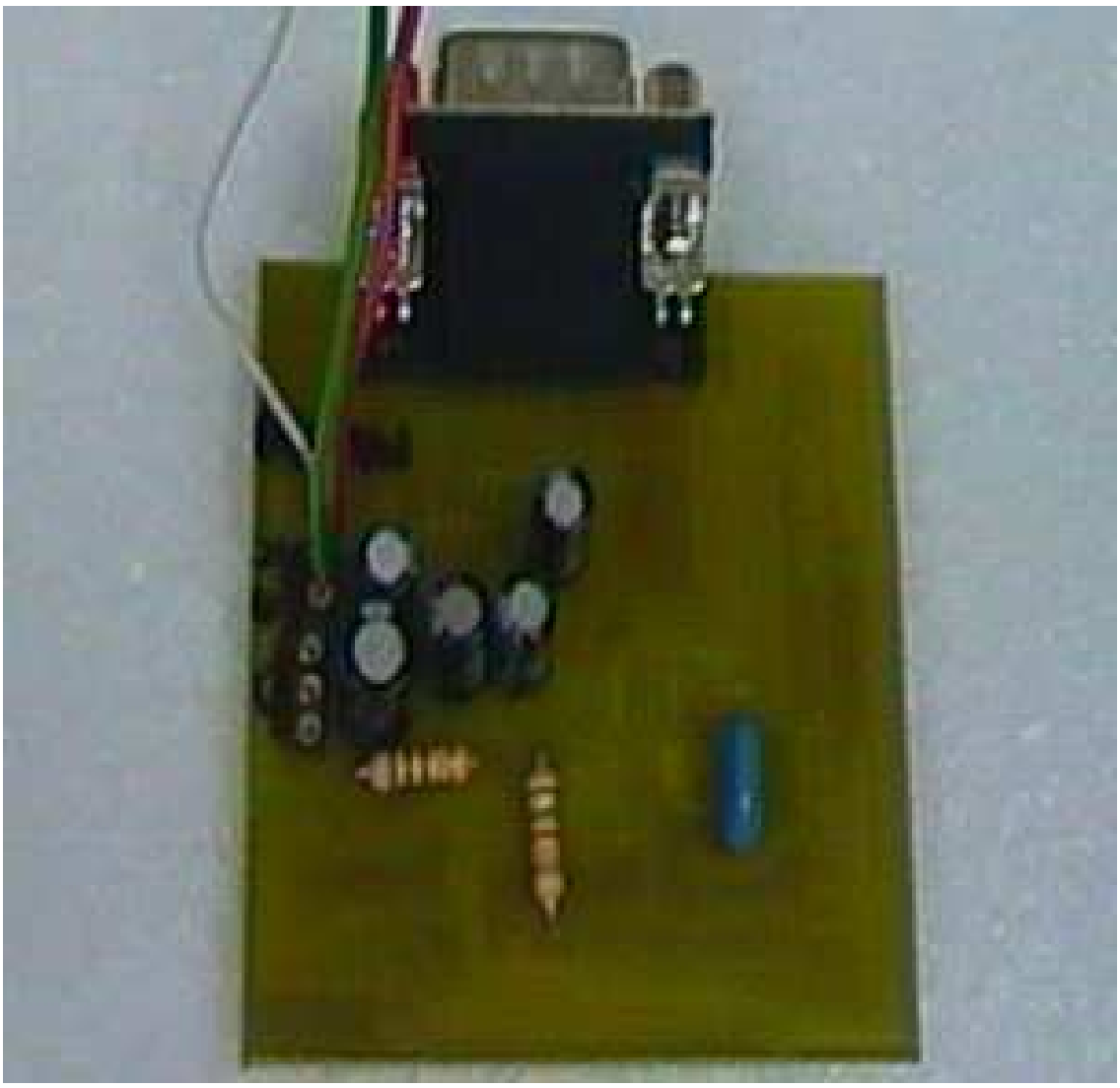


Figura 4.3: Fotografía de la placa TIPO1 con el montaje necesario para programar el LCD

En este circuito, este puerto, permite la comunicación con el LCD de TECDIS. Le envía a éste los comandos necesarios para su programación. El montaje es el mismo que el visto en las secciones 2.2 y 2.3.

Botón

El botón de selección de modo aprendizaje/modo programación permite indicarle al PIC16F628 cuando se desea memorizar señales procedentes de un mando a distancia y cuando la tarea solicitada es la de programación del LCD. El botón va conectado a la patilla del puerto A RA0. Este botón se conectará a Vcc si lo que se desea el programar

el LCD y a Vss si la operación solicitada es la de memorización de las señales del mando a distancia.

PIC16F628

En este caso el funcionamiento del PIC16F628 está regido por los valores de dos entradas independientes. El PIC interpreta las señales recibidas a través del puerto de infrarrojos, y dependiendo de como sea esta señal y del estado del botón de selección del modo de funcionamiento envía las señales correspondientes por el puerto serie RS-232.

Figura 4.4: PIC16F628 SMD

4.3. Software

4.3.1. Selección del modo de funcionamiento

Para seleccionar el modo de funcionamiento Aprendizaje/Programación según el estado del botón en situación de arranque inicial o en situación de espera se emplea el siguiente segmento de código:

```
      btfss    PORTA,0            ; Comprueba el modo de funcionamiento.
      goto     PREBUCLE2          ; Modo aprendizaje.

PREBUCLE1:                        ; Modo programacion.
      bcf      INTCON,GIE         ; Deshabilita interrupciones.
      movlw    '0'
      movwf    auxiliar
      call     ENVIA_MSJ_TECLA_2; Pide la tecla numero 0
                                  ; desde el LCD.

bucle1:
      bsf      INTCON,GIE         ; Habilita las interrupciones.
                                  ; Despues de la inicializacion del
                                  ; sistema espera interrupcion por rb0.
      btfss    PORTA,0
      goto     PREBUCLE2          ; Modo programacion.
      goto     bucle1             ; Modo Aprendizaje.
```

```
PREBUCLE2:
      bcf      INTCON,GIE        ; Deshabilita las interrupciones.
      call     ELIGE_MENS        ; Muestra mensaje definido para el LCD
                                 ; si lo hay, y si no existe solicita que
                                 ; se introduzca con el comando escribir.
bucle2:
      bsf      INTCON,GIE        ; Habilita las interrupciones.
      btfsc    PORTA,0           ; Comprueba modo de funcionamiento.
      goto     PREBUCLE1         ; Modo aprendizaje.
      btfss    PROGRAMACION_REG,3 ; Comprueba si se eligio una opcion
                                 ; para el display.
      goto     bucle2            ; Modo programacion.
```

Si la entrada RA0 está a '1' manda un mensaje solicitando que la tecla 0 sea pulsada (*Tecla 0?*). A continuación espera la llegada de la señal procedente del mando a distancia para que produzca una interrupción y así comenzar después a tomar las medidas de dicha señal.

Si por el contrario RA0 es '0' manda el mensaje guardado en la memoria EEPROM si es que lo hay. Si es la primera vez que se utiliza el sistema después de su construcción manda un mensaje al LCD indicando que es necesario programar (*SIN PROGRAMAR*) el LCD para que éste muestre un texto. Pero antes de poder realizar la operación de escritura es necesario que el módulo memorice las teclas de un mando a distancia.

Si ya hay almacenado un mensaje en la memoria EEPROM, este estará acompañado en esta memoria no volátil por el valor de las variables de configuración de la apariencia del mensaje en el LCD: Visualización y número de caracteres por línea. Para comprobar si existe mensaje se examina el contenido de la posición 1 de la EEPROM:

```
      movlw    1
      call     LEE_EEPROM        ; Comprueba si hay texto en la EEPROM.
      sublw    0xFF
      btfss    STATUS,Z
      goto     SI_TEXTO          ; Si hay texto.
```

Si hay texto se leen el tipo de visualización y el número de caracteres por línea además del número de caracteres almacenados:

```
SI_TEXTO:
      bsf      APRENDIZAJE_REG,7; Indica que hay texto en la EEPROM.
      movlw    124
      call     LEE_EEPROM        ; Lee el numero de caracteres del texto
      movwf    DIRECCION_EEPROM
      movlw    125
      call     LEE_EEPROM        ; Lee el modo de visualizacion.
      movwf    VISUALIZACION
      movlw    126
      call     LEE_EEPROM        ; Lee el numero de caracteres por linea.
      movwf    CAR_LIN
```

4.3.2. Aprendizaje de las señales de las teclas

Si el modo de funcionamiento elegido es el de memorización se solicita, según va siendo necesario, que el usuario pulse la tecla correspondiente. El modo aprendizaje comienza enviando al LCD un mensaje que dice *Tecla 0?*, continua con otro que dice *Tecla 1?* y así sucesivamente. El modo aprendizaje finaliza cuando memorizadas todas las teclas se muestra un mensaje que dice *FIN!*. Hay que dar respuesta a cada mensaje pulsando la tecla del mando a distancia que queremos que tenga la función o número que se indica. De esta forma se irá avanzando en el proceso.

Durante el proceso de memorización se almacenan las señales binarias correspondientes a cada tecla del mando a distancia en la memoria RAM de datos. Esto hace que en el momento en el que se le quite la alimentación al módulo se pierda la información almacenada.

```
T_1:

tecla2_senal:
    btfss   PORTA,0
    retlw   250                         ; Sale si ya no esta en modo
                                        ; aprendizaje.
    btfsc   PORTB,0                     ; Espera que llegue la senal.
    goto    tecla2_senal

    call    Rec_RC5_RE80                ; Recibe datos de tecla "1".

    ; Almacena el valor de la se al recibida

    call    DETECTA_INTERFERENCIAS      ; Senal de ceros?
    bcf     STATUS,RP1                  ; Banco 0.
    iorlw   0
    btfsc   STATUS,Z
    goto    T_1                         ; Hay senal de interferencia.
    call    DETECTA_TECLA               ; Detecta si es una tecla
    subwf   auxiliar_9,0                ; memorizada.
    btfsc   STATUS,C
    goto    T_1
    call    GRABA_BYTE                  ; Guarda la senal de la tecla.
```

La subrrutina *DETECTA_TECLA* ayuda a saber si la señal recibida es correcta o por el contrario es una señal consecuencia de una fuente interferente como por ejemplo una bombilla o la propia luz proveniente del sol. Para asegurarnos de que la señal recibida es correcta nos basamos en la aletoriedad de las interferencias. Teniendo en cuenta esta idea, para que una señal recibida sea válida debe recibirse 2 veces seguidas. Gracias a *DETECTA_TECLA* se puede averiguar si una señal se aprendió en el proceso de memorización y en que posición se memorizó. Esta subrrutina es de doble utilidad porque se

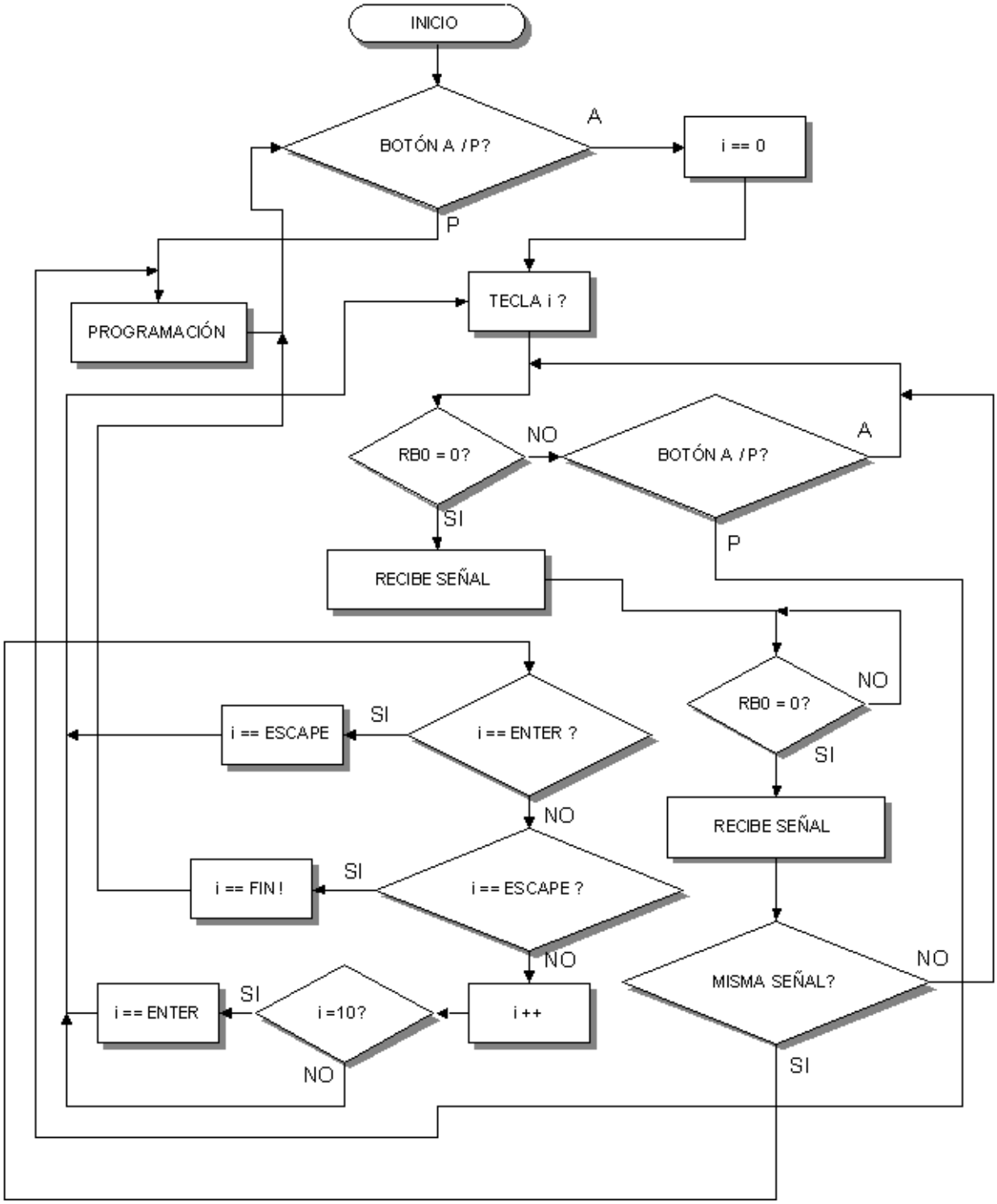


Figura 4.5: Proceso de aprendizaje de las teclas del mando a distancia

encarga de encontrar el número de tecla pulsada en el proceso de programación y en la tarea de aprendizaje permite saber si se recibe dos veces seguidas la misma señal.

```
DETECTA_TECLA:

    clrf    auxiliar_7          ; Guarda el numero de tecla que
                                ; se esta examinando.
NO_ES:
    movlw   12                  ; Numero de teclas total.
    subwf   auxiliar_7,0        ; Comprueba si recorrio todas las
    btfsc   STATUS,Z            ; teclas.
    goto    sale_YA             ; Termino.

    movlw   DIR_TECLA1
    movwf   FSR_TEMP1
    movfw   auxiliar_7
    movwf   auxiliar_3
    incf    auxiliar_7          ; Siguiente tecla.


             ; Solo se comparan los 4 bytes de menor peso.
    movlw   3
sumando:
    addwf   FSR_TEMP1,1         ; Accede al byte6 de la tecla
    decfsz  auxiliar_3          ; Guarda la direccion del byte6.
    goto    sumando
    bsf     STATUS,RP1          ; Banco 2.
    movfw   byte5
    bcf     STATUS,RP1          ; Banco 0.
    movwf   cuenta              ; Copia el byte5.
    movlw   DIR_TECLA1_0
    movwf   FSR
    movfw   auxiliar_7
    addwf   FSR,1
    decf    FSR
    movfw   INDF
    subwf   cuenta,0            ; Compara los byte5 de la senal tomada
    btfss   STATUS,Z            ; y la almacenada.
    goto    NO_ES               ; No coinciden.

    call    COMPARA_BYTESS      ; Compara los bytes 6, 7 y 8.

    movwf   auxiliar_3          ; Comprueba si coincidieron los 3 bytes.
    btfss   auxiliar_3,0
    goto    NO_ES               ; No coincidieron.
    decf    auxiliar_7
sale_YA:
    movfw   auxiliar_7          ; Numero de tecla detectada si la hubo.
                                ; Si no detecto, auxiliar_7 = 12.
    return
```

La subrrutina de recepción de la señal, la que obtiene la señal binaria, es la misma que empleaban tanto el programa con puerto I2C, como el programa con puerto paralelo.

En el proceso de memorización, cuando se ha comprobado que la señal el válida, ésta se almacena. Se guardan los 4 bytes de menor peso; no se almacena el resto de bytes,

si es que hubiese bytes de señal más allá de estos cuatro, por cuestiones de espacio. No hay memoria suficiente. Para el almacenamiento en la memoria RAM de estos bytes se utiliza el direccionamiento indirecto tal y como se muestra en este fragmento de código perteneciente a *GRABA_BYTE*.

```
GRABA_BYTE:
    movfw   auxiliar_2
    movwf   FSR
    bsf     STATUS,RP1          ; Banco 2.
    movfw   byte5               ; Guarda byte5.
    movwf   INDF                ; TECLAx_0.
    bcf     STATUS,RP1          ; Banco 0.
```

4.3.3. Proceso de programación del LCD

Finalizado el aprendizaje, en el momento en el que aparece el mensaje *FIN!* en la pantalla del display, es necesario pulsar el botón para cambiar al modo programación. Este modo es el que nos permite definir el contenido del LCD mediante el uso de un menú mostrado sobre el mismo display.

En el instante en el que llega una señal al receptor IS1U60 y éste envía la señal eléctrica a la patilla RB0 se produce una interrupción y comienza el proceso de obtención de la señal binaria. Es necesario un proceso de detección para saber si la señal recibida coincide con alguna de las almacenadas en la RAM de datos. El proceso de detección, como ya sabemos, lo lleva a cabo *DETECTA_TECLA*. Si los 4 bytes menos significativos de la señal recibida coinciden con los 4 de menor peso de alguna de las señales recibidas entonces se obtiene un número de tecla; se ha pulsado una de las teclas memorizadas. El número de tecla es almacenado en el registro de la RAM de nombre auxiliar.

Una vez se tiene el número de tecla pulsado, si es que se pulso una tecla memorizada, se realiza la operación asociada a esa tecla; la operación inicial que se puede realizar nada más entrar en modo programación es la de navegar por el menú. Se puede subir en el menú o bajar según se desee. Si se pulso la tecla **2** se asciende en el menú, mientras que si se pulsó la tecla **8** se baja dentro de éste. Se comprueba si la tecla pulsada es la **2** o bien la **8** en el siguiente segmento de código:

```
    movlw   2                   ; Busca opcion.
    subwf   auxiliar,0          ; Arriba?
    btfsc   STATUS,Z
    goto    MUESTRA_COMANDO_UP

    movlw   8                   ; Busca opcion.
    subwf   auxiliar,0          ; Abajo?
```

```
    btfsc   STATUS,Z
    goto    MUESTRA_COMANDO_DOWN
```

La opción de comando seleccionada se guarda en el registro comando:

```
MUESTRA_COMANDO_UP:             ; Muestra comando siguiente.
    movlw   4
    subwf   COMANDO,0           ; El comando mas alto tiene el
    btfss   STATUS,Z            ; numero 4.
    goto    INCREMENT           ; Sube comando.
    movlw   1
    movwf   COMANDO             ; El comando mas bajo tiene el
    goto    SALTA_INC           ; numero 1.

INCREMENT:

    INCF    COMANDO             ; Sube el numero de comando
                                ; actual.
SALTA_INC:

    call    ESCRIBE_COMANDO     ; Muestra el comando en el LCD.
    goto    SALIR_YA            ; Sale de la subrrutina.
```

Los mensajes del menú que se envían al LCD se almacenan en una tabla:

```
MENSAJE_ALTERNATIVO:

    addwf PCL
    dt "ALTERNATIVO",0
    dt "CARACT./L",0
    dt "SCROLL",0
    dt "MITAD",0
    dt "ESCRIBIR",0
    dt "VELOCIDAD",0
    dt "RETARDO",0
    dt ">NORMAL",0
    dt "PARPADEO␣",0
    dt "FIN!",0
    dt "TECLA␣",0
    dt "ENTER?",0
    dt "ESCAPE?",0
    dt "VISUALIZACION",0
    dt "SIN␣PROGRAMAR",0
```

Para acceder a cada mensaje se le suma a la parte baja del contador de programa, PCL, el valor correspondiente.

```
TOMA_MENSAJES:
    movwf   cuenta              ; Carga el valor que se anade al PCL para
                                ; acceder al mensaje dentro de la tabla.
_Toma_Caracter:
```

```
    movfw   cuenta
    call    MENSAJE_ALTERNATIVO ; Toma el mensaje de la tabla.
    iorlw   0                   ; Comprueba si termina el mensaje.
    btfsc   STATUS,Z
    return
    call    XOR_ENVIA           ; Envia el caracter al LCD.
    incf    cuenta,F
    goto    _Toma_Caracter
```

La subrrutina que envía cada carácter por el puerto serie RS-232 es la misma que se vió en el apartado 3.1.3.

El envío de un mensaje comienza con la cabecera, el comando y en el ejemplo que se muestra a continuación, que pertenece al comando de envío de mensaje, sigue con el número de línea:

```
CABECERA_TXT:

    call    CARACT_LINEA_16; Configura el LCD a 16 caracteres por
    clrf    auxiliar_7     ; linea.
    movlw   0x02           ; STX
    call    send
    movlw   0x30           ; Comando de envio de mensajes.
    call    XOR_ENVIA
    movlw   0x81           ; Parpadeo.
    call    XOR_ENVIA
    return
```

Como vemos en el ejemplo anterior es necesario asegurarse de que el LCD está configurado con un valor de 16 caracteres por línea, porque si no fuese así no aparecerían correctamente los mensajes en la pantalla. Para establecer 16 caracteres por línea se manda el comando **'cambiar caracteres por línea'** con un valor 16 de su variable. Si el LCD estaba configurado con otro valor diferente de esta funcionalidad, no supone ningún inconveniente realizar esta operación; cada vez que se envía al LCD el mensaje que se desea mantener en éste, se configura antes el display para que tenga los caracteres por línea que tenía establecidos.

Depués del inicio de trama que hemos visto en el ejemplo, el de envío de mensajes, vienen la opción de visualización, la cadena de texto, y en último lugar la finalización de la trama:

```
MSJ_PIDE_MENSAJE_2:
    call CABECERA_TXT           ; Envia la cabecera
    call MODO_NORMAL_COMANDO    ; Envia comando de visualizacion
    call MSJTXT                 ; Envia cadena de texto
    call FINAL_TRAMA            ; Envia el final de trama
    return
```

La función *XOR_ENVIA* manda cada carácter de los que interviene en el checksum final de la trama y además realiza el XOR del carácter enviado con los anteriores caracteres que intervienen en la suma de comprobación:

```
XOR_ENVIA:
    xorwf   auxiliar_7,1
    call    send
    return
```

Dentro del menú de escritura se solicita la introducción de la cadena de texto que se desea mostrar. Los caracteres de esta cadena de texto se almacenan en la memoria EEPROM ocupando un total de 120 posiciones ya que el tamaño máximo de esta cadena es de 120 caracteres. Así se comprueba que no se sobrepasa el número máximo de caracteres:

```
    movlw   120                     ; Numero maximo de caracteres
    subwf   DIRECCION_EEPROM,0      ; Detecta si ya hay 120 caracteres
    btfsc   STATUS,Z
    goto    waiting_TEXTO
```

Como ya sabemos (ver sección 4.1.3) cada tecla tiene asociada una serie de caracteres. El número de tecla queda almacenado en la variable *auxiliar* justo después del proceso de detección de ésta. Para saber el carácter que actual que le corresponde a la tecla durante el proceso de escritura basta con el número de tecla y con el número de veces que se ha pulsado la misma tecla de forma seguida. Este último valor lo almacena la variable *auxiliar_5*. Cuando se recorren todos los caracteres de una tecla esta variable se pone a cero de nuevo.

Durante el proceso de definición del mensaje contenido en el LCD sólo se pueden mostrar en total en la pantalla 16 caracteres simultáneamente:

```
    movlw   0
    subwf   DIRECCION_EEPROM,0      ; Comprueba si la EEPROM esta vacia
    btfsc   STATUS,Z
    call    MSJ_PIDE_MENSAJE_2      ; Pide cadena de texto por pantalla
    btfsc   PROGRAMACION_REG,7      ; Comprueba si va al paso anterior
    goto    SALE_DE_PROGRAMACION_ESCRIBIR
    movlw   0
    subwf   DIRECCION_EEPROM,0
    btfsc   STATUS,Z
    goto    waiting_TEXTO           ; Espera la llegada de un caracter
    movlw   13
    subwf   DIRECCION_EEPROM,0      ; Comprueba si se llena la pantalla
                                    ; de caracteres
    btfsc   STATUS,C
    goto    SOLO_16_CHAR            ; Hay menos de 13 caracteres a mostrar
```

```
    movfw   DIRECCION_EEPROM      ; Hay por lo menos 13 caracteres
    movwf   cuenta                ; Muestra solo los ultimos 12 caracteres
    clrf    auxiliar_2
    goto    MUESTRA_CHAR
SOLO_16_CHAR:                     ; No se llena una pantalla
    movlw   12
    movwf   cuenta                ; cuenta=12(12+4(TXT:)=16(Una pantalla))
    subwf   DIRECCION_EEPROM,0
    movwf   auxiliar_2            ; DIRECCION_EEPROM - 12 = auxiliar_2
MUESTRA_CHAR:
    call    MSJ_PIDE_TXT_2
    movfw   auxiliar_2
CHAR_CHAR:
    call    LEE_EEPROM            ; Lee un caracter de la EEPROM
                                  ; W = dato
    call    XOR_ENVIA
    incf    auxiliar_2            ; Apunta a la siguiente direccion
    movfw   auxiliar_2            ; de la EEPROM
    decfsz  cuenta
    goto    CHAR_CHAR
    movlw   0x03                  ; Envia ETX y CHK
    call    send
    movfw   auxiliar_7
    call    send
```

Si en la EEPROM hay más de 12 caracteres almacenados, sólo se muestran en la pantalla del LCD los últimos 12 porque la pantalla sólo dispone de 16 ventanas y cuatro de ellas son las destinadas a mostrar la cadena **TXT:** que informa al usuario de que se está en proceso de introducción de la cadena de texto. Para esto se comprueba la variable *DIRECCION_EEPROM* que contiene el número de caracteres almacenados. La operación de comprobación aparece en el ejemplo anterior.

Para controlar que carácter debe aparecer en la posicón actual de escritura en la pantalla del LCD y cuando se debe avanzar una posición se utiliza una rutina de control temporal que introduce un retardo de duración constante:

```
RETARDO_10ms:
    movlw   10
    call    Retardo_ms    ; 10 ms de retardo
    return
```

Se avanza una posición de forma automática en el proceso de escritura si el tiempo que pasa desde la llegada de la última señal perteneciente a una tecla cuyo número esté entre 0 y 9 es mayor que 1,2 segundos. Otra condición que produce el avance de una posición en la pantalla del LCD es que la siguiente tecla pulsada del mando sea diferente a la anterior, siempre y cuando la tecla esté entre la número 0 y la número 9. Si la tecla pulsada es ESCAPE se borra un carácter; en el caso de que la tecla presionada sea ENTER se envía

el comando de envío de mensajes; ahora el display ya tiene definido el contenido deseado y está funcionando en modo normal.

Para borrar un carácter no hay que hacer más que disminuir en uno el contador de caracteres *DIRECCION_EEPROM*.

Si la tecla es ENTER, tal y como se apuntó antes, se deja definido el contenido del LCD de TECDIS:

```
    movlw   10                  ; Numero de la tecla ENTER
    subwf   auxiliar,0
    btfsc   STATUS,Z
    goto    MANDA_TEXTO_RS232   ; Manda el texto al LCD
```

Capítulo 5

Presupuesto

5.1. Coste de diseño

5.1.1. Material empleado en los tres circuitos correspondientes a la placa TIPO1 y TIPO2

La lista del material empleado y sus precios correspondientes se presenta en la tabla 5.1.

Componente o material	**Precio unidad (Euros)**
Microchip PIC 16F628-20/P (2k FLASH, 20 MHz, SMD)	4.40
LCD 16x2	7.88
Conector DB9 macho	0.27
Resonador 4Mhz con condensadores integrados	0.38
Barra de pines hembra torneados paso 2,54 mm	1.67
Conector hembra para cable plano 16 hilos	0.42
MAX232 SMD	1.06
Receptor IR IS1U60	3.97
Condensador electrolítico 2,2μF. 25V	0.24
Condensador electrolítico 47μF. 25V	0.15
Resistencia 47Ω 1/2 watios	0.26
Resistencia 10$K\Omega$ 1/2 watios	0.26
Resistencia 5,6$K\Omega$ 1/2 watios	0.26
Placa fibra de vidrio 1 cara virgen 80 x 120 mm	1.56

Cuadro 5.1: Lista del material empleado y su precio

5.1.2. Coste del trabajo de Ingeniero

Ver tabla 5.2. Consideramos que el coste del trabajo del ingeniero para cada uno de los circuitos es un tercio del coste total de ingeniero. Por lo tanto, como el coste del trabajo del ingeniero es de 8400 euros, el coste por circuito es de **2800 euros**.

Meses de trabajo de ingeniero	Coste por mes de ingeniero	Coste total
7	1200 euros	8400euros

Cuadro 5.2: Coste de ingeniero

5.1.3. Coste total de diseño

El coste total de diseño es el resultado de la suma del coste de material y del coste de ingeniero.

Coste total de diseño del circuito de la aplicación práctica

El material empleado por este circuito es el de la tabla 5.3.

Componente o material	Precio unidad (euros)	Unidades
Microchip PIC 16F628-20/P (2k FLASH, 20 MHz, SMD)	4.40	1
MAX232 SMD	1.06	1
Conector DB9 macho	0.27	1
Resonador 4Mhz con condensadores integrados	0.38	1
Barra de pines hembra torneados paso 2,54 mm	1.67	1
Condensador electrolítico 2,2μF. 25V	0.24	4
Condensador electrolítico 47μF. 25V	0.15	1
Resistencia 47Ω 1/2 watios	0.26	1
Resistencia 10$K\Omega$ 1/2 watios	0.26	1
Placa fibra de vidrio 1 cara virgen 80 x 120 mm	1.56	1/6

Cuadro 5.3: Coste de material del circuito de la aplicación práctica

El coste total del material empleado en el circuito de la aplicación práctica es **9.67 euros** y el coste de diseño por lo tanto es el siguiente:

9.67 euros+2800 euros=2809.67 euros.

Coste total de diseño del circuito que contiene el puerto I2C

EL material empleado por este circuito es el de la tabla 5.4.

Componente o material	**Precio unidad (euros)**	**Unidades**
Microchip PIC 16F628-20/P (2k FLASH, 20 MHz, SMD)	4.40	1
MAX232 SMD	1.06	1
Conector DB9 macho	0.27	1
Resonador 4Mhz con condensadores integrados	0.38	1
Barra de pines hembra torneados paso 2,54 mm	1.67	1
Conector hembra para cable plano 16 hilos	0.42	1
Condensador electrolítico 2,2μF. 25V	0.24	4
Condensador electrolítico 47μF. 25V	0.15	1
Resistencia 47Ω 1/2 watios	0.26	1
Resistencia 10$K\Omega$ 1/2 watios	0.26	1
Resistencia 5,6$K\Omega$ 1/2 watios	0.26	1
Placa fibra de vidrio 1 cara virgen 80 x 120 mm	1.56	1/6

Cuadro 5.4: Coste de material del circuito con puerto I2C

El coste total del material del circuito que contiene el puerto I2C es **10.35 euros**. El coste de diseño es el que se muestra a continuación:

10.35 euros+2800 euros=2810.35 euros.

Coste total de diseño del circuito que contiene el puerto Paralelo

EL material empleado por este circuito es el de la tabla 5.5.

El coste total del material del circuito con puerto paralelo es **9.93 euros**. El coste de diseño es este:

9.93 euros+2800 euros=2809.93 euros.

5.2. Coste de producción

El coste de producción se calcula como el material multiplicado por el número de unidades, más el coste de diseño (que es fijo) y más el coste de fabricación multiplicado

Componente o material	**Precio unidad (euros)**	**Unidades**
Microchip PIC 16F628-20/P (2k FLASH, 20 MHz, SMD)	4.40	1
MAX232 SMD	1.06	1
Conector DB9 macho	0.27	1
Resonador 4Mhz con condensadores integrados	0.38	1
Barra de pines hembra torneados paso 2,54 mm	1.67	1
Condensador electrolítico 2,2μF. 25V	0.24	4
Condensador electrolítico 47μF. 25V	0.15	1
Resistencia 47Ω 1/2 watios	0.26	1
Resistencia 10$K\Omega$ 1/2 watios	0.26	1
Resistencia 5,6$K\Omega$ 1/2 watios	0.26	1
Placa fibra de vidrio 1 cara virgen 80 x 120 mm	1.56	1/6

Cuadro 5.5: Coste de material del circuito con puerto Paralelo

también por el número de unidades. El coste de fabricación se refiere a las horas de montaje. Los costes de producción por unidades para cada uno de los circuitos se presentan en las tablas 5.6, 5.7 y 5.8.

Se considera el siguiente coste de fabricación para cada circuito:

3 horas $\times$ 6 euros por hora=18 euros

Unidades	1	10	100	1000	10000
Material (euros)	9.67	91.68	870.3	7736	67690
Coste de fabricación (euros)	18	180	1800	18000	180000
Coste de diseño	2809.67	2809.67	2809.67	2809.67	2809.67
Coste total de producción(euros)	2837.34	3081.35	5479.67	28545.67	250499.67
Precio final + I.V.A. (euros) de cada circuito según el número de unidades que se fabriquen	3688.54	400.57	71.23	37.10	32.56

Cuadro 5.6: Coste de producción y precio final sin I.V.A. del circuito de la aplicación práctica

Unidades	1	10	100	1000	10000
Material (euros)	10.35	98.325	903.15	8280	72450
Coste de fabricación (euros)	18	180	1800	18000	180000
Coste de diseño	2810.35	2810.35	2810.35	2810.35	2810.35
Coste total de producción(euros)	2828.7	3088.67	5513.15	13641.85	255260.35
Precio final + I.V.A. (euros) de cada circuito según el número de unidades que se fabriquen	3690.31	401.52	71.67	37.73	33.18

Cuadro 5.7: Coste de producción del circuito que contiene el puerto I2C

Unidades	1	10	100	1000	10000
Material (euros)	9.93	94.335	893.7	7944	69510
Coste de fabricación (euros)	18	180	1800	18000	180000
Coste de diseño	2809.93	2809.93	2809.93	2809.93	2809.93
Coste total de producción(euros)	2837.86	3084.26	5503.63	28753.93	252319.93
Precio final + I.V.A. (euros) de cada circuito según el número de unidades que se fabriquen	3689.21	400.95	71.54	37.38	32.80

Cuadro 5.8: Coste de producción del circuito que contiene el puerto paralelo

Capítulo 6

Conclusiones

6.1. Resultado final

Después del trabajo realizado se han alcanzado todos los objetivos marcados pero no sin antes tener que superar todas las dificultades que van surgiendo sobre la marcha. Se han conseguido tres circuitos de tamaño ajustado gracias al empleo de circuitos SMD, cuyo montaje ha sido laborioso aunque, finalmente, el esfuerzo queda compensado con creces una vez se aprecia el resultado.

El circuito correspondiente a la aplicación práctica nos permite definir el contenido del display LCD de TECDIS de una forma rápida y sencilla tal y como se buscaba, sin tener que utilizar un PC para dicha tarea. Por otro lado los circuitos con puerto Paralelo e I2C marcan una buena base para el desarrollo de nuevas aplicaciones basadas en el empleo de mandos a distancia tal y como se indica en la siguiente sección.

6.2. Trabajo futuro

Utilizando como punto de partida este proyecto se pueden realizar ciertas ampliaciones y avances que se ven en los siguientes puntos:

- Desarrollo de aplicaciones para PC que utilicen el control remoto por infrarrojos a través del puerto serie RS-232 del ordenador.

- Desarrollo de nuevos dispositivos que dispongan de un puerto I2C, un puerto Paralelo o bien un puerto RS-232 y que se usen también el sistema de control a distancia por infrarrojos. Estos dispositivos se conectarían como esclavos de nuestro módulo universal de infrarrojos.

- Desarrollo de software basado en la rutina de recepción que ha sido diseñada para el empleo de los puertos para actuadores de potencia disponibles.

Apéndice A

Listados

A.1. Aplicación práctica

A.1.1. Proyecto.asm

Listing A.1: Programa

```
;*****************************************************
;                                                    *
; Receptor de mando a distancia por infrarrojos      *
;                                                    *
;*****************************************************
;                                                    *
; Carlos Arevalo Sillero                             *
;                                                    *
;*****************************************************
; Se utiliza un Reloj de cuarzo de 4MHz.             *
; La entrada de la se al es RB0, activa a nivel bajo. *
;*****************************************************

        list      p=16F628, R=DEC
        ERRORLEVEL -305
        #include <p16F628.inc>

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC & _LVP_OFF

                    ; Definicion de constantes.

Direccion_ceros equ     0x43
Direccion_ceros_mas_uno equ 0x44
Direccion_unos  equ     0xAC
DIR_TECLA1      equ     0x129
DIR_TECLA1_0    equ     0xA0
DIR_TECLA2      equ     0x12C
DIR_TECLA2_0    equ     0xA1
DIR_TECLA3      equ     0x12F
DIR_TECLA3_0    equ     0xA2
DIR_TECLA4      equ     0x132
DIR_TECLA4_0    equ     0xA3
DIR_TECLA5      equ     0x135
```

```
35 DIR_TECLA5_0     equ     0xA4
   DIR_TECLA6       equ     0x138
   DIR_TECLA6_0     equ     0xA5
   DIR_TECLA7       equ     0x13B
   DIR_TECLA7_0     equ     0xA6
40 DIR_TECLA8       equ     0x13E
   DIR_TECLA8_0     equ     0xA7
   DIR_TECLA9       equ     0x141
   DIR_TECLA9_0     equ     0xA8
   DIR_TECLA10      equ     0x144
45 DIR_TECLA10_0    equ     0xA9
   DIR_TECLA11      equ     0x147
   DIR_TECLA11_0    equ     0xAA
   DIR_TECLA12      equ     0x14A
   DIR_TECLA12_0    equ     0xAB
50 Dir_Senal        equ     0x121
   numero_de_bytes equ     8

                        ; Definicion de variables

55         cblock h'20'

   CUENTA_TIEMPO          ; Ayuda a temporizar la muestra de caracteres en el LCD.
   MEDIDACERO             ; Medida cero.
   APRENDIZAJE_REG        ; Cada bit tiene una tarea de control.
60 COMANDO                ; Numero de comando actual.
   medidauno_ref          ; Tolerancia en la toma de tiempos.
   auxiliar_6             ; Variable auxiliar de uso general.
   medidacero_ref         ; Tolerancia en la toma de tiempos.
   auxiliar_7             ; Variable auxiliar de uso general.
65 CAR_LIN                ; Numero de caracteres por linea.
   auxiliar_3             ; Variable auxiliar de uso general.
   VISUALIZACION          ; Visualizacion actual en el LCD.
   PROGRAMACION_REG       ; Cada bit tiene una tarea de control.
   TECLA_X                ; Comando seleccionado con el mando.
70 Retardo_ms_Contador
   auxiliar_2             ; Variable auxiliar de uso general.
   auxiliar               ; Variable auxiliar de uso general.
   VELOCIDAD_SCROLL       ; Velocidad actual del scroll.
   RETARDO_M_ALTERN       ; Retardo de modo alternativo.
75 auxiliar_4             ; Variable auxiliar de uso general.
   DIRECCION_EEPROM       ; Direccion de almacenamiento de la EEPROM.
   auxiliar_9             ; Variable auxiliar de uso general.
   auxiliar_10            ; Variable auxiliar de uso general.
   FSR_TEMP1              ; Salvaguarda el FSR.
80 FSR_TEMP2              ; Salvaguarda el FSR.
   auxiliar_5             ; Variables auxiliar de uso general.
   w_temp                 ; Salvaguarda W.
   status_temp            ; Salvaguarda STATUS.
   nummedidascero         ; Numero de medidas cero.
85 MEDIDASCEROIGUALES     ; Numero de medidas cero iguales.
   nummedidasuno          ; Numero de medidas uno.
   MEDIDAUNO              ; Medida uno.
   MEDIDASUNOIGUALES      ; Numero de medidas uno iguales.
   temp_medcero           ; Guarda medida cero.
90 temp_meduno            ; Guarda medida uno.
   MEDIDAUNOPEQUE         ; Medida uno mas pequena.
   MEDIDACEROPEQUE        ; Medida cero mas pequena.
   cuenta                 ; Contador.
```

```
tiempos1                ; Zona de almacenamiento de ceros.

        endc

        cblock h'A0'
TECLA1_0                ; Primer byte de las teclas.
TECLA2_0
TECLA3_0
TECLA4_0
TECLA5_0
TECLA6_0
TECLA7_0
TECLA8_0
TECLA9_0
TECLA10_0
TECLA11_0
TECLA12_0
tiempos2                ; Zona de almacenamiento de unos.

        endc

        cblock h'120'
COMANDO_LCD             ; Comando que se envia al LCD.
byte1                   ; Bytes obtenidos de la senal IR.
byte2
byte3
byte4
byte5
byte6
byte7
byte8

        endc


;-----------------------------------------------------------------------
; Va al inicio del programa
;-----------------------------------------------------------------------

        ORG     0x000           ; Vector de reset.
        goto    Main            ; Va al inicio del programa.

;-----------------------------------------------------------------------
; Controlador de interrupcion.
;
; Comprueba el modo de funcionamiento y si es modo aprendizaje llama a
; la subrrutina MODO_APRENDIZAJE; en caso contrario salta y despues de
; leer la senal compara con los valores de las teclas almacenados y si
; coincide con alguno envia por RS232 el comando correspondiente al
; display.
;-----------------------------------------------------------------------

        ORG     0x004               ; Vector de interrupcion: recibe senal.
        movwf   w_temp              ; Guarda el contenido de W...
        movf    STATUS,w            ; ...y guarda el contenido de status.
        movwf   status_temp
        bcf     INTCON,GIE
```

```
        btfss   PORTA,0
155     goto    M_PROGR_ENVIA       ; Esta en modo programacion.
        call    MODO_APRENDIZAJE
        goto    SALIR_YA            ; Acabo el aprendizaje.

        ; Modo programacion del display TECDIS.
160
M_PROGR_ENVIA:
        bsf     PROGRAMACION_REG,2  ; Indica que se pulso una nueva tecla.
        call    Rec_RC5_RE80        ; Recibe los datos.

165     ; Discrimina la tecla pulsada para averiguar la operacion que
        ; se desea realizar sobre el display.

        call    DETECTA_TECLA       ; Detecta numero de tecla.
        movwf   auxiliar            ; Guarda el numero de tecla.
170     btfss   PROGRAMACION_REG,3
        goto    BUSQUEDA_DE_OPCION  ; No hay seleccion de opcion.
        bsf     PROGRAMACION_REG,1  ; Elegida opcion del LCD.
        goto    SALIR_YA            ; Sale de la subrrutina

175 BUSQUEDA_DE_OPCION:             ; Busca la opcion del LCD

        movlw   10
        subwf   auxiliar,0          ; Enter?
        btfsc   STATUS,Z
180     goto    ELEGIDA_OPCION      ; Elije opcion.

        movlw   2                   ; Busca opcion.
        subwf   auxiliar,0          ; Arriba?
        btfsc   STATUS,Z
185     goto    MUESTRA_COMANDO_UP

        movlw   8                   ; Busca opcion.
        subwf   auxiliar,0          ; Abajo?
        btfsc   STATUS,Z
190     goto    MUESTRA_COMANDO_DOWN

        movlw   11                  ; Escape?
        subwf   auxiliar,0
        btfss   STATUS,Z
195     goto    SALIR_YA

        movlw   4
        subwf   COMANDO,0
        btfss   STATUS,Z
200     call    ELIGE_MENS          ; Comando que muestra mensaje
                                    ; definido si lo hay para el LCD.
        goto    SALIR_YA            ; Sale de la subrrutina.

MUESTRA_COMANDO_UP:                 ; Muestra comando siguiente.
205     movlw   4
        subwf   COMANDO,0           ; El comando mas alto tiene el
        btfss   STATUS,Z            ; numero 4.
        goto    INCREMENT           ; Sube comando.
        movlw   1
210     movwf   COMANDO             ; El comando mas bajo tiene el
        goto    SALTA_INC           ; numero 1.
```

```
INCREMENT:
      INCF    COMANDO              ; Sube el numero de comando
                                   ; actual.
SALTA_INC:
      call    ESCRIBE_COMANDO      ; Muestra el comando en el LCD.
      goto    SALIR_YA             ; Sale de la subrrutina.

MUESTRA_COMANDO_DOWN:              ; Muestra comando anterior
      movlw   1
      subwf   COMANDO,0
      btfss   STATUS,Z
      goto    DECREMENT
      movlw   4
      movwf   COMANDO
      goto    SALTA_DEC

DECREMENT:                         ; Baja el numero de comando
      DECF    COMANDO              ; actual.

SALTA_DEC:
      call    ESCRIBE_COMANDO      ; Muestra el nuevo comando
      goto    SALIR_YA             ; en el LCD.

ELEGIDA_OPCION:
      bsf     PROGRAMACION_REG,3 ; Eligio opcion

SALIR_YA:

      bcf     INTCON,INTF          ; Restaura el flag de interrupcion,
      movf    status_temp,w        ; restaura el contenido de status...
      movwf   STATUS
      swapf   w_temp,f
      swapf   w_temp,w             ; ...y restaura el contenido de w.
      retfie                       ; Retorna de la interrupcion.


;-------------------------------------------------------------------
; Tabla de mensajes que se muestran en el LCD
;-------------------------------------------------------------------

MENSAJE_ALTERNATIVO:
      addwf PCL
      dt "ALTERNATIVO",0
      dt "CARACT./L",0
      dt "SCROLL",0
      dt "MITAD",0
      dt "ESCRIBIR",0
      dt "VELOCIDAD",0
      dt "RETARDO",0
      dt ">NORMAL",0
      dt "PARPADEO␣",0
      dt "FIN!",0
      dt "TECLA␣",0
      dt "ENTER?",0
      dt "ESCAPE?",0
      dt "VISUALIZACION",0
      dt "SIN␣PROGRAMAR",0
```

```
;-----------------------------------------------------------------------
;                Programa principal
;-----------------------------------------------------------------------

Main

        movlw   0x07              ; Pone los comparadores en off
        movwf   CMCON             ; permitiendo el uso de los pines
                                  ; I/O.
        bsf     STATUS,RP0        ; Banco 1.
        movlw   B'10000110'       ; Prescaler del Timer0 = 128, int. en flanco
        movwf   OPTION_REG        ; descendente.
        movlw   B'00000011'
        movwf   TRISB             ; RB0 y RB1 son entradas.
        bsf     TRISA,0           ; Configura como entrada RA0.
                                  ; RA0 indica modo de aprendizaje.
                                  ; o programacion.
        bcf     STATUS,RP0        ; Banco 0.
        call    INI_RS232         ; Inicializacion del puerto RS-232.
        clrf    APRENDIZAJE_REG   ; Inicializa los registros.
        clrf    PROGRAMACION_REG

        movlw   1
        call    LEE_EEPROM        ; Comprueba si hay texto en la EEPROM.
        sublw   0xFF
        btfss   STATUS,Z
        goto    SI_TEXTO          ; Si hay texto.

        movlw   1
        movwf   VISUALIZACION     ; Inicia con el modo normal de visualizacion.
        movlw   0x90
        movwf   CAR_LIN           ; Comienza con 16 caracteres por linea.
        goto    salta_SI_TEXTO

SI_TEXTO:
        bsf     APRENDIZAJE_REG,7; Indica que hay texto en la EEPROM.
        movlw   124
        call    LEE_EEPROM        ; Lee el numero de caracteres del texto
        movwf   DIRECCION_EEPROM
        movlw   125
        call    LEE_EEPROM        ; Lee el modo de visualizacion.
        movwf   VISUALIZACION
        movlw   126
        call    LEE_EEPROM        ; Lee el numero de caracteres por linea.
        movwf   CAR_LIN

salta_SI_TEXTO:
        movlw   4
        movwf   COMANDO           ; Contiene el valor del comando actual.

        movlw   5
        movwf   VELOCIDAD_SCROLL  ; 5 es la velocidad mas lenta.

        movlw   40
        movwf   RETARDO_M_ALTERN  ; 40 es la velocidad mas lenta en el
                                  ; modo alternativo.
        bsf     INTCON,INTE       ; Habilita la interrupcion externa.
```

```
330         btfss   PORTA,0            ; Comprueba el modo de funcionamiento.
            goto    PREBUCLE2          ; Modo aprendizaje.

    PREBUCLE1:                         ; Modo programacion.
            bcf     INTCON,GIE         ; Deshabilita interrupciones.
335         movlw   '0'
            movwf   auxiliar
            call    ENVIA_MSJ_TECLA_2; Pide la tecla numero 0
                                       ; desde el LCD.

340 bucle1:
            bsf     INTCON,GIE         ; Habilita las interrupciones.
                                       ; Despues de la inicializacion del
                                       ; sistema espera interrupcion por rb0.
            btfss   PORTA,0
345         goto    PREBUCLE2          ; Modo programacion.
            goto    bucle1             ; Modo Aprendizaje.

    PREBUCLE2:
            bcf     INTCON,GIE         ; Deshabilita las interrupciones.
350         call    ELIGE_MENS         ; Muestra mensaje definido para el LCD
                                       ; si lo hay, y si no existe solicita que
                                       ; se introduzca con el comando escribir.
    bucle2:
            bsf     INTCON,GIE         ; Habilita las interrupciones.
355         btfsc   PORTA,0            ; Comprueba modo de funcionamiento.
            goto    PREBUCLE1          ; Modo aprendizaje.
            btfss   PROGRAMACION_REG,3 ; Comprueba si se eligio una opcion
                                       ; para el display.
            goto    bucle2             ; Modo programacion.
360

    ;-----------------------------------------------------------------
    ; En las siguientes lineas se programa el display deacuerdo con
    ; las teclas pulsadas por el mando a distancia.
365 ;-----------------------------------------------------------------

            bcf     INTCON,GIE         ; Deshabilita las interrupciones.
                                       ; Comprueba si se eligio la primera de
                                       ; las opciones.
370         goto    comprueba_otra_opcion_0

    V_SCROLL:
            movfw   VELOCIDAD_SCROLL
            movwf   auxiliar_4         ; Salvaguarda la velocidad del scroll
375
    prewaiting:
            call    MSJ_VELOCIDAD_SCROLL_2
            call    R_TECLAS_PEQUE     ; Retardo para no pasar de opcion de
                                               ; forma involuntaria
380 waiting:
            bsf    INTCON,GIE          ; Habilita las interrupciones.
                                       ; Comprueba si se tomo la velocidad de
                                       ; SCROLL.
            btfss   PROGRAMACION_REG,1
385         goto    waiting            ; Espera senal del receptor.
            bcf     INTCON,GIE         ; Deshabilita las interrupciones.
            bcf     PROGRAMACION_REG,1
```

```
        movlw   11                  ; Escape?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    EFECTO_VISUAL_SCROLL

        movlw   2                   ; Arriba?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    VELO_UP             ; Aumenta la velocidad de scroll.

        movlw   8                   ; Abajo?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    VELO_DOWN           ; Disminuye la velocidad de scroll.

        movlw   10                  ; Enter?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    VELOCIDAD_CORRECTA

        goto    waiting             ; No hubo tecla correcta.

VELO_UP:
        movlw   1                   ; Velocidad de scroll maxima.
        subwf   VELOCIDAD_SCROLL,0
        btfsc   STATUS,Z
        goto    prewaiting
        decf    VELOCIDAD_SCROLL ; Incrementa la velocidad del SCROLL
        goto    prewaiting

VELO_DOWN:
        movlw   5                   ; Velocidad de scroll minima.
        subwf   VELOCIDAD_SCROLL,0
        btfsc   STATUS,Z
        goto    prewaiting
        incf    VELOCIDAD_SCROLL ; Incrementa la velocidad de scroll.
        goto    prewaiting

VELOCIDAD_CORRECTA:
        movfw   VELOCIDAD_SCROLL
        movwf   auxiliar_4          ; Copia la velocidad de scroll.


        bsf     auxiliar_4,7        ; Pone a uno el bit mas significativo
                                    ; de la velocidad para que se guarde
                                    ; en la memoria no volatil del LCD.
        movfw   auxiliar_4
        movwf   auxiliar_2          ; Copia de nuevo la velocidad.
        movlw   0x23                ; Comando de modificacion de la
                                    ; velocidad de scroll.
        call    ENVIA_PROTOCOLO     ; Envia el comando al display
                                    ; y muestra el mensaje definido
                                    ; con el modo de visualizacion
                                    ; que este seleccionado.
        goto    SALE_DE_PROGRAMACION_ELIGE_MENS

comprueba_otra_opcion_0:
        ; Comprueba si es la opcion de "VISUALIZACION".
        ; TECLA_X contiene el valor de la opcion elegida.
```

```
        ; Si TECLA_X =0x33  '3' -> VISUALIZACION.

        movlw   '3'
        subwf   TECLA_X,0          ; W = TECLA_X - W
        btfss   STATUS,Z
        goto    comprueba_otra_opcion_1
        goto    salta_EFECTO_VISUAL

EFECTO_VISUAL_SCROLL:
        movfw   auxiliar_4         ; Recupera la velocidad de scroll.
        movwf   VELOCIDAD_SCROLL
        goto    salta_EFECTO_VISUAL

EFECTO_VISUAL:
        movfw   auxiliar_4         ; Recupera el valor de retardo.
        movwf   RETARDO_M_ALTERN

salta_EFECTO_VISUAL:
        movfw   VISUALIZACION
        movwf   auxiliar_4         ; Salvaguarda el modo de visualizacion
                                   ; y muestra mensaje con la opcion visual.
        call    MSJ_OPCION_VISUAL_2

prewaiting_MENS:
        bcf     PROGRAMACION_REG,7; Borra el bit. No va al paso anterior.
        bcf     PROGRAMACION_REG,1; Borra el bit. Aun no se tomo opcion
                                   ; de visualizacion.
        call    MSJ_OPCION_VISUAL_2

waiting_MENS:
        bsf     INTCON,GIE         ; Habilita las interrupciones.
        btfss   PROGRAMACION_REG,1; Tomo modo de visualizacion?
        goto    waiting_MENS       ; Espera senal del receptor.
        bcf     INTCON,GIE         ; Deshabilita las interrupciones.
        bcf     PROGRAMACION_REG,1; Borra la indicacion de eleccion
                                   ; de visualizacion.

        movlw   11                 ; Escape?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    SALE_DE_PROGRAMACION_VIS

        movlw   2                  ; Sube?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    OPC_UP             ; Cambia opcion de visualizacion.

        movlw   8                  ; Baja?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    OPC_DOWN           ; Cambia opcion de visualizacion.

        movlw   10                 ; ENTER?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    EN_PROGRAMACION_VISUALIZACION
        goto    waiting_MENS       ; No se pulso tecla correcta.

OPC_DOWN:
```

```
        movlw   1                   ; Numero mas bajo de los modos
        subwf   VISUALIZACION,0     ; de visualizacion.
        btfsc   STATUS,Z
        goto    pasa_a_5
        decf    VISUALIZACION       ; Cambia de visualizacion.
        goto    prewaiting_MENS

OPC_UP:
        movlw   5                   ; Numero mas alto de los modos
        subwf   VISUALIZACION,0     ; de visualizacion.
        btfsc   STATUS,Z
        goto    pasa_a_1
        incf    VISUALIZACION       ; Cambia de visualizacion.
        goto    prewaiting_MENS

pasa_a_5:
        movlw   5                   ; Salvaguarda el modo de
        movwf   VISUALIZACION       ; visualizacion.
        goto    prewaiting_MENS

pasa_a_1:
        movlw   1
        movwf   VISUALIZACION       ; Salvaguarda el modo de
        goto    prewaiting_MENS     ; visualizacion.

comprueba_otra_opcion_1:
        ; Comprueba si es la opcion de "ENVIAR MENSAJE".
        ; TECLA_X contiene el valor de la opcion elegida.
        ; Si TECLA_X =0x32  '2' -> ENVIAR MENSAJE.

        movlw   '2'
        subwf   TECLA_X,0           ; W = TECLA_X - W
        btfss   STATUS,Z            ; Si cero -> ENVIAR MENSAJE
        goto    comprueba_otra_opcion_2

        movfw   DIRECCION_EEPROM    ; Salvaguarda el numero de
        movwf   auxiliar_4          ; caracteres que hay en la
        clrf    DIRECCION_EEPROM    ; EEPROM.
        clrf    auxiliar_5          ; Almacena el numero de veces
                                    ; seguidas que se pulsa la misma
                                    ; tecla
        bsf     auxiliar_6,7        ; Pone a uno el bit mas significativo
                                    ; para que su valor actual no se
                                    ; confunda con el valor de una tecla
                                    ; pulsada.
        bcf     PROGRAMACION_REG,7; Borra el bit que indica que se va
                                    ; al paso anterior.
        bsf     APRENDIZAJE_REG,0 ; Se pone a 1 el bit que indica que
                                    ; es el primer caracter de una nueva
                                    ; posicion en la pantalla del LCD.
        bcf     INTCON,TOIF

PIDE_TEXTO:

salta_PIDE_TEXTO:
        movlw   0                   ; Comprueba si hay caracteres en la
                                    ; EEPROM
        subwf   DIRECCION_EEPROM,0
        btfsc   STATUS,Z
```

```
                                        ; Muestra mensaje por el LCD solicitando
                                        ; la introduccion de texto.
        call    MSJ_PIDE_MENSAJE_2
                                        ; Comprueba si va al paso anterior.
        btfsc   PROGRAMACION_REG,7
        goto    SALE_DE_PROGRAMACION_ESCRIBIR
        movlw   0
        subwf   DIRECCION_EEPROM,0
        btfsc   STATUS,Z
        goto    waiting_TEXTO
        movlw   13                      ; Comprueba si hay mas de 12 caracteres
        subwf   DIRECCION_EEPROM,0
        btfsc   STATUS,C
        goto    SOLO_16_CHAR            ; Hay mas de 12 caracteres para mostrar
        movfw   DIRECCION_EEPROM        ; en el LCD; solo se muestran en la
        movwf   cuenta                  ; pantalla los 13 ultimos + "TXT:".
        clrf    auxiliar_2
        goto    MUESTRA_CHAR            ; Muestra los caracteres en el LCD.

SOLO_16_CHAR:
        movlw   12
        movwf   cuenta                  ; cuenta = 12.
        subwf   DIRECCION_EEPROM,0
        movwf   auxiliar_2              ; DIRECCION_EEPROM - 12 = auxiliar_2.

MUESTRA_CHAR:
        call    MSJ_PIDE_TXT_2          ; Muestra el mensaje "TXT:"
        movfw   auxiliar_2

CHAR_CHAR:
        call    LEE_EEPROM              ; Lee caracter de la EEPROM.
                                        ; W = dato.
        call    XOR_ENVIA               ; Envia el caracter por el puerto serie.
        incf    auxiliar_2              ; Apunta a la siguiente direccion
        movfw   auxiliar_2              ; de la EEPROM.
        decfsz  cuenta
        goto    CHAR_CHAR               ; Sigue leyendo caracteres.
        movlw   0x03                    ; Envia ETX y CHK.
        call    send
        movfw   auxiliar_7
        call    send

        clrf    CUENTA_TIEMPO
        btfss   APRENDIZAJE_REG,0       ; Si es uno es el primer caracter
        goto    waiting_TEXTO           ; de una nueva posicion.
        bcf     APRENDIZAJE_REG,0       ; A cero otra vez.

VUELTAS:
        call    INCREMENTA_CUENTA       ; Cuenta de 10 en 10ms.
        movlw   35
        subwf   CUENTA_TIEMPO,0         ; Introduce un retardo de 350ms
        btfss   STATUS,C                ; entre caracteres para que de tiempo
        goto    VUELTAS                 ; a verles en la pantalla.

waiting_TEXTO:
        bsf     INTCON,GIE              ; Habilita las interrupciones globales.
        call    INCREMENTA_CUENTA
        btfss   PROGRAMACION_REG,2      ; Comprueba si se pulso una tecla.
        goto    waiting_TEXTO
```

```
625         bcf     INTCON,GIE          ; Deshabilita las interrupciones.
            bcf     PROGRAMACION_REG,2
            movlw   11                  ; Numero de la tecla escape.
            subwf   auxiliar,0          ; W = auxiliar - W.
            btfsc   STATUS,Z
630         goto    E_DEC_DIR_ESC       ; Si es la tecla ESC no se decrementa la
                                        ; direccion de la EEPROM.
            btfsc   APRENDIZAJE_REG,0   ; Si es uno es el primer caracter de una
                                        ; nueva posicion.
            goto    RETARDO_NUEVA_POS_CAR
635         movlw   30                  ; 30 cuentas equivalen a 300ms.
            subwf   CUENTA_TIEMPO,0
            btfss   STATUS,C
            goto    waiting_TEXTO       ; Como no ha pasado el tiempo adecuado desde
                                        ; la ultima pulsacion vuelve a esperar
640                                     ; la llegada de una tecla.

            movlw   120                 ; 120 cuentas = 1,2 segundos.
            subwf   CUENTA_TIEMPO,0     ; Si pasa un tiempo superior a 1,2 seg
            btfsc   STATUS,C            ; avanza una posicion en la pantalla
645                                     ; de forma automatica.
            bsf     APRENDIZAJE_REG,0   ; No decrementa la direccion de la EEPROM.

      RETARDO_NUEVA_POS_CAR:
            movlw   0
650         subwf   DIRECCION_EEPROM,0
            btfsc   STATUS,Z            ; Si no se escribio ningun caracter no
                                        ; no espera la tecla "ENTER".
            goto    SALTA_ENTER
            movlw   10                  ; Numero de la tecla ENTER.
655         subwf   auxiliar,0
            btfsc   STATUS,Z
            goto    MANDA_TEXTO_RS232   ; Manda el texto al LCD por el puerto serie.

      SALTA_ENTER:
660
      COMPROBANDO:
            movlw   0
            subwf   DIRECCION_EEPROM,0  ; Si no hay caracteres almacenados no
            btfsc   STATUS,Z            ; no decrementa la direccion apuntada en
665         goto    EVITA_DEC_DIR       ; la EEPROM.

            btfsc   APRENDIZAJE_REG,0   ; Si es cero decrementa la direccion
            goto    EVITA_DEC_DIR       ; apuntada en la EEPROM.
            movfw   auxiliar
670         subwf   auxiliar_6,0        ; Tecla anterior - Tecla actual.
            btfss   STATUS,Z            ; Si no es la misma tecla no decrementa la
            goto    EVITA_DEC_DIR       ; direccion de la EEPROM.
            bsf     PROGRAMACION_REG,0  ; Indica que la posicion de caracter en el
                                        ; LCD no cambia.
675         decf    DIRECCION_EEPROM    ; Decrementa en uno la direccion de EEPROM.

      EVITA_DEC_DIR:
            movlw   120                 ; Numero maximo de caracteres.
            subwf   DIRECCION_EEPROM,0  ; Detecta si hay 120 caracteres almacenados.
680         btfsc   STATUS,Z
            goto    waiting_TEXTO       ; No hay 120 caracteres aun.

      E_DEC_DIR_ESC:                    ; Hay 120 caracteres.
```

```
        movlw   9
        movwf   auxiliar_2
        movfw   auxiliar
        subwf   auxiliar_2,1
        btfsc   STATUS,C            ; Si el resultado es 1 la tecla pulsada
        goto    TECLA_TRATAMIENTO   ; esta entre 0 y 9.
        movlw   11                  ; Escape?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    TX_ESCAPE
        goto    waiting_TEXTO       ; No se pulso ninguna tecla correcta.

TECLA_TRATAMIENTO:
        btfss   PROGRAMACION_REG,0  ; Comprueba si la posicion del caracter
                                    ; es la misma.
        bsf     auxiliar_6,7        ; Poniendo a 1 el bit mas significativo
                                    ; de este registro se indica que la
                                    ; posicion del caracter en el LCD avanzo.
        bcf     PROGRAMACION_REG,0
        movfw   auxiliar
        subwf   auxiliar_6,0        ; Tecla anterior - Tecla actual.
        btfss   STATUS,Z
        clrf    auxiliar_5          ; Primera pulsacion de tecla actual.

        movlw   1                   ; Tecla 1?
        subwf   auxiliar,0          ; W = auxiliar - 0.
        btfsc   STATUS,Z
        goto    TECLA_CHAR_UNO      ; Tecla 1.
        movlw   0                   ; Tecla 0?
        subwf   auxiliar,0          ; W = auxiliar - 0.
        btfsc   STATUS,Z
        goto    TECLA_CHAR_CERO     ; Tecla 0.
        movlw   0x38
        addwf   auxiliar_5,0        ; Numero de veces pulsada la
                                    ; misma tecla.
        movwf   auxiliar_2
        movfw   auxiliar
        movwf   cuenta              ; Valor del contador necesario
        incf    cuenta              ; para obtener en el bucle BUSCA_CHAR
        movlw   3                   ; el caracter

BUSCA_CHAR:
        addwf   auxiliar_2,1        ; Busca el caracter actual en funcion
        decfsz  cuenta              ; de la tecla pulsada y las veces que
        goto    BUSCA_CHAR          ; esta se ha pulsado.
        btfsc   PROGRAMACION_REG,5  ; Comprueba si se escribio la ' '
                                    ; en la pulsacion de tecla anterior.
        goto    CONTI               ; Se acaba de mostrar la ' '.
        movlw   0x4F                ; Comprueba si es el codigo hx de la 'O'.
        subwf   auxiliar_2,0
        btfss   STATUS,Z            ; Es el codigo hx de la 'O'.
        goto    CONTI
        movlw   165                 ; Codigo hx de la ' '.
        movwf   auxiliar_2          ; Se guarda la ' '.
        bsf     PROGRAMACION_REG,5  ; Indica que se mostro ' '.
        goto    EVITA_CHAR_NUM

CONTI:
        bcf     PROGRAMACION_REG,5
```

```
                                ; Si es la tecla numero 8 se incrementa el
                                ; numero del caracter en 1 porque la tecla
                                ; numero 7 tiene 5 caracteres (uno mas que
                                ; el resto de las teclas excepto que la
                                ; tecla 6).
        movlw   8               ; Tecla 8?
        subwf   auxiliar,0      ; w = auxiliar - 8.
        btfsc   STATUS,Z
        incf    auxiliar_2      ; Se incrementa el caracter de la tecla en 1
                                ; para que envie los caracteres 'T', 'U' y 'V'.
        movlw   9               ; Tecla 9?
                                ; Carateres W X e Y.
        subwf   auxiliar,0      ; w = auxiliar - 9.
        btfsc   STATUS,Z
        incf    auxiliar_2
        incf    auxiliar_5      ; Incrementa el numero de veces que se ha
                                ; pulsado la misma tecla.
        movlw   5
        subwf   auxiliar_5,0
        btfsc   STATUS,Z
        goto    TECLA_7_9       ; Escribe el '7' de la tecla numero 7.
        movlw   4
        subwf   auxiliar_5,0    ; Si es 4 empieza de nuevo con el primer
                                ; caracter; se recorrieron todos los
                                ; caracteres de la tecla.
        btfss   STATUS,Z
        goto    EVITA_CHAR_NUM
        movlw   7               ; Tecla 7?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    TECLA_PQRS      ; Caracteres 'P', 'Q', 'R' y 'S'.
        movlw   9               ; Tecla 9?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    TECLA_WXYZ      ; Muestra caracteres 'W', 'X', 'Y' y 'Z'.
        clrf    auxiliar_5
        movfw   auxiliar
        movwf   auxiliar_2
        movlw   0x30            ; 0x30 + num tecla = caracter numerico.
        addwf   auxiliar_2,1
        goto    EVITA_CHAR_NUM

TECLA_PQRS:
        movlw   0x38            ; Valor desde el que se parte para
                                ; encontrar todos los caracteres.
        addwf   auxiliar_5,0    ; Suma el numero de veces pulsada la
        movwf   auxiliar_2      ; misma tecla para encontrar el caracter
        decf    auxiliar_2      ; actual que se debe mostrar en el LCD.
        movfw   auxiliar
        movwf   cuenta          ; "cuenta" contiene el numero de tecla
        incf    cuenta          ; que se suma (x3) a 0x38 y al numero de
        movlw   3               ; veces pulsada una tecla para obtener el
                                ; caracter actual.
BUSCA_CHAR_2:
        addwf   auxiliar_2,1
        decfsz  cuenta
        goto    BUSCA_CHAR_2    ; Obtiene el caracter actual.
        goto    EVITA_CHAR_NUM
```

```
TECLA_CHAR_CERO:
        bsf     PROGRAMACION_REG,4 ; Indica que la tecla pulsada es 0.

TECLA_CHAR_UNO:                    ; Tecla 1.
        movlw   0
        subwf   auxiliar_5,0       ; Primer caracter de la tecla?
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION
        movlw   0x20               ; W = ' ' = 0x20
        btfss   PROGRAMACION_REG,4 ; Comprueba si la tecla pulsada es 0.
        addlw   0x02               ; W = '"' = 0x22
        movwf   auxiliar_2
        goto    SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION:
        movlw   1
        subwf   auxiliar_5,0       ; Primer caracter de la tecla?
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION_2
        movlw   0x30
        movwf   auxiliar_2
        btfss   PROGRAMACION_REG,4
        incf    auxiliar_2
                                   ; 0x30 = '0'
                                   ; 0x31 = '1'
        goto    SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION_2:
        movlw   2
        subwf   auxiliar_5,0       ; Primer caracter de la tecla?
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION_3
        movlw   0x2B
        btfss   PROGRAMACION_REG,4
        addlw   0x02
                                   ; 0x2B = '+'
                                   ; 0x2D = '-'
        movwf   auxiliar_2
        goto    SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION_3:

        movlw   3
        subwf   auxiliar_5,0       ; Primer caracter de la tecla?
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION_4
        movlw   0x27
        btfsc   PROGRAMACION_REG,4
        addlw   0x38
        movwf   auxiliar_2
                                   ;  '''  = 0x27
                                   ;  '_'  = 0x5F

        goto    SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION_4:

        movlw   4
        subwf   auxiliar_5,0       ; Primer caracter de la tecla?
```

```
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION_5
        movlw   0x2F
        btfss   PROGRAMACION_REG,4
        addlw   0xB
        movwf   auxiliar_2
                                ; 0x3A = ':'
                                ; 0x2F = '/'

        goto SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION_5:
        movlw   5
        subwf   auxiliar_5,0        ; Primer caracter de la tecla?
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION_6
        movlw   0x3F
        btfsc   PROGRAMACION_REG,4
        addlw   0x1D
        movwf   auxiliar_2

                                ; 0x3F = '?'
                                ; 0x5C = '\'
        goto    SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION_6:
        movlw   6
        subwf   auxiliar_5,0        ; Primer caracter de la tecla?
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION_7
        movlw   40
        btfss   PROGRAMACION_REG,4
        addlw   128
        movwf   auxiliar_2

                                ; 168 = ' '
                                ; 40 = '('

        goto    SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION_7:
        movlw   7
        subwf   auxiliar_5,0        ; Primer caracter de la tecla?
        btfss   STATUS,Z
        goto    SALTA_CHAR_PUNTUACION_8
        movlw   0x21
        btfsc   PROGRAMACION_REG,4
        addlw   0x08
        movwf   auxiliar_2

                                ; 0x21 = '!'
                                ; 0x29 = ')'

        goto SALTA_CHAR_PUNTUACION_10

SALTA_CHAR_PUNTUACION_8:
        movlw   8
        subwf   auxiliar_5,0        ; Primer caracter de la tecla?
        btfss   STATUS,Z
```

```
920         goto    SALTA_CHAR_PUNTUACION_9
            movlw   62
            btfss   PROGRAMACION_REG,4
            addlw   111
            movwf   auxiliar_2
925
                                        ; 173 = ' '
                                        ; 62  = '>'

            goto    SALTA_CHAR_PUNTUACION_10
930

    SALTA_CHAR_PUNTUACION_9:
            movlw   9
            subwf   auxiliar_5,0        ; Primer caracter de la tecla?
935         btfss   STATUS,Z
            goto    SALTA_CHAR_PUNTUACION_10
            movlw   0x2E
            btfsc   PROGRAMACION_REG,4
            addlw   0x0E
940         movwf   auxiliar_2

                                        ; 0x3C = '<'
                                        ; 0x2E = '.'

945 SALTA_CHAR_PUNTUACION_10:
           incf    auxiliar_5
           movlw   10                   ; Si se recorrieron los 10 caracteres
           subwf   auxiliar_5,0         ; de la tecla vuelve a empezar con la
           btfsc   STATUS,Z             ; primera de ellas.
950        clrf    auxiliar_5
           bcf     PROGRAMACION_REG,4   ; Indica si es la tecla 0 o la tecla 1.
           goto    EVITA_CHAR_NUM

    TECLA_WXYZ:
955        movlw   0x38                 ; Valor inicial desde el que se parte
           addwf   auxiliar_5,0         ; para obtener el caracter actual;
           movwf   auxiliar_2           ; se le suma el numero de tecla (x3)
           movfw   auxiliar             ; mas el numero de veces que fue
           movwf   cuenta               ; pulsada la tecla.
960        incf    cuenta
           movlw   3                    ; Las teclas anteriores tienen 3
    BUSCA_CHAR_3:                       ; caracteres.
           addwf   auxiliar_2,1
           decfsz  cuenta
965        goto    BUSCA_CHAR_3
           goto    EVITA_CHAR_NUM       ; Encuentra el caracter actual.

    TECLA_7_9:
           movfw   auxiliar
970        movwf   auxiliar_2
           movlw   0x30
           addwf   auxiliar_2,1
           clrf    auxiliar_5           ; Se borra porque se han recorrido
                                        ; todos los caracteres de la tecla
975
    EVITA_CHAR_NUM:
                                        ; "auxiliar_2" = caracter valido.
           movfw   DIRECCION_EEPROM     ; W=Direccion actual en EEPROM.
```

```
        call    ESCRIBE_EEPROM      ; Guarda el caracter en la EEPROM.
980     incf    DIRECCION_EEPROM    ; Apunta a la siguiente direccion.
        goto    COPIA_TECLA_TEXTO

    TX_ESCAPE:
        movlw   3                   ; 30ms
985     subwf   CUENTA_TIEMPO,0
        btfss   STATUS,C            ; Si C = 0 se pulso la tecla escape
                                    ; inmediatamente y se vuelve al paso
                                    ; anterior.
        goto    PRE_PIDE_TEXTO
990     movlw   100
        call    Retardo_ms          ; 100ms de retardo.
        movlw   0
        subwf   DIRECCION_EEPROM,0
        btfsc   STATUS,Z            ; Si la direccion de la EEPROM es menor
995                                 ; que cero no decrementa mas.
        goto    PRE_PIDE_TEXTO      ; Vuelve al paso anterior.
        decf    DIRECCION_EEPROM    ; Un caracter menos que tiene en cuenta.
        goto    COPIA_TECLA_TEXTO

1000 PRE_PIDE_TEXTO:
        bsf     PROGRAMACION_REG,7  ; Indica que se va al paso anterior.
        movfw   auxiliar_4
        movwf   DIRECCION_EEPROM    ; Restaura el numero de caracteres de
                                    ; la EEPROM.
1005    goto    salta_PIDE_TEXTO

    COPIA_TECLA_TEXTO:
        movfw   auxiliar
        movwf   auxiliar_6          ; Copia el numero de tecla pulsado
1010    goto    salta_PIDE_TEXTO    ; para comparar con la siguiente
                                    ; tecla que sea pulsada.
    MANDA_TEXTO_RS232:              ; Escribe le texto definido en el LCD
        movfw   auxiliar_4          ; en el modo de visualizacion activo
        movwf   auxiliar_10         ; en este momento.
1015    call    TEXTO_TECDIS
        bsf     APRENDIZAJE_REG,7   ; Indica que se mostro un texto en el
                                    ; display .
                                    ; Sale del modo programacion, y se pone
                                    ; en funcionamiento normal.
1020    goto    SALE_DE_PROGRAMACION_ELIGE_MENS

    comprueba_otra_opcion_2:

        goto    comprueba_otra_opcion_3
1025
    RETARDO_MALTERNATIVO:
        movfw   RETARDO_M_ALTERN
        movwf   auxiliar_4          ; Salvaguarda el valor actual del retardo.

1030 prewaiting_MA:
                                    ; Cambiar caracteres por linea?
        btfsc   PROGRAMACION_REG,6
        call    MSJ_CARACTERES_LINEA_2
                                    ; Cambiar retardo de modo alternativo?
1035    btfss   PROGRAMACION_REG,6
        call    MSJ_RETARDO_2
```

```
waiting_MA:
        bsf     INTCON,GIE          ; Habilita las interrupciones
                                    ; Comprueba si se tomo el nuevo valor
                                    ; de caracteres por linea o de retardo
                                    ; de modo alternativo.
        btfss   PROGRAMACION_REG,1
        goto    waiting_MA          ; Espera senal del receptor.
        bcf     INTCON,GIE          ; Deshabilita interrupciones globales.
        bcf     PROGRAMACION_REG,1

        movlw   11                  ; Escape?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    ELEGIR_SALTO

        movlw   2                   ; Subir?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    UP                  ; Aumenta el retardo o el numero de
                                    ; caracteres por linea.
        movlw   8                   ; Bajar?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    DOWN                ; Disminuye el retardo o el numero de
                                    ; caracteres por linea.
        movlw   10                  ; Enter?
        subwf   auxiliar,0
        btfsc   STATUS,Z
        goto    RETARDO_CORRECTO_MA
        goto    waiting_MA          ; No hubo tecla correcta.

ELEGIR_SALTO:
        btfsc   PROGRAMACION_REG,6 ; Cambiar caracteres linea o retardo?
        goto    SALE_DE_PROGRAMACION_CL_R
        goto    EFECTO_VISUAL

UP:
        btfsc   PROGRAMACION_REG,6 ; Cambiar caracteres linea o retardo?
        goto    C_L_UP

        movlw   40                  ; Numero asociado al retardo mas
        subwf   RETARDO_M_ALTERN,0 ; grande.
        btfsc   STATUS,Z
        goto    prewaiting_MA
        incf    RETARDO_M_ALTERN    ; Incrementa el retardo.
        goto    prewaiting_MA

C_L_UP:
        movlw   0xB8                ; Numero de caracteres por linea
        subwf   CAR_LIN,0           ; mas alto.
        btfsc   STATUS,Z
        goto    prewaiting_MA
        movlw   0x04
        addwf   CAR_LIN,1           ; Suma 4 caracteres linea

        goto    prewaiting_MA

DOWN:
        btfsc   PROGRAMACION_REG,6 ; Cambiar caracteres linea o retardo?
```

```
        goto    C_L_DOWN
        movlw   1                   ; Numero asociado al retardo mas pequeno.
        subwf   RETARDO_M_ALTERN,0
        btfsc   STATUS,Z
        goto    prewaiting_MA
        decf    RETARDO_M_ALTERN    ; Disminuye el retardo del modo alternativo.
        goto    prewaiting_MA

C_L_DOWN:
        movlw   0x90                ; Numero de caracteres por linea mas
        subwf   CAR_LIN,0           ; pequeno.
        btfsc   STATUS,Z
        goto    prewaiting_MA
        movlw   0x04                ; Resta 4 al numero de caracteres linea.
        subwf   CAR_LIN,1
        goto    prewaiting_MA

RETARDO_CORRECTO_MA:
        btfsc   PROGRAMACION_REG,6  ; Cambiar caracteres linea o retardo?
        goto    COMANDO_NUMCARLIN
        movfw   RETARDO_M_ALTERN    ; Copia el retardo del modo alternativo.
        movwf   auxiliar_4

ENVIA_RETARDO:
                                    ; Envia el comando de cambio del retardo.
        bsf     auxiliar_4,7        ; Pone a uno el bit mas significativo de
                                    ; auxiliar_4 para que se guarde el retardo
                                    ; en la memoria no volatil.
        movfw   auxiliar_4
        movwf   auxiliar_2
        movlw   0x24                ; Comando de cambio de retardo del modo
                                    ; alternativo.
        goto    SALTA_COMANDO_NUMCARLIN

COMANDO_NUMCARLIN:
        movfw   CAR_LIN
        movwf   auxiliar_4          ; Copia el numero de caracteres por linea.
        movlw   0x22                ; Comando del cambio de caracteres linea.

SALTA_COMANDO_NUMCARLIN:
        call    ENVIA_PROTOCOLO     ; Envia el comando al LCD.
        goto    SALE_DE_PROGRAMACION_ELIGE_MENS

comprueba_otra_opcion_3:
                ; Comprueba si se trata de la opcion de cambiar el numero
                ; de caracteres por linea.

                ; TECLA_X contiene el valor correspondiente a la opcion
                ; elegida por el usuario.

                ; Si TECLA_X = 31 ='1' -> CAMBIAR CARACTERES POR LINEA.
        movlw   '1'
        subwf   TECLA_X,0           ; W = TECLA_X - W.
        btfss   STATUS,Z
        goto    SALE_DE_PROGRAMACION

        movfw   CAR_LIN
        movwf   auxiliar_4          ; Salvaguarda el valor actual del numero
                                    ; de caracteres por linea.
```

```
        bsf     PROGRAMACION_REG,6 ; Indica que se trata del comando de
        goto    prewaiting_MA      ; cambio de caracteres por linea.

EN_PROGRAMACION_VISUALIZACION:
        movlw   3                  ; Comprueba si el efecto visual es
        subwf   VISUALIZACION,0    ; SCROLL de un caracter.
        btfsc   STATUS,Z
        goto    V_SCROLL
        movlw   4                  ; Efecto visual ALTERNATIVO?
        subwf   VISUALIZACION,0
        btfsc   STATUS,Z
        goto    RETARDO_MALTERNATIVO
        goto    SALE_DE_PROGRAMACION_ELIGE_MENS

SALE_DE_PROGRAMACION_ESCRIBIR:
        movlw   2
        movwf   COMANDO            ; Muestra el comando escribir.
        goto    SALE_DE_PROGRAMACION

SALE_DE_PROGRAMACION_CL_R:
        movfw   auxiliar_4
        movwf   CAR_LIN            ; ESCAPE: CAR_LIN recupera su valor.
        movlw   1
        movwf   COMANDO            ; Muestra el comando caracteres/linea.
        goto    SALE_DE_PROGRAMACION

SALE_DE_PROGRAMACION_VIS:
        movlw   3
        movwf   COMANDO            ; Muestra el comando visualizacion.
        movfw   auxiliar_4
        movwf   VISUALIZACION      ; Tecla ESCAPE: VISUALIZACION
                                   ; recupera su valor.
SALE_DE_PROGRAMACION:
        call    ESCRIBE_COMANDO
        goto    salta_E_M

SALE_DE_PROGRAMACION_ELIGE_MENS:
        clrf    TECLA_X
        call    ELIGE_MENS

salta_E_M:
        bcf     PROGRAMACION_REG,3 ; Indica el fin de la programacion.
        bcf     PROGRAMACION_REG,1
        bcf     PROGRAMACION_REG,6
        goto    bucle2             ; Vuelve al bucle de programacion.

;************* INCLUDES ************************************
        INCLUDE "recibe.asm"
        INCLUDE "aprende.asm"
        INCLUDE "retardos.asm"
        INCLUDE "detecta.asm"
        INCLUDE "comandos.asm"
        INCLUDE "rs232.asm"
        INCLUDE "eeprom.asm"
        INCLUDE "graba.asm"

        END
```

A.1.2. recibe.asm

Listing A.2: Programa

```
;-----------------------------------------------------------------------
;
; Subrrutina que recibe la transmision. Recibe la senal procedente
; de RB0 y la convierte a una senal binaria. Para conseguirlo examina
; el tipo de codificacion de la senal despues de haber medido todos
; sus tiempos y obtiene bit a bit todos los valores.
;
;-----------------------------------------------------------------------

Rec_RC5_RE80:

        ; Inicializacion de la memoria que almacena la senal.

        bsf     STATUS,RP1              ; Banco 2.
        clrf    byte1
        clrf    byte2
        clrf    byte3
        clrf    byte4
        clrf    byte5
        clrf    byte6
        clrf    byte7
        clrf    byte8
        bcf     STATUS,RP1              ; Banco 0.

        ; Inicializacion de otras variables.

        clrf    nummedidascero
        clrf    MEDIDASCEROIGUALES
        clrf    nummedidasuno
        clrf    MEDIDASUNOIGUALES


        ; Comienza la toma de tiempos.
        ; El tiempo maximo de referencia para tomar medidas 2040us.
        ; Preescala del TMR0 = 8 (cada cuenta seran 8us).

        movlw   Direccion_ceros         ; Primera posicion de los ceros.
        movwf   FSR_TEMP1
        movlw   Direccion_unos          ; Primera posicion de los unos.
        movwf   FSR_TEMP2

otro_tiempo:
        bcf     INTCON,T0IF             ; Borra el T0IF.
        clrf    TMR0                    ; Borra el temporizador.
        call    PREESCALA_1             ; Valor de la preescala 8.

cambia_a_uno:
        btfsc   INTCON,T0IF             ; Comprueba si desbordo.
        goto    es_cabecera             ; Hay un nivel bajo muy largo ->
                                        ; cabecera.
        btfss   PORTB,0                 ; Comprueba la entrada.
        goto    cambia_a_uno            ; Repite mientras es cero.
        movfw   FSR_TEMP1
        movwf   FSR                     ; Inicializacion de FSR: apunta
                                        ; a la direccion de los ceros.
```

```
        movfw   TMR0                    ; Guarda el valor del TMR0.
        movwf   INDF
        incf    nummedidascero          ; Una medida cero mas.
        incf    FSR                     ; Siguiente direccion.
        movfw   FSR
        movwf   FSR_TEMP1               ; Vuelve a guardar el FSR.

        bcf     INTCON,T0IF             ; Borra el T0IF.
        clrf    TMR0                    ; Borra el temporizador.
        call    PREESCALA_1             ; Preescala 8.

cambia_a_cero:
        btfsc   INTCON,T0IF             ; Comprueba si desbordo.
        goto    Comprueba_cab_fin
        btfsc   PORTB,0                 ; Comprueba la entrada.
        goto    cambia_a_cero           ; Repite mientras es cero.
        movfw   FSR_TEMP2
        movwf   FSR                     ; Inicializacion de FSR: apunta
                                        ; a la direccion de los unos.
        movfw   TMR0
        movwf   INDF                    ; La direccion apuntada por FSR
                                        ; pasa a contener el valor
                                        ; temporal de un uno.
        incf    nummedidasuno           ; Un uno mas.
        incf    FSR                     ; Siguiente direccion.
        movfw   FSR
        movwf   FSR_TEMP2               ; Se vuelve a guardar el FSR.
        goto    otro_tiempo

Comprueba_cab_fin:
        movlw   1
        subwf   nummedidascero,0        ; numceros - 1.
        btfsc   STATUS,Z
        goto    es_cabecera2            ; Un solo cero, es cabecera.
        goto    fin_de_trama

es_cabecera:

       ; Espera a que pasen el '0' y el '1' de la cabecera.

cab_pasa_a_uno:
        btfss   PORTB,0                 ; Comprueba la entrada.
        goto    cab_pasa_a_uno

        bcf     INTCON,T0IF             ; Borra el T0IF.
        clrf    TMR0                    ; Borra el temporizador.
        call    PREESCALA_2             ; Preescala 64 para comprobar
                                        ; si la senal se acabo o no.
cab_sigue_a_uno:
        btfsc   INTCON,T0IF
        goto    fin_de_trama            ; Si desborda se acabo la trama.
        btfsc   PORTB,0                 ; Se comprueba la entrada hasta
                                        ; que cambia a cero.
        goto    cab_sigue_a_uno
        goto    otro_tiempo             ; Despues de evitar la cabecera
                                        ; toma medidas de los tiempos.
es_cabecera2:
        decf    nummedidascero
        decf    FSR_TEMP1               ; Vuelve a apuntar a la primera
```

```
                                        ; posicion de los ceros.
        bcf     INTCON,T0IF             ; Borra el T0IF.
        clrf    TMR0                    ; Borra el temporizador.

        call    PREESCALA_2

cab_pasa_a_cero:
        btfsc   INTCON,T0IF
        goto    fin_de_trama            ; Si desborda se acabo la trama.
        btfsc   PORTB,0                 ; Comprueba la entrada.
        goto    cab_pasa_a_cero
        goto    otro_tiempo

fin_de_trama:

        ; Comienza el tratamiento de las medidas.

        movfw   nummedidascero          ; Almacena el valor del numero de
                                        ; medidas uno y cero.
        movwf   temp_medcero
        movfw   nummedidasuno
        movwf   temp_meduno

        ; Se obtiene el minimo de los ceros y minimo de los unos.

        movfw   temp_medcero
        movwf   nummedidascero          ; Numero de medidas cero.
        movlw   Direccion_ceros
        movwf   FSR                     ; Se examinan los ceros.
                                        ; Se busca el "0" mas pequeno.
        movlw   D'1'
        subwf   nummedidascero,0        ; nummedidascero - 1 = W.
        btfss   STATUS,C
        goto    NOHAYCEROS2             ; No hay medidas cero cortas.
        movfw   INDF
        movwf   MEDIDACEROPEQUE         ; Se parte con la primera medida
                                        ; cero como la medida mas pequena
                                        ; para ir comparandola con las demas
                                        ; sucesivamente.
        decfsz  nummedidascero          ; Una medida cero menos que comprobar.
        goto    sigue8
        goto    NOHAYCEROS2             ; Solo habia un cero.
sigue8:
        incf    FSR                     ; Apunta a la siguiente direccion.
        movfw   INDF                    ; Carga otra nueva medida cero para
        movwf   MEDIDACERO              ; comparar.
        subwf   MEDIDACEROPEQUE,0
        btfsc   STATUS,C
        goto    intercambia8            ; La nueva medida es mas pequena.
        goto    nointercambia8          ; La nueva medida no es mas pequena.

intercambia8:
        movfw   MEDIDACERO              ; La nueva medida pasa a ser la mas
        movwf   MEDIDACEROPEQUE         ; pequena.

nointercambia8:
        decfsz  nummedidascero
        goto    sigue8
```

```
        ; Busca el "1" mas pequeno.

buscaunopeque:
        movfw   temp_meduno
        movwf   nummedidasuno
        movlw   Direccion_unos          ; Apunta a la primera direccion
        movwf   FSR                     ; de unos.
        movlw   D'1'
        subwf   nummedidasuno,0         ; nummedidasuno - 1 = w.
        btfss   STATUS,C
        goto    NOHAYUNOS2
        movfw   INDF
        movwf   MEDIDAUNOPEQUE          ; El primer uno es la
        decfsz  nummedidasuno           ; referencia de medida mas
        goto    sigue0                  ; pequena de la que se parte.
        goto    NOHAYUNOS2              ; Solo habia un uno.

sigue0:
        incf    FSR                     ; Apunta a la siguiente
        movfw   INDF                    ; direccion.
        movwf   MEDIDAUNO
        subwf   MEDIDAUNOPEQUE,0        ; Compara con el siguiente uno.
        btfsc   STATUS,C
        goto    intercambia6            ; Si MEDIDAUNO es mas pequena
        goto    nointercambia6          ; es la nueva medida uno.

intercambia6:
        movfw   MEDIDAUNO
        movwf   MEDIDAUNOPEQUE          ; La nueva medida uno pasa a ser
                                        ; la medida mas pequena.
nointercambia6:
        decfsz  nummedidasuno           ; Una medida uno menos que
        goto    sigue0                  ; examinar.

       ; Calcula las tolerancias que marcan el limite para
       ; diferenciar entre una medida larga y una corta,
       ; son los errores admisibles.

       ; Se emplea como tolerancia la mitad del valor minimo
       ; obtenido antes.

        movfw   MEDIDAUNOPEQUE
        movwf   medidauno_ref
        bcf     STATUS,C                ; Borra el carry.
        rrf     medidauno_ref,f         ; Divide por 2.
        movfw   MEDIDACEROPEQUE
        movwf   medidacero_ref
        bcf     STATUS,C                ; Borra el carry.
        rrf     medidacero_ref,f        ; Divide por 2.
        movfw   temp_medcero
        movwf   nummedidascero
        movfw   temp_meduno
        movwf   nummedidasuno
        movlw   Direccion_ceros
        movwf   FSR

        movlw   D'1'
```

```
233         subwf   nummedidascero,0          ; nummedidascero - 1 = w.
            btfss   STATUS,C
            goto    NOHAYCEROS                ; No hay medidas cero.

            ; Comprueba si es una senal pulse coded o space coded.
238         ; Detecta si todos los ceros o los unos de la senal son iguales
            ; exceptuando por supuesto la cabecera.

    compruebaceros:
            movfw   INDF
243         incf    MEDIDASCEROIGUALES        ; Se inicializa a uno porque si
                                              ; hay otro cero igual son ya dos
                                              ; iguales.
            movwf   MEDIDACERO
            decfsz  nummedidascero
248         goto    sigue
            goto    compruebaunos             ; No hay mas ceros.

    sigue:
            incf    FSR                       ; Apunta a la siguiente direccion.
253         movfw   INDF
            subwf   MEDIDACERO,0              ; medida2 - medida1 = w.
            btfsc   STATUS,C                  ; Si el resultado es negativo se
            goto    continua                  ; complementa.
            sublw   D'0'
258
    continua:
            subwf   medidacero_ref,0          ; Compara con el valor de tolerancia.
            btfsc   STATUS,C                  ; Si C=1 son iguales un valor cero
            incf    MEDIDASCEROIGUALES        ; igual mas.
263         movfw   INDF
            movwf   MEDIDACERO
            decfsz  nummedidascero            ; Si no hay mas ceros pasa a tratar
                                              ; los unos.
            goto    sigue
268 compruebaunos:
            movlw   Direccion_unos
            movwf   FSR
            movlw   D'1'
            subwf   nummedidasuno,0           ; nummedidasuno - 1 = w.
273         btfss   STATUS,C
            goto    NOHAYUNOS                 ; No hay unos.

    compruebaunos2:
            movfw   INDF
278         incf    MEDIDASUNOIGUALES         ; Inicializa a uno porque si hay
                                              ; otro uno, son ya dos iguales.
            movwf   MEDIDAUNO
            decfsz  nummedidasuno
            goto    sigue2
283         goto    que_codigo_es             ; No hay mas ceros.
    sigue2:
            incf    FSR                       ; Apunta a la siguiente direccion.
            movfw   INDF
            subwf   MEDIDAUNO,0               ; medida2 - medida1 = w
288         btfsc   STATUS,C                  ; Si el resultado es negativo se
            goto    continua2                  ; complementa.
            sublw   D'0'
    continua2:
```

```
        subwf   medidauno_ref,0         ; Compara con la tolerancia.
        btfsc   STATUS,C                ; Si C=1 son iguales.
        incf    MEDIDASUNOIGUALES       ; Un valor cero igual mas.
        movfw   INDF
        movwf   MEDIDAUNO
        decfsz  nummedidasuno
        goto    sigue2

NOHAYUNOS:
NOHAYCEROS:

que_codigo_es:

        ; Comprueba si es space coded o pulse coded.

        movfw   temp_medcero            ; w = nummedidascero.
        subwf   MEDIDASCEROIGUALES,0    ; Averigua si todos los ceros
        btfsc   STATUS,Z                ; son iguales.
        goto    spacecoded              ; Senal SPACE-CODED.

        movfw   temp_meduno             ; w = nummedidasuno.
        subwf   MEDIDASUNOIGUALES,0     ; Averigua si todos los unos
                                        ; son iguales.
        btfsc   STATUS,Z
        goto    pulsecoded              ; Senal PULSE-CODED.

        ; Senal bifase porque no coinciden los ceros y los unos.
BIFASE:

tratamientodatos:

        movfw   MEDIDAUNOPEQUE
        movwf   medidauno_ref
        bcf     STATUS,C                ; Borra el carry.
        rrf     medidauno_ref,f         ; Divide por 2.
        movfw   MEDIDACEROPEQUE
        movwf   medidacero_ref
        bcf     STATUS,C                ; Borra el carry.
        rrf     medidacero_ref,f        ; Divide por 2.
        movfw   temp_medcero
        movwf   nummedidascero          ; Numero de medidas cero.

sihay:
        movfw   temp_meduno
        movwf   nummedidasuno           ; Numero de medidas uno.

sihay2:
        movlw   Direccion_ceros
        movwf   FSR_TEMP1
        movwf   FSR                     ; Direccion del primer cero.
        movfw   INDF
        movwf   MEDIDACERO              ; Carga la medida cero(2).
        decf    nummedidascero
        movlw   Direccion_unos
        movwf   FSR_TEMP2
        movwf   FSR
        movfw   INDF
        movwf   MEDIDAUNO               ; Carga la medida uno(2).
```

```
        decf    nummedidasuno

        movfw   MEDIDACEROPEQUE
        subwf   MEDIDACERO,0            ; medidacero - medidaceropeque = w.
        movwf   MEDIDACERO              ; Lo que queda de cero se guarda.

        ; Un cero se considera largo cuando sea superior que un cero
        ; minimo mas la tolerancia.

        subwf   medidacero_ref,0
        btfsc   STATUS,C                ; Si es negativo entonces es cerouno.
        goto    antesdeUNOCERO
        goto    CEROUNO

antesdeUNOCERO:                         ; Toma la medida cero siguiente.

        movlw   D'1'
        subwf   nummedidascero,0        ; Comprueba si hay mas medidas cero.
        btfss   STATUS,Z
        goto    simasceros              ; Hay mas de un cero.

        call    ROTA_BYTE1              ; Es un bit uno porque hay una
        goto    FINAL                   ; transicion de cero a uno.

simasceros:
        decf    nummedidascero
        incf    FSR_TEMP1
        movfw   FSR_TEMP1
        movwf   FSR
        movfw   INDF
        movwf   MEDIDACERO              ; Toma la medida cero siguiente.

UNOCERO:

        call    ROTA_BYTE1              ; Bit uno.

        movfw   MEDIDACEROPEQUE         ; Medida cero mas pequena.
        subwf   MEDIDACERO,0            ; MEDIDACERO - MEDIDACEROPEQUE.
        movwf   MEDIDACERO              ; Lo que queda de uno se guarda.

        ; Una medida uno se considera un uno largo cuando es mayor que el
        ; uno mas pequeno mas la tolerancia.

        subwf   medidacero_ref,0
        btfsc   STATUS,C                ; Si es negativo entonces es unocero.
        goto    preUNOCERO              ; No queda uno.
        movlw   D'1'
        subwf   nummedidasuno,0         ; Comprueba si hay mas medidas uno.
        btfss   STATUS,Z
        goto    sihayuno                ; Si hay mas medidas uno.
        call    ROTA_BYTE0              ; Es un bit cero.
        incf    FSR_TEMP2               ; Apunta a la siguiente medida uno.
        movfw   FSR_TEMP2
        movwf   FSR
        movfw   INDF
        movwf   MEDIDAUNO               ; Carga la nueva medida uno.
        movfw   MEDIDAUNOPEQUE
        subwf   MEDIDAUNO,0             ; MEDIDAUNO - MEDIDAUNOPEQUE = w.
        movwf   MEDIDAUNO               ; Guarda lo que queda de uno.
```

```
        subwf   medidauno_ref,0         ; medidauno_ref - w.
        btfsc   STATUS,C
        goto    otrobitcero             ; Es un bit cero porque queda
                                        ; mas medida uno.
        call    ROTA_BYTE1              ; Es un bit uno porque no queda
                                        ; mas medida uno.

        incf    FSR_TEMP1               ; Siguiente direccion.
        movfw   FSR_TEMP1
        movwf   FSR
        movfw   INDF
        movwf   MEDIDACERO
        movfw   MEDIDACEROPEQUE
        subwf   MEDIDACERO,0            ; MEDIDACERO-MEDIDACEROPEQUE=w.
        movwf   MEDIDACERO              ; Guarda lo que queda de cero.
        subwf   medidacero_ref,0        ; Lo que queda de cero es
        btfsc   STATUS,C                ; mas grande que la tolerancia?
        goto    final3
        call    ROTA_BYTE0              ; Es un bit cero.

final3:
        goto    FINAL

otrobitcero:
        call    ROTA_BYTE0              ; Bit cero.
        goto    FINAL

sihayuno:
        decf    nummedidasuno           ; Aun quedan medidas uno.
        incf    FSR_TEMP2               ; Siguiente direccion.
        movfw   FSR_TEMP2
        movwf   FSR
        movfw   INDF
        movwf   MEDIDAUNO               ; Carga la nueva medida uno.
        goto    CEROUNO

preUNOCERO:
        movlw   D'1'
        subwf   nummedidasuno,0         ; Comprueba si aun quedan unos.
        btfss   STATUS,Z
        goto    masunos                 ; Si quedan medidas uno.
        call    ROTA_BYTE1              ; Es un bit uno.
        incf    FSR_TEMP1               ; Siguiente direccion.
        movfw   FSR_TEMP1
        movwf   FSR
        movfw   INDF
        movwf   MEDIDACERO              ; Carga la siguiente medida cero.
        movfw   MEDIDACEROPEQUE
        subwf   MEDIDACERO,0            ; MEDIDACERO-MEDIDACEROPEQUE=w.
        movwf   MEDIDACERO              ; Guarda el resto de medida cero.
        subwf   medidacero_ref,0        ; Lo que queda de cero es
        btfsc   STATUS,C                ; mayor que la tolerancia?
        goto    FIN
        call    ROTA_BYTE0              ; Es un bit cero.


FIN:
        goto    FINAL
```

```
masunos:
        decf    nummedidasuno           ; Un uno menos que analizar.
        decf    nummedidascero          ; Un cero menos que analizar.
        incf    FSR_TEMP1               ; Siguiente direccion de los ceros.
        incf    FSR_TEMP2               ; Siguiente direccion de los unos.
        movfw   FSR_TEMP1
        movwf   FSR
        movfw   INDF
        movwf   MEDIDACERO              ; Carga un cero.
        movfw   FSR_TEMP2
        movwf   FSR
        movfw   INDF
        movwf   MEDIDAUNO               ; Carga un uno.
        goto    UNOCERO

CEROUNO:
        call    ROTA_BYTE0              ; Es un bit cero.
        movfw   MEDIDAUNOPEQUE
        subwf   MEDIDAUNO,0             ; MEDIDAUNO-MEDIDAUNOPEQUE=w.
        movwf   MEDIDAUNO               ; Guarda lo que queda de medida uno.
        subwf   medidauno_ref,0         ; Lo que queda de medida uno es
        btfsc   STATUS,C                ; mayor que la tolerancia?
        goto    preCEROUNO
        movlw   D'1'
        subwf   nummedidascero,0        ; Comprueba si hay mas medidas cero.
        btfss   STATUS,Z
        goto    sihayuno2               ; Hay mas unos porque queda mas
                                        ; de una medida cero.
        call    ROTA_BYTE1              ; Es un bit uno. Solo hay una
        incf    FSR_TEMP1               ; medida cero mas.
        movfw   FSR_TEMP1
        movwf   FSR
        movfw   INDF
        movwf   MEDIDACERO              ; Se carga la ultima medida cero.
        movfw   MEDIDACEROPEQUE
        subwf   MEDIDACERO,0            ; MEDIDACERO-MEDIDACEROPEQUE=w.
        movwf   MEDIDACERO              ; Guarda lo que queda de cero.
        subwf   medidacero_ref,0        ; Comprueba si lo que queda de
        btfsc   STATUS,C                ; cero es mayor que la tolerancia.
        goto    otrobituno              ; Es un bit 1 porque queda medida 0.
        call    ROTA_BYTE0              ; Es un bit cero, el ultimo bit.
        goto    FINAL

otrobituno:
        call    ROTA_BYTE1              ; Es un bit uno.
        goto    FINAL

sihayuno2:
        decf    nummedidascero          ; Una medida cero menos.
        incf    FSR_TEMP1               ; Siguiente direccion.
        movfw   FSR_TEMP1
        movwf   FSR
        movfw   INDF
        movwf   MEDIDACERO              ; Carga la siguiente medida cero.
        goto    UNOCERO

preCEROUNO:
        movlw   D'1'
        subwf   nummedidascero,0
```

```
528         btfss   STATUS,Z                  ; Solo queda un cero?
            goto    haymasunos                ; Como hay mas de un cero, hay
                                              ; mas unos antes.
            call    ROTA_BYTE0                ; Es un bit cero.
            goto    FINAL
533
    haymasunos:
            decf    nummedidasuno             ; Una medida uno menos.
            decf    nummedidascero            ; Una medida cero menos.
            incf    FSR_TEMP1                 ; Siguiente direccion de ceros.
538         incf    FSR_TEMP2                 ; Siguiente direccion de unos.
            movfw   FSR_TEMP1
            movwf   FSR
            movfw   INDF
            movwf   MEDIDACERO                ; Carga la siguiente medida cero.
543         movfw   FSR_TEMP2
            movwf   FSR
            movfw   INDF
            movwf   MEDIDAUNO                 ; Carga la siguiente medida uno.
            goto    CEROUNO
548

    spacecoded:
           ; Tratamiento de space coded.
           ; Solo se examinan los pulsos y no se tienen en cuenta los
553        ; valores cero que ya se han examinado: analisis de temp2


    SPACE_A_BITS:

558        ; Convierte la senal unos y ceros.

            movfw   temp_meduno
            movwf   nummedidasuno
            movlw   Direccion_unos
563         movwf   FSR                        ; Apunta al primero de los unos.
    comparaunos:
            movfw   INDF
            movwf   MEDIDAUNO
            movfw   MEDIDAUNOPEQUE
568         subwf   MEDIDAUNO,0                ; Resta al uno la medida
            subwf   medidauno_ref,0            ; uno mas pequena.
            btfsc   STATUS,C                   ; Resta la tolerancia para
            goto    esbitceroS                 ; averiguar si hay medida uno.
    esbitunoS:                                 ; Si queda medida uno->Bit uno.
573         call    ROTA_BYTE1                 ; Es un bit uno.
            goto    continua5

    esbitceroS:
            call    ROTA_BYTE0                 ; Es un bit cero.
578
    continua5:
            decfsz  nummedidasuno              ; Una medida uno menos.
            goto    sigue5
            goto    FINAL
583
    sigue5:
            incf    FSR
            goto    comparaunos
```

```
pulsecoded:
        ; Tratamiento de la senal pulse coded.
        ; Solo se examinan los ceros y no se tienen en cuenta los valores
        ; uno que ya se han examinado: analisis de temp1.

PULSE_A_BITS:
        ; Transforma el pulsecoded a unos y ceros.

        movfw   temp_medcero
        movwf   nummedidascero
        decf    nummedidascero

        movlw   Direccion_ceros
        movwf   FSR                         ; Direccion del primer cero.

comparaceros:
        movfw   INDF
        movwf   MEDIDACERO
        movfw   MEDIDACEROPEQUE             ; Resta a la medida cero la medida
        subwf   MEDIDACERO,0                ; cero mas pequena.
        subwf   medidacero_ref,0            ; Resta la tolerancia
        btfsc   STATUS,C                    ; y si queda cero aun es un bit 1.
        goto    esbitceroP

esbitunoP:
        call    ROTA_BYTE1                  ; Bit uno.
        goto    continua6

esbitceroP:
        call    ROTA_BYTE0                  ; Bit cero.

continua6:
        decfsz  nummedidascero              ; Una medida cero menos que tratar.
        goto    sigue6
        goto    FINAL

sigue6:
        incf    FSR
        goto    comparaceros

NOHAYUNOS2:
NOHAYCEROS2:

FINAL:

        return


;-----------------------------------------------------------------
; Rota los bits de cada byte de senal. Pone un bit 1.
;-----------------------------------------------------------------

ROTA_BYTE1:
        bsf     STATUS,RP1                  ; Banco 2.
        call    ROTA_IZQUIERDA              ; Rota los bits a la izquierda.
        bsf     byte8,0                     ; Bit a 1.
```

```
        bcf    STATUS,RP1                  ; Banco 0.
        return
;-----------------------------------------------------------------
; Rota los bits de cada byte de senal. Pone un bit 0.
;-----------------------------------------------------------------

ROTA_BYTE0:
        bsf    STATUS,RP1                  ; Banco 2.
        call   ROTA_IZQUIERDA              ; Rota los bits a la izquierda.
        bcf    byte8,0                     ; Bit a cero.
        bcf    STATUS,RP1                  ; Banco 0.
        return

;----------------------------------------------------------------
; Rota los bits de cada byte de senal.
;----------------------------------------------------------------

ROTA_IZQUIERDA:
        rlf    byte8,f                     ; Rota los bits a la izquierda.
        rlf    byte7,f
        rlf    byte6,f
        rlf    byte5,f
        rlf    byte4,f
        rlf    byte3,f
        rlf    byte2,f
        rlf    byte1,f
        return


;-----------------------------------------------------------------
; Preescala 1:8
;-----------------------------------------------------------------

PREESCALA_1:
        bsf    STATUS,RP0                  ; Banco 1.
        bcf    OPTION_REG,PS0
        bsf    OPTION_REG,PS1
        bcf    OPTION_REG,PS2
        bcf    STATUS,RP0                  ; Banco 0.
        return

;-----------------------------------------------------------------
; Preescala 1:64
;-----------------------------------------------------------------



PREESCALA_2:
        bsf    STATUS,RP0                  ; Banco 1.
        bsf    OPTION_REG,PS0
        bcf    OPTION_REG,PS1
        bsf    OPTION_REG,PS2
        bcf    STATUS,RP0                  ; Banco 0.
        return
```

A.1.3. aprende.asm

Listing A.3: Programa

```
;-----------------------------------------------------------------
; Recibe y memoriza los valores de las 12 teclas.
;-----------------------------------------------------------------
MODO_APRENDIZAJE:

        ; TECLA NUMERO "0"

        call    Rec_RC5_RE80            ; Recibe datos de tecla "1".
        call    DETECTA_INTERFERENCIAS
        bcf     STATUS,RP1              ; Banco 0.
        iorlw   0                       ; Comprueba si se recibio una
        btfsc   STATUS,Z                ; senal de ceros.
        return

        movlw   DIR_TECLA1_0            ; Direccion del primer byte de
        movwf   FSR                     ; la primera  tecla.
        movwf   auxiliar_2
        movlw   DIR_TECLA1              ; Direccion del segundo byte de
        movwf   auxiliar_10             ; la primera tecla.

        call    GRABA_BYTE              ; Graba los bytes de la senal
                                        ; recibida.
tecla0_senal:
        btfss   PORTA,0                 ; Modo aprendizaje?
        return                          ; Sale, no es modo aprendizaje.

        btfsc   PORTB,0                 ; Espera que llegue otra senal.
        goto    tecla0_senal
        call    Rec_RC5_RE80            ; Recibe datos de tecla "2".
        call    DETECTA_TECLA           ; Comprueba que no es una
                                        ; tecla ya memorizada.
        iorlw   0

        btfss   STATUS,Z
        return

        btfss   PORTA,0
        return                          ; Sale si ya no esta en modo
                                        ; aprendizaje.
        call    APRENDIENDO             ; Sigue memorizando teclas.
        sublw   250
        btfsc   STATUS,Z                ; Comprueba si no esta en modo
        goto    FINALIZA_Apren          ; aprendizaje.

        call    ENVIA_MSJ_APREN_FIN_2
        bsf     APRENDIZAJE_REG,5       ; Indica que memorizo teclas.

FINALIZA_Apren:
        btfsc   PORTA,0                 ; Espera que se pulse el boton
                                        ; de modo de funcionamiento.
        goto    FINALIZA_Apren

        return
```

```
;-----------------------------------------------------------------
; APRENDE LAS TECLAS
;-----------------------------------------------------------------

APRENDIENDO:
        movlw   0
        movwf   auxiliar_9
        movlw   '1'
        movwf   auxiliar                ; Numero de tecla que se va
                                        ; a memorizar.
PRE_T_1:
        call    ENVIA_MSJ_TECLA_2       ; Solicita a traves del LCD
T_1:                                    ; que se pulse tecla numero 1.

tecla2_senal:
        btfss   PORTA,0
        retlw   250                     ; Sale si ya no esta en modo
                                        ; aprendizaje.
        btfsc   PORTB,0                 ; Espera que llegue la senal.
        goto    tecla2_senal

        call    Rec_RC5_RE80            ; Recibe datos de tecla "1".

        ; Almacena el valor de la se al recibida

        call    DETECTA_INTERFERENCIAS  ; Senal de ceros?
        bcf     STATUS,RP1              ; Banco 0.
        iorlw   0
        btfsc   STATUS,Z
        goto    T_1                     ; Hay senal de interferencia.
        call    DETECTA_TECLA           ; Detecta si es una tecla
        subwf   auxiliar_9,0            ; memorizada.
        btfsc   STATUS,C
        goto    T_1
        call    GRABA_BYTE              ; Guarda la senal de la tecla.

misma_tecla_senal:
        btfss   PORTA,0
        retlw   250                     ; Modo aprendizaje?
        btfsc   PORTB,0                 ; Espera que llegue la senal.
        goto    misma_tecla_senal
        call    Rec_RC5_RE80            ; Recibe la senal de nuevo
                                        ; para comprobar si otra senal
                                        ; coincide lo que muestra que
                                        ; no ocurrieron interferencias.
        call    DETECTA_TECLA
        addlw   48
        subwf   auxiliar,0              ; Compara con num. tecla actual
                                        ; ->tiene que ser el mismo.
        btfss   STATUS,Z
        goto    antes_de_T_1
        goto    evita_antes_de_T_1

antes_de_T_1:
        call    DISMINUYE               ; Apuntar a las direcciones
        goto    T_1                     ; de tecla actual.

evita_antes_de_T_1:
        incf    auxiliar_9
```

```
        incf    auxiliar
        movlw   9
        subwf   auxiliar_9,0            ; ENTER?
        btfsc   STATUS,Z
        goto    ENTER_T
        movlw   10
        subwf   auxiliar_9,0            ; ESCAPE?
        btfsc   STATUS,Z
        goto    ESC_T
        movlw   11
        subwf   auxiliar_9,0            ; Memorizo todas las
        btfsc   STATUS,Z                ; teclas?
        goto    FIN_FIN_FIN
        goto    PRE_T_1

ENTER_T:
        call    ENVIA_MSJ_TECLA_ENTER_2 ; TECLA ENTER?
        goto    T_1

ESC_T:
        call    ENVIA_MSJ_TECLA_ESCAPE_2; TECLA ESCAPE?
        goto    T_1

FIN_FIN_FIN:
        return

;-----------------------------------------------------------
; Disminuye direcciones
;-----------------------------------------------------------

DISMINUYE:
       decf    auxiliar_2
       movlw   3
       subwf   auxiliar_10,1
       return

;-----------------------------------------------------------
; Detecta si se han producido interferencias
;-----------------------------------------------------------

       ; Detecta si los bytes de la senal recibida son ceros.

DETECTA_INTERFERENCIAS:
       bsf     STATUS,RP1
       movlw   0
       subwf   byte8,0
       btfss   STATUS,Z
       retlw   1
       movlw   0
       subwf   byte7,0
       btfss   STATUS,Z
       retlw   1
       movlw   0
       subwf   byte6,0
       btfsc   STATUS,Z
       retlw   0
       retlw   1
```

A.1.4. rs232.asm

Listing A.4: Programa

```
; ----------------------------------------------------------
; Inicializa los puertos para la transmision RS-232.
; ----------------------------------------------------------


INI_RS232:
        bsf     STATUS,RP0          ; Banco 1.

; ----------------------------------------------------------
; Establece la velocidad de comunicacion con el PC.
; ----------------------------------------------------------
; Baudios = 9600, Sin paridad, 1 Bit de Stop.

        movlw   0x19                ; 0x19=9600 bps.
        movwf   SPBRG
        movlw   b'00100100'         ; brgh = Alto (2).
        movwf   TXSTA               ; Habilita la transmision asincrona.
        bcf     STATUS,RP0          ; Banco 0.
        movlw   b'10010000'         ; Habilita la recepcion asincrona.
        movwf   RCSTA

; ---------------------------------------------------------
; Establece un tiempo para el arranque.
; ---------------------------------------------------------

        clrf    auxiliar_6
cuent:  decfsz  auxiliar_6,F
        goto    cuent
        movf    RCREG,W
        movf    RCREG,W
        movf    RCREG,W             ; Descarga el buffer de recepcion.
        return


; ----------------------------------------------------------
; Envia byte a traves del puerto serie RS-232.
; ----------------------------------------------------------

send:   movwf   TXREG               ; Envia el dato de W.

        bsf     STATUS,RP0          ; Banco 1.
AQUII:  btfss   TXSTA,TRMT          ; (1) La transmision se completa si
                                    ; esta a '1'.
        goto    AQUII

        bcf     STATUS,RP0          ; Banco 0.
        return
```

A.1.5. detecta.asm

Listing A.5: Programa

```
;-----------------------------------------------------------------
; Detecta el numero de tecla pulsado.
; W recibe el numero de tecla.
;-----------------------------------------------------------------

DETECTA_TECLA:

            clrf    auxiliar_7          ; Guarda el numero de tecla que
                                        ; se esta examinando.
NO_ES:
            movlw   12                  ; Numero de teclas total.
            subwf   auxiliar_7,0        ; Comprueba si recorrio todas las
            btfsc   STATUS,Z            ; teclas.
            goto    sale_YA             ; Termino.

            movlw   DIR_TECLA1
            movwf   FSR_TEMP1
            movfw   auxiliar_7
            movwf   auxiliar_3
            incf    auxiliar_7          ; Siguiente tecla.


            ; Solo se comparan los 4 bytes de menor peso.
            movlw   3
sumando:    addwf   FSR_TEMP1,1         ; Accede al byte6 de la tecla actual.
            decfsz  auxiliar_3          ; Guarda la direccion del byte6.
            goto    sumando

            bsf     STATUS,RP1          ; Banco 2.
            movfw   byte5
            bcf     STATUS,RP1          ; Banco 0.
            movwf   cuenta              ; Copia el byte5.
            movlw   DIR_TECLA1_0
            movwf   FSR
            movfw   auxiliar_7
            addwf   FSR,1
            decf    FSR
            movfw   INDF
            subwf   cuenta,0            ; Compara los byte5 de la senal tomada
            btfss   STATUS,Z            ; y la almacenada.
            goto    NO_ES               ; No coinciden.

            call    COMPARA_BYTESS      ; Compara los bytes 6, 7 y 8.

            movwf   auxiliar_3          ; Comprueba si coincidieron los 3 bytes.
            btfss   auxiliar_3,0
            goto    NO_ES               ; No coincidieron.
            decf    auxiliar_7
sale_YA:
            movfw   auxiliar_7          ; Numero de tecla detectada si la hubo.
                                        ; Si no detecto, auxiliar_7 = 12.
            return


;-----------------------------------------------------------------
```

```
; Compara los bytes recibidos con los almacenados.
;-----------------------------------------------------------------

COMPARA_BYTESS:
            movlw   3
            movwf   cuenta
            clrf    Retardo_ms_Contador
            movlw   Dir_Senal          ; Direccion del byte1.
            addlw   5                  ; Accede al byte6 de la senal
            movwf   FSR                ; recibida.
            movwf   FSR_TEMP2

avanza:
            bsf     STATUS,IRP
            movfw   FSR_TEMP2
            movwf   FSR
            movfw   INDF
            movwf   auxiliar_3         ; byte de la senal recibida.
            incf    FSR                ; Siguiente byte.
            movfw   FSR
            movwf   FSR_TEMP2          ; Salvaguarda la direccion del
            movfw   FSR_TEMP1          ; byte de la senal recibida.
            movwf   FSR
            movfw   INDF
            movwf   Retardo_ms_Contador  ; byte de tecla comprobada.
            incf    FSR                ; Apunta al byte de la tecla.
            movfw   FSR
            movwf   FSR_TEMP1          ; Salvaguarda la direccion.
            movfw   auxiliar_3
            bcf     STATUS,IRP
            subwf   Retardo_ms_Contador,0; Compara los bytes para ver si
            btfss   STATUS,Z           ;  son iguales.
            retlw   0                  ; No coinciden los bytes: no es la
            decfsz  cuenta             ; tecla.
            goto    avanza             ; Prueba con los siguientes bytes.
            bcf     STATUS,IRP


            retlw   1
```

A.1.6. graba.asm

Listing A.6: Programa

```
;-----------------------------------------------------------
; Graba las senales recibidas en el modo aprendizaje.
;-----------------------------------------------------------

GRABA_BYTE:
        movfw   auxiliar_2
        movwf   FSR
        bsf     STATUS,RP1         ; Banco 2.
        movfw   byte5              ; Guarda byte5.
        movwf   INDF               ; TECLAx_0.
        bcf     STATUS,RP1         ; Banco 0.
        incf    FSR
```

```
        movfw   FSR
        movwf   auxiliar_2          ; Guarda la direccion actual de TECLAx_0.
        movfw   auxiliar_10
        movwf   FSR                 ; Apunta a la direccion que almacena el
                                    ; byte6 (en el Banco2 -> IRP a '1').
        bsf     STATUS,IRP
        bsf     STATUS,RP1
        movfw   byte6               ; Guarda byte6.
        movwf   INDF
        incf    FSR                 ; Apunta a la siguiente direccion.
        movfw   byte7
        movwf   INDF                ; Guarda byte7.
        incf    FSR
        movfw   byte8               ; Guarda byte8.
        movwf   INDF
        incf    FSR
        bcf     STATUS,RP1
        bcf     STATUS,IRP
        movfw   FSR
        movwf   auxiliar_10         ; Salvaguarda FSR.
        return
```

A.1.7. eeprom.asm

Listing A.7: Programa

```

;-----------------------------------------------------------------
; Lee un byte de la EEPROM.
;-----------------------------------------------------------------

LEE_EEPROM:
          bsf    STATUS,RP0       ; Banco 1.
                                  ; En w esta la direccion de la EEPROM.
          movwf  EEADR            ; Se indica la direccion de lectura.
          bsf    EECON1,RD        ; Lectura de la EEPROM.
          movfw  EEDATA           ; W = DATO.
          bcf    EECON1,RD
          bcf    STATUS,RP0       ; Banco 0.

          return


;-----------------------------------------------------------------
; Escribe un byte en la EEPROM.
;-----------------------------------------------------------------

ESCRIBE_EEPROM:
         bsf    STATUS,RP0        ; Banco 1.
                                  ; En w esta la direccion de escritura.
         movwf  EEADR
         bcf    STATUS,RP0        ; Banco 0.
         movfw  auxiliar_2        ; auxiliar contiene el byte que se almacena.
         bsf    STATUS,RP0        ; Banco 1.
         movwf  EEDATA
         bsf    EECON1, WREN      ; Habilita escritura.
```

```
        movlw  0x55
        movwf  EECON2
        movlw  0xAA
        movwf  EECON2
        bsf    EECON1,WR
EE_ESPERA:

        btfsc  EECON1,WR
        goto   EE_ESPERA
        bcf    EECON1,EEIF

        bcf    STATUS,RP0

        return
```

A.1.8. comandos.asm

Listing A.8: Programa

```
;-----------------------------------------------------------------
; Escribe en el LCD el comando actual.
; Dependiendo del valor de la variable COMANDO, que varia con las
; pulsaciones del mando a distancia se manda el mensaje al LCD
; que corresponda informando sobre cual es el comando actual.
;-----------------------------------------------------------------

ESCRIBE_COMANDO:

        movlw  1                ; Comando de cambio del numero de
        subwf  COMANDO,0        ; caracteres por linea.
        btfsc  STATUS,Z
        call   MSJ_COMANDO_2    ; Envia el mensaje.

        movlw  2                ; Comando de escritura en el LCD.
        subwf  COMANDO,0
        btfsc  STATUS,Z
        call   MSJ_COMANDO_3    ; Envia el mensaje.

        movlw  3                ; Comando de modo de visualizacion.
        subwf  COMANDO,0
        btfsc  STATUS,Z
        call   MSJ_COMANDO_5    ; Envia el mensaje.

        movlw  4                ; Modo de funcionamiento normal del LCD.
        subwf  COMANDO,0
        btfss  STATUS,Z
        goto   RETORNA

        call   ELIGE_MENS       ; Muestra el mensaje definido
                                ; si lo hay. Si no se creo mensaje
RETORNA:                        ; indica que es necesario programar.
        call   RETARDO_ENTRE_TECLAS

        return

;-------------------------------------------------------------
```

```
; Toma un mensaje de la tabla dependiendo del valor de W.
;-------------------------------------------------------

TOMA_MENSAJES:
        movwf   cuenta          ; Carga el valor que se anade al PCL para
                                ; acceder al mensaje dentro de la tabla.
_Toma_Caracter:
        movfw   cuenta
        call    MENSAJE_ALTERNATIVO ; Toma el mensaje de la tabla.
        iorlw   0               ; Comprueba si termina el mensaje.
        btfsc   STATUS,Z
        return
        call    XOR_ENVIA       ; Envia el caracter al LCD.
        incf    cuenta,F
        goto    _Toma_Caracter


;-------------------------------------------------------------------
; Manda al LCD el mensaje que indica visualizacion en modo alternativo.
;-------------------------------------------------------------------

PRIMER_MSJ:
        movlw   0
        call    TOMA_MENSAJES
        return


;-------------------------------------------------------------------
; Manda al LCD "->".
;-------------------------------------------------------------------

MSJ_FLECHA:
        movlw   '-'
        call    XOR_ENVIA
        movlw   '>'
        call    XOR_ENVIA
        return



;-----------------------------------------------------------------
; Manda al LCD el mensaje que muestra la opcion de cambiar el
; numero de caracteres por linea.
;-----------------------------------------------------------------

MSJ_COMANDO_2:

        call    CABECERA_TXT        ; Cabecera del mensaje.
        call    MODO_NORMAL_COMANDO  ; Modo normal de visualizacion.
        call    MSJ_UNO             ; 1->
        call    MSJ_FLECHA          ; Envia el mensaje "CARACT./L".
        call    MSJ_CARACT_L
        call    FINAL_TRAMA         ; Final del mensaje.
        movlw   '1'
        movwf   TECLA_X
        return

;---------------------------------------------------------------
```

```
; Envia al LCD el mensaje "CARACT./L".
;-----------------------------------------------------------------
MSJ_CARACT_L:

        movlw   12
        call    TOMA_MENSAJES
        return




;-----------------------------------------------------------------
; Manda al LCD un mensaje indicando opcion de escribir.
;-----------------------------------------------------------------
MSJ_COMANDO_3:

        call    CABECERA_TXT     ; Cabecera del mensaje
        call    MODO_NORMAL_COMANDO ; Visualizacion normal.
        call    MSJ_DOS          ; 2->
        call    MSJ_FLECHA
        movlw   35
        call    TOMA_MENSAJES    ; Envia el mensaje al LCD.
        call    FINAL_TRAMA      ; Final del mensaje.
        movlw   '2'
        movwf   TECLA_X
        return


;-----------------------------------------------------------------
; Envia mensaje al LCD solicitando la pulsacion de la tecla
; que se desea considerar como telca ESCAPE.
;-----------------------------------------------------------------
MSJ_COMANDO_5:

        call    CABECERA_TXT     ; Cabecera del mensaje.
        call    MODO_NORMAL_COMANDO ; Visualizacion normal.
        call    MSJ_TRES         ; 3->
        call    MSJ_FLECHA
        movlw   107
        call    TOMA_MENSAJES    ; Envia el mensaje.
        call    FINAL_TRAMA
        movlw   '3'
        movwf   TECLA_X
        return


;-----------------------------------------------------------
; Envia la cabecera de los mensajes con parpadeo al LCD.
;-----------------------------------------------------------
CABECERA_TXT:

        call    CARACT_LINEA_16; Configura el LCD a 16 caracteres por
        clrf    auxiliar_7     ; linea.
```

```
        movlw   0x02            ; STX
        call    send
        movlw   0x30            ; Comando de envio de mensajes.
        call    XOR_ENVIA
        movlw   0x81            ; Parpadeo.
        call    XOR_ENVIA
        return

;-----------------------------------------------------------------
; Envia el fin de mensaje al LCD.
;-----------------------------------------------------------------

FINAL_TRAMA:
        movlw   0x03            ; ETX
        call    send
        movfw   auxiliar_7      ; CHECKSUM
        call    send
        return


;-----------------------------------------------------------------
; Envia '1' al LCD.
;-----------------------------------------------------------------

MSJ_UNO:
        movlw   '1'
        call    XOR_ENVIA
        return

;-----------------------------------------------------------------
; Envia '2' al LCD.
;-----------------------------------------------------------------

MSJ_DOS:
        movlw   '2'
        call    XOR_ENVIA
        return

;-----------------------------------------------------------------
; Realiza xor del byte que se envia con los anteriores y
; lo envia por RS-232.
;-----------------------------------------------------------------

XOR_ENVIA:
        xorwf   auxiliar_7,1
        call    send
        return

;-----------------------------------------------------------------
; Envia '6' al LCD.
;-----------------------------------------------------------------

MSJ_SEIS:
        movlw   '6'
        call    XOR_ENVIA
        return


;-----------------------------------------------------------------
```

```
; Envia '5' al LCD.
;-----------------------------------------------------------------

MSJ_CINCO:
        movlw   '5'
        call    XOR_ENVIA
        return


;-----------------------------------------------------------------
; Envia '4' al LCD.
;-----------------------------------------------------------------

MSJ_CUATRO:
        movlw   '4'
        call    XOR_ENVIA
        return


;-----------------------------------------------------------------
; Envia '0' al LCD.
;-----------------------------------------------------------------

MSJ_CERO:
        movlw   '0'
        call    XOR_ENVIA
        return


;-----------------------------------------------------------------
; Envia '3' al LCD.
;-----------------------------------------------------------------

MSJ_TRES:
        movlw   '3'
        call    XOR_ENVIA
        return

;-----------------------------------------------------------------
; Envia '8' al LCD.
;-----------------------------------------------------------------

MSJ_OCHO:
        movlw   '8'
        call    XOR_ENVIA
        return

;-----------------------------------------------------------------
; Envia el numero de caracteres por linea actual seleccionado.
;-----------------------------------------------------------------

MSJ_CARACTERES_LINEA_2:

        call    CABECERA_TXT        ; Cabecera del mensaje.
        call    MODO_NORMAL_COMANDO; Visualizacion en modo normal.
        call    MSJ_CARACT_L        ; Mensaje "CARACT./L".
        call    MSJ_FLECHA          ; ->

        movlw   0x90                ; 16 caracteres por linea.
```

```
275         subwf   CAR_LIN,0           ; Comprueba si la configuracion es de 16
            btfss   STATUS,Z            ; caracteres por linea.
            goto    NO_NO_NO1
            call    MSJ_UNO             ; Muestra mensaje en el LCD: 16.
            call    MSJ_SEIS
280         goto    SE_ACABA_YA

    NO_NO_NO1:
            movlw   0x94                ; 20 caracteres por linea.
            subwf   CAR_LIN,0           ; Comprueba si la configuracion es de 20
285         btfss   STATUS,Z            ; caracteres por linea.
            goto    NO_NO_NO2
            call    MSJ_DOS             ; Muestra mensaje en el LCD: 20.
            call    MSJ_CERO
            goto    SE_ACABA_YA
290
    NO_NO_NO2:
            movlw   0x98
            subwf   CAR_LIN,0
            btfss   STATUS,Z
295         goto    NO_NO_NO3
            call    MSJ_DOS
            call    MSJ_CUATRO
            goto    SE_ACABA_YA

300 NO_NO_NO3:
            movlw   0x9C
            subwf   CAR_LIN,0
            btfss   STATUS,Z
            goto    NO_NO_NO4
305         call    MSJ_DOS
            call    MSJ_OCHO
            goto    SE_ACABA_YA

    NO_NO_NO4:
310         movlw   0xA0
            subwf   CAR_LIN,0
            btfss   STATUS,Z
            goto    NO_NO_NO5
            call    MSJ_TRES
315         call    MSJ_DOS
            goto    SE_ACABA_YA

    NO_NO_NO5:
            movlw   0xA4
320         subwf   CAR_LIN,0
            btfss   STATUS,Z
            goto    NO_NO_NO6
            call    MSJ_TRES
            call    MSJ_SEIS
325         goto    SE_ACABA_YA

    NO_NO_NO6:
            movlw   0xA8
            subwf   CAR_LIN,0
330         btfss   STATUS,Z
            goto    NO_NO_NO7
            call    MSJ_CUATRO
            call    MSJ_CERO
```

```
        goto    SE_ACABA_YA

NO_NO_NO7:
        movlw   0xAC
        subwf   CAR_LIN,0
        btfss   STATUS,Z
        goto    NO_NO_NO8
        call    MSJ_CUATRO
        call    MSJ_CUATRO
        goto    SE_ACABA_YA

NO_NO_NO8:
        movlw   0xB0
        subwf   CAR_LIN,0
        btfss   STATUS,Z
        goto    NO_NO_NO9
        call    MSJ_CUATRO
        call    MSJ_OCHO
        goto    SE_ACABA_YA

NO_NO_NO9:
        movlw   0xB4
        subwf   CAR_LIN,0
        btfss   STATUS,Z
        goto    NO_NO_NO10
        call    MSJ_CINCO
        call    MSJ_DOS
        goto    SE_ACABA_YA

NO_NO_NO10:
        call    MSJ_CINCO
        call    MSJ_SEIS

SE_ACABA_YA:
        call    FINAL_TRAMA
        call    RETARDO_ENTRE_TECLAS ; Evita pasar de comando
        return                       ; de forma involuntaria.


;-----------------------------------------------------------------
; Envia comando al display
;-----------------------------------------------------------------

MODO_NORMAL_COMANDO:
        movlw   0x80                 ; Modo de visualizacion normal
        call    XOR_ENVIA
        return


;-----------------------------------------------------------------
; Envia mensaje al LCD: "TXT:".
;-----------------------------------------------------------------

MSJ_PIDE_MENSAJE_2:

        call    CABECERA_TXT        ; Cabecera del mensaje.
        call    MODO_NORMAL_COMANDO; Visualizacion normal.
        call    MSJTXT
        call    FINAL_TRAMA
```

```
        return

;-----------------------------------------------------------------
; Envia mensaje al LCD: "TXT:" (No envia el fin de trama).
;-----------------------------------------------------------------

MSJ_PIDE_TXT_2:

        call    CABECERA_TXT        ; Cabecera del mensaje.
        call    MODO_NORMAL_COMANDO; Comando de visualizacion normal.
        call    MSJTXT
        return

;-----------------------------------------------------------------
; Envia "TXT:".
;-----------------------------------------------------------------

MSJTXT:
        movlw   'T'
        call    XOR_ENVIA
        movlw   'X'
        call    XOR_ENVIA
        movlw   'T'
        call    XOR_ENVIA
        movlw   ':'
        call    XOR_ENVIA
        return


;-----------------------------------------------------------------
; Envia mensaje al LCD indicando la velocidad de scroll que
; esta seleccionada.
;-----------------------------------------------------------------

MSJ_VELOCIDAD_SCROLL_2:
        call    CABECERA_TXT      ; Cabecera del mensaje.
        call    MODO_NORMAL_COMANDO; Visualizacion modo normal.
        movlw   44
        call    TOMA_MENSAJES     ; Envia el mensaje "VELOCIDAD".
        call    MSJ_FLECHA        ; ->
        movlw   5                 ; Codigo asociado a la velocidad minima.
        subwf   VELOCIDAD_SCROLL,0
        btfss   STATUS,Z
        goto    NO_1
        call    MSJ_UNO           ; Es la velocidad minima: Muestra en el
        goto    ACABA_TRAMA       ; LCD un 1 (velocidad).

NO_1:
        movlw   4
        subwf   VELOCIDAD_SCROLL,0
        btfss   STATUS,Z
        goto    NO_2
        call    MSJ_DOS
        goto    ACABA_TRAMA

NO_2:
        movlw   3
```

```
            subwf   VELOCIDAD_SCROLL,0
            btfss   STATUS,Z
            goto    NO_3
            call    MSJ_TRES
            goto    ACABA_TRAMA

    NO_3:
            movlw   2
            subwf   VELOCIDAD_SCROLL,0
            btfss   STATUS,Z
            goto    NO_4
            call    MSJ_CUATRO
            goto    ACABA_TRAMA

    NO_4:
            call    MSJ_CINCO
                                    ; Velocidad maxima: 5.

    ACABA_TRAMA:
            call    FINAL_TRAMA
            return


    ;-------------------------------------------------------
    ; Muestra en el LCD el retardo del modo alternativo
    ; seleccionado.
    ;-------------------------------------------------------

    MSJ_RETARDO_2:
            call    CABECERA_TXT    ; Cabecera del mensaje.
            call    MODO_NORMAL_COMANDO; Visualizacion normal.

            movlw   54              ; Envia el mensaje "RETARDO".
            call    TOMA_MENSAJES
            call    MSJ_FLECHA      ; ->
            movlw   10
            subwf   RETARDO_M_ALTERN,0 ; Comprueba si retardo < 10.
            btfsc   STATUS,C        ; Si es negativo es menor que 10.
            goto    NO_NO1
            movfw   RETARDO_M_ALTERN
            addlw   48              ; Convierte el numero en caracter
            call    XOR_ENVIA       ; numerico.
            goto    F_F

    NO_NO1:
            movlw   20              ; Comprueba si retardo < 20.
            subwf   RETARDO_M_ALTERN,0
            btfsc   STATUS,C        ; Si es negativo es menor que 20.
            goto    NO_NO2
            call    MSJ_UNO
            movfw   RETARDO_M_ALTERN
            addlw   38              ; No se le suma 48 para obtener el
                                    ; caracter numerico porque el numero
                                    ; a convertir es mayor que 10.
            call    XOR_ENVIA
            goto    F_F

    NO_NO2:
            movlw   30
```

```
        subwf   RETARDO_M_ALTERN,0
        btfsc   STATUS,C
        goto    NO_NO3
        call    MSJ_DOS
        movfw   RETARDO_M_ALTERN
        addlw   28              ; No se le suma 48 para obtener el caracter
                                ; numerico porque el numero a convertir es
                                ; mayor que 20.
        call    XOR_ENVIA
        goto    F_F

NO_NO3:
        movlw   40
        subwf   RETARDO_M_ALTERN,0
        btfsc   STATUS,C
        goto    NO_NO4
        call    MSJ_TRES
        movfw   RETARDO_M_ALTERN
        addlw   18              ; No se le suma 48 para obtener el caracter
                                ; numerico porque el numero a convertir es
                                ; mayor que 30.
        call    XOR_ENVIA
        goto    F_F

NO_NO4:
        call    MSJ_CUATRO
        call    MSJ_CERO

F_F:
        call    FINAL_TRAMA
        call    R_TECLAS_PEQUE  ; Retardo que evita que se reciban
                                ; senales IR seguidas y salte
                                ; involuntariamente este menu.
        return

;-----------------------------------------------------------------------
; Envia el mensaje "SCROLL".
;-----------------------------------------------------------------------

MSJ_SCROLL:
       movlw   22
       call    TOMA_MENSAJES
       return

;-----------------------------------------------------------------------
; Envia '>'.
;-----------------------------------------------------------------------

CAR_MAYOR:
      movlw   '>'
      call    XOR_ENVIA
      return


;-----------------------------------------------------------------------
; Muestra en el display el modo de visualizacion seleccionado.
;-----------------------------------------------------------------------
```

```
MSJ_OPCION_VISUAL_2:
        call    CABECERA_TXT    ; Cabecera del mensaje.
        call    MODO_NORMAL_COMANDO; Visualizacion normal.
        movlw   5
        subwf   VISUALIZACION,0; Comprueba si el modo de visualizacion
        btfss   STATUS,Z        ; es parpadeo de la mitad de la linea.
        goto    NO_1_1
        call    CAR_MAYOR
        call    MSJ_PARPADEO    ; Envia el mensaje "PARPADEO MITAD".
        movlw   2
        call    TOMA_MENSAJES
        goto    ACABA_TRAMA_1

NO_1_1:
        movlw   4               ; Comprueba si el modo de visualizacion
        subwf   VISUALIZACION,0; es el alternativo.
        btfss   STATUS,Z
        goto    NO_2_1
        call    CAR_MAYOR
        call    PRIMER_MSJ      ; Envia mensaje "ALTERNATIVO".
        goto    ACABA_TRAMA_1

NO_2_1:
        movlw   3               ; Comprueba si el modo de visualizacion
        subwf   VISUALIZACION,0; es el scroll.
        btfss   STATUS,Z
        goto    NO_3_1
        call    CAR_MAYOR
        call    MSJ_SCROLL      ; Envia mensaje "SCROLL".
        goto    ACABA_TRAMA_1

NO_3_1:
        movlw   2               ; Comprueba si el modo de visualizacion
        subwf   VISUALIZACION,0; es el de parpadeo de toda la linea.
        btfss   STATUS,Z
        goto    NO_4_1
        call    CAR_MAYOR
        call    MSJ_PARPADEO    ; Envia mensaje "PARPADEO".
        goto    ACABA_TRAMA_1

NO_4_1:
        movlw   62
        call    TOMA_MENSAJES   ; Envia mensaje "NORMAL".

ACABA_TRAMA_1:
        call    FINAL_TRAMA
        call    R_TECLAS_PEQUE
        return

;-----------------------------------------------------------------------
; Envia mensaje "PARPADEO".
;-----------------------------------------------------------------------

MSJ_PARPADEO:
       movlw   70
       call    TOMA_MENSAJES
       return
```

```
;-----------------------------------------------------------------------
; Envia mensaje "FIN!".
;-----------------------------------------------------------------------

ENVIA_MSJ_APREN_FIN_2:
        call    CABECERA_TXT
        movlw   0x81            ; Modo de parpadeo del mensaje.
        call    XOR_ENVIA
        movlw   80
        call    TOMA_MENSAJES
        call    FINAL_TRAMA
        return



;-----------------------------------------------------------------------
; Envia mensaje al LCD solicitando la pulsacion de una tecla.
;-----------------------------------------------------------------------
ENVIA_MSJ_TECLA_2:
        call    CABECERA_TXT
        movlw   0x81            ; Modo de parpadeo del mensaje.
        call    XOR_ENVIA
        call    CAR_INTERROGA
        call    MSJ_TECLA_2
        movfw   auxiliar        ; Numero de tecla solicitado.
        call    XOR_ENVIA
        movlw   '?'
        call    XOR_ENVIA
        call    FINAL_TRAMA
        return

;-----------------------------------------------------------------------
; Envia el mensaje  "TECLA" al LCD.
;-----------------------------------------------------------------------

MSJ_TECLA_2:
        movlw   85
        call    TOMA_MENSAJES
        return


;-----------------------------------------------------------------------
; Envia '?' al LCD.
;-----------------------------------------------------------------------

CAR_INTERROGA:
       movlw   168
       call    XOR_ENVIA
       return


;-----------------------------------------------------------------------
; Envia al LCD mensaje solicitando pulsacion de la tecla ENTER.
;-----------------------------------------------------------------------

ENVIA_MSJ_TECLA_ENTER_2:
       call    CABECERA_TXT
       movlw   0x81             ; Modo de parpadeo del mensaje.
```

```
        call    XOR_ENVIA
        call    CAR_INTERROGA
        call    MSJ_TECLA_2
        movlw   92
        call    TOMA_MENSAJES
        call    FINAL_TRAMA
        return

;-----------------------------------------------------------------
; Envia al LCD mensaje solicitando pulsacion de la tecla ESCAPE.
;-----------------------------------------------------------------

ENVIA_MSJ_TECLA_ESCAPE_2:
        call    CABECERA_TXT
        movlw   0x81            ; Modo de parpadeo del mensaje.
        call    XOR_ENVIA
        call    CAR_INTERROGA
        call    MSJ_TECLA_2
        movlw   99
        call    TOMA_MENSAJES
        call    FINAL_TRAMA
        return

;-----------------------------------------------------------------
; Configura el LCD a 16 caracteres por linea para mostrar los
; mensajes de programacion.
;-----------------------------------------------------------------

CARACT_LINEA_16:
        clrf    auxiliar_7
        movlw   0x02            ; STX
        call    send
        movlw   0x22            ; Comando de cambio de caracteres
        call    XOR_ENVIA       ; por linea.
        movlw   0x81
        call    XOR_ENVIA
        movlw   0x90            ; 16 caracteres por linea.
        call    XOR_ENVIA
        call    FINAL_TRAMA
        return

;-----------------------------------------------------------------
; Configura el LCD con el numero de caracteres por linea
; establecido.
;-----------------------------------------------------------------

CARACT_LINEA_X:
        clrf    auxiliar_7
        movlw   0x02
        call    send
        movlw   0x22
        call    XOR_ENVIA
        movlw   0x81
        call    XOR_ENVIA
        movfw   CAR_LIN         ; Caracteres por linea establecidos.
        call    XOR_ENVIA
        call    FINAL_TRAMA
```

```
        return

;-----------------------------------------------------------------
; Envia el mensaje definido para mostrar en el LCD
; en el modo de funcionamiento normal.
;-----------------------------------------------------------------

TEXTO_TECDIS:
        call    CARACT_LINEA_X
        clrf    auxiliar_3
        movlw   0x02            ; STX.
        call    send
        movlw   0x30            ; Comando de envio de texto.
        xorwf   auxiliar_3,f
        call    send
        movlw   0x81            ; NLIN.
        xorwf   auxiliar_3,f
        call    send
        movlw   1
        subwf   VISUALIZACION,0 ; Comprueba si el modo de visualizacion es
        btfss   STATUS,Z        ; el normal.
        goto    SIG_1
        movlw   0x80
        goto    MANDA

SIG_1:
        movlw   2               ; Comprueba si el modo de visualizacion es
        subwf   VISUALIZACION,0; el parpadeo.
        btfss   STATUS,Z
        goto    SIG_2
        movlw   0x81
        goto    MANDA

SIG_2:
        movlw   3               ; Comprueba si el modo de visualizacion es
        subwf   VISUALIZACION,0; scroll.
        btfss   STATUS,Z
        goto    SIG_3
        movlw   0x82
        goto    MANDA

SIG_3:
        movlw   4               ; Comprueba si el modo de visualizacion es
        subwf   VISUALIZACION,0; el alternativo.
        btfss   STATUS,Z
        goto    SIG_4
        movlw   0x84
        goto    MANDA

SIG_4:
        movlw   0x88            ; Es el modo de visualizacion de parpadeo
                                ; de la mitad de la linea.
MANDA:
        xorwf   auxiliar_3,f    ; Manda la opcion.
        call    send
        movfw   DIRECCION_EEPROM; Numero de caracteres almacenados en la
                                ; EEPROM.
        movwf   cuenta
```

```
        clrf    auxiliar_2
        movfw   auxiliar_2     ; Pone la direccion de la EEPROM.

MUESTRA_CHAR_LCD:
        call    LEE_EEPROM
        xorwf   auxiliar_3,1   ; XOR dato y dato anterior = auxiliar_3.
                               ; W = dato.
        call    send
        incf    auxiliar_2     ; Apunta a la siguiente direccion EEPROM.
        movfw   auxiliar_2     ; W = siguiente direccion de la EEPROM.
        decfsz  cuenta
        goto    MUESTRA_CHAR_LCD
        movlw   0x03           ; ETX.
        call    send
        movfw   auxiliar_3     ; CHK.
        call    send

        ; Guarda el numero de caracteres almacenados en la EEPROM y
        ; el tipo de visualizacion ademas del numero de caracteres
        ; por linea.

        movfw   DIRECCION_EEPROM
        movwf   auxiliar_2
        movlw   124
        call    ESCRIBE_EEPROM ; Guarda el numero de caracteres de texto
                               ; que hay en la EEPROM.
        movfw   VISUALIZACION
        movwf   auxiliar_2
        movlw   125
        call    ESCRIBE_EEPROM ; Guarda el tipo de visualizacion.
        movfw   CAR_LIN
        movwf   auxiliar_2
        movlw   126
        call    ESCRIBE_EEPROM ; Guarda el numero de caracteres por
                               ; linea.
        return


;-----------------------------------------------------------------
; Envia la secuencia del comando correspondiente al display LCD.
;-----------------------------------------------------------------

ENVIA_PROTOCOLO:
        bsf     STATUS,RP1     ; Banco 2.
        movwf   COMANDO_LCD
        bcf     STATUS,RP1     ; Banco 0.
        movlw   0x02           ; STX.
        call    send
        bsf     STATUS,RP1     ; Banco 2.
        movfw   COMANDO_LCD    ; Comando.
        bcf     STATUS,RP1     ; Banco 0.
        xorwf   auxiliar_2,f
        call    send
        movlw   0x81           ; NLIN.
        xorwf   auxiliar_2,f
        call    send
        movfw   auxiliar_4     ; VELSCR.
        call    send
        movlw   0x03           ; ETX.
```

```
865         call    send
            movfw   auxiliar_2      ; CHK.
            call    send
            return

870
    ;-----------------------------------------------------------------
    ; Elige entre dos mensajes a enviar al LCD: El definido si lo hay,
    ; o el que indica que es necesario programar.
    ;-----------------------------------------------------------------
875
    ELIGE_MENS:
            movlw   4
            movwf   COMANDO
            btfss   APRENDIZAJE_REG,7; Comprueba si el LCD esta
880         goto    NO_MSJ          ; programado.
            call    TEXTO_TECDIS
            return

    NO_MSJ:
885         call    ENVIA_NO_MENSAJE
            return



890 ;-----------------------------------------------------------------
    ; Envia mensaje definido o mensaje de indicacion de la
    ; necesidad de programar.
    ;-----------------------------------------------------------------

895 ENVIA_NO_MENSAJE:
            call    CABECERA_TXT
            movlw   0x81            ; Modo de parpadeo del mensaje
            call    XOR_ENVIA
            movlw   121
900         call    TOMA_MENSAJES
            call    FINAL_TRAMA
            movlw   8
            movwf   cuenta
    otra_vez_mas2:
905
            btfss   PORTB,0
            goto    termina_ya2

            movlw   250
910         call    Retardo_ms
            decfsz  cuenta
            goto    otra_vez_mas2
            movlw   16
            movwf   cuenta
915         call    CABECERA_TXT
            movlw   0x81            ; Modo de parpadeo del mensaje
            call    XOR_ENVIA

    y_mas:
920         movlw   '_'
            call    XOR_ENVIA
            decfsz  cuenta
            goto    y_mas
```

```
        call    FINAL_TRAMA

termina_ya2:
       return
```

A.1.9. retardos.asm

Listing A.9: Programa

```
;-----------------------------------------------------------------
; Retardo de 10ms
;-----------------------------------------------------------------

RETARDO_10ms:
        movlw   10
        call    Retardo_ms          ; 10 ms de retardo.
        return


;-----------------------------------------------------------------
; Retardo para no tomar varias senales de infrarrojos seguidas.
;-----------------------------------------------------------------

RETARDO_ENTRE_TECLAS:

        movlw   3
        call    BUCLE_RETARDO
        return


;-----------------------------------------------------------------
; Retardo para no tomar varias senales de infrarrojos seguidas.
;-----------------------------------------------------------------

R_TECLAS_PEQUE:
        movlw   2
        call    BUCLE_RETARDO
        return

;-----------------------------------------------------------------
; Bucle de retardo para no tomar varias senales IR seguidas.
; Introduce un retardo igual a W multiplicado por 250ms.
;-----------------------------------------------------------------

BUCLE_RETARDO:
        movwf   cuenta
retardando:
        movlw   250
        call    Retardo_ms
        decfsz  cuenta              ; Cada vuelta son 250ms.
        goto    retardando
        return



;-----------------------------------------------------------------
```

```
; Subrrutina de retardo
; El valor que contiene w es el retardo en ms.
;-----------------------------------------------------------------

Retardo_ms:
        movwf   Retardo_ms_Contador+1
        clrf    Retardo_ms_Contador+0

        ; Loop interno de 1ms
Retardo_ms_Loop:
        nop
        decfsz  Retardo_ms_Contador+0,F
        goto    Retardo_ms_Loop
        nop
        decfsz  Retardo_ms_Contador+1,F
        goto    Retardo_ms_Loop
        return


;-----------------------------------------------------------------
; Incrementa la cuenta de tiempo para el control de la
; aparicion de los caracteres.
;-----------------------------------------------------------------

INCREMENTA_CUENTA:
        call RETARDO_10ms
        movlw 200
        subwf CUENTA_TIEMPO,0
        btfss STATUS,C
        INCF CUENTA_TIEMPO          ; Incrementa la cuenta de tiempo cada
                                    ; 10ms.
        return
```

A.2. Aplicación con puerto I2C

A.2.1. I2C6.asm (Maestro)

Listing A.10: Programa

```
;*****************************************************************
;   Receptor de mando a distancia por infrarrojos.              *
;   Envia por puerto I2C, por puerto RS-232 y a un LCD 16x2 el *
;   codigo hexadecimal de las teclas recibidas.                 *
;                                                               *
;*****************************************************************
;       Carlos Arevalo Sillero                                  *
;                                                               *
;*****************************************************************
;       Se utiliza un Reloj de cuarzo de 4MHz.                  *
;       La entrada de la seal es RB0, activa a nivel bajo.      *
;*****************************************************************

        list      p=16F628, R=DEC
        ERRORLEVEL -305
        #include <p16F628.inc>
```

```
17
          __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC & _LVP_OFF


          ; Definicion de constantes.
22
  #define                        I2C_PORT PORTA
  #define                        I2C_SDA  4
  #define                        I2C_SCL  3

27 Direccion_ceros          equ 0x3B
   Direccion_ceros_mas_uno  equ 0x3C
   Direccion_unos           equ 0xA0
   Dir_Senal                equ 0x121
   numero_de_bytes          equ 8
32
   LCD_RS                   equ 1        ; Comando /dato
   LCD_E                    equ 3        ; Habilita Lcd

   LCD_DB4                  equ 4
37 LCD_DB5                  equ 5
   LCD_DB6                  equ 6
   LCD_DB7                  equ 7


42        ; Macro que habilita el LCD.

   SET_EN           MACRO
                    bsf  PORTB,LCD_E
                    ENDM
47
          ; Macro que deshabilita el LCD.
   CLEAR_EN         MACRO
                    bcf  PORTB,LCD_E
                    ENDM
52
          ; Macro que habilita 1us el LCD.

   EN_STROBE        MACRO
                    SET_EN
57                  nop
                    CLEAR_EN
                    ENDM


62        ; Definicion de variables.

          cblock h'20'
   auxiliar_6               ; Variable auxiliar de uso generico.
   DIR_ESCLAVO              ; Direccion del esclavo direccionado en I2C.
67 I2C_TEMP
   cuenta
   cuentabytes
   auxiliar
   tmpLcdRegistro           ; Variables auxiliares para utilizar con el
72 tmpLcdRegistro1          ; LCD 16x2.
   msRetardoContador        ; Variables de cuenta.
   msRetardoContador1
   medidacero_ref           ; Variables para salvaguardar las medidas
```

```
   medidauno_ref            ; de tiempos cero y uno.
77 FSR_TEMP1                ; Almacenan temporalmente el FSR.
   FSR_TEMP2
   w_temp                   ; Guarda W.
   status_temp              ; Guarda STATUS.
   nummedidascero
82 MEDIDACERO               ; Guarda medidas cero.
   MEDIDASCEROIGUALES       ; Contiene el numero de medidas cero iguales.
   nummedidasuno            ; Guarda el numero de medidas uno.
   MEDIDAUNO
   MEDIDASUNOIGUALES        ; Guarda el numero de medidas uno iguales.
87 temp_medcero             ; Almacenan temporalmente medidas.
   temp_meduno
   MEDIDAUNOPEQUE           ; Medida uno mas pequena.
   MEDIDACEROPEQUE          ; Medida cero mas pequena.
   temp_peque
92
   tiempos1                 ; Direccion de inicio de la zona de memoria
                            ; se almacenan los ceros de la senal.
           endc

97         cblock h'A0'
   tiempos2                 ; Direccion de inicio de la zona de memoria
                            ; donde se almacenan los unos.
           endc

102        cblock h'120'
   RS232_LCD                ; Indica si el byte se envia al LCD o por I2C.
   byte1                    ; Bytes que almacenan la senal IR.
   byte2
   byte3
107 byte4
   byte5
   byte6
   byte7
   byte8
112
           endc

           ORG     0x000    ; Vector de reset.
           goto    Main     ; Va al inicio del programa.
117
   ;-----------------------------------------------------------------
   ; Controlador de interrupcion.
   ; Detecta la senal, obtiene la senal binaria correspondiente
   ; y la envia por los puertos.
122 ;-----------------------------------------------------------------

           ORG     0x004              ; Interrupcion: se ha recibido senal.
           movwf   w_temp             ; Guarda el contenido de W...
           movf    STATUS,w           ; ...y guarda el contenido de status.
127        movwf   status_temp

           call    Rec_RC5_RE80       ; Recibe los datos.

           call    LcdInic            ; Inicializa el LCD.
132
           call    I2C_COMUNICACION   ; Manda los bytes de la senal por
                                      ; puerto I2C.
```

```
        bsf     STATUS,RP1
        bcf     RS232_LCD,5         ; Indica que se escribe en el LCD.
        bcf     STATUS,RP1

        ;Posiciona el cursor del LCD en la posicion 0,0.
        movlw   0x00
        call    LcdColoca
        call    ENVIA_DATO          ; Escribe en el LCD.


        bsf     STATUS,RP1          ; Banco 2.
        bsf     RS232_LCD,5         ; Pone a '1' el bit 5 para indicar
        bcf     STATUS,RP1          ; que envia dato via RS232.
        call    INI_RS232

        call    ENVIA_DATO          ; Envia los caracteres de la senal
                                    ; por RS232.
        bsf     STATUS,RP0          ; Banco 1

        movlw B'10000110'           ; Int. en flanco descendente.
        movwf   OPTION_REG
        bcf     STATUS,RP0          ; Banco 0.
        bcf     INTCON,INTF         ; Restaura el flag de interrupcion,
        movf    status_temp,w       ; restaura el contenido de status...
        movwf   STATUS
        swapf   w_temp,f
        swapf   w_temp,w            ; ...y restaura el contenido de w.
        retfie                      ; retorna de la interrupcion.



;-----------------------------------------------------------------------
; Envia byte a byte los bytes de la senal por el puerto RS232 o bien al
; LCD 16x2 en funcion del bit 5 de RS232_LCD.
;-----------------------------------------------------------------------

ENVIA_DATO:
        movlw   8                   ; Recorre 8 bytes de datos.
        movwf   cuentabytes
        clrf    cuenta              ; Variable cuyo valor se suma al PCL
                                    ; para obtener el byte que se envia.
        goto    byte_a_byte         ; Toma el byte que se envia.

_Toma_Caracter:
        incf    cuenta,F
        incf    cuenta,F

byte_a_byte:
        movfw   cuenta
        bsf     STATUS,RP1          ; Banco 2.
        call    BYTE_a_W            ; Suma cuenta al PCL y toma byte.

continua_mas:
        bcf     STATUS,RP1          ; Banco 0.
        andlw   b'11110000'         ; Toma 4 bits mas significativos.
        movwf   auxiliar
        bcf     STATUS,C
        rrf     auxiliar,f          ; Desplaza 4 a la derecha.
```

```
        rrf     auxiliar,f
        rrf     auxiliar,f
        rrf     auxiliar,f
        call    A_CARACTER          ; Lo pasa a hexadecimal.
        movfw   cuenta
        bsf     STATUS,RP1          ; Banco 2.
        call    BYTE_a_W

continua_mas2:
        bcf     STATUS,RP1
        andlw   b'00001111'         ; Toma 4 bits menos significativos.
        movwf   auxiliar
        call    A_CARACTER          ; Lo pasa a hexadecimal.
        decfsz  cuentabytes
        goto    _Toma_Caracter
        return

A_CARACTER:
        movlw   0xA                 ; Obtiene el caracter en formato hx.
        subwf   auxiliar,0
        btfsc   STATUS,C
        goto    mayor_igual_que_A

menor_que_A:
        movfw   auxiliar            ; 0 - 9.
        addlw   48
        goto    salta

mayor_igual_que_A:                  ; A - F.
        movfw   auxiliar
        addlw   55

salta:
        bsf     STATUS,RP1          ; Banco 1.
        btfsc   RS232_LCD,5
        goto    RS232
        bcf     STATUS,RP1          ; Banco 0.
        call    LcdEnviaDato        ; Envia el caracter al LCD.
        goto    vuelve

RS232:
        bcf     STATUS,RP1          ; Banco 0.
        call    send                ; Envia caracter via RS232.
vuelve:
        return


;-------------------------------------------------------------
; Elige el byte correspondiente que se copia en W, segun
; el valor de PCL.
;-------------------------------------------------------------


BYTE_a_W:
        addwf  PCL
        movfw byte1
        return
        movfw byte2
        return
```

```
        movfw byte3
        return
        movfw byte4
        return
        movfw byte5
        return
        movfw byte6
        return
        movfw byte7
        return
        movfw byte8
        return


;-----------------------------------------------------------------
;                Programa principal
;-----------------------------------------------------------------

Main
        bcf     STATUS,RP0          ; Banco 0.
        bcf     STATUS,RP1
        movlw   0x07                ; Pone los comparadores en off
        movwf   CMCON               ; permitiendo el uso de los pines
                                    ; I/O.
        bsf     STATUS,RP0          ; Banco 1.

        movlw   B'10000110'         ; Int. en flanco descendente por RB0.
        movwf   OPTION_REG
        movlw   B'00000011'
        movwf   TRISB               ; Configura RB0 y RB1 como entradas.
        bsf     TRISA,0             ; Configura como entrada RA0.


        bcf     TRISA,1
        bcf     STATUS,RP0          ; Banco 0.

        call    LcdInic             ; Inicializa el LCD.

        bsf     INTCON,INTE         ; Habilita la interrupcion externa.
        bsf     INTCON,GIE          ; Habilita las interrupciones.

BUCLE_PRINCIPAL:
        goto BUCLE_PRINCIPAL        ; Espera que la llegada de senal
                                    ; produzca una interrupcion.

        INCLUDE "recibe.asm"
        INCLUDE "lcd.asm"
        INCLUDE "rs232.asm"
        INCLUDE "i2c.asm"

        END
```

A.2.2. recibe.asm

Ver sección A.1.2.

A.2.3. i2c.asm

Listing A.11: Programa

```
;-----------------------------------------------------------------
; Condicion de inicio generada que genera el maestro en el bus I2C.
;-----------------------------------------------------------------

I2C_INICIO:
        bsf    STATUS,RP0
        bcf    TRISA,I2C_SDA
        bcf    STATUS,RP0

        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL          ; INICIO I2C: Cambia SDA de
        call   I2C_Retardo               ; 1 a 0 cuando SCL = 1.
        bcf    I2C_PORT,I2C_SDA
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL
        return

;-----------------------------------------------------------------
; Condicion de parada que genera el maestro en el bus I2C.
;-----------------------------------------------------------------

I2C_PARADA:
        bsf    STATUS,RP0
        bcf    TRISA,I2C_SDA
        bcf    STATUS,RP0

        bcf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL           ; PARADA I2C: Cambia SDA de
        call   I2C_Retardo                ; 0 a 1 cuando SCL = 1.
        bsf    I2C_PORT,I2C_SDA
        return


;-----------------------------------------------------------------
; Espera el ACK del dispositivo esclavo.
;-----------------------------------------------------------------

I2C_ACK_esclavo:
        bsf    STATUS,RP0
        bsf    TRISA,I2C_SDA
        bcf    STATUS,RP0

        call   I2C_Retardo                ; ACK del esclavo I2C:
        bsf    I2C_PORT,I2C_SCL           ; SDA a 1 por el maestro.
        call   I2C_Retardo                ; SDA=1 si esclavo no responde.
        bsf    STATUS,Z                   ; SDA=0 si esclavo responde.
        btfsc  I2C_PORT,I2C_SDA
        bcf    STATUS,Z                   ; STATUS,Z=1 si respondio.
        bcf    I2C_PORT,I2C_SCL           ; STATUS,Z=0 si no respondio.
        return

;-------------------------------------------------------------------
; Envia el byte por el puerto I2C al dispositivo esclavo direccionado.
;-------------------------------------------------------------------

```

```
I2C_ENVIA:
        movwf  I2C_TEMP                    ; Envia por I2C el byte de W.

        bsf    STATUS,RP0
        bcf    TRISA,I2C_SDA
        bcf    STATUS,RP0

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,7
        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,6
        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,5
        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,4
        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,3
        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,2
        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,1
        bsf    I2C_PORT,I2C_SDA
        bsf    I2C_PORT,I2C_SCL
        call   I2C_Retardo
        bcf    I2C_PORT,I2C_SCL

        bcf    I2C_PORT,I2C_SDA
        btfsc  I2C_TEMP,0
        bsf    I2C_PORT,I2C_SDA
```

```
115         bsf    I2C_PORT,I2C_SCL
            call   I2C_Retardo
            bcf    I2C_PORT,I2C_SCL

            return
120

    I2C_Retardo:
           goto    $+1                    ; Frecuencia del PIC = 4MHz.
           goto    $+1                    ; Espera 12us (12 ciclos de 1us).
125        goto    $+1                    ; Frecuencia I2C = 83KHz (1/12 *1000000).
           goto    $+1
           return


130
    ;-------------------------------------------------------------
    ; Envia por I2C todos los bytes de la senal IR recibida.
    ;-------------------------------------------------------------

135 I2C_COMUNICACION:

            movlw  b'00000000'
            movwf  DIR_ESCLAVO
            bsf    I2C_PORT,I2C_SDA
140
            ; Configura el puerto I2C.

            bsf    STATUS,RP0                  ; Banco 1.
            bcf    TRISA,I2C_SCL               ; SCL y SDA son salidas.
145         bcf    TRISA,I2C_SDA
            bcf    STATUS,RP0                  ; Banco 0.
            bsf    I2C_PORT,I2C_SDA


150 mas:    call   I2C_INICIO                  ; Condicion de comienzo.
            movfw  DIR_ESCLAVO                 ; Guarda la direccion del esclavo.
            call   I2C_ENVIA
            call   I2C_ACK_esclavo             ; Espera el noveno pulso: ACK.
            btfss  STATUS,Z                    ; Comprueba Z y si es 1 hubo ACK.
155         goto   N_ACK                       ; Esclavo no respondio.

            bsf    STATUS,RP1                  ; Banco 2.
            movfw  byte1
            bcf    STATUS,RP1                  ; Banco 0.
160         call   I2C_ENVIA                   ; Envia el byte por I2C.
            call   I2C_ACK_esclavo             ; Espera el ACK del esclavo.
            btfss  STATUS,Z                    ; Comprueba Z y si es 1 hubo ACK.
            goto   N_ACK

165         bsf    STATUS,RP1
            movfw  byte2
            bcf    STATUS,RP1
            call   I2C_ENVIA
            call   I2C_ACK_esclavo
170         btfss  STATUS,Z
            goto   N_ACK

            bsf    STATUS,RP1
```

```
        movfw  byte3
        bcf    STATUS,RP1
        call   I2C_ENVIA
        call   I2C_ACK_esclavo
        btfss  STATUS,Z
        goto   N_ACK

        bsf    STATUS,RP1
        movfw  byte4
        bcf    STATUS,RP1
        call   I2C_ENVIA
        call   I2C_ACK_esclavo
        btfss  STATUS,Z
        goto   N_ACK

        bsf    STATUS,RP1
        movfw  byte5
        bcf    STATUS,RP1
        call   I2C_ENVIA
        call   I2C_ACK_esclavo
        btfss  STATUS,Z
        goto   N_ACK

        bsf    STATUS,RP1
        movfw  byte6
        bcf    STATUS,RP1
        call   I2C_ENVIA
        call   I2C_ACK_esclavo
        btfss  STATUS,Z
        goto   N_ACK

        bsf    STATUS,RP1
        movfw  byte7
        bcf    STATUS,RP1
        call   I2C_ENVIA
        call   I2C_ACK_esclavo
        btfss  STATUS,Z
        goto   N_ACK

        bsf    STATUS,RP1
        movfw  byte8
        bcf    STATUS,RP1
        call   I2C_ENVIA
        call   I2C_ACK_esclavo
        btfss  STATUS,Z
        goto   N_ACK

N_ACK   call   I2C_PARADA                    ; Condicion de parada.

        return
```

A.2.4. rs232.asm

Ver sección A.1.4.

A.2.5. lcd.asm

Listing A.12: Programa

```
;-----------------------------------------------------------------------
; Inicializa el LCD 16x2.
;-----------------------------------------------------------------------

LcdInic:
        bcf     PORTB,LCD_E                     ; Deshabilita el LCD.
        bcf     PORTA,LCD_RS                    ; Pone el LCD en modo comando.
        movlw   10
        call    msRetardo

        ; Envia al LCD la secuencia de reset

        bsf     PORTB,LCD_DB4                   ; Envia 0x03
        bsf     PORTB,LCD_DB5
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        EN_STROBE

        movlw   2
        call    msRetardo                       ; Espera 2ms.

        EN_STROBE

        movlw   2
        call    msRetardo

        EN_STROBE

        movlw   2
        call    msRetardo

        bcf     PORTB,LCD_DB4                   ; Envia 0x02.
        bsf     PORTB,LCD_DB5
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        EN_STROBE

        movlw   2
        call    msRetardo

        ; Configura el bus de 4 bits.

        movlw   0x28
        call    LcdEnviaComando

        ; Incremento, no desplazamiento.

        movlw   0x06
        call    LcdEnviaComando

        ; Display ON, Cursor OFF, Parpadeo OFF.
```

```
        movlw   0x0C
        call    LcdEnviaComando

        movlw   0x01
        call    LcdEnviaComando

        return

;-----------------------------------------------------------------
; Coloca el cursor en el LCD.
;-----------------------------------------------------------------

LcdColoca:
        movwf   tmpLcdRegistro+0
        movlw   0x80
        movwf   tmpLcdRegistro+1
        movf    tmpLcdRegistro+0,W
        andlw   0x0F
        iorwf   tmpLcdRegistro+1,F
        btfsc   tmpLcdRegistro+0,4
        bsf     tmpLcdRegistro+1,6
        movf    tmpLcdRegistro+1,W
        call    LcdEnviaComando
        return

;-----------------------------------------------------------------
; Envia un dato al LCD
;-----------------------------------------------------------------

LcdEnviaDato:
        bsf     PORTA,LCD_RS            ; Pone RS en modo dato.
        call    LcdEnviaByte            ; Envia el byte al LCD.
        return

;-----------------------------------------------------------------
; Envia un comando al LCD
;-----------------------------------------------------------------

LcdEnviaComando:
        bcf     PORTA,LCD_RS            ; Pone RS en modo comando.
        call    LcdEnviaByte            ; Envia el comando al LCD.
        return

;-----------------------------------------------------------------
; Envia un byte a traves del bus de 4 hilos.
;-----------------------------------------------------------------

LcdEnviaByte
        ; Guarda el valor que se envia.
        movwf   tmpLcdRegistro

        ; Envia los 4 bits mas significativos.

        bcf     PORTB,LCD_DB4
        bcf     PORTB,LCD_DB5
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        btfsc   tmpLcdRegistro,4
```

```
        bsf     PORTB,LCD_DB4
        btfsc   tmpLcdRegistro,5
        bsf     PORTB,LCD_DB5
        btfsc   tmpLcdRegistro,6
        bsf     PORTB,LCD_DB6
        btfsc   tmpLcdRegistro,7
        bsf     PORTB,LCD_DB7

        EN_STROBE                       ; Habilita y deshabilita
                                        ; el LCD.
        movlw   2
        call    msRetardo               ; Espera 2ms.

        ; Envia los 4 bits menos significativos.

        bcf     PORTB,LCD_DB4
        bcf     PORTB,LCD_DB5
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        btfsc   tmpLcdRegistro,0
        bsf     PORTB,LCD_DB4
        btfsc   tmpLcdRegistro,1
        bsf     PORTB,LCD_DB5
        btfsc   tmpLcdRegistro,2
        bsf     PORTB,LCD_DB6
        btfsc   tmpLcdRegistro,3
        bsf     PORTB,LCD_DB7

        EN_STROBE

        movlw   2
        call    msRetardo

        return

;-----------------------------------------------------------------
; Rutina de retardo.
;
; W = retardo en ms (con cristal de 4MHz)
;-----------------------------------------------------------------

msRetardo:
        movwf   msRetardoContador+1
        clrf    msRetardoContador+0

        ; Loop interno de 1 ms.
msRetardoBucle:
        nop
        decfsz  msRetardoContador+0,F
        goto    msRetardoBucle
        nop

        decfsz  msRetardoContador+1,F
        goto    msRetardoBucle

        return
```

A.2.6. I2C8.asm (Esclavo)

Listing A.13: Programa

```
;*************************************************************
;   Programa esclavo I2C.                                    *
;   Recibe por puerto I2C los bytes de la senal IR           *
;   y los muestra por un LCD en formato hexadecimal.         *
;                                                            *
;*************************************************************
;     Carlos Arevalo Sillero                                 *
;                                                            *
;*************************************************************
;     Se utiliza un Reloj de cuarzo de 4MHz.                 *
;*************************************************************


        LIST P=16F84A, R=DEC
        ERRORLEVEL -305,-302
        INCLUDE "P16F84A.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF


; Definicion de constantes.

I2C_PUERTO      equ  PORTB
SDA             equ  0
SCL             equ  1


LCD_RS          equ  2        ; Comando/dato.
LCD_E           equ  3        ; Habilitacion del Lcd.

; Bus de datos del LCD

LCD_DB4         equ  4
LCD_DB5         equ  5
LCD_DB6         equ  6
LCD_DB7         equ  7


; Macro que habilita el LCD.

SET_EN          MACRO
                bsf  PORTB,LCD_E
                ENDM

; Macro que deshabilita el LCD.

CLEAR_EN        MACRO
                bcf  PORTB,LCD_E
                ENDM

; Macro que habilita el LCD 1us.

EN_STROBE       MACRO
                SET_EN
```

```
                nop
                CLEAR_EN
                ENDM

        ORG     0x0C

; Definicion de variables.

inicio          res     1       ; Indicar condicion de start.
parada          res     1       ; Indicar condicion de stop.
W_TEMP          res     1       ; Variables para salvar el contexto en
STATUS_TEMP     res     1       ; las interrupciones.
para_seleccion  res     1       ; Para comprobar si he sido seleccionado.
para_datos      res     1       ; Para ver si se mando comando o dato.
I2C_Data        res     1       ; Dato recibido.
contaplus       res     1       ; Variable auxiliar.
tmpLcdRegistro  res     2
msRetardoContador  res  2
caracteraux     res     1       ; Almacena de forma temporal un byte
                                ; que se se pasa a hx.
caracteraux2    res     1
byte1           res     1       ; Bytes de la senal IR.
byte2           res     1
byte3           res     1
byte4           res     1
byte5           res     1
byte6           res     1
byte7           res     1
byte8           res     1
contador        res     1


        ORG     0
        goto    start

        ORG     4

        ; Salva el contexto.

        MOVWF   W_TEMP          ; W_TEMP=W.
        SWAPF   STATUS,W        ; W=swap(STATUS).
        MOVWF   STATUS_TEMP     ; STATUS_TEMP=W.

        btfss   I2C_PUERTO,SCL  ; SCL = 1?
        goto    no_interesa     ; Si SCL=0 y hay cambio en SDA no
                                ; es condicion de start ni de stop.
        btfsc   I2C_PUERTO,SDA  ; SDA=0?
        goto    condicion_stop  ; SCL=1 y SDA =0->1 condicion de stop.

        clrf    INTCON
        movlw   d'16'           ; SCL=1 y SDA =1->0 condicion de start.
        movwf   inicio          ; inicio(4)->1.
        bcf     INTCON,RBIF     ; borramos el bit pegajoso.

        ; Recupera el contexto.
```

```
        SWAPF   STATUS_TEMP,W   ; W=swap(STATUS_TEMP).
        MOVWF   STATUS          ; STATUS=W.
        SWAPF   W_TEMP,F        ; W_TEMP=swap(W_TEMP).
        SWAPF   W_TEMP,W        ; W=SWAP(W_TEMP).

        retfie

condicion_stop

        movlw   d'16'
        movwf   parada          ; parada(4)->1.
        bcf     INTCON,RBIF     ; borramos bit pegajoso.

        ; Recupera el contexto.

        SWAPF   STATUS_TEMP,W   ; W=swap(STATUS_TEMP).
        MOVWF   STATUS          ; STATUS=W.
        SWAPF   W_TEMP,F        ; W_TEMP=swap(W_TEMP).
        SWAPF   W_TEMP,W        ; W=SWAP(W_TEMP).

        retfie

no_interesa
        bcf     INTCON,RBIF     ; borramos bit pegajoso.

        ; Recupera el contexto.

        SWAPF   STATUS_TEMP,W   ; W=swap(STATUS_TEMP).
        MOVWF   STATUS          ; STATUS=W.
        SWAPF   W_TEMP,F        ; W_TEMP=swap(W_TEMP).
        SWAPF   W_TEMP,W        ; W=SWAP(W_TEMP).

        retfie

start   ; Comienzo del programa principal.

        bsf      STATUS,RP0
        movlw    b'00000011'    ; RB0 y RB1 entradas.
        movwf    TRISB
        movlw    b'00000000'
        movwf    OPTION_REG
        bcf      STATUS,RP0


        bsf     STATUS,RP0      ; Seleccion del banco 1 de registros.
        movlw   b'10010000'     ; GIE(7)=1 RBIE(3)=1 RBIF(0).
        movwf   INTCON          ; Definir interrupciones habilitadas.
        bcf     STATUS,RP0      ; Banco 0.

        clrf    inicio          ; inicio=0, inicializacion de registros.
        clrf    parada          ; parada=0.


esperar_inicio

        btfss   inicio,4        ; inicio(4)=1?->si, entonces hubo start
        goto    esperar_inicio  ; sino esperar.
```

```
174         bsf     STATUS,RP0        ; Banco 1.
            clrf    INTCON            ; Borramos el registro de interrupciones.
            bcf     STATUS,RP0        ; Banco 0.

            bcf     inicio,4          ; Borramos bits de indicacion.
179         bcf     parada,4          ; Borramos bit de indicacion de stop.

            call    Recibe_Byte_I2C   ; Recibe el primer byte que identifica
                                      ; un esclavo.

184         movlw   b'00000000'       ; Cargamos en w el valor a comparar.
            movwf   para_seleccion    ; Actualizamos registro de seleccion.
            movfw   I2C_Data          ; Cargamos en w el valor recibido.
            subwf   para_seleccion,W; Comparamos con nuestra secuencia de
                                      ; activacion.
189         btfss   STATUS,Z          ; Z = 1 ?
            goto    esperar_inicio    ; no eramos nosotros para quien iba la
                                      ; informacion
            call    Envio_ACK_I2C     ; enviamos el bit de que nos hemos
                                      ; enterado.
194
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte1             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
            call    Envio_ACK_I2C     ; Enviamos reconocimiento: SDA = 0.
199
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte2             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
            call    Envio_ACK_I2C     ; Enviamos reconocimiento: SDA = 0.
204
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte3             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
            call    Envio_ACK_I2C     ; Enviamos reconocimiento: SDA = 0.
209
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte4             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
            call    Envio_ACK_I2C     ; Enviamos reconocimiento: SDA = 0.
214
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte5             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
            call    Envio_ACK_I2C     ; Enviamos reconocimiento: SDA = 0.
219
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte6             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
            call    Envio_ACK_I2C     ; Enviamos reconocimiento: SDA = 0.
224
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte7             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
            call    Envio_ACK_I2C     ; Enviamos reconocimiento: SDA = 0.
229
            call    Recibe_Byte_I2C   ; Recibimos el  byte.
            movwf   byte8             ; Cargamos en w el byte recibido.
                                      ; Aqui se trata de un caracter.
```

```
        call    Envio_ACK_I2C   ; Enviamos reconocimiento: SDA = 0.
234
        bsf     STATUS,RP0      ; Banco 1.

        movlw   00011110B       ; Configura el puerto A.
        movwf   TRISA
239
        movlw   00000010B       ; Configura el puerto B.
        movwf   TRISB
        bcf     STATUS,RP0      ; Banco 0.

244     ; Inicializa el display LCD.
        call    LcdInic

        ; Coloca el cursor en la posicion 0,0.

249     movlw   0x00
        call    LcdColoca

        ; Envia los bytes al LCD.

254     movfw   byte1
        call    Pinta

        movfw   byte2
        call    Pinta
259
        movfw   byte3
        call    Pinta

        movfw   byte4
264     call    Pinta

        movfw   byte5
        call    Pinta

269     movfw   byte6
        call    Pinta

        movfw   byte7
        call    Pinta
274
        movfw   byte8
        call    Pinta


279     ; Se realiza una nueva inicializacion.

        bsf      STATUS,RP0
        movlw    b'00000011'
        movwf    TRISB          ; RB0 y RB1 son entradas.
284     movlw    b'00000000'
        movwf    OPTION_REG
        bcf      STATUS,RP0


289     bsf     STATUS,RP0      ; Banco 1.
        movlw   b'10010000'     ; GIE(7)=1 RBIE(3)=1 RBIF(0)
        movwf   INTCON          ; Definir interrupciones habilitadas.
```

```
        bcf     STATUS,RP0      ; Banco 0.

        clrf    inicio          ; inicio=0, inicializacion de registros.
        clrf    parada          ; parada=0.

        ; Acaba la nueva reinicializacion y vuelve al principio a esperar
        ; interrupcion por la linea SDA.

        goto    esperar_inicio  ; Espera nueva secuencia.


        INCLUDE   "lcdesc.asm"
        INCLUDE   "i2cesc.asm"

        END
```

A.2.7. i2cesc.asm

Listing A.14: Programa

```
;-----------------------------------------------------------------
; Recibe byte por el puerto I2C.
;-----------------------------------------------------------------

Recibe_Byte_I2C:
        clrf    I2C_Data        ; I2C_Data = 0.
        movlw   d'8'            ; W=8 n de bits que vamos a recibir.
        movwf   contaplus       ; contaplus = 8.
rotar
        rlf     I2C_Data,F      ; Rotamos para meter luego el valor
                                ; recibido.
aguantar
        btfss   I2C_PUERTO,SCL  ; SCL=1? ---  RA2 = 1?.
        goto    aguantar        ; Esperamos hasta que SCL=1 para
                                ; leer el dato.
        btfss   I2C_PUERTO,SDA  ; SDA = 1?
        goto    SDA_cero
        bsf     I2C_Data,0      ; bit recibido uno.
        goto    continuar
SDA_cero
        bcf     I2C_Data,0      ; bit recibido uno.
continuar
        ; Verificaremos que no se ha producido condicion de inicio
        ; o stop.

        btfsc   I2C_PUERTO,SCL  ; SCL = 0?
        goto    continuar

        decfsz  contaplus,F     ; conatdor=0?
        goto    rotar           ; contador<>0 seguir rotando.
        movfw   I2C_Data
        return                  ; contador=0 return.

;-----------------------------------------------------------------
; Envia el ACK al Maestro.
;-----------------------------------------------------------------

```

```
Envio_ACK_I2C:                          ; Cuando se llame a esta subrutina el
                                        ; reloj estara en baja, luego el maestro
                                        ; sube y baja el reloj y comprueba si le
                                        ; ha llegado el ACK.
        call    SDA_Low                 ; SDA a nivel bajo para mandar el ACK.
esperar btfss   I2C_PUERTO,SCL          ; SCL=1?
        goto    esperar                 ; SCL<>1, esperamos al pulso de reloj.

esp     btfsc   I2C_PUERTO,SCL          ; SCL=0?
        goto    esp                     ; SCL=1 -> se espera a que baje el reloj.

        call    SDA_High                ; Para que el maestro pueda mandar mas
        return                          ; datos.

;-----------------------------------------------------------------
SDA_High:
        bsf   STATUS,RP0                ; Banco 1.
        bsf   TRISB,SDA                 ; SDA entrada nivel alto.
        bcf   STATUS,RP0                ; Banco 0.
        return
;-----------------------------------------------------------------

SDA_Low:
        bcf   I2C_PUERTO,SDA            ; SDA = 0.
        bsf   STATUS,RP0                ; Banco 1.
        bcf   TRISB,SDA                 ; SDA salida.
        bcf   STATUS,RP0                ; Banco 0.
        return
;-----------------------------------------------------------------

SCL_High:
        bsf   STATUS,RP0                ; Banco 1.
        bsf   TRISB,SCL                 ; SCL entrada nivel alto.
        bcf   STATUS,RP0                ; Banco 0.
        return
;-----------------------------------------------------------------

SCL_Low:
        bcf   I2C_PUERTO,SCL            ; SCL = 0.
        bsf   STATUS,RP0                ; Banco 1.
        bcf   TRISB,SCL                 ; SCL salida
        bcf   STATUS,RP0                ; Banco 0.
        return
;-----------------------------------------------------------------

I2C_Init:
        call SDA_High
        call SCL_High
        retlw 0
;-----------------------------------------------------------------

I2C_Delay:
        nop
        retlw 0                     ; Duracion 3+2 = 5 us.
;-----------------------------------------------------------------
```

A.2.8. lcdesc.asm

Listing A.15: Programa

```
;-----------------------------------------------------------
; Inicializa el LCD 16x2.
;-----------------------------------------------------------

LcdInic:
        bcf     PORTB,LCD_E             ; Deshabilita el LCD.
        bcf     PORTB,LCD_RS            ; Pone el LCD en modo comando.
        movlw   10
        call    msRetardo

        ; Envia al LCD la secuencia de reset

        bsf     PORTB,LCD_DB4           ; Envia 0x03
        bsf     PORTB,LCD_DB5
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        EN_STROBE

        movlw   2
        call    msRetardo               ; Espera 2ms.

        EN_STROBE

        movlw   2
        call    msRetardo

        EN_STROBE

        movlw   2
        call    msRetardo

        bcf     PORTB,LCD_DB4           ; Envia 0x02.
        bsf     PORTB,LCD_DB5
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        EN_STROBE

        movlw   2
        call    msRetardo

        ; Configura el bus de 4 bits.

        movlw   0x28
        call    LcdEnviaComando

        ; Incremento, no desplazamiento.

        movlw   0x06
        call    LcdEnviaComando

        ; Display ON, Cursor OFF, Parpadeo OFF.
```

```
57          movlw   0x0C
            call    LcdEnviaComando

            movlw   0x01
            call    LcdEnviaComando
62
            return


;-----------------------------------------------------------
67 ; Coloca el cursor en el LCD.
;-----------------------------------------------------------

LcdColoca:
            movwf   tmpLcdRegistro+0
72          movlw   0x80
            movwf   tmpLcdRegistro+1
            movf    tmpLcdRegistro+0,W
            andlw   0x0F
            iorwf   tmpLcdRegistro+1,F
77          btfsc   tmpLcdRegistro+0,4
            bsf     tmpLcdRegistro+1,6
            movf    tmpLcdRegistro+1,W
            call    LcdEnviaComando
            return
82

;-----------------------------------------------------------
; Envia un dato al LCD
;-----------------------------------------------------------
87
LcdEnviaDato:
            bsf     PORTB,LCD_RS        ; Pone RS en modo dato.
            call    LcdEnviaByte        ; Envia el byte al LCD.
            return
92
;-----------------------------------------------------------
; Envia un comando al LCD
;-----------------------------------------------------------

97 LcdEnviaComando:
            bcf     PORTB,LCD_RS        ; Pone RS en modo comando.
            call    LcdEnviaByte        ; Envia el comando al LCD.
            return


102
;-----------------------------------------------------------
; Envia un byte a traves del bus de 4 hilos.
;-----------------------------------------------------------

107 LcdEnviaByte
            ; Guarda el valor que se envia.
            movwf   tmpLcdRegistro

            ; Envia los 4 bits mas significativos.
112
            bcf     PORTB,LCD_DB4
            bcf     PORTB,LCD_DB5
```

```
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        btfsc   tmpLcdRegistro,4
        bsf     PORTB,LCD_DB4
        btfsc   tmpLcdRegistro,5
        bsf     PORTB,LCD_DB5
        btfsc   tmpLcdRegistro,6
        bsf     PORTB,LCD_DB6
        btfsc   tmpLcdRegistro,7
        bsf     PORTB,LCD_DB7

        EN_STROBE                       ; Habilita y deshabilita el LCD.

        movlw   2
        call    msRetardo               ; Espera 2ms.

        ; Envia los 4 bits menos significativos.

        bcf     PORTB,LCD_DB4
        bcf     PORTB,LCD_DB5
        bcf     PORTB,LCD_DB6
        bcf     PORTB,LCD_DB7

        btfsc   tmpLcdRegistro,0
        bsf     PORTB,LCD_DB4
        btfsc   tmpLcdRegistro,1
        bsf     PORTB,LCD_DB5
        btfsc   tmpLcdRegistro,2
        bsf     PORTB,LCD_DB6
        btfsc   tmpLcdRegistro,3
        bsf     PORTB,LCD_DB7

        EN_STROBE

        movlw   2
        call    msRetardo

        return



;-----------------------------------------------------------
; Rutina de retardo.
;
; W = retardo en ms (con cristal de 4MHz)
;-----------------------------------------------------------

msRetardo:
        movwf   msRetardoContador+1
        clrf    msRetardoContador+0

        ; Loop interno de 1 ms.
msRetardoBucle:
        nop
        decfsz  msRetardoContador+0,F
```

```
        goto    msRetardoBucle
        nop

        decfsz  msRetardoContador+1,F
        goto    msRetardoBucle

        return


;-----------------------------------------------------------------
; Envia al LCD un byte en formato hexadecimal.
;-----------------------------------------------------------------

Pinta:  ; W contiene el byte que se enviara al display.

        movwf   caracteraux2
        andlw   b'11110000'     ; Toma la parte alta del byte.
        movwf   caracteraux
        bcf     STATUS,C
        rrf     caracteraux,f   ; Desplaza 4 posiciones a la derecha.
        rrf     caracteraux,f
        rrf     caracteraux,f
        rrf     caracteraux,f

        movlw   2
        movwf   contador

        goto    A_CARACTER      ; Convierte a caracter y se muestra en
                                ; hexadecimal.
mas_bits:
        movfw   caracteraux2
        andlw   b'00001111'     ; Toma la parte baja del byte.
        movwf   caracteraux
        goto    A_CARACTER      ; Convierte a caracter y lo muestra en
                                ; hexadecimal.
A_CARACTER
        movlw   0xA
        subwf   caracteraux,0
        btfsc   STATUS,C
        goto    mayor_igual_que_A
menor_que_A:
        movfw   caracteraux     ; 0 - 9.
        addlw   48
        goto    salta
mayor_igual_que_A:              ; A - F.
        movfw   caracteraux
        addlw   55

salta:  call    LcdEnviaDato    ; Envia el caracter al LCD.
        decfsz  contador
        goto    mas_bits

        return
```

A.3. Aplicación con puerto PARALELO

A.3.1. PARAL6.asm (Maestro)

Listing A.16: Programa

```
;*************************************************************
; Receptor de mando a distancia por infrarrojos que envia por *
; puerto paralelo y RS-232 los codigos de las senales IR que  *
; recibe.                                                     *
;                                                             *
;*************************************************************
; Carlos Arevalo Sillero                                      *
;                                                             *
;*************************************************************
; Se utiliza un Reloj de cuarzo de 4MHz.                      *
;                                                             *
;*************************************************************

        list      p=16F628, R=DEC
        ERRORLEVEL -305
        #include <p16F628.inc>

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC & _LVP_OFF


; Definicion de constantes.

#define                 PORT_II_ALTO PORTA
#define                 PORT_II_BAJO PORTB

CONTR_II_MA             equ  4
CONTR_II_ES             equ  3
Direccion_ceros         equ  0x38
Direccion_ceros_mas_uno equ  0x39
Direccion_unos          equ  0xA0
numero_de_bytes         equ  8
Dir_Senal               equ  0x120


; Definicion de variables.

        cblock h'20'
RECIBIR_REG             ; El bit 0 indica si esclavo recibio.
auxiliar_6              ; Variables auxiliares.
cuentabytes
cuenta
caracteraux             ; Guarda el caracter que se envia al esclavo.
auxiliar                ; Variable auxiliar.
medidacero_ref          ; Tolerancias a tener en cuenta en la recepcion.
medidauno_ref
FSR_TEMP1               ; Almacenan temporalmente los FSR.
FSR_TEMP2
w_temp                  ; Variable utilizada para guardar w.
status_temp             ; Variable utilizada para guardar el status.
nummedidascero          ; Numero de medidas cero.
```

```
MEDIDACERO                  ; Almacena las medidas cero.
MEDIDASCEROIGUALES          ; Numero de medidas cero iguales.
nummedidasuno               ; Numero de medidas uno.
MEDIDAUNO                   ; Guarda medidas uno.
MEDIDASUNOIGUALES           ; Numero de medidas uno iguales.
temp_medcero                ; Almacenan temporalmente las medidas
temp_meduno                 ; cero y uno.
MEDIDAUNOPEQUE              ; Guarda las medidas uno mas pequenas.
MEDIDACEROPEQUE             ; Almacena las medidas cero mas pequenas.
tiempos1                    ; Zona de almacenamiento de los ceros de la
                            ; senal IR.
        endc

        cblock h'A0'
tiempos2                    ; Zona de almacenamiento de los unos de la
       endc                 ; senal IR.


       cblock h'120'
byte1                       ; Bytes que contienen la senal IR.
byte2
byte3
byte4
byte5
byte6
byte7
byte8

        endc



        ORG     0x000               ; Vector de reset.
        goto    Main                ; Va al inicio del programa.

;-----------------------------------------------------------------
;               Controlador de interrupcion.
;-----------------------------------------------------------------

        ORG     0x004               ; Interrupcion: se ha recibido senal.
        movwf   w_temp              ; Guarda el contenido de W...
        movf    STATUS,w            ; ...y guarda el contenido de status.
        movwf   status_temp
        bcf     INTCON,GIE          ; Deshabilita interrupciones.

        call    Rec_RC5_RE80        ; Recibe los datos.

       ; Envia los bytes de la senal de IR por el puerto
       ; paralelo.

        call    ENVIA_SENAL_COMPLETA

        call    INI_RS232           ; Inicializa el USART.
        call    ENVIA_DATO          ; Envia los bytes IR por puerto RS232.
        bsf     STATUS,RP0          ; Banco 1.

        movlw   B'10000110'         ; Prescaler del Timer0 = 128, int. en
        movwf   OPTION_REG          ; flanco descendente.
```

```
        movlw   b'00001111'     ; Configura PORTA y PORTB.
        movwf   TRISA
        movlw   B'11111011'     ; RB0 y RB1 son entradas.
        movwf   TRISB           ; Configura RB0 y RB1 como entradas.
        bcf     STATUS,RP0      ; Banco 0.
        bcf     INTCON,INTF     ; Restaura el flag de interrupcion,
        bsf     INTCON,GIE      ; Habilita las interrupciones.
        movf    status_temp,w   ; restaura el contenido de status...
        movwf   STATUS
        swapf   w_temp,f
        swapf   w_temp,w        ; ...y restaura el contenido de w.
        retfie                  ; Retorna de la interrupcion.


;-----------------------------------------------------------------
; Envia los bytes de la senal de IR por el puerto serie RS232.
;-----------------------------------------------------------------

ENVIA_DATO:
        movlw   8               ; Recorre 8 bytes de datos.
        movwf   cuentabytes
        clrf    cuenta          ; Variable cuyo valor se suma al PCL
                                ; para obtener el byte que se envia.
        goto    byte_a_byte     ; Toma el byte que se envia.

_Toma_Caracter:
        incf    cuenta,F
        incf    cuenta,F

byte_a_byte:
        movfw   cuenta
        bsf     STATUS,RP1      ; Banco 2.
        call    BYTE_a_W        ; Suma cuenta al PCL y toma el byte.

continua_mas:
        bcf     STATUS,RP1      ; Banco 0.
        andlw   b'11110000'     ; Toma los 4 bits mas significativos.
        movwf   auxiliar
        bcf     STATUS,C
        rrf     auxiliar,f      ; Desplaza 4 posiciones a la derecha.
        rrf     auxiliar,f
        rrf     auxiliar,f
        rrf     auxiliar,f
        call    A_CARACTER      ; Lo pasa a hexadecimal.
        movfw   cuenta
        bsf     STATUS,RP1      ; Banco 2.
        call    BYTE_a_W

continua_mas2:
        bcf     STATUS,RP1
        andlw   b'00001111'     ; Toma los 4 bits menos significativos.
        movwf   auxiliar
        call    A_CARACTER      ; Lo pasa a hexadecimal.
        decfsz  cuentabytes
        goto    _Toma_Caracter
        return
```

```
A_CARACTER:
        movlw   0xA             ; Obtiene el caracter en formato hx.
        subwf   auxiliar,0
        btfsc   STATUS,C
        goto    mayor_igual_que_A

menor_que_A:
        movfw   auxiliar        ; 0 - 9.
        addlw   48
        goto    salta

mayor_igual_que_A:              ; A - F.
        movfw   auxiliar
        addlw   55
salta:

        call    send            ; Envia caracter via RS232.
        return


;-------------------------------------------------------------------
; Elige el byte correspondiente que se copia en W, segun el valor
; de PCL.
;-------------------------------------------------------------------

BYTE_a_W:
        addwf   PCL
        movfw   byte1
        return
        movfw   byte2
        return
        movfw   byte3
        return
        movfw   byte4
        return
        movfw   byte5
        return
        movfw   byte6
        return
        movfw   byte7
        return
        movfw   byte8
        return


;-----------------------------------------------------------------------
;               Programa principal
;-----------------------------------------------------------------------

Main
        movlw   0x07            ; Pone los comparadores en off
        movwf   CMCON           ; permitiendo el uso de los pines
                                ; I/O.
        bsf     STATUS,RP0      ; Banco 1.
        movlw   B'10000110'     ; int. en flanco descendente.
        movwf   OPTION_REG
```

```
        movlw   b'00001111'     ; Configura el puerto A y B.
        movwf   TRISA
        movlw   B'11111011'
        movwf   TRISB           ; Configura RB0 y RB1 como entradas.
        bcf     STATUS,RP0      ; Banco 0.
        bsf     INTCON,INTE
        bsf     INTCON,GIE      ; Habilita las interrupciones.

BUCLE_PRINCIPAL:

        goto    BUCLE_PRINCIPAL

        INCLUDE "recibe.asm"
        INCLUDE "rs232.asm"
        INCLUDE "paralelo.asm"

        END
```

A.3.2. paralelo.asm

Listing A.17: Programa

```
;-----------------------------------------------------------------
; Envia un byte a traves del puerto paralelo al dispositivo esclavo.
;-----------------------------------------------------------------

ENVIA_PARALELO_BYTES:

        ; Prepara el TMR0
        bcf     INTCON,T0IF         ; Borra el bit de interrupcion.
        clrf    TMR0                ; Borra el temporizador.
        bsf     STATUS,RP0          ; Banco 1.
        bcf     OPTION_REG,PSA      ; Preescala al TMR0.
        bcf     OPTION_REG,PS0      ; Preescala 1:64.
        bcf     OPTION_REG,PS1      ; Temporizacion = ((255 * 64)+2)us.
        bsf     OPTION_REG,PS2
        bcf     STATUS,RP0          ; Banco 0.
        bsf     STATUS,RP0
        movlw   b'00000000'         ; PORTA todo salidas.
        movwf   TRISA
        bcf     STATUS,RP0
        bcf     PORTA,4             ; Pone a cero RB3.

no_pasa:
        btfsc   INTCON,T0IF
        goto    Esclavo_No_Resp
        btfsc   PORTB,3
        goto    no_pasa
                                    ; RA0-RA3 Y RB7-RB4 pasan a ser salidas.
        bsf     STATUS,RP0          ; Banco 1.
```

```
        movlw   b'00000000'         ; Todo salidas.
        movwf   TRISA               ; Configura PORTA como salidas.
        movlw   B'00001011'
        movwf   TRISB               ; Configura RB0 y RB1 como entradas.

        bcf     STATUS,RP0          ; Banco 0.

        ; Pone el dato en el bus.
        movfw   caracteraux         ; Contiene el caracter que se envia.
        call    ENVIA_DATOII        ; Envia el dato por puerto paralelo.
        bsf     PORTA,4             ; Indica que el dato esta listo.

espera_espera:
        btfsc   INTCON,T0IF         ; Comprueba si acabo el TMR0.
                                    ; Si TMR0 termino de contar no se
                                    ; obtuvo respuesta del esclavo.
        goto    Esclavo_No_Resp     ; El esclavo no respondio.
        btfss   PORTB,3             ; Espera que el esclavo lea dato.
        goto    espera_espera

        bsf     APRENDIZAJE_REG,0   ; Esclavo recibio el dato.

Esclavo_No_Resp:
        bsf     PORTA,4             ; Pone a uno RB3 para no producir
                                    ; una nueva int. al esclavo.

        ; RA0-RA3 Y RB7-RB4 pasan a ser entradas.
        ; Aqui el esclavo leyo el dato.
        bsf     STATUS,RP0          ; Banco 1.
        movlw   b'00001111'         ; Parte alta de PORTA salidas.
        movwf   TRISA
        movlw   B'11111011'
        movwf   TRISB               ; RB0 y RB1 entradas.
        bcf     STATUS,RP0          ; Banco 0.

        btfss   APRENDIZAJE_REG,0   ; Comprueba si esclavo recibio.
        goto    ENVIA_PARALELO_BYTES; Si no lo recibio lo vuelve a
        bcf     APRENDIZAJE_REG,0   ; enviar.

        return


;-----------------------------------------------------------------
; Envia 8 bytes a traves del puerto paralelo al dispositivo esclavo.
;-----------------------------------------------------------------

ENVIA_SENAL_COMPLETA:

        movlw   0xFF                ; Empiece de linea. Se indica con
                                    ; 0xFF.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia codigo de inicio de linea.
        bsf     STATUS,RP1          ; Banco 2.
        movfw   byte1
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia byte1.

        bsf     STATUS,RP1          ; Banco 2.
```

```
        movfw   byte2
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia el byte2.

        bsf     STATUS,RP1          ; Banco 2.
        movfw   byte3
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia byte3.

        bsf     STATUS,RP1          ; Banco 2.
        movfw   byte4
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia byte4.

        bsf     STATUS,RP1          ; Banco 2.
        movfw   byte5
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia byte5.

        bsf     STATUS,RP1          ; Banco 2.
        movfw   byte6
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia byte6.

        bsf     STATUS,RP1          ; Banco 2.
        movfw   byte7
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia byte7.

        bsf     STATUS,RP1          ; Banco 2.
        movfw   byte8
        bcf     STATUS,RP1          ; Banco 0.
        movwf   caracteraux
        call    ENVIA_PARALELO_BYTES; Envia byte8.
        return


;-----------------------------------------------------------
; Pone un byte en el bus paralelo.
; La parte baja del dato se coloca en las patillas RB4-RB7
; del puerto B mientras que la parte alta del dato se manda a
; traves de la parte baja del puerto A: RA0-RA3.
;-----------------------------------------------------------

ENVIA_DATOII:
        movwf   caracteraux         ; Byte que se envia por el puerto.
        movwf   cuenta              ; Se hace otra copia del byte.
        bcf     STATUS,C            ; Borra el carry.
        rrf     cuenta,f            ; Desplaza cuatro bits a la derecha para
        rrf     cuenta,f            ; colocar la parte alta del dato sobre
        rrf     cuenta,f            ; la parte baja de PORTA.
        rrf     cuenta,f
```

```
150
        btfss   PORTA,4             ; Mantiene el valor de la senal de
        bcf     cuenta,4            ; control del puerto paralelo.
        bsf     cuenta,4

155     movfw   cuenta              ; Coloca la parte alta del dato en
        movwf   PORTA               ; el bus.

        movfw   caracteraux         ; Copia el byte que se envia.
        movwf   cuenta
160     bcf     STATUS,C            ; Borra el carry.
        rlf     cuenta,f            ; Rota a la izquierda cuatro
        rlf     cuenta,f            ; posiciones.
        rlf     cuenta,f
        rlf     cuenta,f
165     movfw   cuenta              ; Coloca la parte baja del dato
        movwf   PORTB               ; sobre la parte alta de PORTB.

        return
```

A.3.3. recibe.asm

Ver sección A.1.2.

A.3.4. rs232.asm

Ver sección A.1.4.

A.3.5. PARAL8.asm (Esclavo)

Listing A.18: Programa

```
;*****************************************************************
2 ; Programa esclavo que recibe a traves de un puerto paralelo    *
; los bytes de la senal IR que recibe el sistema maestro.         *
;                                                                 *
;                                                                 *
;*****************************************************************
7 ; Carlos Arevalo Sillero                                         *
;                                                                 *
;*****************************************************************
; Se utiliza un Reloj de cuarzo de 4MHz.                          *
;                                                                 *
12 ;*****************************************************************


        LIST P=16F84A, R=DEC
17      ERRORLEVEL -305,-302
        INCLUDE "P16F84A.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF

22
```

```
; Definicion de constantes.

#define                PORT_II_ALTO PORTA
#define                PORT_II_BAJO PORTB

CONTR_II_MA            equ  0
CONTR_II_ES            equ  1

; Lineas de control del LCD.

LCD_RS                 equ  2                    ; Comando/dato.
LCD_E                  equ  3                    ; Habilitacion del LCD.

; Bus de datos del LCD.

LCD_DB4                equ  4
LCD_DB5                equ  5
LCD_DB6                equ  6
LCD_DB7                equ  7


; Habilita el LCD.

SET_EN            MACRO
                  bsf  PORTB,LCD_E
                  ENDM

; Deshabilita el LCD.

CLEAR_EN          MACRO
                  bcf  PORTB,LCD_E
                  ENDM

; Habilita el LCD 1us.

EN_STROBE         MACRO
                  SET_EN
                  nop
                  CLEAR_EN
                  ENDM



          ORG     0x0C

; Definicion de variables.

W_TEMP            res     1        ; Variables para salvar el contexto en
STATUS_TEMP       res     1        ; las interrupciones.
tmpLcdRegistro    res     2        ; Varible auxiliar para configurar el LCD.
msRetardoContador  res    2        ; Contador.
caracteraux       res     1        ; Almacena de forma temporal un byte que se
                                   ; se pasa a hx.
AUX_II            res     1        ; Registro auxiliar para el puerto paralelo.
caracteraux2      res     1        ; Almacena un caracter.
contador          res     1
byte1             res     1        ; Bytes que se muestran en el LCD. Almacenan
```

```
82 byte2          res     1        ; la senal IR.
   byte3          res     1
   byte4          res     1
   byte5          res     1
   byte6          res     1
87 byte7          res     1
   byte8          res     1
   MUESTRA_LCD    res     1        ; Variable auxiliar.


92

          ORG     0
          goto    start

97        ORG     4

          ; Salva el contexto.

          MOVWF   W_TEMP           ; W_TEMP=W.
102       SWAPF   STATUS,W         ; W=swap(STATUS).
          MOVWF   STATUS_TEMP      ; STATUS_TEMP=W.

          bsf     STATUS,RP0
          bcf     INTCON,GIE       ; Deshabilita interrupciones
107       bcf     STATUS,RP0


                  ; Codigo del puerto paralelo

112       ; Prepara el TMR0
          bcf      INTCON,T0IF     ; Borra el bit de interrupcion.
          clrf     TMR0            ; Borra el temporizador.
          bsf      STATUS,RP0      ; Banco 1.
          bcf      OPTION_REG,PSA  ; Preescala al TMR0.
117       bsf      OPTION_REG,PS0  ; Preescala 1:4.
          bcf      OPTION_REG,PS1  ; Temporizacion = ((255 * 4)+2)us.
          bcf      OPTION_REG,PS2
          bcf      STATUS,RP0      ; Banco 0.

122


          ;**********************INICIO DE TRAMA******************

127       bcf      PORTB,1         ; Pone RB1 a cero, esta listo para
          bsf      STATUS,RP0      ; recibir.
          bcf      TRISB,1         ; RB1 es salida
          bcf      STATUS,RP0

132       bcf      PORTB,1         ; RB1 a cero.
   XX:    btfsc    INTCON,T0IF     ; Comprueba si paso demasiado tiempo
          goto     Maestro_No_Resp ; desde que se espera otra respuesta
          btfss    PORTB,0         ; del maestro. Comprueba la entrada.
          goto     XX              ; Espera que este a '1'.
137
          ; Lee el dato que ha puesto el maestro en el bus.
          call     RECIBE_II
          movwf    byte1           ; Guarda el dato.
```

```
        bsf      PORTB,1          ; RB1 a 1. Indica que tomo el dato.

        ;****************************BYTE1************************
NO_CERO11:
        btfsc    INTCON,TOIF      ; Comprueba si ha pasado mucho
        goto     Maestro_No_Resp  ; tiempo desde la ultima indicacion
        btfsc    PORTB,0          ; del maestro. Comprueba si hay
        goto     NO_CERO11        ; indicacion del maestro.

        bcf      PORTB,1          ; RB1 a cero. Indica al maestro
XX11:   btfsc    INTCON,TOIF      ; que esta listo.
        goto     Maestro_No_Resp  ; Acabo el tiempo y el maestro
        btfss    PORTB,0          ; no respondio. Comprueba si hay
        goto     XX11             ; respuesta del maestro.

        ; Lee el dato que hay en el puerto paralelo.
        call     RECIBE_II
        movwf    byte1            ; Guarda el byte.
        bsf      PORTB,1          ; RB1 a 1.


        ; Se repite la secuencia anterior para recibir el resto de
        ; bytes por el puerto paralelo.

      ;****************************BYTE2************************
NO_CERO1:
        btfsc    INTCON,TOIF
        goto     Maestro_No_Resp
        btfsc    PORTB,0
        goto     NO_CERO1

        bcf      PORTB,1          ; RB1 a cero.
XX1:    btfsc    INTCON,TOIF
        goto     Maestro_No_Resp
        btfss    PORTB,0
        goto     XX1

        ; Lee el dato que hay en el puerto paralelo.
        call     RECIBE_II
        movwf    byte2
        bsf      PORTB,1          ; RB1 a 1.

        ;****************************BYTE3************************
NO_CERO2:
        btfsc    INTCON,TOIF
        goto     Maestro_No_Resp

        btfsc    PORTB,0
        goto     NO_CERO2


        bcf      PORTB,1          ; RB1 a cero.
XX2:    btfsc    INTCON,TOIF
        goto     Maestro_No_Resp
        btfss    PORTB,0
        goto     XX2

        ; Lee el dato que hay en el puerto paralelo.
```

```
            call    RECIBE_II
            movwf   byte3
            bsf     PORTB,1             ; RB1 a 1.

            ;***************************BYTE4***********************
NO_CERO3:
            btfsc   INTCON,T0IF
            goto    Maestro_No_Resp

            btfsc   PORTB,0
            goto    NO_CERO3

            bcf     PORTB,1             ; RB1 a cero.
XX3:        btfsc   INTCON,T0IF
            goto    Maestro_No_Resp
            btfss   PORTB,0
            goto    XX3

            ; Lee el dato que hay en el puerto paralelo.
            call    RECIBE_II
            movwf   byte4
            bsf     PORTB,1             ; RB1 a 1.

            ;***************************BYTE5***********************
NO_CERO4:
            btfsc   INTCON,T0IF
            goto    Maestro_No_Resp

            btfsc   PORTB,0
            goto    NO_CERO4


            bcf     PORTB,1             ; RB1 a cero.
XX4:        btfsc   INTCON,T0IF
            goto    Maestro_No_Resp
            btfss   PORTB,0
            goto    XX4

            ; Lee el dato que hay en el puerto paralelo.
            call    RECIBE_II
            movwf   byte5
            bsf     PORTB,1             ; RB1 a 1.

            ;***************************BYTE6***********************
NO_CERO5:
            btfsc   INTCON,T0IF
            goto    Maestro_No_Resp
            btfsc   PORTB,0
            goto    NO_CERO5


            bcf     PORTB,1             ; RB1 a cero.
XX5:        btfsc   INTCON,T0IF
            goto    Maestro_No_Resp
            btfss   PORTB,0
            goto    XX5

            ; Lee el dato que hay en el puerto paralelo.
            call    RECIBE_II
```

```
          movwf   byte6
          bsf     PORTB,1            ; RB1 a 1.

         ;****************************BYTE7************************
NO_CERO6:
          btfsc   INTCON,T0IF
          goto    Maestro_No_Resp
          btfsc   PORTB,0
          goto    NO_CERO6


          bcf     PORTB,1            ; RB1 a cero.
XX6:      btfsc   INTCON,T0IF
          goto    Maestro_No_Resp
          btfss   PORTB,0
          goto    XX6

          ; Lee el dato que hay en el puerto paralelo.
          call    RECIBE_II
          movwf   byte7
          bsf     PORTB,1            ; RB1 a 1.

         ;****************************BYTE8************************
NO_CERO7:
           btfsc  INTCON,T0IF
           goto   Maestro_No_Resp

           btfsc  PORTB,0
           goto   NO_CERO7


           bcf    PORTB,1            ; RB1 a cero.
XX7:       btfsc  INTCON,T0IF
           goto   Maestro_No_Resp
           btfss  PORTB,0
           goto   XX7

           ; AQUI LEE EL DATO QUE HA PUESTO EL MAESTRO
           call   RECIBE_II
           movwf  byte8
           bsf    PORTB,1            ; RB1 a 1.
           bsf    MUESTRA_LCD,0
           bsf    STATUS,RP0
           bsf    TRISB,1            ; RB1 vuelve a ser entrada.
           bcf    STATUS,RP0

           ; Se prepara la comunicacion con el display

Maestro_No_Resp:
           bsf    PORTB,1            ; RB1 a 1.
           bsf    STATUS,RP0
           bsf    TRISB,1            ; RB1 vuelve a ser entrada.
           bcf    STATUS,RP0

           ; Fin de codigo del puerto paralelo.

           bsf    STATUS,RP0         ; Prepara los puertos para la
           movlw  b'11110111'        ; comunicacion con el LCD.
           movwf  TRISB
```

```
            movlw   b'11111111'
            movwf   TRISA
            movlw   b'10000000'        ; Produce interrupcion por flanco de
                                       ; bajada de RB0.
            movwf   OPTION_REG
            bcf     STATUS,RP0


            bsf     STATUS,RP0         ; Banco 1.
            movlw   b'10010000'        ; GIE(7)=1 RBIE(3)=1 RBIF(0)
            movwf   INTCON             ; Definir interrupciones habilitadas
            bcf     STATUS,RP0         ; Banco 0.

           ; Recupera el contexto.

           SWAPF    STATUS_TEMP,W      ; W=swap(STATUS_TEMP).
           MOVWF    STATUS             ; STATUS=W.
           SWAPF    W_TEMP,F           ; W_TEMP=swap(W_TEMP).
           SWAPF    W_TEMP,W           ; W=SWAP(W_TEMP).

           retfie



;-----------------------------------------------------------
; Subrrutina que recibe los datos por puerto paralelo
;-----------------------------------------------------------

RECIBE_II:


           ; La parte baja del dato que se recibe es la parte alta
           ; de PORTB y la parte alta del dato es la parte baja de
           ; PORTA.

           movfw    PORTA              ; Copia el contenido del puerto A.
           movwf    caracteraux
           bcf      STATUS,C           ; Borra el carry.

           rlf      caracteraux,f      ; Desplaza a la izquierda porque
           rlf      caracteraux,f      ; es la parte alta del dato.
           rlf      caracteraux,f
           rlf      caracteraux,f

           movfw    caracteraux        ; Copia el dato de PORTB.
           andlw    b'11110000'        ; Borra la parte baja.
           movwf    caracteraux        ; Se copia el dato.
           movfw    PORTB
           movwf    AUX_II

           rrf      AUX_II,f           ; Desplaza a la derecha cuatro bits,
           rrf      AUX_II,f           ; para colocar la parte baja del byte
           rrf      AUX_II,f           ; en su sitio.
           rrf      AUX_II,f

           movfw    AUX_II             ; El dato recibido pasa a w.
           andlw    b'00001111'
           addwf    caracteraux,f
```

```
377         movfw   caracteraux
            movwf   AUX_II
            return


382
    start   ; Comienzo del programa principal.


    comienzo
387
            bsf      PORTB,1            ; RB1 a 1.

            bsf     STATUS,RP0          ; Banco 1.
            movlw   b'11110111'         ; Configura los puertos.
392         movwf   TRISB
            movlw   b'11111111'
            movwf   TRISA
            movlw   b'10000000'         ; Produce interrupcion por flanco de
                                        ; bajada de RB0.
397         movwf   OPTION_REG
            bcf     STATUS,RP0


            bsf     STATUS,RP0          ; Banco 1.
402         movlw   b'10010000'         ; GIE(7)=1 RBIE(3)=1 RBIF(0)
            movwf   INTCON              ; Definir interrupciones habilitadas.
            bcf     STATUS,RP0          ; Banco 0.

            bcf     MUESTRA_LCD,0       ; No hay senal que mostrar en el LCD.
407

    bucle:
            btfss   MUESTRA_LCD,0       ; Comprueba si hay senal que mostrar
            goto    bucle               ; en el LCD.
412         bcf     MUESTRA_LCD,0       ; Recogio senal para mostrar en el LCD.
            bsf     STATUS,RP0          ; Banco 1.
            movlw   00000011B           ; Configura el puerto B.
            movwf   TRISB
            bcf     STATUS,RP0          ; Banco 0.
417
            ;Inicializa el LCD.

            call    LcdInic

422         ;Coloca cursor en posicion 0,0.

            movlw   0x00
            call    LcdColoca

427 continua:
            movfw   byte1                ; Muestra a traves del LCD todos
            call    Pinta                ; los bytes de la senal.
            movfw   byte2
            call    Pinta
432         movfw   byte3
            call    Pinta
            movfw   byte4
            call    Pinta
```

```
            movfw   byte5
437         call    Pinta
            movfw   byte6
            call    Pinta
            movfw   byte7
            call    Pinta
442         movfw   byte8
            call    Pinta


    SALTA_PRIMER_VALOR:
447         bsf     PORTB,1             ; RB1 a 1.
            bsf     STATUS,RP0
            bsf     TRISB,1             ; RB1 vuelve a ser entrada.
            bcf     STATUS,RP0
            bcf     PORTB,LCD_E         ; Deshabilita el display.
452
            bsf     STATUS,RP0          ; Configura los puertos para
            movlw   b'11110111'         ; tener activo el puerto paralelo.
            movwf   TRISB
            movlw   b'11111111'
457         movwf   TRISA

            bcf     STATUS,RP0

            goto    bucle
462
            INCLUDE "lcdesc.asm"


            END
```

A.3.6. lcdesc.asm

Ver sección A.2.8.

Apéndice B

Placas PCB

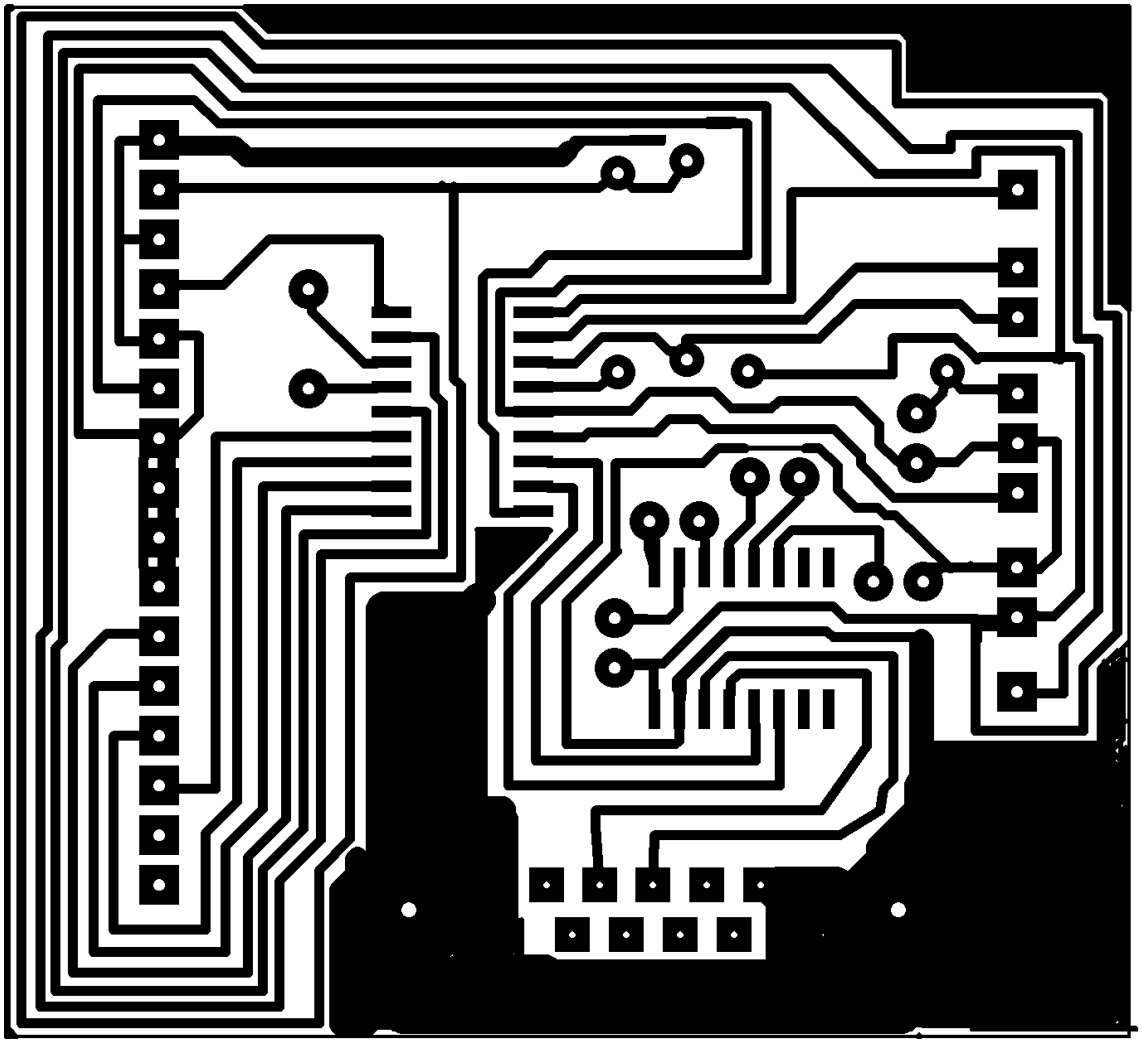


Figura B.1: PCB TIPO 1

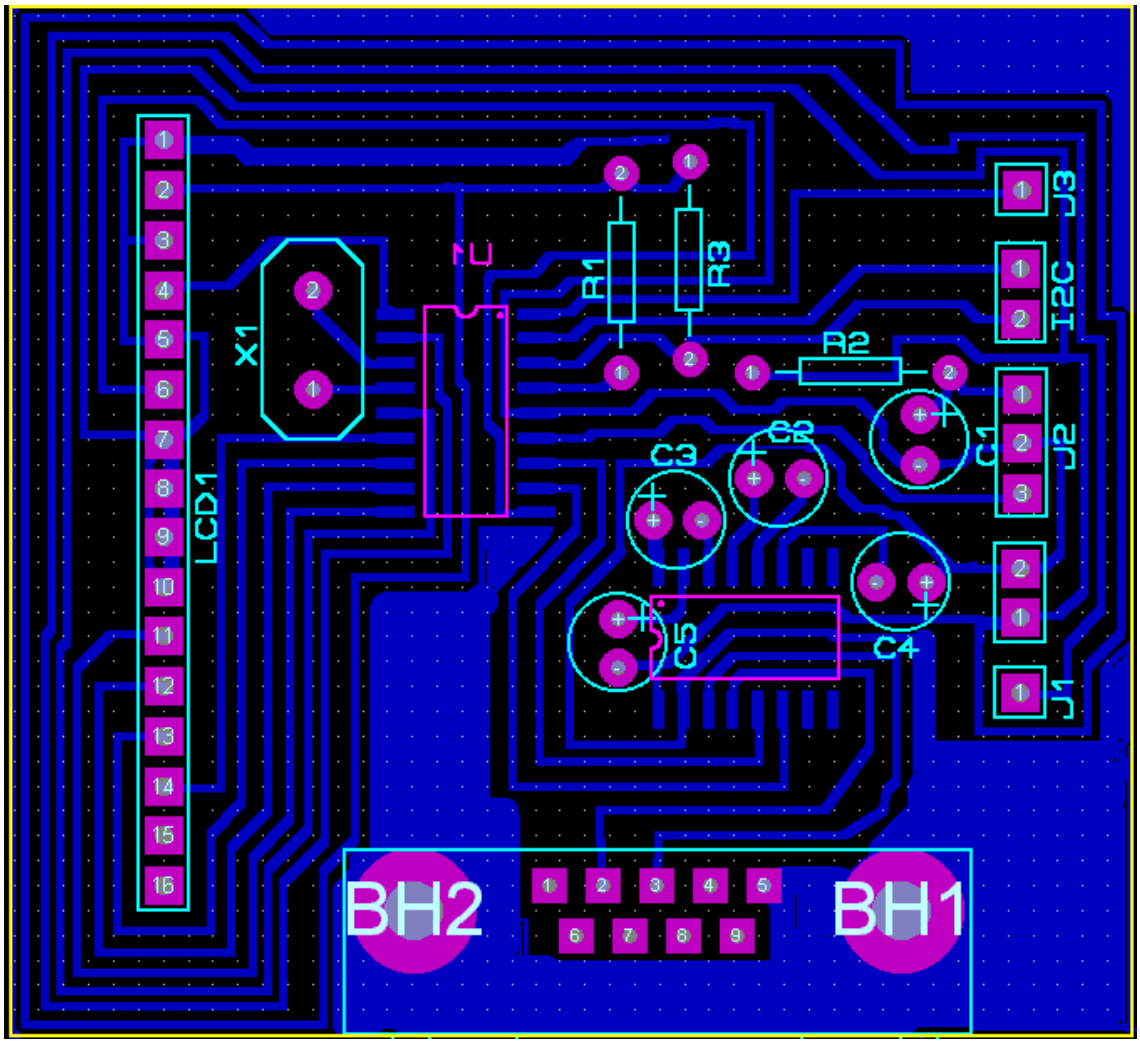


Figura B.2: PCB TIPO 1 completa

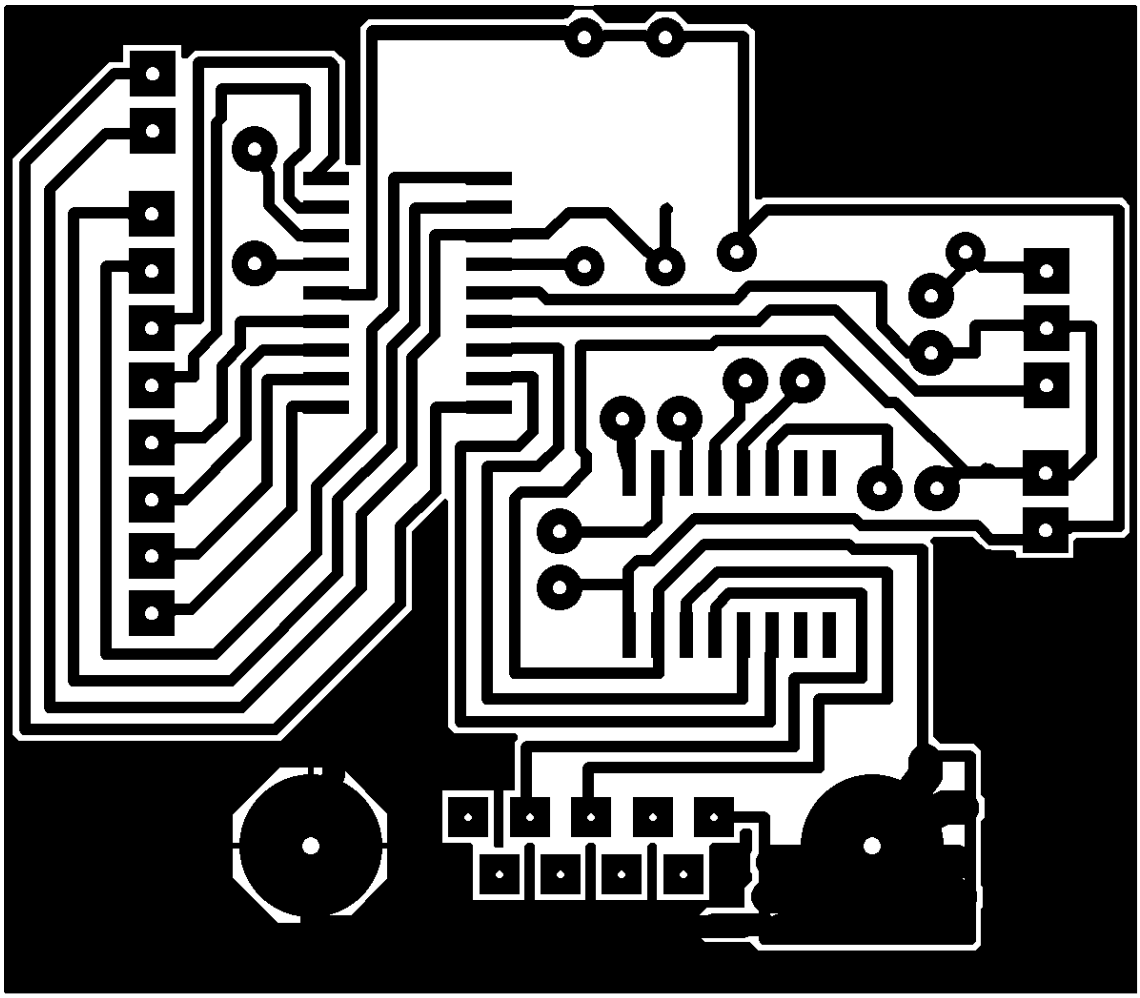


Figura B.3: PCB TIPO 2

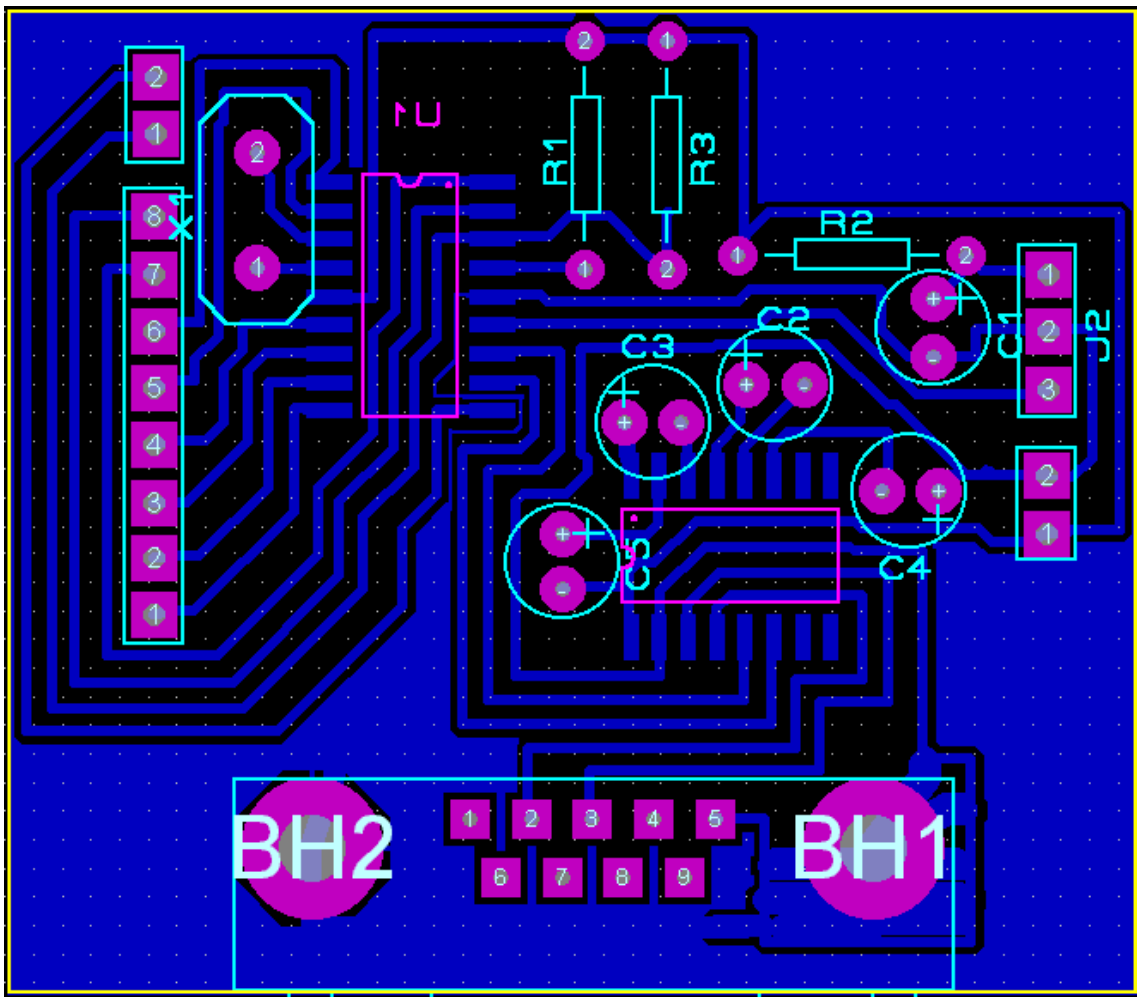


Figura B.4: PCB TIPO 2 completa

Bibliografía

[1] Capa física Interfaz RS-232.
http://ceres.ugr.es/~alumnos/redrs232/fisica.htm.

[2] Connecting and using an lcd module.
http://homepage.ntlworld.com/matthew.rowe/micros/virbook/lcd.htm.

[3] Conversor rs232 a ttl sin max232.
http://www.pablin.com.ar/electron/circuito/mc/ttl232/.

[4] Estándar de comunicaciones rs-232c.
http://www.euskalnet.net/shizuka/rs232.htm.

[5] Field programmable rc5/sony infrared remote receiver/decoder.
http://home1.stofanet.dk/hvaba/fprc5rx/.

[6] Hardware of the pic16f628.
http://www.interq.or.jp/japan/se-inoue/e_pic8.htm.

[7] Infrared remote control.
http://www.ustr.net/infrared/infrared1.shtml.

[8] Infrared remote control technology.
http://www.epanorama.net/links/irremote.html.

[9] Infrared remote controled pc.
http://www.technology.niagarac.on.ca/students/l/blachapelle/.

[10] IR-Fernbedienung der RC-5 Code.
http://www.sprut.de/electronic/ir/rc5.htm.

[11] Ir remote.
http://snakecat.virtualave.net/remote/surfe.html.

[12] Lcd interfacing reference page.
http://www.myke.com/lcd.htm.

[13] Lcd modules.
http://www.winpicprog.co.uk/pic_tutorial3.htm.

[14] Pic.
http://usuarios.lycos.es/sfriswolker/pic/picindex.htm.

[15] Pic ir decoder.
http://www.xs4all.nl/~sbp/projects/ircontrol/picir/hardware.htm.

[16] PIC to RS232 6 ways for a reliable hardware interface.
http://surducan.netfirms.com/RS232.html.

[17] Puerto serie.
http://www.ctv.es/pckits/tpseriee.html.

[18] SC546 Project An analytical study of IR signals used by a SONY remote control.
http://scv.bu.edu/GC/shammi/ir/.

[19] Universal infrared receiver.
http://fly.cc.fer.hr/~mozgic/UIR/.

[20] X infrared remote control.
http://www.linuxtoys.org/xirrc/xirrc.html.

[21] Gabriele Bellini. A universal infrared remote controller for your sony md walkman.
http://web.tiscali.it/minidisc/.

[22] Juan Menéndez Blanco. Mandos a distancia por infrarrojos.
http://www.fortunecity.es/arcoiris/tarot/572/mandos.html.

[23] Davide Bucci. Il protocollo seriale i2c con un pic.
http://www.geocities.com/leibowitz.geo/pic_i2c.html.

[24] Moisés Cambra. Control remoto de pc por infrarrojos.
http://www.geocities.com/CapeCanaveral/Lab/1475/remotespa.html.

[25] Bret Connell, Chris Craig, and Dana Smith. A remote control study.
http://www.ee.washington.edu/conselec/A95/projects/pierreg/main.htm.

[26] Eduardo Coquet. El bus i2c.
http://www.comunidadelectronicos.com/articulos/i2c.htm.

[27] Datasheet. *IS1U60*.
http://www.chipdocs.com/datasheets/datasheet-pdf/Sharp/IS1U60.html.

[28] Datasheet. *MAX232*.
http://www.alldatasheet.co.kr/datasheet-pdf/view/MAXIM/MAX232.html.

[29] Datasheet. *PIC16F627/8*.
http://ww1.microchip.com/downloads/en/DeviceDoc/40300c.pdf.

[30] Javier de Lope Asiaín. Robots móviles Emisión y recepción de infrarrojos.
http://jdlope.tripod.com/infra.html.

[31] Luis Germán González. Analizador de señales infrarrojas con pc.
http://www.comunidadelectronicos.com/proyectos/infrarrojos.htm.

[32] Martha Alicia Vega González. Bus rs-232.
http://www.elet.itchihuahua.edu.mx/academia/jnevarez/RS232/Rs-232.htm.

[33] Chuck McManis. Driving the lcd on the lab-x3.
http://www.mcmanis.com/chuck/robotics/projects/lab-x3/lcd0.html.

[34] University of ULM. Universal ir-receiver for serial interface.
http://mikro.e-technik.uni-ulm.de/research/urcr.html.

[35] Jim Pollock. Modulated ir signals.
http://us.geocities.com/jpollock_2000/infrared.htm.

[36] Juergen Putzger. Decoding ir remote controls.
http://www.ee.washington.edu/circuit_archive/text/ir_decode.txt.

[37] Zdenek Sara. Modulations and protocols for infra-red transmission.
http://www.hw-server.com/docs/IrDA_standards_and_protocols.html.

Apéndice C

Datasheets

IS1U60/IS1U60L

Sensors with 1-Package Design of Remote Control Detecting Functions owing to OPIC

■ Features

1. 1-package design owing to adoption of OPIC
2. Compact
   (Volume : About 1/8 compared with **GP1U58X**)
3. B.P.F. (Band Pass Frequency) : (TYP. 38kHz)
4. Aspherical lens

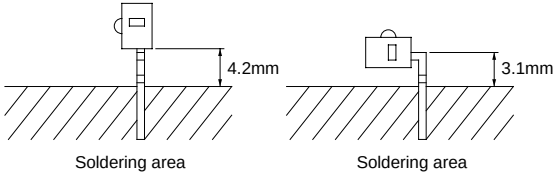
■ Applications

1. Audio equipment
2. Cameras

■ Absolute Maximum Ratings (Ta=25˚C)

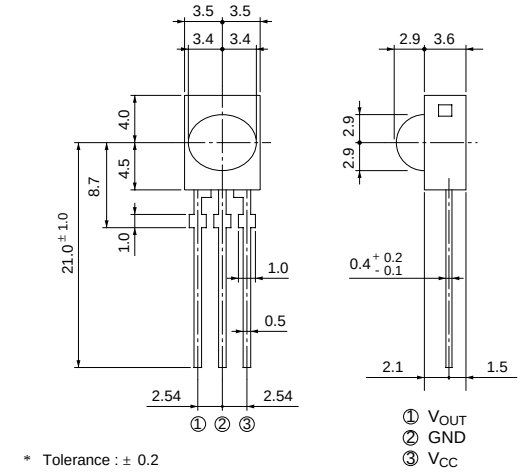
Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	0 to 6.0	V
*1Operating temperature	T_{opr}	- 10 to +60	˚C
Storage temperature	T_{stg}	- 20 to +70	˚C
*2Soldering temperature	T_{sol}	260	˚C

*1 No dew condensation is allowed.
*2 For 5 seconds

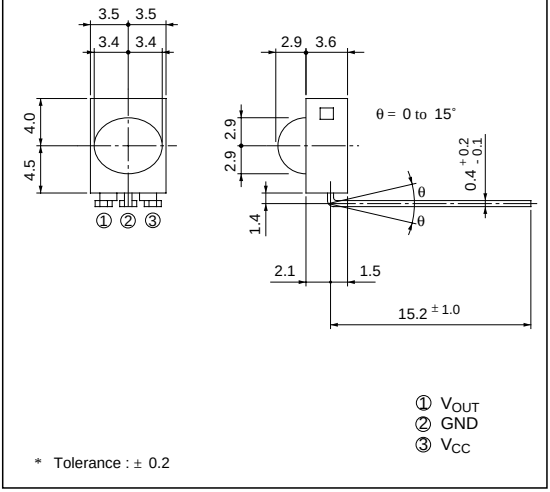


■ Outline Dimensions (Unit : mm)

IS1U60



IS1U60L



* "OPIC" (Optical IC) is a trademark of the SHARP Corporation.
An OPIC consists of a light-detecting element and signal-processing circuit integrated onto a single chip.

■ Recommended Operating Conditions

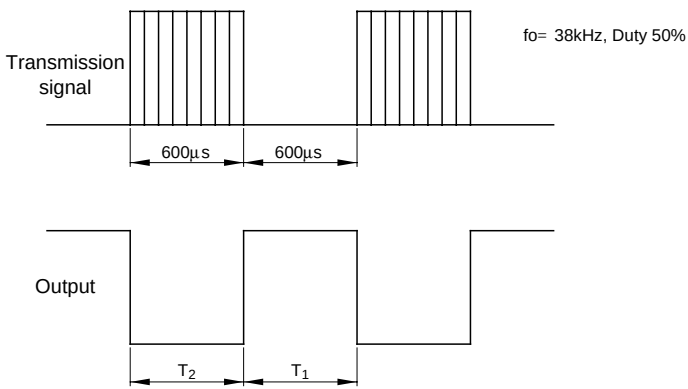
Parameter	Symbol	Recommended operating conditions	Unit
Operating supply voltage	V_{CC}	4.7 to 5.3	V

■ Electrical Characteristics

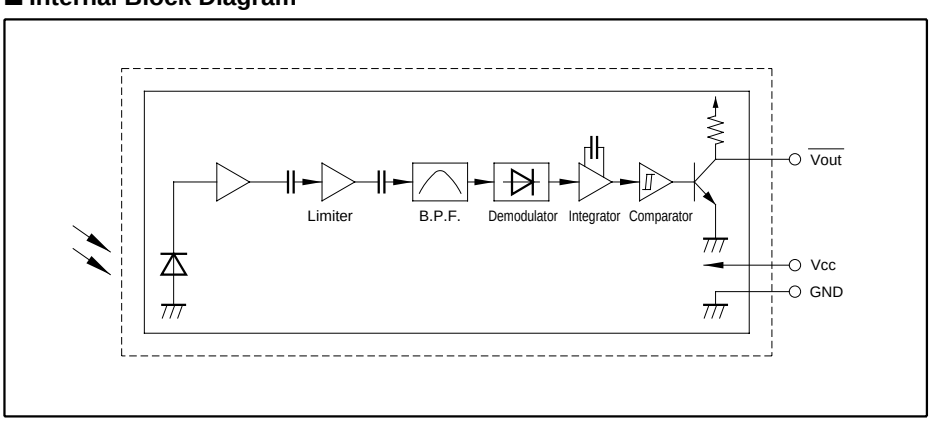
(Ta=25˚C, V_{CC} =+5V)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Dissipation current	I_{CC}	No input light	-	2.8	4.5	mA
High level output voltage	V_{OH}	*3, Output terminal OPEN	V_{CC}- 0.2	-	-	V
Low level output voltage	V_{OL}	*3, *4	-	0.45	0.6	V
High level pulse width	T_1	*3	400	-	800	μs
Low level pulse width	T_2		400	-	800	μs
B.P.F. center frequency	f_O		-	38	-	kHz
Linear ultimate distance	L	ϕ , θ = 0˚, E_e < 10 lx	5.0	-	-	m
Linear ultimate distance	L_1	ϕ = ± 30˚ (θ = 0˚) θ = ± 15˚ (ϕ = 0˚) E_e < 10 lx	3.0	-	-	m

*3 The burst wave as shown in the following figure shall be transmitted.
*4 Pull-up resistance : 2.2kΩ
*5 By SHARP transmitter



■ Internal Block Diagram



■ Performance

Using the transmitter shown in Fig. 1, the output signal of the light detecting unit is good enough to meet the following items in the standard optical system in Fig. 2.

(1) Linear reception distance characteristics

When L=0.2 to 5 m, Ee < 10 lx (*4) and ϕ =0° in Fig. 2, the output signal shall meet the electrical characteristics in the attached list.

(2) Sensitivity angle reception distance characteristics

When L=0.2 to 3 m, Ee < 10 lx (*4) and ϕ<= 30 ° in the direction X and θ =0° in the direction Y in Fig. 2, the output signal shall meet the electrical characteristics in the attached list Further, the electrical characteristics shall be met when L=0.2 to 5 m, Ee < 10 lx (*4) and ϕ =0° in the direction X and θ<= 15° in the direction Y.

*4 It refers to detector face illuminance.

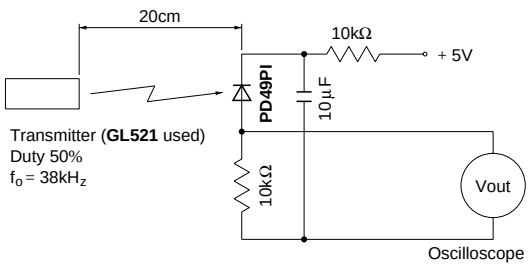


Fig. 1 Transmitter

In the above figure, the transmitter should be set so that the output Vout can be $40mV_{P-P}$.

However, the **PD49PI** to be used here should be of the short-circuit current I_{SC}=2.6μ A at Ev=100 lx.

(Ev is an illuminance by CIE standard light source A (tungsten lamp).)

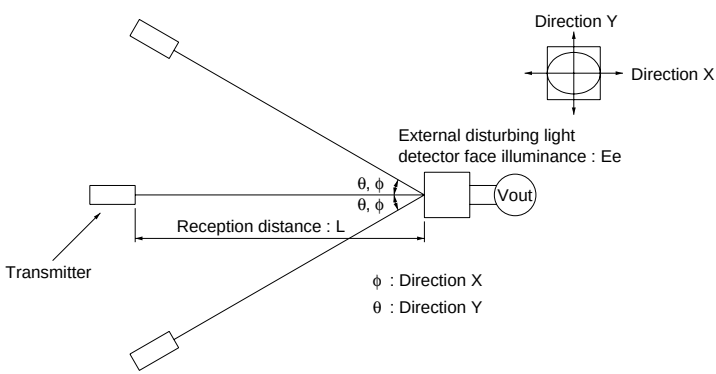


Fig. 2 Standard optical system

Fig. 1 B.P.F. Frequency Characteristics (TYP.)

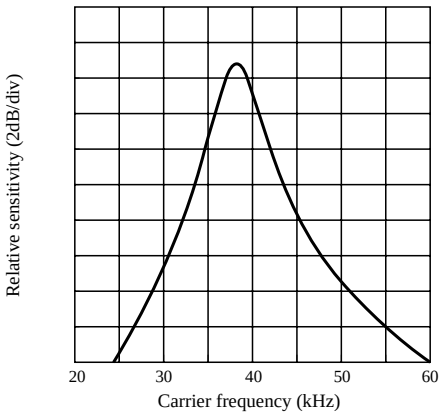


Fig. 2 Sensitivity Angle (Direction X) Characteristics (TYP.) for Reference

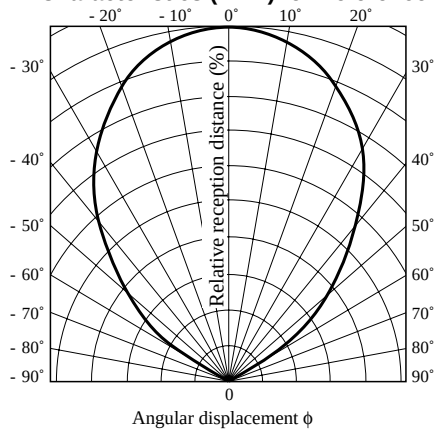


Fig. 3 Sensitivity Angle (Direction Y) Characteristics (TYP.) for Reference

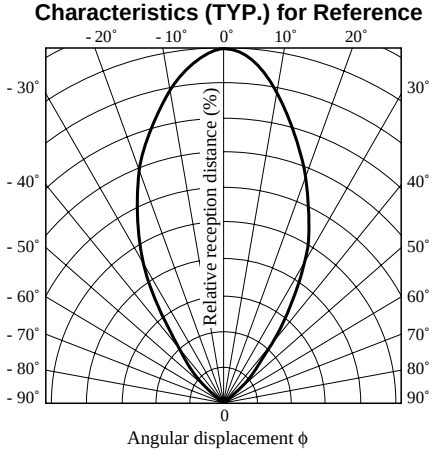


Fig. 4 Relative Reception Distance vs. Ambient Temperature (TYP.) for Reference

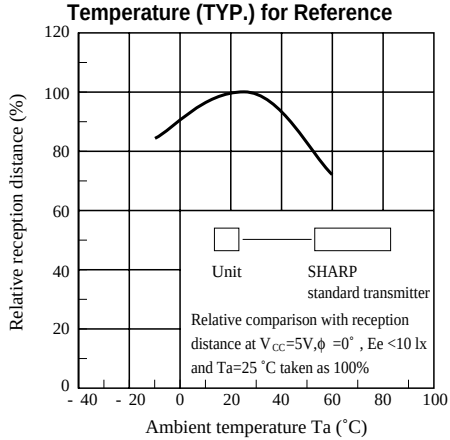
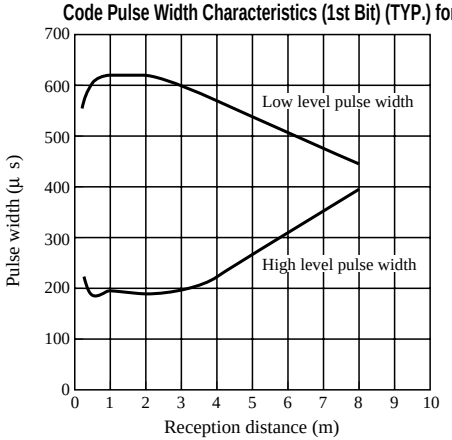


Fig. 5 AEHA (Japan Association of Electrical Home Appliances) Code Pulse Width Characteristics (1st Bit) (TYP.) for Reference



(Conditions)

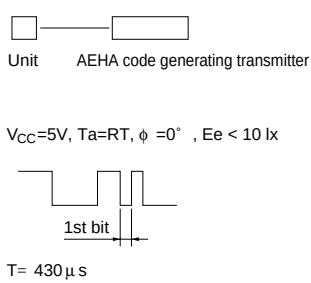
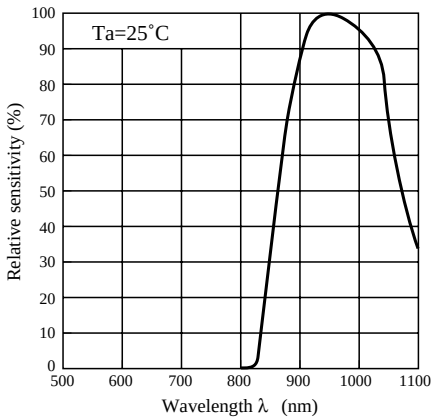


Fig. 6 Spectral Sensitivity for Reference



■ Precautions for Operation

(1) Use the light emitting unit (remote control transmitter), in consideration of performance, characteristics, operating conditions of light emitting device and the characteristics of the light detecting unit.

(2) Pay attention to a malfunction of the light detecting unit when the surface is stained with dust and refuse.
Care must be taken not to touch the light detector surface.

- Conduct cleaning as follows.

(3) Cleaning

Solvent dip cleaning : Solvent temperature of 45 ˚C max., dipping time : Within 3 minutes

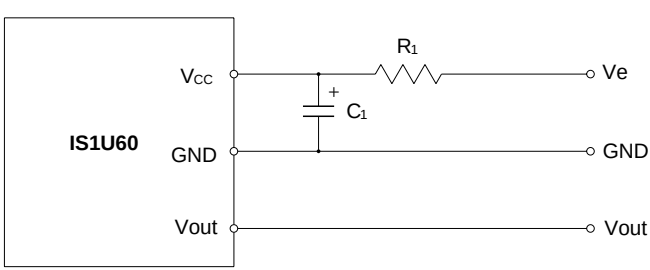
Ultrasonic cleaning : Elements are affected differently depending on the size of cleaning bath, ultrasonic output, time, size of PWB and mounting method of elements.
Conduct trial cleaning on actual operating conditions in advance to make sure that no problem results.

- Use the following solvents only.
  Solvents : Ethyl alcohol, methyl alcohol or isopropyl alcohol

(4) To avoid the electrostatic breakdown of IC, handle the unit under the condition of grounding with human body, soldering iron, etc.

(5) Do not apply unnecessary force to the terminal.

(6) Example of recommended external circuit (mount outer mounting parts near the sensor as much as possible.)



(Circuit constant)
R_1 = 47Ω ± 5%
C_1 = 47μF

NOTICE

- The circuit application examples in this publication are provided to explain representative applications of SHARP devices and are not intended to guarantee any circuit design or license any intellectual property rights. SHARP takes no responsibility for any problems related to any intellectual property right of a third party resulting from the use of SHARP's devices.
- Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device. SHARP reserves the right to make changes in the specifications, characteristics, data, materials, structure, and other contents described herein at any time without notice in order to improve design or reliability. Manufacturing locations are also subject to change without notice.
- Observe the following points when using any devices in this publication. SHARP takes no responsibility for damage caused by improper use of the devices which does not meet the conditions and absolute maximum ratings to be used specified in the relevant specification sheet nor meet the following conditions:

(i) The devices in this publication are designed for use in general electronic equipment designs such as:
– Personal computers
– Office automation equipment
– Telecommunication equipment [terminal]
– Test and measurement equipment
– Industrial control
– Audio visual equipment
– Consumer electronics

(ii) Measures such as fail-safe function and redundant design should be taken to ensure reliability and safety when SHARP devices are used for or in connection with equipment that requires higher reliability such as:
– Transportation control and safety equipment (i.e., aircraft, trains, automobiles, etc.)
– Traffic signals
– Gas leakage sensor breakers
– Alarm equipment
– Various safety devices, etc.

(iii) SHARP devices shall not be used for or in connection with equipment that requires an extremely high level of reliability and safety such as:
– Space applications
– Telecommunication equipment [trunk lines]
– Nuclear power control equipment
– Medical and other life support equipment (e.g., scuba).

- Contact a SHARP representative in advance when intending to use SHARP devices for any "specific" applications other than those recommended by SHARP or when it is unclear which category mentioned above controls the intended use.
- If the SHARP devices listed in this publication fall within the scope of strategic products described in the Foreign Exchange and Foreign Trade Control Law of Japan, it is necessary to obtain approval to export such SHARP devices.
-
- Contact and consult with a SHARP representative if there are any questions about the contents of this publication.

PIC16F62X

Data Sheet

FLASH-Based

8-Bit CMOS Microcontroller

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products.

Trademarks

Printed on recycled paper.

DNV Certification, Inc. USA
ANSI • RAB QMS ACCREDITED
DNV
DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

PIC16F62X

FLASH-Based 8-Bit CMOS Microcontrollers

Devices Included in this Data Sheet:

- PIC16F627
- PIC16F628

Referred to collectively as PIC16F62X

High Performance RISC CPU:

- Only 35 instructions to learn
- All single cycle instructions (200 ns), except for program branches which are two-cycle
- Operating speed:
  - DC - 20 MHz clock input
  - DC - 200 ns instruction cycle

Device	Memory		
	FLASH Program	RAM Data	EEPROM Data
PIC16F627	1024 x 14	224 x 8	128 x 8
PIC16F628	2048 x 14	224 x 8	128 x 8

- Interrupt capability
- 16 special function hardware registers
- 8-level deep hardware stack
- Direct, Indirect and Relative addressing modes

Peripheral Features:

- 16 I/O pins with individual direction control
- High current sink/source for direct LED drive
- Analog comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (V_{REF}) module
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible
- Timer0: 8-bit timer/counter with 8-bit programmable prescaler
- Timer1: 16-bit timer/counter with external crystal/clock capability
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Capture, Compare, PWM (CCP) module
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- Universal Synchronous/Asynchronous Receiver/Transmitter USART/SCI
- 16 Bytes of common RAM

Special Microcontroller Features:

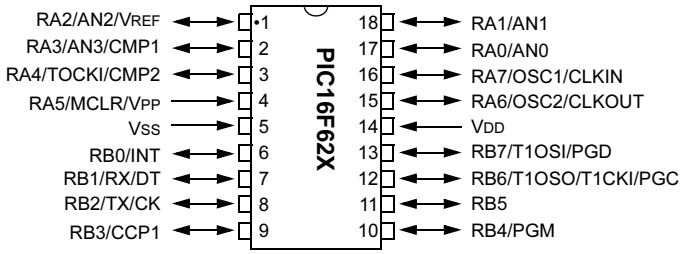
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Brown-out Detect (BOD)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Multiplexed $\overline{\text{MCLR}}$-pin
- Programmable weak pull-ups on PORTB
- Programmable code protection
- Low voltage programming
- Power saving SLEEP mode
- Selectable oscillator options
  - FLASH configuration bits for oscillator options
  - ER (External Resistor) oscillator
    - Reduced part count
  - Dual speed INTRC
    - Lower current consumption
  - EC External Clock input
  - XT Oscillator mode
  - HS Oscillator mode
  - LP Oscillator mode
- In-circuit Serial Programming™ (via two pins)
- Four user programmable ID locations

CMOS Technology:

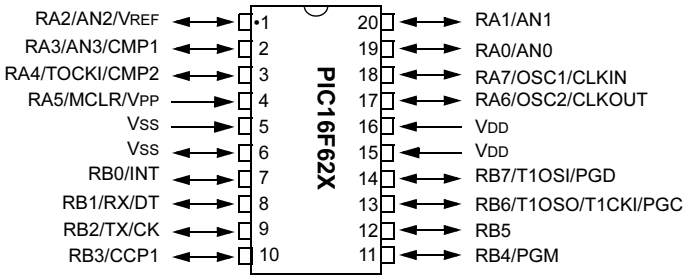
- Low power, high speed CMOS FLASH technology
- Fully static design
- Wide operating voltage range
  - PIC16F627 - 3.0V to 5.5V
  - PIC16F628 - 3.0V to 5.5V
  - PIC16LF627 - 2.0V to 5.5V
  - PIC16LF628 - 2.0V to 5.5V
- Commercial, industrial and extended temperature range
- Low power consumption
  - < 2.0 mA @ 5.0V, 4.0 MHz
  - 15 μA typical @ 3.0V, 32 kHz
  - < 1.0 μA typical standby current @ 3.0V

Pin Diagrams

PDIP, SOIC



SSOP



Device Differences

Device	Voltage Range	Oscillator	Process Technology (Microns)
PIC16F627	3.0 - 5.5	**(Note 1)**	0.7
PIC16F628	3.0 - 5.5	**(Note 1)**	0.7
PIC16LF627	2.0 - 5.5	**(Note 1)**	0.7
PIC16LF628	2.0 - 5.5	**(Note 1)**	0.7

Note 1: If you change from this device to another device, please verify oscillator characteristics in your application.

Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@mail.microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 792-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com/cn** to receive the most current information on all of our products.

NOTES:

1.0 PIC16F62X DEVICE VARIETIES

A variety of frequency ranges and packaging options are available. Depending on application and production requirements, the proper device option can be selected using the information in the PIC16F62X Product Identification System section (Page 167) at the end of this data sheet. When placing orders, please use this page of the data sheet to specify the correct part number.

1.1 FLASH Devices

FLASH devices can be erased and reprogrammed electrically. This allows the same device to be used for prototype development, pilot programs and production.

A further advantage of the electrically-erasable FLASH is that it can be erased and reprogrammed in-circuit, or by device programmers, such as Microchip's PICSTART® Plus, or PRO MATE® II programmers.

1.2 Quick-Turnaround Production (QTP) Devices

Microchip offers a QTP Programming Service for factory production orders. This service is made available for users who chose not to program a medium-to-high quantity of units and whose code patterns have stabilized. The devices are standard FLASH devices but with all program locations and configuration options already programmed by the factory. Certain code and prototype verification procedures apply before production shipments are available. Please contact your Microchip Technology sales office for more details.

1.3 Serialized Quick-Turnaround Production (SQTP^{sm}) Devices

Microchip offers a unique programming service where a few user-defined locations in each device are programmed with different serial numbers. The serial numbers may be random, pseudo-random or sequential.

Serial programming allows each device to have a unique number which can serve as an entry-code, password or ID number.

NOTES:

2.0 ARCHITECTURAL OVERVIEW

The high performance of the PIC16F62X family can be attributed to a number of architectural features commonly found in RISC microprocessors. To begin with, the PIC16F62X uses a Harvard architecture, in which, program and data are accessed from separate memories using separate buses. This improves bandwidth over traditional Von Neumann architecture where program and data are fetched from the same memory. Separating program and data memory further allows instructions to be sized differently than 8-bit wide data word. Instruction opcodes are 14-bits wide making it possible to have all single-word instructions. A 14-bit wide program memory access bus fetches a 14-bit instruction in a single cycle. A two-stage pipeline overlaps fetch and execution of instructions. Consequently, all instructions (35) execute in a single cycle (200 ns @ 20 MHz) except for program branches.

The Table below lists program memory (FLASH, Data and EEPROM).

TABLE 2-1: DEVICE DESCRIPTION

Device	Memory		
	FLASH Program	RAM Data	EEPROM Data
PIC16F627	1024 x 14	224 x 8	128 x 8
PIC16F628	2048 x 14	224 x 8	128 x 8
PIC16LF627	1024 x 14	224 x 8	128 x 8
PIC16LF628	2048 x 14	224 x 8	128 x 8

The PIC16F62X can directly or indirectly address its register files or data memory. All Special Function registers, including the program counter, are mapped in the data memory. The PIC16F62X have an orthogonal (symmetrical) instruction set that makes it possible to carry out any operation, on any register, using any Addressing mode. This symmetrical nature, and lack of ‘special optimal situations’ make programming with the PIC16F62X simple yet efficient. In addition, the learning curve is reduced significantly.

The PIC16F62X devices contain an 8-bit ALU and working register. The ALU is a general purpose arithmetic unit. It performs arithmetic and Boolean functions between data in the working register and any register file.

The ALU is 8-bit wide and capable of addition, subtraction, shift and logical operations. Unless otherwise mentioned, arithmetic operations are two's complement in nature. In two-operand instructions, typically one operand is the working register (W register). The other operand is a file register or an immediate constant. In single operand instructions, the operand is either the W register or a file register.

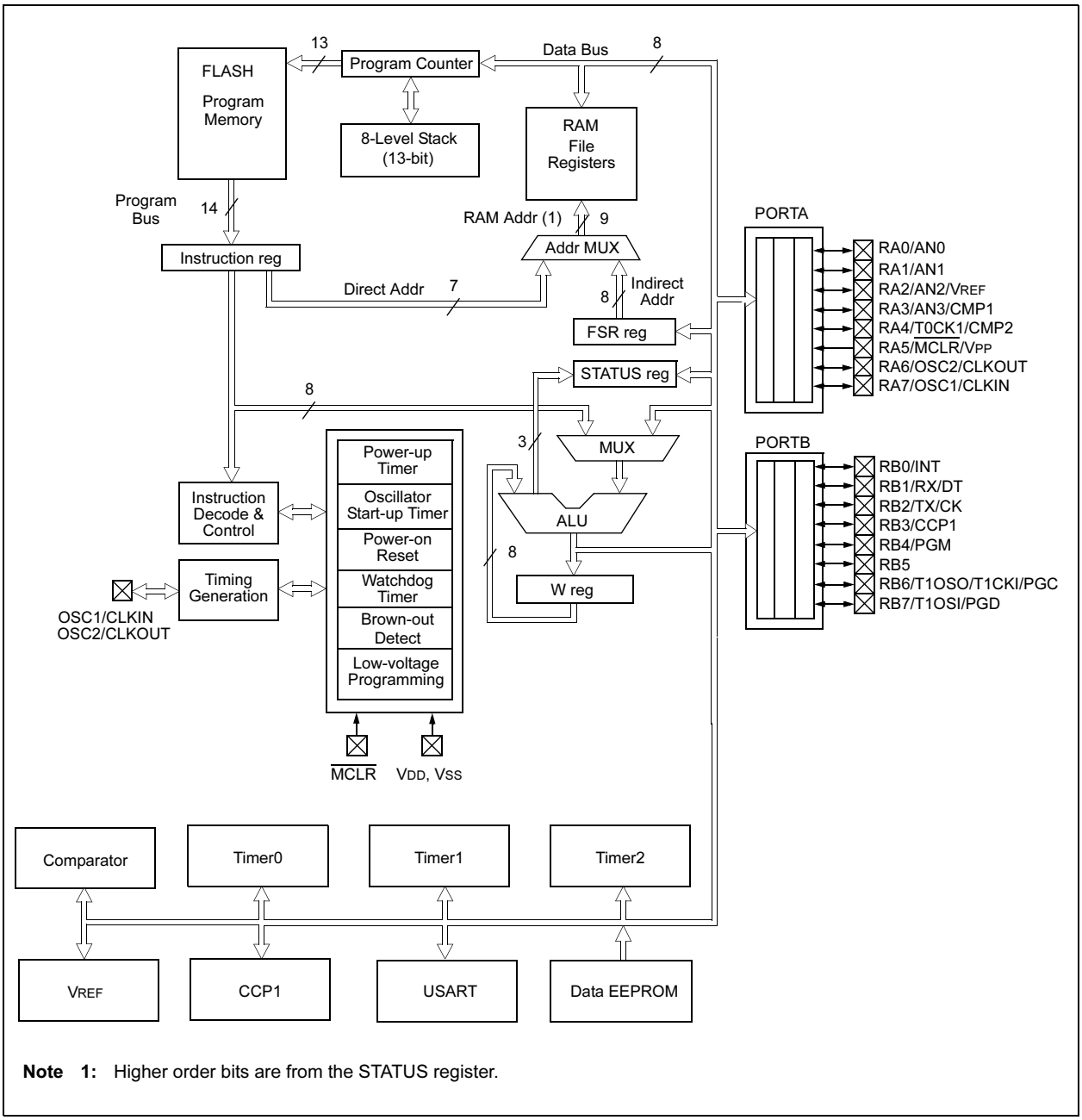
The W register is an 8-bit working register used for ALU operations. It is not an addressable register.

Depending on the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), and Zero (Z) bits in the STATUS register. The C and DC bits operate as a $\overline{\text{Borrow}}$ and $\overline{\text{Digit Borrow}}$ out bit, respectively, bit in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

A simplified block diagram is shown in Figure 2-1, and a description of the device pins in Table 2-1.

Two types of data memory are provided on the PIC16F62X devices. Non-volatile EEPROM data memory is provided for long term storage of data such as calibration values, lookup table data, and any other data which may require periodic updating in the field. This data is not lost when power is removed. The other data memory provided is regular RAM data memory. Regular RAM data memory is provided for temporary storage of data during normal operation. It is lost when power is removed.

FIGURE 2-1: BLOCK DIAGRAM



Note 1: Higher order bits are from the STATUS register.

TABLE 2-1: PIC16F62X PINOUT DESCRIPTION

Name	Function	Input Type	Output Type	Description
RA0/AN0	RA0	ST	CMOS	Bi-directional I/O port
	AN0	AN	—	Analog comparator input
RA1/AN1	RA1	ST	CMOS	Bi-directional I/O port
	AN1	AN	—	Analog comparator input
RA2/AN2/VREF	RA2	ST	CMOS	Bi-directional I/O port
	AN2	AN	—	Analog comparator input
	VREF	—	AN	VREF output
RA3/AN3/CMP1	RA3	ST	CMOS	Bi-directional I/O port
	AN3	AN	—	Analog comparator input
	CMP1	—	CMOS	Comparator 1 output
RA4/T0CKI/CMP2	RA4	ST	OD	Bi-directional I/O port
	T0CKI	ST	—	Timer0 clock input
	CMP2	—	OD	Comparator 2 output
RA5/$\overline{\text{MCLR}}$/VPP	RA5	ST	—	Input port
	$\overline{\text{MCLR}}$	ST	—	Master clear
	VPP	—	—	Programming voltage input. When configured as $\overline{\text{MCLR}}$, this pin is an active low RESET to the device. Voltage on $\overline{\text{MCLR}}$/VPP must not exceed VDD during normal device operation.
RA6/OSC2/CLKOUT	RA6	ST	CMOS	Bi-directional I/O port
	OSC2	XTAL	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
	CLKOUT	—	CMOS	In ER/INTRC mode, OSC2 pin can output CLKOUT, which has 1/4 the frequency of OSC1
RA7/OSC1/CLKIN	RA7	ST	CMOS	Bi-directional I/O port
	OSC1	XTAL	—	Oscillator crystal input
	CLKIN	ST	—	External clock source input. ER biasing pin.
RB0/INT	RB0	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	INT	ST	—	External interrupt.
RB1/RX/DT	RB1	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	RX	ST	—	USART receive pin
	DT	ST	CMOS	Synchronous data I/O.
RB2/TX/CK	RB2	TTL	CMOS	Bi-directional I/O port.
	TX	—	CMOS	USART transmit pin
	CK	ST	CMOS	Synchronous clock I/O. Can be software programmed for internal weak pull-up.
RB3/CCP1	RB3	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	CCP1	ST	CMOS	Capture/Compare/PWM I/O

Legend: O = Output; — = Not used; TTL = TTL Input; CMOS = CMOS Output; I = Input; OD = Open Drain Output; P = Power; ST = Schmitt Trigger Input; AN = Analog

TABLE 2-1: PIC16F62X PINOUT DESCRIPTION (CONTINUED)

Name	Function	Input Type	Output Type	Description
RB4/PGM	RB4	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	PGM	ST	—	Low voltage programming input pin. Interrupt-on-pin change. When low voltage programming is enabled, the interrupt-on-pin change and weak pull-up resistor are disabled.
RB5	RB5	TTL	CMOS	Bi-directional I/O port. Interrupt-on-pin change. Can be software programmed for internal weak pull-up.
RB6/T1OSO/T1CKI/PGC	RB6	TTL	CMOS	Bi-directional I/O port. Interrupt-on-pin change. Can be software programmed for internal weak pull-up.
	T1OSO	—	XTAL	Timer1 oscillator output.
	T1CKI	ST	—	Timer1 clock input.
	PGC	ST	—	ICSP™ Programming Clock.
RB7/T1OSI/PGD	RB7	TTL	CMOS	Bi-directional I/O port. Interrupt-on-pin change. Can be software programmed for internal weak pull-up.
	T1OSI	XTAL	—	Timer1 oscillator input. Wake-up from SLEEP on pin change. Can be software programmed for internal weak pull-up.
	PGD	ST	CMOS	ICSP Data I/O
V_{SS}	V_{SS}	Power	—	Ground reference for logic and I/O pins
V_{DD}	V_{DD}	Power	—	Positive supply for logic and I/O pins

Legend: O = Output; — = Not used; TTL = TTL Input; CMOS = CMOS Output; I = Input; OD = Open Drain Output; P = Power; ST = Schmitt Trigger Input; AN = Analog

2.1 Clocking Scheme/Instruction Cycle

The clock input (OSC1/CLKIN/RA7 pin) is internally divided by four to generate four non-overlapping quadrature clocks namely Q1, Q2, Q3 and Q4. Internally, the program counter (PC) is incremented every Q1, the instruction is fetched from the program memory and latched into the instruction register in Q4. The instruction is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow is shown in Figure 2-2.

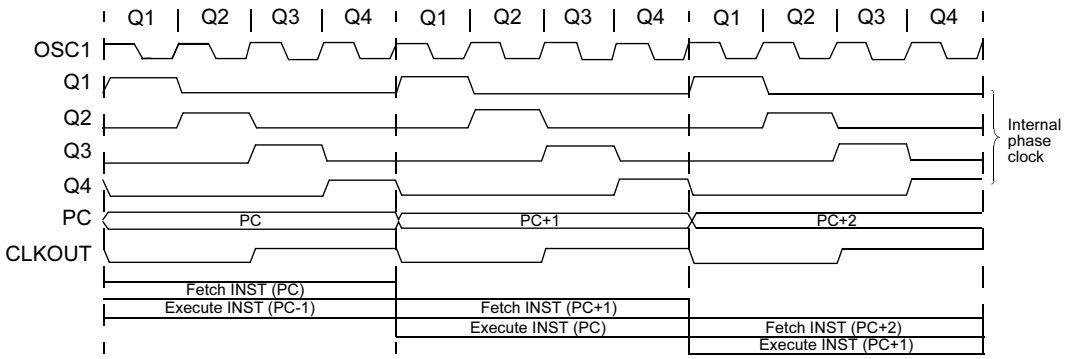
2.2 Instruction Flow/Pipelining

An “Instruction Cycle” consists of four Q cycles (Q1, Q2, Q3 and Q4). The instruction fetch and execute are pipelined such that fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change, (e.g., `GOTO`) then two cycles are required to complete the instruction (Example 2-1).

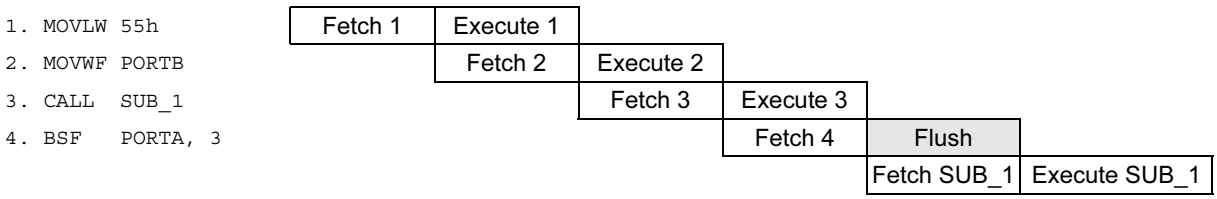
A fetch cycle begins with the program counter (PC) incrementing in Q1.

In the execution cycle, the fetched instruction is latched into the “Instruction Register (IR)” in cycle Q1. This instruction is then decoded and executed during the Q2, Q3, and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).

FIGURE 2-2: CLOCK/INSTRUCTION CYCLE



EXAMPLE 2-1: INSTRUCTION PIPELINE FLOW



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is “flushed” from the pipeline while the new instruction is being fetched and then executed.

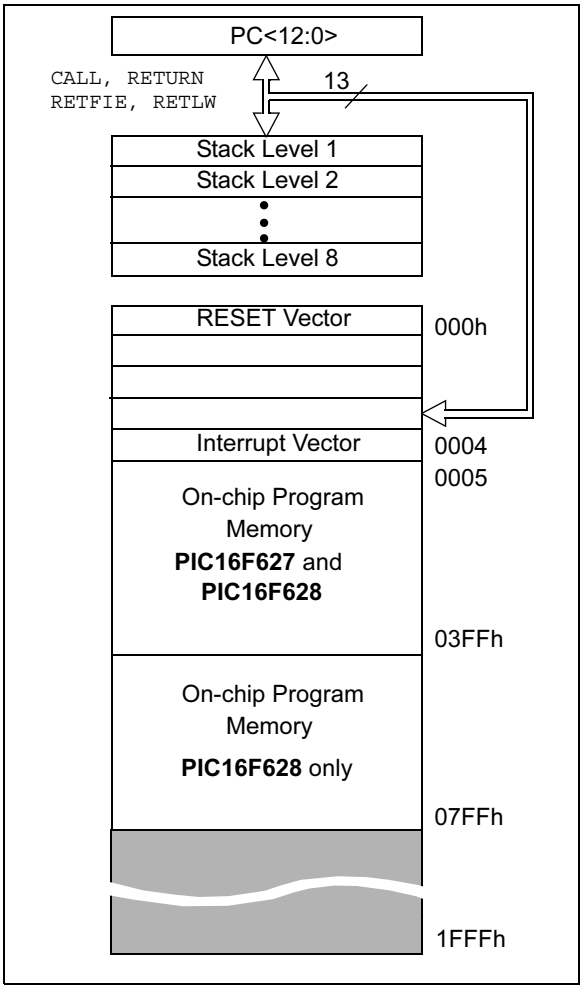
NOTES:

3.0 MEMORY ORGANIZATION

3.1 Program Memory Organization

The PIC16F62X has a 13-bit program counter capable of addressing an 8K x 14 program memory space. Only the first 1K x 14 (0000h - 03FFh) for the PIC16F627 and 2K x 14 (0000h - 07FFh) for the PIC16F628 are physically implemented. Accessing a location above these boundaries will cause a wrap-around within the first 1K x 14 space (PIC16F627) or 2K x 14 space (PIC16F628). The RESET vector is at 0000h and the interrupt vector is at 0004h (Figure 3-1).

FIGURE 3-1: PROGRAM MEMORY MAP AND STACK



3.2 Data Memory Organization

The data memory (Figure 3-2) is partitioned into four banks, which contain the general purpose registers and the Special Function Registers (SFR). The SFR's are located in the first 32 locations of each Bank. Register locations 20-7Fh, A0h-FFh, 120h-14Fh, 170h-17Fh and 1F0h-1FFh are general purpose registers implemented as static RAM.

The Table below lists how to access the four banks of registers:

	RP1	RP0
Bank0	0	0
Bank1	0	1
Bank2	1	0
Bank3	1	1

Addresses F0h-FFh, 170h-17Fh and 1F0h-1FFh are implemented as common RAM and mapped back to addresses 70h-7Fh.

3.2.1 GENERAL PURPOSE REGISTER FILE

The register file is organized as 224 x 8 in the PIC16F62X. Each is accessed either directly or indirectly through the File Select Register FSR (See Section 3.4).

FIGURE 3-2: DATA MEMORY MAP OF THE PIC16F627 AND PIC16F628

Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3	File Address
Indirect addr.(1)	00h	Indirect addr.(1)	80h	Indirect addr.(1)	100h	Indirect addr.(1)	180h
TMR0	01h	OPTION	81h	TMR0	101h	OPTION	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
	07h		87h		107h		187h
	08h		88h		108h		188h
	09h		89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch
	0Dh		8Dh		10Dh		18Dh
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh
TMR1H	0Fh		8Fh		10Fh		18Fh
T1CON	10h		90h				
TMR2	11h		91h				
T2CON	12h	PR2	92h				
	13h		93h				
	14h		94h				
CCPR1L	15h		95h				
CCPR1H	16h		96h				
CCP1CON	17h		97h				
RCSTA	18h	TXSTA	98h				
TXREG	19h	SPBRG	99h				
RCREG	1Ah	EEDATA	9Ah				
	1Bh	EEADR	9Bh				
	1Ch	EECON1	9Ch				
	1Dh	EECON2(1)	9Dh				
	1Eh		9Eh				
CMCON	1Fh	VRCON	9Fh		11Fh		
General Purpose Register 80 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 48 Bytes	120h		
					14Fh		
					150h		
	6Fh		EFh		16Fh		1EFh
16 Bytes	70h	accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
	7Fh		FFh		17Fh		1FFh

Unimplemented data memory locations, read as '0'.

Note 1: Not a physical register.

3.2.2 SPECIAL FUNCTION REGISTERS

The SFRs are registers used by the CPU and Peripheral functions for controlling the desired operation of the device (Table 3-1). These registers are static RAM.

The special registers can be classified into two sets (core and peripheral). The SFRs associated with the “core” functions are described in this section. Those related to the operation of the peripheral features are described in the section of that peripheral feature.

TABLE 3-1: SPECIAL REGISTERS SUMMARY BANK 0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset(1)	Details on Page
Bank 0											
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	25
01h	TMR0	Timer0 Module’s Register								xxxx xxxx	43
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	13
03h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	19
04h	FSR	Indirect data memory address pointer								xxxx xxxx	25
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx 0000	29
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	34
07h	—	Unimplemented								—	—
08h	—	Unimplemented								—	—
09h	—	Unimplemented								—	—
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of program counter					---0 0000	25
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	21
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	23
0Dh	—	Unimplemented								—	—
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1								xxxx xxxx	46
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1								xxxx xxxx	46
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	--00 0000	46
11h	TMR2	TMR2 module’s register								0000 0000	50
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	50
13h	—	Unimplemented								—	—
14h	—	Unimplemented								—	—
15h	CCPR1L	Capture/Compare/PWM register (LSB)								xxxx xxxx	61
16h	CCPR1H	Capture/Compare/PWM register (MSB)								xxxx xxxx	61
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	61
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	67
19h	TXREG	USART Transmit data register								0000 0000	74
1Ah	RCREG	USART Receive data register								0000 0000	77
1Bh	—	Unimplemented								—	—
1Ch	—	Unimplemented								—	—
1Dh	—	Unimplemented								—	—
1Eh	—	Unimplemented								—	—
1Fh	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0000	53

Legend: — = Unimplemented locations read as ‘0’, u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: For the Initialization Condition for Registers Tables, refer to Table 14-7 and Table 14-8 on page 98.

TABLE 3-2: SPECIAL FUNCTION REGISTERS SUMMARY BANK 1

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset(1)	Details on Page
Bank 1											
80h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	25
81h	OPTION	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	20
82h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	25
83h	STATUS	IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	19
84h	FSR	Indirect data memory address pointer								xxxx xxxx	25
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	29
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	34
87h	—	Unimplemented								—	—
88h	—	Unimplemented								—	—
89h	—	Unimplemented								—	—
8Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of program counter					---0 0000	25
8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	21
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	22
8Dh	—	Unimplemented								—	—
8Eh	PCON	—	—	—	—	OSCF	—	$\overline{\text{POR}}$	$\overline{\text{BOD}}$	---- 1-0x	24
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	—	Unimplemented								—	—
92h	PR2	Timer2 Period Register								1111 1111	50
93h	—	Unimplemented								—	—
94h	—	Unimplemented								—	—
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	69
99h	SPBRG	Baud Rate Generator Register								0000 0000	69
9Ah	EEDATA	EEPROM data register								xxxx xxxx	87
9Bh	EEADR	—	EEPROM address register							xxxx xxxx	87
9Ch	EECON1	—	—	—	—	WRERR	WREN	WR	RD	---- x000	87
9Dh	EECON2	EEPROM control register 2 (not a physical register)								--------	87
9Eh	—	Unimplemented								—	—
9Fh	VRCON	VREN	VROE	VRR	—	VR3	VR2	VR1	VR0	000- 0000	59

Legend: — = Unimplemented locations read as ‘0’, u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: For the Initialization Condition for Registers Tables, refer to Table 14-7 and Table 14-8 on page 98.

TABLE 3-3: SPECIAL FUNCTION REGISTERS SUMMARY BANK 2

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset(1)	Details on Page
Bank 2											
100h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	25
101h	TMR0	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	43
102h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	25
103h	STATUS	IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	19
104h	FSR	Indirect data memory address pointer								xxxx xxxx	25
105h	—	Unimplemented								—	—
106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	34
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of program counter					---0 0000	25
10Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	21
10Ch	—	Unimplemented								—	—
10Dh	—	Unimplemented								—	—
10Eh	—	Unimplemented								—	—
10Fh	—	Unimplemented								—	—
110h	—	Unimplemented								—	—
111h	—	Unimplemented								—	—
112h	—	Unimplemented								—	—
113h	—	Unimplemented								—	—
114h	—	Unimplemented								—	—
115h	—	Unimplemented								—	—
116h	—	Unimplemented								—	—
117h	—	Unimplemented								—	—
118h	—	Unimplemented								—	—
119h	—	Unimplemented								—	—
11Ah	—	Unimplemented								—	—
11Bh	—	Unimplemented								—	—
11Ch	—	Unimplemented								—	—
11Dh	—	Unimplemented								—	—
11Eh	—	Unimplemented								—	—
11Fh	—	Unimplemented								—	—

Legend: — = Unimplemented locations read as ‘0’, u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented.

Note 1: For the Initialization Condition for Registers Tables, refer to Table 14-7 and Table 14-8 on page 98.

TABLE 3-4: SPECIAL FUNCTION REGISTERS SUMMARY BANK 3

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset(1)	Details on Page
Bank 3											
180h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	25
181h	OPTION	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	20
182h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	25
183h	STATUS	IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	19
184h	FSR	Indirect data memory address pointer								xxxx xxxx	25
185h	—	Unimplemented								—	—
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	34
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of program counter					---0 0000	25
18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	21
18Ch	—	Unimplemented								—	—
18Dh	—	Unimplemented								—	—
18Eh	—	Unimplemented								—	—
18Fh	—	Unimplemented								—	—
190h	—	Unimplemented								—	—
191h	—	Unimplemented								—	—
192h	—	Unimplemented								—	—
193h	—	Unimplemented								—	—
194h	—	Unimplemented								—	—
195h	—	Unimplemented								—	—
196h	—	Unimplemented								—	—
197h	—	Unimplemented								—	—
198h	—	Unimplemented								—	—
199h	—	Unimplemented								—	—
19Ah	—	Unimplemented								—	—
19Bh	—	Unimplemented								—	—
19Ch	—	Unimplemented								—	—
19Dh	—	Unimplemented								—	—
19Eh	—	Unimplemented								—	—
19Fh	—	Unimplemented								—	—

Legend: — = Unimplemented locations read as ‘0’, u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: For the Initialization Condition for Registers Tables, refer to Table 14-7 and Table 14-8 on page 98.

3.2.2.1 STATUS Register

The STATUS register, shown in Register 3-1, contains the arithmetic status of the ALU, the RESET status and the bank select bits for data memory (SRAM).

The STATUS register can be the destination for any instruction, like any other register. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to the device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper-three bits and set the Z bit. This leaves the STATUS register as `000uu1uu` (where `u` = unchanged).

It is recommended, therefore, that only `BCF, BSF, SWAPF` and `MOVWF` instructions are used to alter the STATUS register because these instructions do not affect any STATUS bit. For other instructions, not affecting any STATUS bits, see the "Instruction Set Summary".

> **Note 1:** The C and DC bits operate as a $\overline{\text{Borrow}}$ and $\overline{\text{Digit Borrow}}$ out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

REGISTER 3-1: STATUS REGISTER (ADDRESS: 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP**: Register Bank Select bit (used for indirect addressing)
`1` = Bank 2, 3 (100h - 1FFh)
`0` = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
`00` = Bank 0 (00h - 7Fh)
`01` = Bank 1 (80h - FFh)
`10` = Bank 2 (100h - 17Fh)
`11` = Bank 3 (180h - 1FFh)

bit 4 **$\overline{TO}$**: Timeout bit
`1` = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
`0` = A WDT timeout occurred

bit 3 **$\overline{PD}$**: Power-down bit
`1` = After power-up or by the `CLRWDT` instruction
`0` = By execution of the `SLEEP` instruction

bit 2 **Z**: Zero bit
`1` = The result of an arithmetic or logic operation is zero
`0` = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{\text{borrow}}$ the polarity is reversed)
`1` = A carry-out from the 4th low order bit of the result occurred
`0` = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
`1` = A carry-out from the Most Significant bit of the result occurred
`0` = No carry-out from the Most Significant bit of the result occurred

> **Note 1:** For $\overline{\text{borrow}}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF, RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

3.2.2.2 OPTION Register

The OPTION register is a readable and writable register which contains various control bits to configure the TMR0/WDT prescaler, the external RB0/INT interrupt, TMR0, and the weak pull-ups on PORTB.

> **Note:** To achieve a 1:1 prescaler assignment for TMR0, assign the prescaler to the WDT (PSA = 1). See Section 6.3.1

REGISTER 3-2: OPTION REGISTER (ADDRESS: 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

3.2.2.3 INTCON Register

The INTCON register is a readable and writable register which contains the various enable and flag bits for all interrupt sources except the comparator module. See Section 3.2.2.4 and Section 3.2.2.5 for a description of the comparator enable and flag bits.

Note:	Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).

REGISTER 3-3: INTCON REGISTER (ADDRESS: 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = When at least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

3.2.2.4 PIE1 Register

This register contains interrupt enable bits.

REGISTER 3-4: PIE1 REGISTER (ADDRESS: 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **EEIE:** EE Write Complete Interrupt Enable Bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit 6 **CMIE**: Comparator Interrupt Enable bit
1 = Enables the comparator interrupt
0 = Disables the comparator interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **Unimplemented**: Read as ‘0’

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

3.2.2.5 PIR1 Register

This register contains interrupt flag bits.

> **Note:** Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 3-5: PIR1 REGISTER (ADDRESS: 0Ch)

R/W-0	R/W-0	R-0	R-0	U-0	R/W-0	R/W-0	R/W-0
EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **EEIF:** EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation has not completed or has not been started

bit 6 **CMIF**: Comparator Interrupt Flag bit
1 = Comparator output has changed
0 = Comparator output has not changed

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **Unimplemented**: Read as ‘0’

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture Mode
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare Mode
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM Mode
Unused in this mode

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

3.2.2.6 PCON Register

The PCON register contains flag bits to differentiate between a Power-on Reset, an external $\overline{\text{MCLR}}$ Reset, WDT Reset or a Brown-out Detect.

> **Note:** $\overline{\text{BOD}}$ is unknown on Power-on Reset. It must then be set by the user and checked on subsequent RESETS to see if $\overline{\text{BOD}}$ is cleared, indicating a brown-out has occurred. The $\overline{\text{BOD}}$ STATUS bit is a “don't care” and is not necessarily predictable if the brown-out circuit is disabled (by clearing the BODEN bit in the Configuration word).

REGISTER 3-6: PCON REGISTER (ADDRESS: 0Ch)

U-0	U-0	U-0	U-0	R/W-1	U-0	R/W-q	R/W-q
—	—	—	—	OSCF	—	$\overline{\text{POR}}$	$\overline{\text{BOD}}$
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as '0'

bit 3 **OSCF**: INTRC/ER oscillator frequency
1 = 4 MHz typical[(1)]
0 = 37 KHz typical

bit 2 **Unimplemented:** Read as '0'

bit 1 **$\overline{\text{POR}}$**: Power-on Reset STATUS bit
1 = No Power-on Reset occurred
0 = A Power-on Reset occurred (must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{\text{BOD}}$**: Brown-out Detect STATUS bit
1 = No Brown-out Reset occurred
0 = A Brown-out Reset occurred (must be set in software after a Brown-out Reset occurs)

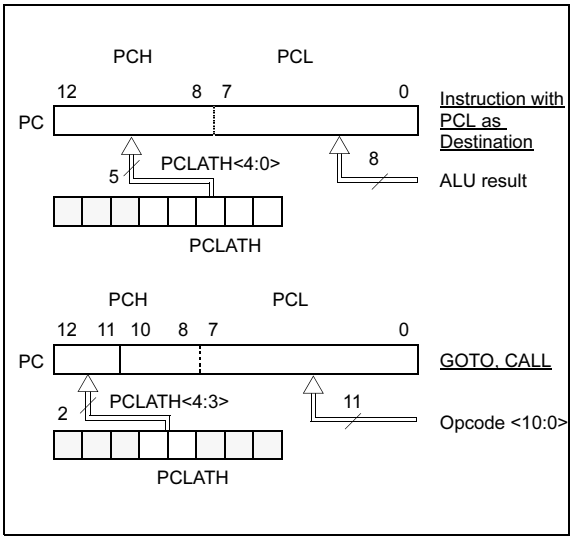
Note 1: When in ER Oscillator mode, setting OSCF = 1 will cause the oscillator frequency to change to the frequency specified by the external resistor.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

3.3 PCL and PCLATH

The program counter (PC) is 13-bits wide. The low byte comes from the PCL register, which is a readable and writable register. The high byte (PC<12:8>) is not directly readable or writable and comes from PCLATH. On any RESET, the PC is cleared. Figure 3-3 shows the two situations for the loading of the PC. The upper example in the figure shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). The lower example in the figure shows how the PC is loaded during a `CALL` or `GOTO` instruction (PCLATH<4:3> → PCH).

FIGURE 3-3: LOADING OF PC IN DIFFERENT SITUATIONS



3.3.1 COMPUTED GOTO

A computed `GOTO` is accomplished by adding an offset to the program counter (`ADDWF PCL`). When doing a table read using a computed `GOTO` method, care should be exercised if the table location crosses a PCL memory boundary (each 256 byte block). Refer to the application note *"Implementing a Table Read"* (AN556).

3.3.2 STACK

The PIC16F62X family has an 8-level deep x 13-bit wide hardware stack (Figure 3-1 and Figure 3-2). The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a `CALL` instruction is executed or an interrupt causes a branch. The stack is POPed in the event of a `RETURN`, `RETLW` or a `RETFIE` instruction execution. PCLATH is not affected by a PUSH or POP operation.

The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

> **Note 1:** There are no STATUS bits to indicate stack overflow or stack underflow conditions.
>
> **2:** There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the `CALL`, `RETURN`, `RETLW` and `RETFIE` instructions, or the vectoring to an interrupt address.

3.4 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

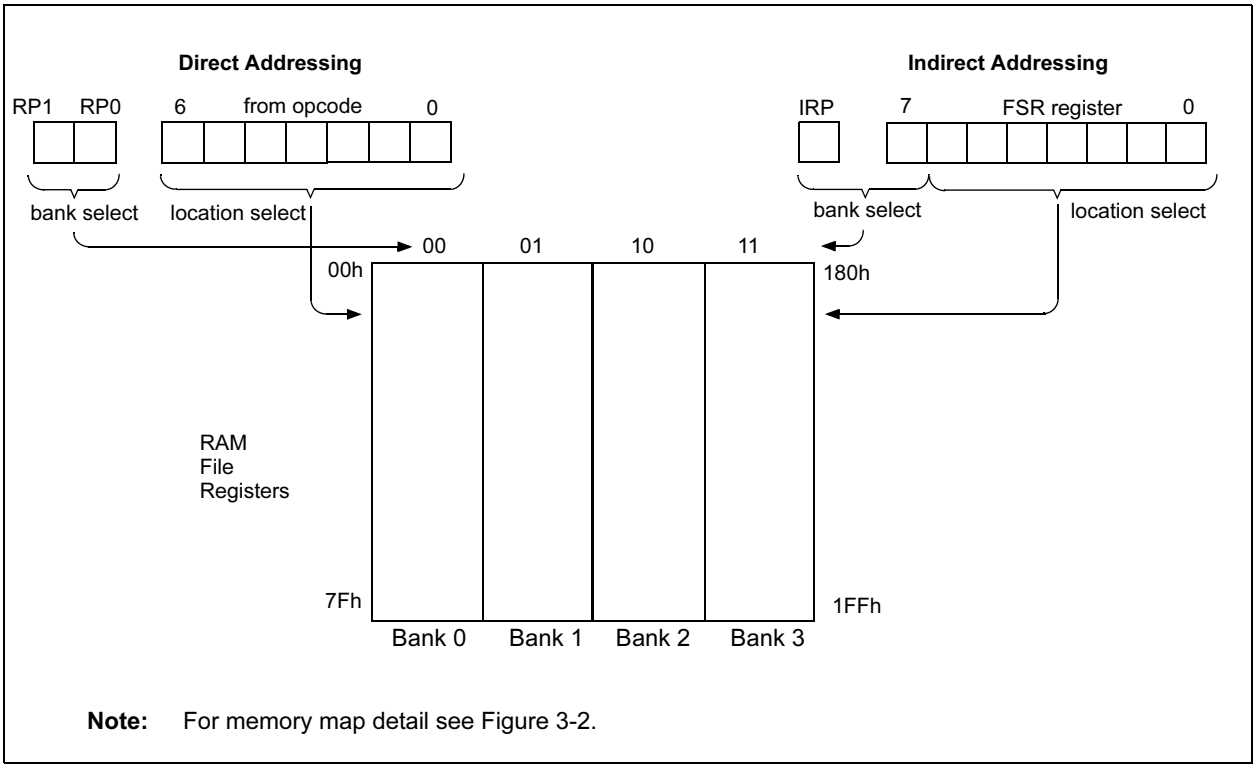
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses data pointed to by the file select register (FSR). Reading INDF itself indirectly will produce 00h. Writing to the INDF register indirectly results in a no-operation (although STATUS bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 3-4.

A simple program to clear RAM location 20h-2Fh using indirect addressing is shown in Example 3-1.

EXAMPLE 3-1: Indirect Addressing

```
        movlw   0x20    ;initialize pointer
        movwf   FSR     ;to RAM
NEXT    clrf    INDF    ;clear INDF register
        incf    FSR     ;inc pointer
        btfss   FSR,4   ;all done?
        goto    NEXT    ;no clear next
                        ;yes continue
```

FIGURE 3-4: DIRECT/INDIRECT ADDRESSING PIC16F62X



Note: For memory map detail see Figure 3-2.

4.0 GENERAL DESCRIPTION

The PIC16F62X are 18-Pin FLASH-based members of the versatile PIC16CXX family of low cost, high performance, CMOS, fully static, 8-bit microcontrollers.

All PICmicro® microcontrollers employ an advanced RISC architecture. The PIC16F62X have enhanced core features, eight-level deep stack, and multiple internal and external interrupt sources. The separate instruction and data buses of the Harvard architecture allow a 14-bit wide instruction word with the separate 8-bit wide data. The two-stage instruction pipeline allows all instructions to execute in a single cycle, except for program branches (which require two cycles). A total of 35 instructions (reduced instruction set) are available. Additionally, a large register set gives some of the architectural innovations used to achieve a very high performance.

PIC16F62X microcontrollers typically achieve a 2:1 code compression and a 4:1 speed improvement over other 8-bit microcontrollers in their class.

PIC16F62X devices have special features to reduce external components, thus reducing system cost, enhancing system reliability and reducing power consumption.

The PIC16F62X has eight oscillator configurations. The single pin ER oscillator provides a low cost solution. The LP oscillator minimizes power consumption, XT is a standard crystal, INTRC is a self-contained internal oscillator. The HS is for High Speed crystals. The EC mode is for an external clock source.

The SLEEP (Power-down) mode offers power savings. The user can wake-up the chip from SLEEP through several external interrupts, internal interrupts, and RESETS.

A highly reliable Watchdog Timer with its own on-chip RC oscillator provides protection against software lock-up.

Table 4-1 shows the features of the PIC16F62X mid-range microcontroller families.

A simplified block diagram of the PIC16F62X is shown in Figure 2.1.

The PIC16F62X series fits in applications ranging from battery chargers to low power remote sensors. The FLASH technology makes customization of application programs (detection levels, pulse generation, timers, etc.) extremely fast and convenient. The small footprint packages make this microcontroller series ideal for all applications with space limitations. Low cost, low power, high performance, ease of use and I/O flexibility make the PIC16F62X very versatile.

4.1 Development Support

The PIC16F62X family is supported by a full featured macro assembler, a software simulator, an in-circuit emulator, a low cost development programmer and a full-featured programmer. A Third Party “C” compiler support tool is also available.

TABLE 4-1: PIC16F62X FAMILY OF DEVICES

		PIC16F627	PIC16F628	PIC16LF627	PIC16LF628
Clock	Maximum Frequency of Operation (MHz)	20	20	4	4
Memory	FLASH Program Memory (words)	1024	2048	1024	2048
	RAM Data Memory (bytes)	224	224	224	224
	EEPROM Data Memory (bytes)	128	128	128	128
Peripherals	Timer Module(s)	TMR0, TMR1, TMR2	TMR0, TMR1, TMR2	TMR0, TMR1, TMR2	TMR0, TMR1, TMR2
	Comparator(s)	2	2	2	2
	Capture/Compare/PWM modules	1	1	1	1
	Serial Communications	USART	USART	USART	USART
	Internal Voltage Reference	Yes	Yes	Yes	Yes
Features	Interrupt Sources	10	10	10	10
	I/O Pins	16	16	16	16
	Voltage Range (Volts)	3.0-5.5	3.0-5.5	2.0-5.5	2.0-5.5
	Brown-out Detect	Yes	Yes	Yes	Yes
	Packages	18-pin DIP, SOIC, 20-pin SSOP	18-pin DIP, SOIC, 20-pin SSOP	18-pin DIP, SOIC, 20-pin SSOP	18-pin DIP, SOIC, 20-pin SSOP

All PICmicro® Family devices have Power-on Reset, selectable Watchdog Timer, selectable code protect and high I/O current capability. All PIC16F62X Family devices use serial programming with clock pin RB6 and data pin RB7.

NOTES:

5.0 I/O PORTS

The PIC16F62X have two ports, PORTA and PORTB. Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

5.1 PORTA and TRISA Registers

PORTA is an 8-bit wide latch. RA4 is a Schmitt Trigger input and an open drain output. Port RA4 is multiplexed with the T0CKI clock input. RA5 is a Schmitt Trigger input only and has no output drivers. All other RA port pins have Schmitt Trigger input levels and full CMOS output drivers. All pins have data direction bits (TRIS registers) which can configure these pins as input or output.

A '1' in the TRISA register puts the corresponding output driver in a Hi-impedance mode. A '0' in the TRISA register puts the contents of the output latch on the selected pin(s).

Reading the PORTA register reads the status of the pins whereas writing to it will write to the port latch. All write operations are read-modify-write operations. So a write to a port implies that the port pins are first read, then this value is modified and written to the port data latch.

The PORTA pins are multiplexed with comparator and voltage reference functions. The operation of these pins are selected by control bits in the CMCON (comparator control register) register and the VRCON (voltage reference control register) register. When selected as a comparator input, these pins will read as '0's.

Note:	RA5 shares function with VPP. When VPP voltage levels are applied to RA5, the device will enter Programming mode.

Note 1:	On RESET, the TRISA register is set to all inputs. The digital inputs are disabled and the comparator inputs are forced to ground to reduce current consumption.
2:	TRISA<6:7> is overridden by oscillator configuration. When PORTA<6:7> is overridden, the data reads ‘0’ and the TRISA<6:7> bits are ignored.

TRISA controls the direction of the RA pins, even when they are being used as comparator inputs. The user must make sure to keep the pins configured as inputs when using them as comparator inputs.

The RA2 pin will also function as the output for the voltage reference. When in this mode, the VREF pin is a very high impedance output. The user must configure TRISA<2> bit as an input and use high impedance loads.

In one of the Comparator modes defined by the CMCON register, pins RA3 and RA4 become outputs of the comparators. The TRISA<4:3> bits must be cleared to enable outputs to use this function.

EXAMPLE 5-1: Initializing PORTA

```
CLRF   PORTA       ;Initialize PORTA by
                   ;setting output data latches
MOVLW 0x07         ;Turn comparators off and
MOVWF CMCON        ;enable pins for I/O
                   ;functions
BCF    STATUS, RP1
BSF    STATUS, RP0;Select Bank1
MOVLW 0x1F         ;Value used to initialize
                   ;data direction
MOVWF TRISA        ;Set RA<4:0> as inputs
                   ;TRISA<5> always
                   ;read as ‘1’.
                   ;TRISA<7:6>
                   ;depend on oscillator mode
```

FIGURE 5-1: BLOCK DIAGRAM OF RA0/AN0:RA1/AN1 PINS

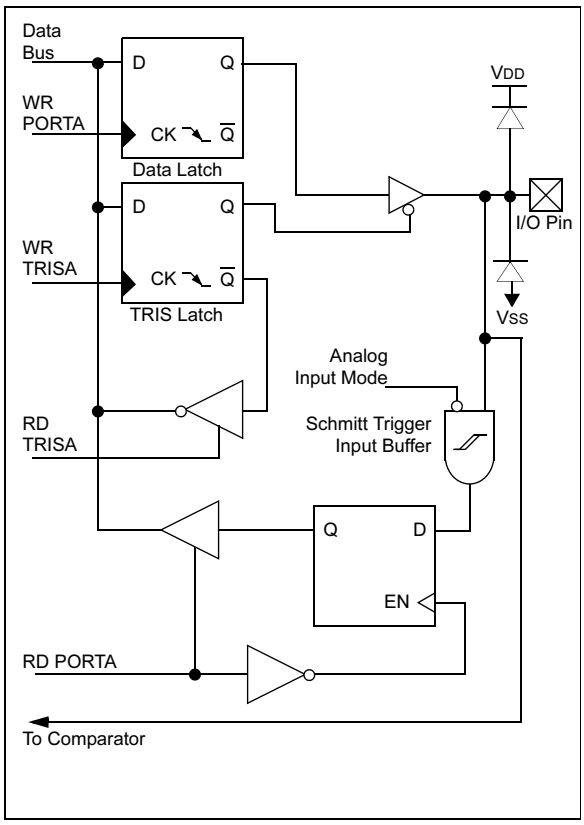


FIGURE 5-2: BLOCK DIAGRAM OF RA2/VREF PIN

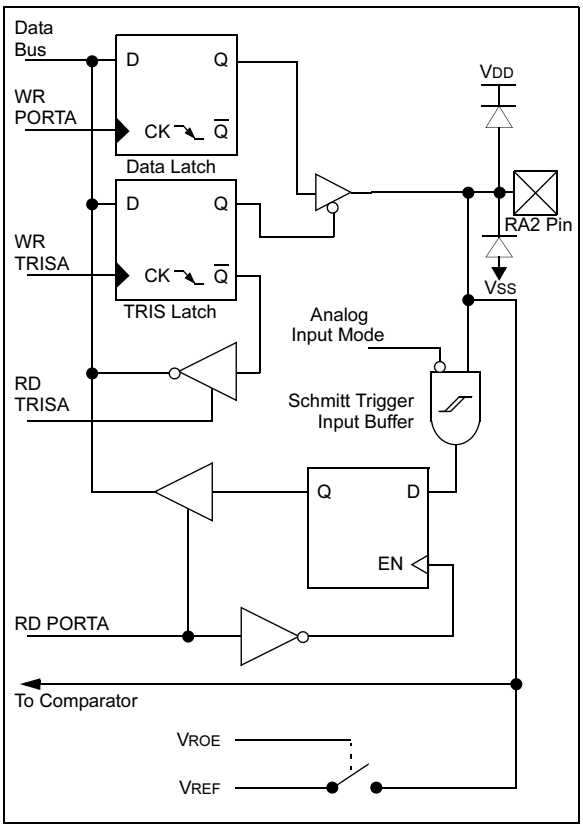


FIGURE 5-3: BLOCK DIAGRAM OF THE RA3/AN3 PIN

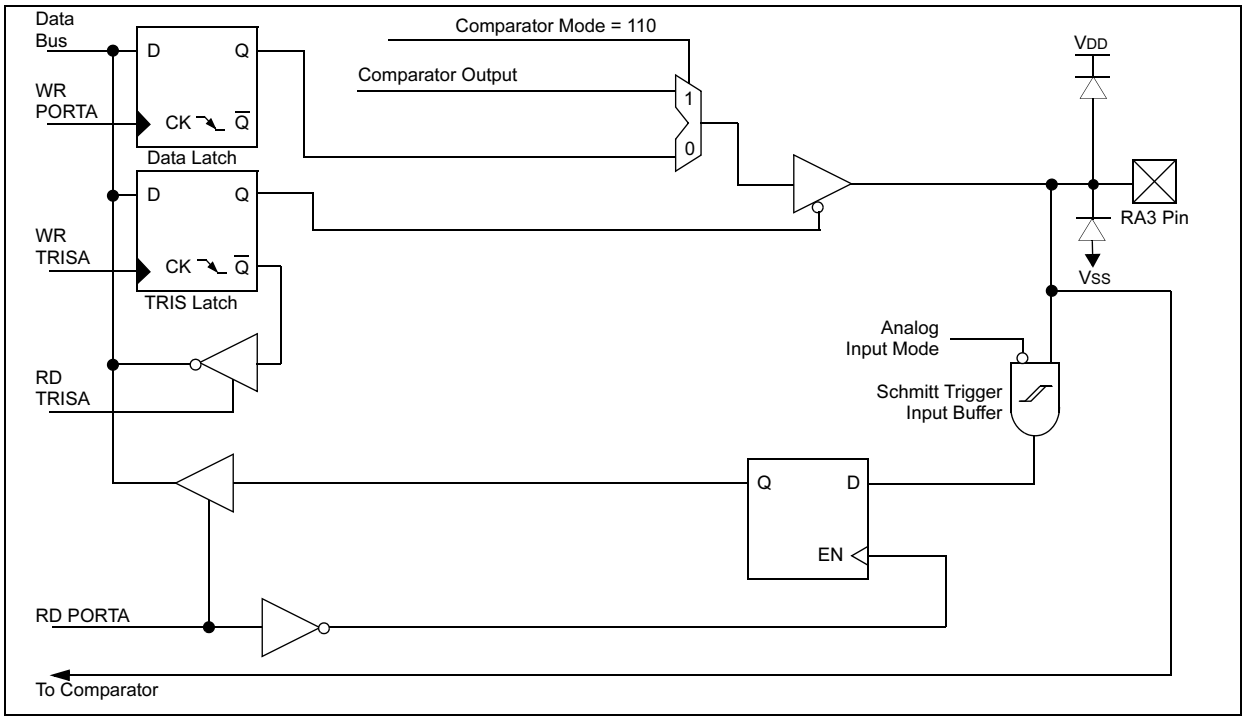


FIGURE 5-4: BLOCK DIAGRAM OF RA4/T0CKI PIN

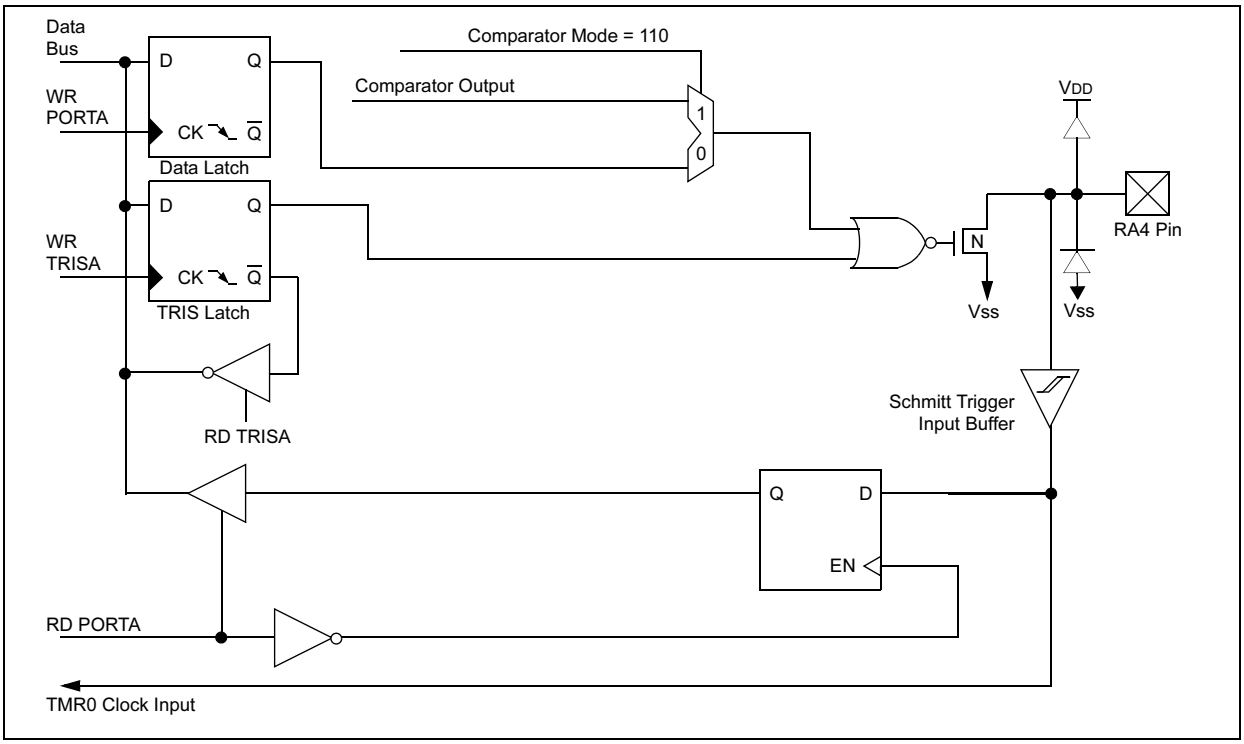


FIGURE 5-5: BLOCK DIAGRAM OF THE RA5/$\overline{\text{MCLR}}$/VPP PIN

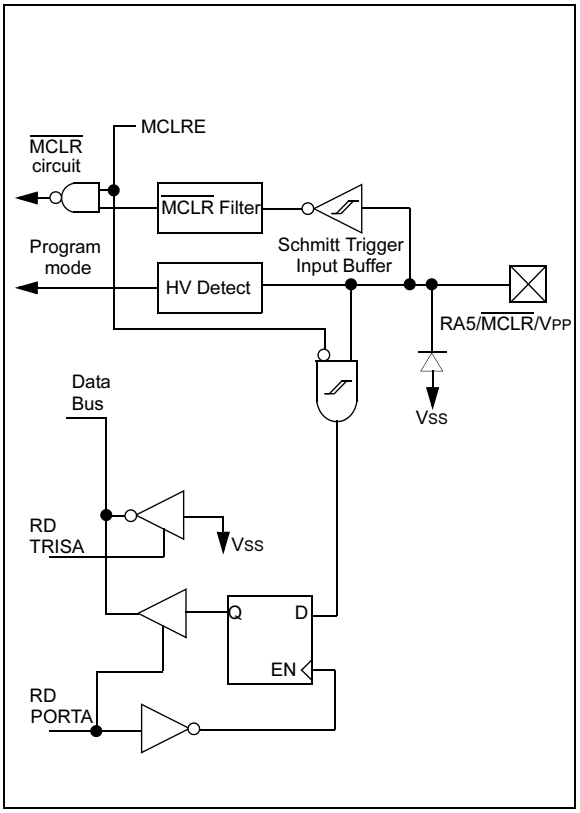
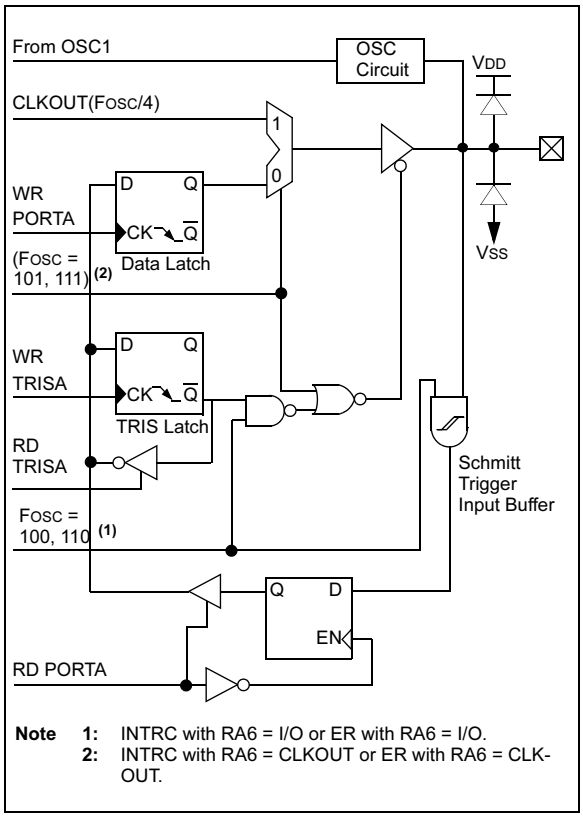
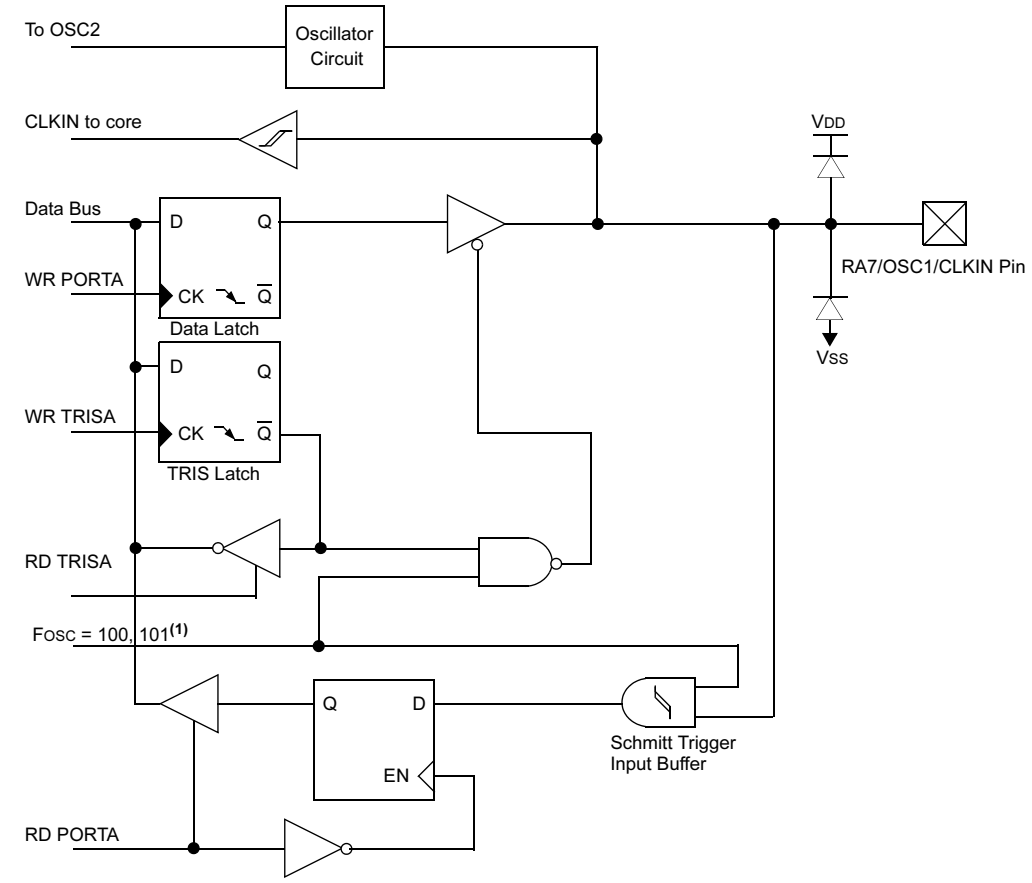


FIGURE 5-6: BLOCK DIAGRAM OF RA6/OSC2/CLKOUT PIN



Note 1: INTRC with RA6 = I/O or ER with RA6 = I/O.
2: INTRC with RA6 = CLKOUT or ER with RA6 = CLKOUT.

FIGURE 5-7: BLOCK DIAGRAM OF RA7/OSC1/CLKIN PIN



Note 1: INTRC with CLKOUT, and INTRC with I/O.

TABLE 5-1: PORTA FUNCTIONS

Name	Function	Input Type	Output Type	Description
RA0/AN0	RA0	ST	CMOS	Bi-directional I/O port
	AN0	AN	—	Analog comparator input
RA1/AN1	RA1	ST	CMOS	Bi-directional I/O port
	AN1	AN	—	Analog comparator input
RA2/AN2/VREF	RA2	ST	CMOS	Bi-directional I/O port
	AN2	AN	—	Analog comparator input
	VREF	—	AN	VREF output
RA3/AN3/CMP1	RA3	ST	CMOS	Bi-directional I/O port
	AN3	AN	—	Analog comparator input
	CMP1	—	CMOS	Comparator 1 output
RA4/T0CKI/CMP2	RA4	ST	OD	Bi-directional I/O port
	T0CKI	ST	—	External clock input for TMR0 or comparator output. Output is open drain type
	CMP2	—	OD	Comparator 2 output
RA5/$\overline{\text{MCLR}}$/VPP	RA5	ST	—	Input port
	$\overline{\text{MCLR}}$	ST	—	Master clear
	VPP	HV	—	Programming voltage input. When configured as $\overline{\text{MCLR}}$, this pin is an active low RESET to the device. Voltage on $\overline{\text{MCLR}}$/VPP must not exceed VDD during normal device operation
RA6/OSC2/CLKOUT	RA6	ST	CMOS	Bi-directional I/O port.
	OSC2	XTAL	—	Oscillator crystal output. Connects to crystal resonator in Crystal Oscillator mode.
	CLKOUT	—	CMOS	In ER/INTRC mode, OSC2 pin can output CLKOUT, which has 1/4 the frequency of OSC1
RA7/OSC1/CLKIN	RA7	ST	CMOS	Bi-directional I/O port
	OSC1	XTAL	—	Oscillator crystal input
	CLKIN	ST	—	External clock source input. ER biasing pin.

Legend: ST = Schmitt Trigger input HV = High Voltage OD = Open Drain AN = Analog

TABLE 5-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA(1)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on All Other RESETS
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx 0000	xxxu 0000
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	1111 1111
1Fh	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0000	0000 0000
9Fh	VRCON	VREN	VROE	VRR	—	VR3	VR2	VR1	VR0	000- 0000	000- 0000

Legend: — = Unimplemented locations, read as ‘0’, u = unchanged, x = unknown
Note 1: Shaded bits are not used by PORTA.

5.2 PORTB and TRISB Registers

PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. A '1' in the TRISB register puts the corresponding output driver in a Hi-impedance mode. A '0' in the TRISB register puts the contents of the output latch on the selected pin(s).

PORTB is multiplexed with the external interrupt, USART, CCP module and the TMR1 clock input/output. The standard port functions and the alternate port functions are shown in Table 5-3. Alternate port functions override TRIS setting when enabled.

Reading PORTB register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. So a write to a port implies that the port pins are first read, then this value is modified and written to the port data latch.

Each of the PORTB pins has a weak internal pull-up (≈200 μA typical). A single control bit can turn on all the pull-ups. This is done by clearing the $\overline{\text{RBPU}}$ (OPTION<7>) bit. The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on Power-on Reset.

Four of PORTB’s pins, RB<7:4>, have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB<7:4> pin configured as an output is excluded from the interrupt-on-change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The “mismatch” outputs of RB7:RB4 are OR’ed together to generate the RBIF interrupt (flag latched in INTCON<0>).

This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB. This will end the mismatch condition.
b) Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared.

This interrupt on mismatch feature, together with software configurable pull-ups on these four pins allow easy interface to a key pad and make it possible for wake-up on key-depression. (See AN552)

Note:	If a change on the I/O pin should occur when a read operation is being executed (start of the Q2 cycle), then the RBIF interrupt flag may not get set.

The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

FIGURE 5-8: BLOCK DIAGRAM OF RB0/INT PIN

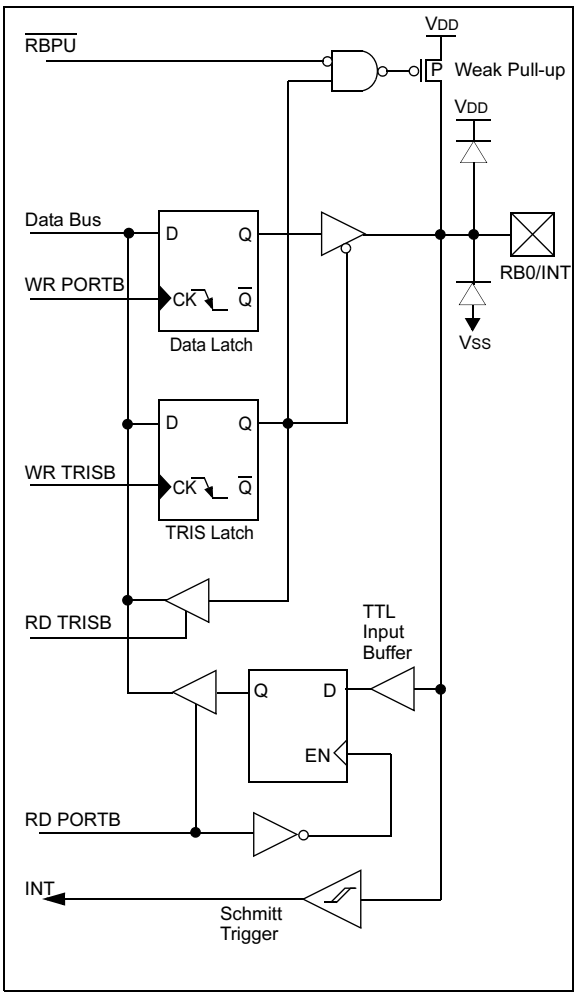
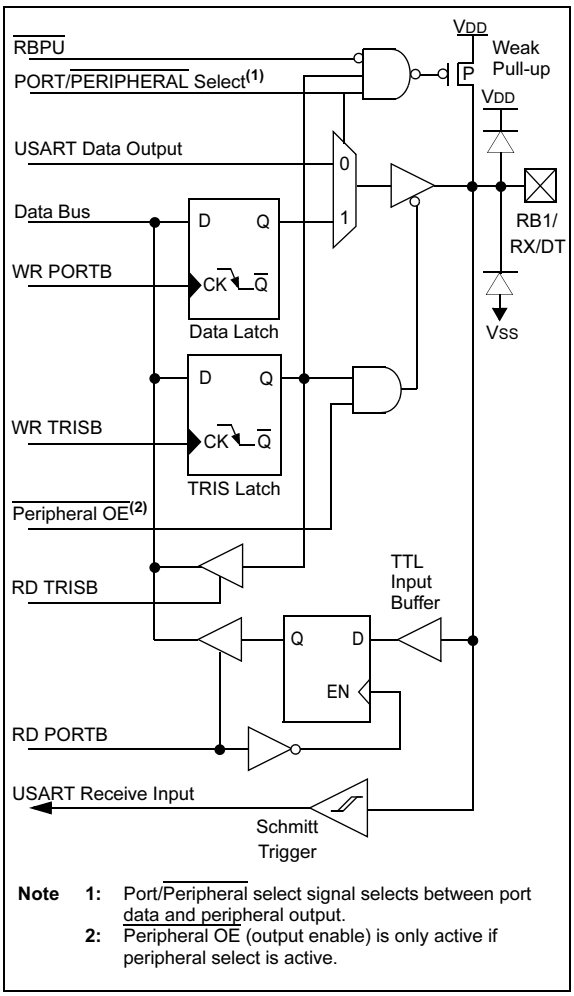
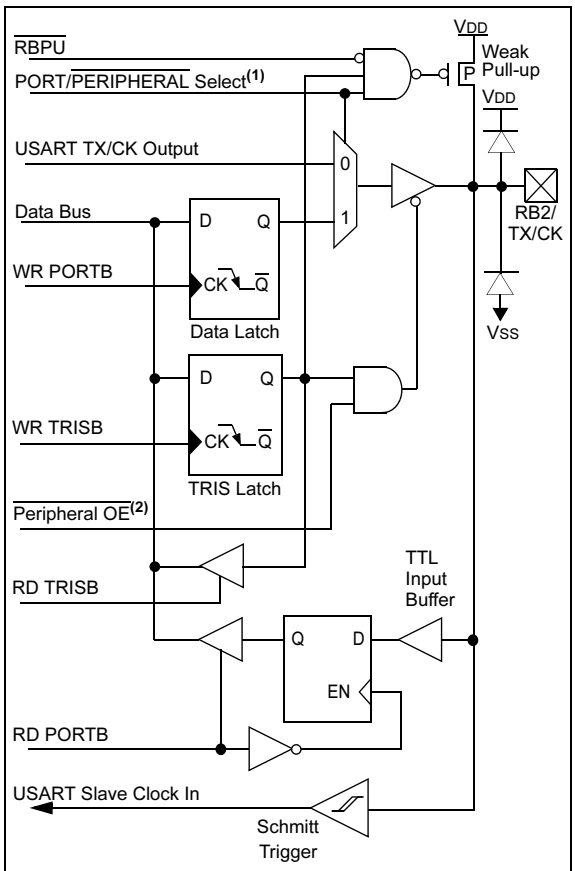


FIGURE 5-9: BLOCK DIAGRAM OF RB1/RX/DT PIN



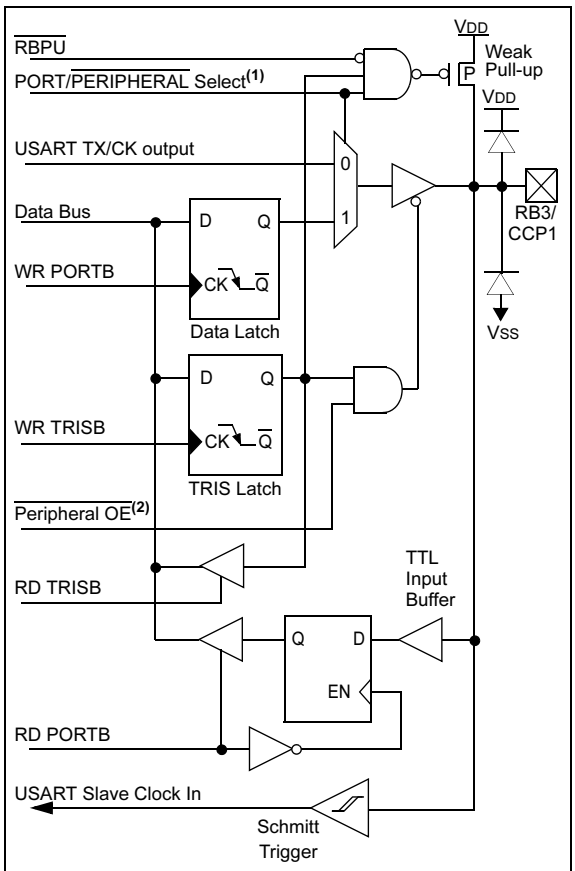
Note 1: Port/Peripheral select signal selects between port data and peripheral output.
2: Peripheral OE (output enable) is only active if peripheral select is active.

FIGURE 5-10: BLOCK DIAGRAM OF RB2/TX/CK PIN



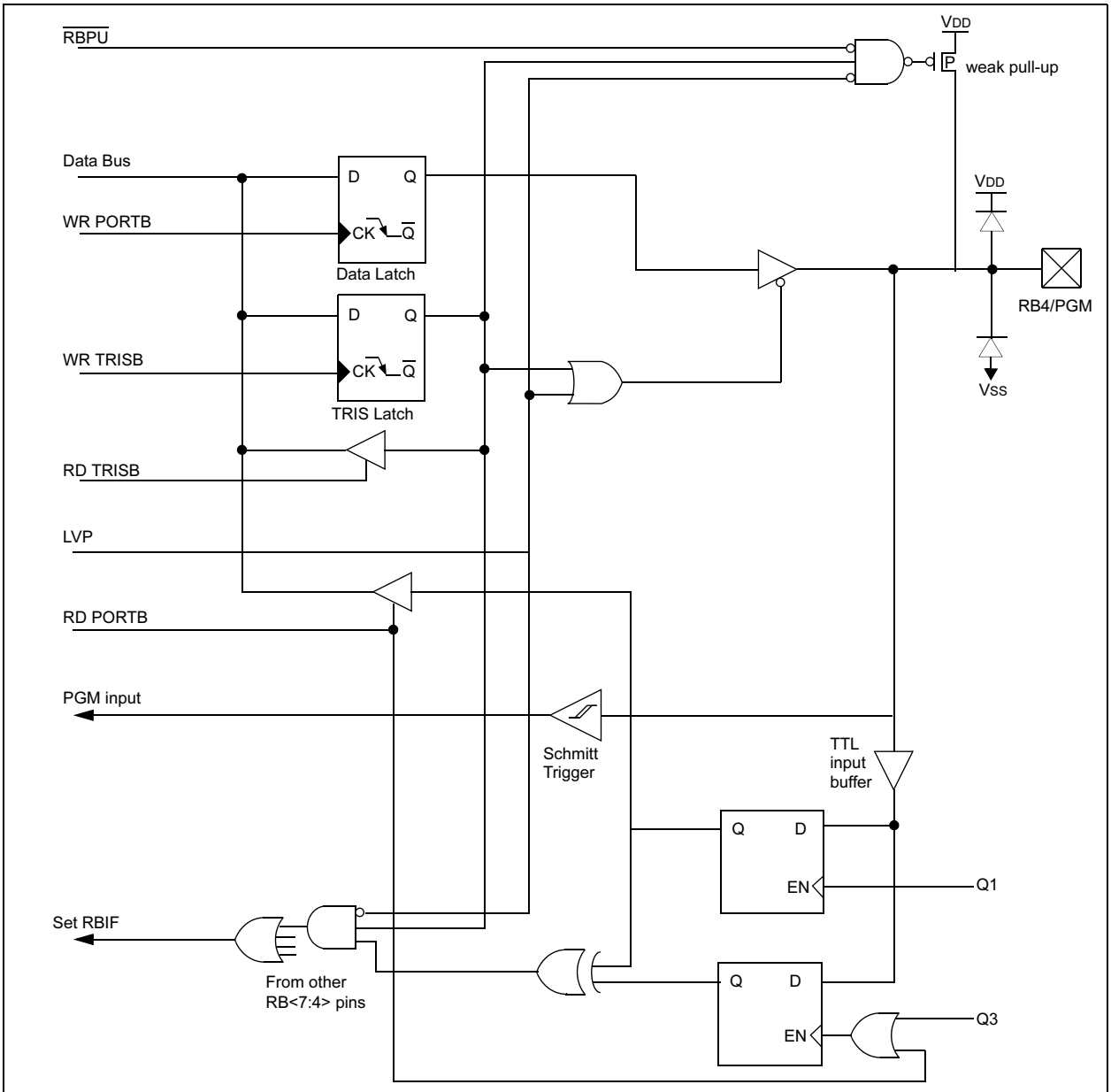
Note 1: Port/Peripheral select signal selects between port data and peripheral output.
2: Peripheral OE (output enable) is only active if peripheral select is active.

FIGURE 5-11: BLOCK DIAGRAM OF RB3/CCP1 PIN



Note 1: Port/Peripheral select signal selects between port data and peripheral output.
2: Peripheral OE (output enable) is only active if peripheral select is active.

FIGURE 5-12: BLOCK DIAGRAM OF RB4/PGM PIN



Note 1: The low voltage programming disables the interrupt-on-change and the weak pull-ups on RB4.

FIGURE 5-13: BLOCK DIAGRAM OF RB5 PIN

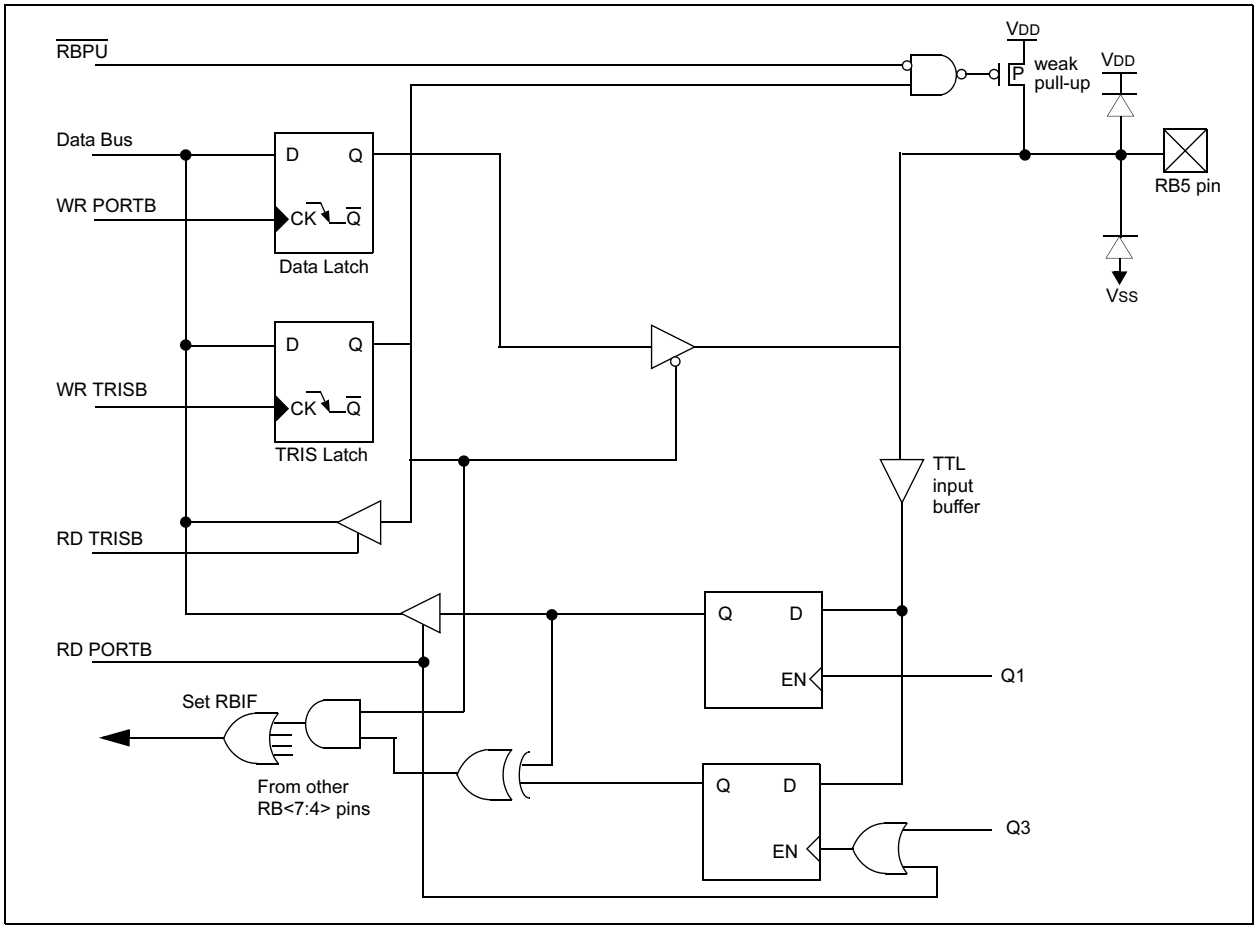


FIGURE 5-14: BLOCK DIAGRAM OF RB6/T1OSO/T1CKI PIN

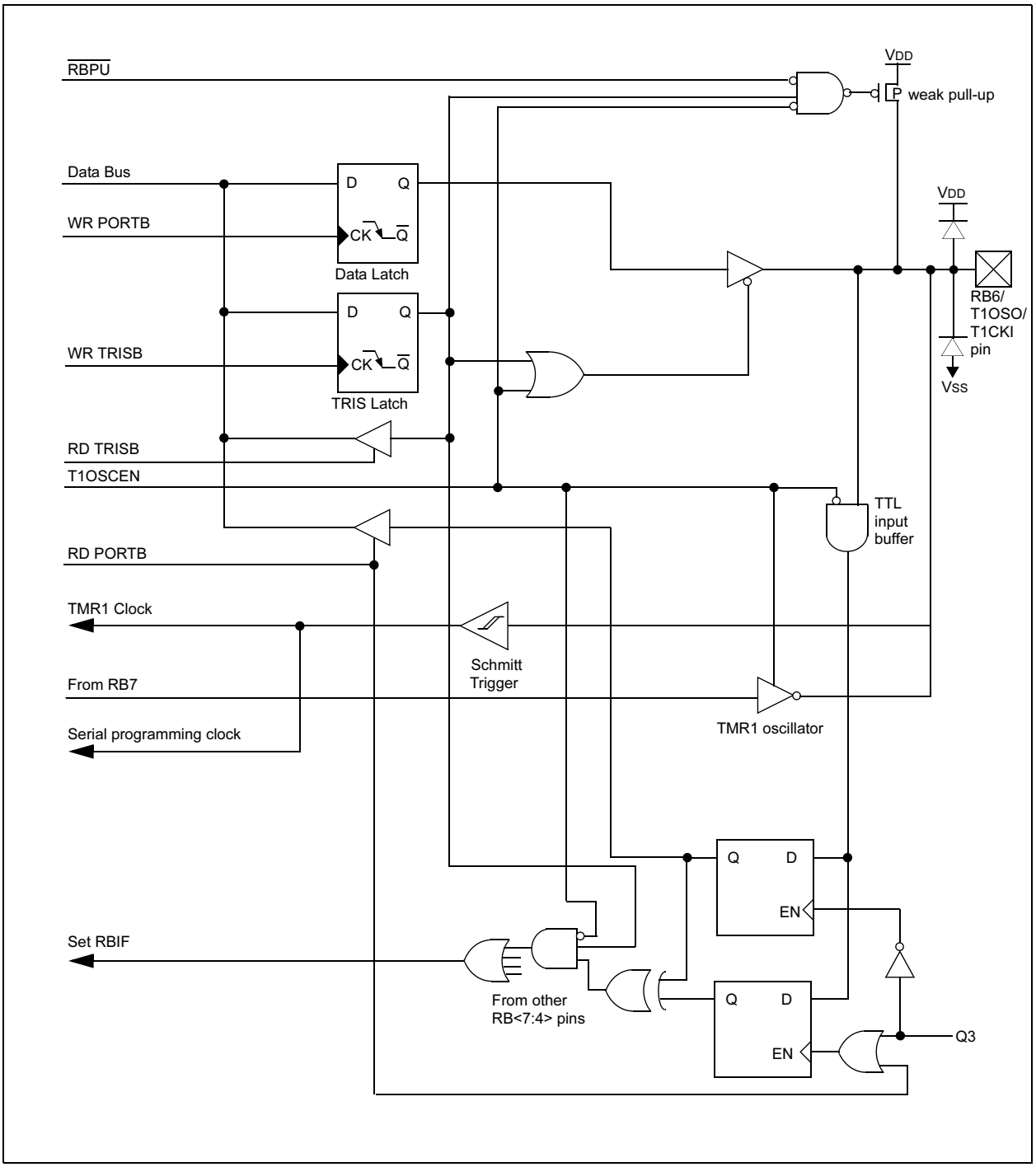


FIGURE 5-15: BLOCK DIAGRAM OF THE RB7/T1OSI PIN

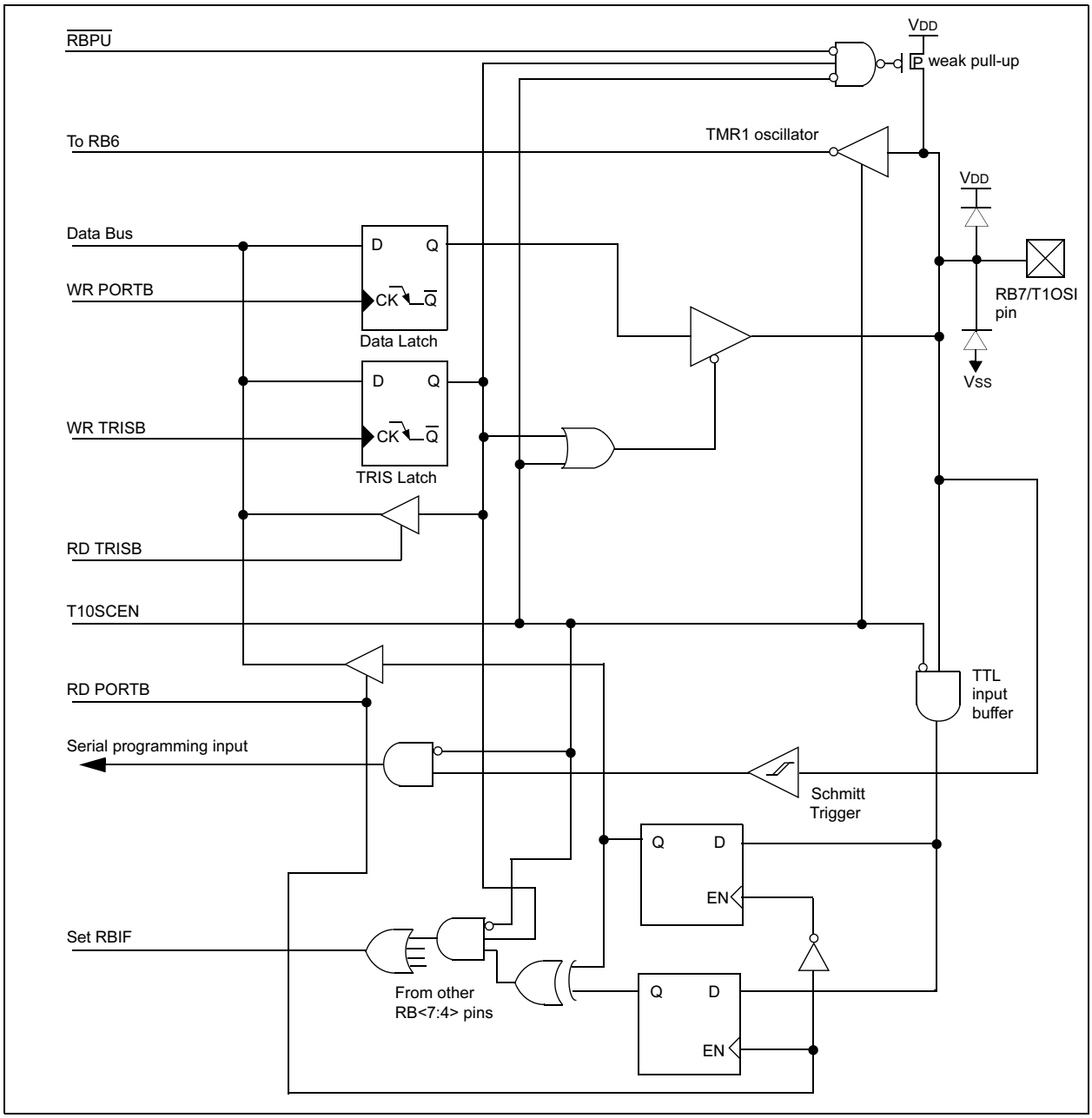


TABLE 5-3: PORTB FUNCTIONS

Name	Function	Input Type	Output Type	Description
RB0/INT	RB0	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	INT	ST	—	External interrupt.
RB1/RX/DT	RB1	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	RX	ST	—	USART Receive Pin
	DT	ST	CMOS	Synchronous data I/O
RB2/TX/CK	RB2	TTL	CMOS	Bi-directional I/O port
	TX	—	CMOS	USART Transmit Pin
	CK	ST	CMOS	Synchronous Clock I/O. Can be software programmed for internal weak pull-up.
RB3/CCP1	RB3	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	CCP1	ST	CMOS	Capture/Compare/PWM/I/O
RB4/PGM	RB4	TTL	CMOS	Bi-directional I/O port. Can be software programmed for internal weak pull-up.
	PGM	ST	—	Low voltage programming input pin. Interrupt-on-pin change. When low voltage programming is enabled, the interrupt-on-pin change and weak pull-up resistor are disabled.
RB5	RB5	TTL	CMOS	Bi-directional I/O port. Interrupt-on-pin change. Can be software programmed for internal weak pull-up.
RB6/T1OSO/T1CKI/PGC	RB6	TTL	CMOS	Bi-directional I/O port. Interrupt-on-pin change. Can be software programmed for internal weak pull-up.
	T1OSO	—	XTAL	Timer1 Oscillator Output
	T1CKI	ST	—	Timer1 Clock Input
	PGC	ST	—	ICSP Programming Clock
RB7/T1OSI/PGD	RB7	TTL	CMOS	Bi-directional I/O port. Interrupt-on-pin change. Can be software programmed for internal weak pull-up.
	T1OSI	XTAL	—	Timer1 Oscillator Input
	PGD	ST	CMOS	ICSP Data I/O

Legend: O = Output; — = Not used; TTL = TTL Input; CMOS = CMOS Output; I = Input; OD = Open Drain Output; P = Power; ST = Schmitt Trigger Input; AN = Analog

TABLE 5-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB(1)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on All Other RESETS
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h, 181h	OPTION	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: u = unchanged, x = unknown

Note 1: Shaded bits are not used by PORTB.

5.3 I/O Programming Considerations

5.3.1 BI-DIRECTIONAL I/O PORTS

Any instruction which writes, operates internally as a read followed by a write operation. The BCF and BSF instructions, for example, read the register into the CPU, execute the bit operation and write the result back to the register. Caution must be used when these instructions are applied to a port with both inputs and outputs defined. For example, a BSF operation on Bit 5 of PORTB will cause all eight bits of PORTB to be read into the CPU. Then the BSF operation takes place on Bit 5 and PORTB is written to the output latches. If another bit of PORTB is used as a bi-directional I/O pin (e.g., Bit 0) and it is defined as an input at this time, the input signal present on the pin itself would be read into the CPU and rewritten to the data latch of this particular pin, overwriting the previous content. As long as the pin stays in the Input mode, no problem occurs. However, if Bit 0 is switched into Output mode later on, the content of the data latch may now be unknown.

Reading a port register, reads the values of the port pins. Writing to the port register writes the value to the port latch. When using read-modify-write instructions (ex. BCF, BSF, etc.) on a port, the value of the port pins is read, the desired operation is done to this value, and this value is then written to the port latch.

Example 5-2 shows the effect of two sequential read-modify-write instructions (ex., BCF, BSF, etc.) on an I/O port.

A pin actively outputting a Low or High should not be driven from external devices at the same time in order to change the level on this pin ("wired-or", "wired-and"). The resulting high output currents may damage the chip.

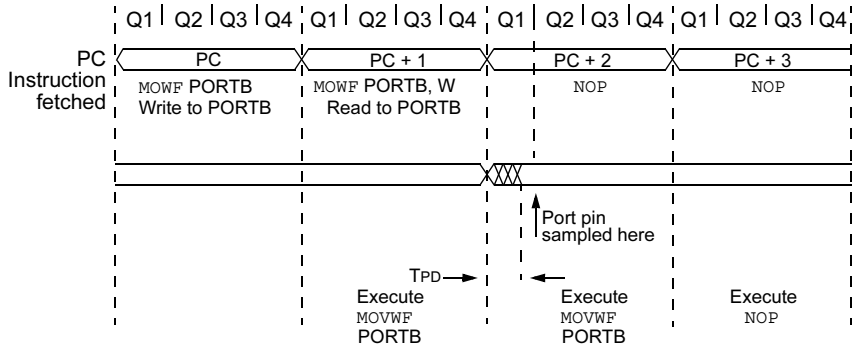
EXAMPLE 5-2: READ-MODIFY-WRITE INSTRUCTIONS ON AN I/O PORT

```
;Initial PORT settings:PORTB<7:4> Inputs
;
;                       PORTB<3:0> Outputs
;PORTB<7:6> have external pull-up and are not
;connected to other circuitry
;
;                       PORT latchPORT Pins
                        ---------- ----------
    BCF STATUS, RP0     ;
    BCF PORTB, 7        ;01pp pppp 11pp pppp
    BSF STATUS, RP0     ;
    BCF TRISB, 7        ;10pp pppp 11pp pppp
    BCF TRISB, 6        ;10pp pppp 10pp pppp
;
;Note that the user may have expected the pin
;values to be 00pp pppp. The 2nd BCF caused
;RB7 to be latched as the pin value (High).
```

5.3.2 SUCCESSIVE OPERATIONS ON I/O PORTS

The actual write to an I/O port happens at the end of an instruction cycle, whereas for reading, the data must be valid at the beginning of the instruction cycle (Figure 5-16). Therefore, care must be exercised if a write followed by a read operation is carried out on the same I/O port. The sequence of instructions should be such to allow the pin voltage to stabilize (load dependent) before the next instruction which causes that file to be read into the CPU is executed. Otherwise, the previous state of that pin may be read into the CPU rather than the new state. When in doubt, it is better to separate these instructions with a NOP or another instruction not accessing this I/O port.

FIGURE 5-16: SUCCESSIVE I/O OPERATION



Note 1: This example shows write to PORTB followed by a read from PORTB.
2: Data setup time = (0.25 TCY - TPD) where TCY = instruction cycle and TPD = propagation delay of Q1 cycle to output valid. Therefore, at higher clock frequencies, a write followed by a read may be problematic.

6.0 TIMER0 MODULE

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter
- Readable and writable
- 8-bit software programmable prescaler
- Internal or external clock select
- Interrupt on overflow from FFh to 00h
- Edge select for external clock

Figure 6-1 is a simplified block diagram of the Timer0 module. Additional information available in the PICmicro™ Mid-Range MCU Family Reference Manual, DS31010A.

Timer mode is selected by clearing the T0CS bit (OPTION<5>). In Timer mode, the TMR0 will increment every instruction cycle (without prescaler). If Timer0 is written, the increment is inhibited for the following two cycles. The user can work around this by writing an adjusted value to TMR0.

Counter mode is selected by setting the T0CS bit. In this mode Timer0 will increment either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the source edge (T0SE) control bit (OPTION<4>). Clearing the T0SE bit selects the rising edge. Restrictions on the external clock input are discussed in detail in Section 6.2.

The prescaler is shared between the Timer0 module and the Watchdog Timer. The prescaler assignment is controlled in software by the control bit PSA (OPTION<3>). Clearing the PSA bit will assign the prescaler to Timer0. The prescaler is not readable or writable. When the prescaler is assigned to the Timer0 module, prescale value of 1:2, 1:4,..., 1:256 are selectable. Section 6.3 details the operation of the prescaler.

6.1 TIMER0 Interrupt

Timer0 interrupt is generated when the TMR0 register timer/counter overflows from FFh to 00h. This overflow sets the T0IF bit. The interrupt can be masked by clearing the T0IE bit (INTCON<5>). The T0IF bit (INTCON<2>) must be cleared in software by the Timer0 module interrupt service routine before re-enabling this interrupt. The Timer0 interrupt cannot wake the processor from SLEEP since the timer is shut off during SLEEP.

6.2 Using Timer0 with External Clock

When an external clock input is used for Timer0, it must meet certain requirements. The external clock requirement is due to internal phase clock (TOSC) synchronization. Also, there is a delay in the actual incrementing of Timer0 after synchronization.

6.2.1 EXTERNAL CLOCK SYNCHRONIZATION

When no prescaler is used, the external clock input is the same as the prescaler output. The synchronization of T0CKI with the internal phase clocks is accomplished by sampling the prescaler output on the Q2 and Q4 cycles of the internal phase clocks (Figure 6-1). Therefore, it is necessary for T0CKI to be high for at least 2TOSC (and a small RC delay of 20 ns) and low for at least 2TOSC (and a small RC delay of 20 ns). Refer to the electrical specification of the desired device.

When a prescaler is used, the external clock input is divided by the asynchronous ripple-counter type prescaler so that the prescaler output is symmetrical. For the external clock to meet the sampling requirement, the ripple-counter must be taken into account. Therefore, it is necessary for T0CKI to have a period of at least 4TOSC (and a small RC delay of 40 ns) divided by the prescaler value. The only requirement on T0CKI high and low time is that they do not violate the minimum pulse width requirement of 10 ns. Refer to parameters 40, 41 and 42 in the electrical specification of the desired device. See Table 17-7.

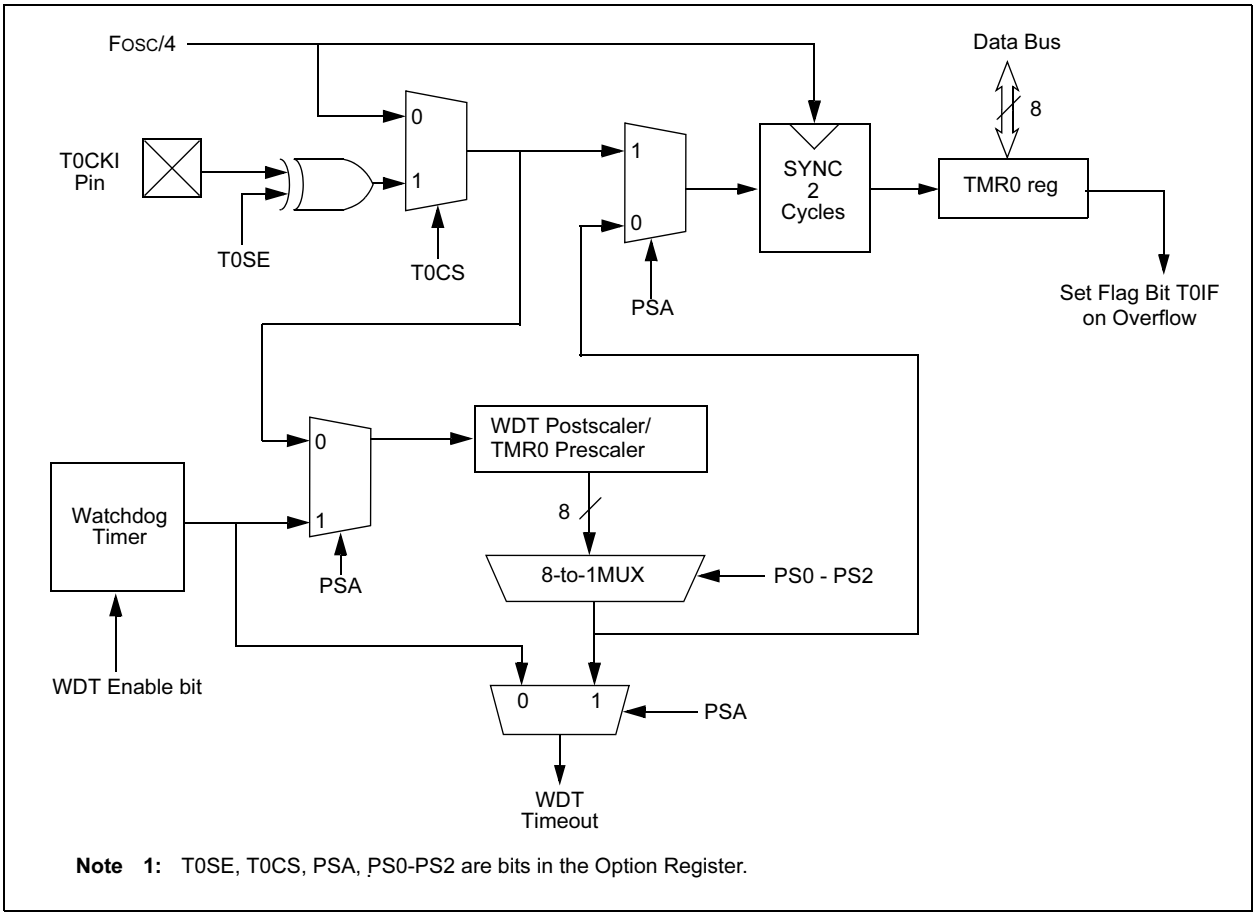
6.3 Timer0 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module, or as a postscaler for the Watchdog Timer. A prescaler assignment for the Timer0 module means that there is no postscaler for the Watchdog Timer, and vice-versa.

The PSA and PS2:PS0 bits (OPTION<3:0>) determine the prescaler assignment and prescale ratio.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., `CLRF 1, MOVWF 1, BSF 1, x....etc.`) will clear the prescaler. When assigned to WDT, a `CLRWDT` instruction will clear the prescaler along with the Watchdog Timer. The prescaler is not readable or writable.

FIGURE 6-1: BLOCK DIAGRAM OF THE TIMER0/WDT



Note 1: T0SE, T0CS, PSA, PS0-PS2 are bits in the Option Register.

6.3.1 SWITCHING PRESCALER ASSIGNMENT

The prescaler assignment is fully under software control (i.e., it can be changed “on the fly” during program execution). Use the instruction sequences, shown in Example 6-1, when changing the prescaler assignment from Timer0 to WDT, to avoid an unintended device RESET.

EXAMPLE 6-1: CHANGING PRESCALER (TIMER0→WDT)

```
BCF     STATUS, RP0   ;Skip if already in
                      ;Bank 0
CLRWDT                ;Clear WDT
CLRF    TMR0          ;Clear TMR0 & Prescaler
BSF     STATUS, RP0   ;Bank 1
MOVLW   '00101111’b   ;These 3 lines
                      ;(5, 6, 7)
MOVWF   OPTION_REG    ;are required only
                      ;if desired PS<2:0>
                      ;are
CLRWDT                ;000 or 001
MOVLW   '00101xxx’b   ;Set Postscaler to
MOVWF   OPTION_REG    ;desired WDT rate
BCF     STATUS, RP0   ;Return to Bank 0
```

To change prescaler from the WDT to the TMR0 module use the sequence shown in Example 6-2. This precaution must be taken even if the WDT is disabled.

EXAMPLE 6-2: CHANGING PRESCALER (WDT→TIMER0)

```
CLRWDT                ;Clear WDT and
                      ;prescaler
BSF     STATUS, RP0
MOVLW   b'xxxx0xxx'   ;Select TMR0, new
                      ;prescale value and
                      ;clock source
MOVWF   OPTION_REG
BCF     STATUS, RP0
```

TABLE 6-1: REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on All Other RESETS
01h	TMR0	Timer0 module register								xxxx xxxx	uuuu uuuu
0Bh/8Bh/ 10Bh/18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
81h, 181h	OPTION(2)	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	1111 1111

Legend: — = Unimplemented locations, read as ‘0’, u = unchanged, x = unknown

Note 1: Shaded bits are not used by TMR0 module.

2: Option is referred by `OPTION_REG` in MPLAB.

7.0 TIMER1 MODULE

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 Interrupt, if enabled, is generated on overflow which is latched in interrupt flag bit TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a timer
- As a counter

The Operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle. In Counter mode, it increments on every rising edge of the external clock input.

Timer1 can be enabled/disabled by setting/clearing control bit TMR1ON (T1CON<0>).

Timer1 also has an internal “RESET input”. This RESET can be generated by the CCP module (Section 11.0). Register 7-1 shows the Timer1 Control register.

For the PIC16F627 and PIC16F628, when the Timer1 oscillator is enabled (T1OSCEN is set), the RB7/T1OSI and RB6/T1OSO/T1CKI pins become inputs. That is, the TRISB<7:6> value is ignored.

REGISTER 7-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS: 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut off(1)

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Control bit
TMR1CS = 1
1 = Do not synchronize external clock input
0 = Synchronize external clock input
TMR1CS = 0
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RB6/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON**: Timer1 On bit
1 = Disables Timer1
0 = Stops Timer1

Note 1: The oscillator inverter and feedback resistor are turned off to eliminate power drain.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

7.1 Timer1 Operation in Timer Mode

Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is FOSC/4. The synchronize control bit $\overline{\text{T1SYNC}}$ (T1CON<2>) has no effect since the internal clock is always in sync.

7.2 Timer1 Operation in Synchronized Counter Mode

Counter mode is selected by setting bit TMR1CS. In this mode the timer increments on every rising edge of clock input on pin RB7/T1OSI when bit T1OSCEN is set or pin RB6/T1OSO/T1CKI when bit T1OSCEN is cleared.

If $\overline{\text{T1SYNC}}$ is cleared, then the external clock input is synchronized with internal phase clocks. The synchronization is done after the prescaler stage. The prescaler stage is an asynchronous ripple-counter.

In this configuration, during SLEEP mode, Timer1 will not increment even if the external clock is present, since the synchronization circuit is shut off. The prescaler however will continue to increment.

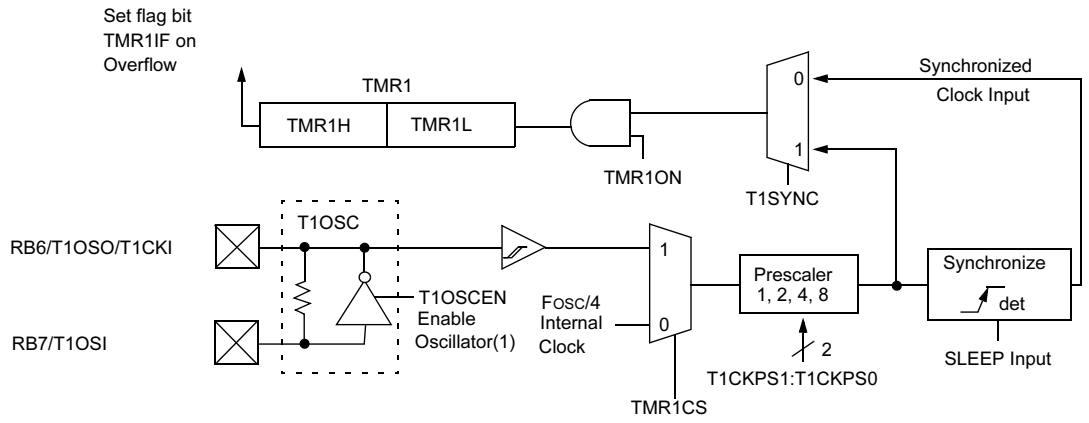
7.2.1 EXTERNAL CLOCK INPUT TIMING FOR SYNCHRONIZED COUNTER MODE

When an external clock input is used for Timer1 in Synchronized Counter mode, it must meet certain requirements. The external clock requirement is due to internal phase clock (Tosc) synchronization. Also, there is a delay in the actual incrementing of TMR1 after synchronization.

When the prescaler is 1:1, the external clock input is the same as the prescaler output. The synchronization of T1CKI with the internal phase clocks is accomplished by sampling the prescaler output on the Q2 and Q4 cycles of the internal phase clocks. Therefore, it is necessary for T1CKI to be high for at least 2Tosc (and a small RC delay of 20 ns) and low for at least 2Tosc (and a small RC delay of 20 ns). Refer to the appropriate electrical specifications, parameters 45, 46, and 47.

When a prescaler other than 1:1 is used, the external clock input is divided by the asynchronous ripple-counter type prescaler so that the prescaler output is symmetrical. In order for the external clock to meet the sampling requirement, the ripple-counter must be taken into account. Therefore, it is necessary for T1CKI to have a period of at least 4Tosc (and a small RC delay of 40 ns) divided by the prescaler value. The only requirement on T1CKI high and low time is that they do not violate the minimum pulse width requirements of 10 ns). Refer to the appropriate electrical specifications, parameters 40, 42, 45, 46, and 47.

FIGURE 7-1: TIMER1 BLOCK DIAGRAM



Note 1: When the T1OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

7.3 Timer1 Operation in Asynchronous Counter Mode

If control bit $\overline{\text{T1SYNC}}$ (T1CON<2>) is set, the external clock input is not synchronized. The timer continues to increment asynchronous to the internal phase clocks. The timer will continue to run during SLEEP and can generate an interrupt on overflow which will wake-up the processor. However, special precautions in software are needed to read/write the timer (Section 7.3.2).

In Asynchronous Counter mode, Timer1 can not be used as a time-base for capture or compare operations.

7.3.1 EXTERNAL CLOCK INPUT TIMING WITH UNSYNCHRONIZED CLOCK

If control bit $\overline{\text{T1SYNC}}$ is set, the timer will increment completely asynchronously. The input clock must meet certain minimum high-time and low-time requirements. Refer to the appropriate Electrical Specifications section, Timing Parameters 45, 46, and 47.

7.3.2 READING AND WRITING TIMER1 IN ASYNCHRONOUS COUNTER MODE

Reading TMR1H or TMR1L while the timer is running, from an external asynchronous clock, will ensure a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values itself poses certain problems since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers while the register is incrementing. This may produce an unpredictable value in the timer register.

Reading the 16-bit value requires some care. Example 7-1 is an example routine to read the 16-bit timer value. This is useful if the timer cannot be stopped.

EXAMPLE 7-1: READING A 16-BIT FREE-RUNNING TIMER

```
; All interrupts are disabled
   MOVF   TMR1H, W  ;Read high byte
   MOVWF  TMPH      ;
   MOVF   TMR1L, W  ;Read low byte
   MOVWF  TMPL      ;
   MOVF   TMR1H, W  ;Read high byte
   SUBWF  TMPH,  W  ;Sub 1st read
                    ; with 2nd read
   BTFSC  STATUS,Z  ;Is result = 0
   GOTO   CONTINUE  ;Good 16-bit read
;
; TMR1L may have rolled over between the read
; of the high and low bytes. Reading the high
; and low bytes now will read a good value.
;
   MOVF   TMR1H, W  ;Read high byte
   MOVWF  TMPH      ;
   MOVF   TMR1L, W  ;Read low byte
   MOVWF  TMPL      ;
; Re-enable the Interrupts (if required)
CONTINUE            ;Continue with your code
```


7.4 Timer1 Oscillator

A crystal oscillator circuit is built in between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting control bit T1OSCEN (T1CON<3>). The oscillator is a low power oscillator rated up to 200 kHz. It will continue to run during SLEEP. It is primarily intended for a 32 kHz crystal. Table 7-1 shows the capacitor selection for the Timer1 oscillator.

The Timer1 oscillator is identical to the LP oscillator. The user must provide a software time delay to ensure proper oscillator start-up.

TABLE 7-1: CAPACITOR SELECTION FOR THE TIMER1 OSCILLATOR

Osc Type	Freq	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF

Note 1: These values are for design guidance only. Consult AN826 (DS00826A) for further information on Crystal/Capacitor Selection.

7.5 Resetting Timer1 Using a CCP Trigger Output

If the CCP1 module is configured in Compare mode to generate a “special event trigger” (CCP1M3:CCP1M0 = `1011`), this signal will reset Timer1.

Note: The special event triggers from the CCP1 module will not set interrupt flag bit TMR1IF (PIR1<0>).

Timer1 must be configured for either Timer or Synchronized Counter mode to take advantage of this feature. If Timer1 is running in Asynchronous Counter mode, this RESET operation may not work.

In the event that a write to Timer1 coincides with a special event trigger from CCP1, the write will take precedence.

In this mode of operation, the CCPRxH:CCPRxL registers pair effectively becomes the period register for Timer1.

7.6 Resetting of Timer1 Register Pair (TMR1H, TMR1L)

TMR1H and TMR1L registers are not reset to 00h on a POR or any other RESET except by the CCP1 special event triggers.

T1CON register is reset to 00h on a Power-on Reset or a Brown-out Reset, which shuts off the timer and leaves a 1:1 prescale. In all other RESETS, the register is unaffected.

7.7 Timer1 Prescaler

The prescaler counter is cleared on writes to the TMR1H or TMR1L registers.

TABLE 7-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Bh/8Bh/ 10Bh/18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	`0000 000x`	`0000 000u`
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	`0000 -000`	`0000 -000`
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	`0000 -000`	`0000 -000`
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 register								`xxxx xxxx`	`uuuu uuuu`
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 register								`xxxx xxxx`	`uuuu uuuu`
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	`--00 0000`	`--uu uuuu`

Legend: `x` = unknown, `u` = unchanged, `-` = unimplemented read as '0'. Shaded cells are not used by the Timer1 module.

8.0 TIMER2 MODULE

Timer2 is an 8-bit timer with a prescaler and a postscaler. It can be used as the PWM time-base for PWM mode of the CCP module. The TMR2 register is readable and writable, and is cleared on any device RESET.

The input clock (Fosc/4) has a prescale option of 1:1, 1:4 or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>).

The Timer2 module has an 8-bit Period Register PR2. Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register. The PR2 register is initialized to FFh upon RESET.

The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a TMR2 interrupt (latched in flag bit TMR2IF, (PIR1<1>)).

Timer2 can be shut off by clearing control bit TMR2ON (T2CON<2>) to minimize power consumption.

Register 8-1 shows the Timer2 Control register.

8.1 Timer2 Prescaler and Postscaler

The prescaler and postscaler counters are cleared when any of the following occurs:

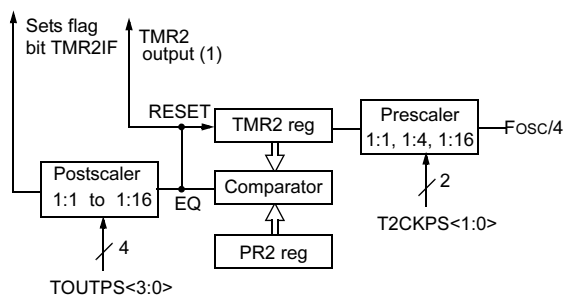
- A write to the TMR2 register
- A write to the T2CON register
- Any device RESET (Power-on Reset, $\overline{\text{MCLR}}$ Reset, Watchdog Timer Reset, or Brown-out Reset)

TMR2 is not cleared when T2CON is written.

8.2 Output of TMR2

The output of TMR2 (before the postscaler) is fed to the Synchronous Serial Port module which optionally uses it to generate shift clock.

FIGURE 8-1: TIMER2 BLOCK DIAGRAM



Note 1: TMR2 register output can be software selected by the SSP Module as a baud clock.

REGISTER 8-1: T2CON: TIMER CONTROL REGISTER (ADDRESS: 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
`0000` = 1:1 Postscale Value
`0001` = 1:2 Postscale Value
•
•
•
`1111` = 1:16 Postscale

bit 2 **TMR2ON**: Timer2 On bit
`1` = Timer2 is on
`0` = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
`00` = 1:1 Prescaler Value
`01` = 1:4 Prescaler Value
`1x` = 1:16 Prescaler Value

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

TABLE 8-1: REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Bh/8Bh/ 10Bh/18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	`0000 000x`	`0000 000u`
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	`0000 -000`	`0000 -000`
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	`0000 -000`	`0000 -000`
11h	TMR2	Timer2 module’s register								`0000 0000`	`0000 0000`
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	`-000 0000`	`-000 0000`
92h	PR2	Timer2 Period Register								`1111 1111`	`1111 1111`

Legend: `x` = unknown, `u` = unchanged, `-` = unimplemented read as '0'. Shaded cells are not used by the Timer2 module.

NOTES:

9.0 COMPARATOR MODULE

The Comparator module contains two analog comparators. The inputs to the comparators are multiplexed with the RA0 through RA3 pins. The On-chip Voltage Reference (Section 10.0) can also be an input to the comparators.

The CMCON register, shown in Register 9-1, controls the comparator input and output multiplexers. A block diagram of the comparator is shown in Figure 9-1.

REGISTER 9-1: CMCON REGISTER (ADDRESS: 01Fh)

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

bit 7 **C2OUT**: Comparator 2 Output
When C2INV = 0:
1 = C2 V_{IN}+ > C2 V_{IN}-
0 = C2 V_{IN}+ < C2 V_{IN}-

When C2INV = 1:
1 = C2 V_{IN}+ < C2 V_{IN}-
0 = C2 V_{IN}+ > C2 V_{IN}-

bit 6 **C1OUT**: Comparator 1 Output
When C1INV = 0:
1 = C1 V_{IN}+ > C1 V_{IN}-
0 = C1 V_{IN}+ < C1 V_{IN}-

When C1INV = 1:
1 = C1 V_{IN}+ < C1 V_{IN}-
0 = C1 V_{IN}+ > C1 V_{IN}-

bit 5 **C2INV**: Comparator 2 Output Inversion
1 = C2 Output inverted
0 = C2 Output not inverted

bit 4 **C1INV**: Comparator 1 Output Inversion
1 = C1 Output inverted
0 = C1 Output not inverted

bit 3 **CIS**: Comparator Input Switch
When CM2:CM0: = 001
Then:
1 = C1 V_{IN}- connects to RA3
0 = C1 V_{IN}- connects to RA0

When CM2:CM0 = 010
Then:
1 = C1 V_{IN}- connects to RA3
C2 V_{IN}- connects to RA2
0 = C1 V_{IN}- connects to RA0
C2 V_{IN}- connects to RA1

bit 2-0 **CM2:CM0**: Comparator Mode
Figure 9-1 shows the Comparator modes and CM2:CM0 bit settings

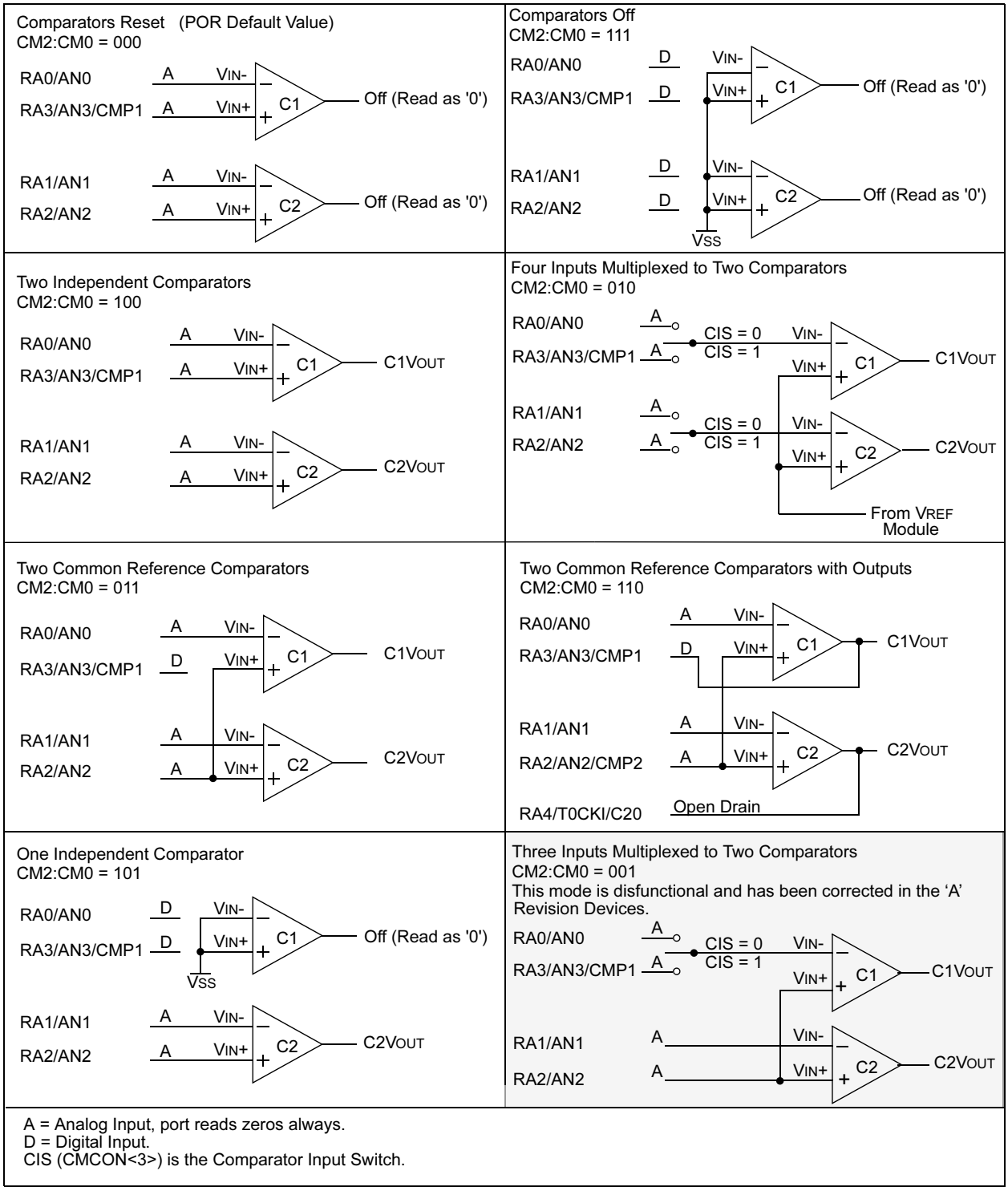
Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

9.1 Comparator Configuration

There are eight modes of operation for the comparators. The CMCON register is used to select the mode. Figure 9-1 shows the eight possible modes. The TRISA register controls the data direction of the comparator pins for each mode. If the Comparator mode is changed, the comparator output level may not be valid for the specified mode change delay shown in Table 17-1.

> **Note:** Comparator interrupts should be disabled during a Comparator mode change otherwise a false interrupt may occur.

FIGURE 9-1: COMPARATOR I/O OPERATING MODES



The code example in Example 9-1 depicts the steps required to configure the Comparator module. RA3 and RA4 are configured as digital output. RA0 and RA1 are configured as the V- inputs and RA2 as the V+ input to both comparators.

EXAMPLE 9-1: INITIALIZING COMPARATOR MODULE

```
FLAG_REG EQU          0X20
CLRF     FLAG_REG     ;Init flag register
CLRF     PORTA        ;Init PORTA
MOVF     CMCON, W     ;Load comparator bits
ANDLW    0xC0         ;Mask comparator bits
IORWF    FLAG_REG,F   ;Store bits in flag register
MOVLW    0x03         ;Init comparator mode
MOVWF    CMCON        ;CM<2:0> = 011
BSF      STATUS,RP0   ;Select Bank1
MOVLW    0x07         ;Initialize data direction
MOVWF    TRISA        ;Set RA<2:0> as inputs
                      ;RA<4:3> as outputs
                      ;TRISA<7:5> always read '0'
BCF      STATUS,RP0   ;Select Bank 0
CALL     DELAY10      ;10µs delay
MOVF     CMCON,F      ;Read CMCON to end change condition
BCF      PIR1,CMIF    ;Clear pending interrupts
BSF      STATUS,RP0   ;Select Bank 1
BSF      PIE1,CMIE    ;Enable comparator interrupts
BCF      STATUS,RP0   ;Select Bank 0
BSF      INTCON,PEIE  ;Enable peripheral interrupts
BSF      INTCON,GIE   ;Global interrupt enable
```

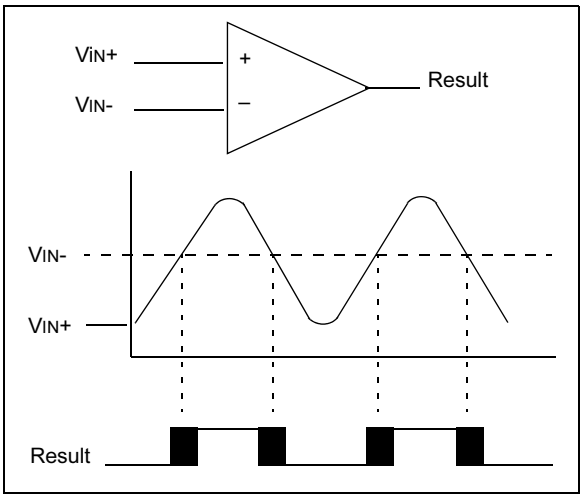
9.2 Comparator Operation

A single comparator is shown in Figure 9-2 along with the relationship between the analog input levels and the digital output. When the analog input at VIN+ is less than the analog input VIN-, the output of the comparator is a digital low level. When the analog input at VIN+ is greater than the analog input VIN-, the output of the comparator is a digital high level. The shaded areas of the output of the comparator in Figure 9-2 represent the uncertainty due to input offsets and response time.

9.3 Comparator Reference

An external or internal reference signal may be used depending on the Comparator Operating mode. The analog signal that is present at VIN- is compared to the signal at VIN+, and the digital output of the comparator is adjusted accordingly (Figure 9-2).

FIGURE 9-2: SINGLE COMPARATOR



9.3.1 EXTERNAL REFERENCE SIGNAL

When external voltage references are used, the comparator module can be configured to have the comparators operate from the same or different reference sources. However, threshold detector applications may require the same reference. The reference signal must be between VSS and VDD, and can be applied to either pin of the comparator(s).

9.3.2 INTERNAL REFERENCE SIGNAL

The Comparator module also allows the selection of an internally generated voltage reference for the comparators. Section 10.0, Voltage Reference Manual, contains a detailed description of the Voltage Reference module that provides this signal. The internal reference signal is used when the comparators are in mode CM<2:0>=010 (Figure 9-1). In this mode, the internal voltage reference is applied to the VIN+ pin of both comparators.

9.4 Comparator Response Time

Response time is the minimum time, after selecting a new reference voltage or input source, before the comparator output is ensured to have a valid level. If the internal reference is changed, the maximum delay of the internal voltage reference must be considered when using the comparator outputs. Otherwise the maximum delay of the comparators should be used (Table 17-1**).**

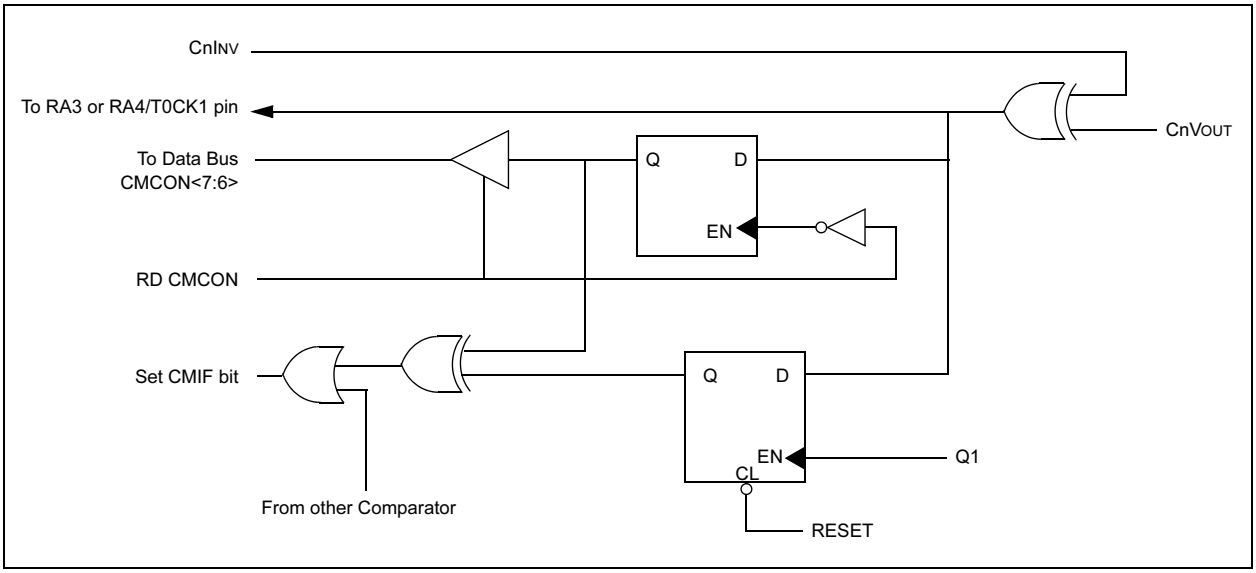
9.5 Comparator Outputs

The comparator outputs are read through the CMCON register. These bits are read only. The comparator outputs may also be directly output to the RA3 and RA4 I/O pins. When the CM<2:0> = 110, multiplexors in the output path of the RA3 and RA4/T0CK1 pins will switch and the output of each pin will be the unsynchronized output of the comparator. The uncertainty of each of the comparators is related to the input offset voltage and the response time given in the specifications. Figure 9-3 shows the comparator output block diagram.

The TRISA bits will still function as an output enable/disable for the RA3 and RA4/T0CK1 pins while in this mode.

Note 1:	When reading the PORT register, all pins configured as analog inputs will read as a '0'. Pins configured as digital inputs will convert an analog input according to the Schmitt Trigger input specification.
2:	Analog levels on any pin that is defined as a digital input may cause the input buffer to consume more current than is specified.

FIGURE 9-3: COMPARATOR OUTPUT BLOCK DIAGRAM



9.6 Comparator Interrupts

The Comparator Interrupt flag is set whenever there is a change in the output value of either comparator. Software will need to maintain information about the status of the output bits, as read from CMCON<7:6>, to determine the actual change that has occurred. The CMIF bit, PIR1<6>, is the Comparator Interrupt Flag. The CMIF bit must be RESET by clearing ‘0’. Since it is also possible to write a '1' to this register, a simulated interrupt may be initiated.

The CMIE bit (PIE1<6>) and the PEIE bit (INTCON<6>) must be set to enable the interrupt. In addition, the GIE bit must also be set. If any of these bits are clear, the interrupt is not enabled, though the CMIF bit will still be set if an interrupt condition occurs.

Note:	If a change in the CMCON register (C1OUT or C2OUT) should occur when a read operation is being executed (start of the Q2 cycle), then the CMIF (PIR1<6>) interrupt flag may not get set.

The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Any write or read of CMCON. This will end the mismatch condition.

b) Clear flag bit CMIF.

A mismatch condition will continue to set flag bit CMIF. Reading CMCON will end the mismatch condition, and allow flag bit CMIF to be cleared.

9.7 Comparator Operation During SLEEP

When a comparator is active and the device is placed in SLEEP mode, the comparator remains active and the interrupt is functional if enabled. This interrupt will wake-up the device from SLEEP mode when enabled. While the comparator is powered-up, higher SLEEP currents than shown in the power-down current specification will occur. Each comparator that is operational will consume additional current as shown in the comparator specifications. To minimize power consumption while in SLEEP mode, turn off the comparators, CM<2:0> = 111, before entering SLEEP. If the device wakes-up from SLEEP, the contents of the CMCON register are not affected.

9.8 Effects of a RESET

A device RESET forces the CMCON register to its RESET state. This forces the Comparator module to be in the comparator RESET mode, CM2:CM0 = 000. This ensures that all potential inputs are analog inputs. Device current is minimized when analog inputs are present at RESET time. The comparators will be powered-down during the RESET interval.

9.9 Analog Input Connection Considerations

A simplified circuit for an analog input is shown in Figure 9-4. Since the analog pins are connected to a digital output, they have reverse biased diodes to V_{DD} and V_{SS}. The analog input therefore, must be between V_{SS} and V_{DD}. If the input voltage deviates from this range by more than 0.6V in either direction, one of the diodes is forward biased and a latchup may occur. A maximum source impedance of 10 kΩ is recommended for the analog sources. Any external component connected to an analog input pin, such as a capacitor or a Zener diode, should have very little leakage current.

FIGURE 9-4: ANALOG INPUT MODE

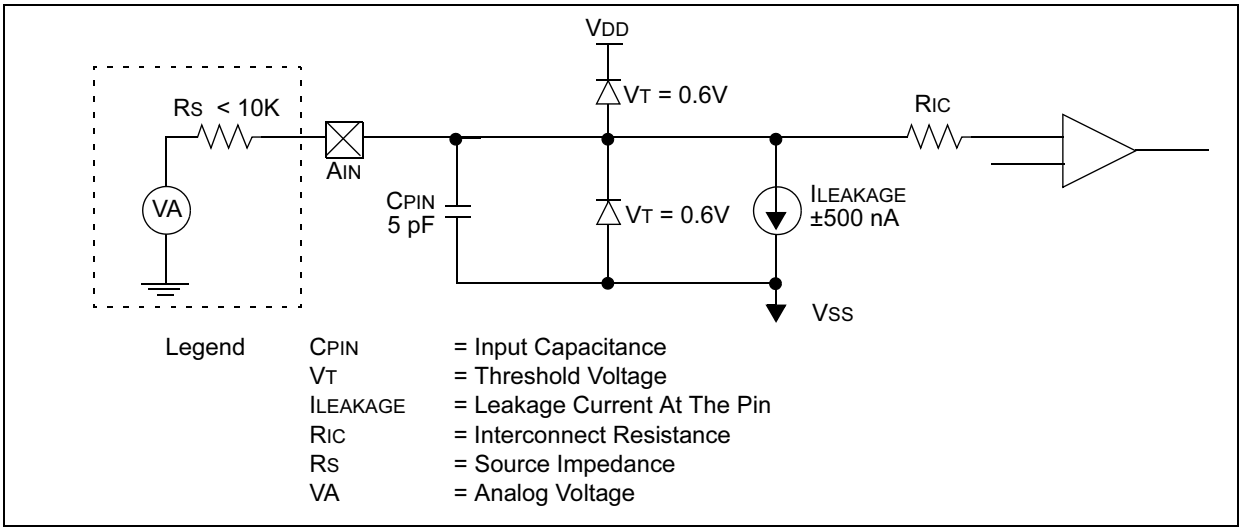


TABLE 9-1: REGISTERS ASSOCIATED WITH COMPARATOR MODULE

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on All Other RESETS
1Fh	CMCON	C2OUT	C1OUT	C2INV	C1NV	CIS	CM2	CM1	CM0	0000 0000	0000 0000
0Bh/8Bh/ 10Bh/18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	1111 1111

Legend: x = Unknown, u = Unchanged, - = Unimplemented, read as ‘0’

10.0 VOLTAGE REFERENCE MODULE

The Voltage Reference is a 16-tap resistor ladder network that provides a selectable voltage reference. The resistor ladder is segmented to provide two ranges of VREF values and has a power-down function to conserve power when the reference is not being used. The VRCON register controls the operation of the reference as shown in Figure 10-1. The block diagram is given in Figure 10-1.

10.1 Configuring the Voltage Reference

The Voltage Reference can output 16 distinct voltage levels for each range.

The equations used to calculate the output of the Voltage Reference are as follows:

if VRR = 1: VREF = (VR<3:0>/24) x VDD

if VRR = 0: VREF = (VDD x 1/4) + (VR<3:0>/32) x VDD

The setting time of the Voltage Reference must be considered when changing the VREF output (Table 17-2). Example 10-1 shows an example of how to configure the Voltage Reference for an output voltage of 1.25V with VDD = 5.0V.

REGISTER 10-1: VRCON REGISTER (ADDRESS: 9Fh)

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
VREN	VROE	VRR	—	VR3	VR2	VR1	VR0
bit 7							bit 0

bit 7 **VREN:** VREF Enable
1 = VREF circuit powered on
0 = VREF circuit powered down, no IDD drain

bit 6 **VROE:** VREF Output Enable
1 = VREF is output on RA2 pin
0 = VREF is disconnected from RA2 pin

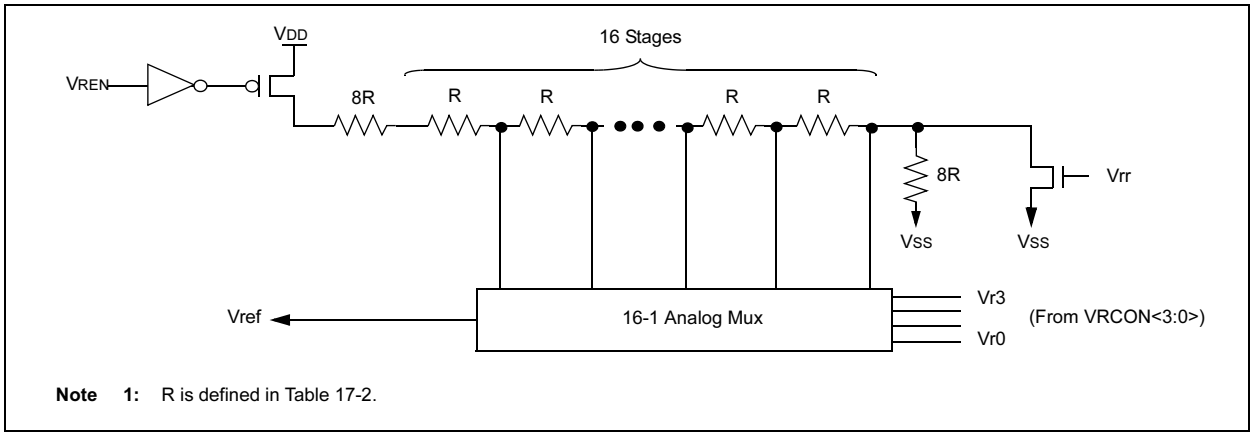
bit 5 **VRR:** VREF Range selection
1 = Low Range
0 = High Range

bit 4 **Unimplemented:** Read as '0'

bit 3-0 **VR<3:0>**: VREF value selection 0 ≤ VR [3:0] ≤ 15
When VRR = 1: VREF = (VR<3:0>/ 24) * VDD
When VRR = 0: VREF = 1/4 * VDD + (VR<3:0>/ 32) * VDD

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

FIGURE 10-1: VOLTAGE REFERENCE BLOCK DIAGRAM



Note 1: R is defined in Table 17-2.

EXAMPLE 10-1: VOLTAGE REFERENCE CONFIGURATION

```
MOVLW   0x02        ; 4 Inputs Muxed
MOVWF   CMCON       ; to 2 comps.
BSF     STATUS,RP0  ; go to Bank 1
MOVLW   0x07        ; RA3-RA0 are
MOVWF   TRISA       ; outputs
MOVLW   0xA6        ; enable VREF
MOVWF   VRCON       ; low range
                    ; set VR<3:0>=6
BCF     STATUS,RP0  ; go to Bank 0
CALL    DELAY10     ; 10µs delay
```

10.2 Voltage Reference Accuracy/Error

The full range of VSS to VDD cannot be realized due to the construction of the module. The transistors on the top and bottom of the resistor ladder network (Figure 10-1) keep VREF from approaching VSS or VDD. The Voltage Reference is VDD derived and therefore, the VREF output changes with fluctuations in VDD. The tested absolute accuracy of the Voltage Reference can be found in Table 17-2.

10.3 Operation During SLEEP

When the device wakes-up from SLEEP through an interrupt or a Watchdog Timer timeout, the contents of the VRCON register are not affected. To minimize current consumption in SLEEP mode, the Voltage Reference should be disabled.

10.4 Effects of a RESET

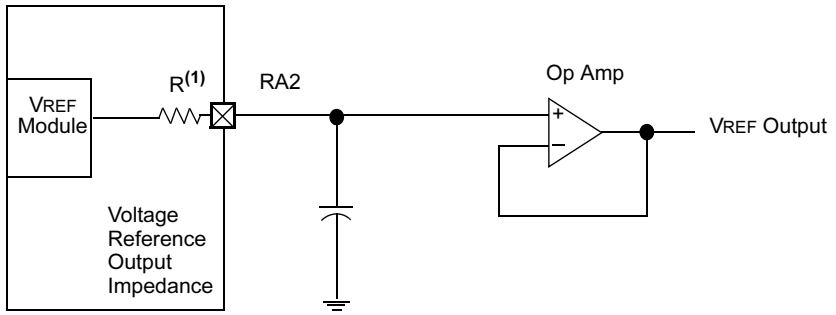
A device RESET disables the Voltage Reference by clearing bit VREN (VRCON<7>). This RESET also disconnects the reference from the RA2 pin by clearing bit VROE (VRCON<6>) and selects the high voltage range by clearing bit VRR (VRCON<5>). The VREF value select bits, VRCON<3:0>, are also cleared.

10.5 Connection Considerations

The Voltage Reference module operates independently of the Comparator module. The output of the reference generator may be connected to the RA2 pin if the TRISA<2> bit is set and the VROE bit, VRCON<6>, is set. Enabling the Voltage Reference output onto the RA2 pin with an input signal present will increase current consumption. Connecting RA2 as a digital output with VREF enabled will also increase current consumption.

The RA2 pin can be used as a simple D/A output with limited drive capability. Due to the limited drive capability, a buffer must be used in conjunction with the Voltage Reference output for external connections to VREF. Figure 10-2 shows an example buffering technique.

FIGURE 10-2: VOLTAGE REFERENCE OUTPUT BUFFER EXAMPLE



Note 1: R is dependent upon the Voltage Reference Configuration VRCON<3:0> and VRCON<5>.

TABLE 10-1: REGISTERS ASSOCIATED WITH VOLTAGE REFERENCE

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value On POR	Value On All Other RESETS
9Fh	VRCON	VREN	VROE	VRR	—	VR3	VR2	VR1	VR0	000- 0000	000- 0000
1Fh	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0000	0000 0000
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	1111 1111

Note 1: — = Unimplemented, read as ‘0’.

11.0 CAPTURE/COMPARE/PWM (CCP) MODULE

The CCP (Capture/Compare/PWM) module contains a 16-bit register which can operate as a 16-bit capture register, as a 16-bit compare register or as a PWM master/slave Duty Cycle register. Table 11-1 shows the timer resources of the CCP Module modes.

CCP1 Module

Capture/Compare/PWM Register1 (CCPR1) is comprised of two 8-bit registers: CCPR1L (low byte) and CCPR1H (high byte). The CCP1CON register controls the operation of CCP1. All are readable and writable.

Additional information on the CCP module is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

TABLE 11-1: CCP MODE - TIMER RESOURCE

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

REGISTER 11-1: CCP1CON REGISTER (ADDRESS: 17h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCP1X:CCP1Y**: PWM Least Significant bits
Capture Mode: Unused
Compare Mode: Unused
PWM Mode: These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCP1M3:CCP1M0**: CCPx Mode Select bits
`0000` = Capture/Compare/PWM off (resets CCP1 module)
`0100` = Capture mode, every falling edge
`0101` = Capture mode, every rising edge
`0110` = Capture mode, every 4th rising edge
`0111` = Capture mode, every 16th rising edge
`1000` = Compare mode, set output on match (CCP1IF bit is set)
`1001` = Compare mode, clear output on match (CCP1IF bit is set)
`1010` = Compare mode, generate software interrupt on match (CCP1IF bit is set, CCP1 pin is unaffected)
`1011` = Compare mode, trigger special event (CCP1IF bit is set; CCP1 resets TMR1
`11xx` = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

11.1 Capture Mode

In Capture mode, CCPR1H:CCPR1L captures the 16-bit value of the TMR1 register when an event occurs on pin RB3/CCP1. An event is defined as:

- Every falling edge
- Every rising edge
- Every 4th rising edge
- Every 16th rising edge

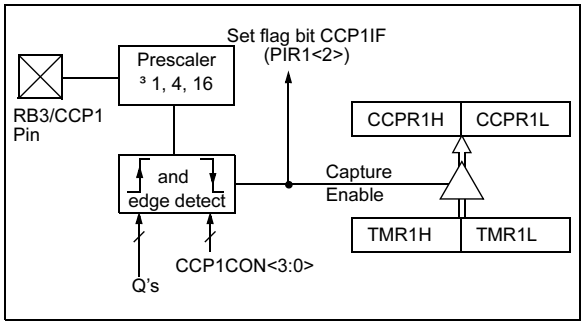
An event is selected by control bits CCP1M3:CCP1M0 (CCP1CON<3:0>). When a capture is made, the Interrupt Request Flag bit CCP1IF (PIR1<2>) is set. It must be cleared in software. If another capture occurs before the value in register CCPR1 is read, the old captured value will be lost.

11.1.1 CCP PIN CONFIGURATION

In Capture mode, the RB3/CCP1 pin should be configured as an input by setting the TRISB<3> bit.

Note:	If the RB3/CCP1 is configured as an output, a write to the port can cause a capture condition.

TABLE 11-2: CAPTURE MODE OPERATION BLOCK DIAGRAM



11.1.2 TIMER1 MODE SELECTION

Timer1 must be running in Timer mode or Synchronized Counter mode for the CCP module to use the capture feature. In Asynchronous Counter mode, the capture operation may not work.

11.1.3 SOFTWARE INTERRUPT

When the Capture mode is changed, a false capture interrupt may be generated. The user should keep bit CCP1IE (PIE1<2>) clear to avoid false interrupts and should clear the flag bit CCP1IF following any such change in Operating mode.

11.1.4 CCP PRESCALER

There are four prescaler settings, specified by bits CCP1M3:CCP1M0. Whenever the CCP module is turned off, or the CCP module is not in Capture mode, the prescaler counter is cleared. This means that any RESET will clear the prescaler counter.

Switching from one capture prescaler to another may generate an interrupt. Also, the prescaler counter will not be cleared, therefore the first capture may be from a non-zero prescaler. Example 11-1 shows the recommended method for switching between capture prescalers. This example also clears the prescaler counter and will not generate the “false” interrupt.

EXAMPLE 11-1: CHANGING BETWEEN CAPTURE PRESCALERS

```
CLRF    CCP1CON       ;Turn CCP module off
MOVLW   NEW_CAPT_PS   ;Load the W reg with
                      ; the new prescaler
                      ; mode value and CCP ON
MOVWF   CCP1CON       ;Load CCP1CON with this
                      ; value
```

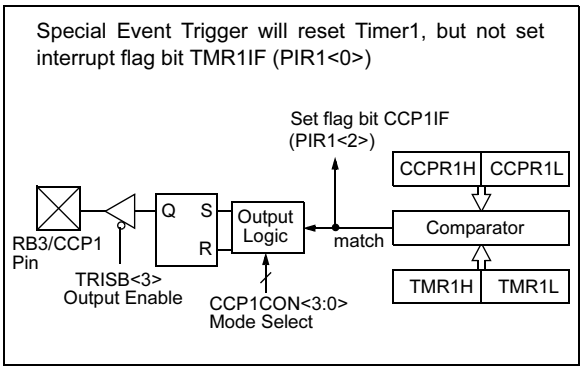
11.2 Compare Mode

In Compare mode, the 16-bit CCPR1 register value is constantly compared against the TMR1 register pair value. When a match occurs, the RB3/CCP1 pin is:

- Driven High
- Driven Low
- Remains Unchanged

The action on the pin is based on the value of control bits CCP1M3:CCP1M0 (CCP1CON<3:0>). At the same time, interrupt flag bit CCP1IF is set.

FIGURE 11-1: COMPARE MODE OPERATION BLOCK DIAGRAM



11.2.1 CCP PIN CONFIGURATION

The user must configure the RB3/CCP1 pin as an output by clearing the TRISB<3> bit.

Note:	Clearing the CCP1CON register will force the RB3/CCP1 compare output latch to the default low level. This is not the data latch.

11.2.2 TIMER1 MODE SELECTION

Timer1 must be running in Timer mode or Synchronized Counter mode if the CCP module is using the compare feature. In Asynchronous Counter mode, the compare operation may not work.

11.2.3 SOFTWARE INTERRUPT MODE

When generate software interrupt is chosen, the CCP1 pin is not affected. Only a CCP interrupt is generated (if enabled).

11.2.4 SPECIAL EVENT TRIGGER

In this mode, an internal hardware trigger is generated which may be used to initiate an action.

The special event trigger output of CCP1 resets the TMR1 register pair. This allows the CCPR1 register to effectively be a 16-bit programmable period register for Timer1.

TABLE 11-3: REGISTERS ASSOCIATED WITH CAPTURE, COMPARE, AND TIMER1

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Bh/8Bh/ 10Bh/ 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
8Ch	PIE1	EEIE	CMIF	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
87h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Capture/Compare/PWM register1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM register1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are not used by Capture and Timer1.

11.3 PWM Mode

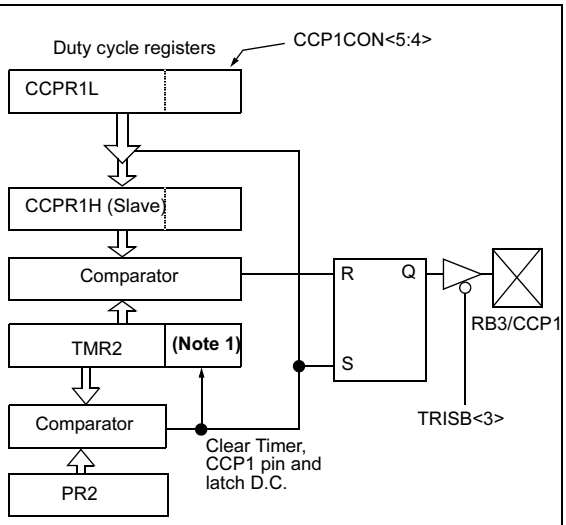
In Pulse Width Modulation (PWM) mode, the CCP1 pin produces up to a 10-bit resolution PWM output. Since the CCP1 pin is multiplexed with the PORTB data latch, the TRISB<3> bit must be cleared to make the CCP1 pin an output.

Note:	Clearing the CCP1CON register will force the CCP1 PWM output latch to the default low level. This is not the PORTB I/O data latch.

Figure 11-2 shows a simplified block diagram of the CCP module in PWM mode.

For a step-by-step procedure on how to set up the CCP module for PWM operation, see Section 11.3.3.

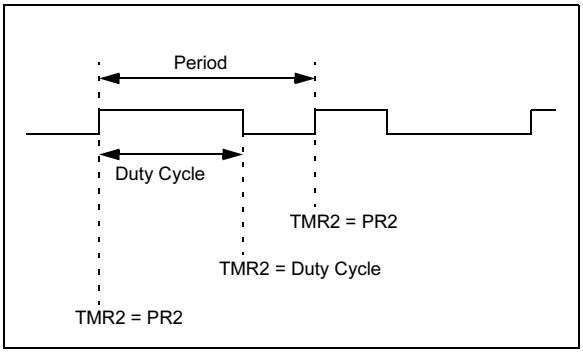
FIGURE 11-2: SIMPLIFIED PWM BLOCK DIAGRAM



Note 1: 8-bit timer is concatenated with the 2-bit internal Q clock, or 2 bits of the prescaler to create 10-bit time-base.

A PWM output (Figure 11-3) has a time-base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

FIGURE 11-3: PWM OUTPUT



11.3.1 PWM PERIOD

The PWM period is specified by writing to the PR2register. The PWM period can be calculated using the following formula:

$$\text{PWM period} = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 prescale value})$$

PWM frequency is defined as 1 / [PWM period].

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCP1 pin is set (exception: if PWM duty cycle = 0%, the CCP1 pin will not be set)
- The PWM duty cycle is latched from CCPR1L into CCPR1H

Note:	The Timer2 postscaler (see Section 8.0) is not used in the determination of the PWM frequency. The postscaler could be used to have an interrupt occur at a different frequency than the PWM output.

11.3.2 PWM DUTY CYCLE

The PWM duty cycle is specified by writing to the CCPR1L register and to the CCP1CON<5:4> bits. Up to 10-bit resolution is available: the CCPR1L contains the eight MSbs and the CCP1CON<5:4> contains the two LSbs. This 10-bit value is represented by CCPR1L:CCP1CON<5:4>. The following equation is used to calculate the PWM duty cycle in time:

EQUATION 11-1: PWM DUTY CYCLE

PWM duty cycle = (CCPR1L:CCP1CON<5:4>) • Tosc • (TMR2 prescale value)

CCPR1L and CCP1CON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPR1H until after a match between PR2 and TMR2 occurs (i.e., the period is complete). In PWM mode, CCPR1H is a read-only register.

The CCPR1H register and a 2-bit internal latch are used to double buffer the PWM duty cycle. This double buffering is essential for glitchless PWM operation.

When the CCPR1H and 2-bit latch match TMR2 concatenated with an internal 2-bit Q clock or 2 bits of the TMR2 prescaler, the CCP1 pin is cleared.

Maximum PWM resolution (bits) for a given PWM frequency:

EQUATION 11-2: MAXIMUM PWM RESOLUTION

$$\text{PWM Resolution} = \frac{\log\left(\frac{\text{Fosc}}{\text{Fpwm x TMR2 Prescaler}}\right)}{\log(2)} \text{ bits}$$

Note: If the PWM duty cycle value is longer than the PWM period, the CCP1 pin will not be cleared.

For an example on the PWM period and duty cycle calculation, see the PICmicro™ Mid-Range Reference Manual (DS33023).

11.3.3 SET-UP FOR PWM OPERATION

The following steps should be taken when configuring the CCP module for PWM operation:

1. Set the PWM period by writing to the PR2 register.
2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
3. Make the CCP1 pin an output by clearing the TRISB<3> bit.
4. Set the TMR2 prescale value and enable Timer2 by writing to T2CON.

TABLE 11-4: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS AT 20 MHz

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.5

TABLE 11-5: REGISTERS ASSOCIATED WITH PWM AND TIMER2

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Bh/8Bh/ 10Bh/18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
87h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
11h	TMR2	Timer2 module's register								0000 0000	0000 0000
92h	PR2	Timer2 module's period register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	uuuu
15h	CCPR1L	Capture/Compare/PWM register1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM register1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are not used by PWM and Timer2.

NOTES:

12.0 UNIVERSAL SYNCHRONOUS/ ASYNCHRONOUS RECEIVER/ TRANSMITTER (USART) MODULE

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. (USART is also known as a Serial Communications Interface or SCI). The USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices such as CRT terminals and personal computers, or it can be configured as a half duplex synchronous system that can communicate with peripheral devices such as A/D or D/A integrated circuits, Serial EEPROMs etc.

The USART can be configured in the following modes:

- Asynchronous (full duplex)
- Synchronous - Master (half duplex)
- Synchronous - Slave (half duplex)

Bit SPEN (RCSTA<7>), and bits TRISB<2:1>, have to be set in order to configure pins RB2/TX/CK and RB1/RX/DT as the Universal Synchronous Asynchronous Receiver Transmitter.

REGISTER 12-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS: 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC**: Clock Source Select bit
Asynchronous mode
Don't care
Synchronous mode
1 = Master mode (Clock generated internally from BRG)
0 = Slave mode (Clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit(1)
1 = Transmit enabled
0 = Transmit disabled

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode
1 = High speed
0 = Low speed
Synchronous mode
Unused in this mode

bit 1 **TRMT**: Transmit Shift Register STATUS bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D**: 9th bit of transmit data. Can be PARITY bit.

Note 1: SREN/CREN overrides TXEN in SYNC mode.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 12-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS: 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN**: Serial Port Enable bit
(Configures RB1/RX/DT and RB2/TX/CK pins as serial port pins when bits TRISB<2:17> are set)
1 = Serial port enabled
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
Asynchronous mode:
Don't care
Synchronous mode - master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode - slave:
Unused in this mode

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enable interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received, and ninth bit can be used as PARITY bit
Asynchronous mode 8-bit (RX9=0):
Unused in this mode
Synchronous mode
Unused in this mode

bit 2 **FERR**: Framing Error bit
1 = Framing error (Can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (Can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D**: 9th bit of received data (Can be PARITY bit)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

12.1 USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode bit BRGH (TXSTA<2>) also controls the baud rate. In Synchronous mode bit BRGH is ignored. Table 12-1 shows the formula for computation of the baud rate for different USART modes which only apply in Master mode (internal clock).

Given the desired baud rate and Fosc, the nearest integer value for the SPBRG register can be calculated using the formula in Table 12-1. From this, the error in baud rate can be determined.

Example 12-1 shows the calculation of the baud rate error for the following conditions:

FOSC = 16 MHz

Desired Baud Rate = 9600

BRGH = 0

SYNC = 0

EXAMPLE 12-1: CALCULATING BAUD RATE ERROR

Desired Baud rate = FOSC / (64(X + 1))

9600 = 16000000 / (64(+1))X
X = î25.042°

Calculated Baud Rate = 16000000 / (64(25 + 1))
= 9615

$$\text{Error} = \frac{(\text{Calculated Baud Rate} = \text{Desired Baud Rate})}{\text{Desired Baud Rate}}$$

= (9615 - 9600)/ 9600
= 0.16%

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks. This is because the FOSC/(16(X + 1)) equation can reduce the baud rate error in some cases.

Writing a new value to the SPBRG register, causes the BRG timer to be RESET (or cleared), this ensures the BRG does not wait for a timer overflow before outputting the new baud rate.

TABLE 12-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64(X+1))	Baud Rate= FOSC/(16(X+1))
1	(Synchronous) Baud Rate = FOSC/(4(X+1))	NA

Legend: X = value in SPBRG (0 to 255)

TABLE 12-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'.
Shaded cells are not used by the BRG.

TABLE 12-3: BAUD RATES FOR SYNCHRONOUS MODE

BAUD RATE (K)	Fosc = 20 MHz		SPBRG value (decimal)	16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
0.3	NA	—	—	NA	—	—	NA	—	—
1.2	NA	—	—	NA	—	—	NA	—	—
2.4	NA	—	—	NA	—	—	NA	—	—
9.6	NA	—	—	NA	—	—	9.766	+1.73%	255
19.2	19.53	+1.73%	255	19.23	+0.16%	207	19.23	+0.16%	129
76.8	76.92	+0.16%	64	76.92	+0.16%	51	75.76	-1.36%	32
96	96.15	+0.16%	51	95.24	-0.79%	41	96.15	+0.16%	25
300	294.1	-1.96	16	307.69	+2.56%	12	312.5	+4.17%	7
500	500	0	9	500	0	7	500	0	4
HIGH	5000	—	0	4000	—	0	2500	—	0
LOW	19.53	—	255	15.625	—	255	9.766	—	255

BAUD RATE (K)	Fosc = 7.15909 MHz		SPBRG value (decimal)	5.0688 MHz		SPBRG value (decimal)	4 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
0.3	NA	—	—	NA	—	—	NA	—	—
1.2	NA	—	—	NA	—	—	NA	—	—
2.4	NA	—	—	NA	—	—	NA	—	—
9.6	9.622	+0.23%	185	9.6	0	131	9.615	+0.16%	103
19.2	19.24	+0.23%	92	19.2	0	65	19.231	+0.16%	51
76.8	77.82	+1.32	22	79.2	+3.13%	15	75.923	+0.16%	12
96	94.20	-1.88	18	97.48	+1.54%	12	1000	+4.17%	9
300	298.3	-0.57	5	316.8	5.60%	3	NA	—	—
500	NA	—	—	NA	—	—	NA	—	—
HIGH	1789.8	—	0	1267	—	0	100	—	0
LOW	6.991	—	255	4.950	—	255	3.906	—	255

BAUD RATE (K)	Fosc = 3.579545 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
0.3	NA	—	—	NA	—	—	0.303	+1.14%	26
1.2	NA	—	—	1.202	+0.16%	207	1.170	-2.48%	6
2.4	NA	—	—	2.404	+0.16%	103	NA	—	—
9.6	9.622	+0.23%	92	9.615	+0.16%	25	NA	—	—
19.2	19.04	-0.83%	46	19.24	+0.16%	12	NA	—	—
76.8	74.57	-2.90%	11	83.34	+8.51%	2	NA	—	—
96	99.43	+3.57%	8	NA	—	—	NA	—	—
300	298.3	0.57%	2	NA	—	—	NA	—	—
500	NA	—	—	NA	—	—		—	—
HIGH	894.9	—	0	250	—	0	8.192	—	0
LOW	3.496	—	255	0.9766	—	255	0.032	—	255

TABLE 12-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	FOSC = 20 MHz		SPBRG value (decimal)	16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
0.3	NA	—	—	NA	—	—	NA	—	—
1.2	1.221	+1.73%	255	1.202	+0.16%	207	1.202	+0.16%	129
2.4	2.404	+0.16%	129	2.404	+0.16%	103	2.404	+0.16%	64
9.6	9.469	-1.36%	32	9.615	+0.16%	25	9.766	+1.73%	15
19.2	19.53	+1.73%	15	19.23	+0.16%	12	19.53	+1.73V	7
76.8	78.13	+1.73%	3	83.33	+8.51%	2	78.13	+1.73%	1
96	104.2	+8.51%	2	NA	—	—	NA	—	—
300	312.5	+4.17%	0	NA	—	—	NA	—	—
500	NA	—	—	NA	—	—	NA	—	—
HIGH	312.5	—	0	250	—	0	156.3	—	0
LOW	1.221	—	255	0.977	—	255	0.6104	—	255

BAUD RATE (K)	FOSC = 7.15909 MHz		SPBRG value (decimal)	5.0688 MHz		SPBRG value (decimal)	4 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
0.3	NA	—	—	0.31	+3.13%	255	0.3005	-0.17%	207
1.2	1.203	+0.23%	92	1.2	0	65	1.202	+1.67%	51
2.4	2.380	-0.83%	46	2.4	0	32	2.404	+1.67%	25
9.6	9.322	-2.90%	11	9.9	+3.13%	7	NA	—	—
19.2	18.64	-2.90%	5	19.8	+3.13%	3	NA	—	—
76.8	NA	—	—	79.2	+3.13%	0	NA	—	—
96	NA	—	—	NA	—	—	NA	—	—
300	NA	—	—	NA	—	—	NA	—	—
500	NA	—	—	NA	—	—	NA	—	—
HIGH	111.9	—	0	79.2	—	0	62.500	—	0
LOW	0.437	—	255	0.3094	—	255	3.906	—	255

BAUD RATE (K)	FOSC = 3.579545 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
0.3	0.301	+0.23%	185	0.300	+0.16%	51	0.256	-14.67%	1
1.2	1.190	-0.83%	46	1.202	+0.16%	12	NA	—	—
2.4	2.432	+1.32%	22	2.232	-6.99%	6	NA	—	—
9.6	9.322	-2.90%	5	NA	—	—	NA	—	—
19.2	18.64	-2.90%	2	NA	—	—	NA	—	—
76.8	NA	—	—	NA	—	—	NA	—	—
96	NA	—	—	NA	—	—	NA	—	—
300	NA	—	—	NA	—	—	NA	—	—
500	NA	—	—	NA	—	—	NA	—	—
HIGH	55.93	—	0	15.63	—	0	0.512	—	0
LOW	0.2185	—	255	0.0610	—	255	0.0020	—	255

TABLE 12-5: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 20 MHz		SPBRG value (decimal)	16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
9600	9.615	+0.16%	129	9.615	+0.16%	103	9.615	+0.16%	64
19200	19.230	+0.16%	64	19.230	+0.16%	51	18.939	-1.36%	32
38400	37.878	-1.36%	32	38.461	+0.16%	25	39.062	+1.7%	15
57600	56.818	-1.36%	21	58.823	+2.12%	16	56.818	-1.36%	10
115200	113.636	-1.36%	10	111.111	-3.55%	8	125	+8.51%	4
250000	250	0	4	250	0	3	NA	—	—
625000	625	0	1	NA	—	—	625	0	0
1250000	1250	0	0	NA	—	—	NA	—	—

BAUD RATE (K)	Fosc = 7.16 MHz		SPBRG value (decimal)	5.068 MHz		SPBRG value (decimal)	4 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
9600	9.520	-0.83%	46	9598.485	0.016%	32	9615.385	0.160%	25
19200	19.454	+1.32%	22	18632.35	-2.956%	16	19230.77	0.160%	12
38400	37.286	-2.90%	11	39593.75	3.109%	7	35714.29	-6.994%	6
57600	55.930	-2.90%	7	52791.67	-8.348%	5	62500	8.507%	3
115200	111.860	-2.90%	3	105583.3	-8.348%	2	125000	8.507%	1
250000	NA	—	—	316750	26.700%	0	250000	0.000%	0
625000	NA	—	—	NA	—	—	NA	—	—
1250000	NA	—	—	NA	—	—	NA	—	—

BAUD RATE (K)	Fosc = 3.579 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 MHz		SPBRG value (decimal)
	KBAUD	ERROR		KBAUD	ERROR		KBAUD	ERROR	
9600	9725.543	1.308%	22	8.928	-6.994%	6	NA	NA	NA
19200	18640.63	-2.913%	11	20833.3	8.507%	2	NA	NA	NA
38400	37281.25	-2.913%	5	31250	-18.620%	1	NA	NA	NA
57600	55921.88	-2.913%	3	62500	+8.507	0	NA	NA	NA
115200	111243.8	-2.913%	1	NA	—	—	NA	NA	NA
250000	223687.5	-10.525%	0	NA	—	—	NA	NA	NA
625000	NA	—	—	NA	—	—	NA	NA	NA
1250000	NA	—	—	NA	—	—	NA	NA	NA

The data on the RB1/RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin. If bit BRGH (TXSTA<2>) is clear (i.e., at the low baud rates), the sampling is done on the seventh, eighth and ninth falling edges of a x16 clock (Figure 12-3). If bit BRGH is set (i.e., at the high baud rates), the sampling is done on the 3 clock edges preceding the second rising edge after the first falling edge of a x4 clock (Figure 12-4 and Figure 12-5).

FIGURE 12-1: RX PIN SAMPLING SCHEME. BRGH = 0

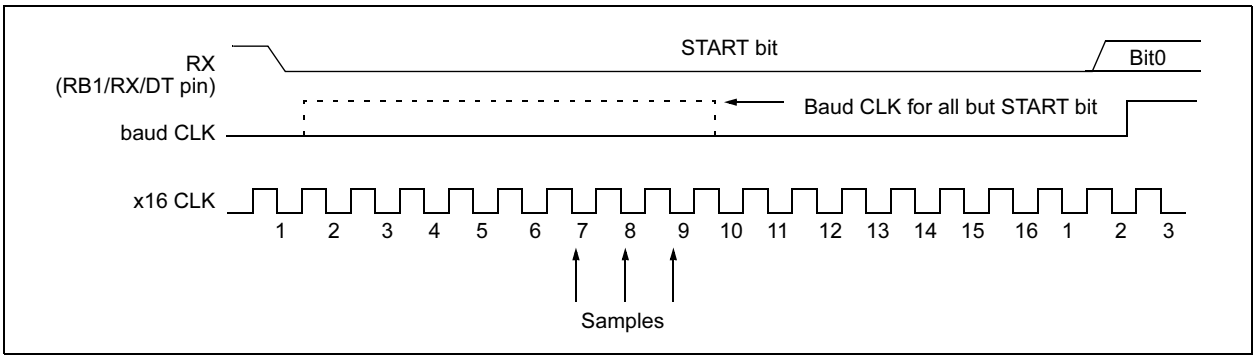


FIGURE 12-2: RX PIN SAMPLING SCHEME, BRGH = 1

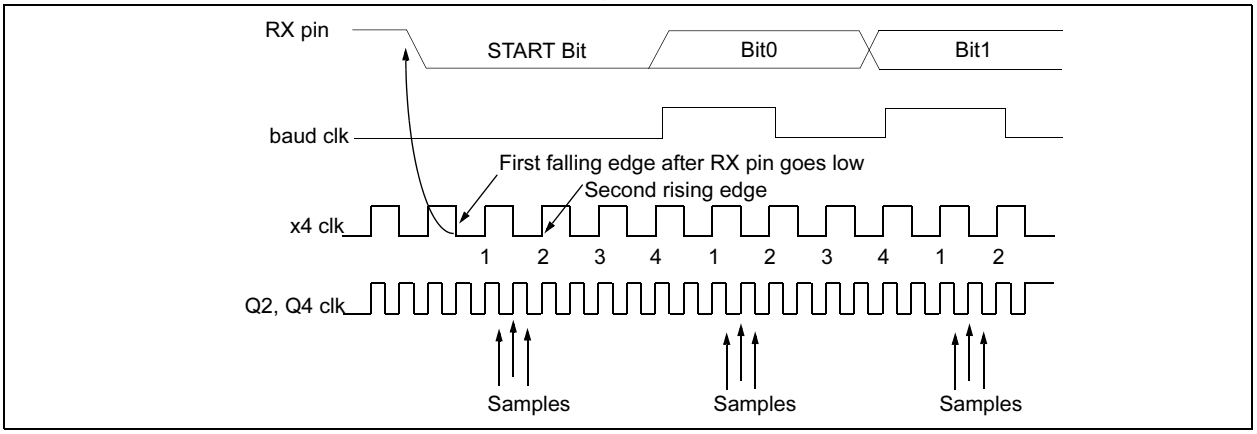


FIGURE 12-3: RX PIN SAMPLING SCHEME, BRGH = 1

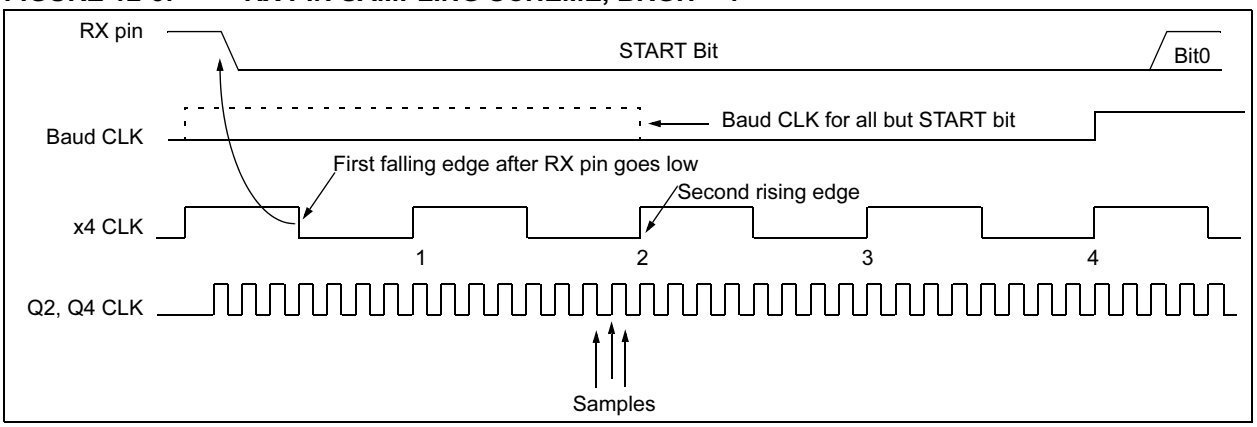
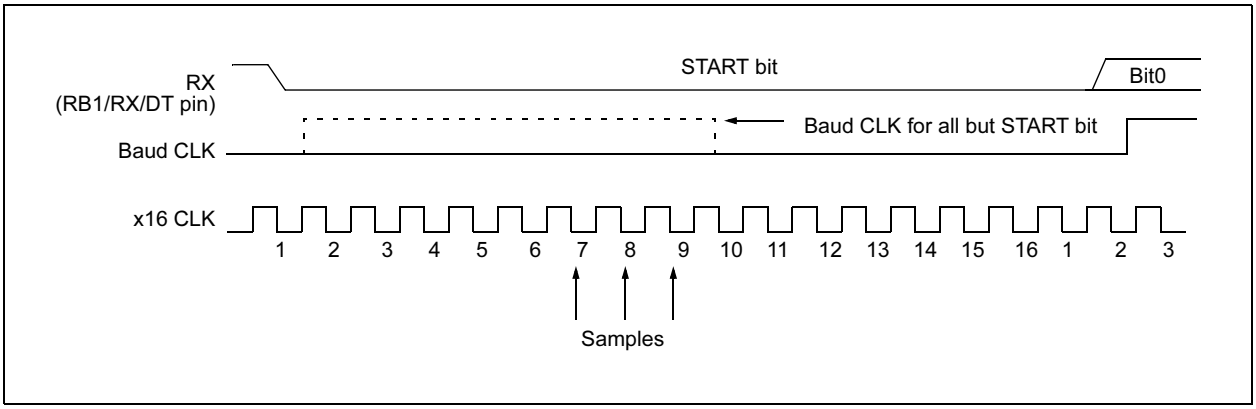


FIGURE 12-4: RX PIN SAMPLING SCHEME, BRGH = 0 OR BRGH = 1



12.2 USART Asynchronous Mode

In this mode, the USART uses standard non-return to zero (NRZ) format (one START bit, eight or nine data bits and one STOP bit). The most common data format is 8 bits. A dedicated 8-bit baud rate generator is used to derive baud rate frequencies from the oscillator. The USART transmits and receives the LSb first. The USART's transmitter and receiver are functionally independent but use the same data format and baud rate. The baud rate generator produces a clock either x16 or x64 of the bit shift rate, depending on bit BRGH (TXSTA<2>). Parity is not supported by the hardware, but can be implemented in software (and stored as the ninth data bit). Asynchronous mode is stopped during SLEEP.

Asynchronous mode is selected by clearing bit SYNC (TXSTA<4>).

The USART Asynchronous module consists of the following important elements:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

12.2.1 USART ASYNCHRONOUS TRANSMITTER

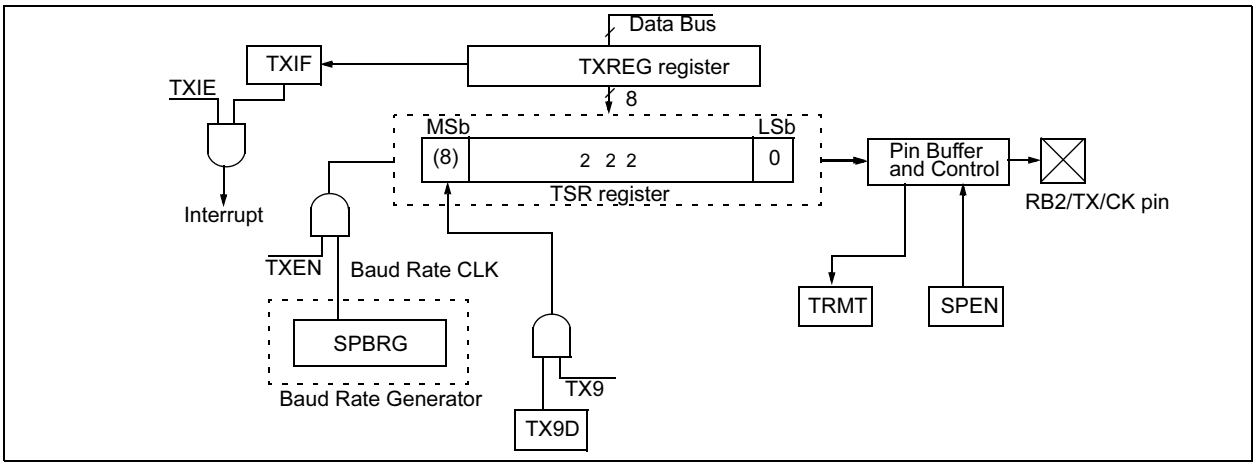
The USART transmitter block diagram is shown in Figure 12-5. The heart of the transmitter is the transmit (serial) shift register (TSR). The shift register obtains its data from the read/write transmit buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one TCY), the TXREG register is empty and flag bit TXIF (PIR1<4>) is set. This interrupt can be enabled/disabled by setting/clearing enable bit TXIE (PIE1<4>). Flag bit TXIF will be set regardless of the state of enable bit TXIE and cannot be cleared in software. It will RESET only when new data is loaded into the TXREG register. While flag bit TXIF indicated the status of the TXREG register, another bit TRMT (TXSTA<1>) shows the status of the TSR register. STATUS bit TRMT is a read only bit which is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty.

> **Note 1:** The TSR register is not mapped in data memory so it is not available to the user.
>
> **2:** Flag bit TXIF is set when enable bit TXEN is set.

Transmission is enabled by setting enable bit TXEN (TXSTA<5>). The actual transmission will not occur until the TXREG register has been loaded with data and the baud rate generator (BRG) has produced a shift clock (Figure 12-5). The transmission can also be started by first loading the TXREG register and then setting enable bit TXEN. Normally when transmission is first started, the TSR register is empty, so a transfer to the TXREG register will result in an immediate transfer to TSR resulting in an empty TXREG. A back-to-back transfer is thus possible (Figure 12-7). Clearing enable bit TXEN during a transmission will cause the transmission to be aborted and will RESET the transmitter. As a result the RB2/TX/CK pin will revert to hi-impedance.

In order to select 9-bit transmission, transmit bit TX9 (TXSTA<6>) should be set and the ninth bit should be written to TX9D (TXSTA<0>). The ninth bit must be written before writing the 8-bit data to the TXREG register. This is because a data write to the TXREG register can result in an immediate transfer of the data to the TSR register (if the TSR is empty). In such a case, an incorrect ninth data bit may be loaded in the TSR register.

FIGURE 12-5: USART TRANSMIT BLOCK DIAGRAM



Steps to follow when setting up an Asynchronous Transmission:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH. (Section 12.1)
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, then set enable bit TXIE.
4. If 9-bit transmission is desired, then set transmit bit TX9.
5. Enable the transmission by setting bit TXEN, which will also set bit TXIF.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Load data to the TXREG register (starts transmission).

FIGURE 12-6: ASYNCHRONOUS TRANSMISSION

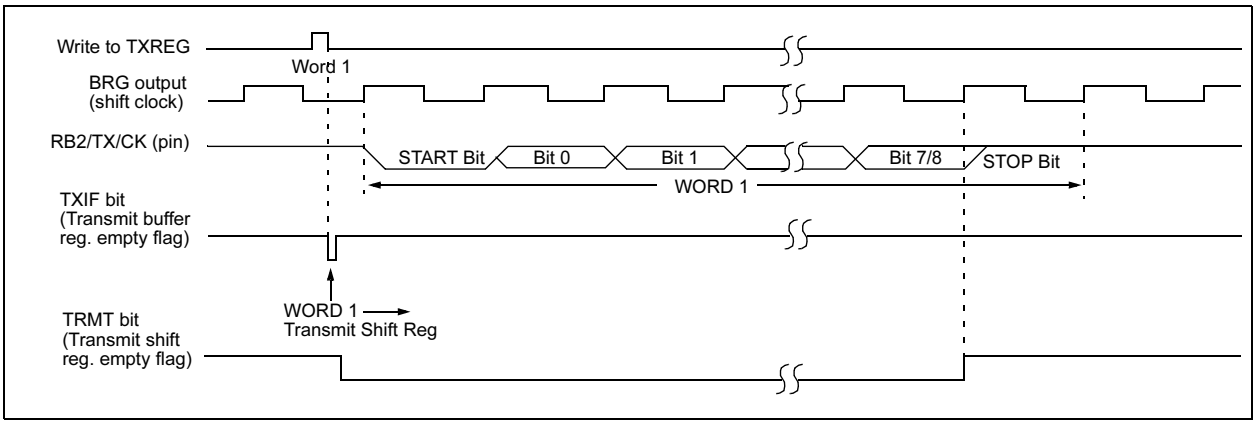
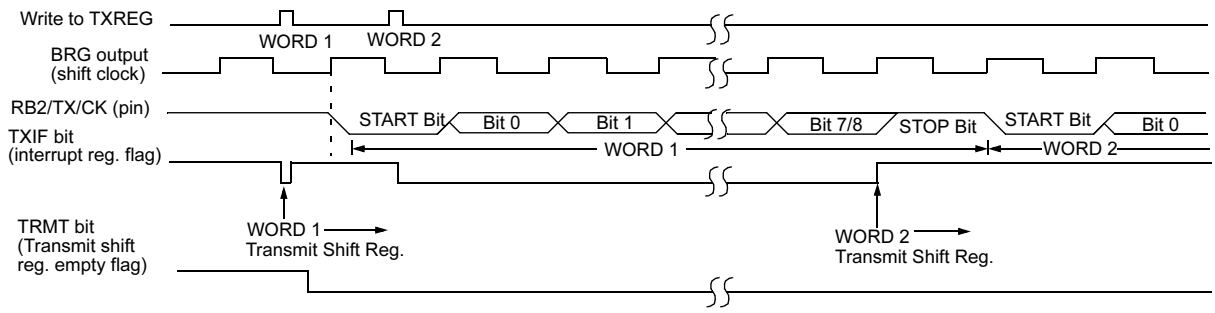


FIGURE 12-7: ASYNCHRONOUS TRANSMISSION (BACK TO BACK)



Note 1: This.timing diagram shows two consecutive transmissions.

TABLE 12-6: REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'.
Shaded cells are not used for Asynchronous Transmission.

12.2.2 ADEN USART ASYNCHRONOUS RECEIVER

The receiver block diagram is shown in Figure 12-8. The data is received on the RB1/RX/DT pin and drives the data recovery block. The data recovery block is actually a high speed shifter operating at x16 times the baud rate, whereas the main receive serial shifter operates at the bit rate or at FOSC.

Once Asynchronous mode is selected, reception is enabled by setting bit CREN (RCSTA<4>).

The heart of the receiver is the Receive (serial) Shift register (RSR). After sampling the STOP bit, the received data in the RSR is transferred to the RCREG register (if it is empty). If the transfer is complete, flag bit RCIF (PIR1<5>) is set. The actual interrupt can be enabled/disabled by setting/clearing enable bit RCIE (PIE1<5>). Flag bit RCIF is a read only bit which is cleared by the hardware. It is cleared when the RCREG register has been read and is empty. The RCREG is a double buffered register (i.e., it is a two-deep FIFO).

It is possible for two bytes of data to be received and transferred to the RCREG FIFO, and a third byte begin shifting to the RSR register. On the detection of the STOP bit of the third byte, if the RCREG register is still full, then overrun error bit OERR (RCSTA<1>) will be set. The word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. Overrun bit OERR has to be cleared in software. This is done by resetting the receive logic (CREN is cleared and then set). If bit OERR is set, transfers from the RSR register to the RCREG register are inhibited, so it is essential to clear error bit OERR if it is set. Framing error bit FERR (RCSTA<2>) is set if a STOP bit is detected as clear. Bit FERR and the 9th receive bit are buffered the same way as the receive data. Reading the RCREG will load bits RX9D and FERR with new values, therefore it is essential for the user to read the RCSTA register <u>before</u> reading the RCREG register in order not to lose the old FERR and RX9D information.

FIGURE 12-8: USART RECEIVE BLOCK DIAGRAM

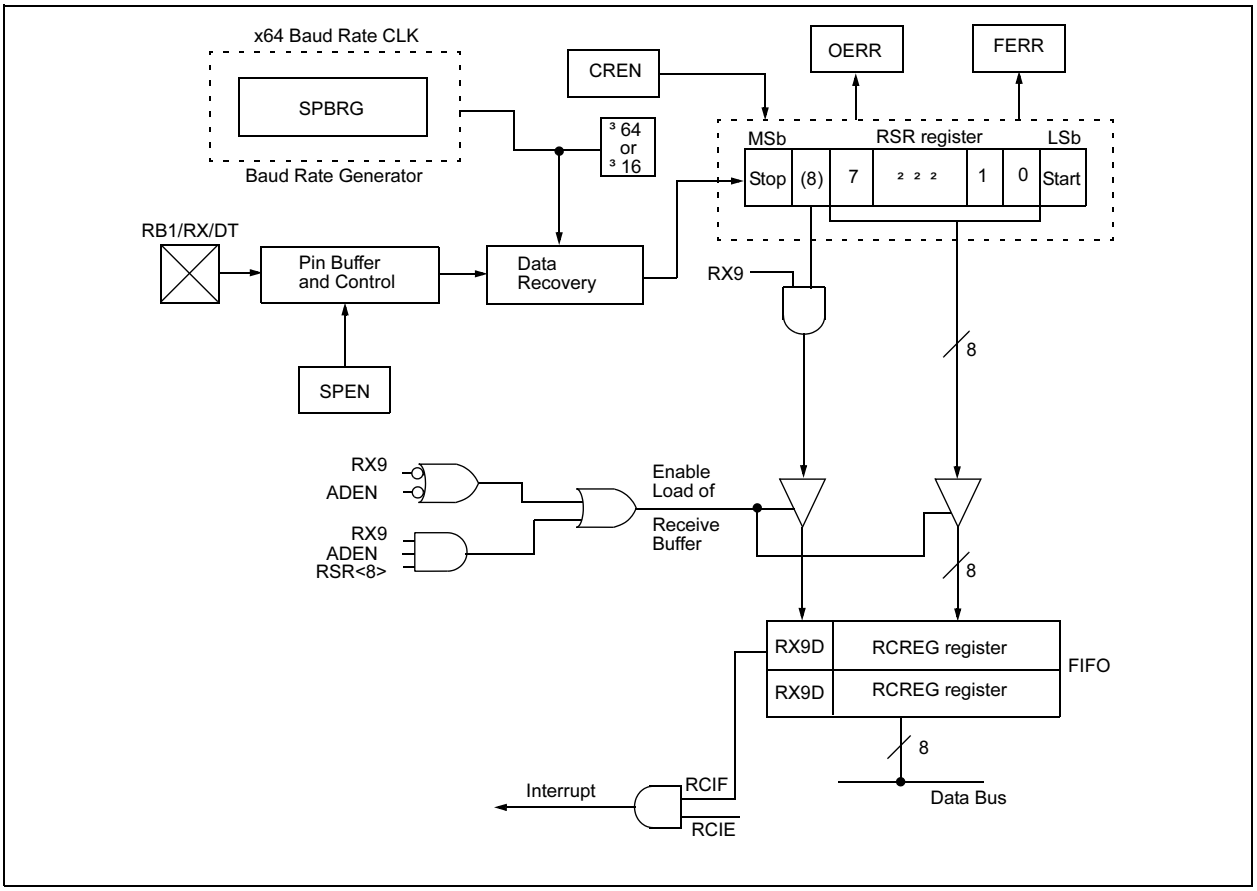
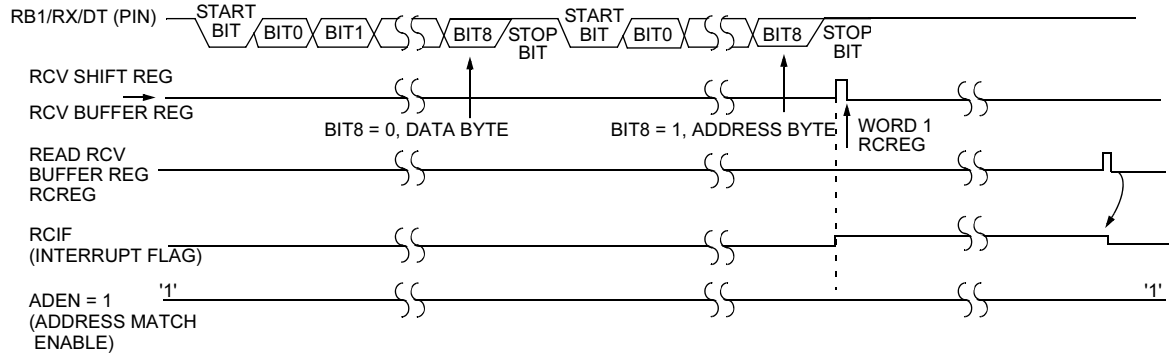
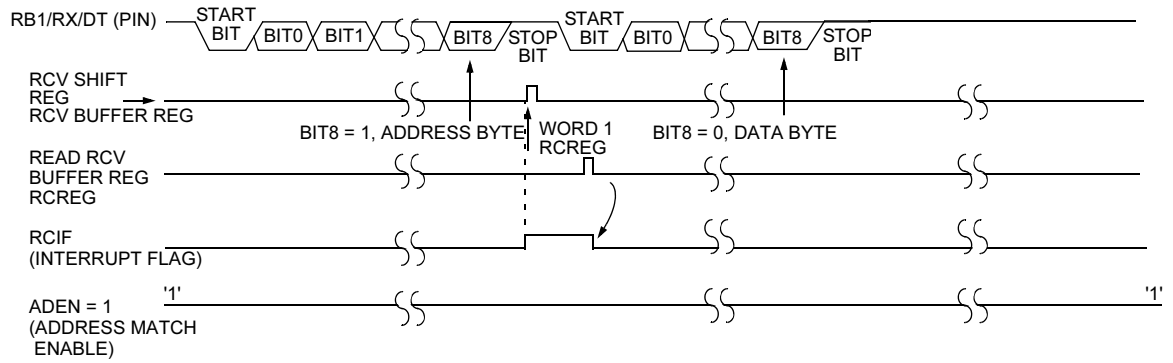


FIGURE 12-9: ASYNCHRONOUS RECEPTION WITH ADDRESS DETECT



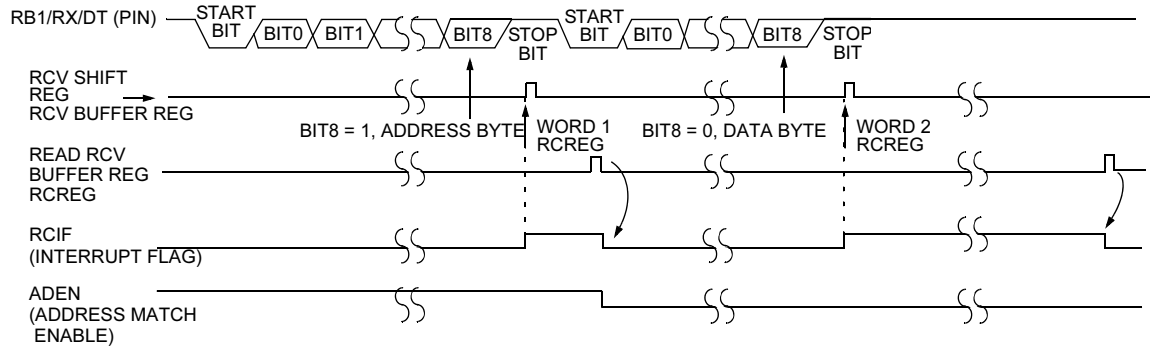
Note 1: This timing diagram shows a data byte followed by an address byte. The data byte is not read into the RCREG (Receive Buffer) because ADEN = 1 and Bit 8 = 0.

FIGURE 12-10: ASYNCHRONOUS RECEPTION WITH ADDRESS BYTE FIRST



Note 1: This timing diagram shows an address byte followed by an data byte. The data byte is not read into the RCREG (receive buffer) because ADEN was not updated (still = 1) and Bit 8 = 0.

FIGURE 12-11: ASYNCHRONOUS RECEPTION WITH ADDRESS BYTE FIRST FOLLOWED BY VALID DATA BYTE



Note 1: This timing diagram shows an address byte followed by an data byte. The data byte is read into the RCREG (Receive Buffer) because ADEN was updated after an address match, and was cleared to a ‘0’, so the contents of the Receive Shift Register (RSR) are read into the Receive Buffer regardless of the value of Bit 8.

Steps to follow when setting up an Asynchronous Reception:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH. (Section 12.1).
2. Enable the asynchronous serial port by clearing bit SYNC, and setting bit SPEN.
3. If interrupts are desired, then set enable bit RCIE.
4. If 9-bit reception is desired, then set bit RX9.
5. Enable the reception by setting bit CREN.
6. Flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE was set.
7. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
8. Read the 8-bit received data by reading the RCREG register.
9. If any error occurred, clear the error by clearing enable bit CREN.

TABLE 12-7: REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for Asynchronous Reception.

12.3 USART Function

The USART function is similar to that on the PIC16C74B, which includes the BRGH = 1 fix.

12.3.1 USART 9-BIT RECEIVER WITH ADDRESS DETECT

When the RX9 bit is set in the RCSTA register, 9 bits are received and the ninth bit is placed in the RX9D bit of the RCSTA register. The USART module has a special provision for multiprocessor communication. Multiprocessor communication is enabled by setting the ADEN bit (RCSTA<3>) along with the RX9 bit. The port is now programmed so when the last bit is received, the contents of the Receive Shift Register (RSR) are transferred to the receive buffer. The ninth bit of the RSR (RSR<8>) is transferred to RX9D, and the receive interrupt is set if, and only, if RSR<8> = 1. This feature can be used in a multiprocessor system as follows:

A master processor intends to transmit a block of data to one of many slaves. It must first send out an address byte that identifies the target slave. An address byte is identified by setting the ninth bit (RSR<8>) to a '1' (instead of a '0' for a data byte). If the ADEN and RX9 bits are set in the slave's RCSTA register, enabling multiprocessor communication, all data bytes will be ignored. However, if the ninth received bit is equal to a '1', indicating that the received byte is an address, the slave will be interrupted and the contents of the RSR register will be transferred into the receive buffer. This allows the slave to be interrupted only by addresses, so that the slave can examine the received byte to see if it is being addressed. The addressed slave will then clear its ADEN bit and prepare to receive data bytes from the master.

When ADEN is enabled (='1'), all data bytes are ignored. Following the STOP bit, the data will not be loaded into the receive buffer, and no interrupt will occur. If another byte is shifted into the RSR register, the previous data byte will be lost.

The ADEN bit will only take effect when the receiver is configured in 9-bit mode (RX9 = '1'). When ADEN is disabled (='0'), all data bytes are received and the 9th bit can be used as the PARITY bit.

The USART Receive Block Diagram is shown in Figure 12-8.

Reception is enabled by setting bit CREN (RCSTA<4>).

12.3.1.1 Setting up 9-bit mode with Address Detect

Steps to follow when setting up an Asynchronous or Synchronous Reception with Address Detect Enabled:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH.
2. Enable asynchronous or synchronous communication by setting or clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, then set enable bit RCIE.
4. Set bit RX9 to enable 9-bit reception.
5. Set ADEN to enable address detect.
6. Enable the reception by setting enable bit CREN or SREN.
7. Flag bit RCIF will be set when reception is complete, and an interrupt will be generated if enable bit RCIE was set.
8. Read the 8-bit received data by reading the RCREG register to determine if the device is being addressed.
9. If any error occurred, clear the error by clearing enable bit CREN if it was already set.
10. If the device has been addressed (RSR<8> = 1 with address match enabled), clear the ADEN and RCIF bits to allow data bytes and address bytes to be read into the receive buffer and interrupt the CPU.

TABLE 12-8: REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	RX7	RX6	RX5	RX4	RX3	RX2	RX1	RX0	0000 0000	0000 0000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for Asynchronous Reception.

12.4 USART Synchronous Master Mode

In Synchronous Master mode, the data is transmitted in a half-duplex manner (i.e., transmission and reception do not occur at the same time). When transmitting data, the reception is inhibited and vice versa. Synchronous mode is entered by setting bit SYNC (TXSTA<4>). In addition, enable bit SPEN (RCSTA<7>) is set in order to configure the RB2/TX/CK and RB1/RX/DT I/O pins to CK (clock) and DT (data) lines respectively. The Master mode indicates that the processor transmits the master clock on the CK line. The Master mode is entered by setting bit CSRC (TXSTA<7>).

12.4.1 USART SYNCHRONOUS MASTER TRANSMISSION

The USART Transmitter Block Diagram is shown in Figure 12-5. The heart of the transmitter is the Transmit (serial) Shift register (TSR). The Shift register obtains its data from the read/write transmit buffer register TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the last bit has been transmitted from the previous load. As soon as the last bit is transmitted, the TSR is loaded with new data from the TXREG (if available). Once the TXREG register transfers the data to the TSR register (occurs in one Tcycle), the TXREG is empty and interrupt bit, TXIF (PIR1<4>) is set. The interrupt can be enabled/disabled by setting/clearing enable bit TXIE (PIE1<4>). Flag bit TXIF will be set regardless of the state of enable bit TXIE and cannot be cleared in software. It will RESET only when new data is loaded into the TXREG register. While flag bit TXIF indicates the status of the TXREG register, another bit TRMT (TXSTA<1>) shows the status of the TSR register. TRMT is a read only bit which is set when the TSR is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty. The TSR is not mapped in data memory so it is not available to the user.

Transmission is enabled by setting enable bit TXEN (TXSTA<5>). The actual transmission will not occur until the TXREG register has been loaded with data. The first data bit will be shifted out on the next available rising edge of the clock on the CK line. Data out is stable around the falling edge of the synchronous clock (Figure 12-12). The transmission can also be started by first loading the TXREG register and then setting bit TXEN (Figure 12-13). This is advantageous when slow baud rates are selected, since the BRG is kept in RESET when bits TXEN, CREN, and SREN are clear. Setting enable bit TXEN will start the BRG, creating a shift clock immediately. Normally when transmission is first started, the TSR register is empty, so a transfer to the TXREG register will result in an immediate transfer to TSR resulting in an empty TXREG. Back-to-back transfers are possible.

Clearing enable bit TXEN, during a transmission, will cause the transmission to be aborted and will RESET the transmitter. The DT and CK pins will revert to hi-impedance. If either bit CREN or bit SREN is set, during a transmission, the transmission is aborted and the DT pin reverts to a hi-impedance state (for a reception). The CK pin will remain an output if bit CSRC is set (internal clock). The transmitter logic however is not RESET although it is disconnected from the pins. In order to RESET the transmitter, the user has to clear bit TXEN. If bit SREN is set (to interrupt an on-going transmission and receive a single word), then after the single word is received, bit SREN will be cleared and the serial port will revert back to transmitting since bit TXEN is still set. The DT line will immediately switch from Hi-impedance Receive mode to transmit and start driving. To avoid this, bit TXEN should be cleared.

In order to select 9-bit transmission, the TX9 (TXSTA<6>) bit should be set and the ninth bit should be written to bit TX9D (TXSTA<0>). The ninth bit must be written before writing the 8-bit data to the TXREG register. This is because a data write to the TXREG can result in an immediate transfer of the data to the TSR register (if the TSR is empty). If the TSR was empty and the TXREG was written before writing the “new” TX9D, the “present” value of bit TX9D is loaded.

Steps to follow when setting up a Synchronous Master Transmission:

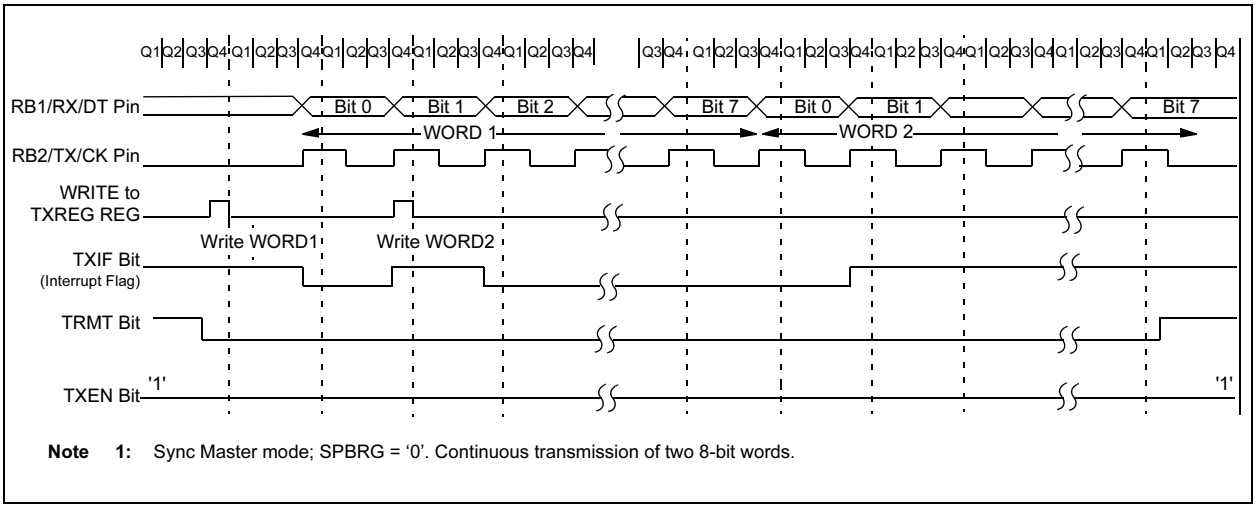
1. Initialize the SPBRG register for the appropriate baud rate (Section 12.1).
2. Enable the synchronous master serial port by setting bits SYNC, SPEN, and CSRC.
3. If interrupts are desired, then set enable bit TXIE.
4. If 9-bit transmission is desired, then set bit TX9.
5. Enable the transmission by setting bit TXEN.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Start transmission by loading data to the TXREG register.

TABLE 12-9: REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER TRANSMISSION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

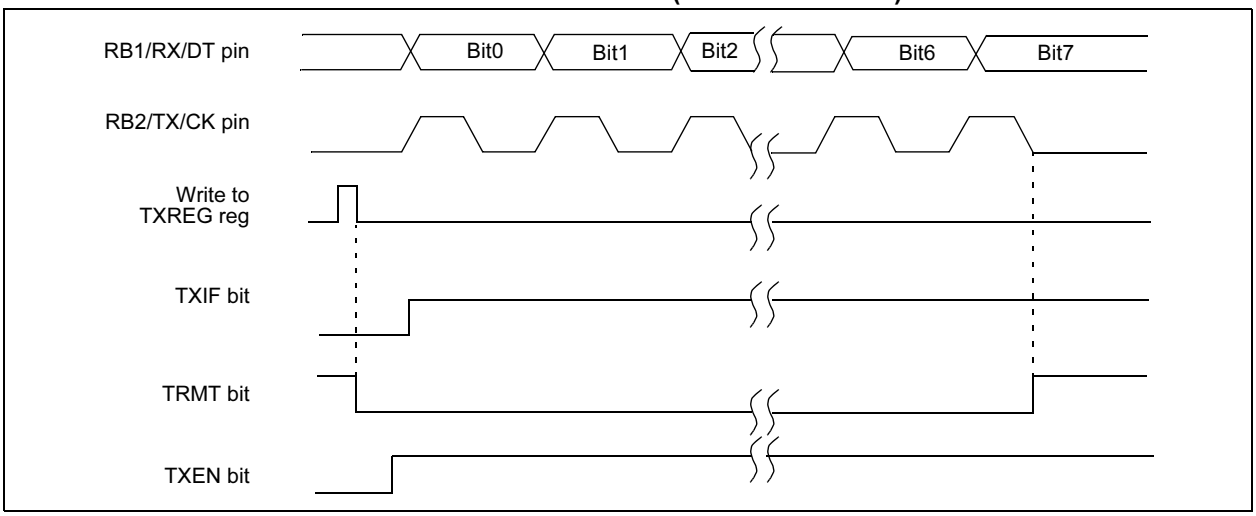
Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used for Synchronous Master Transmission.

FIGURE 12-12: SYNCHRONOUS TRANSMISSION



Note 1: Sync Master mode; SPBRG = '0'. Continuous transmission of two 8-bit words.

FIGURE 12-13: SYNCHRONOUS TRANSMISSION (THROUGH TXEN)



12.4.2 USART SYNCHRONOUS MASTER RECEPTION

Once Synchronous mode is selected, reception is enabled by setting either enable bit SREN (RCSTA<5>) or enable bit CREN (RCSTA<4>). Data is sampled on the RB1/RX/DT pin on the falling edge of the clock. If enable bit SREN is set, then only a single word is received. If enable bit CREN is set, the reception is continuous until CREN is cleared. If both bits are set, then CREN takes precedence. After clocking the last bit, the received data in the Receive Shift Register (RSR) is transferred to the RCREG register (if it is empty). When the transfer is complete, interrupt flag bit RCIF (PIR1<5>) is set. The actual interrupt can be enabled/disabled by setting/clearing enable bit RCIE (PIE1<5>). Flag bit RCIF is a read only bit which is RESET by the hardware. In this case, it is RESET when the RCREG register has been read and is empty. The RCREG is a double buffered register (i.e., it is a two-deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte to begin shifting into the RSR register. On the clocking of the last bit of the third byte, if the RCREG register is still full then overrun error bit OERR (RCSTA<1>) is set. The word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. Bit OERR has to be cleared in software (by clearing bit CREN). If bit OERR is set, transfers from the RSR to the RCREG are inhibited, so it is essential to clear bit OERR if it is set. The 9th receive bit is buffered the same way as the receive data. Reading the RCREG register, will load bit RX9D with a new value, therefore it is essential for the user to read the RCSTA register before reading RCREG in order not to lose the old RX9D information.

Steps to follow when setting up a Synchronous Master Reception:

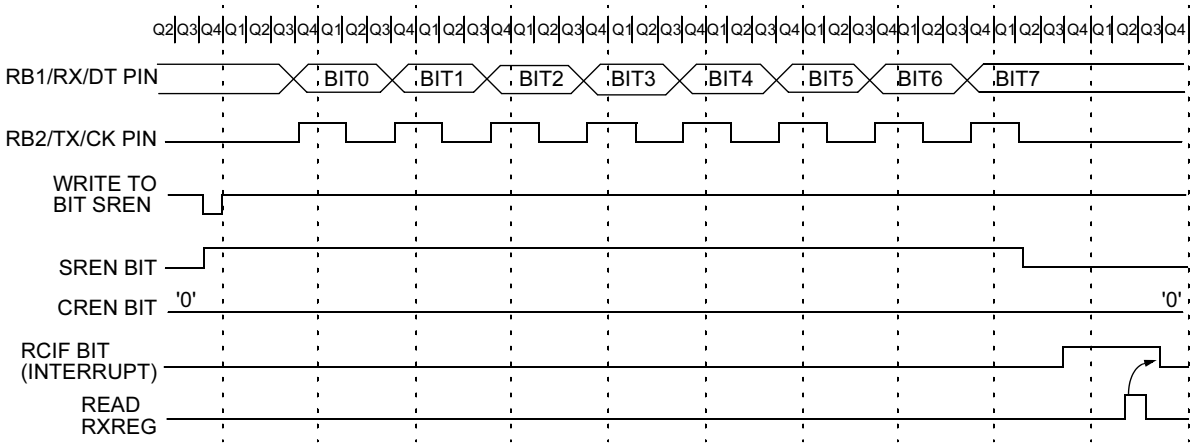
1. Initialize the SPBRG register for the appropriate baud rate. (Section 12.1)
2. Enable the synchronous master serial port by setting bits SYNC, SPEN, and CSRC.
3. Ensure bits CREN and SREN are clear.
4. If interrupts are desired, then set enable bit RCIE.
5. If 9-bit reception is desired, then set bit RX9.
6. If a single reception is required, set bit SREN. For continuous reception set bit CREN.
7. Interrupt flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE was set.
8. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
9. Read the 8-bit received data by reading the RCREG register.
10. If any error occurred, clear the error by clearing bit CREN.

TABLE 12-10: REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR	Value on all other RESETS
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	EEPIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	-000 0000	-000 -000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'. Shaded cells are not used for Synchronous Master Reception.

FIGURE 12-14: SYNCHRONOUS RECEPTION (MASTER MODE, SREN)



Note 1: Timing diagram demonstrates Sync Master mode with bit SREN = ‘1’ and bit BRG = ‘0’.

12.5 USART Synchronous Slave Mode

Synchronous Slave mode differs from the Master mode in the fact that the shift clock is supplied externally at the RB2/TX/CK pin (instead of being supplied internally in Master mode). This allows the device to transfer or receive data while in SLEEP mode. Slave mode is entered by clearing bit CSRC (TXSTA<7>).

12.5.1 USART SYNCHRONOUS SLAVE TRANSMIT

The operation of the Synchronous Master and Slave modes are identical except in the case of the SLEEP mode.

If two words are written to the TXREG and then the `SLEEP` instruction is executed, the following will occur:

a) The first word will immediately transfer to the TSR register and transmit.
b) The second word will remain in TXREG register.
c) Flag bit TXIF will not be set.
d) When the first word has been shifted out of TSR, the TXREG register will transfer the second word to the TSR and flag bit TXIF will now be set.
e) If enable bit TXIE is set, the interrupt will wake the chip from SLEEP and if the global interrupt is enabled, the program will branch to the interrupt vector (0004h).

Steps to follow when setting up a Synchronous Slave Transmission:

1. Enable the synchronous slave serial port by setting bits SYNC and SPEN and clearing bit CSRC.
2. Clear bits CREN and SREN.
3. If interrupts are desired, then set enable bit TXIE.
4. If 9-bit transmission is desired, then set bit TX9.
5. Enable the transmission by setting enable bit TXEN.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Start transmission by loading data to the TXREG register.

12.5.2 USART SYNCHRONOUS SLAVE RECEPTION

The operation of the Synchronous Master and Slave modes is identical except in the case of the SLEEP mode. Also, bit SREN is a don't care in Slave mode.

If receive is enabled, by setting bit CREN, prior to the `SLEEP` instruction, then a word may be received during SLEEP. On completely receiving the word, the RSR register will transfer the data to the RCREG register and if enable bit RCIE bit is set, the interrupt generated will wake the chip from SLEEP. If the global interrupt is enabled, the program will branch to the interrupt vector (0004h).

Steps to follow when setting up a Synchronous Slave Reception:

1. Enable the synchronous master serial port by setting bits SYNC and SPEN and clearing bit CSRC.
2. If interrupts are desired, then set enable bit RCIE.
3. If 9-bit reception is desired, then set bit RX9.
4. To enable reception, set enable bit CREN.
5. Flag bit RCIF will be set when reception is complete and an interrupt will be generated, if enable bit RCIE was set.
6. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
7. Read the 8-bit received data by reading the RCREG register.
8. If any error occurred, clear the error by clearing bit CREN.

TABLE 12-11: REGISTERS ASSOCIATED WITH SYNCHRONOUS SLAVE TRANSMISSION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'. Shaded cells are not used for Synchronous Slave Transmission.

TABLE 12-12: REGISTERS ASSOCIATED WITH SYNCHRONOUS SLAVE RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other RESETS
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'. Shaded cells are not used for Synchronous Slave Reception.

NOTES:

13.0 DATA EEPROM MEMORY

The EEPROM data memory is readable and writable during normal operation (full VDD range). This memory is not directly mapped in the register file space. Instead it is indirectly addressed through the Special Function Registers (SFRs). There are four SFRs used to read and write this memory. These registers are:

- EECON1
- EECON2 (Not a physically implemented register)
- EEDATA
- EEADR

EEDATA holds the 8-bit data for read/write, and EEADR holds the address of the EEPROM location being accessed. PIC16F62X devices have 128 bytes of data EEPROM with an address range from 0h to 7Fh.

The EEPROM data memory allows byte read and write. A byte write automatically erases the location and writes the new data (erase before write). The EEPROM data memory is rated for high erase/write cycles. The write time is controlled by an on-chip timer. The write-time will vary with voltage and temperature as well as from chip to chip. Please refer to AC specifications for exact limits.

When the device is code protected, the CPU may continue to read and write the data EEPROM memory. The device programmer can no longer access this memory.

Additional information on the Data EEPROM is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

REGISTER 13-1: EEADR REGISTER (ADDRESS: 9Bh)

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reserved	EADR6	EADR5	EADR4	EADR3	EADR2	EADR1	EADR0
bit 7							bit 0

bit 7 **Unimplemented Address**: Must be set to ‘0’

bit 6-0 **EEADR**: Specifies one of 128 locations of EEPROM Read/Write Operation

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

13.1 EEADR

The EEADR register can address up to a maximum of 256 bytes of data EEPROM. Only the first 128 bytes of data EEPROM are implemented and only seven of the eight bits in the register (EEADR<6:0>) are required.

The upper bit is address decoded. This means that this bit should always be '0' to ensure that the address is in the 128 byte memory space.

13.2 EECON1 AND EECON2 REGISTERS

EECON1 is the control register with five low order bits physically implemented. The upper-three bits are non-existent and read as '0's.

Control bits RD and WR initiate read and write, respectively. These bits cannot be cleared, only set, in software. They are cleared in hardware at completion of the read or write operation. The inability to clear the WR bit in software prevents the accidental, premature termination of a write operation.

The WREN bit, when set, will allow a write operation. On power-up, the WREN bit is clear. The WRERR bit is set when a write operation is interrupted by a $\overline{\text{MCLR}}$ Reset or a WDT Timeout Reset during normal operation. In these situations, following RESET, the user can check the WRERR bit and rewrite the location. The data and address will be unchanged in the EEDATA and EEADR registers.

Interrupt flag bit EEIF in the PIR1 register is set when write is complete. This bit must be cleared in software.

EECON2 is not a physical register. Reading EECON2 will read all '0's. The EECON2 register is used exclusively in the Data EEPROM write sequence.

REGISTER 13-2: EECON1 REGISTER (ADDRESS: 9Ch)

U-0	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-x
—	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as '0'

bit 3 **WRERR**: EEPROM Error Flag bit

1 = A write operation is prematurely terminated (any $\overline{\text{MCLR}}$ Reset, any WDT Reset during normal operation or BOD Reset)

0 = The write operation completed

bit 2 **WREN**: EEPROM Write Enable bit

1 = Allows write cycles

0 = Inhibits write to the data EEPROM

bit 1 **WR**: Write Control bit

1 = Initiates a write cycle. (The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.

0 = Write cycle to the data EEPROM is complete

bit 0 **RD**: Read Control bit

1 = Initiates an EEPROM read (read takes one cycle. RD is cleared in hardware. The RD bit can only be set (not cleared) in software).

0 = Does not initiate an EEPROM read

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

13.3 READING THE EEPROM DATA MEMORY

To read a data memory location, the user must write the address to the EEADR register and then set control bit RD (EECON1<0>). The data is available, in the very next cycle, in the EEDATA register; therefore it can be read in the next instruction. EEDATA will hold this value until another read or until it is written to by the user (during a write operation).

EXAMPLE 13-1: DATA EEPROM READ

```
BSF     STATUS, RP0  ; Bank 1
MOVLW   CONFIG_ADDR  ;
MOVWF   EEADR        ; Address to read
BSF     EECON1, RD   ; EE Read
MOVF    EEDATA, W    ; W = EEDATA
BCF     STATUS, RP0  ; Bank 0
```

13.4 WRITING TO THE EEPROM DATA MEMORY

To write an EEPROM data location, the user must first write the address to the EEADR register and the data to the EEDATA register. Then the user must follow a specific sequence to initiate the write for each byte.

EXAMPLE 13-2: DATA EEPROM WRITE

```
        BSF     STATUS, RP0    ; Bank 1
        BSF     EECON1, WREN   ; Enable write
        BCF     INTCON, GIE    ; Disable INTs.
Required MOVLW  55h            ;
Sequence MOVWF  EECON2         ; Write 55h
        MOVLW   AAh            ;
        MOVWF   EECON2         ; Write AAh
        BSF     EECON1,WR      ; Set WR bit
                               ; begin write
        BSF     INTCON, GIE    ; Enable INTs.
```

The write will not initiate if the above sequence is not exactly followed (write 55h to EECON2, write AAh to EECON2, then set WR bit) for each byte. We strongly recommend that interrupts be disabled during this code segment. A cycle count is executed during the required sequence. Any number that is not equal to the required cycles to execute the required sequence will cause the data not to be written into the EEPROM.

Additionally, the WREN bit in EECON1 must be set to enable write. This mechanism prevents accidental writes to data EEPROM due to errant (unexpected) code execution (i.e., lost programs). The user should keep the WREN bit clear at all times, except when updating EEPROM. The WREN bit is not cleared by hardware.

After a write sequence has been initiated, clearing the WREN bit will not affect this write cycle. The WR bit will be inhibited from being set unless the WREN bit is set.

At the completion of the write cycle, the WR bit is cleared in hardware and the EE Write Complete Interrupt Flag bit (EEIF) is set. The user can either enable this interrupt or poll this bit. The EEIF bit in the PIR1 registers must be cleared by software.

13.5 WRITE VERIFY

Depending on the application, good programming practice may dictate that the value written to the Data EEPROM should be verified (Example 13-3) to the desired value to be written. This should be used in applications where an EEPROM bit will be stressed near the specification limit.

EXAMPLE 13-3: WRITE VERIFY

```
    BSF   STATUS, RP0  ; Bank 1
    MOVF  EEDATA, W
    BSF   EECON1, RD   ; Read the
                       ; value written
;
; Is the value written (in W reg) and
; read (in EEDATA) the same?
;
    SUBWF EEDATA, W    ;
    BCF   STATUS, RP0  ; Bank0
    BTFSS STATUS, Z    ; Is difference 0?
    GOTO  WRITE_ERR    ; NO, Write error
    :                  ; YES, Good write
    :                  ; Continue program
```

13.6 PROTECTION AGAINST SPURIOUS WRITE

There are conditions when the device may not want to write to the data EEPROM memory. To protect against spurious EEPROM writes, various mechanisms have been built in. On power-up, WREN is cleared. Also, the Power-up Timer (72 ms duration) prevents EEPROM write.

The write initiate sequence, and the WREN bit together help prevent an accidental write during brown-out, power glitch, or software malfunction.

13.7 DATA EEPROM OPERATION DURING CODE PROTECT

When the device is code protected, the CPU is able to read and write unscrambled data to the Data EEPROM.

TABLE 13-1: REGISTERS/BITS ASSOCIATED WITH DATA EEPROM

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
9Ah	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
9Bh	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
9Ch	EECON1	—	—	—	—	WRERR	WREN	WR	RD	---- x000	---- q000
9Dh	EECON2(1)	EEPROM control register 2								---- ----	---- ----

Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends upon condition.
Shaded cells are not used by data EEPROM.

Note 1: EECON2 is not a physical register

14.0 SPECIAL FEATURES OF THE CPU

Special circuits to deal with the needs of real-time applications are what sets a microcontroller apart from other processors. The PIC16F62X family has a host of such features intended to maximize system reliability, minimize cost through elimination of external components, provide power saving Operating modes and offer code protection.

These are:

1. OSC selection
2. RESET
3. Power-on Reset (POR)
4. Power-up Timer (PWRT)
5. Oscillator Start-Up Timer (OST)
6. Brown-out Reset (BOD)
7. Interrupts
8. Watchdog Timer (WDT)
9. SLEEP
10. Code protection
11. ID Locations
12. In-circuit Serial Programming

The PIC16F62X has a Watchdog Timer which is controlled by configuration bits. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in RESET until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay of 72 ms (nominal) on power-up only, designed to keep the part in RESET while the power supply stabilizes. There is also circuitry to RESET the device if a Brown-out occurs, which provides at least a 72 ms RESET. With these three functions on-chip, most applications need no external RESET circuitry.

The SLEEP mode is designed to offer a very low current Power-down mode. The user can wake-up from SLEEP through external RESET, Watchdog Timer wake-up or through an interrupt. Several oscillator options are also made available to allow the part to fit the application. The ER oscillator option saves system cost while the LP crystal option saves power. A set of configuration bits are used to select various options.

14.1 Configuration Bits

The configuration bits can be programmed (read as '0') or left unprogrammed (read as '1') to select various device configurations. These bits are mapped in program memory location 2007h.

The user will note that address 2007h is beyond the user program memory space. In fact, it belongs to the special configuration memory space (2000h – 3FFFh), which can be accessed only during programming. See Programming Specification.

REGISTER 14-1: CONFIGURATION WORD

CP1	CP0	CP1	CP0	—	CPD	LVP	BODEN	MCLRE	FOSC2	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
bit 13													bit 0

bit 13-10: **CP1:CP0:** Code Protection bits [2]
Code protection for 2K program memory
11 = Program memory code protection off
10 = 0400h-07FFh code protected
01 = 0200h-07FFh code protected
00 = 0000h-07FFhcode protected

Code protection for 1K program memory
11 = Program memory code protection off
10 = Program memory code protection off
01 = 0200h-03FFh code protected
00 = 0000h-03FFh code protected

bit 9: **Unimplemented**: Read as ‘0’

bit 8: **CPD:** Data Code Protection bit[3]
1 = Data memory code protection off
0 = Data memory code protected

bit 7: **LVP**: Low Voltage Programming Enable
1 = RB4/PGM pin has PGM function, low voltage programming enabled
0 = RB4/PGM is digital I/O, HV on $\overline{\text{MCLR}}$ must be used for programming

bit 6: **BODEN**: Brown-out Detect Reset Enable bit [1]
1 = BOD Reset enabled
0 = BOD Reset disabled

bit 5: **MCLRE**: RA5/$\overline{\text{MCLR}}$ pin function select
1 = RA5/$\overline{\text{MCLR}}$ pin function is $\overline{\text{MCLR}}$
0 = RA5/$\overline{\text{MCLR}}$ pin function is digital Input, $\overline{\text{MCLR}}$ internally tied to VDD

bit 3: **$\overline{\text{PWRTEN}}$**: Power-up Timer Enable bit [1]
1 = PWRT disabled
0 = PWRT enabled

bit 2: **WDTEN**: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 4, 1-0: **FOSC2:FOSC0**: Oscillator Selection bits[4]
111 = ER oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, Resistor on RA7/OSC1/CLKIN
110 = ER oscillator: I/O function on RA6/OSC2/CLKOUT pin, Resistor on RA7/OSC1/CLKIN
101 = INTRC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
100 = INTRC oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
011 = EC: I/O function on RA6/OSC2/CLKOUT pin, CLKIN on RA7/OSC1/CLKIN
010 = HS oscillator: High speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
001 = XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
000 = LP oscillator: Low power crystal on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN

Note 1: Enabling Brown-out Detect Reset automatically enables Power-up Timer (PWRT) regardless of the value of bit $\overline{\text{PWRTE}}$. Ensure the Power-up Timer is enabled anytime Brown-out Detect Reset is enabled.
2: All of the CP1:CP0 pairs have to be given the same value to enable the code protection scheme listed.
3: The entire data EEPROM will be erased when the code protection is turned off.
4: When $\overline{\text{MCLR}}$ is asserted in INTRC or ER mode, the internal clock oscillator is disabled.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	1 = bit is set	0 = bit is cleared	x = bit is unknown

14.2 Oscillator Configurations

14.2.1 OSCILLATOR TYPES

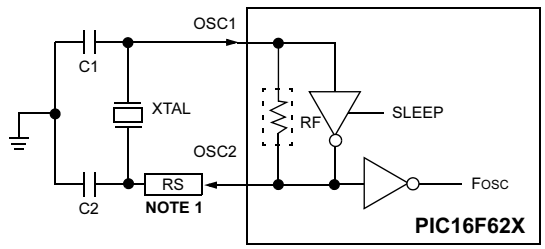
The PIC16F62X can be operated in eight different oscillator options. The user can program three configuration bits (FOSC2 thru FOSC0) to select one of these eight modes:

- LP Low Power Crystal
- XT Crystal/Resonator
- HS High Speed Crystal/Resonator
- ER External Resistor (2 modes)
- INTRC Internal Resistor/Capacitor (2 modes)
- EC External Clock In

14.2.2 CRYSTAL OSCILLATOR / CERAMIC RESONATORS

In XT, LP or HS modes a crystal or ceramic resonator is connected to the OSC1 and OSC2 pins to establish oscillation (Figure 14-1). The PIC16F62X oscillator design requires the use of a parallel cut crystal. Use of a series cut crystal may give a frequency out of the crystal manufacturers specifications. When in XT, LP or HS modes, the device can have an external clock source to drive the OSC1 pin (Figure 14-4).

FIGURE 14-1: CRYSTAL OPERATION (OR CERAMIC RESONATOR) (HS, XT OR LP OSC CONFIGURATION)



Note 1: A series resistor may be required for some crystals.
2: See Table 14-1 and Table 14-2 for recommended values of C1 and C2.

TABLE 14-1: CAPACITOR SELECTION FOR CERAMIC RESONATORS

Ranges Characterized:			
Mode	**Freq**	**OSC1(C1)**	**OSC2(C2)**
XT	455 kHz 2.0 MHz 4.0 MHz	22 - 100 pF 15 - 68 pF 15 - 68 pF	22 - 100 pF 15 - 68 pF 15 - 68 pF
HS	8.0 MHz 16.0 MHz	10 - 68 pF 10 - 22 pF	10 - 68 pF 10 - 22 pF

Note 1: Higher capacitance increases the stability of the oscillator but also increases the start-up time. These values are for design guidance only. Since each resonator has its own characteristics, the user should consult the resonator manufacturer for appropriate values of external components.

TABLE 14-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Mode	Freq	OSC1(C1)	OSC2(C2)
LP	32 kHz 200 kHz	68 - 100 pF 15 - 30 pF	68 - 100 pF 15 - 30 pF
XT	100 kHz 2 MHz 4 MHz	68 - 150 pF 15 - 30 pF 15 - 30 pF	150 - 200 pF 15 - 30 pF 15 - 30 pF
HS	8 MHz 10 MHz 20 MHz	15 - 30 pF 15 - 30 pF 15 - 30 pF	15 - 30 pF 15 - 30 pF 15 - 30 pF

Note 1: Higher capacitance increases the stability of the oscillator but also increases the start-up time. These values are for design guidance only. Rs may be required in HS mode as well as XT mode to avoid overdriving crystals with low drive level specification. Since each crystal has its own characteristics, the user should consult the crystal manufacturer for appropriate values of external components.

14.2.3 EXTERNAL CRYSTAL OSCILLATOR CIRCUIT

Either a prepackaged oscillator can be used, or a simple oscillator circuit with TTL gates can be built. Prepackaged oscillators provide a wide operating range and better stability. A well-designed crystal oscillator will provide good performance with TTL gates. Two types of crystal oscillator circuits can be used; one with series resonance, or one with parallel resonance.

Figure 14-2 shows implementation of a parallel resonant oscillator circuit. The circuit is designed to use the fundamental frequency of the crystal. The 74AS04 inverter performs the 180° phase shift that a parallel oscillator requires. The 4.7 kΩ resistor provides the negative feedback for stability. The 10 kΩ potentiometers bias the 74AS04 in the linear region. This could be used for external oscillator designs.

FIGURE 14-2: EXTERNAL PARALLEL RESONANT CRYSTAL OSCILLATOR CIRCUIT

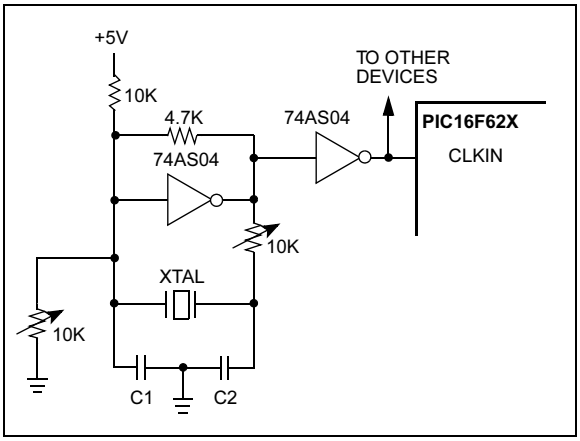
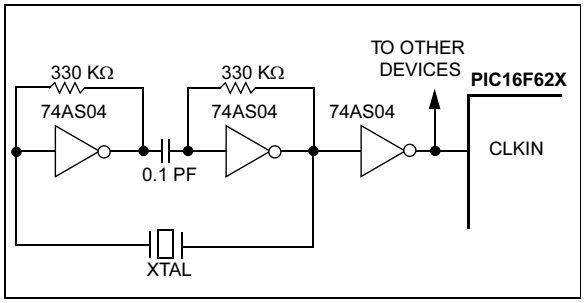


Figure 14-3 shows a series resonant oscillator circuit. This circuit is also designed to use the fundamental frequency of the crystal. The inverter performs a 180° phase shift in a series resonant oscillator circuit. The 330 kΩ resistors provide the negative feedback to bias the inverters in their linear region.

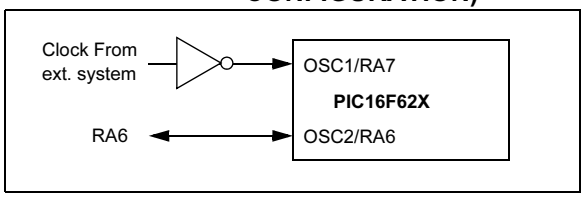
FIGURE 14-3: EXTERNAL SERIES RESONANT CRYSTAL OSCILLATOR CIRCUIT



14.2.4 EXTERNAL CLOCK IN

For applications, where a clock is already available elsewhere, users may directly drive the PIC16F62X provided that this external clock source meets the AC/DC timing requirements listed in Section 17.4. Figure 14-4 shows how an external clock circuit should be configured.

FIGURE 14-4: EXTERNAL CLOCK INPUT OPERATION (EC, HS, XT OR LP OSC CONFIGURATION)



14.2.5 ER OSCILLATOR

For timing insensitive applications, the ER (External Resistor) Clock mode offers additional cost savings. Only one external component, a resistor to VSS, is needed to set the operating frequency of the internal oscillator. The resistor draws a DC bias current which controls the oscillation frequency. In addition to the resistance value, the oscillator frequency will vary from unit to unit, and as a function of supply voltage and temperature. Since the controlling parameter is a DC current and not a capacitance, the particular package type and lead frame will not have a significant effect on the resultant frequency.

Figure 14-5 shows how the controlling resistor is connected to the PIC16F62X. For REXT values below 10k, the oscillator operation becomes sensitive to temperature. For very high REXT values (e.g., 1M), the oscillator becomes sensitive to leakage and may stop completely. Thus, we recommend keeping REXT between 10k and 1M.

FIGURE 14-5: EXTERNAL RESISTOR

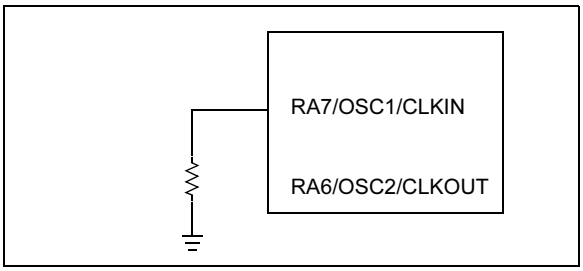


Table 14-3 shows the relationship between the resistance value and the operating frequency.

TABLE 14-3: RESISTANCE AND FREQUENCY RELATIONSHIP

Resistance	Frequency
0	10.4 MHz
1K	10 MHz
10K	7.4 MHz
20K	5.3 MHz
47K	3 MHz
100K	1.6 MHz
220K	800 kHz
470K	300 kHz
1M	200 kHz

The ER Oscillator mode has two options that control the unused OSC2 pin. The first allows it to be used as a general purpose I/O port. The other configures the pin as an output providing the FOSC signal (internal clock divided by 4) for test or external synchronization purposes.

14.2.6 INTERNAL 4 MHz OSCILLATOR

The internal RC oscillator provides a fixed 4 MHz (nominal) system clock at VDD = 5V and 25°C, see "Electrical Specifications" section for information on variation over voltage and temperature.

14.2.7 CLKOUT

The PIC16F62X can be configured to provide a clock out signal by programming the configuration word. The oscillator frequency, divided by 4 can be used for test purposes or to synchronize other logic.

14.3 Special Feature: Dual Speed Oscillator Modes

A software programmable Dual Speed Oscillator mode is provided when the PIC16F62X is configured in either ER or INTRC Oscillator modes. This feature allows users to dynamically toggle the oscillator speed between 4 MHz and 37 kHz. In ER mode, the 4 MHz setting will vary depending on the value of the external resistor. Also in ER mode, the 37 kHz operation is fixed and does not vary with resistor value. Applications that require low current power savings, but cannot tolerate putting the part into SLEEP, may use this mode.

The OSCF bit in the PCON register is used to control Dual Speed mode. See Section 3.2.2.6, Register 3-4.

14.4 RESET

The PIC16F62X differentiates between various kinds of RESET:

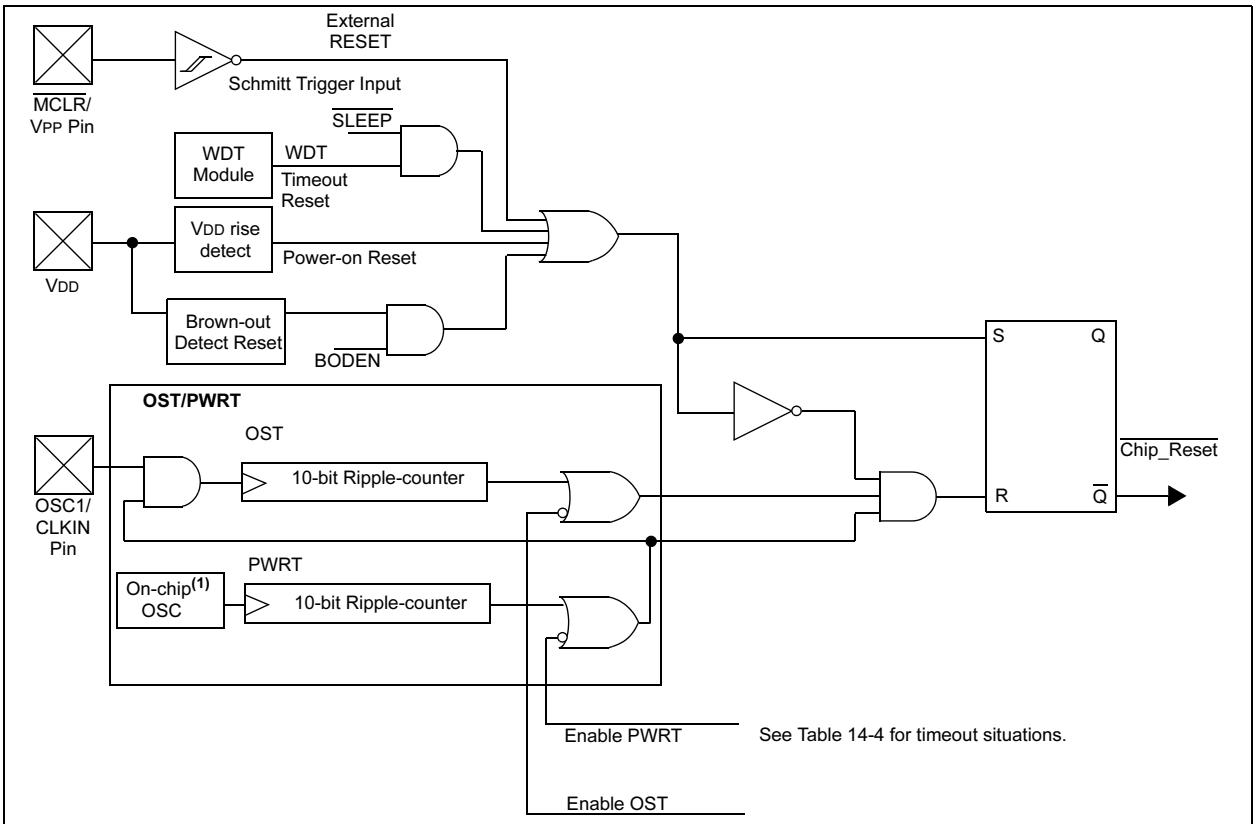
a) Power-on Reset (POR)
b) $\overline{\text{MCLR}}$ Reset during normal operation
c) $\overline{\text{MCLR}}$ Reset during SLEEP
d) WDT Reset (normal operation)
e) WDT Wake-up (SLEEP)
f) Brown-out Detect (BOD)

Some registers are not affected in any RESET condition; their status is unknown on POR and unchanged in any other RESET. Most other registers are reset to a "RESET state" on Power-on Reset, $\overline{\text{MCLR}}$ Reset, WDT Reset and $\overline{\text{MCLR}}$ Reset during SLEEP. They are not affected by a WDT Wake-up, since this is viewed as the resumption of normal operation. $\overline{\text{TO}}$ and $\overline{\text{PD}}$ bits are set or cleared differently in different RESET situations as indicated in Table 14-5. These bits are used in software to determine the nature of the RESET. See Table 14-8 for a full description of RESET states of all registers.

A simplified block diagram of the on-chip RESET circuit is shown in Figure 14-6.

The $\overline{\text{MCLR}}$ Reset path has a noise filter to detect and ignore small pulses. See Table 17-6 for pulse width specification.

FIGURE 14-6: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



Note 1: This is a separate oscillator from the INTRC/ER oscillator.

14.5 Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST) and Brown-out Detect (BOD)

14.5.1 POWER-ON RESET (POR)

The on-chip POR circuit holds the chip in RESET until VDD has reached a high enough level for proper operation. To take advantage of the POR, just tie the MCLR pin through a resistor to VDD. This will eliminate external RC components usually needed to create Power-on Reset. A maximum rise time for VDD is required. See Electrical Specifications for details.

The POR circuit does not produce an internal RESET when VDD declines.

When the device starts normal operation (exits the RESET condition), device operating parameters (voltage, frequency, temperature, etc.) must be met to ensure operation. If these conditions are not met, the device must be held in RESET until the operating conditions are met.

For additional information, refer to Application Note AN607, “Power-up Trouble Shooting”.

14.5.2 POWER-UP TIMER (PWRT)

The PWRT provides a fixed 72 ms (nominal) timeout on power-up only, from POR or Brown-out Detect Reset. The PWRT operates on an internal RC oscillator. The chip is kept in RESET as long as PWRT is active. The PWRT delay allows the VDD to rise to an acceptable level. A configuration bit, PWRTE can disable (if set) or enable (if cleared or programmed) the PWRT. The PWRT should always be enabled when Brown-out Detect Reset is enabled.

The Power-Up Time delay will vary from chip to chip and due to VDD, temperature and process variation. See DC parameters for details.

14.5.3 OSCILLATOR START-UP TIMER (OST)

The OST provides a 1024 oscillator cycle (from OSC1 input) delay after the PWRT delay is over. This ensures that the crystal oscillator or resonator has started and stabilized.

The OST timeout is invoked only for XT, LP and HS modes and only on Power-on Reset or wake-up from SLEEP.

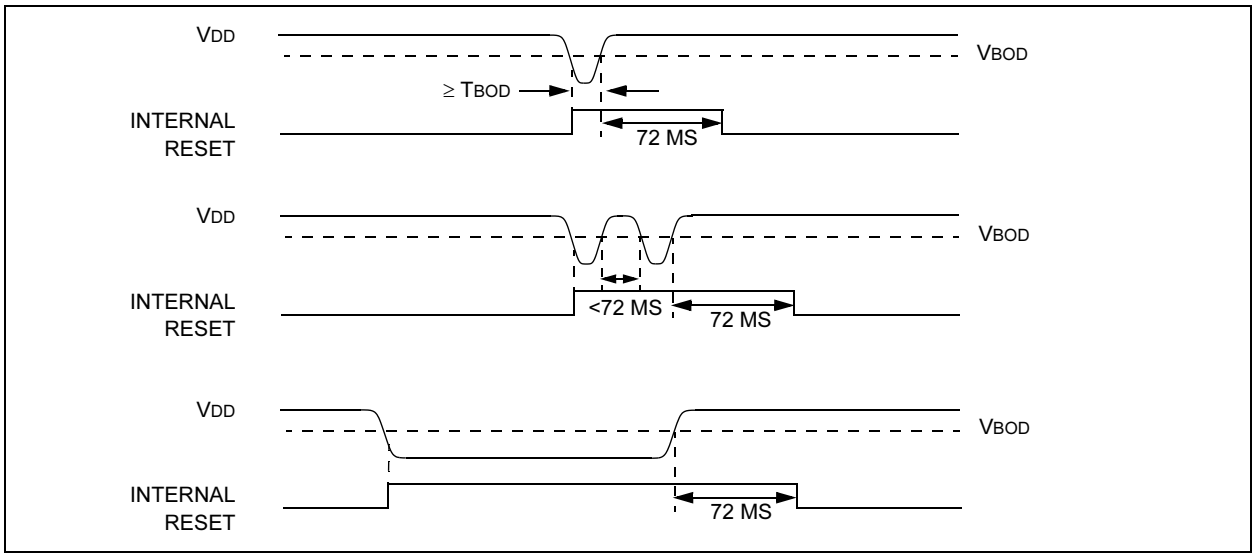
14.5.4 BROWN-OUT DETECT (BOD) RESET

The PIC16F62X members have on-chip BOD circuitry. A configuration bit, BODEN, can disable (if clear/programmed) or enable (if set) the BOD Reset circuitry. If VDD falls below VBOD for longer than TBOD, the brown-out situation will RESET the chip. A RESET is not guaranteed to occur if VDD falls below VBOD for shorter than TBOD. VBOD and TBOD are defined in Table 17-1 and Table 17-6, respectively.

On any RESET (Power-on, Brown-out, Watchdog, etc.) the chip will remain in RESET until VDD rises above VBOD. The Power-up Timer will now be invoked and will keep the chip in RESET an additional 72 ms.

If VDD drops below VBOD while the Power-up Timer is running, the chip will go back into a Brown-out Detect Reset and the Power-up Timer will be re-initialized. Once VDD rises above VBOD, the Power-Up Timer will execute a 72 ms RESET. The Power-up Timer should always be enabled when Brown-out Detect is enabled. Figure 14-7 shows typical Brown-out situations.

FIGURE 14-7: BROWN-OUT SITUATIONS



14.5.5 TIMEOUT SEQUENCE

On power-up the timeout sequence is as follows: First PWRT timeout is invoked after POR has expired. Then OST is activated. The total timeout will vary based on oscillator configuration and $\overline{\text{PWRTE}}$ bit status. For example, in ER mode with $\overline{\text{PWRTE}}$ bit erased (PWRT disabled), there will be no timeout at all. Figure 14-8, Figure 14-9 and Figure 14-10 depict timeout sequences.

Since the timeouts occur from the POR pulse, if $\overline{\text{MCLR}}$ is kept low long enough, the timeouts will expire. Then bringing $\overline{\text{MCLR}}$ high will begin execution immediately (see Figure 14-9). This is useful for testing purposes or to synchronize more than one PIC16F62X device operating in parallel.

Table 14-7 shows the RESET conditions for some special registers, while Table 14-8 shows the RESET conditions for all the registers.

14.5.6 POWER CONTROL (PCON) STATUS REGISTER

The Power Control/STATUS register, PCON (address 8Eh) has two bits.

Bit0 is $\overline{\text{BOD}}$ (Brown-out). $\overline{\text{BOD}}$ is unknown on Power-on Reset. It must then be set by the user and checked on subsequent RESETS to see if $\overline{\text{BOD}}$ = 0 indicating that a brown-out has occurred. The $\overline{\text{BOD}}$ STATUS bit is a don't care and is not necessarily predictable if the brown-out circuit is disabled (by setting BODEN bit = 0 in the Configuration word).

Bit1 is $\overline{\text{POR}}$ (Power-on Reset). It is a '0' on Power-on Reset and unaffected otherwise. The user must write a '1' to this bit following a Power-on Reset. On a subsequent RESET if $\overline{\text{POR}}$ is '0', it will indicate that a Power-on Reset must have occurred (V_{DD} may have gone too low).

TABLE 14-4: TIMEOUT IN VARIOUS SITUATIONS

Oscillator Configuration	Power-up		Brown-out Detect Reset	Wake-up from SLEEP
	$\overline{\text{PWRTE}}$ = 0	$\overline{\text{PWRTE}}$ = 1		
XT, HS, LP	72 ms + 1024 TOSC	1024 TOSC	72 ms + 1024 TOSC	1024 TOSC
ER, INTRC, EC	72 ms	—	72 ms	—

TABLE 14-5: STATUS/PCON BITS AND THEIR SIGNIFICANCE

$\overline{\text{POR}}$	$\overline{\text{BOD}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	
0	X	1	1	Power-on Reset
0	X	0	X	Illegal, $\overline{\text{TO}}$ is set on $\overline{\text{POR}}$
0	X	X	0	Illegal, $\overline{\text{PD}}$ is set on $\overline{\text{POR}}$
1	0	X	X	Brown-out Detect Reset
1	1	0	u	WDT Reset
1	1	0	0	WDT Wake-up
1	1	u	u	$\overline{\text{MCLR}}$ Reset during normal operation
1	1	1	0	$\overline{\text{MCLR}}$ Reset during SLEEP

Legend: u = unchanged, x = unknown.

TABLE 14-6: SUMMARY OF REGISTERS ASSOCIATED WITH BROWN-OUT

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset	Value on all other RESETS(1)
03h	STATUS	IRP	RP1	RPO	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	000q quuu
8Eh	PCON	—	—	—	—	OSCF	Reset	$\overline{\text{POR}}$	$\overline{\text{BOD}}$	---- 1-0x	---- u-uq

Note 1: Other (non Power-up) Resets include $\overline{\text{MCLR}}$ Reset, Brown-out Detect Reset and Watchdog Timer Reset during normal operation.

TABLE 14-7: INITIALIZATION CONDITION FOR SPECIAL REGISTERS

Condition	Program Counter	STATUS Register	PCON Register
Power-on Reset	000h	0001 1xxx	---- 1-0x
$\overline{\text{MCLR}}$ Reset during normal operation	000h	000u uuuu	---- 1-uu
$\overline{\text{MCLR}}$ Reset during SLEEP	000h	0001 0uuu	---- 1-uu
WDT Reset	000h	0000 uuuu	---- 1-uu
WDT Wake-up	PC + 1	uuu0 0uuu	---- u-uu
Brown-out Detect Reset	000h	000x xuuu	---- 1-u0
Interrupt Wake-up from SLEEP	PC + 1(1)	uuu1 0uuu	---- u-uu

Legend: u = unchanged, x = unknown, - = unimplemented bit, reads as '0'.

Note 1: When the wake-up is due to an interrupt and global enable bit, GIE is set, the PC is loaded with the interrupt vector (0004h) after execution of PC+1.

TABLE 14-8: INITIALIZATION CONDITION FOR REGISTERS

Register	Address	Power-on Reset	• $\overline{\text{MCLR}}$ Reset during normal operation • $\overline{\text{MCLR}}$ Reset during SLEEP • WDT Reset • Brown-out Detect Reset (1)	• Wake-up from SLEEP through interrupt • Wake-up from SLEEP through WDT timeout
W	—	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	00h	—	—	—
TMR0	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	02h	0000 0000	0000 0000	PC + 1(3)
STATUS	03h	0001 1xxx	000q quuu(4)	uuuq quuu(4)
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA	05h	xxxx 0000	xxxx u000	xxxx 0000
PORTB	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu
T1CON	10h	--00 0000	--uu uuuu	--uu uuuu
T2CON	12h	-000 0000	-000 0000	-uuu uuuu
CCP1CON	17h	--00 0000	--00 0000	--uu uuuu
RCSTA	18h	0000 -00x	0000 -00x	uuuu -uuu
CMCON	1Fh	0000 0000	0000 0000	uu-- uuuu
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uqqq(2)
PIR1	0Ch	0000 -000	0000 -000	-q-- ----(2,5)
OPTION	81h	1111 1111	1111 1111	uuuu uuuu
TRISA	85h	11-1 1111	11-- 1111	uu-u uuuu
TRISB	86h	1111 1111	1111 1111	uuuu uuuu
PIE1	8Ch	0000 -000	0000 -000	uuuu -uuu
PCON	8Eh	---- 1-0x	---- 1-uq(1,6)	---- --uu
TXSTA	98h	0000 -010	0000 -010	uuuu -uuu
EECON1	9Ch	---- x000	---- q000	---- uuuu
VRCON	9Fh	000- 0000	000- 0000	uuu- uuuu

Legend: u = unchanged, x = unknown, - = unimplemented bit, reads as '0', q = value depends on condition.

Note 1: If VDD goes too low, Power-on Reset will be activated and registers will be affected differently.
2: One or more bits in INTCON, PIR1 and/or PIR2 will be affected (to cause wake-up).
3: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).
4: See Table 14-7 for RESET value for specific condition.
5: If wake-up was due to comparator input changing, then Bit 6 = 1. All other interrupts generating a wake-up will cause Bit 6 = u.
6: If RESET was due to brown-out, then Bit 0 = 0. All other RESETS will cause Bit 0 = u.

FIGURE 14-8: TIMEOUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ NOT TIED TO VDD): CASE

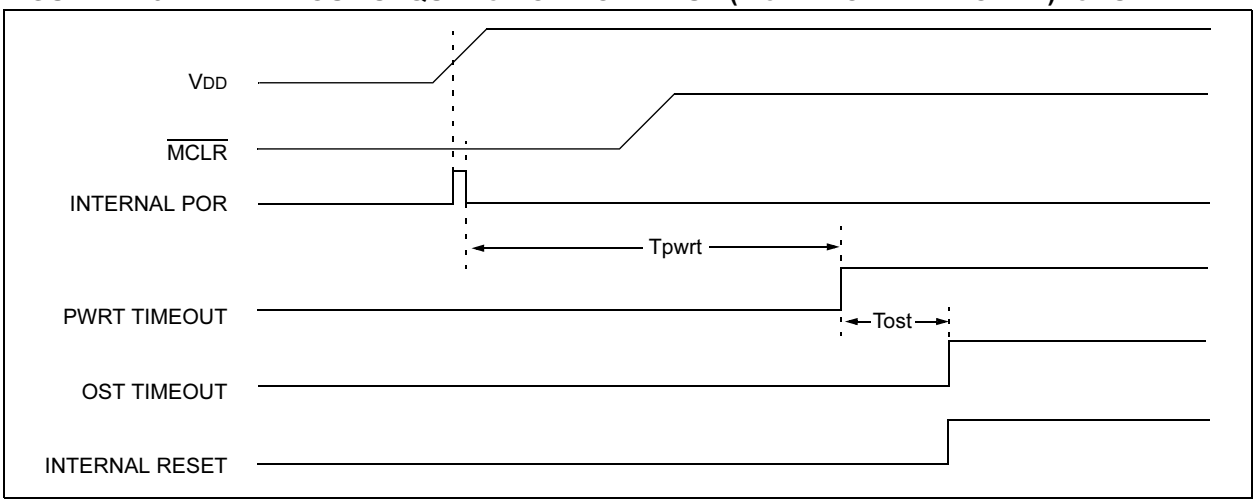


FIGURE 14-9: TIMEOUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ NOT TIED TO VDD): CASE 2

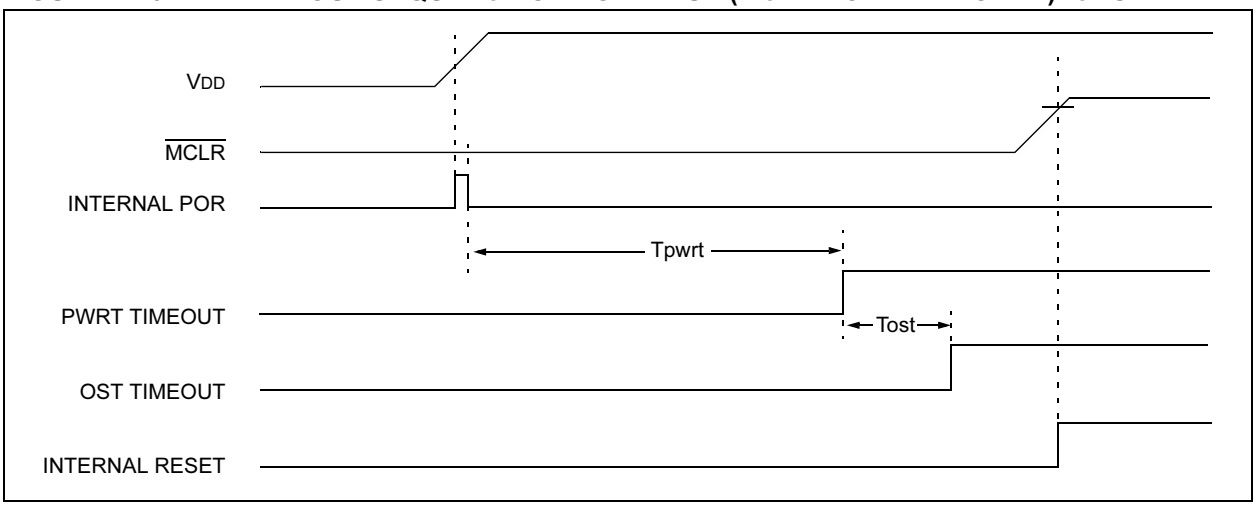


FIGURE 14-10: TIMEOUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ TIED TO VDD)

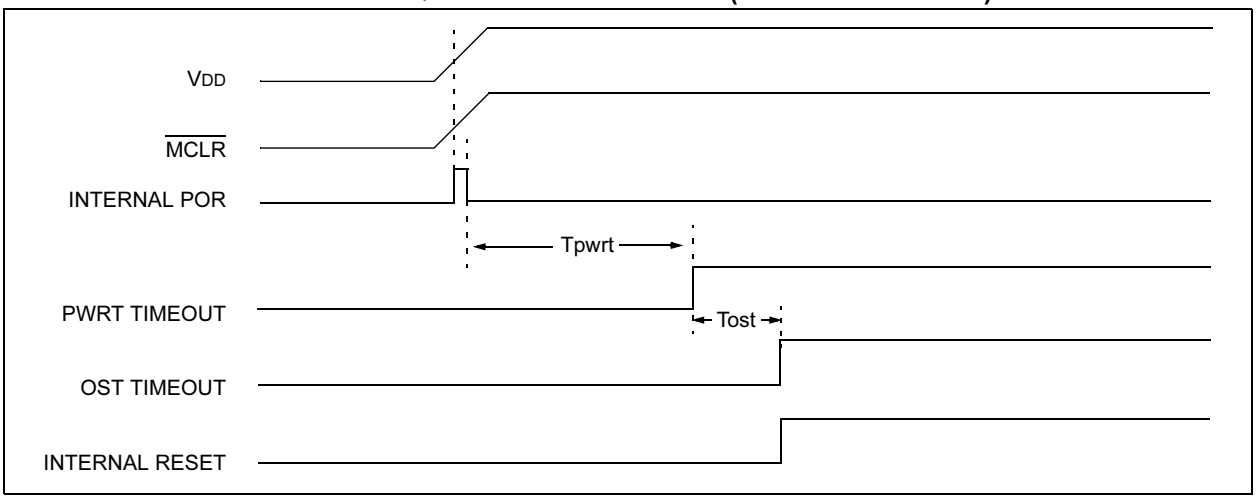
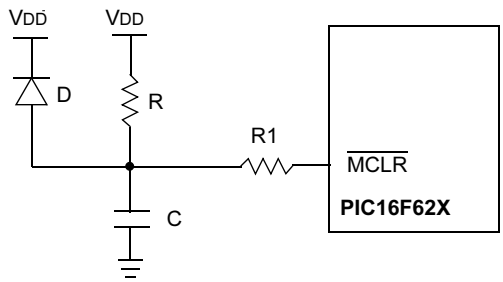


FIGURE 14-11: EXTERNAL POWER-ON RESET CIRCUIT (FOR SLOW VDD POWER-UP)

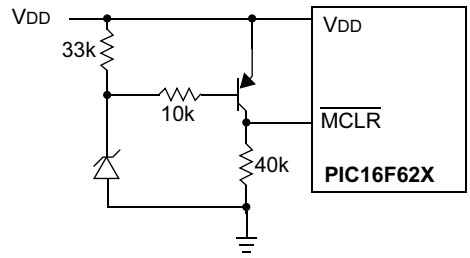


Note 1: External Power-on Reset circuit is required only if VDD power-up slope is too slow. The diode D helps discharge the capacitor quickly when VDD powers down.

2: R < 40 kΩ is recommended to make sure that voltage drop across R does not violate the device's electrical specification.

3: R1 = 100Ω to 1 kΩ will limit any current flowing into $\overline{\text{MCLR}}$ from external capacitor C in the event of $\overline{\text{MCLR}}$/VPP pin breakdown due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS).

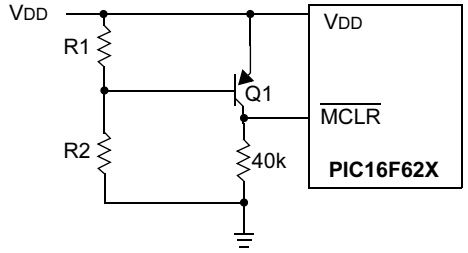
FIGURE 14-12: EXTERNAL BROWN-OUT PROTECTION CIRCUIT 1



Note 1: This circuit will activate RESET when VDD goes below (Vz + 0.7V) where Vz = Zener voltage.

2: Internal Brown-out Detect Reset circuitry should be disabled when using this circuit.

FIGURE 14-13: EXTERNAL BROWN-OUT PROTECTION CIRCUIT 2



Note 1: This Brown-out Circuit is less expensive, albeit less accurate. Transistor Q1 turns off when VDD is below a certain level such that:

$$V_{DD} \times \frac{R1}{R1 + R2} = 0.7V$$

2: Internal Brown-out Detect Reset should be disabled when using this circuit.

3: Resistors should be adjusted for the characteristics of the transistor.

14.6 Interrupts

The PIC16F62X has 10 sources of interrupt:

- External Interrupt RB0/INT
- TMR0 Overflow Interrupt
- PORTB Change Interrupts (pins RB7:RB4)
- Comparator Interrupt
- USART Interrupt TX
- USART Interrupt RX
- CCP Interrupt
- TMR1 Overflow Interrupt
- TMR2 Match Interrupt
- EEPROM

The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also has individual and global interrupt enable bits.

A global interrupt enable bit, GIE (INTCON<7>) enables (if set) all un-masked interrupts or disables (if cleared) all interrupts. Individual interrupts can be disabled through their corresponding enable bits in INTCON register. GIE is cleared on RESET.

The “return from interrupt” instruction, `RETFIE`, exits interrupt routine as well as sets the GIE bit, which re-enable RB0/INT interrupts.

The INT pin interrupt, the RB port change interrupt and the TMR0 overflow interrupt flags are contained in the INTCON register.

The peripheral interrupt flag is contained in the special register PIR1. The corresponding interrupt enable bit is contained in special registers PIE1.

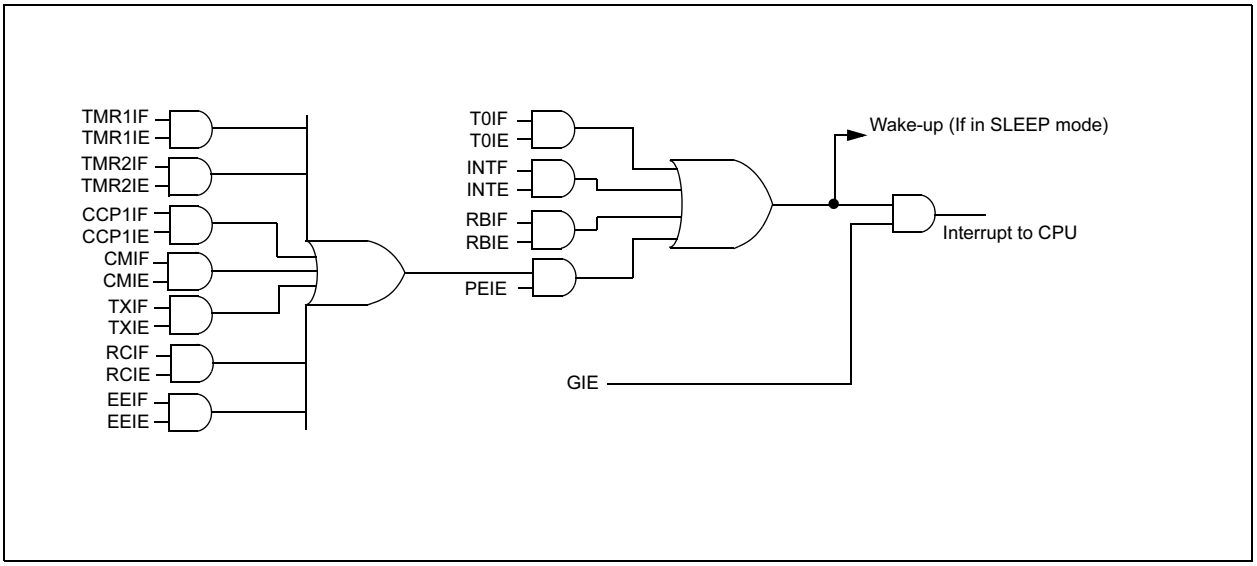
When an interrupt is responded to, the GIE is cleared to disable any further interrupt, the return address is pushed into the stack and the PC is loaded with 0004h. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid RB0/INT recursive interrupts.

For external interrupt events, such as the INT pin or PORTB change interrupt, the interrupt latency will be three or four instruction cycles. The exact latency depends when the interrupt event occurs (Figure 14-15). The latency is the same for one or two cycle instructions. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid multiple interrupt requests. Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

Note 1: Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

2: When an instruction that clears the GIE bit is executed, any interrupts that were pending for execution in the next cycle are ignored. The CPU will execute a `NOP` in the cycle immediately following the instruction which clears the GIE bit. The interrupts which were ignored are still pending to be serviced when the GIE bit is set again.

FIGURE 14-14: INTERRUPT LOGIC



14.6.1 RB0/INT INTERRUPT

External interrupt on RB0/INT pin is edge triggered: either rising if INTEDG bit (OPTION<6>) is set, or falling, if INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, the INTF bit (INTCON<1>) is set. This interrupt can be disabled by clearing the INTE control bit (INTCON<4>). The INTF bit must be cleared in software in the interrupt service routine before re-enabling this interrupt. The RB0/INT interrupt can wake-up the processor from SLEEP, if the INTE bit was set prior to going into SLEEP. The status of the GIE bit decides whether or not the processor branches to the interrupt vector following wake-up. See Section 14.9 for details on SLEEP, and Figure 14-17 for timing of wake-up from SLEEP through RB0/INT interrupt.

14.6.2 TMR0 INTERRUPT

An overflow (FFh → 00h) in the TMR0 register will set the T0IF (INTCON<2>) bit. The interrupt can be enabled/disabled by setting/clearing T0IE (INTCON<5>) bit. For operation of the Timer0 module, see Section 6.0.

14.6.3 PORTB INTERRUPT

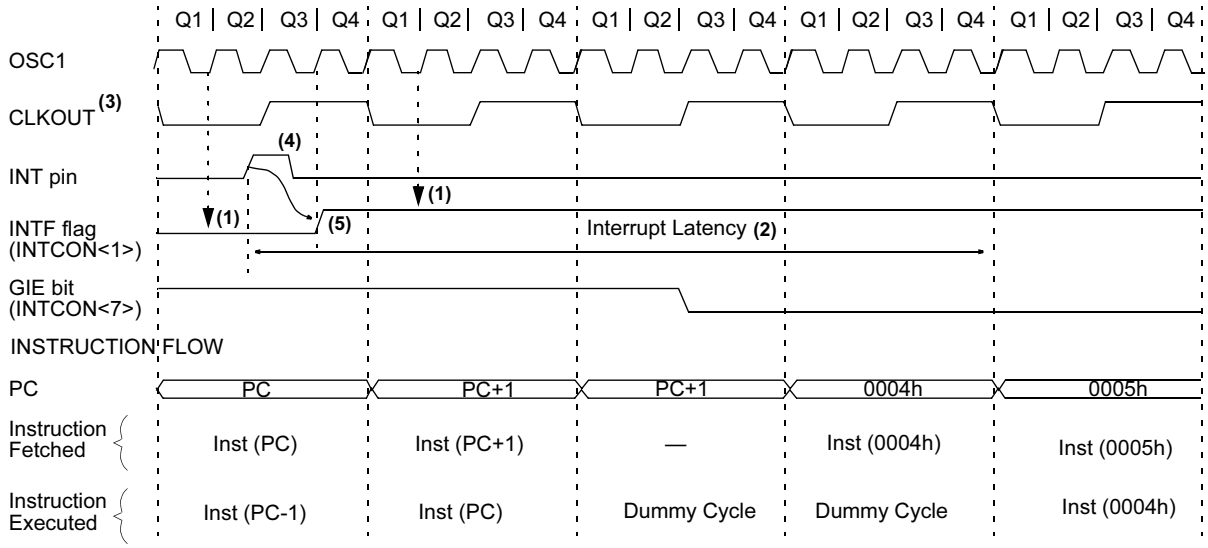
An input change on PORTB <7:4> sets the RBIF (INTCON<0>) bit. The interrupt can be enabled/disabled by setting/clearing the RBIE (INTCON<4>) bit. For operation of PORTB (Section 5.2).

Note:	If a change on the I/O pin should occur when the read operation is being executed (start of the Q2 cycle), then the RBIF interrupt flag may not get set.

14.6.4 COMPARATOR INTERRUPT

See Section 9.6 for complete description of comparator interrupts.

FIGURE 14-15: INT PIN INTERRUPT TIMING



Note 1: INTF flag is sampled here (every Q1).
2: Asynchronous interrupt latency = 3-4 Tcy. Synchronous latency = 3 Tcy, where Tcy = instruction cycle time. Latency is the same whether Inst (PC) is a single cycle or a 2-cycle instruction.
3: CLKOUT is available in ER and INTRC Oscillator mode.
4: For minimum width of INT pulse, refer to AC specs.
5: INTF is enabled to be set anytime during the Q4-Q1 cycles.

TABLE 14-9: SUMMARY OF INTERRUPT REGISTERS

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset	Value on all other RESETS(1)
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	—	CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000

Note 1: Other (non Power-up) Resets include $\overline{\text{MCLR}}$ Reset, Brown-out Detect Reset and Watchdog Timer Reset during normal operation.

14.7 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users may wish to save key registers during an interrupt (e.g., W register and STATUS register). This will have to be implemented in software.

Example 14-2 stores and restores the STATUS and W registers. The user register, W_TEMP, must be defined in a common memory location (i.e., W_TEMP is defined at 0x70 in Bank 0 and is therefore, accessible at 0xF0, 0x17 and 0xIFD). The Example 14-2:

- Stores the W register
- Stores the STATUS register
- Executes the ISR code
- Restores the STATUS (and bank select bit register)
- Restores the W register

EXAMPLE 14-2: SAVING THE STATUS AND W REGISTERS IN RAM

```
MOVWF   W_TEMP          ;copy W to temp register,
                         could be in either bank
SWAPF   STATUS,W        ;swap status to be saved
                         into W
BCF     STATUS,RP0      ;change to bank 0 regardless
                         of current bank
MOVWF   STATUS_TEMP     ;save status to bank 0
                         register
      :
      :  (ISR)
      :
SWAPF   STATUS_TEMP,W   ;swap STATUS_TEMP register
                         into W, sets bank to origi-
                         nal
                         state
MOVWF   STATUS          ;move W into STATUS register
SWAPF   W_TEMP,F        ;swap W_TEMP
SWAPF   W_TEMP,W        ;swap W_TEMP into W
```

14.8 Watchdog Timer (WDT)

The Watchdog Timer is a free running on-chip RC oscillator which does not require any external components. This RC oscillator is separate from the ER oscillator of the CLKIN pin. That means that the WDT will run, even if the clock on the OSC1 and OSC2 pins of the device has been stopped, for example, by execution of a `SLEEP` instruction. During normal operation, a WDT timeout generates a device RESET. If the device is in SLEEP mode, a WDT timeout causes the device to wake-up and continue with normal operation. The WDT can be permanently disabled by programming the configuration bit WDTE as clear (Section 14.1).

14.8.1 WDT PERIOD

The WDT has a nominal timeout period of 18 ms (with no prescaler). The timeout periods vary with temperature, V_{DD} and process variations from part to part (see DC specs). If longer timeout periods are desired, a postscaler with a division ratio of up to 1:128 can be assigned to the WDT under software control by writing to the OPTION register. Thus, timeout periods up to 2.3 seconds can be realized.

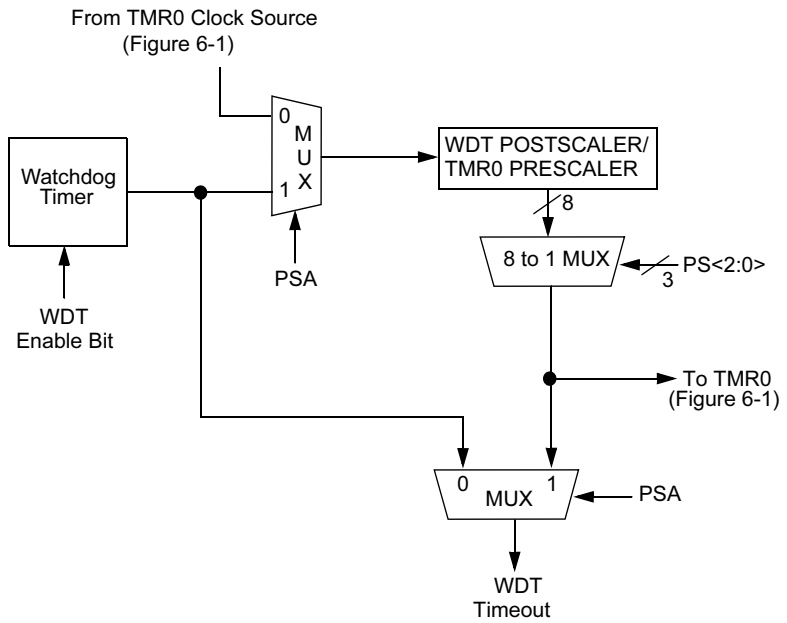
The `CLRWDT` and `SLEEP` instructions clear the WDT and the postscaler, if assigned to the WDT, and prevent it from timing out and generating a device RESET.

The $\overline{\text{TO}}$ bit in the STATUS register will be cleared upon a Watchdog Timer timeout.

14.8.2 WDT PROGRAMMING CONSIDERATIONS

It should also be taken in account that under worst case conditions (V_{DD} = Min., Temperature = Max., max. WDT prescaler), it may take several seconds before a WDT timeout occurs.

FIGURE 14-16: WATCHDOG TIMER BLOCK DIAGRAM



Note 1: T0SE, T0CS, PSA, PS0-PS2 are bits in the OPTION register.

TABLE 14-10: SUMMARY OF WATCHDOG TIMER REGISTERS

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset	Value on all other RESETS
2007h	Config. bits	LVP	BODEN	MCLRE	FOSC2	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0	uuuu uuuu	uuuu uuuu
81h	OPTION	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: – = Unimplemented location, read as “0”, + = Reserved for future use
Note 1: Shaded cells are not used by the Watchdog Timer.

14.9 Power-Down Mode (SLEEP)

The Power-down mode is entered by executing a `SLEEP` instruction.

If enabled, the Watchdog Timer will be cleared but keeps running, the $\overline{\text{PD}}$ bit in the STATUS register is cleared, the $\overline{\text{TO}}$ bit is set, and the oscillator driver is turned off. The I/O ports maintain the status they had, before `SLEEP` was executed (driving high, low, or hi-impedance).

For lowest current consumption in this mode, all I/O pins should be either at V_{DD}, or V_{SS}, with no external circuitry drawing current from the I/O pin and the comparators, and V_{REF} should be disabled. I/O pins that are hi-impedance inputs should be pulled high or low externally to avoid switching currents caused by floating inputs. The T0CKI input should also be at V_{DD} or V_{SS} for lowest current consumption. The contribution from on-chip pull-ups on PORTB should be considered.

The $\overline{\text{MCLR}}$ pin must be at a logic high level (V_{IHMC}).

Note:	It should be noted that a RESET generated by a WDT timeout does not drive $\overline{\text{MCLR}}$ pin low.

14.9.1 WAKE-UP FROM SLEEP

The device can wake-up from SLEEP through one of the following events:

1. External RESET input on $\overline{\text{MCLR}}$ pin
2. Watchdog Timer Wake-up (if WDT was enabled)
3. Interrupt from RB0/INT pin, RB Port change, or the Peripheral Interrupt (Comparator).

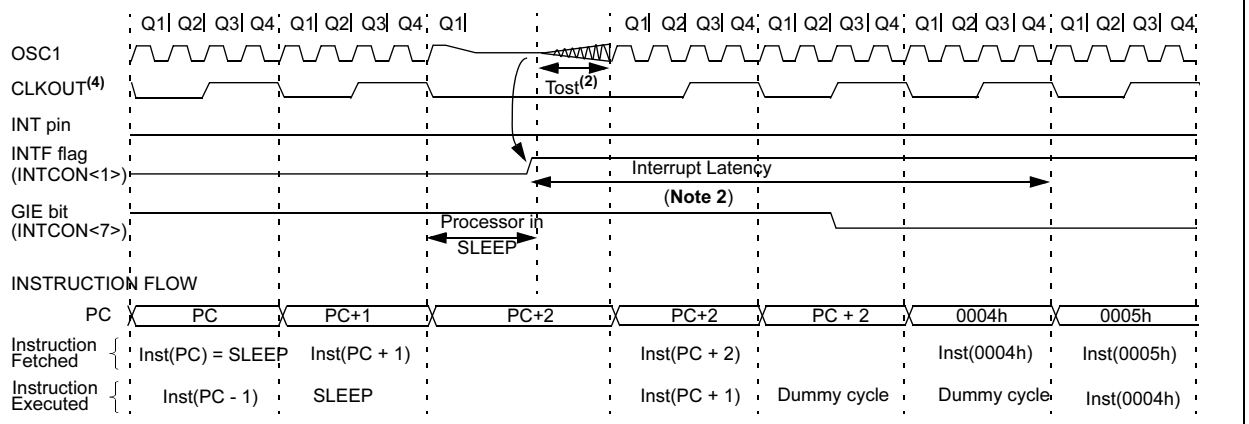
The first event will cause a device RESET. The two latter events are considered a continuation of program execution. The $\overline{\text{TO}}$ and $\overline{\text{PD}}$ bits in the STATUS register can be used to determine the cause of device RESET. $\overline{\text{PD}}$ bit, which is set on power-up is cleared when SLEEP is invoked. $\overline{\text{TO}}$ bit is cleared if WDT Wake-up occurred.

When the `SLEEP` instruction is being executed, the next instruction (PC + 1) is pre-fetched. For the device to wake-up through an interrupt event, the corresponding interrupt enable bit must be set (enabled). Wake-up is regardless of the state of the GIE bit. If the GIE bit is clear (disabled), the device continues execution at the instruction after the `SLEEP` instruction. If the GIE bit is set (enabled), the device executes the instruction after the `SLEEP` instruction and then branches to the interrupt address (0004h). In cases where the execution of the instruction following `SLEEP` is not desirable, the user should have an `NOP` after the `SLEEP` instruction.

> **Note:** If the global interrupts are disabled (GIE is cleared), but any interrupt source has both its interrupt enable bit and the corresponding interrupt flag bits set, the device will immediately wake-up from SLEEP. The `SLEEP` instruction is completely executed.

The WDT is cleared when the device wakes-up from SLEEP, regardless of the source of wake-up.

FIGURE 14-17: WAKE-UP FROM SLEEP THROUGH INTERRUPT



Note 1: XT, HS or LP Oscillator mode assumed.
2: TOST = 1024TOSC (drawing not to scale). Approximately 1 μs delay will be there for ER Osc mode.
3: GIE = '1' assumed. In this case after wake- up, the processor jumps to the interrupt routine. If GIE = '0', execution will continue in-line.
4: CLKOUT is not available in these Osc modes, but shown here for timing reference.

14.10 Code Protection

If the code protection bit(s) have not been programmed, the on-chip program memory can be read out for verification purposes.

> **Note:** The entire data EEPROM and FLASH program memory will be erased when the code protection is turned off. The INTRC calibration data is not erased.

14.11 User ID Locations

Four memory locations (2000h-2003h) are designated as user ID locations where the user can store checksum or other code-identification numbers. These locations are not accessible during normal execution but are readable and writable during program/verify. Only the Least Significant 4 bits of the user ID locations are used.

14.12 In-Circuit Serial Programming

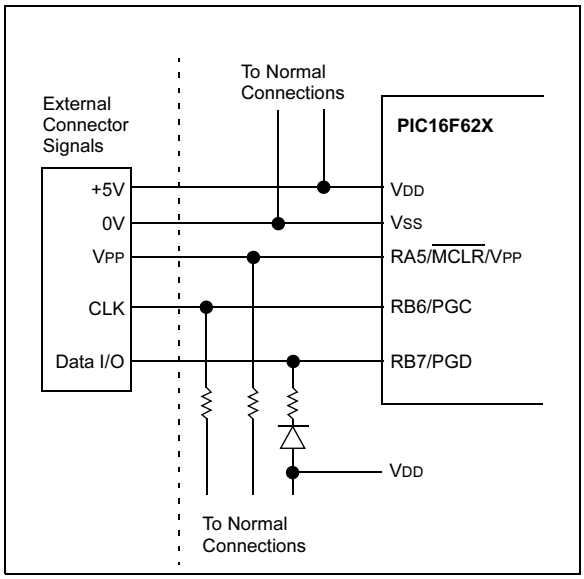
The PIC16F62X microcontrollers can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data, and three other lines for power, ground, and the programming voltage. This allows customers to manufacture boards with unprogrammed devices, and then program the microcontroller just before shipping the product. This also allows the most recent firmware, or a custom firmware to be programmed.

The device is placed into a Program/Verify mode by holding the RB6 and RB7 pins low while raising the $\overline{MCLR}$ (V_{PP}) pin from V_{IL} to V_{IHH} (see programming specification). RB6 becomes the programming clock and RB7 becomes the programming data. Both RB6 and RB7 are Schmitt Trigger inputs in this mode.

After RESET, to place the device into Programming/Verify mode, the program counter (PC) is at location 00h. A 6-bit command is then supplied to the device. Depending on the command, 14 bits of program data are then supplied to or from the device, depending if the command was a load or a read. For complete details of serial programming, please refer to the Programming Specifications.

A typical in-circuit serial programming connection is shown in Figure 14-18.

FIGURE 14-18: TYPICAL IN-CIRCUIT SERIAL PROGRAMMING CONNECTION



14.13 Low Voltage Programming

The LVP bit of the configuration word, enables the low voltage programming. This mode allows the microcontroller to be programmed via ICSP using only a 5V source. This mode removes the requirement of V_{IHH} to be placed on the $\overline{MCLR}$ pin. The LVP bit is normally erased to '1', which enables the low voltage programming. In this mode, the RB4/PGM pin is dedicated to the programming function and ceases to be a general purpose I/O pin. The device will enter Programming mode when a '1' is placed on the RB4/PGM pin. The HV Programming mode is still available by placing V_{IHH} on the $\overline{MCLR}$ pin.

Note 1: While in this mode, the RB4 pin can no longer be used as a general purpose I/O pin.

2: V_{DD} must be 5.0V ±10% during erase/program operations while in low voltage Programming mode.

If Low voltage Programming mode is not used, the LVP bit can be programmed to a '0', and RB4/PGM becomes a digital I/O pin. To program the device, V_{IHH} must be placed onto $\overline{MCLR}$ during programming. The LVP bit may only be programmed when programming is entered with V_{IHH} on $\overline{MCLR}$. The LVP bit cannot be programmed when programming is entered with RB4/PGM.

It should be noted, that once the LVP bit is programmed to 0, High voltage Programming mode can be used to program the device.

15.0 INSTRUCTION SET SUMMARY

Each PIC16F62X instruction is a 14-bit word divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. The PIC16F62X instruction set summary in Table 15-2 lists byte-oriented, bit-oriented, and literal and control operations. Table 15-1 shows the opcode field descriptions.

For byte-oriented instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For bit-oriented instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located.

For literal and control operations, 'k' represents an eight or eleven bit constant or literal value.

TABLE 15-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
label	Label name
TOS	Top of Stack
PC	Program Counter
PCLATH	Program Counter High Latch
GIE	Global Interrupt Enable bit
WDT	Watchdog Timer/Counter
TO	Timeout bit
PD	Power-down bit
`dest`	Destination either the W register or the specified register file location
[ ]	Options
( )	Contents
→	Assigned to
< >	Register bit field
∈	In the set of
italics	User defined term (font is courier)

The instruction set is highly orthogonal and is grouped into three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

All instructions are executed within one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a `NOP`. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 µs. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2 µs.

Table 15-2 lists the instructions recognized by the MPASM™ assembler.

Figure 15-1 shows the three general formats that the instructions can have.

> **Note 1:** Any unused opcode is reserved. Use of any reserved opcode may cause unexpected operation.
>
> **2:** To maintain upward compatibility with future PICmicro® products, do not use the `OPTION` and `TRIS` instructions.

All examples use the following format to represent a hexadecimal number:

0xhh

where h signifies a hexadecimal digit.

FIGURE 15-1: GENERAL FORMAT FOR INSTRUCTIONS

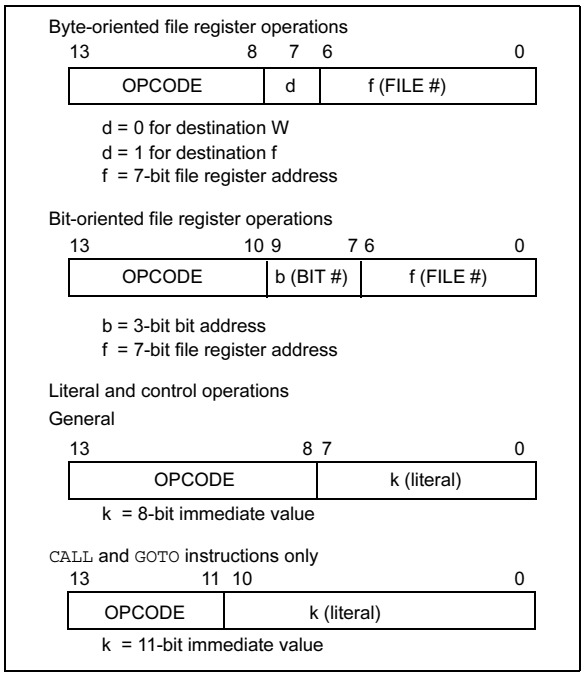


TABLE 15-2: PIC16F62X INSTRUCTION SET

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	1fff	ffff	Z	2
CLRW	—	Clear W	1	00	0001	0000	0011	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	1fff	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`.

15.1 Instruction Descriptions

ADDLW	Add Literal and W
Syntax:	[*label*] ADDLW k
Operands:	$0 \le k \le 255$
Operation:	(W) + k → (W)
Status Affected:	C, DC, Z
Encoding:	11 | 111x | kkkk | kkkk
Description:	The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.
Words:	1
Cycles:	1
Example	`ADDLW 0x15` Before Instruction W = 0x10 After Instruction W = 0x25

ADDWF	Add W and f
Syntax:	[*label*] ADDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) + (f) → (dest)
Status Affected:	C, DC, Z
Encoding:	00 | 0111 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example	`ADDWF REG1, 0` Before Instruction W = 0x17 REG1 = 0xC2 After Instruction W = 0xD9 REG1 = 0xC2 Z = 0 C = 0 DC = 0

ANDLW	AND Literal with W
Syntax:	[*label*] ANDLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .AND. (k) → (W)
Status Affected:	Z
Encoding:	11 | 1001 | kkkk | kkkk
Description:	The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1
Example	`ANDLW 0x5F` Before Instruction W = 0xA3 After Instruction W = 0x03

ANDWF	AND W with f
Syntax:	[*label*] ANDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .AND. (f) → (dest)
Status Affected:	Z
Encoding:	00 | 0101 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example	`ANDWF REG1, 1` Before Instruction W = 0x17 REG1 = 0xC2 After Instruction W = 0x17 REG1 = 0x02

BCF	Bit Clear f
Syntax:	[*label*] BCF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$0 \rightarrow$ (f<b>)
Status Affected:	None
Encoding:	01 | 00bb | bfff | ffff
Description:	Bit 'b' in register 'f' is cleared.
Words:	1
Cycles:	1
Example	`BCF REG1, 7` Before Instruction REG1 = 0xC7 After Instruction REG1 = 0x47

BSF	Bit Set f
Syntax:	[*label*] BSF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$1 \rightarrow$ (f<b>)
Status Affected:	None
Encoding:	01 | 01bb | bfff | ffff
Description:	Bit 'b' in register 'f' is set.
Words:	1
Cycles:	1
Example	`BSF REG1, 7` Before Instruction REG1 = 0x0A After Instruction REG1 = 0x8A

BTFSC	Bit Test f, Skip if Clear
Syntax:	[*label*] BTFSC f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	skip if (f<b>) = 0
Status Affected:	None
Encoding:	01 | 10bb | bfff | ffff
Description:	If bit 'b' in register 'f' is '0' then the next instruction is skipped. If bit 'b' is '0' then the next instruction fetched during the current instruction execution is discarded, and a `NOP` is executed instead, making this a two-cycle instruction.
Words:	1
Cycles:	1(2)

Example	`HERE BTFSC REG1`
	`FALSE GOTO PROCESS_CODE`
	`TRUE` •
	•
	•
	Before Instruction
	PC = address `HERE`
	After Instruction
	if REG<1> = 0,
	PC = address `TRUE`
	if REG<1>=1,
	PC = address `FALSE`

BTFSS	Bit Test f, Skip if Set
Syntax:	[*label*] BTFSS f,b
Operands:	$0 \le f \le 127$ $0 \le b < 7$
Operation:	skip if (f<b>) = 1
Status Affected:	None
Encoding:	01 | 11bb | bfff | ffff
Description:	If bit 'b' in register 'f' is '1' then the next instruction is skipped. If bit 'b' is '1', then the next instruction fetched during the current instruction execution, is discarded and a NOP is executed instead, making this a two-cycle instruction.
Words:	1
Cycles:	1(2)
Example	HERE BTFSS REG1 FALSE GOTO PROCESS_CODE TRUE • • • Before Instruction PC = address HERE After Instruction if FLAG<1> = 0, PC = address FALSE if FLAG<1> = 1, PC = address TRUE

CALL	Call Subroutine
Syntax:	[*label*] CALL k
Operands:	$0 \le k \le 2047$
Operation:	(PC)+ 1→ TOS, k → PC<10:0>, (PCLATH<4:3>) → PC<12:11>
Status Affected:	None
Encoding:	10 | 0kkk | kkkk | kkkk
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.
Words:	1
Cycles:	2
Example	HERE CALL THERE Before Instruction PC = Address HERE After Instruction PC = Address THERE TOS = Address HERE+1

CLRF	Clear f
Syntax:	[*label*] CLRF f
Operands:	$0 \le f \le 127$
Operation:	00h → (f) 1 → Z
Status Affected:	Z
Encoding:	00 | 0001 | 1fff | ffff
Description:	The contents of register 'f' are cleared and the Z bit is set.
Words:	1
Cycles:	1
Example	CLRF REG1 Before Instruction REG1 = 0x5A After Instruction REG1 = 0x00 Z = 1

CLRW	Clear W
Syntax:	[*label*] CLRW
Operands:	None
Operation:	00h → (W) 1 → Z
Status Affected:	Z
Encoding:	00 | 0001 | 0000 | 0011
Description:	W register is cleared. Zero bit (Z) is set.
Words:	1
Cycles:	1
Example	`CLRW` Before Instruction W = 0x5A After Instruction W = 0x00 Z = 1

CLRWDT	Clear Watchdog Timer
Syntax:	[*label*] CLRWDT
Operands:	None
Operation:	00h → WDT 0 → WDT prescaler, 1 → $\overline{TO}$ 1 → $\overline{PD}$
Status Affected:	$\overline{TO}$, $\overline{PD}$
Encoding:	00 | 0000 | 0110 | 0100
Description:	`CLRWDT` instruction resets the Watchdog Timer. It also resets the prescaler of the WDT. STATUS bits $\overline{TO}$ and $\overline{PD}$ are set.
Words:	1
Cycles:	1
Example	`CLRWDT` Before Instruction WDT counter = ? After Instruction WDT counter = 0x00 WDT prescaler = 0 $\overline{TO}$ = 1 $\overline{PD}$ = 1

COMF	Complement f
Syntax:	[*label*] COMF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	$(\bar{f}) \rightarrow$ (dest)
Status Affected:	Z
Encoding:	00 | 1001 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example	`COMF REG1, 0` Before Instruction REG1 = 0x13 After Instruction REG1 = 0x13 W = 0xEC

DECF	Decrement f
Syntax:	[*label*] DECF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) - 1 → (dest)
Status Affected:	Z
Encoding:	00 | 0011 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example	`DECF CNT, 1` Before Instruction CNT = 0x01 Z = 0 After Instruction CNT = 0x00 Z = 1

DECFSZ	Decrement f, Skip if 0
Syntax:	[*label*] DECFSZ f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) - 1 → (dest); skip if result = 0
Status Affected:	None
Encoding:	00 | 1011 | dfff | ffff
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 0, the next instruction, which is already fetched, is discarded. A NOP is executed instead making it a two-cycle instruction.
Words:	1
Cycles:	1(2)
Example	HERE DECFSZ REG1, 1 GOTO LOOP CONTINUE • • • Before Instruction PC = address HERE After Instruction REG1 = REG1 - 1 if REG1 = 0, PC = address CONTINUE if REG1 ≠ 0, PC = address HERE+1

GOTO	Unconditional Branch
Syntax:	[*label*] GOTO k
Operands:	$0 \le k \le 2047$
Operation:	k → PC<10:0> PCLATH<4:3> → PC<12:11>
Status Affected:	None
Encoding:	10 | 1kkk | kkkk | kkkk
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.
Words:	1
Cycles:	2
Example	GOTO THERE After Instruction PC = Address THERE

INCF	Increment f
Syntax:	[*label*] INCF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) + 1 → (dest)
Status Affected:	Z
Encoding:	00 | 1010 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example	`INCF REG1, 1` Before Instruction REG1 = 0xFF Z = 0 After Instruction REG1 = 0x00 Z = 1

INCFSZ	Increment f, Skip if 0
Syntax:	[*label*] INCFSZ f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) + 1 → (dest), skip if result = 0
Status Affected:	None
Encoding:	00 | 1111 | dfff | ffff
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 0, the next instruction, which is already fetched, is discarded. A `NOP` is executed instead making it a two-cycle instruction.
Words:	1
Cycles:	1(2)

Example	`HERE INCFSZ REG1, 1`
	`GOTO LOOP`
	`CONTINUE` •
	•
	•
	Before Instruction
	PC = address `HERE`
	After Instruction
	REG1 = REG1 + 1
	if CNT = 0,
	PC = address `CONTINUE`
	if REG1 ≠ 0,
	PC = address `HERE +1`

IORLW	Inclusive OR Literal with W
Syntax:	[*label*] IORLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .OR. k → (W)
Status Affected:	Z
Encoding:	11 | 1000 | kkkk | kkkk
Description:	The contents of the W register is OR'ed with the eight bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1
Example	`IORLW 0x35` Before Instruction W = 0x9A After Instruction W = 0xBF Z = 0

MOVLW	Move Literal to W
Syntax:	[*label*] MOVLW k
Operands:	$0 \le k \le 255$
Operation:	k → (W)
Status Affected:	None
Encoding:	11 | 00xx | kkkk | kkkk
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.
Words:	1
Cycles:	1
Example	`MOVLW 0x5A` After Instruction W = 0x5A

IORWF	Inclusive OR W with f
Syntax:	[*label*] IORWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .OR. (f) → (dest)
Status Affected:	Z
Encoding:	00 | 0100 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example	`IORWF REG1, 0` Before Instruction REG1 = 0x13 W = 0x91 After Instruction REG1 = 0x13 W = 0x93 Z = 1

MOVF	Move f
Syntax:	[*label*] MOVF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) → (dest)
Status Affected:	Z
Encoding:	00 | 1000 | dfff | ffff
Description:	The contents of register f is moved to a destination dependent 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.
Words:	1
Cycles:	1
Example	`MOVF REG1, 0` After Instruction W = value in REG1 register Z = 1

MOVWF	Move W to f
Syntax:	[*label*] MOVWF f
Operands:	$0 \le f \le 127$
Operation:	(W) → (f)
Status Affected:	None
Encoding:	00 | 0000 | 1fff | ffff
Description:	Move data from W register to register 'f'.
Words:	1
Cycles:	1
Example	`MOVWF REG1` Before Instruction REG1 = 0xFF W = 0x4F After Instruction REG1 = 0x4F W = 0x4F

OPTION	Load Option Register
Syntax:	[*label*] OPTION
Operands:	None
Operation:	(W) → OPTION
Status Affected:	None
Encoding:	00 | 0000 | 0110 | 0010
Description:	The contents of the W register are loaded in the OPTION register. This instruction is supported for code compatibility with PIC16C5X products. Since OPTION is a readable/writable register, the user can directly address it. Using only register instruction such as `MOVWF`.
Words:	1
Cycles:	1
Example	**To maintain upward compatibility with future PICmicro® products, do not use this instruction.**

NOP	No Operation
Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Encoding:	00 | 0000 | 0xx0 | 0000
Description:	No operation.
Words:	1
Cycles:	1
Example	`NOP`

RETFIE	Return from Interrupt
Syntax:	[*label*] RETFIE
Operands:	None
Operation:	TOS → PC, 1 → GIE
Status Affected:	None
Encoding:	00 | 0000 | 0000 | 1001
Description:	Return from Interrupt. Stack is POPed and Top of Stack (TOS) is loaded in the PC. Interrupts are enabled by setting Global Interrupt Enable bit, GIE (INTCON<7>). This is a two-cycle instruction.
Words:	1
Cycles:	2
Example	`RETFIE` After Interrupt PC = TOS GIE = 1

RETLW	Return with Literal in W
Syntax:	[*label*] RETLW k
Operands:	$0 \le k \le 255$
Operation:	k → (W); TOS → PC
Status Affected:	None
Encoding:	11 | 01xx | kkkk | kkkk
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.
Words:	1
Cycles:	2

Example

```
CALL TABLE;W contains table
          ;offset value
•         ;W now has table
value
•
•
ADDWF PC  ;W = offset
RETLW k1  ;Begin table
RETLW k2  ;
•
•
•
RETLW kn  ; End of table
```

TABLE

Before Instruction
W = 0x07
After Instruction
W = value of k8

RETURN	Return from Subroutine
Syntax:	[*label*] RETURN
Operands:	None
Operation:	TOS → PC
Status Affected:	None
Encoding:	00 | 0000 | 0000 | 1000
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.
Words:	1
Cycles:	2

Example

```
RETURN
```

After Interrupt
PC = TOS

RLF	Rotate Left f through Carry
Syntax:	[*label*] RLF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C
Encoding:	00 | 1101 | dfff | ffff
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'.

Words:	1
Cycles:	1

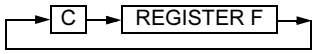
Example

```
RLF     REG1, 0
```

Before Instruction
REG1 = 1110 0110
C = 0
After Instruction
REG1 = 1110 0110
W = 1100 1100
C = 1

RRF — Rotate Right f through Carry

Syntax:	[*label*] RRF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C
Encoding:	00 | 1100 | dfff | ffff
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'.



Words:	1
Cycles:	1
Example	`RRF REG1, 0`

Before Instruction
REG1 = 1110 0110
C = 0
After Instruction
REG1 = 1110 0110
W = 0111 0011
C = 0

SLEEP

Syntax:	[*label*] SLEEP
Operands:	None
Operation:	00h → WDT, 0 → $\overline{\text{WDT}}$ prescaler, 1 → $\overline{\text{TO}}$, 0 → $\overline{\text{PD}}$
Status Affected:	$\overline{\text{TO}}$, $\overline{\text{PD}}$
Encoding:	00 | 0000 | 0110 | 0011
Description:	The power-down STATUS bit, $\overline{\text{PD}}$ is cleared. Timeout STATUS bit, $\overline{\text{TO}}$ is set. Watchdog Timer and its prescaler are cleared. The processor is put into SLEEP mode with the oscillator stopped. See Section 14.9 for more details.
Words:	1
Cycles:	1
Example:	`SLEEP`

SUBLW — Subtract W from Literal

Syntax:	[*label*] SUBLW k
Operands:	$0 \le k \le 255$
Operation:	k - (W) → (W)
Status Affected:	C, DC, Z
Encoding:	11 | 110x | kkkk | kkkk
Description:	The W register is subtracted (2's complement method) from the eight bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1
Example 1:	`SUBLW 0x02`

Before Instruction
W = 1
C = ?
After Instruction
W = 1
C = 1; result is positive

Example 2: Before Instruction
W = 2
C = ?
After Instruction
W = 0
C = 1; result is zero

Example 3: Before Instruction
W = 3
C = ?
After Instruction
W = 0xFF
C = 0; result is negative

SUBWF	Subtract W from f
Syntax:	[*label*] SUBWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) - (W) → (dest)
Status Affected:	C, DC, Z
Encoding:	00 | 0010 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example 1:	`SUBWF REG1, 1` Before Instruction REG1 = 3 W = 2 C = ? After Instruction REG1 = 1 W = 2 C = 1; result is positive Z = DC = 1
Example 2:	Before Instruction REG1 = 2 W = 2 C = ? After Instruction REG1 = 0 W = 2 C = 1; result is zero Z = DC = 1
Example 3:	Before Instruction REG1 = 1 W = 2 C = ? After Instruction REG1 = 0xFF W = 2 C = 0; result is negative Z = DC = 0

SWAPF	Swap Nibbles in f
Syntax:	[*label*] SWAPF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f<3:0>) → (dest<7:4>), (f<7:4>) → (dest<3:0>)
Status Affected:	None
Encoding:	00 | 1110 | dfff | ffff
Description:	The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0 the result is placed in W register. If 'd' is 1 the result is placed in register 'f'.
Words:	1
Cycles:	1
Example	`SWAPF REG1, 0` Before Instruction REG1 = 0xA5 After Instruction REG1 = 0xA5 W = 0x5A

TRIS	Load TRIS Register
Syntax:	[*label*] TRIS f
Operands:	$5 \le f \le 7$
Operation:	(W) → TRIS register f;
Status Affected:	None
Encoding:	00 | 0000 | 0110 | 0fff
Description:	The instruction is supported for code compatibility with the PIC16C5X products. Since TRIS registers are readable and writable, the user can directly address them.
Words:	1
Cycles:	1
Example	**To maintain upward compatibility with future PICmicro® products, do not use this instruction.**

XORLW	Exclusive OR Literal with W
Syntax:	[*label*] XORLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .XOR. k → (W)
Status Affected:	Z
Encoding:	11 | 1010 | kkkk | kkkk
Description:	The contents of the W register are XOR'ed with the eight bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1
Example:	`XORLW 0xAF` Before Instruction W = 0xB5 After Instruction W = 0x1A

XORWF	Exclusive OR W with f
Syntax:	[*label*] XORWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .XOR. (f) → (dest)
Status Affected:	Z
Encoding:	00 | 0110 | dfff | ffff
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'.
Words:	1
Cycles:	1
Example	`XORWF REG1, 1` Before Instruction REG1 = 0xAF W = 0xB5 After Instruction REG1 = 0x1A W = 0xB5

16.0 DEVELOPMENT SUPPORT

The PICmicro® microcontrollers are supported with a full range of hardware and software development tools:

- Integrated Development Environment
  - MPLAB® IDE Software
- Assemblers/Compilers/Linkers
  - MPASM™ Assembler
  - MPLAB C17 and MPLAB C18 C Compilers
  - MPLINK™ Object Linker/ MPLIB™ Object Librarian
  - MPLAB C30 C Compiler
  - MPLAB ASM30 Assembler/Linker/Library
- Simulators
  - MPLAB SIM Software Simulator
  - MPLAB dsPIC30 Software Simulator
- Emulators
  - MPLAB ICE 2000 In-Circuit Emulator
  - MPLAB ICE 4000 In-Circuit Emulator
- In-Circuit Debugger
  - MPLAB ICD 2
- Device Programmers
  - PRO MATE® II Universal Device Programmer
  - PICSTART® Plus Development Programmer
- Low Cost Demonstration Boards
  - PICDEM™ 1 Demonstration Board
  - PICDEM.net™ Demonstration Board
  - PICDEM 2 Plus Demonstration Board
  - PICDEM 3 Demonstration Board
  - PICDEM 17 Demonstration Board
  - PICDEM 18R Demonstration Board
  - PICDEM LIN Demonstration Board
  - PICDEM USB Demonstration Board
- Evaluation Kits
  - KEELOQ®
  - PICDEM MSC
  - microID®
  - CAN
  - PowerSmart®
  - Analog

16.1 MPLAB Integrated Development Environment Software

The MPLAB IDE software brings an ease of software development previously unseen in the 8/16-bit microcontroller market. The MPLAB IDE is a Windows® based application that contains:

- An interface to debugging tools
  - simulator
  - programmer (sold separately)
  - emulator (sold separately)
  - in-circuit debugger (sold separately)
- A full-featured editor with color coded context
- A multiple project manager
- Customizable data windows with direct edit of contents
- High level source code debugging
- Mouse over variable inspection
- Extensive on-line help

The MPLAB IDE allows you to:

- Edit your source files (either assembly or C)
- One touch assemble (or compile) and download to PICmicro emulator and simulator tools (automatically updates all project information)
- Debug using:
  - source files (assembly or C)
  - absolute listing file (mixed assembly and C)
  - machine code

MPLAB IDE supports multiple debugging tools in a single development paradigm, from the cost effective simulators, through low cost in-circuit debuggers, to full-featured emulators. This eliminates the learning curve when upgrading to tools with increasing flexibility and power.

16.2 MPASM Assembler

The MPASM assembler is a full-featured, universal macro assembler for all PICmicro MCUs.

The MPASM assembler generates relocatable object files for the MPLINK object linker, Intel® standard HEX files, MAP files to detail memory usage and symbol reference, absolute LST files that contains source lines and generated machine code and COFF files for debugging.

The MPASM assembler features include:

- Integration into MPLAB IDE projects
- User defined macros to streamline assembly code
- Conditional assembly for multi-purpose source files
- Directives that allow complete control over the assembly process

16.3 MPLAB C17 and MPLAB C18 C Compilers

The MPLAB C17 and MPLAB C18 Code Development Systems are complete ANSI C compilers for Microchip's PIC17CXXX and PIC18CXXX family of microcontrollers. These compilers provide powerful integration capabilities, superior code optimization and ease of use not found with other compilers.

For easy source level debugging, the compilers provide symbol information that is optimized to the MPLAB IDE debugger.

16.4 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK object linker combines relocatable objects created by the MPASM assembler and the MPLAB C17 and MPLAB C18 C compilers. It can link relocatable objects from pre-compiled libraries, using directives from a linker script.

The MPLIB object librarian manages the creation and modification of library files of pre-compiled code. When a routine from a library is called from a source file, only the modules that contain that routine will be linked in with the application. This allows large libraries to be used efficiently in many different applications.

The object linker/library features include:

- Efficient linking of single libraries instead of many smaller files
- Enhanced code maintainability by grouping related modules together
- Flexible creation of libraries with easy module listing, replacement, deletion and extraction

16.5 MPLAB C30 C Compiler

The MPLAB C30 C compiler is a full-featured, ANSI compliant, optimizing compiler that translates standard ANSI C programs into dsPIC30F assembly language source. The compiler also supports many command-line options and language extensions to take full advantage of the dsPIC30F device hardware capabilities, and afford fine control of the compiler code generator.

MPLAB C30 is distributed with a complete ANSI C standard library. All library functions have been validated and conform to the ANSI C library standard. The library includes functions for string manipulation, dynamic memory allocation, data conversion, time-keeping, and math functions (trigonometric, exponential and hyperbolic). The compiler provides symbolic information for high level source debugging with the MPLAB IDE.

16.6 MPLAB ASM30 Assembler, Linker, and Librarian

MPLAB ASM30 assembler produces relocatable machine code from symbolic assembly language for dsPIC30F devices. MPLAB C30 compiler uses the assembler to produce it's object file. The assembler generates relocatable object files that can then be archived or linked with other relocatable object files and archives to create an executable file. Notable features of the assembler include:

- Support for the entire dsPIC30F instruction set
- Support for fixed-point and floating-point data
- Command line interface
- Rich directive set
- Flexible macro language
- MPLAB IDE compatibility

16.7 MPLAB SIM Software Simulator

The MPLAB SIM software simulator allows code development in a PC hosted environment by simulating the PICmicro series microcontrollers on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a file, or user defined key press, to any pin. The execution can be performed in Single-Step, Execute Until Break, or Trace mode.

The MPLAB SIM simulator fully supports symbolic debugging using the MPLAB C17 and MPLAB C18 C Compilers, as well as the MPASM assembler. The software simulator offers the flexibility to develop and debug code outside of the laboratory environment, making it an excellent, economical software development tool.

16.8 MPLAB SIM30 Software Simulator

The MPLAB SIM30 software simulator allows code development in a PC hosted environment by simulating the dsPIC30F series microcontrollers on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a file, or user defined key press, to any of the pins.

The MPLAB SIM30 simulator fully supports symbolic debugging using the MPLAB C30 C Compiler and MPLAB ASM30 assembler. The simulator runs in either a Command Line mode for automated tasks, or from MPLAB IDE. This high speed simulator is designed to debug, analyze and optimize time intensive DSP routines.

16.9 MPLAB ICE 2000 High Performance Universal In-Circuit Emulator

The MPLAB ICE 2000 universal in-circuit emulator is intended to provide the product development engineer with a complete microcontroller design tool set for PICmicro microcontrollers. Software control of the MPLAB ICE 2000 in-circuit emulator is advanced by the MPLAB Integrated Development Environment, which allows editing, building, downloading and source debugging from a single environment.

The MPLAB ICE 2000 is a full-featured emulator system with enhanced trace, trigger and data monitoring features. Interchangeable processor modules allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the MPLAB ICE in-circuit emulator allows expansion to support new PICmicro microcontrollers.

The MPLAB ICE 2000 in-circuit emulator system has been designed as a real-time emulation system with advanced features that are typically found on more expensive development tools. The PC platform and Microsoft® Windows 32-bit operating system were chosen to best make these features available in a simple, unified application.

16.10 MPLAB ICE 4000 High Performance Universal In-Circuit Emulator

The MPLAB ICE 4000 universal in-circuit emulator is intended to provide the product development engineer with a complete microcontroller design tool set for high-end PICmicro microcontrollers. Software control of the MPLAB ICE in-circuit emulator is provided by the MPLAB Integrated Development Environment, which allows editing, building, downloading and source debugging from a single environment.

The MPLAB ICD 4000 is a premium emulator system, providing the features of MPLAB ICE 2000, but with increased emulation memory and high speed performance for dsPIC30F and PIC18XXXX devices. Its advanced emulator features include complex triggering and timing, up to 2 Mb of emulation memory, and the ability to view variables in real-time.

The MPLAB ICE 4000 in-circuit emulator system has been designed as a real-time emulation system with advanced features that are typically found on more expensive development tools. The PC platform and Microsoft Windows 32-bit operating system were chosen to best make these features available in a simple, unified application.

16.11 MPLAB ICD 2 In-Circuit Debugger

Microchip's In-Circuit Debugger, MPLAB ICD 2, is a powerful, low cost, run-time development tool, connecting to the host PC via an RS-232 or high speed USB interface. This tool is based on the FLASH PICmicro MCUs and can be used to develop for these and other PICmicro microcontrollers. The MPLAB ICD 2 utilizes the in-circuit debugging capability built into the FLASH devices. This feature, along with Microchip's In-Circuit Serial Programming™ (ICSP™) protocol, offers cost effective in-circuit FLASH debugging from the graphical user interface of the MPLAB Integrated Development Environment. This enables a designer to develop and debug source code by setting breakpoints, single-stepping and watching variables, CPU status and peripheral registers. Running at full speed enables testing hardware and applications in real-time. MPLAB ICD 2 also serves as a development programmer for selected PICmicro devices.

16.12 PRO MATE II Universal Device Programmer

The PRO MATE II is a universal, CE compliant device programmer with programmable voltage verification at V_{DDMIN} and V_{DDMAX} for maximum reliability. It features an LCD display for instructions and error messages and a modular detachable socket assembly to support various package types. In Stand-Alone mode, the PRO MATE II device programmer can read, verify, and program PICmicro devices without a PC connection. It can also set code protection in this mode.

16.13 PICSTART Plus Development Programmer

The PICSTART Plus development programmer is an easy-to-use, low cost, prototype programmer. It connects to the PC via a COM (RS-232) port. MPLAB Integrated Development Environment software makes using the programmer simple and efficient. The PICSTART Plus development programmer supports most PICmicro devices up to 40 pins. Larger pin count devices, such as the PIC16C92X and PIC17C76X, may be supported with an adapter socket. The PICSTART Plus development programmer is CE compliant.

16.14 PICDEM 1 PICmicro Demonstration Board

The PICDEM 1 demonstration board demonstrates the capabilities of the PIC16C5X (PIC16C54 to PIC16C58A), PIC16C61, PIC16C62X, PIC16C71, PIC16C8X, PIC17C42, PIC17C43 and PIC17C44. All necessary hardware and software is included to run basic demo programs. The sample microcontrollers provided with the PICDEM 1 demonstration board can be programmed with a PRO MATE II device programmer, or a PICSTART Plus development programmer. The PICDEM 1 demonstration board can be connected to the MPLAB ICE in-circuit emulator for testing. A prototype area extends the circuitry for additional application components. Features include an RS-232 interface, a potentiometer for simulated analog input, push button switches and eight LEDs.

16.15 PICDEM.net Internet/Ethernet Demonstration Board

The PICDEM.net demonstration board is an Internet/Ethernet demonstration board using the PIC18F452 microcontroller and TCP/IP firmware. The board supports any 40-pin DIP device that conforms to the standard pinout used by the PIC16F877 or PIC18C452. This kit features a user friendly TCP/IP stack, web server with HTML, a 24L256 Serial EEPROM for Xmodem download to web pages into Serial EEPROM, ICSP/MPLAB ICD 2 interface connector, an Ethernet interface, RS-232 interface, and a 16 x 2 LCD display. Also included is the book and CD-ROM *"TCP/IP Lean, Web Servers for Embedded Systems,"* by Jeremy Bentham

16.16 PICDEM 2 Plus Demonstration Board

The PICDEM 2 Plus demonstration board supports many 18-, 28-, and 40-pin microcontrollers, including PIC16F87X and PIC18FXX2 devices. All the necessary hardware and software is included to run the demonstration programs. The sample microcontrollers provided with the PICDEM 2 demonstration board can be programmed with a PRO MATE II device programmer, PICSTART Plus development programmer, or MPLAB ICD 2 with a Universal Programmer Adapter. The MPLAB ICD 2 and MPLAB ICE in-circuit emulators may also be used with the PICDEM 2 demonstration board to test firmware. A prototype area extends the circuitry for additional application components. Some of the features include an RS-232 interface, a 2 x 16 LCD display, a piezo speaker, an on-board temperature sensor, four LEDs, and sample PIC18F452 and PIC16F877 FLASH microcontrollers.

16.17 PICDEM 3 PIC16C92X Demonstration Board

The PICDEM 3 demonstration board supports the PIC16C923 and PIC16C924 in the PLCC package. All the necessary hardware and software is included to run the demonstration programs.

16.18 PICDEM 17 Demonstration Board

The PICDEM 17 demonstration board is an evaluation board that demonstrates the capabilities of several Microchip microcontrollers, including PIC17C752, PIC17C756A, PIC17C762 and PIC17C766. A programmed sample is included. The PRO MATE II device programmer, or the PICSTART Plus development programmer, can be used to reprogram the device for user tailored application development. The PICDEM 17 demonstration board supports program download and execution from external on-board FLASH memory. A generous prototype area is available for user hardware expansion.

16.19 PICDEM 18R PIC18C601/801 Demonstration Board

The PICDEM 18R demonstration board serves to assist development of the PIC18C601/801 family of Microchip microcontrollers. It provides hardware implementation of both 8-bit Multiplexed/De-multiplexed and 16-bit Memory modes. The board includes 2 Mb external FLASH memory and 128 Kb SRAM memory, as well as serial EEPROM, allowing access to the wide range of memory types supported by the PIC18C601/801.

16.20 PICDEM LIN PIC16C43X Demonstration Board

The powerful LIN hardware and software kit includes a series of boards and three PICmicro microcontrollers. The small footprint PIC16C432 and PIC16C433 are used as slaves in the LIN communication and feature on-board LIN transceivers. A PIC16F874 FLASH microcontroller serves as the master. All three microcontrollers are programmed with firmware to provide LIN bus communication.

16.21 PICDEM USB PIC16C7X5 Demonstration Board

The PICDEM USB Demonstration Board shows off the capabilities of the PIC16C745 and PIC16C765 USB microcontrollers. This board provides the basis for future USB products.

16.22 Evaluation and Programming Tools

In addition to the PICDEM series of circuits, Microchip has a line of evaluation kits and demonstration software for these products.

- KEELOQ evaluation and programming tools for Microchip's HCS Secure Data Products
- CAN developers kit for automotive network applications
- Analog design boards and filter design software
- PowerSmart battery charging evaluation/ calibration kits
- IrDA® development kit
- microID development and RFLab™ development software
- SEEVAL® designer kit for memory evaluation and endurance calculations
- PICDEM MSC demo boards for Switching mode power supply, high power IR driver, delta sigma ADC, and flow rate sensor

Check the Microchip web page and the latest Product Line Card for the complete list of demonstration and evaluation kits.

TABLE 16-1: DEVELOPMENT TOOLS FROM MICROCHIP

		PIC12CXXX	PIC12FXXX	PIC14000	PIC16C5X	PIC16C6X	PIC16CXXX	PIC16C43X	PIC16F62X	PIC16C7X	PIC16C7XX	PIC16C7X5	PIC16C8X	PIC16F8XX	PIC16C9XX	PIC17C4X	PIC17C7XX	PIC18CXX2	PI18CX01	PIC18FXXX	dsPIC30F
Software Tools	MPLAB Integrated Development Environment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
	MPLAB C17 C Compiler															✓	✓				
	MPLAB C18 C Compiler																	✓		✓	
	MPASM Assembler/ MPLINK Object Linker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
	MPLAB C30 C Compiler																				✓
	MPLAB ASM30 Assembler/Linker/Librarian																				✓
Emulators	MPLAB ICE 2000 In-Circuit Emulator	✓	✓	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
	MPLAB ICE 4000 In-Circuit Emulator	✓			✓	✓	✓			✓	✓		✓		✓		✓	✓	✓	✓	✓
Debugger	MPLAB ICD 2 In-Circuit Debugger		✓			✓*				✓*				✓					✓	✓	✓
Programmers	PICSTART Plus Entry Level Development Programmer	✓	✓	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
	PRO MATE II Universal Device Programmer	✓	✓	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
Demo Boards and Eval Kits	PICDEM 1 Demonstration Board				✓		✓			✓†			✓			✓					
	PICDEM.net Demonstration Board																	✓			
	PICDEM 2 Plus Demonstration Board					✓†				✓†								✓		✓	
	PICDEM 3 Demonstration Board														✓						
	PICDEM 14A Demonstration Board			✓																	
	PICDEM 17 Demonstration Board																✓				
	PICDEM 18R Demonstration Board																		✓		
	PICDEM LIN Demonstration Board							✓						✓							
	PICDEM USB Demonstration Board											✓		✓							

* Contact the Microchip web site at www.microchip.com for information on how to use the MPLAB ICD In-Circuit Debugger (DV164001) with PIC16C62, 63, 64, 65, 72, 73, 74, 76, 77.
** Contact Microchip Technology Inc. for availability date.
† Development tool is available on select devices.

17.0 ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings†

Ambient temperature under bias ..... -40 to +125°C

Storage temperature ..... -65°C to +150°C

Voltage on V_{DD} with respect to V_{SS} ..... -0.3 to +6.5V

Voltage on $\overline{MCLR}$ and RA4 with respect to V_{SS} ..... -0.3 to +14V

Voltage on all other pins with respect to V_{SS} ..... -0.3V to V_{DD} + 0.3V

Total power dissipation(1) ..... 800 mW

Maximum current out of V_{SS} pin ..... 300 mA

Maximum current into V_{DD} pin ..... 250 mA

Input clamp current, I_{IK} (V_I < 0 or V_I > V_{DD}) ..... ± 20 mA

Output clamp current, I_{OK} (Vo < 0 or Vo >V_{DD}) ..... ± 20 mA

Maximum output current sunk by any I/O pin ..... 25 mA

Maximum output current sourced by any I/O pin ..... 25 mA

Maximum current sunk by PORTA and PORTB ..... 200 mA

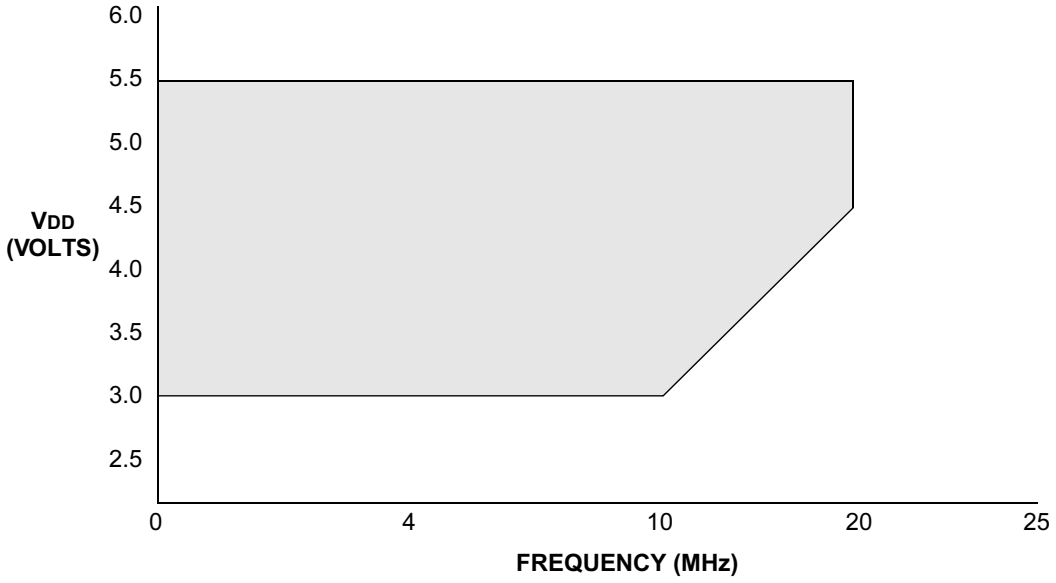
Maximum current sourced by PORTA and PORTB ..... 200 mA

Note 1: Power dissipation is calculated as follows: $P_{DIS} = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD}-V_{OH}) \times I_{OH}\} + \sum(V_{OI} \times I_{OL})$

† NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

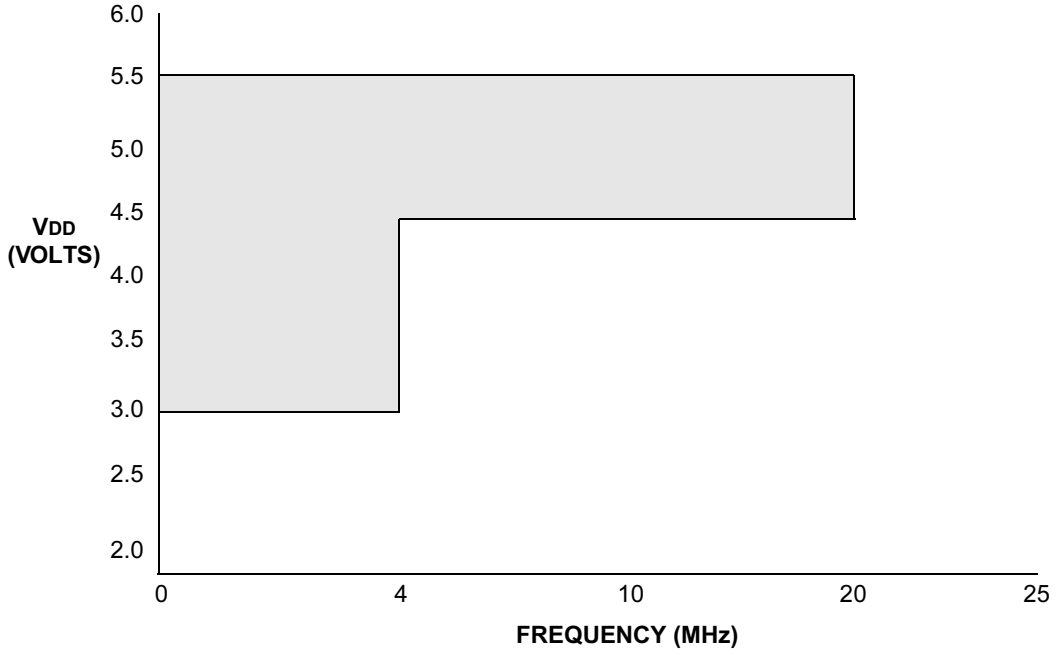
Note: Voltage spikes below V_{SS} at the $\overline{MCLR}$ pin, inducing currents greater than 80 mA, may cause latchup. Thus, a series resistor of 50-100 Ω should be used when applying a “low” level to the $\overline{MCLR}$ pin rather than pulling this pin directly to V_{SS}

FIGURE 17-1: PIC16F62X VOLTAGE-FREQUENCY GRAPH, 0°C ≤ TA ≤ +70°C



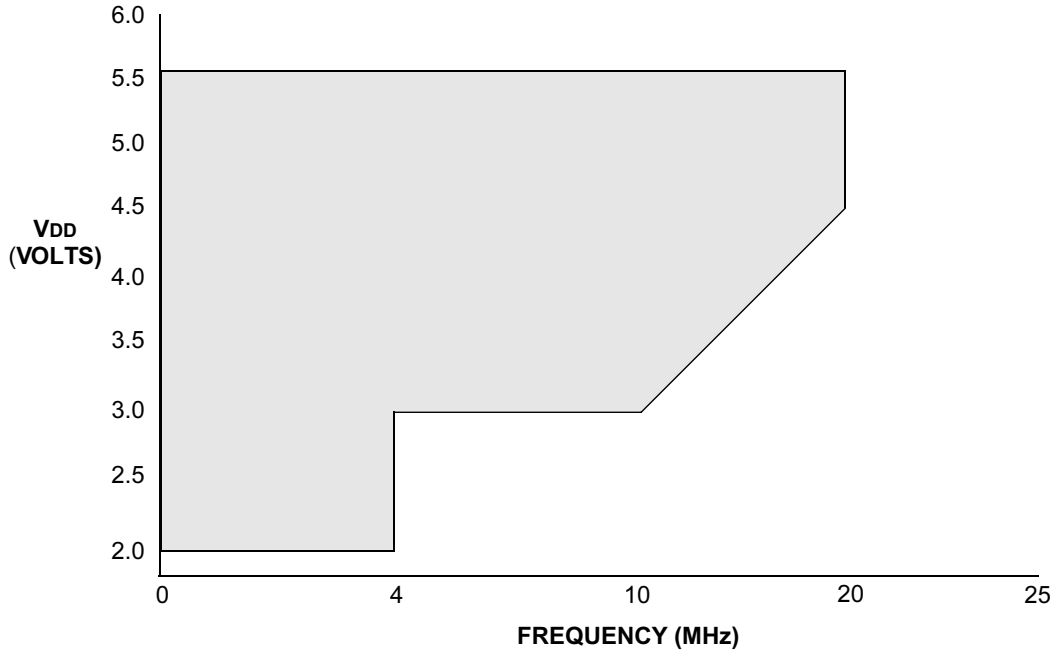
Note 1: The shaded region indicates the permissible combinations of voltage and frequency.

FIGURE 17-2: PIC16F62X VOLTAGE-FREQUENCY GRAPH, -40°C ≤ TA < 0°C, +70°C < TA ≤ 85°C



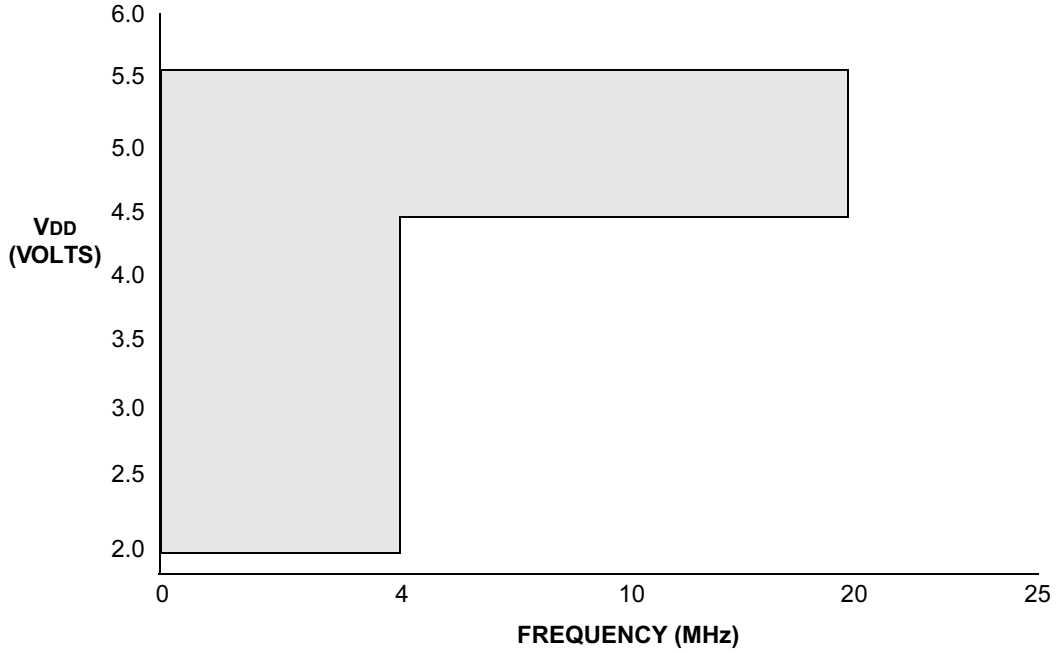
Note 1: The shaded region indicates the permissible combinations of voltage and frequency.

FIGURE 17-3: PIC16LF62X VOLTAGE-FREQUENCY GRAPH, 0°C ≤ TA ≤ +70°C



Note 1: The shaded region indicates the permissible combinations of voltage and frequency.

FIGURE 17-4: PIC16LF62X VOLTAGE-FREQUENCY GRAPH, -40°C ≤ TA < 0°C, +70°C < TA ≤ 85°C



Note 1: The shaded region indicates the permissible combinations of voltage and frequency.

17.1 DC Characteristics: PIC16F62X-04 (Commercial, Industrial, Extended) PIC16F62X-20 (Commercial, Industrial, Extended) PIC16LF62X-04 (Commercial, Industrial)

PIC16LF62X-04 (Commercial, Industrial)	**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ Ta ≤ +85°C for industrial and 0°C ≤ Ta ≤ +70°C for commercial
PIC16F62X-04 **PIC16F62X-20** (Commercial, Industrial, Extended)	**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ Ta ≤ +85°C for industrial and 0°C ≤ Ta ≤ +70°C for commercial and -40°C ≤ Ta ≤ +125°C for extended

Param No.	Sym	Characteristic/Device	Min	Typ†	Max	Units	Conditions
	VDD	**Supply Voltage**					
D001		PIC16LF62X	2.0	—	5.5	V	
D001		PIC16F62X	3.0	—	5.5	V	
D002	VDR	**RAM Data Retention Voltage(1)**	—	1.5	—	V	Device in SLEEP mode*
D003	VPOR	**VDD Start Voltage** to ensure Power-on Reset	—	VSS	—	V	See section on Power-on Reset for details
D004	SVDD	**VDD Rise Rate** to ensure Power-on Reset	0.05	—	—	V/ms	See section on Power-on Reset for details*
D005	VBOD	**Brown-out Detect Voltage**	3.65 3.65	4.0 —	4.35 4.4	V V	BODEN configuration bit is set BODEN configuration bit is set, Extended
	IDD	**Supply Current(2), (5)**					
D010		PIC16LF62X	—	0.30	0.6	mA	Fosc = 4.0 MHz, VDD = 2.0(5)
			—	1.10	2.0	mA	FOSC = 4.0 MHz, VDD = 5.5*
D013			—	4.0	7.0	mA	Fosc = 20.0 MHz, VDD = 5.5
			—	3.80	6.0	mA	Fosc = 20.0 MHz, VDD = 4.5*
			—	—	2.0	mA	FOSC = 10.0 MHz, VDD = 3.0(6)
			—	20	30	μA	FOSC = 32 kHz, VDD = 2.0
D010		PIC16F62X	—	0.60	0.7	mA	Fosc = 4.0 MHz, VDD = 3.0
			—	1.10	2.0	mA	Fosc = 4.0 MHz, VDD = 5.5*
D013			—	4.0	7.0	mA	FOSC = 20.0 MHz, VDD = 5.5
			—	3.80	6.0	mA	FOSC = 20.0 MHz, VDD = 4.5*
			—	—	2.0	mA	FOSC = 10.0 MHz, VDD = 3.0*(6)
D014				20	30	μA	FOSC = 32 kHz, VDD = 3.0*

Legend: Rows with standard voltage device data only are shaded for improved readability.

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C, unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: This is the limit to which VDD can be lowered in SLEEP mode without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors such as I/O pin loading and switching rate, oscillator type, internal code execution pattern, and temperature also have an impact on the current consumption.

The test conditions for all IDD measurements in active Operation mode are:
OSC1 = external square wave, from rail to rail; all I/O pins tri-stated, pulled to VDD,
$\overline{\text{MCLR}}$ = VDD; WDT enabled/disabled as specified.

3: The power-down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins in hi-impedance state and tied to VDD or VSS.

4: The Δ current is the additional current consumed when this peripheral is enabled. This current should be added to the base IDD or IPD measurement.

5: For RC osc configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2REXT (mA) with REXT in kΩ.

6: Commercial temperature only.

17.1 DC Characteristics: PIC16F62X-04 (Commercial, Industrial, Extended) PIC16F62X-20 (Commercial, Industrial, Extended) PIC16LF62X-04 (Commercial, Industrial)

PIC16LF62X-04 (Commercial, Industrial)	**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ Ta ≤ +85°C for industrial and 0°C ≤ Ta ≤ +70°C for commercial
PIC16F62X-04 **PIC16F62X-20** (Commercial, Industrial, Extended)	**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ Ta ≤ +85°C for industrial and 0°C ≤ Ta ≤ +70°C for commercial and -40°C ≤ Ta ≤ +125°C for extended

Param No.	Sym	Characteristic/Device	Min	Typ†	Max	Units	Conditions
D020	IPD	**Power Down Current*(2), (3)**					
		PIC16LF62X	—	0.20	2.0	μA	VDD = 2.0
			—	0.20	2.2	μA	VDD = 5.5
D020		PIC16F62X	—	0.20	2.2	μA	VDD = 3.0
			—	0.20	5.0	μA	VDD = 4.5*
			—	0.20	9.0	μA	VDD = 5.5
			—	2.70	15.0	μA	VDD = 5.5 Extended
D023	ΔIWDT	WDT Current(4)	—	6.0	15	μA	VDD = 3.0V
	ΔIBOD	Brown-out Detect Current(4)	—	75	125	μA	BOD enabled, VDD = 5.0V
	ΔICOMP	Comparator Current for each Comparator(4)	—	30	50	μA	VDD = 3.0V
	ΔIVREF	VREF Current(4)	—		135	μA	VDD = 3.0V
D023	ΔIWDT	WDT Current(4)	—	6.0	20	μA	VDD = 4.0V, Commercial, Industrial
					25	μA	VDD = 4.0V, Extended
	ΔIBOD	Brown-out Detect Current(4)	—	75	125	μA	BOD enabled, VDD = 5.0V
	ΔICOMP	Comparator Current for each Comparator(4)	—	30	50	μA	VDD = 4.0V
	ΔIVREF	VREF Current(4)	—		135	μA	VDD = 4.0V

Legend: Rows with standard voltage device data only are shaded for improved readability.

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C, unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: This is the limit to which VDD can be lowered in SLEEP mode without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors such as I/O pin loading and switching rate, oscillator type, internal code execution pattern, and temperature also have an impact on the current consumption.

The test conditions for all IDD measurements in active Operation mode are:
OSC1 = external square wave, from rail to rail; all I/O pins tri-stated, pulled to VDD,
MCLR = VDD; WDT enabled/disabled as specified.

3: The power-down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins in hi-impedance state and tied to VDD or VSS.

4: The Δ current is the additional current consumed when this peripheral is enabled. This current should be added to the base IDD or IPD measurement.

5: For RC osc configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2REXT (mA) with REXT in kΩ.

17.2 DC Characteristics: PIC16F62X (Commercial, Industrial, Extended) PIC16LF62X (Commercial, Industrial)

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and 0°C ≤ TA ≤ +70°C for commercial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 17-1 and Table 17-2				
Param. No.	**Sym**	**Characteristic/Device**	**Min**	**Typ†**	**Max**	**Unit**	**Conditions**
	VIL	**Input Low Voltage**					
		I/O ports					
D030		with TTL buffer	VSS	—	0.8	V	VDD = 4.5V to 5.5V
					0.15 VDD	V	otherwise
D031		with Schmitt Trigger input	VSS		0.2 VDD	V	
D032		$\overline{\text{MCLR}}$, RA4/T0CKI,OSC1 (in ER mode)	VSS	—	0.2 VDD	V	**(Note1)**
D033		OSC1 (in XT and HS)	VSS	—	0.3 VDD	V	
		OSC1 (in LP)	VSS	—	0.6 VDD - 1.0	V	
	VIH	**Input High Voltage**					
		I/O ports					
D040		with TTL buffer	2.0V	—	VDD	V	VDD = 4.5V to 5.5V
			.25 VDD + 0.8V		VDD	V	otherwise
D041		with Schmitt Trigger input	0.8 VDD	—	VDD	V	
D042		$\overline{\text{MCLR}}$ RA4/T0CKI	0.8 VDD	—	VDD	V	
D043		OSC1 (XT, HS and LP)	0.7 VDD	—	VDD	V	
D043A		OSC1 (in ER mode)	0.9 VDD			V	**(Note1)**
D070	IPURB	**PORTB weak pull-up current**	50	200	400	μA	VDD = 5.0V, VPIN = VSS
	IIL	**Input Leakage Current(2), (3)**					
		I/O ports (Except PORTA)			±1.0	μA	VSS ≤ VPIN ≤ VDD, pin at hi-impedance
D060		PORTA	—	—	±0.5	μA	VSS ≤ VPIN ≤ VDD, pin at hi-impedance
D061		RA4/T0CKI	—	—	±1.0	μA	VSS ≤ VPIN ≤ VDD
D063		OSC1, $\overline{\text{MCLR}}$	—	—	±5.0	μA	VSS ≤ VPIN ≤ VDD, XT, HS and LP osc configuration
	VOL	**Output Low Voltage**					
D080		I/O ports	—	—	0.6	V	IOL=8.5 mA, VDD=4.5V, -40° to +85°C
			—	—	0.6	V	IOL=7.0 mA, VDD=4.5V, +125°C
D083		OSC2/CLKOUT (ER only)	—	—	0.6	V	IOL=1.6 mA, VDD=4.5V, -40° to +85°C
			—	—	0.6	V	IOL=1.2 mA, VDD=4.5V, +125°C
	VOH	**Output High Voltage(3)**					
D090		I/O ports (Except RA4)	VDD - 0.7	—	—	V	IOH=-3.0 mA, VDD=4.5V, -40° to +85°C
			VDD - 0.7	—	—	V	IOH=-2.5 mA, VDD=4.5V, +125°C
D092		OSC2/CLKOUT (ER only)	VDD - 0.7	—	—	V	IOH=-1.3 mA, VDD=4.5V, -40° to +85°C
			VDD - 0.7	—	—	V	IOH=-1.0 mA, VDD=4.5V, +125°C
D150	VOD	**Open-Drain High Voltage**		—	8.5	V	RA4 pin PIC16F62X, PIC16LF62X*
		Capacitive Loading Specs on Output Pins					
D100*	COSC2	OSC2 pin		—	15	pF	In XT, HS and LP modes when external clock used to drive OSC1.
D101*	Cio	All I/O pins/OSC2 (in ER mode)		—	50	pF	

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: In ER oscillator configuration, the OSC1 pin is a Schmitt Trigger input. It is not recommended that the PIC16F62X be driven with external clock in ER mode.

2: The leakage current on the $\overline{\text{MCLR}}$ pin is strongly dependent on applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as coming out of the pin.

TABLE 17-1: COMPARATOR SPECIFICATIONS

	Operating Conditions: 3.0V < VDD <5.5V, -40°C < TA < +125°C, unless otherwise stated.						
Param No.	**Characteristics**	**Sym**	**Min**	**Typ**	**Max**	**Units**	**Comments**
D300	Input offset voltage	VIOFF	—	±5.0	±10	mV	
D301*	Input Common mode voltage	VICM	0	—	VDD - 1.5	V	
D302*	Common Mode Rejection Ratio	CMRR	55	—	—	db	
300* 300A	Response Time(1)	TRESP	—	150	400 600	ns ns	16F62X 16LF62X
301	Comparator Mode Change to Output Valid*	TMC2OV	—	—	10	µs	

* These parameters are characterized but not tested.

Note 1: Response time measured with one comparator input at (VDD - 1.5)/2 while the other input transitions from VSS to VDD.

TABLE 17-2: VOLTAGE REFERENCE SPECIFICATIONS

	Operating Conditions: 3.0V < VDD < 5.5V, -40°C < TA < +125°C, unless otherwise stated.						
Spec No.	**Characteristics**	**Sym**	**Min**	**Typ**	**Max**	**Units**	**Comments**
D310	Resolution	VRES	VDD/24	—	VDD/32	LSb	
D311	Absolute Accuracy	VRaa	— —	— —	1/4 1/2	LSb LSb	Low Range (VRR = 1) High Range (VRR = 0)
D312*	Unit Resistor Value (R)	VRur	—	2k	—	Ω	
310*	Settling Time(1)	Tset	—	—	10	µs	

* These parameters are characterized but not tested.

Note 1: Settling time measured while VRR = 1 and VR<3:0> transitions from `0000` to `1111`.

17.3 Timing Parameter Symbology

The timing parameter symbols have been created with one of the following formats:

1. TppS2ppS
2. TppS

T			
F	Frequency	T	Time

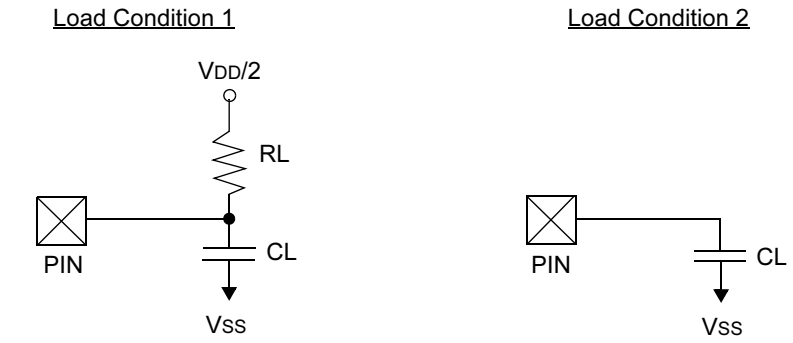
Lowercase subscripts (pp) and their meanings:

pp			
ck	CLKOUT	osc	OSC1
io	I/O port	t0	T0CKI
mc	$\overline{\text{MCLR}}$		

Uppercase letters and their meanings:

S			
F	Fall	P	Period
H	High	R	Rise
I	Invalid (Hi-impedance)	V	Valid
L	Low	Z	Hi-Impedance

FIGURE 17-5: LOAD CONDITIONS



R_L = 464Ω

C_L = 50 pF for all pins except OSC2

15 pF for OSC2 output

TABLE 17-3: DC CHARACTERISTICS: PIC16F62X, PIC16LF62X

DC Characteristics			Standard Operating Conditions (unless otherwise stated)				
Parameter No.	**Sym**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
		Data EEPROM Memory					
D120	ED	Endurance	1M*	10M	—	E/W	25°C at 5V
D121	VDRW	VDD for read/write	VMIN	—	5.5	V	VMIN = Minimum operating voltage
D122	TDEW	Erase/Write cycle time	—	4	8*	ms	
		Program FLASH Memory					
D130	EP	Endurance	1000*	10000	—	E/W	
D131	VPR	VDD for read	Vmin	—	5.5	V	VMIN = Minimum operating voltage
D132	VPEW	VDD for erase/write	4.5	—	5.5	V	
D133	TPEW	Erase/Write cycle time	—	4	8*	ms	

* These parameters are characterized but not tested.

† Data in “Typ” column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

17.4 Timing Diagrams and Specifications

FIGURE 17-6: EXTERNAL CLOCK TIMING

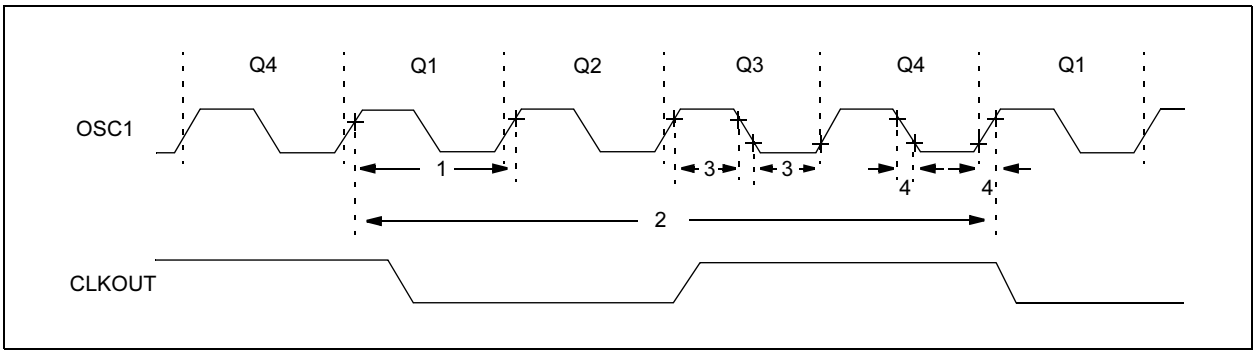


TABLE 17-4: EXTERNAL CLOCK TIMING REQUIREMENTS

Param No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
	Fosc	External CLKIN Frequency(1)	DC	—	4	MHz	XT and ER Osc mode, VDD = 5.0V
			DC	—	20	MHz	HS Osc mode
			DC	—	200	kHz	LP Osc mode
		Oscillator Frequency(1)		—	4	MHz	ER Osc mode, VDD = 5.0V
			0.1	—	4	MHz	XT Osc mode
			1	—	20	MHz	HS Osc mode
				—	200	kHz	LP Osc mode
			3.65	4	4.28	MHz	INTRC mode (fast), VDD = 5.0V
				37		kHz	INTRC mode (slow)
4	INTRC	Internal Calibrated RC	3.65	4.00	4.28	MHz	VDD = 5.0V
5	ER	External Biased ER Frequency	10 kHz		8 MHz		VDD = 5.0V
1	Tosc	External CLKIN Period(1)	250	—	—	ns	XT and ER Osc mode
			50	—	—	ns	HS Osc mode
			5	—	—	μs	LP Osc mode
		Oscillator Period(1)	250	—	—	ns	ER Osc mode
			250	—	10,000	ns	XT Osc mode
			50	—	1,000	ns	HS Osc mode
			5			μs	LP Osc mode
				250		ns	INTRC mode (fast)
				27		μs	INTRC mode (slow)
2	Tcy	Instruction Cycle Time	1.0	TCY	DC	ns	TCY = 4/FOSC
3	TosL, TosH	External CLKIN (OSC1) High External CLKIN Low	100 *	—	—	ns	XT oscillator, TOSC L/H duty cycle*

* These parameters are characterized but not tested.

† Data in “Typ” column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: Instruction cycle period (Tcy) equals four times the input oscillator time-based period. All specified values are based on characterization data for that particular oscillator type under standard operating conditions with the device executing code. Exceeding these specified limits may result in an unstable oscillator operation and/or higher than expected current consumption. All devices are tested to operate at “Min.” values with an external clock applied to the OSC1 pin. When an external clock input is used, the “Max” cycle time limit is “DC” (no clock) for all devices.

FIGURE 17-7: CLKOUT AND I/O TIMING

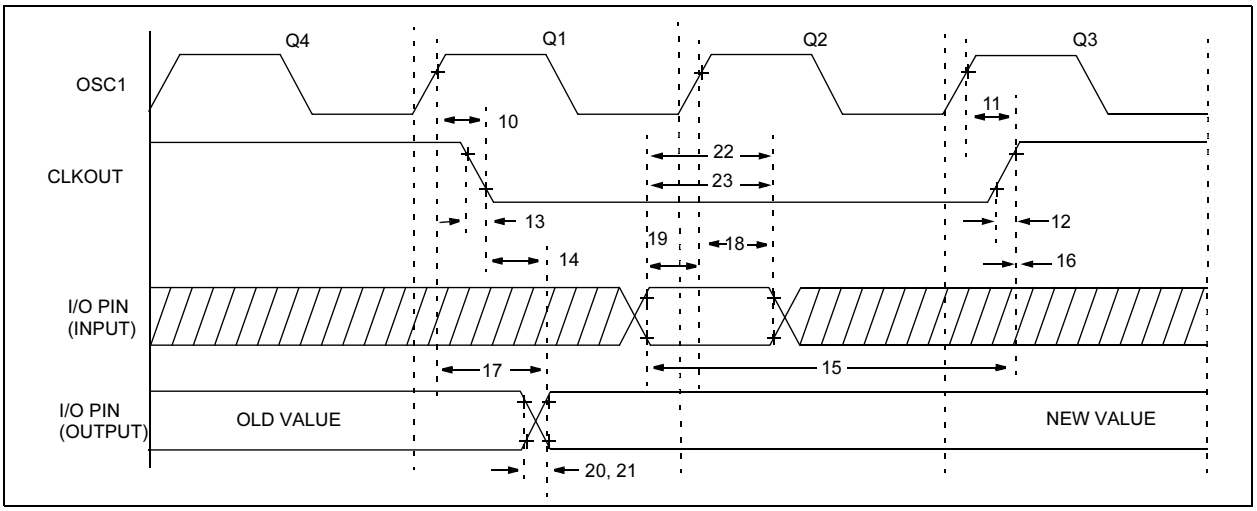


TABLE 17-5: CLKOUT AND I/O TIMING REQUIREMENTS

Param No.	Sym	Characteristic		Min	Typ†	Max	Units
10*	TosH2ckL	OSC1↑ to CLKOUT↓	16F62X	—	75	200	ns
10A*			16LF62X	—	—	400	ns
11*	TosH2ckH	OSC1↑ to CLKOUT↑	16F62X	—	75	200	ns
11A*			16LF62X	—	—	400	ns
12*	TckR	CLKOUT rise time	16F62X	—	35	100	ns
12A*			16LF62X	—	—	200	ns
13*	TckF	CLKOUT fall time	16F62X	—	35	100	ns
13A*			16LF62X	—	—	200	ns
14*	TckL2ioV	CLKOUT ↓ to Port out valid		—	—	20	ns
15*	TioV2ckH	Port in valid before	16F62X	Tosc+200 ns	—	—	ns
		CLKOUT ↑	16LF62X	Tosc=400 ns	—	—	ns
16*	TckH2ioI	Port in hold after CLKOUT ↑		0	—	—	ns
17*	TosH2ioV	OSC1↑ (Q1 cycle) to	16F62X	—	50	150*	ns
		Port out valid	16LF62X	—	—	300	ns
18*	TosH2ioI	OSC1↑ (Q2 cycle) to Port input invalid (I/O in hold time)		100 200	—	—	ns

* These parameters are characterized but not tested.

† Data in “Typ” column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

FIGURE 17-8: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER TIMING

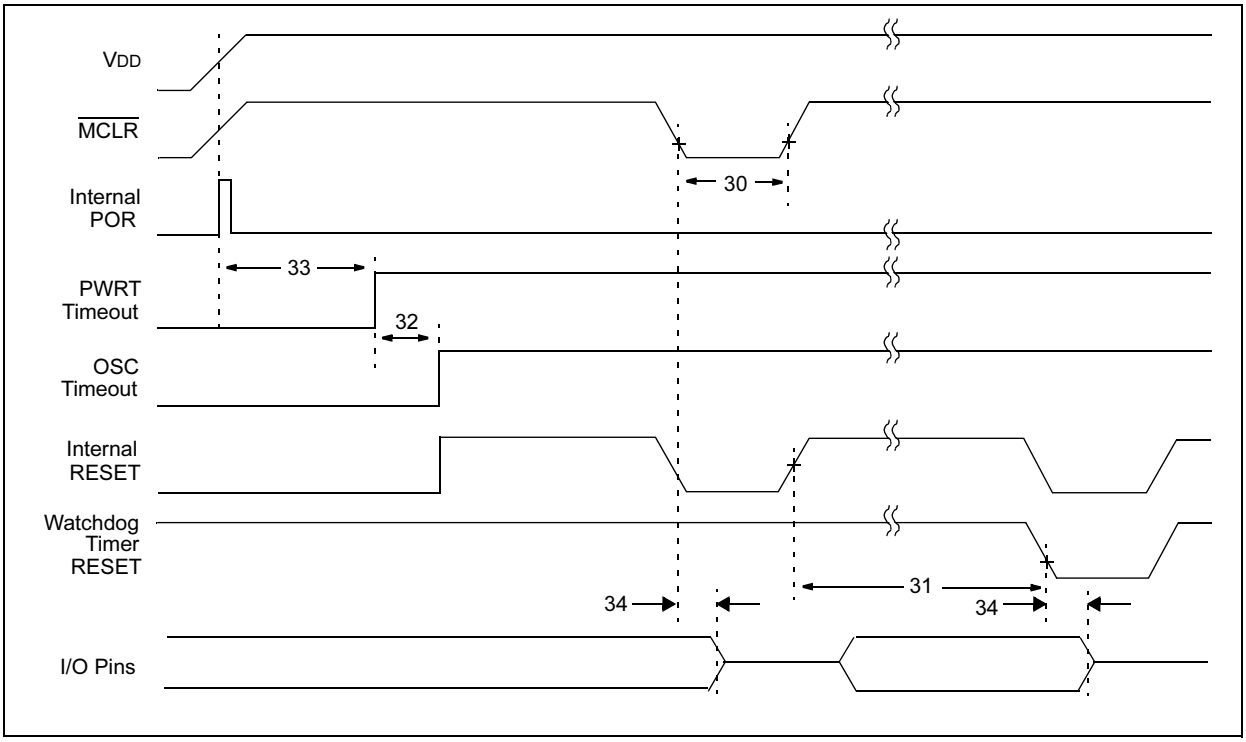


FIGURE 17-9: BROWN-OUT DETECT TIMING

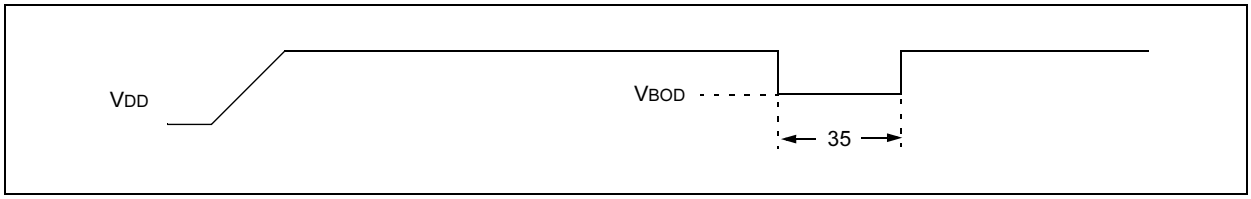


TABLE 17-6: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER REQUIREMENTS

Param No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
30	TmcL	$\overline{\text{MCLR}}$ Pulse Width (low)	2000 TBD	— TBD	— TBD	ns ms	VDD = 5V, -40°C to +85°C Extended temperature
31	Twdt	Watchdog Timer Timeout Period (No Prescaler)	7 TBD	18 TBD	33 TBD	ms ms	VDD = 5V, -40°C to +85°C Extended temperature
32	Tost	Oscillation Start-up Timer Period	—	1024TOSC	—	—	TOSC = OSC1 period
33*	Tpwrt	Power-up Timer Period	28 TBD	72 TBD	132 TBD	ms ms	VDD = 5V, -40°C to +85°C Extended temperature
34	TIOZ	I/O Hi-impedance from $\overline{\text{MCLR}}$ Low or Watchdog Timer Reset	—	—	2.0	μs	
35	TBOD	Brown-out Detect pulse width	100	—	—	μs	VDD ≤ VBOD (D005)

* These parameters are characterized but not tested.

† Data in “Typ” column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

FIGURE 17-10: TIMER0 AND TIMER1 EXTERNAL CLOCK TIMINGS

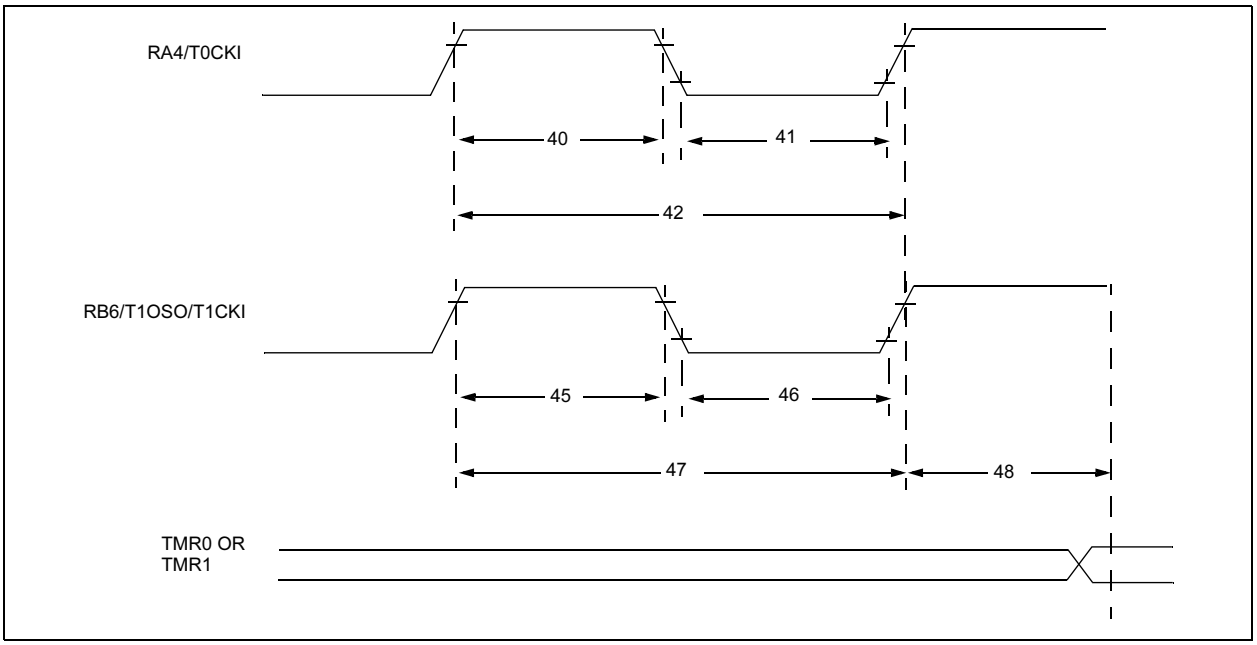


TABLE 17-7: TIMER0 AND TIMER1 EXTERNAL CLOCK REQUIREMENTS

Param No.	Sym	Characteristic			Min	Typ†	Max	Units	Conditions
40*	Tt0H	T0CKI High Pulse Width		No Prescaler	0.5TCY + 20	—	—	ns	
				With Prescaler	10	—	—	ns	
41*	Tt0L	T0CKI Low Pulse Width		No Prescaler	0.5TCY + 20	—	—	ns	
				With Prescaler	10	—	—	ns	
42*	Tt0P	T0CKI Period			Greater of: TCY + 40 / N	—	—	ns	N = prescale value (2, 4, ..., 256)
45*	Tt1H	T1CKI High Time	Synchronous, No Prescaler		0.5TCY + 20	—	—	ns	
			Synchronous, with Prescaler	16F62X	15	—	—	ns	
				16LF62X	25	—	—	ns	
			Asynchronous	16F62X	30	—	—	ns	
				16LF62X	50	—	—	ns	
46*	Tt1L	T1CKI Low Time	Synchronous, No Prescaler		0.5TCY + 20	—	—	ns	
			Synchronous, with Prescaler	16F62X	15	—	—	ns	
				16LF62X	25	—	—	ns	
			Asynchronous	16F62X	30	—	—	ns	
				16LF62X	50	—	—	ns	
47*	Tt1P	T1CKI input period	Synchronous	16F62X	Greater of: TCY + 40 / N	—	—	ns	N = prescale value (1, 2, 4, 8)
				16LF62X	Greater of: TCY + 40 / N	—	—	—	
			Asynchronous	16F62X	60	—	—	ns	
				16LF62X	100	—	—	ns	
	Ft1	Timer1 oscillator input frequency range (oscillator enabled by setting bit T1OSCEN)			DC	—	200	kHz	
48	TCKEZtmr1	Delay from external clock edge to timer increment			2Tosc	—	7Tosc	—	

* These parameters are characterized but not tested.

† Data in “Typ” column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

FIGURE 17-11: CAPTURE/COMPARE/PWM TIMINGS

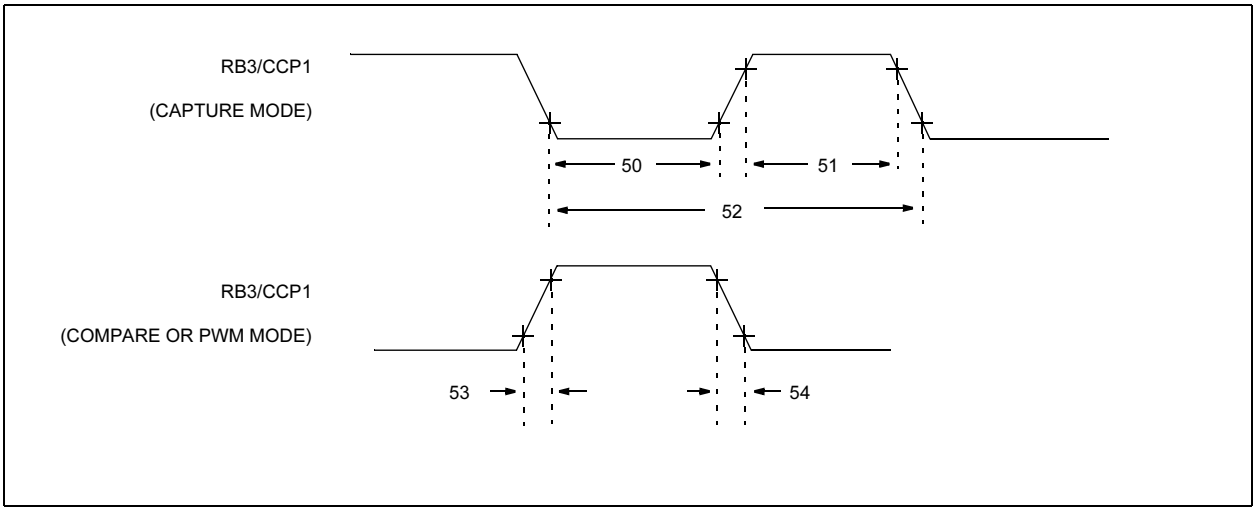


TABLE 17-8: CAPTURE/COMPARE/PWM REQUIREMENTS

Param No.	Sym	Characteristic			Min	Typ†	Max	Units	Conditions
50*	TccL	CCP input low time	No Prescaler		0.5TCY + 20	—	—	ns	
			With Prescaler	16F62X	10	—	—	ns	
				16LF62X	20	—	—	ns	
51*	TccH	CCP input high time	No Prescaler		0.5TCY + 20	—	—	ns	
			With Prescaler	16F62X	10	—	—	ns	
				16LF62X	20	—	—	ns	
52*	TccP	CCP input period			$\frac{3T_{CY} + 40}{N}$	—	—	ns	N = prescale value (1,4 or 16)
53*	TccR	CCP output rise time		16F62X		10	25	ns	
				16LF62X		25	45	ns	
54*	TccF	CCP output fall time		16F62X		10	25	ns	
				16LF62X		25	45	ns	

* These parameters are characterized but not tested.

† Data in “Typ” column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

FIGURE 17-12: TIMER0 CLOCK TIMING

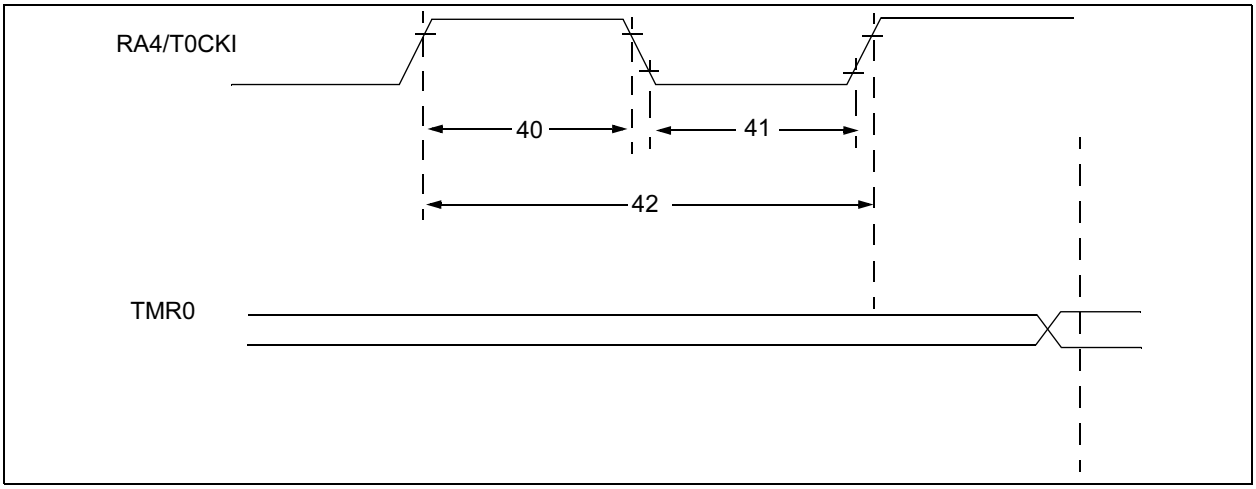


TABLE 17-9: TIMER0 CLOCK REQUIREMENTS

Param No.	Sym	Characteristic		Min	Typ†	Max	Units	Conditions
40	Tt0H	T0CKI High Pulse Width	No Prescaler	0.5 TCY + 20*	—	—	ns	
			With Prescaler	10*	—	—	ns	
41	Tt0L	T0CKI Low Pulse Width	No Prescaler	0.5 TCY + 20*	—	—	ns	
			With Prescaler	10*	—	—	ns	
42	Tt0P	T0CKI Period		$\frac{T_{CY} + 40}{N}$*	—	—	ns	N = prescale value (1, 2, 4, ..., 256)

* These parameters are characterized but not tested.

† Data in “Typ” column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

NOTES:

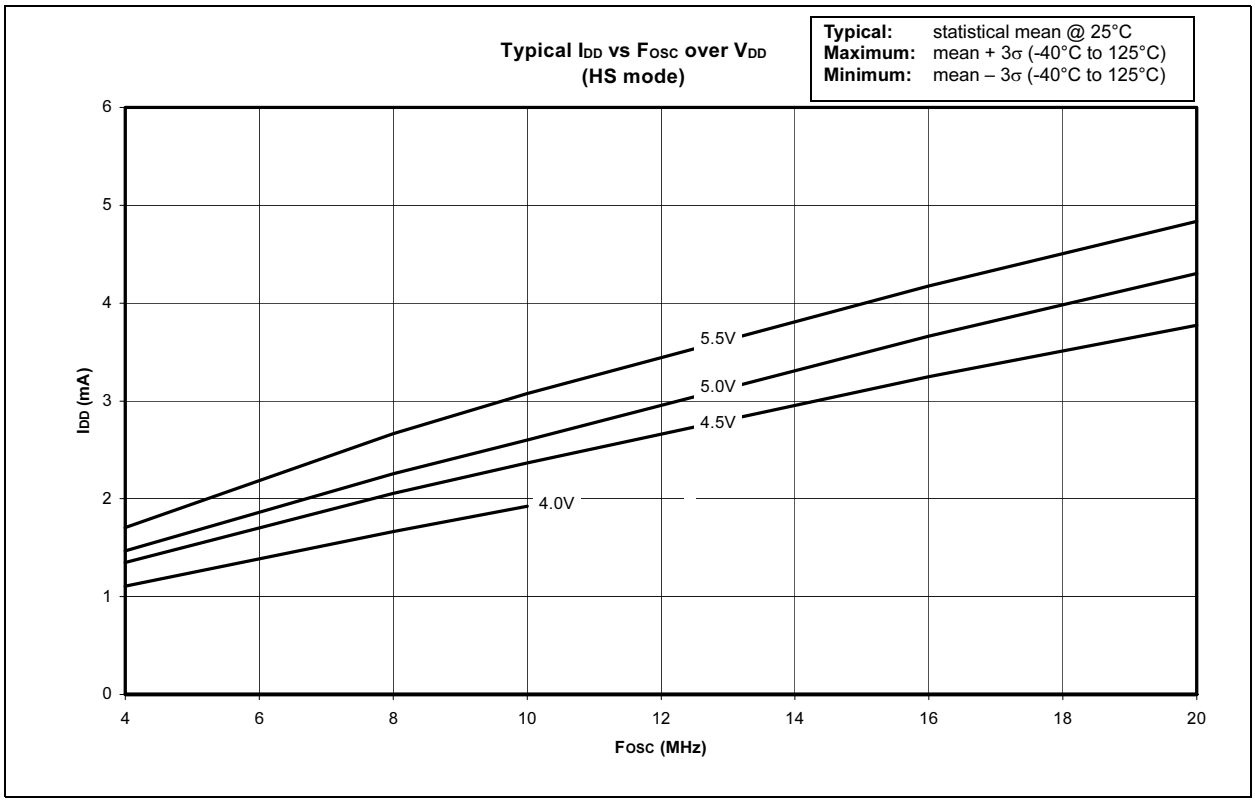
18.0 DC AND AC CHARACTERISTICS GRAPHS AND TABLES

In some graphs or tables, the data presented is outside specified operating range (i.e., outside specified VDD range). This is for information only and devices are ensured to operate properly only within the specified range.

The data presented in this section is a statistical summary of data collected on units from different lots over a period of time and matrix samples. 'Typical' represents the mean of the distribution at 25°C. 'max or min.' represents (mean + 3σ) or (mean - 3σ) respectively, where σ is standard deviation, over the whole temperature range.

Note:	The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-1: TYPICAL IDD VS FOSC OVER VDD – HS MODE



.

Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-2: MAXIMUM IDD VS FOSC OVER VDD (HS MODE)

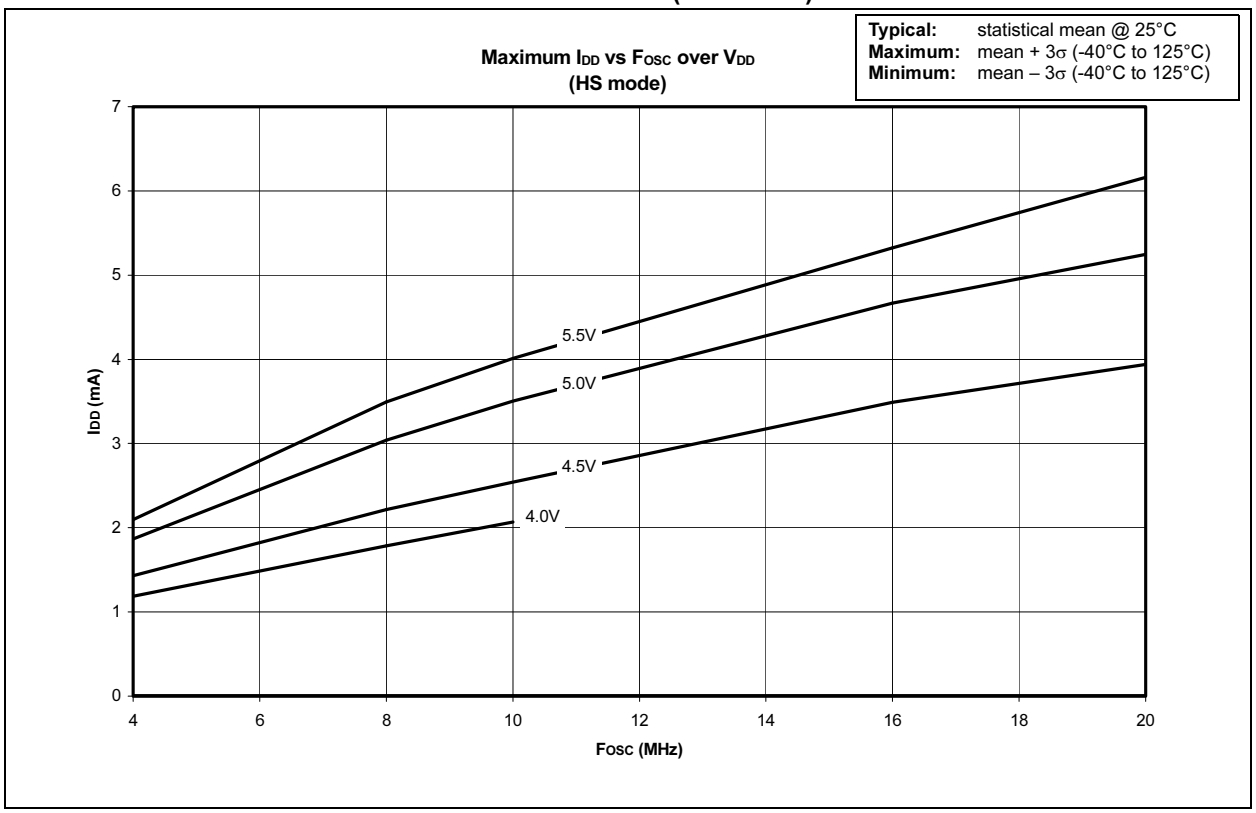
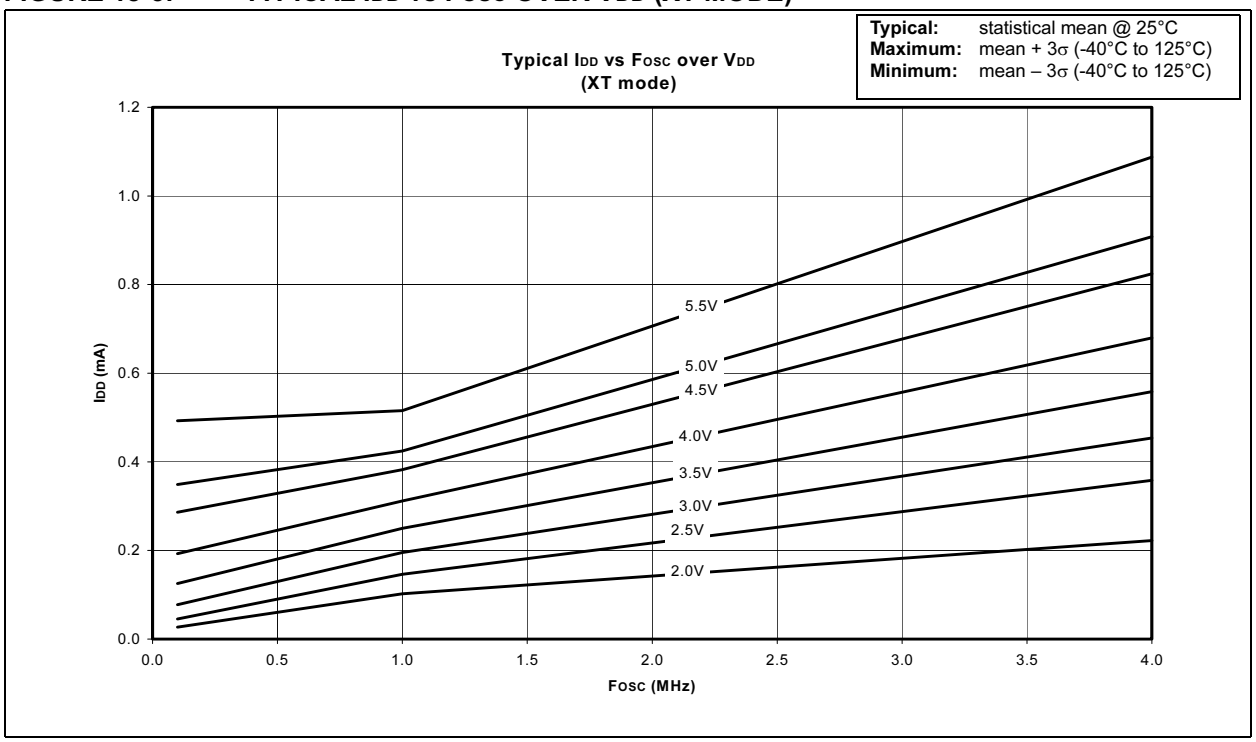


FIGURE 18-3: TYPICAL IDD VS FOSC OVER VDD (XT MODE)



.

Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-4: TYPICAL IDD VS FOSC OVER VDD (XT MODE)

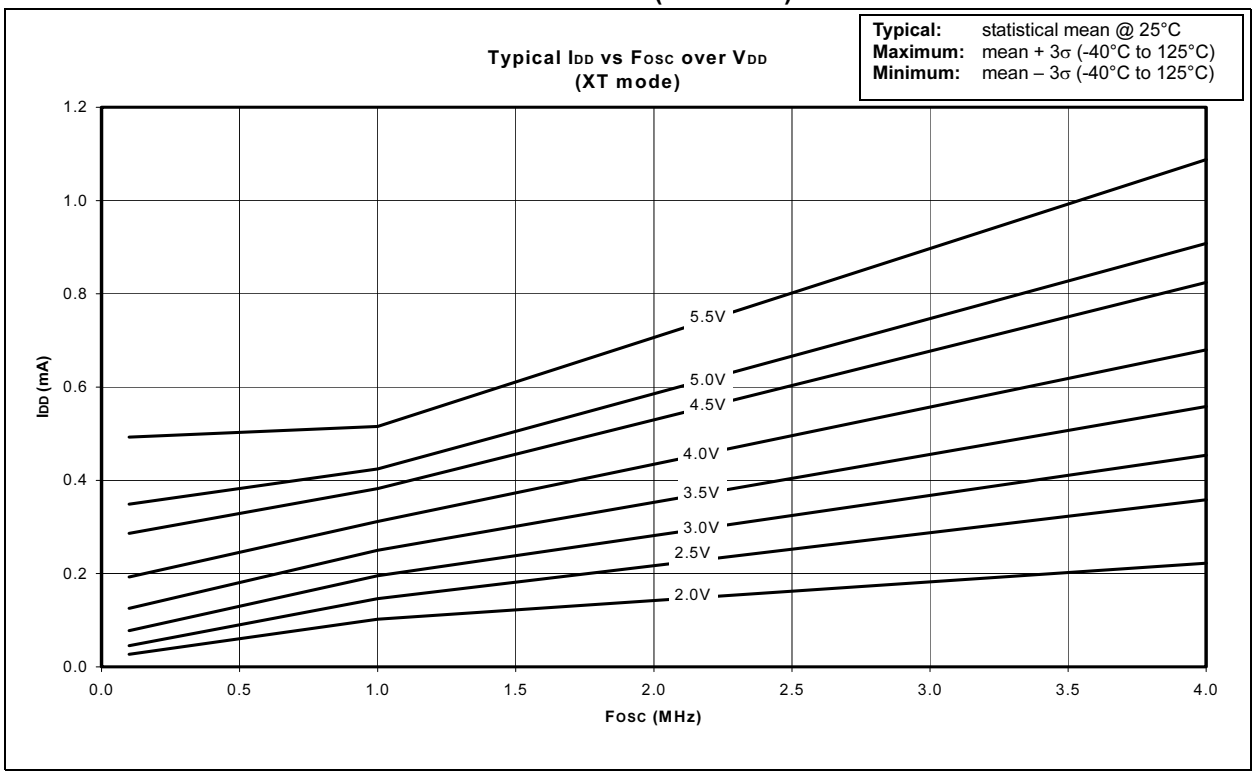
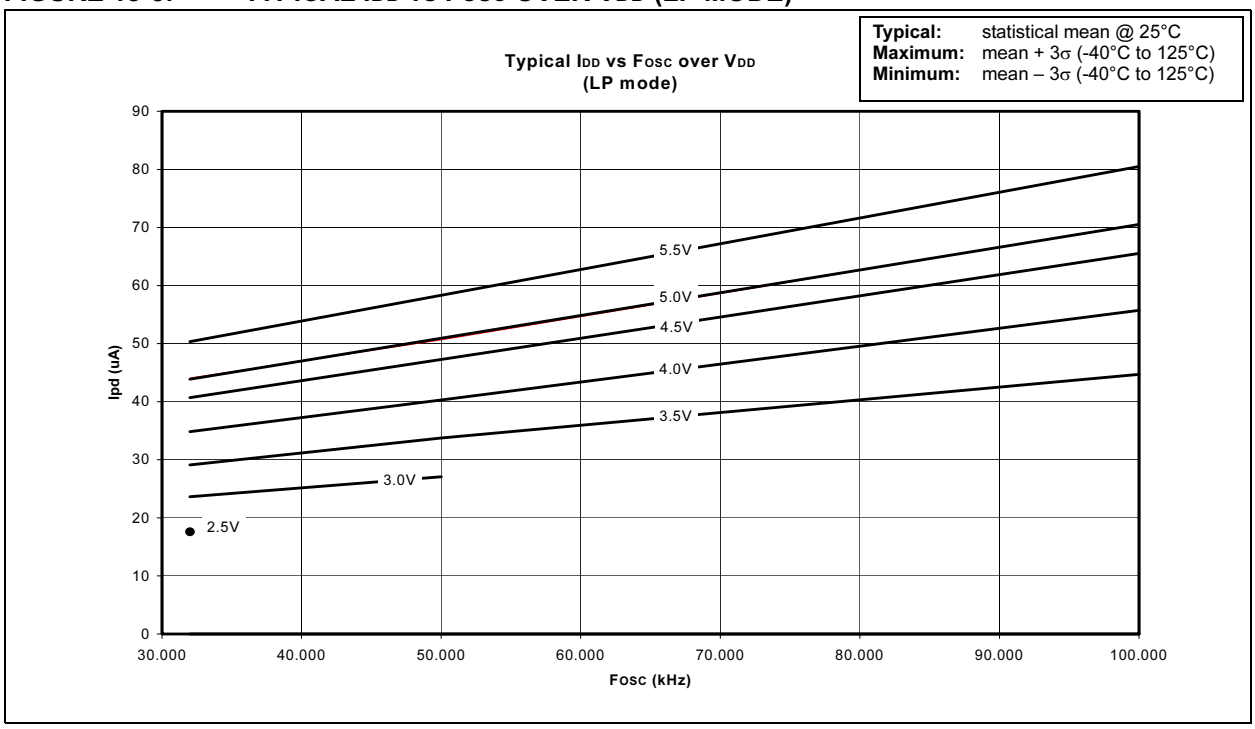


FIGURE 18-5: TYPICAL IDD VS FOSC OVER VDD (LP MODE)



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-6: MAXIMUM IDD VS FOSC OVER VDD (LP MODE)

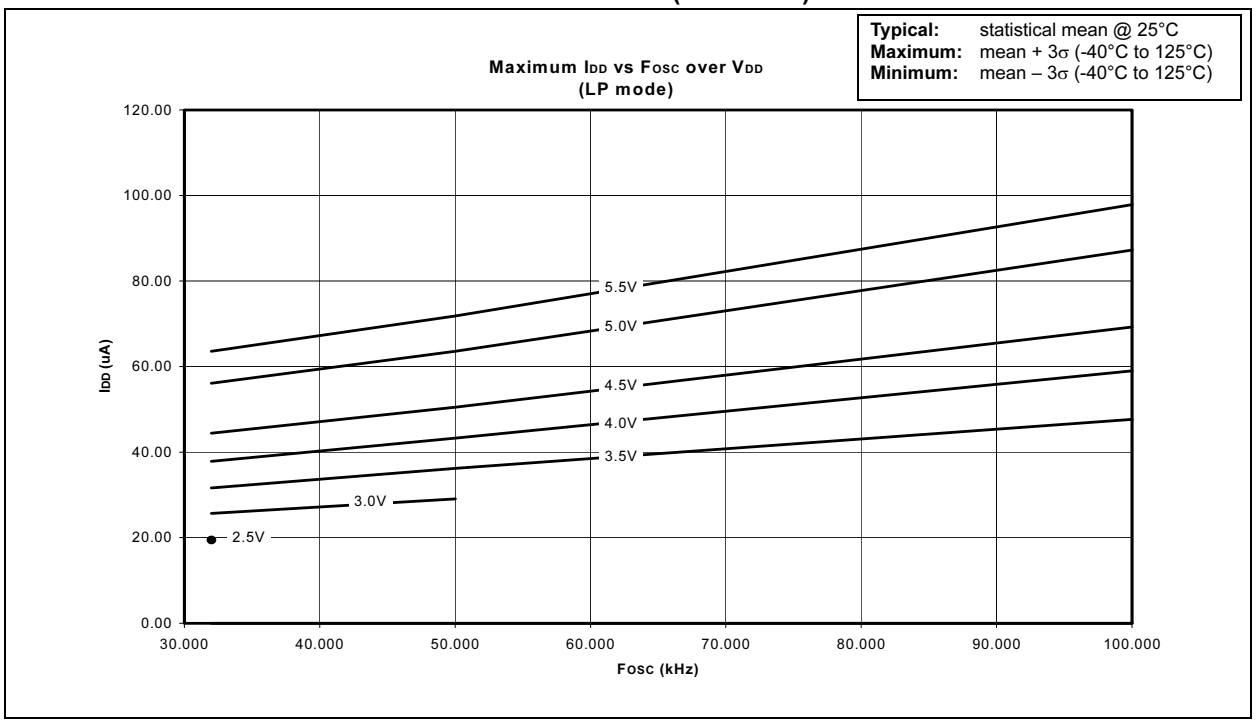
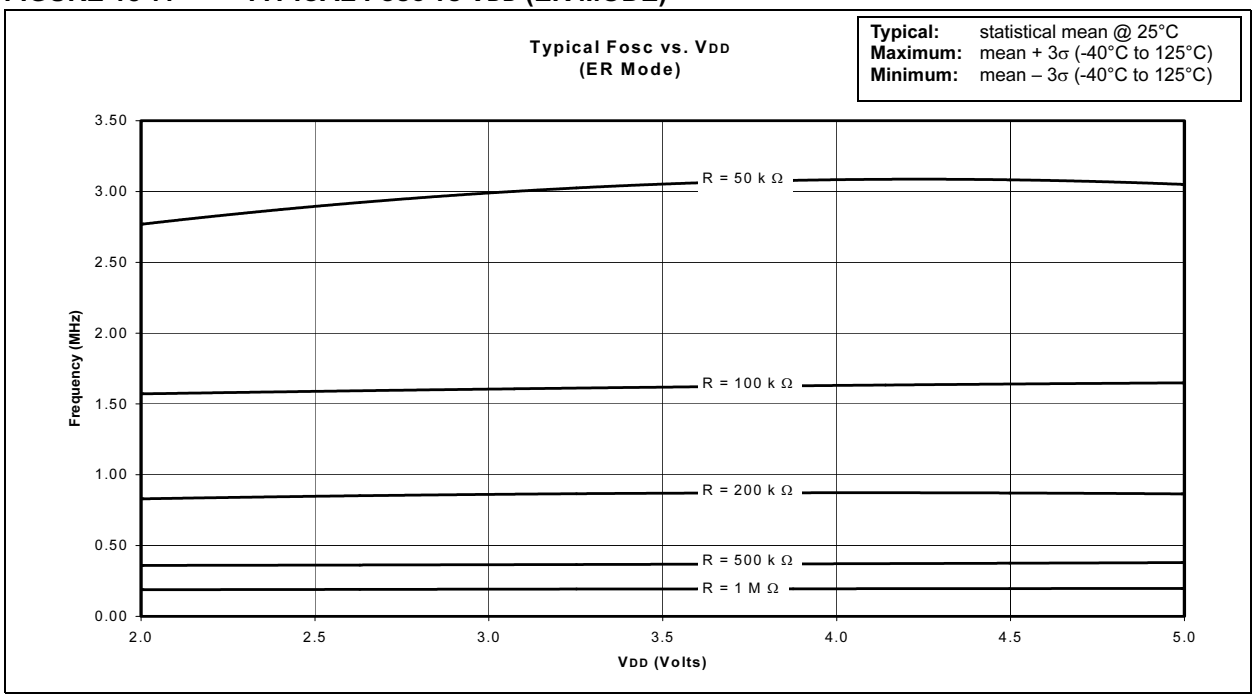


FIGURE 18-7: TYPICAL FOSC VS VDD (ER MODE)



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-8: TYPICAL INTERNAL RC FOSC VS VDD TEMPERATURE (-40 TO 125°C) INTERNAL 4 MHz OSCILLATOR

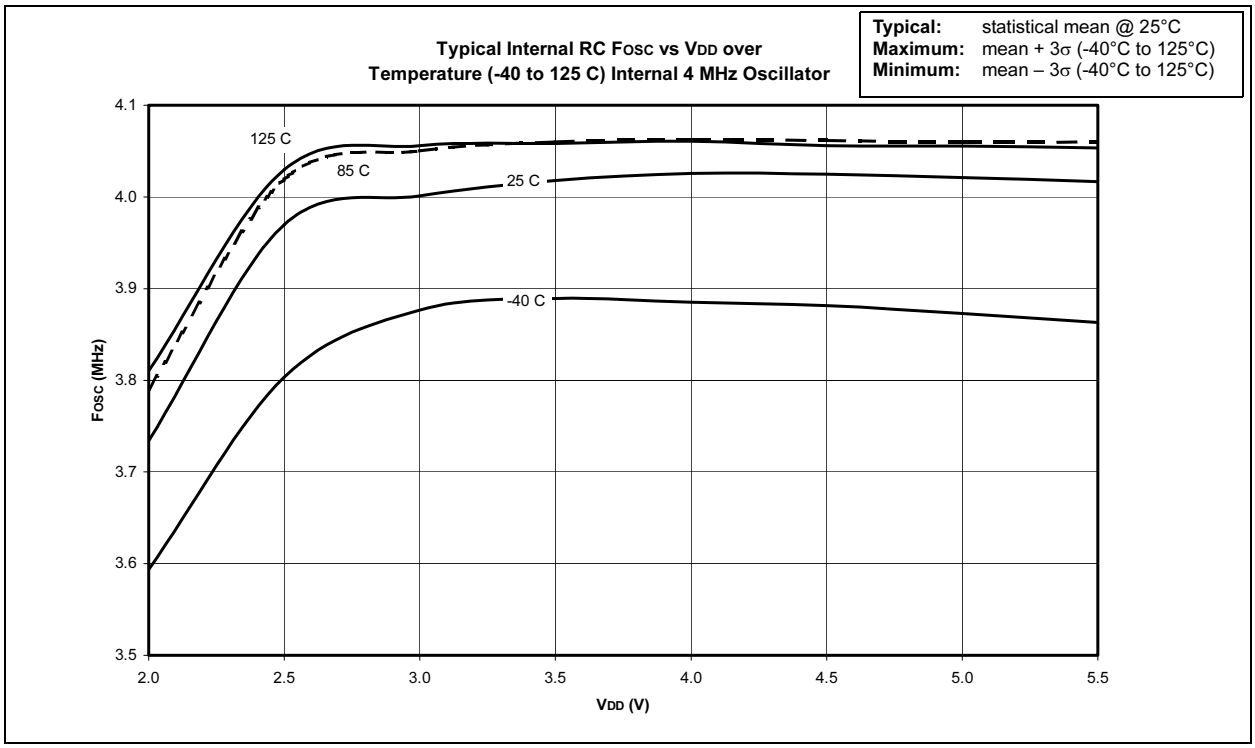
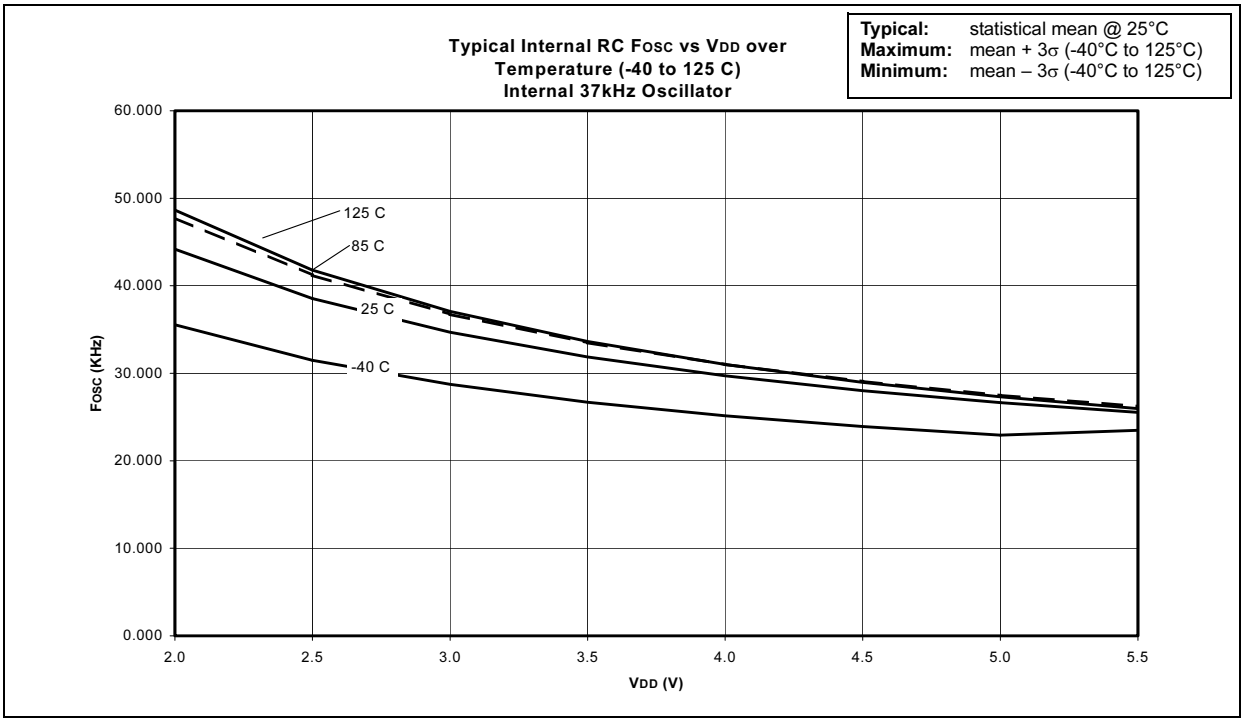


FIGURE 18-9: TYPICAL INTERNAL RC FOSC VS VDD OVER TEMPERATURE (-40 TO 125°C) INTERNAL 37 kHz OSCILLATOR



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-10: IPD VS VDD SLEEP MODE, ALL PERIPHERALS DISABLED

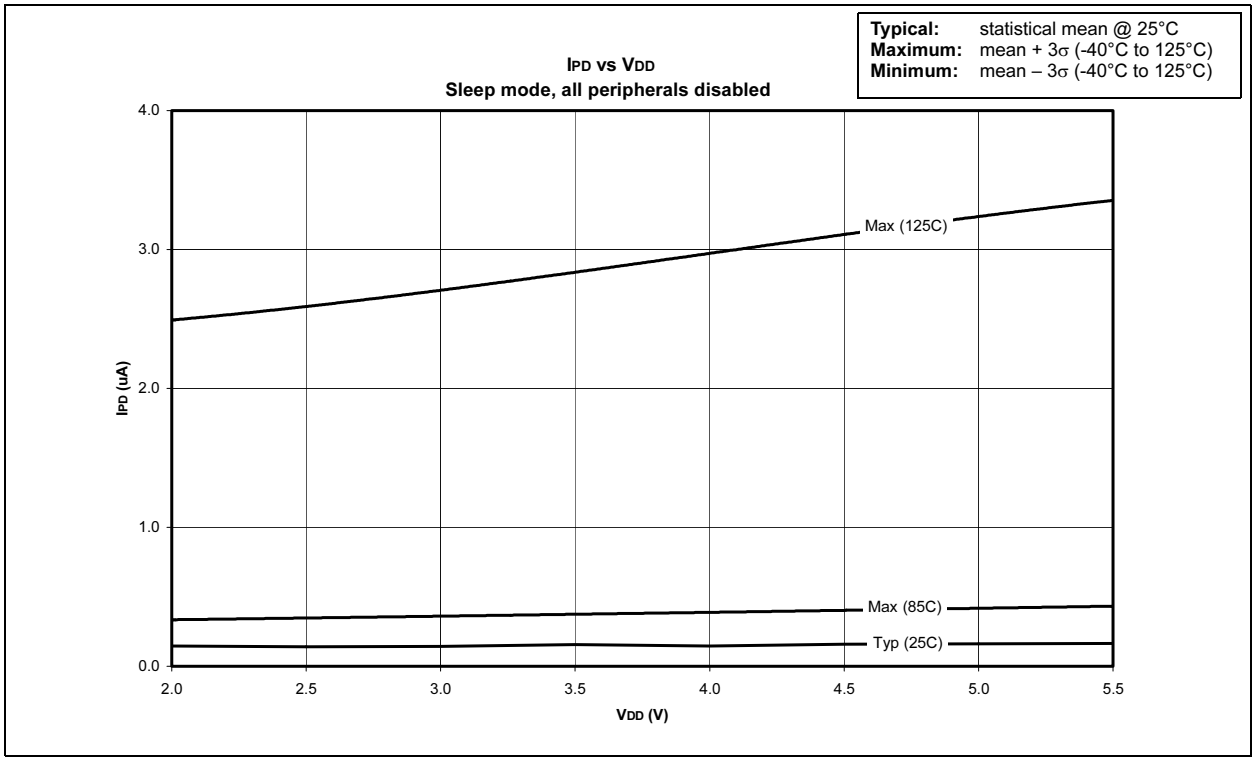
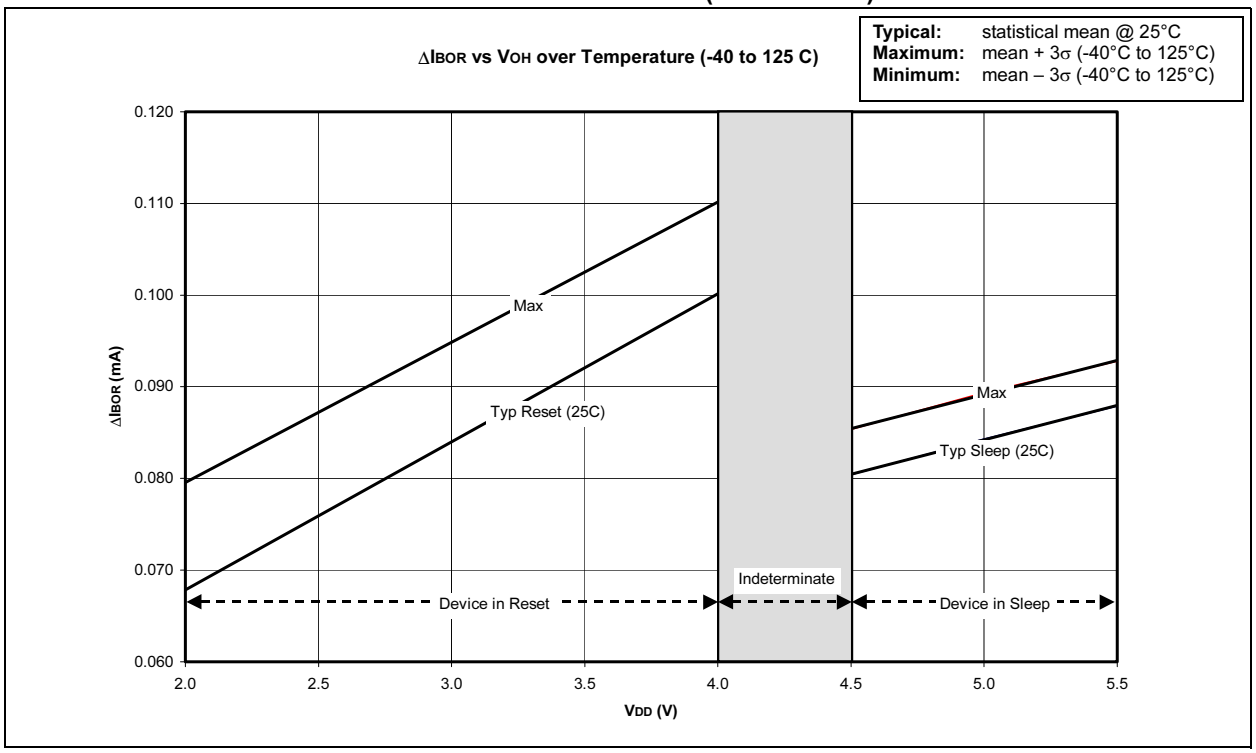


FIGURE 18-11: ΔIBOD VS VOH OVER TEMPERATURE (-40 to 125°C)



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-12: ΔITMR1OSC VS VDD OVER TEMP (0C to +70°C) SLEEP MODE, TIMER1 OSCILLATOR, 32 kHz XTAL

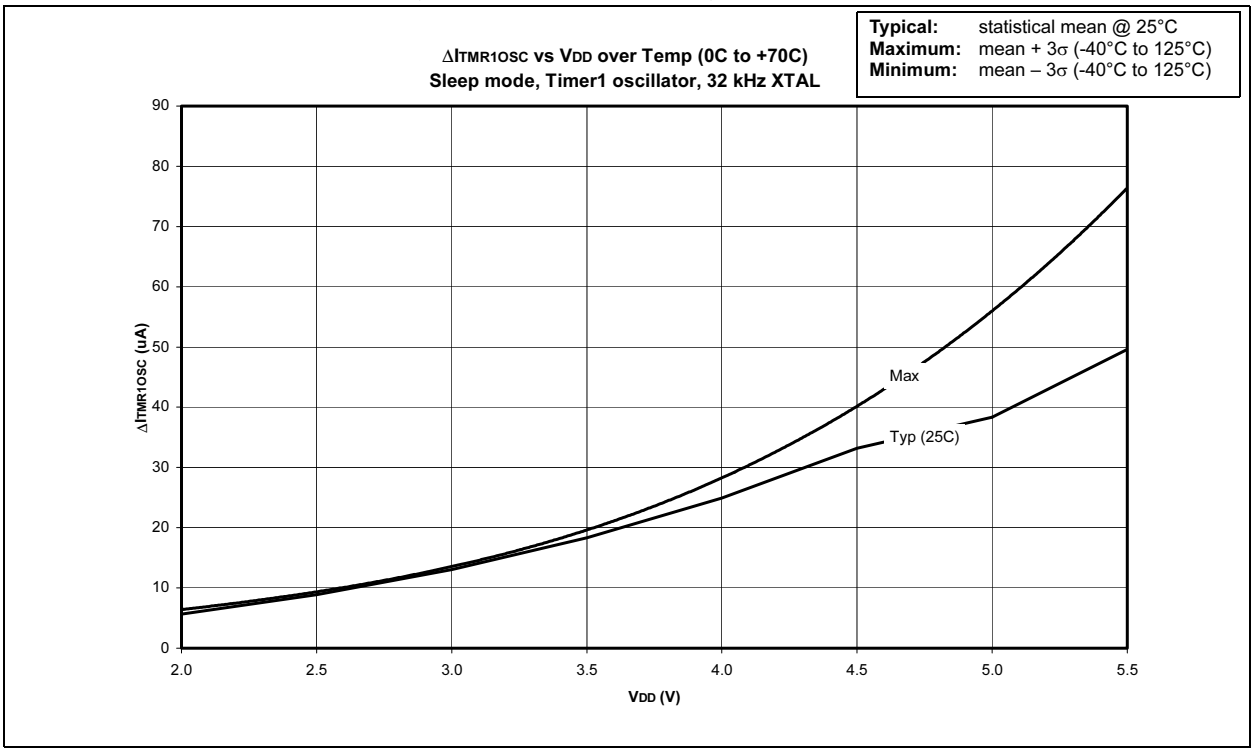
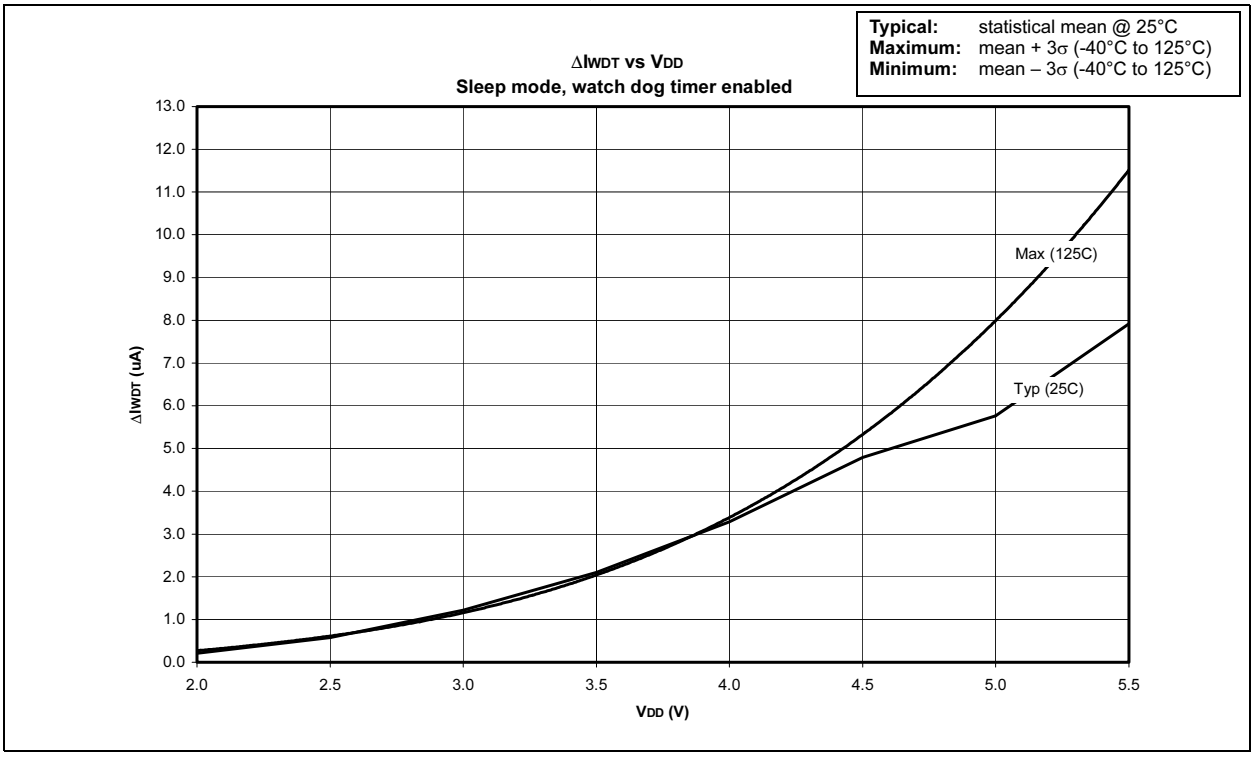


FIGURE 18-13: ΔIWDT VS VDD SLEEP MODE, WATCH DOG TIMER ENABLED



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-14: ΔICOMP VS VDD SLEEP MODE, COMPARATORS ENABLED

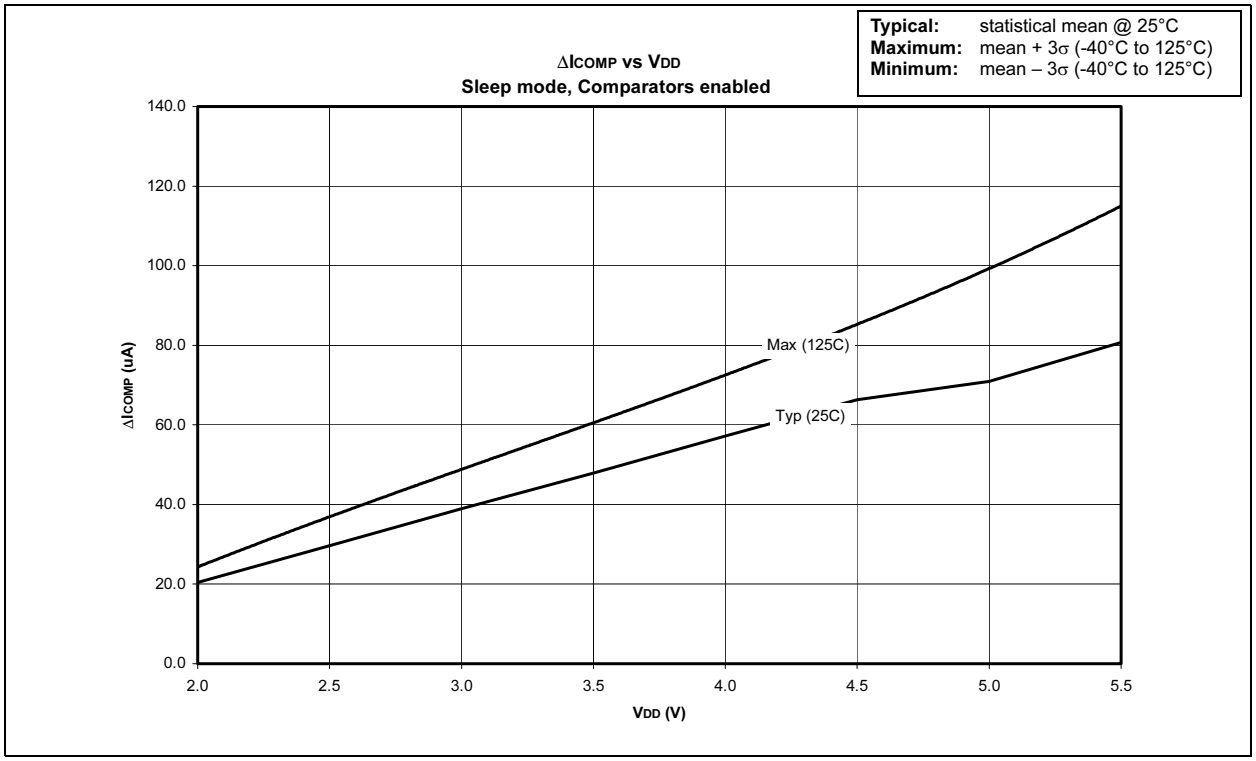
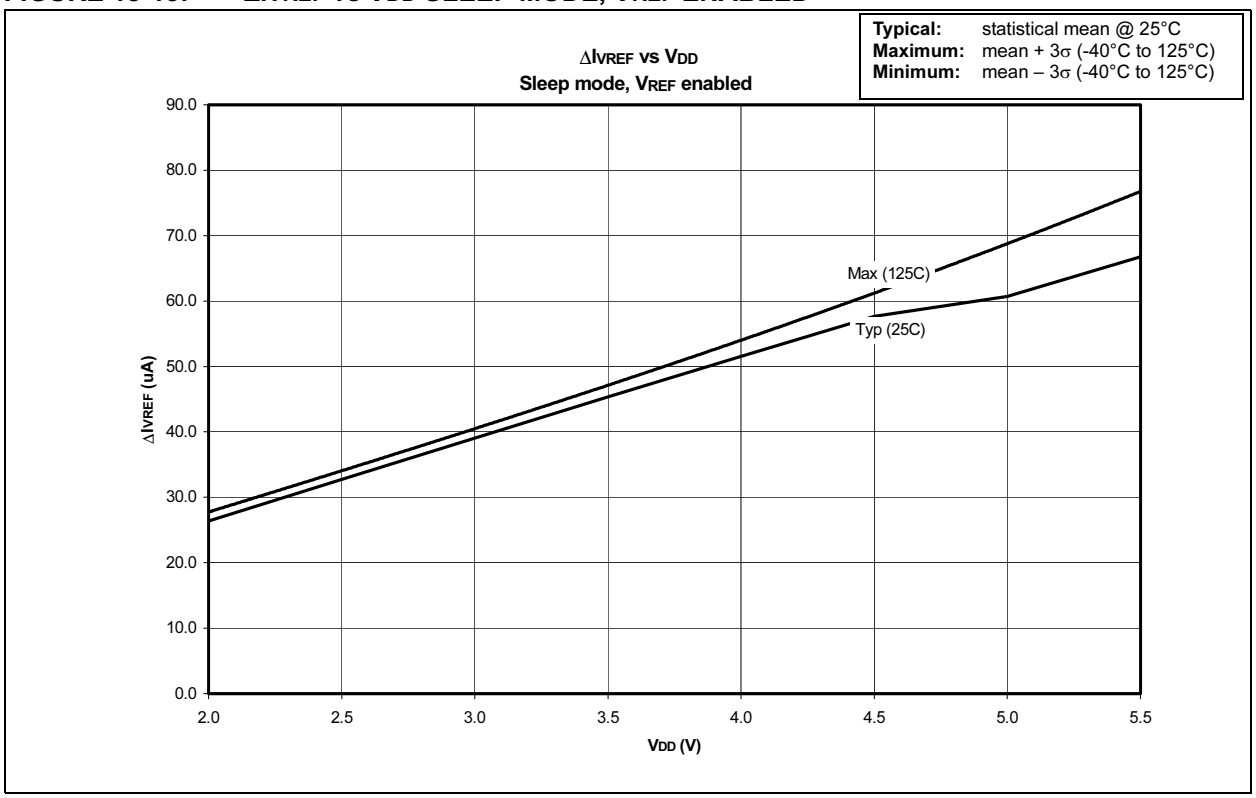


FIGURE 18-15: ΔIVREF VS VDD SLEEP MODE, VREF ENABLED



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-16: MINIMUM, TYPICAL and MAXIMUM WDT PERIOD vs VDD (-40°C to +125°C)

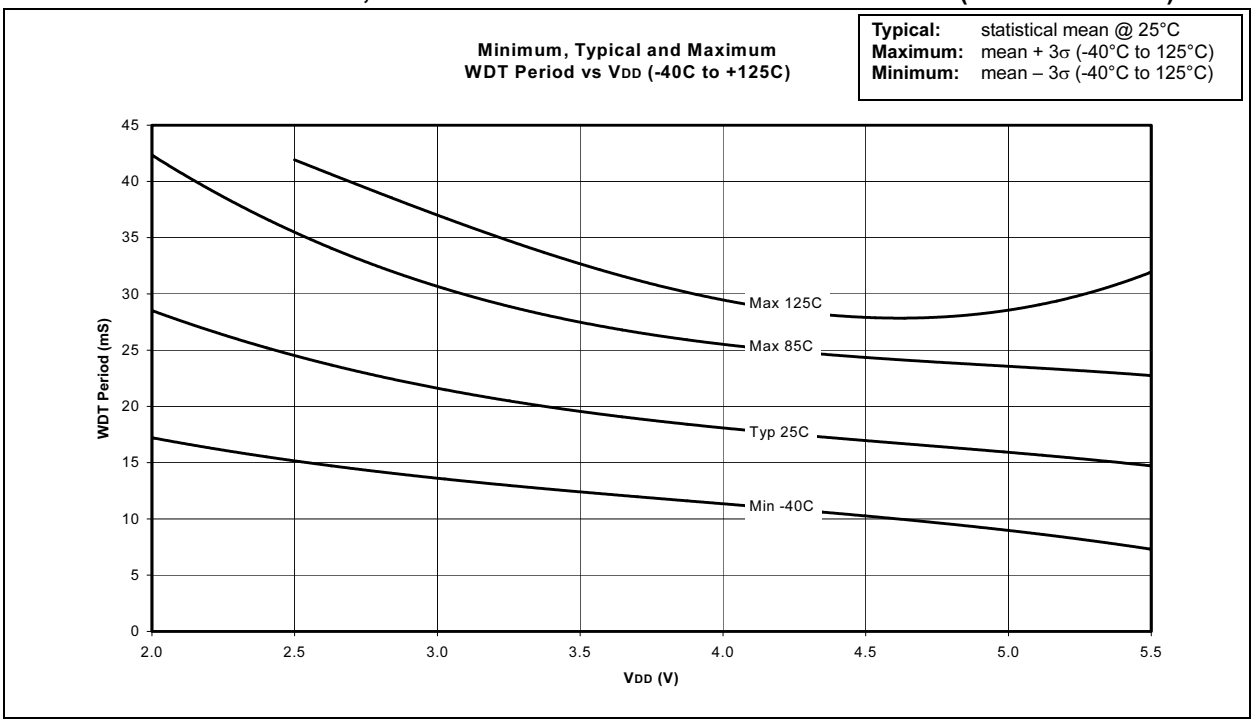
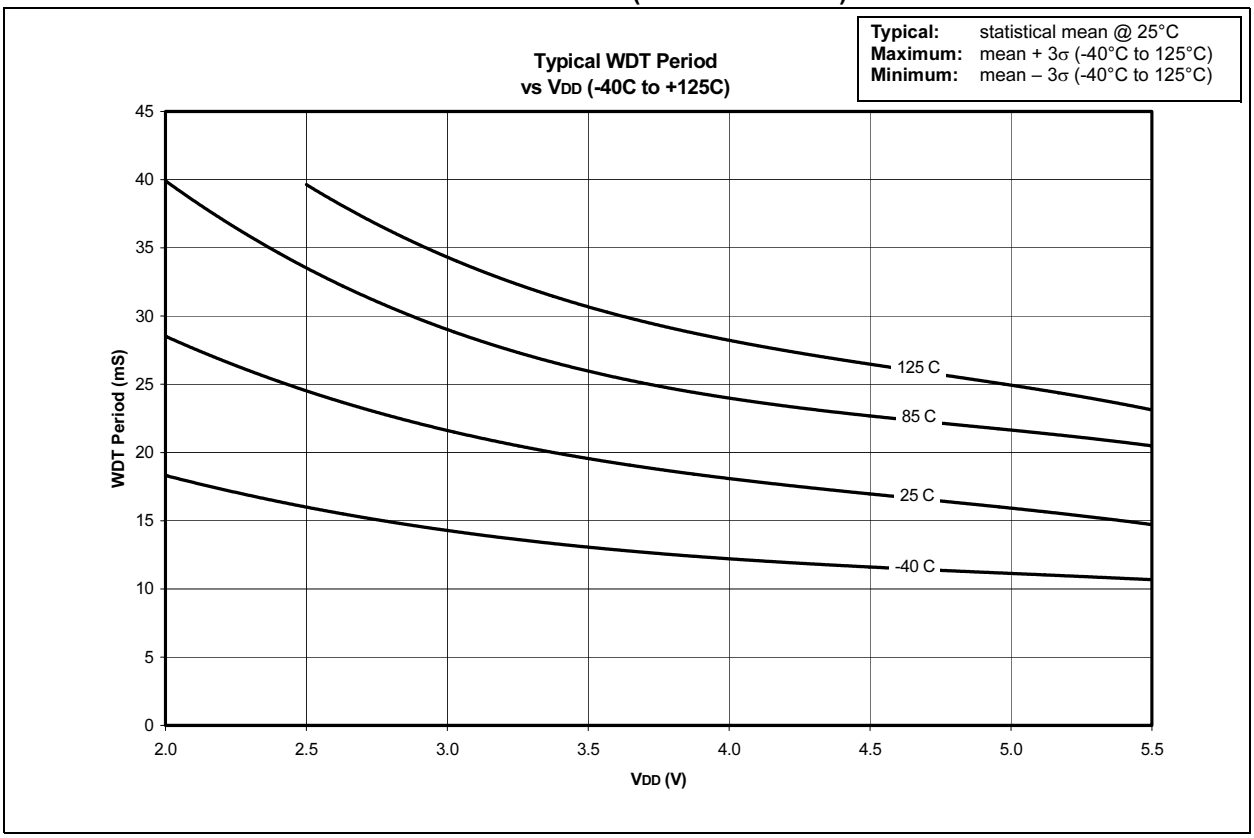


FIGURE 18-17: TYPICAL WDT PERIOD vs VDD (-40°C to +125°C)



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-18: VOH VS IOH OVER TEMP (C) VDD = 5V

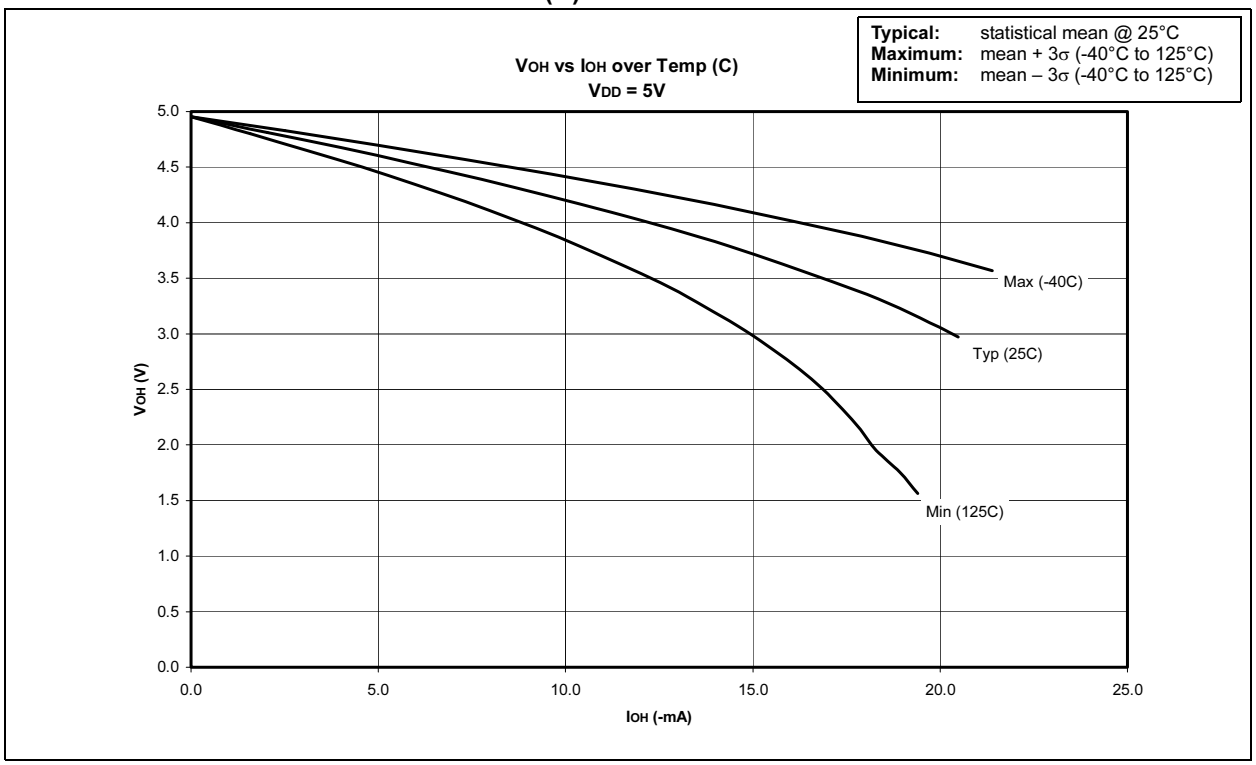
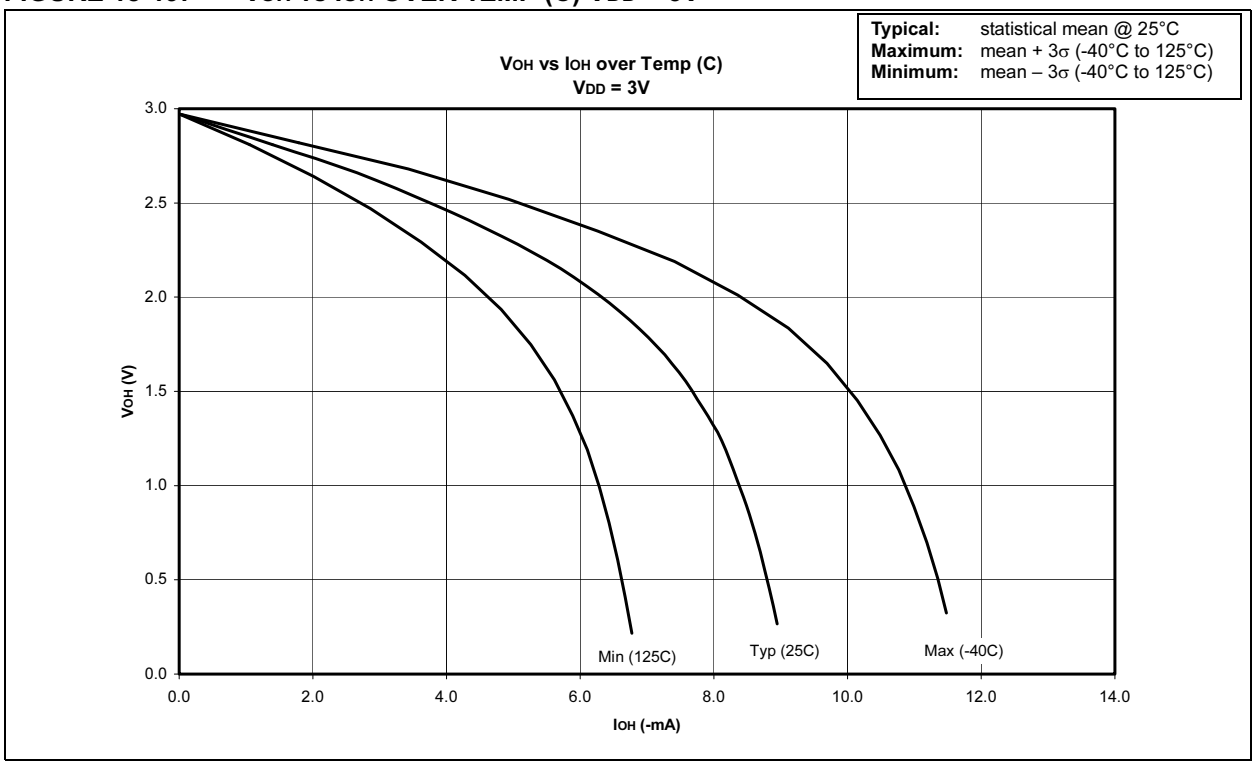


FIGURE 18-19: VOH VS IOH OVER TEMP (C) VDD = 3V



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-20: VOL VS IOL OVER TEMP (C) VDD = 5V

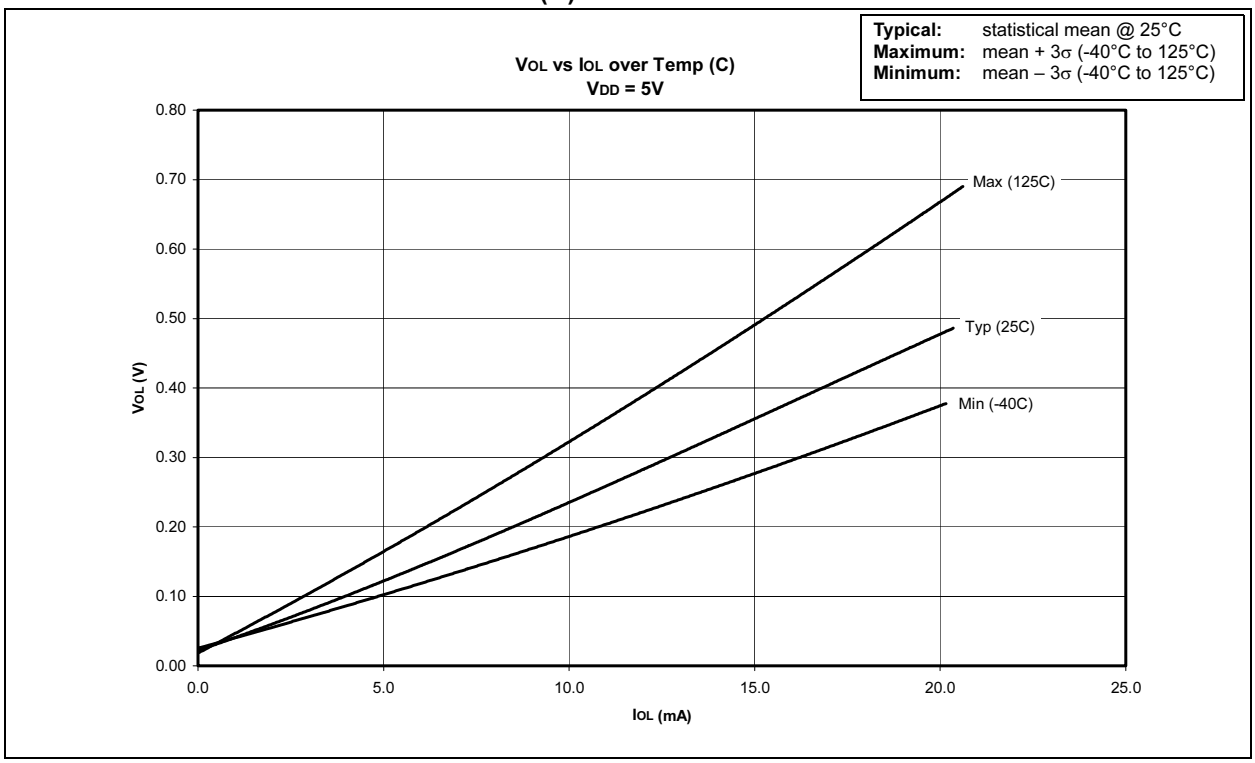
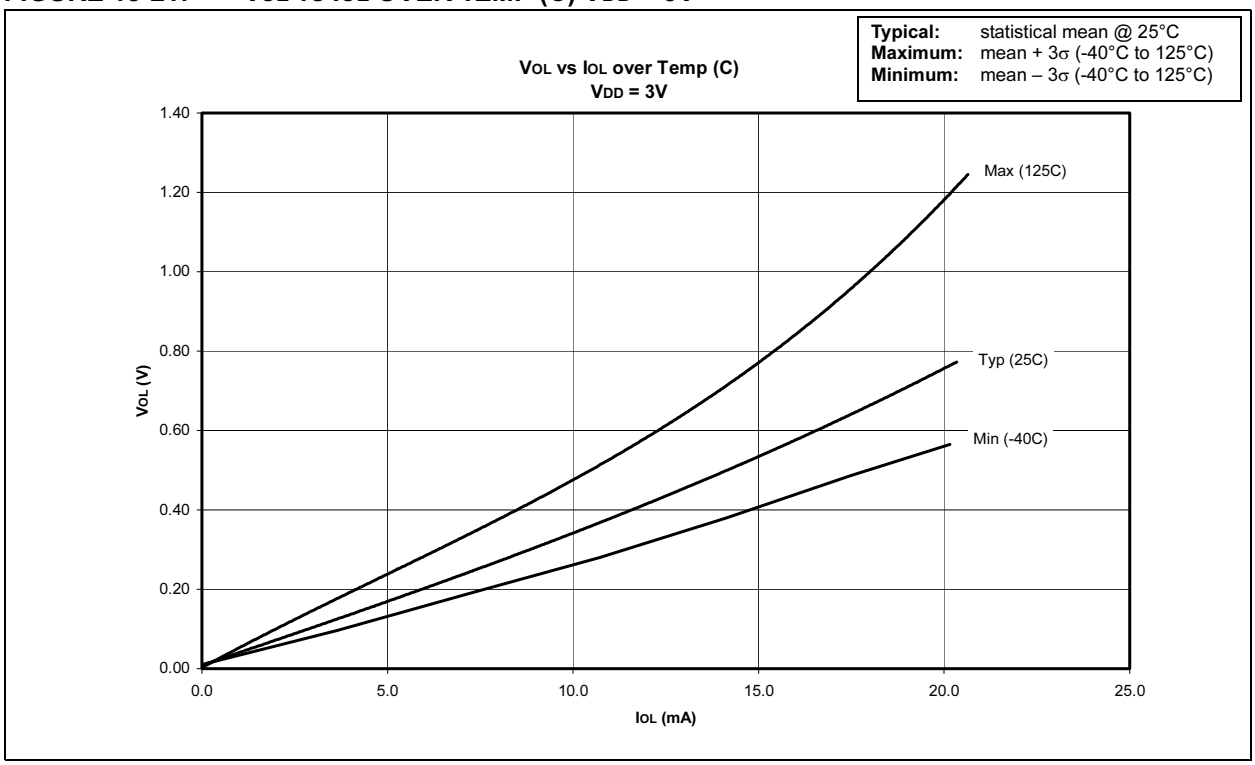


FIGURE 18-21: VOL VS IOL OVER TEMP (C) VDD = 3V



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-22: VIN VS VDD TTL

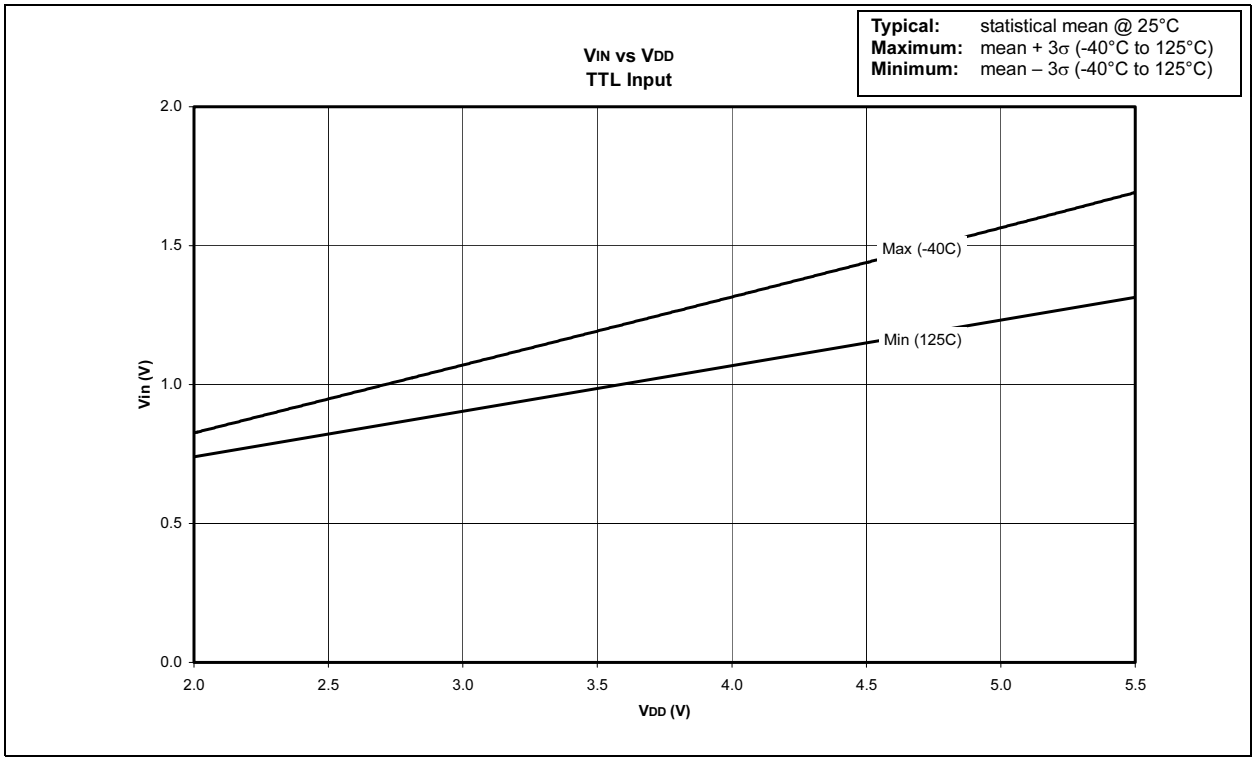
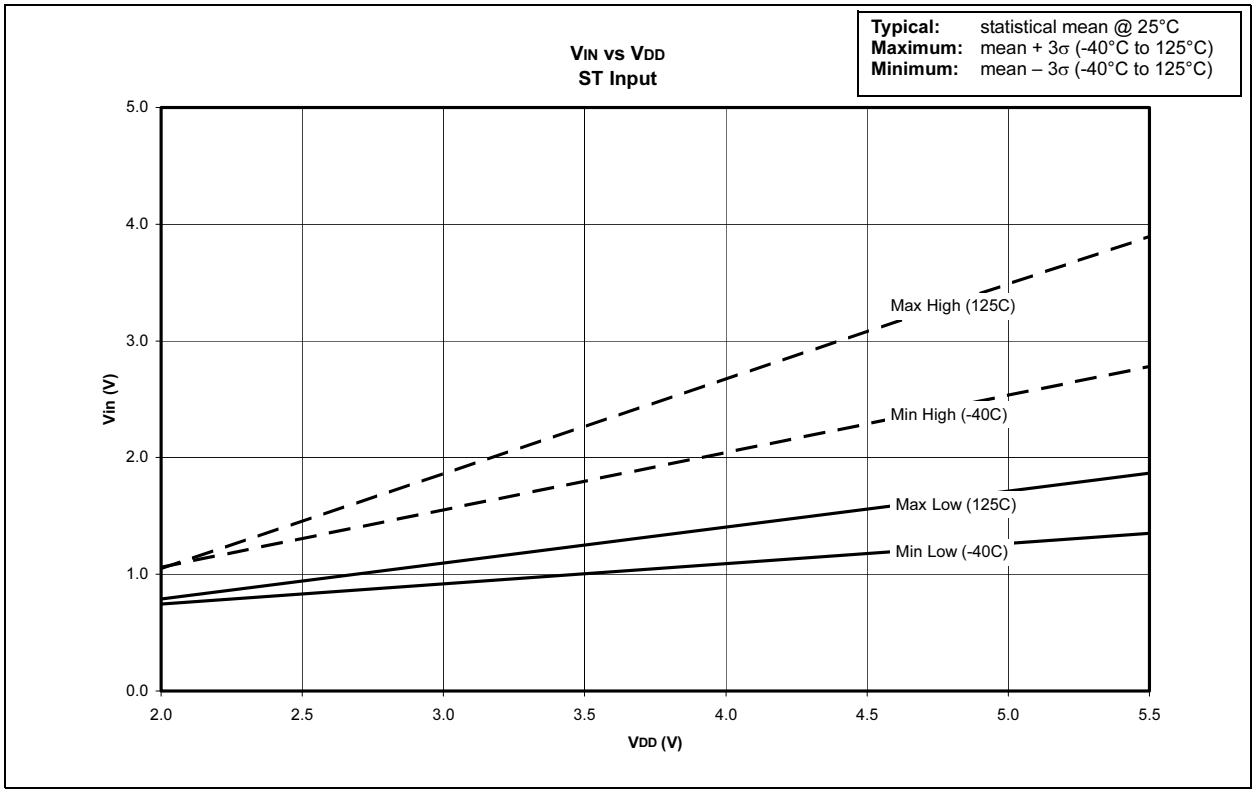


FIGURE 18-23: VIN VS VDD ST INPUT



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-24: MAXIMUM IDD VS VDD OVER TEMPERATURE (-40 TO +125°C) INTERNAL 37 kHz OSCILLATOR

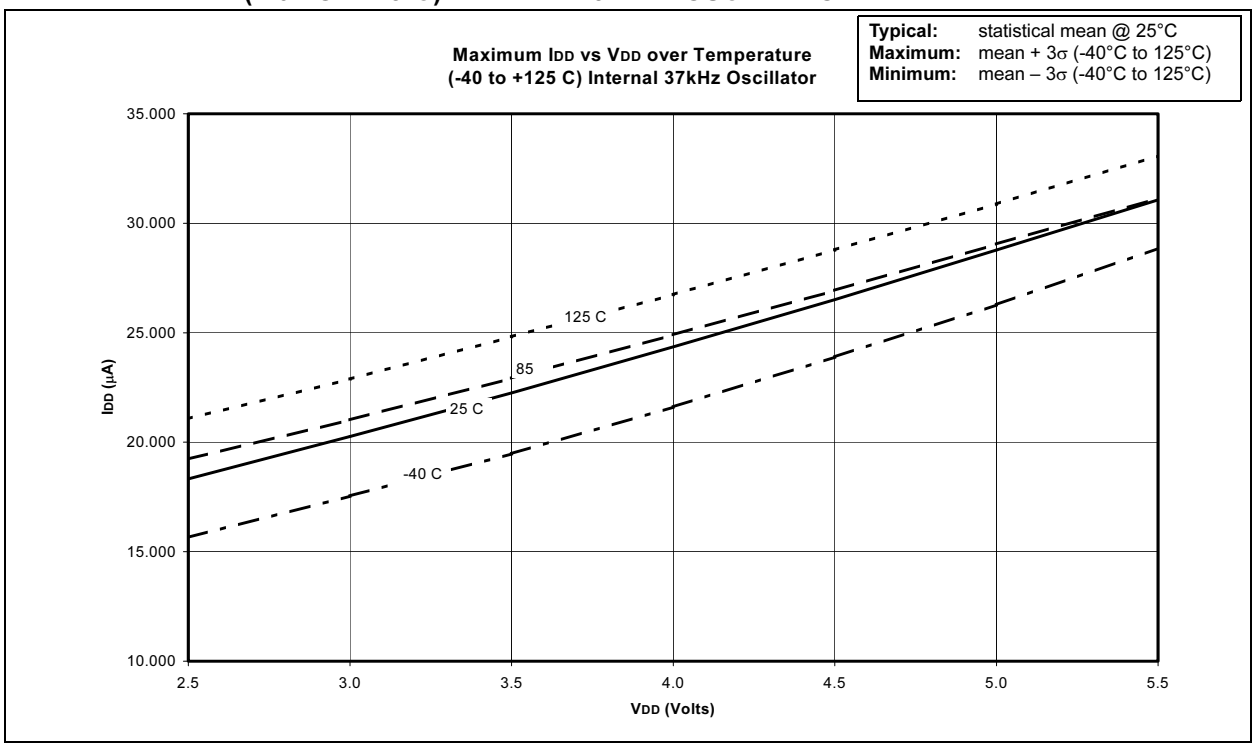
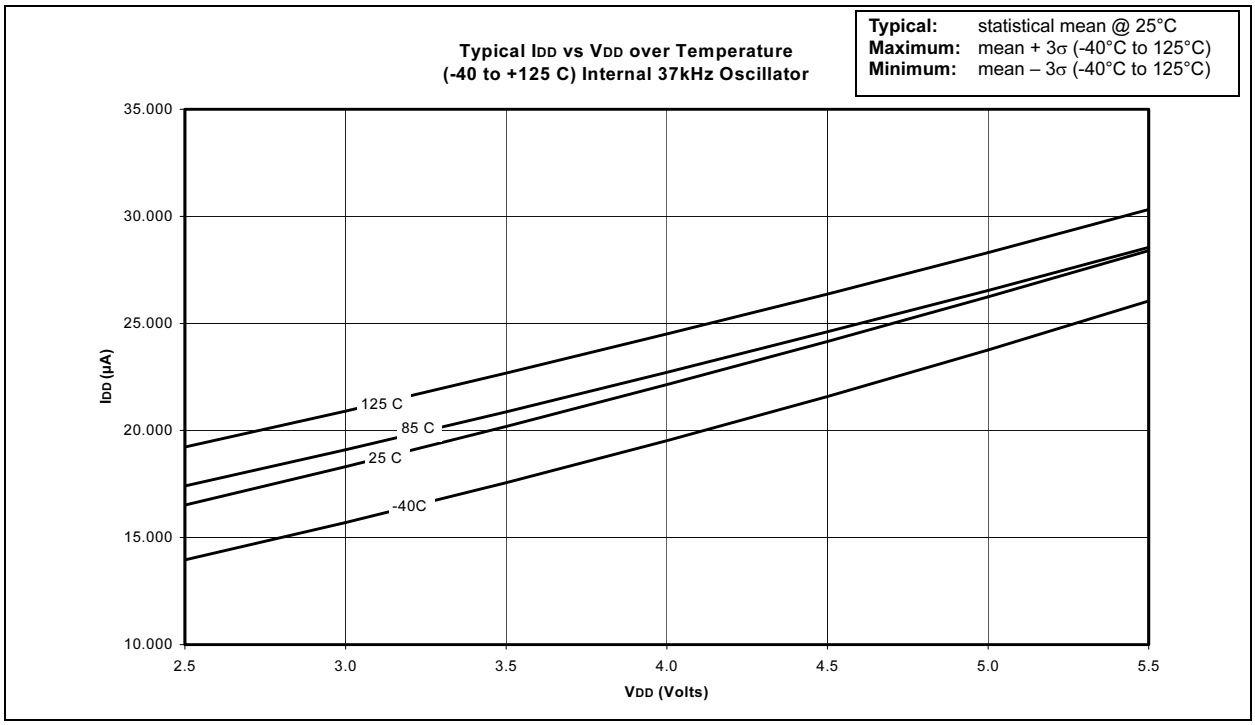


FIGURE 18-25: TYPICAL IDD VS VDD OVER TEMPERATURE (-40 TO +125°C) INTERNAL 37 kHz OSCILLATOR



Note: The graphs and tables provided in this section are for design guidance and are not tested.

FIGURE 18-26: MAXIMUM IDD VS VDD OVER TEMPERATURE (-40 TO +125°C) INTERNAL 4 MHz OSCILLATOR

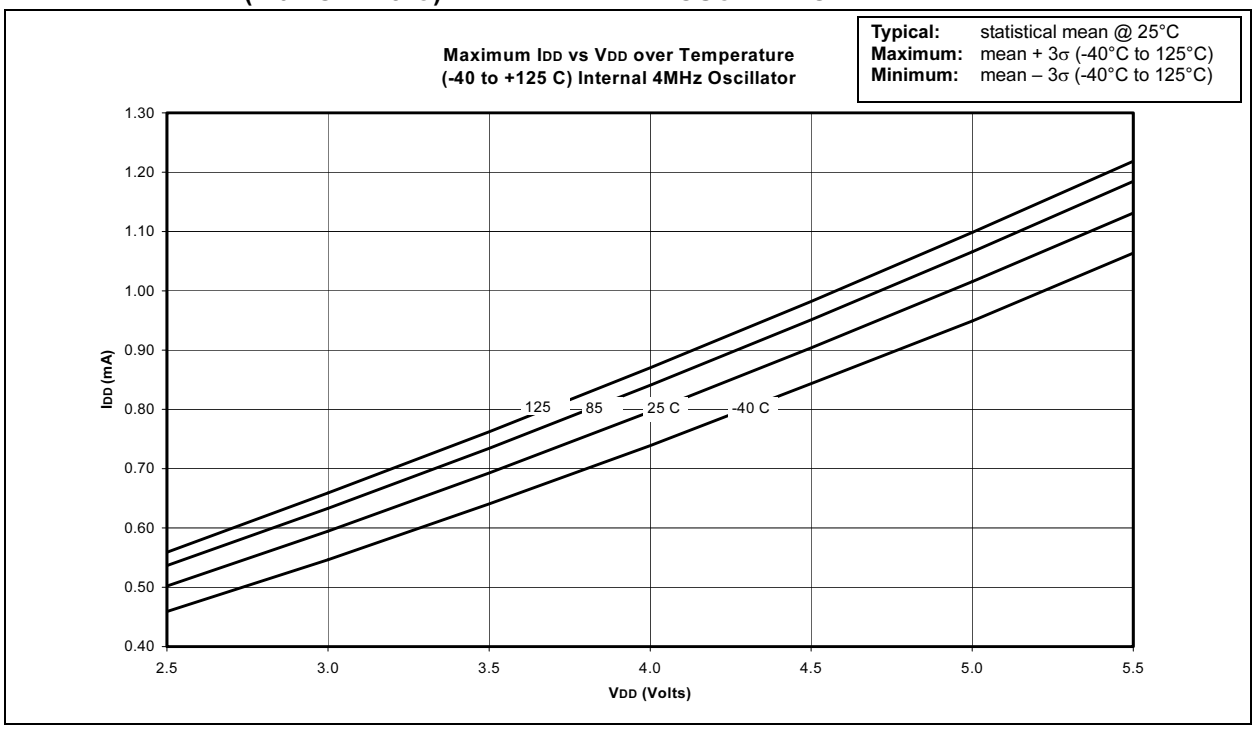
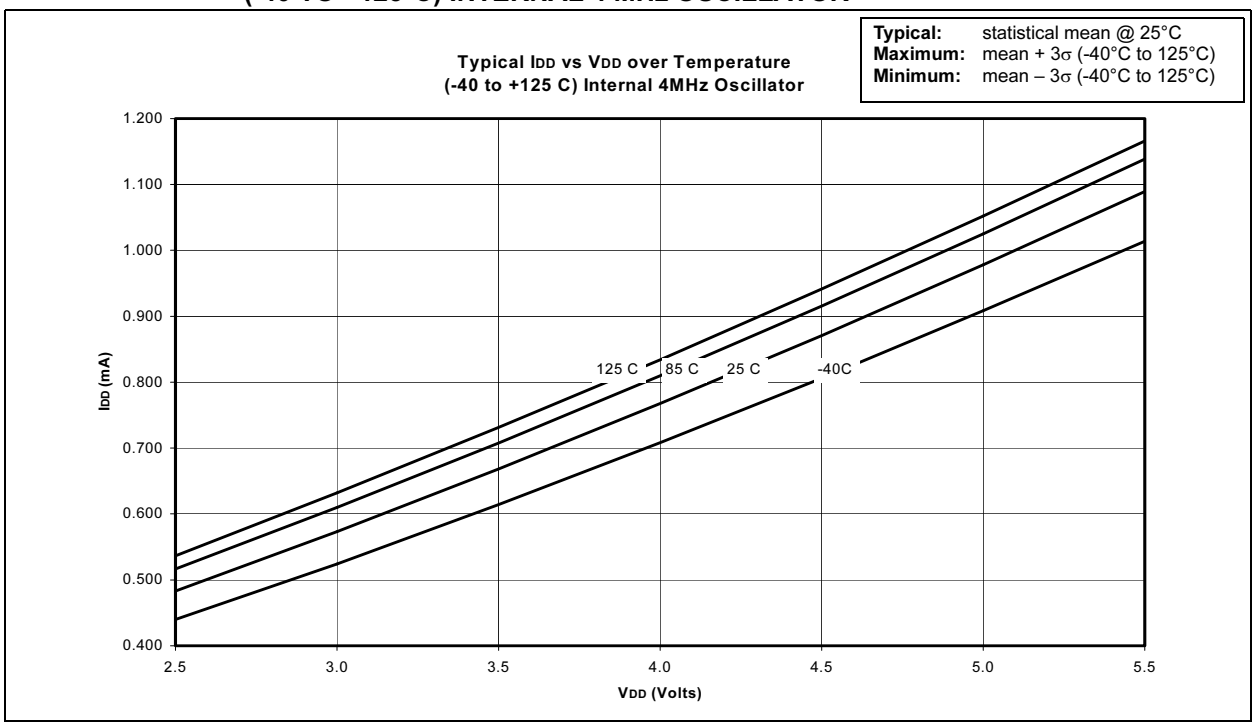


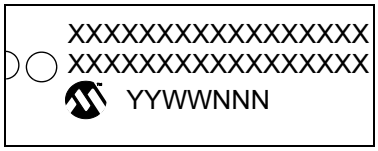
FIGURE 18-27: TYPICAL IDD VS VDD OVER TEMPERATURE (-40 TO +125°C) INTERNAL 4 MHz OSCILLATOR



19.0 PACKAGING INFORMATION

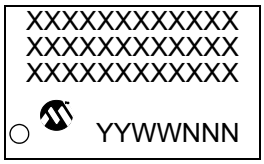
19.1 Package Marking Information

18-LEAD PDIP

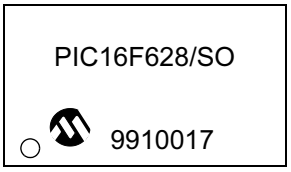


EXAMPLE

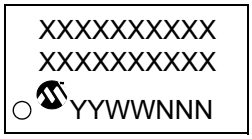
18-LEAD SOIC (.300")



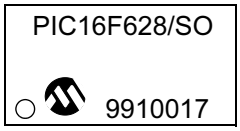
EXAMPLE



20-LEAD SSOP



EXAMPLE

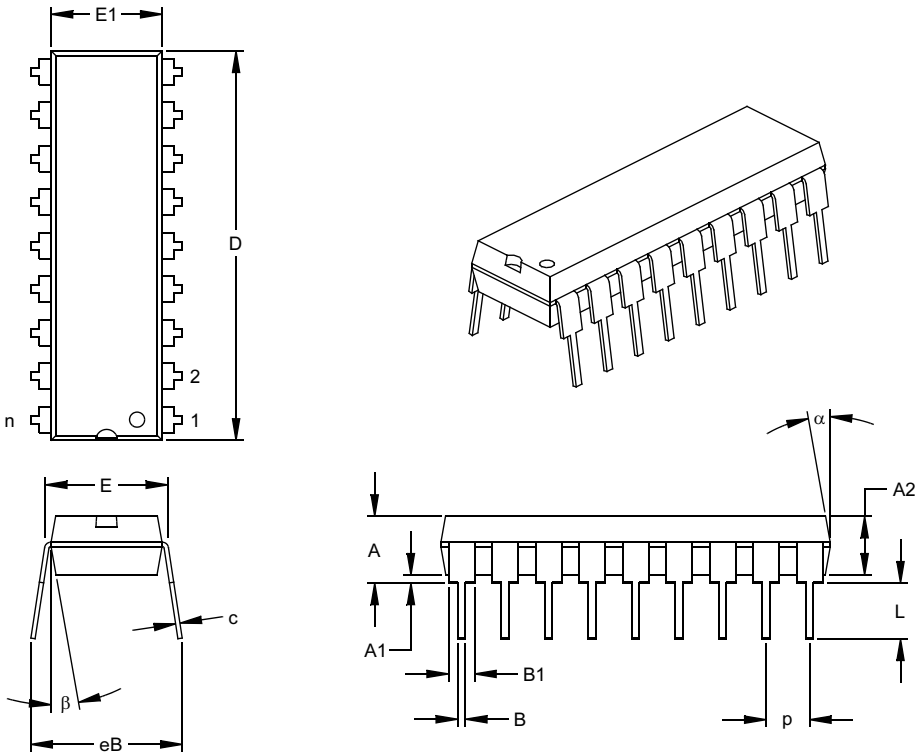


Legend:	MM...M	Microchip part number information
	XX...X	Customer specific information(1)
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week ‘01’)
	NNN	Alphanumeric traceability code

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line thus limiting the number of available characters for customer specific information.

* Standard OTP marking consists of Microchip part number, year code, week code, facility code, mask rev#, and assembly code. For OTP marking beyond this, certain price adders apply. Please check with your Microchip Sales Office. For QTP devices, any special marking adders are included in QTP price.

K04-007 18-Lead Plastic Dual In-line (P) – 300 mil

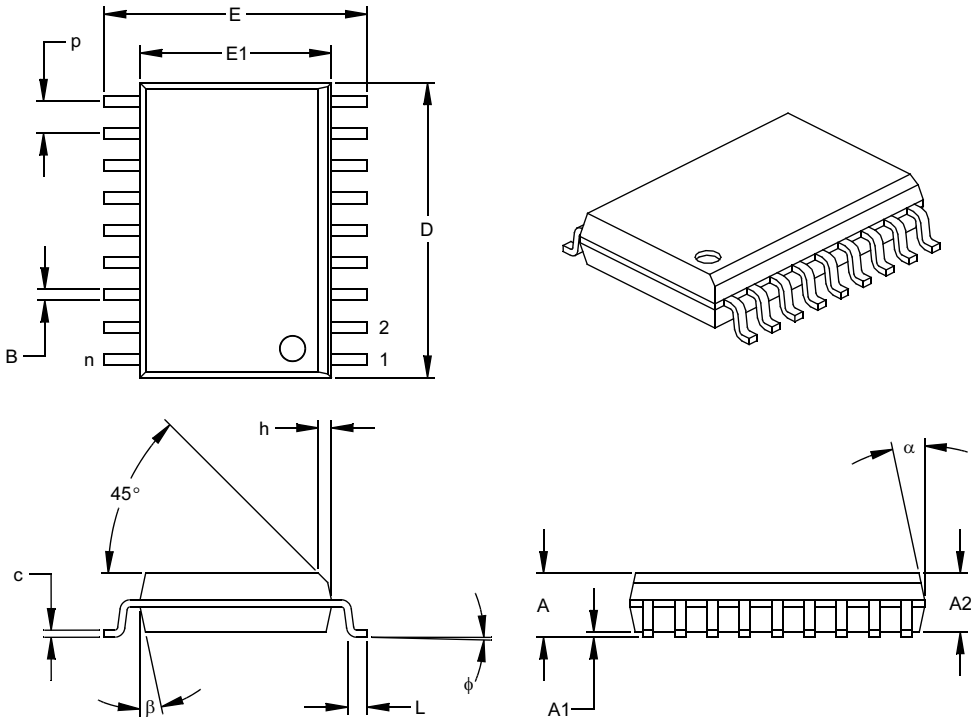


Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		18			18	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.140	.155	.170	3.56	3.94	4.32
Molded Package Thickness	A2	.115	.130	.145	2.92	3.30	3.68
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.300	.313	.325	7.62	7.94	8.26
Molded Package Width	E1	.240	.250	.260	6.10	6.35	6.60
Overall Length	D	.890	.898	.905	22.61	22.80	22.99
Tip to Seating Plane	L	.125	.130	.135	3.18	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.045	.058	.070	1.14	1.46	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.310	.370	.430	7.87	9.40	10.92
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MS-001
Drawing No. C04-007

K04-051 18-Lead Plastic Small Outline (SO) – Wide, 300 mil

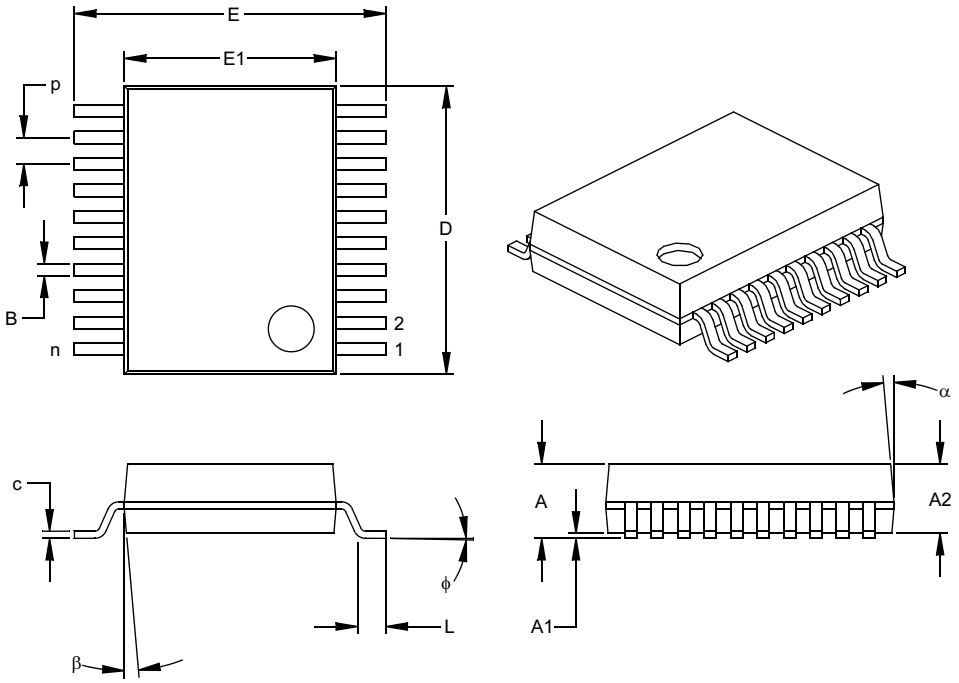


Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		18			18	
Pitch	p		.050			1.27	
Overall Height	A	.093	.099	.104	2.36	2.50	2.64
Molded Package Thickness	A2	.088	.091	.094	2.24	2.31	2.39
Standoff §	A1	.004	.008	.012	0.10	0.20	0.30
Overall Width	E	.394	.407	.420	10.01	10.34	10.67
Molded Package Width	E1	.291	.295	.299	7.39	7.49	7.59
Overall Length	D	.446	.454	.462	11.33	11.53	11.73
Chamfer Distance	h	.010	.020	.029	0.25	0.50	0.74
Foot Length	L	.016	.033	.050	0.41	0.84	1.27
Foot Angle	φ	0	4	8	0	4	8
Lead Thickness	c	.009	.011	.012	0.23	0.27	0.30
Lead Width	B	.014	.017	.020	0.36	0.42	0.51
Mold Draft Angle Top	α	0	12	15	0	12	15
Mold Draft Angle Bottom	β	0	12	15	0	12	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MS-013
Drawing No. C04-051

K04-072 20-Lead Plastic Shrink Small Outline (SS) – 5.30 mm



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		20			20	
Pitch	p		.026			0.65	
Overall Height	A	.068	.073	.078	1.73	1.85	1.98
Molded Package Thickness	A2	.064	.068	.072	1.63	1.73	1.83
Standoff §	A1	.002	.006	.010	0.05	0.15	0.25
Overall Width	E	.299	.309	.322	7.59	7.85	8.18
Molded Package Width	E1	.201	.207	.212	5.11	5.25	5.38
Overall Length	D	.278	.284	.289	7.06	7.20	7.34
Foot Length	L	.022	.030	.037	0.56	0.75	0.94
Lead Thickness	c	.004	.007	.010	0.10	0.18	0.25
Foot Angle	φ	0	4	8	0.00	101.60	203.20
Lead Width	B	.010	.013	.015	0.25	0.32	0.38
Mold Draft Angle Top	α	0	5	10	0	5	10
Mold Draft Angle Bottom	β	0	5	10	0	5	10

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MO-150
Drawing No. C04-072

INDEX

M

N

O

P

Q

R

S

T

U

V

W

X

NOTES:

ON-LINE SUPPORT

Microchip provides on-line support on the Microchip World Wide Web site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape® or Microsoft® Internet Explorer. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available at the following URL:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://ftp.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for products, Development Systems, technical information and more
- Listing of seminars and events

SYSTEMS INFORMATION AND UPGRADE HOT LINE

The Systems Information and Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive the most current upgrade kits.The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

1-480-792-7302 for the rest of the world.

092002

READER RESPONSE

It is our intention to provide you with the best documentation possible to ensure successful use of your Microchip product. If you wish to provide your comments on organization, clarity, subject matter, and ways in which our documentation can better serve you, please FAX your comments to the Technical Publications Manager at (480) 792-4150.

Please list the following information, and use this outline to provide us with your comments about this document.

To: Technical Publications Manager Total Pages Sent _____

RE: Reader Response

From: Name _____

Company _____

Address _____

City / State / ZIP / Country _____

Telephone: (_____) _____ - _____ FAX: (_____) _____ - _____

Application (optional):

Would you like a reply?____Y ____N

Device: PIC16F62X Literature Number: DS40300C

Questions:

1. What are the best features of this document?

_____

_____

2. How does this document meet your hardware and software development needs?

_____

_____

3. Do you find the organization of this document easy to follow? If not, why?

_____

_____

4. What additions to the document do you think would enhance the structure and subject?

_____

_____

5. What deletions from the document could be made without affecting the overall usefulness?

_____

_____

6. Is there any incorrect or misleading information (what and where)?

_____

_____

7. How would you improve this document?

_____

_____

PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	-XX	X	/XX	XXX
Device	Frequency Range	Temperature Range	Package	Pattern

Device	PIC16F62X:	Standard VDD range 3.0V to 5.5V	
	PIC16F62XT	VDD range 3.0V to 5.5V (Tape and Reel)	
	PIC16LF62X:	VDD range 2.0V to 5.5V	
	PIC16LF62XT:	VDD range 2.0V to 5.5V (Tape and Reel)	
Frequency Range	04 =	200 kHz (LP osc)	
	04 =	4 MHz (XT and ER osc)	
	20 =	20 MHz (HS osc)	
Temperature Range	- =	0°C to +70°C	
	I =	-40°C to +85°C	
	E =	-40°C to +125°C	
Package	P =	PDIP	
	SO =	SOIC (Gull Wing, 300 mil body)	
	SS =	SSOP (209 mil)	
Pattern	3-Digit Pattern Code for QTP (blank otherwise).		

Examples:

a) PIC16F627 - 04/P 301 = Commercial Temp., PDIP package, 4 MHz, normal VDD limits, QTP pattern #301.

b) PIC16LF627 - 04I/SO = Industrial Temp., SOIC package, 200 kHz, extended VDD limits.

* JW Devices are UV erasable and can be programmed to any device configuration. JW Devices meet the electrical requirement of each oscillator type.

Sales and Support

Data Sheets
Products supported by a preliminary Data Sheet may have an errata sheet describing minor operational differences and recommended workarounds. To determine if an errata sheet exists for a particular device, please contact one of the following:

1. Your local Microchip sales office
2. The Microchip Corporate Literature Center U.S. FAX: (480) 792-7277
3. The Microchip Worldwide Site (www.microchip.com)

Please specify which device, revision of silicon and Data Sheet (include Literature #) you are using.

New Customer Notification System
Register on our web site (www.microchip.com/cn) to receive the most current information on our products.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-4338

Atlanta
3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200 Fax: 86-28-86766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Hong Kong SAR
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380 Fax: 86-755-82966626

China - Qingdao
Rm. B503, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Microchip Technology (Barbados) Inc.,
Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria
Microchip Technology Austria GmbH
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Microchip Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

12/05/02

19-4323; Rev 11; 2/03

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multidrop RS-232 Networks

Features

Superior to Bipolar

- ♦ **Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)**
- ♦ **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- ♦ **Meet All EIA/TIA-232E and V.28 Specifications**
- ♦ **Multiple Drivers and Receivers**
- ♦ **3-State Driver and Receiver Outputs**
- ♦ **Open-Line Detection (MAX243)**

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	0.1	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package

+5V-Powered, Multichannel RS-232 Drivers/Receivers

ABSOLUTE MAXIMUM RATINGS—MAX220/222/232A/233A/242/243

Supply Voltage (V_{CC}) .....-0.3V to +6V
Input Voltages
T_{IN}.....-0.3V to (V_{CC} - 0.3V)
R_{IN} (Except MAX220) .....±30V
R_{IN} (MAX220).....±25V
T_{OUT} (Except MAX220) (Note 1) .....±15V
T_{OUT} (MAX220).....±13.2V
Output Voltages
T_{OUT}.....±15V
R_{OUT}.....-0.3V to (V_{CC} + 0.3V)
Driver/Receiver Output Short Circuited to GND.....Continuous
Continuous Power Dissipation (T_A = +70°C)
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C)....842mW
18-Pin Plastic DIP (derate 11.11mW/°C above +70°C)....889mW
20-Pin Plastic DIP (derate 8.00mW/°C above +70°C) ..440mW
16-Pin Narrow SO (derate 8.70mW/°C above +70°C) ...696mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C).....762mW
18-Pin Wide SO (derate 9.52mW/°C above +70°C).....762mW
20-Pin Wide SO (derate 10.00mW/°C above +70°C)....800mW
20-Pin SSOP (derate 8.00mW/°C above +70°C) 640mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C).....800mW
18-Pin CERDIP (derate 10.53mW/°C above +70°C).....842mW
Operating Temperature Ranges
MAX2_ _AC_ _, MAX2_ _C_ _.....0°C to +70°C
MAX2_ _AE_ _, MAX2_ _E_ _ .....-40°C to +85°C
MAX2_ _AM_ _, MAX2_ _M_ _.....-55°C to +125°C
Storage Temperature Range .....-65°C to +160°C
Lead Temperature (soldering, 10s) .....+300°C

Note 1: Input voltage measured with T_{OUT} in high-impedance state, $\overline{SHDN}$ or V_{CC} = 0V.
Note 2: For the MAX220, V+ and V- can have a maximum magnitude of 7V, but their absolute difference cannot exceed 13V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, MAX220, C1 = 0.047µF, C2–C4 = 0.33µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±8		V
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High	All devices except MAX220		2	1.4		V
	MAX220: V_{CC} = 5.0V		2.4			
Logic Pull-Up/Input Current	All except MAX220, normal operation			5	40	µA
	$\overline{SHDN}$ = 0V, MAX222/242, shutdown, MAX220			±0.01	±1	
Output Leakage Current	V_{CC} = 5.5V, $\overline{SHDN}$ = 0V, V_{OUT} = ±15V, MAX222/242			±0.01	±10	µA
	V_{CC} = $\overline{SHDN}$ = 0V, V_{OUT} = ±15V			±0.01	±10	
Data Rate				200	116	kbps
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V		±7	±22		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±30	V
RS-232 Input Threshold Low	V_{CC} = 5V	All except MAX243 $R2_{IN}$	0.8	1.3		V
		MAX243 $R2_{IN}$ (Note 2)	-3			
RS-232 Input Threshold High	V_{CC} = 5V	All except MAX243 $R2_{IN}$		1.8	2.4	V
		MAX243 $R2_{IN}$ (Note 2)		-0.5	-0.1	
RS-232 Input Hysteresis	All except MAX243, V_{CC} = 5V, no hysteresis in shdn.		0.2	0.5	1	V
	MAX243			1		
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243 (continued)

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, MAX220, C1 = 0.047µF, C2–C4 = 0.33µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
TTL/CMOS Output Leakage Current	$\overline{SHDN}$ = V_{CC} or $\overline{EN}$ = V_{CC} ($\overline{SHDN}$ = 0V for MAX222), 0V ≤ V_{OUT} ≤ V_{CC}			±0.05	±10	µA
$\overline{EN}$ Input Threshold Low	MAX242			1.4	0.8	V
$\overline{EN}$ Input Threshold High	MAX242		2.0	1.4		V
Operating Supply Voltage			4.5		5.5	V
V_{CC} Supply Current ($\overline{SHDN}$ = V_{CC}), Figures 5, 6, 11, 19	No load	MAX220		0.5	2	mA
		MAX222/232A/233A/242/243		4	10	
	3kΩ load both inputs	MAX220		12		
		MAX222/232A/233A/242/243		15		
Shutdown Supply Current	MAX222/242	T_A = +25°C		0.1	10	µA
		T_A = 0°C to +70°C		2	50	
		T_A = -40°C to +85°C		2	50	
		T_A = -55°C to +125°C		35	100	
$\overline{SHDN}$ Input Leakage Current	MAX222/242				±1	µA
$\overline{SHDN}$ Threshold Low	MAX222/242			1.4	0.8	V
$\overline{SHDN}$ Threshold High	MAX222/242		2.0	1.4		V
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V	MAX222/232A/233A/242/243	6	12	30	V/µs
		MAX220	1.5	3	30	
Transmitter Propagation Delay TLL to RS-232 (Normal Operation), Figure 1	t_{PHLT}	MAX222/232A/233A/242/243		1.3	3.5	µs
		MAX220		4	10	
	t_{PLHT}	MAX222/232A/233A/242/243		1.5	3.5	
		MAX220		5	10	
Receiver Propagation Delay RS-232 to TLL (Normal Operation), Figure 2	t_{PHLR}	MAX222/232A/233A/242/243		0.5	1	µs
		MAX220		0.6	3	
	t_{PLHR}	MAX222/232A/233A/242/243		0.6	1	
		MAX220		0.8	3	
Receiver Propagation Delay RS-232 to TLL (Shutdown), Figure 2	t_{PHLS}	MAX242		0.5	10	µs
	t_{PLHS}	MAX242		2.5	10	
Receiver-Output Enable Time, Figure 3	t_{ER}	MAX242		125	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}	MAX242		160	500	ns
Transmitter-Output Enable Time ($\overline{SHDN}$ Goes High), Figure 4	t_{ET}	MAX222/242, 0.1µF caps (includes charge-pump start-up)		250		µs
Transmitter-Output Disable Time ($\overline{SHDN}$ Goes Low), Figure 4	t_{DT}	MAX222/242, 0.1µF caps		600		ns
Transmitter + to - Propagation Delay Difference (Normal Operation)	t_{PHLT} - t_{PLHT}	MAX222/232A/233A/242/243		300		ns
		MAX220		2000		
Receiver + to - Propagation Delay Difference (Normal Operation)	t_{PHLR} - t_{PLHR}	MAX222/232A/233A/242/243		100		ns
		MAX220		225		

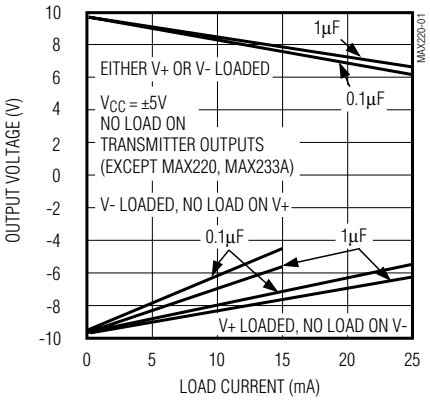
Note 3: MAX243 $R2_{OUT}$ is guaranteed to be low when $R2_{IN}$ is ≥ 0V or is floating.

+5V-Powered, Multichannel RS-232 Drivers/Receivers

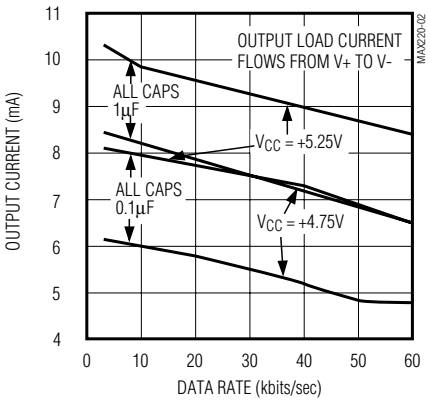
Typical Operating Characteristics

MAX220/MAX222/MAX232A/MAX233A/MAX242/MAX243

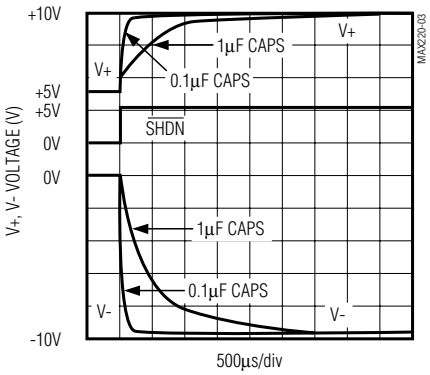
OUTPUT VOLTAGE vs. LOAD CURRENT



AVAILABLE OUTPUT CURRENT vs. DATA RATE



MAX222/MAX242 ON-TIME EXITING SHUTDOWN



ABSOLUTE MAXIMUM RATINGS—MAX223/MAX230–MAX241

V_{CC} .....-0.3V to +6V
V+ .....(V_{CC} - 0.3V) to +14V
V- .....+0.3V to -14V
Input Voltages
T_{IN} .....-0.3V to (V_{CC} + 0.3V)
R_{IN}.....±30V
Output Voltages
T_{OUT} .....(V+ + 0.3V) to (V- - 0.3V)
R_{OUT}.....-0.3V to (V_{CC} + 0.3V)
Short-Circuit Duration, T_{OUT} .....Continuous
Continuous Power Dissipation (T_A = +70°C)
14-Pin Plastic DIP (derate 10.00mW/°C above +70°C)....800mW
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C)....842mW
20-Pin Plastic DIP (derate 11.11mW/°C above +70°C)....889mW
24-Pin Narrow Plastic DIP
(derate 13.33mW/°C above +70°C) 1.07W
24-Pin Plastic DIP (derate 9.09mW/°C above +70°C).....500mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C).....762mW
20-Pin Wide SO (derate 10 00mW/°C above +70°C).....800mW
24-Pin Wide SO (derate 11.76mW/°C above +70°C).....941mW
28-Pin Wide SO (derate 12.50mW/°C above +70°C) 1W
44-Pin Plastic FP (derate 11.11mW/°C above +70°C) 889mW
14-Pin CERDIP (derate 9.09mW/°C above +70°C).....727mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C).....800mW
20-Pin CERDIP (derate 11.11mW/°C above +70°C).....889mW
24-Pin Narrow CERDIP
(derate 12.50mW/°C above +70°C) 1W
24-Pin Sidebraze (derate 20.0mW/°C above +70°C).....1.6W
28-Pin SSOP (derate 9.52mW/°C above +70°C).....762mW
Operating Temperature Ranges
MAX2 _ _ C _ _.....0°C to +70°C
MAX2 _ _ E _ _.....-40°C to +85°C
MAX2 _ _ M _ _ .....-55°C to +125°C
Storage Temperature Range .....-65°C to +160°C
Lead Temperature (soldering, 10s) .....+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX223/MAX230–MAX241

(MAX223/230/232/234/236/237/238/240/241, V_{CC} = +5V ±10; MAX233/MAX235, V_{CC} = 5V ±5%, C1–C4 = 1.0µF; MAX231/MAX239, V_{CC} = 5V ±10%; V+ = 7.5V to 13.2V; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to ground		±5.0	±7.3		V
V_{CC} Power-Supply Current	No load, T_A = +25°C	MAX232/233		5	10	mA
		MAX223/230/234–238/240/241		7	15	
		MAX231/239		0.4	1	
V+ Power-Supply Current		MAX231		1.8	5	mA
		MAX239		5	15	
Shutdown Supply Current	T_A = +25°C	MAX223		15	50	µA
		MAX230/235/236/240/241		1	10	
Input Logic Threshold Low	T_{IN}; EN, $\overline{SHDN}$ (MAX233); $\overline{EN}$, SHDN (MAX230/235–241)				0.8	V
Input Logic Threshold High	T_{IN}		2.0			V
	EN, $\overline{SHDN}$ (MAX223); $\overline{EN}$, SHDN (MAX230/235/236/240/241)		2.4			
Logic Pull-Up Current	T_{IN} = 0V			1.5	200	µA
Receiver Input Voltage Operating Range			-30		30	V

+5V-Powered, Multichannel RS-232 Drivers/Receivers

ELECTRICAL CHARACTERISTICS—MAX223/MAX230–MAX241 (continued)

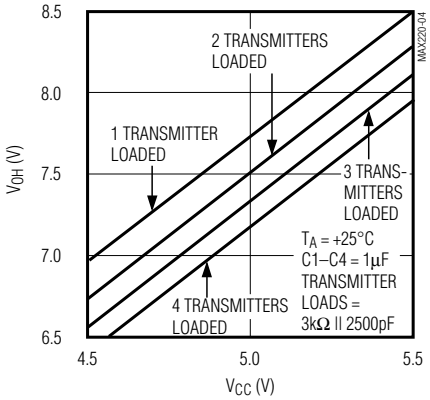
(MAX223/230/232/234/236/237/238/240/241, V_{CC} = +5V ±10; MAX233/MAX235, V_{CC} = 5V ±5%, C1–C4 = 1.0µF; MAX231/MAX239, V_{CC} = 5V ±10%; V+ = 7.5V to 13.2V; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS			MIN	TYP	MAX	UNITS
RS-232 Input Threshold Low	T_A = +25°C, V_{CC} = 5V	Normal operation $\overline{SHDN}$ = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)		0.8	1.2		V
		Shutdown (MAX223) $\overline{SHDN}$ = 0V, EN = 5V ($R4_{IN}$, $R5_{IN}$)		0.6	1.5		
RS-232 Input Threshold High	T_A = +25°C, V_{CC} = 5V	Normal operation $\overline{SHDN}$ = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)			1.7	2.4	V
		Shutdown (MAX223) $\overline{SHDN}$ = 0V, EN = 5V ($R4_{IN}$, $R5_{IN}$)			1.5	2.4	
RS-232 Input Hysteresis	V_{CC} = 5V, no hysteresis in shutdown			0.2	0.5	1.0	V
RS-232 Input Resistance	T_A = +25°C, V_{CC} = 5V			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 1.6mA (MAX231/232/233, I_{OUT} = 3.2mA)					0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1mA			3.5	V_{CC} - 0.4		V
TTL/CMOS Output Leakage Current	0V ≤ R_{OUT} ≤ V_{CC}; EN = 0V (MAX223); $\overline{EN}$ = V_{CC} (MAX235–241)				0.05	±10	µA
Receiver Output Enable Time	Normal operation	MAX223			600		ns
		MAX235/236/239/240/241			400		
Receiver Output Disable Time	Normal operation	MAX223			900		ns
		MAX235/236/239/240/241			250		
Propagation Delay	RS-232 IN to TTL/CMOS OUT, C_L = 150pF	Normal operation			0.5	10	µs
		$\overline{SHDN}$ = 0V (MAX223)	t_{PHLS}		4	40	
			t_{PLHS}		6	40	
Transition Region Slew Rate	MAX223/MAX230/MAX234–241, T_A = +25°C, V_{CC} = 5V, R_L = 3kΩ to 7kΩ, C_L = 50pF to 2500pF, measured from +3V to -3V or -3V to +3V			3	5.1	30	V/µs
	MAX231/MAX232/MAX233, T_A = +25°C, V_{CC} = 5V, R_L = 3kΩ to 7kΩ, C_L = 50pF to 2500pF, measured from +3V to -3V or -3V to +3V				4	30	
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V			300			Ω
Transmitter Output Short-Circuit Current					±10		mA

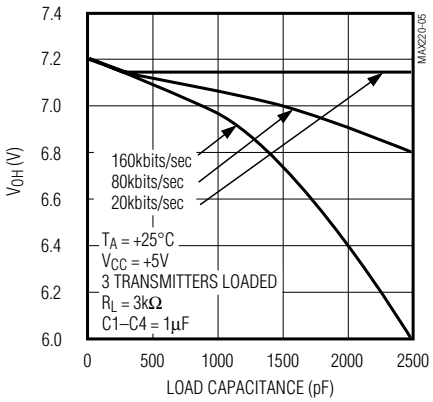
Typical Operating Characteristics

MAX223/MAX230–MAX241

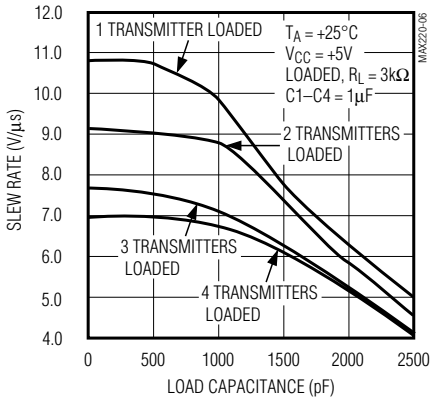
TRANSMITTER OUTPUT VOLTAGE (V_{OH}) vs. V_{CC}



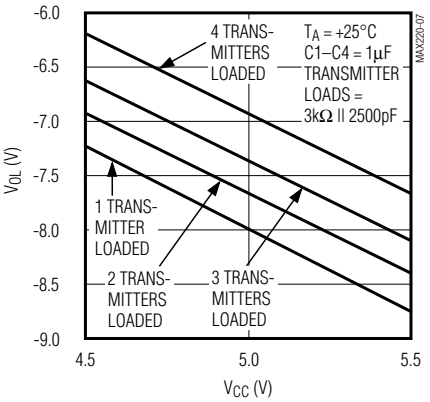
TRANSMITTER OUTPUT VOLTAGE (V_{OH}) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



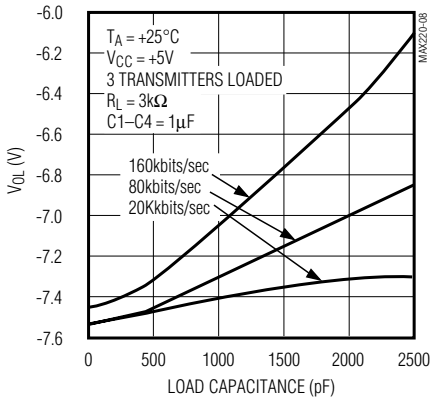
TRANSMITTER SLEW RATE vs. LOAD CAPACITANCE



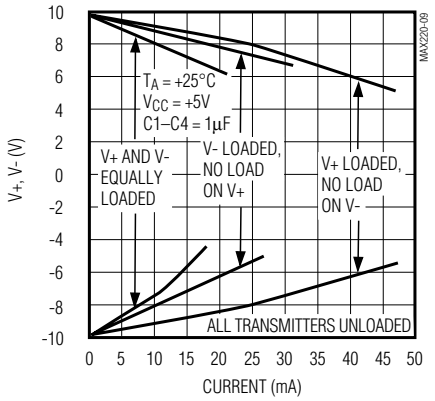
TRANSMITTER OUTPUT VOLTAGE (V_{OL}) vs. V_{CC}



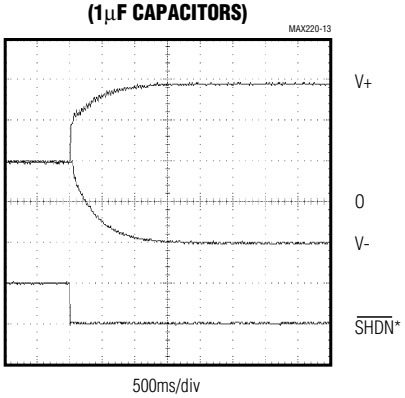
TRANSMITTER OUTPUT VOLTAGE (V_{OL}) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



TRANSMITTER OUTPUT VOLTAGE (V+, V-) vs. LOAD CURRENT



V+, V- WHEN EXITING SHUTDOWN (1µF CAPACITORS)



*SHUTDOWN POLARITY IS REVERSED FOR NON MAX241 PARTS

+5V-Powered, Multichannel RS-232 Drivers/Receivers

ABSOLUTE MAXIMUM RATINGS—MAX225/MAX244–MAX249

Supply Voltage (V_{CC}) .....-0.3V to +6V
Input Voltages
T_{IN}, $\overline{ENA}$, $\overline{ENB}$, $\overline{ENR}$, $\overline{ENT}$, $\overline{ENRA}$, $\overline{ENRB}$, $\overline{ENTA}$, $\overline{ENTB}$.....-0.3V to (V_{CC} + 0.3V)
R_{IN} .....±25V
T_{OUT} (Note 3).....±15V
R_{OUT} .....-0.3V to (V_{CC} + 0.3V)
Short Circuit (one output at a time)
T_{OUT} to GND .....Continuous
R_{OUT} to GND.....Continuous
Continuous Power Dissipation (T_A = +70°C)
28-Pin Wide SO (derate 12.50mW/°C above +70°C) 1W
40-Pin Plastic DIP (derate 11.11mW/°C above +70°C) ...611mW
44-Pin PLCC (derate 13.33mW/°C above +70°C) 1.07W
Operating Temperature Ranges
MAX225C_ _, MAX24_C_ _ .....0°C to +70°C
MAX225E_ _, MAX24_E_ _ .....-40°C to +85°C
Storage Temperature Range .....-65°C to +160°C
Lead Temperature (soldering,10s) .....+300°C

Note 4: Input voltage measured with transmitter output in a high-impedance state, shutdown, or V_{CC} = 0V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX225/MAX244–MAX249

(MAX225, V_{CC} = 5.0V ±5%; MAX244–MAX249, V_{CC} = +5.0V ±10%, external capacitors C1–C4 = 1µF; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High			2	1.4		V
Logic Pull-Up/Input Current	Tables 1a–1d	Normal operation		10	50	µA
		Shutdown		±0.01	±1	
Data Rate	Tables 1a–1d, normal operation			120	64	kbps
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±7.5		V
Output Leakage Current (Shutdown)	Tables 1a–1d	$\overline{ENA}$, $\overline{ENB}$, $\overline{ENT}$, $\overline{ENTA}$, $\overline{ENTB}$ = V_{CC}, V_{OUT} = ±15V		±0.01	±25	µA
		V_{CC} = 0V, V_{OUT} = ±15V		±0.01	±25	
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V (Note 4)		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V		±7	±30		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±25	V
RS-232 Input Threshold Low	V_{CC} = 5V		0.8	1.3		V
RS-232 Input Threshold High	V_{CC} = 5V			1.8	2.4	V
RS-232 Input Hysteresis	V_{CC} = 5V		0.2	0.5	1.0	V
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		
TTL/CMOS Output Leakage Current	Normal operation, outputs disabled, Tables 1a–1d, 0V ≤ V_{OUT} ≤ V_{CC}, $\overline{ENR\_}$ = V_{CC}			±0.05	±0.10	µA

ELECTRICAL CHARACTERISTICS—MAX225/MAX244–MAX249 (continued)

(MAX225, V_{CC} = 5.0V ±5%; MAX244–MAX249, V_{CC} = +5.0V ±10%, external capacitors C1–C4 = 1µF; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
POWER SUPPLY AND CONTROL LOGIC						
Operating Supply Voltage		MAX225	4.75		5.25	V
		MAX244–MAX249	4.5		5.5	
V_{CC} Supply Current (Normal Operation)	No load	MAX225		10	20	mA
		MAX244–MAX249		11	30	
	3kΩ loads on all outputs	MAX225		40		
		MAX244–MAX249		57		
Shutdown Supply Current	T_A = +25°C			8	25	µA
	T_A = T_{MIN} to T_{MAX}				50	
Control Input	Leakage current				±1	µA
	Threshold low			1.4	0.8	V
	Threshold high		2.4	1.4		
AC CHARACTERISTICS						
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V		5	10	30	V/µs
Transmitter Propagation Delay TLL to RS-232 (Normal Operation), Figure 1	t_{PHLT}			1.3	3.5	µs
	t_{PLHT}			1.5	3.5	
Receiver Propagation Delay TLL to RS-232 (Normal Operation), Figure 2	t_{PHLR}			0.6	1.5	µs
	t_{PLHR}			0.6	1.5	
Receiver Propagation Delay TLL to RS-232 (Low-Power Mode), Figure 2	t_{PHLS}			0.6	10	µs
	t_{PLHS}			3.0	10	
Transmitter + to - Propagation Delay Difference (Normal Operation)	t_{PHLT} - t_{PLHT}			350		ns
Receiver + to - Propagation Delay Difference (Normal Operation)	t_{PHLR} - t_{PLHR}			350		ns
Receiver-Output Enable Time, Figure 3	t_{ER}			100	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}			100	500	ns
Transmitter Enable Time	t_{ET}	MAX246–MAX249 (excludes charge-pump startup)		5		µs
		MAX225/MAX245–MAX249 (includes charge-pump startup)		10		ms
Transmitter Disable Time, Figure 4	t_{DT}			100		ns

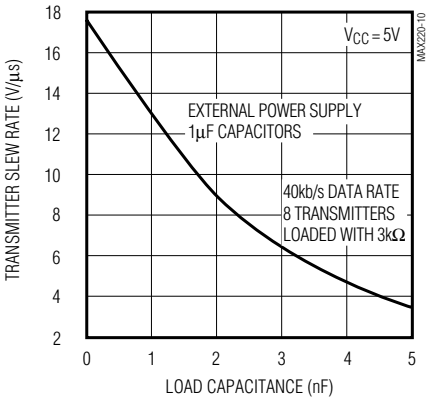
Note 5: The 300Ω minimum specification complies with EIA/TIA-232E, but the actual resistance when in shutdown mode or V_{CC} = 0V is 10MΩ as is implied by the leakage specification.

+5V-Powered, Multichannel RS-232 Drivers/Receivers

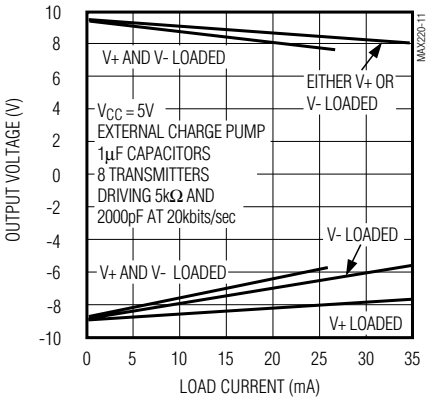
Typical Operating Characteristics

MAX225/MAX244–MAX249

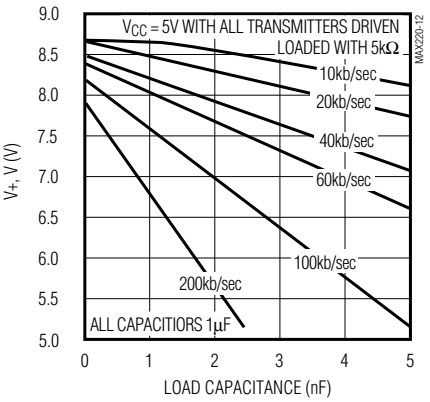
TRANSMITTER SLEW RATE vs. LOAD CAPACITANCE



OUTPUT VOLTAGE vs. LOAD CURRENT FOR V+ AND V-



TRANSMITTER OUTPUT VOLTAGE (V+, V-) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



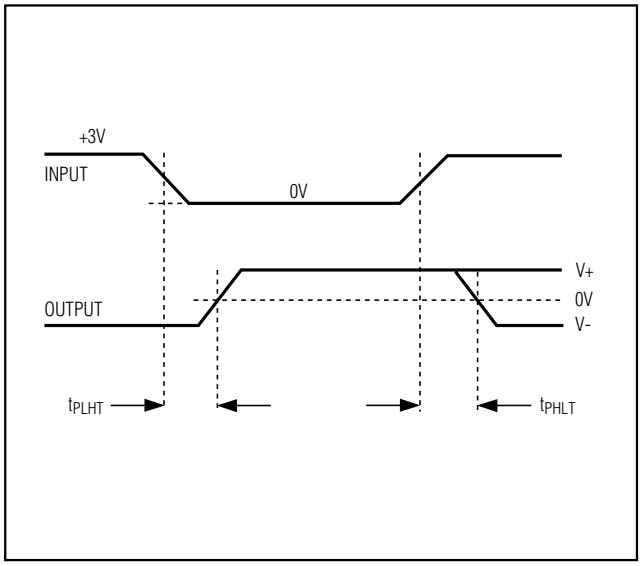


Figure 1. Transmitter Propagation-Delay Timing

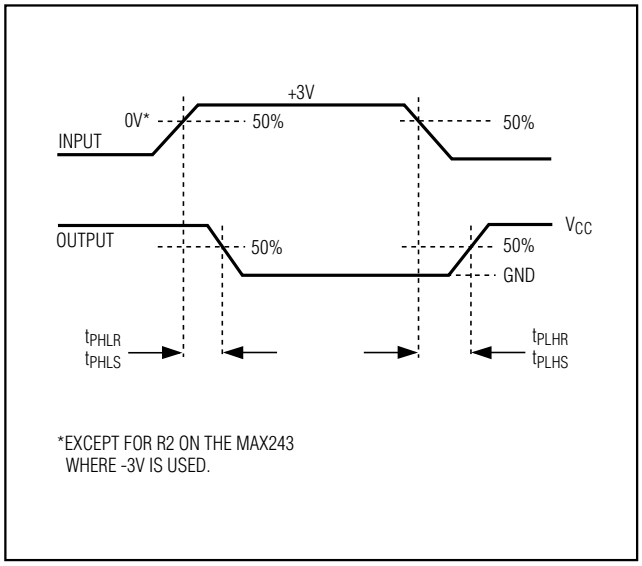


Figure 2. Receiver Propagation-Delay Timing

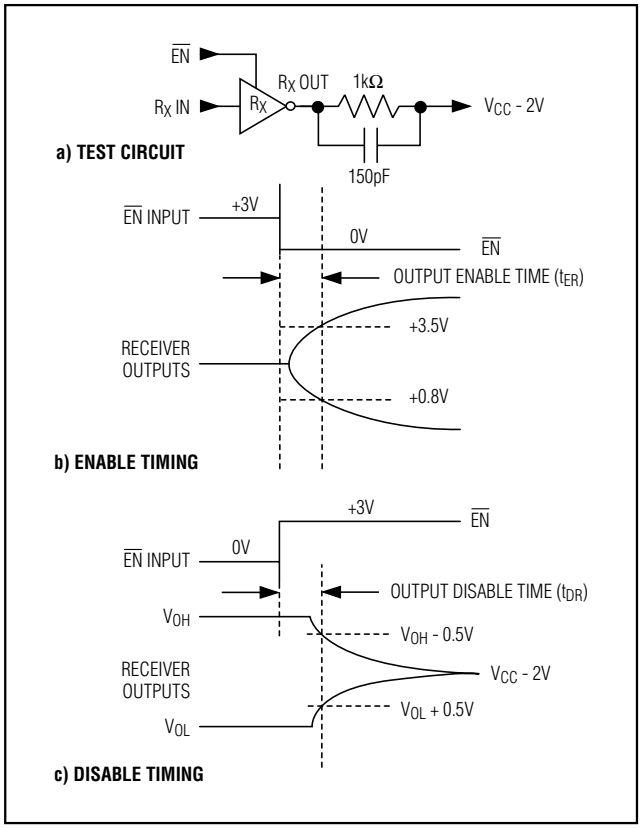


Figure 3. Receiver-Output Enable and Disable Timing

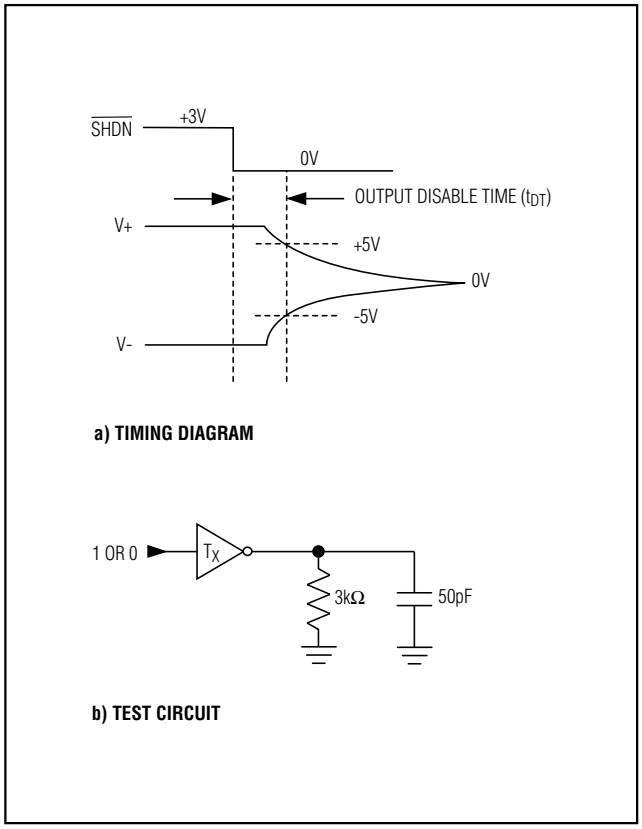


Figure 4. Transmitter-Output Disable Timing

+5V-Powered, Multichannel RS-232 Drivers/Receivers

Table 1a. MAX245 Control Pin Configurations

$\overline{ENT}$	$\overline{ENR}$	OPERATION STATUS	TRANSMITTERS	RECEIVERS
0	0	Normal Operation	All Active	All Active
0	1	Normal Operation	All Active	All 3-State
1	0	Shutdown	All 3-State	All Low-Power Receive Mode
1	1	Shutdown	All 3-State	All 3-State

Table 1b. MAX245 Control Pin Configurations

$\overline{ENT}$	$\overline{ENR}$	OPERATION STATUS	TRANSMITTERS		RECEIVERS	
			TA1–TA4	TB1–TB4	RA1–RA5	RB1–RB5
0	0	Normal Operation	All Active	All Active	All Active	All Active
0	1	Normal Operation	All Active	All Active	RA1–RA4 3-State, RA5 Active	RB1–RB4 3-State, RB5 Active
1	0	Shutdown	All 3-State	All 3-State	All Low-Power Receive Mode	All Low-Power Receive Mode
1	1	Shutdown	All 3-State	All 3-State	RA1–RA4 3-State, RA5 Low-Power Receive Mode	RB1–RB4 3-State, RB5 Low-Power Receive Mode

Table 1c. MAX246 Control Pin Configurations

$\overline{ENA}$	$\overline{ENB}$	OPERATION STATUS	TRANSMITTERS		RECEIVERS	
			TA1–TA4	TB1–TB4	RA1–RA5	RB1–RB5
0	0	Normal Operation	All Active	All Active	All Active	All Active
0	1	Normal Operation	All Active	All 3-State	All Active	RB1–RB4 3-State, RB5 Active
1	0	Shutdown	All 3-State	All Active	RA1–RA4 3-State, RA5 Active	All Active
1	1	Shutdown	All 3-State	All 3-State	RA1–RA4 3-State, RA5 Low-Power Receive Mode	RB1–RB4 3-State, RA5 Low-Power Receive Mode

Table 1d. MAX247/MAX248/MAX249 Control Pin Configurations

$\overline{ENTA}$	$\overline{ENTB}$	$\overline{ENRA}$	$\overline{ENRB}$	OPERATION STATUS		TRANSMITTERS		RECEIVERS	
					MAX247	TA1–TA4	TB1–TB4	RA1–RA4	RB1–RB5
					MAX248	TA1–TA4	TB1–TB4	RA1–RA4	RB1–RB4
					MAX249	TA1–TA3	TB1–TB3	RA1–RA5	RB1–RB5
0	0	0	0	Normal Operation		All Active	All Active	All Active	All Active
0	0	0	1	Normal Operation		All Active	All Active	All Active	All 3-State, except RB5 stays active on MAX247
0	0	1	0	Normal Operation		All Active	All Active	All 3-State	All Active
0	0	1	1	Normal Operation		All Active	All Active	All 3-State	All 3-State, except RB5 stays active on MAX247
0	1	0	0	Normal Operation		All Active	All 3-State	All Active	All Active
0	1	0	1	Normal Operation		All Active	All 3-State	All Active	All 3-State, except RB5 stays active on MAX247
0	1	1	0	Normal Operation		All Active	All 3-State	All 3-State	All Active
0	1	1	1	Normal Operation		All Active	All 3-State	All 3-State	All 3-State, except RB5 stays active on MAX247
1	0	0	0	Normal Operation		All 3-State	All Active	All Active	All Active
1	0	0	1	Normal Operation		All 3-State	All Active	All Active	All 3-State, except RB5 stays active on MAX247
1	0	1	0	Normal Operation		All 3-State	All Active	All 3-State	All Active
1	0	1	1	Normal Operation		All 3-State	All Active	All 3-State	All 3-State, except RB5 stays active on MAX247
1	1	0	0	Shutdown		All 3-State	All 3-State	Low-Power Receive Mode	Low-Power Receive Mode
1	1	0	1	Shutdown		All 3-State	All 3-State	Low-Power Receive Mode	All 3-State, except RB5 stays active on MAX247
1	1	1	0	Shutdown		All 3-State	All 3-State	All 3-State	Low-Power Receive Mode
1	1	1	1	Shutdown		All 3-State	All 3-State	All 3-State	All 3-State, except RB5 stays active on MAX247

+5V-Powered, Multichannel RS-232 Drivers/Receivers

Detailed Description

The MAX220–MAX249 contain four sections: dual charge-pump DC-DC voltage converters, RS-232 drivers, RS-232 receivers, and receiver and transmitter enable control inputs.

Dual Charge-Pump Voltage Converter

The MAX220–MAX249 have two internal charge-pumps that convert +5V to ±10V (unloaded) for RS-232 driver operation. The first converter uses capacitor C1 to double the +5V input to +10V on C3 at the V+ output. The second converter uses capacitor C2 to invert +10V to -10V on C4 at the V- output.

A small amount of power may be drawn from the +10V (V+) and -10V (V-) outputs to power external circuitry (see the *Typical Operating Characteristics* section), except on the MAX225 and MAX245–MAX247, where these pins are not available. V+ and V- are not regulated, so the output voltage drops with increasing load current. Do not load V+ and V- to a point that violates the minimum ±5V EIA/TIA-232E driver output voltage when sourcing current from V+ and V- to external circuitry.

When using the shutdown feature in the MAX222, MAX225, MAX230, MAX235, MAX236, MAX240, MAX241, and MAX245–MAX249, avoid using V+ and V- to power external circuitry. When these parts are shut down, V- falls to 0V, and V+ falls to +5V. For applications where a +10V external supply is applied to the V+ pin (instead of using the internal charge pump to generate +10V), the C1 capacitor must not be installed and the $\overline{SHDN}$ pin must be tied to V_{CC}. This is because V+ is internally connected to V_{CC} in shutdown mode.

RS-232 Drivers

The typical driver output voltage swing is ±8V when loaded with a nominal 5kΩ RS-232 receiver and V_{CC} = +5V. Output swing is guaranteed to meet the EIA/TIA-232E and V.28 specification, which calls for ±5V minimum driver output levels under worst-case conditions. These include a minimum 3kΩ load, V_{CC} = +4.5V, and maximum operating temperature. Unloaded driver output voltage ranges from (V+ -1.3V) to (V- +0.5V).

Input thresholds are both TTL and CMOS compatible. The inputs of unused drivers can be left unconnected since 400kΩ input pull-up resistors to V_{CC} are built in (except for the MAX220). The pull-up resistors force the outputs of unused drivers low because all drivers invert. The internal input pull-up resistors typically source 12µA, except in shutdown mode where the pull-ups are disabled. Driver outputs turn off and enter a high-impedance state—where leakage current is typically microamperes (maximum 25µA)—when in shutdown mode, in three-state mode, or when device power is removed. Outputs can be driven to ±15V. The power-supply current typically drops to 8µA in shutdown mode. The MAX220 does not have pull-up resistors to force the outputs of the unused drivers low. Connect unused inputs to GND or V_{CC}.

The MAX239 has a receiver three-state control line, and the MAX223, MAX225, MAX235, MAX236, MAX240, and MAX241 have both a receiver three-state control line and a low-power shutdown control. Table 2 shows the effects of the shutdown control and receiver three-state control on the receiver outputs.

The receiver TTL/CMOS outputs are in a high-impedance, three-state mode whenever the three-state enable line is high (for the MAX225/MAX235/MAX236/MAX239–MAX241), and are also high-impedance whenever the shutdown control line is high.

When in low-power shutdown mode, the driver outputs are turned off and their leakage current is less than 1µA with the driver output pulled to ground. The driver output leakage remains less than 1µA, even if the transmitter output is backdriven between 0V and (V_{CC} + 6V). Below -0.5V, the transmitter is diode clamped to ground with 1kΩ series impedance. The transmitter is also zener clamped to approximately V_{CC} + 6V, with a series impedance of 1kΩ.

The driver output slew rate is limited to less than 30V/µs as required by the EIA/TIA-232E and V.28 specifications. Typical slew rates are 24V/µs unloaded and 10V/µs loaded with 3Ω and 2500pF.

RS-232 Receivers

EIA/TIA-232E and V.28 specifications define a voltage level greater than 3V as a logic 0, so all receivers invert. Input thresholds are set at 0.8V and 2.4V, so receivers respond to TTL level inputs as well as EIA/TIA-232E and V.28 levels.

The receiver inputs withstand an input overvoltage up to ±25V and provide input terminating resistors with

Table 2. Three-State Control of Receivers

PART	SHDN	$\overline{SHDN}$	EN	$\overline{EN(R)}$	RECEIVERS
MAX223	—	Low High High	X Low High	—	High Impedance Active High Impedance
MAX225	—	—	—	Low High	High Impedance Active
MAX235 MAX236 MAX240	Low Low High	—	—	Low High X	High Impedance Active High Impedance

nominal 5kΩ values. The receivers implement Type 1 interpretation of the fault conditions of V.28 and EIA/TIA-232E.

The receiver input hysteresis is typically 0.5V with a guaranteed minimum of 0.2V. This produces clear output transitions with slow-moving input signals, even with moderate amounts of noise and ringing. The receiver propagation delay is typically 600ns and is independent of input swing direction.

Low-Power Receive Mode

The low-power receive-mode feature of the MAX223, MAX242, and MAX245–MAX249 puts the IC into shutdown mode but still allows it to receive information. This is important for applications where systems are periodically awakened to look for activity. Using low-power receive mode, the system can still receive a signal that will activate it on command and prepare it for communication at faster data rates. This operation conserves system power.

Negative Threshold—MAX243

The MAX243 is pin compatible with the MAX232A, differing only in that RS-232 cable fault protection is removed on one of the two receiver inputs. This means that control lines such as CTS and RTS can either be driven or left floating without interrupting communication. Different cables are not needed to interface with different pieces of equipment.

The input threshold of the receiver without cable fault protection is -0.8V rather than +1.4V. Its output goes positive only if the input is connected to a control line that is actively driven negative. If not driven, it defaults to the 0 or "OK to send" state. Normally, the MAX243's other receiver (+1.4V threshold) is used for the data line (TD or RD), while the negative threshold receiver is connected to the control line (DTR, DTS, CTS, RTS, etc.).

Other members of the RS-232 family implement the optional cable fault protection as specified by EIA/TIA-232E specifications. This means a receiver output goes high whenever its input is driven negative, left floating, or shorted to ground. The high output tells the serial communications IC to stop sending data. To avoid this, the control lines must either be driven or connected with jumpers to an appropriate positive voltage level.

Shutdown—MAX222–MAX242

On the MAX222, MAX235, MAX236, MAX240, and MAX241, all receivers are disabled during shutdown. On the MAX223 and MAX242, two receivers continue to operate in a reduced power mode when the chip is in shutdown. Under these conditions, the propagation delay increases to about 2.5µs for a high-to-low input transition. When in shutdown, the receiver acts as a CMOS inverter with no hysteresis. The MAX223 and MAX242 also have a receiver output enable input ($\overline{EN}$ for the MAX242 and EN for the MAX223) that allows receiver output control independent of $\overline{SHDN}$ (SHDN for MAX241). With all other devices, $\overline{SHDN}$ (SHDN for MAX241) also disables the receiver outputs.

The MAX225 provides five transmitters and five receivers, while the MAX245 provides ten receivers and eight transmitters. Both devices have separate receiver and transmitter-enable controls. The charge pumps turn off and the devices shut down when a logic high is applied to the ENT input. In this state, the supply current drops to less than 25µA and the receivers continue to operate in a low-power receive mode. Driver outputs enter a high-impedance state (three-state mode). On the MAX225, all five receivers are controlled by the $\overline{ENR}$ input. On the MAX245, eight of the receiver outputs are controlled by the $\overline{ENR}$ input, while the remaining two receivers (RA5 and RB5) are always active. RA1–RA4 and RB1–RB4 are put in a three-state mode when $\overline{ENR}$ is a logic high.

Receiver and Transmitter Enable Control Inputs

The MAX225 and MAX245–MAX249 feature transmitter and receiver enable controls.

The receivers have three modes of operation: full-speed receive (normal active), three-state (disabled), and low-power receive (enabled receivers continue to function at lower data rates). The receiver enable inputs control the full-speed receive and three-state modes. The transmitters have two modes of operation: full-speed transmit (normal active) and three-state (disabled). The transmitter enable inputs also control the shutdown mode. The device enters shutdown mode when all transmitters are disabled. Enabled receivers function in the low-power receive mode when in shutdown.

Tables 1a–1d define the control states. The MAX244 has no control pins and is not included in these tables.

The MAX246 has ten receivers and eight drivers with two control pins, each controlling one side of the device. A logic high at the A-side control input ($\overline{ENA}$) causes the four A-side receivers and drivers to go into a three-state mode. Similarly, the B-side control input ($\overline{ENB}$) causes the four B-side drivers and receivers to go into a three-state mode. As in the MAX245, one A-side and one B-side receiver (RA5 and RB5) remain active at all times. The entire device is put into shutdown mode when both the A and B sides are disabled ($\overline{ENA}$ = $\overline{ENB}$ = +5V).

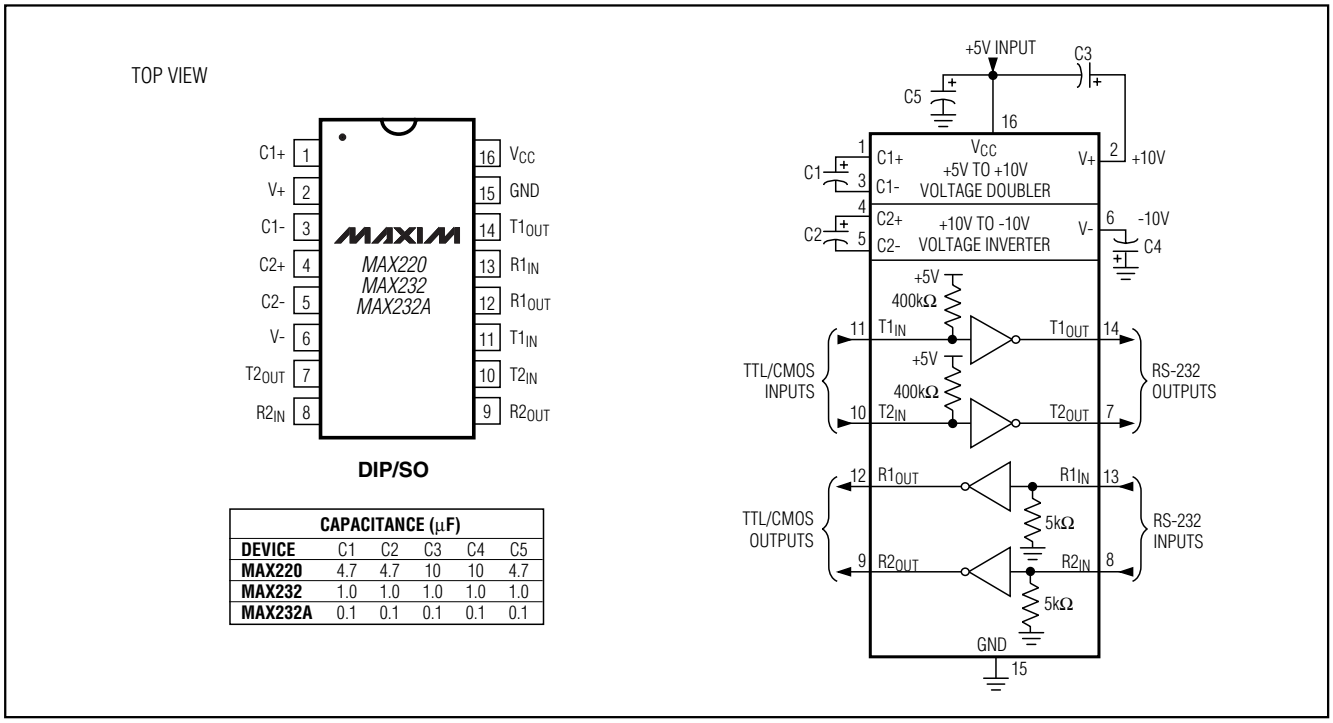
The MAX247 provides nine receivers and eight drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control four receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs each control four drivers. The ninth receiver (RB5) is always active. The device enters shutdown mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$.

The MAX248 provides eight receivers and eight drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control four receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs control four drivers each. This part does not have an always-active receiver. The device enters shutdown mode and transmitters go into a three-state mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$.

The MAX249 provides ten receivers and six drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control five receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs control three drivers each. There is no always-active receiver. The device enters shutdown mode and transmitters go into a three-state mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$. In shutdown mode, active receivers operate in a low-power receive mode at data rates up to 20kbits/sec.

Applications Information

Figures 5 through 25 show pin configurations and typical operating circuits. In applications that are sensitive to power-supply noise, V_{CC} should be decoupled to ground with a capacitor of the same value as C1 and C2 connected as close as possible to the device.



CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

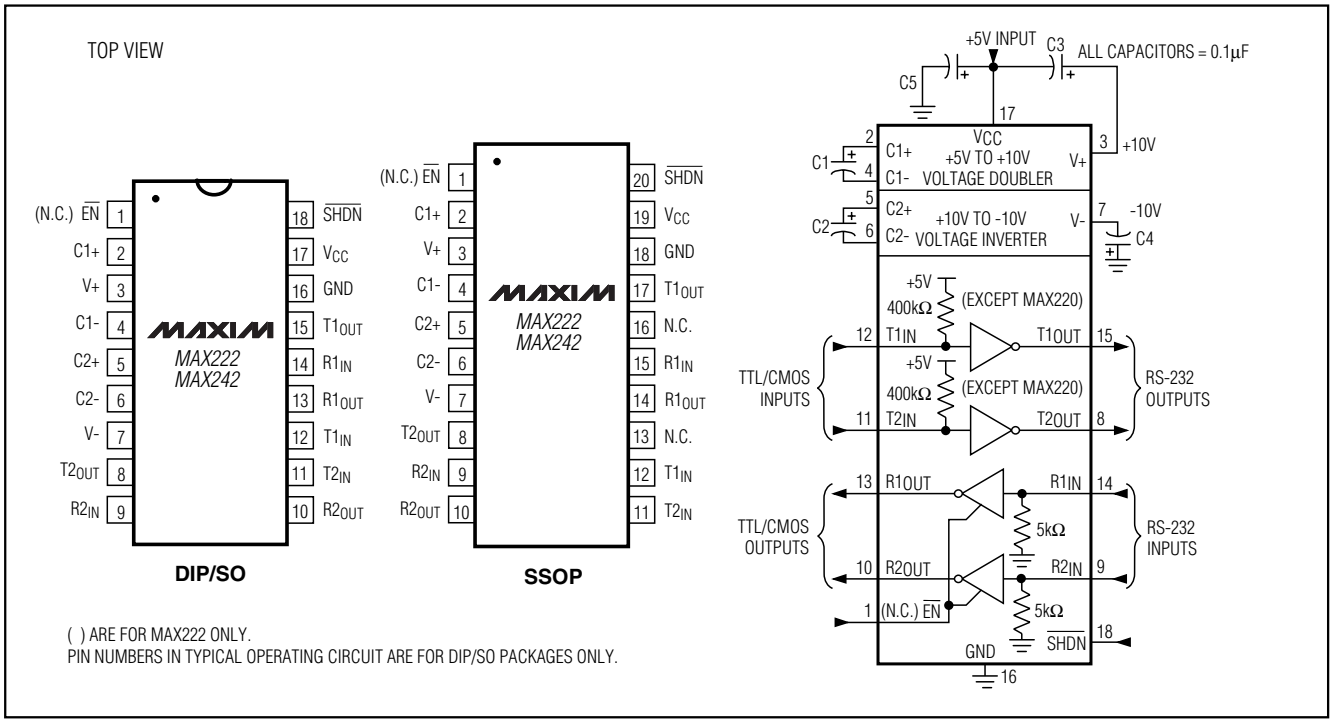


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

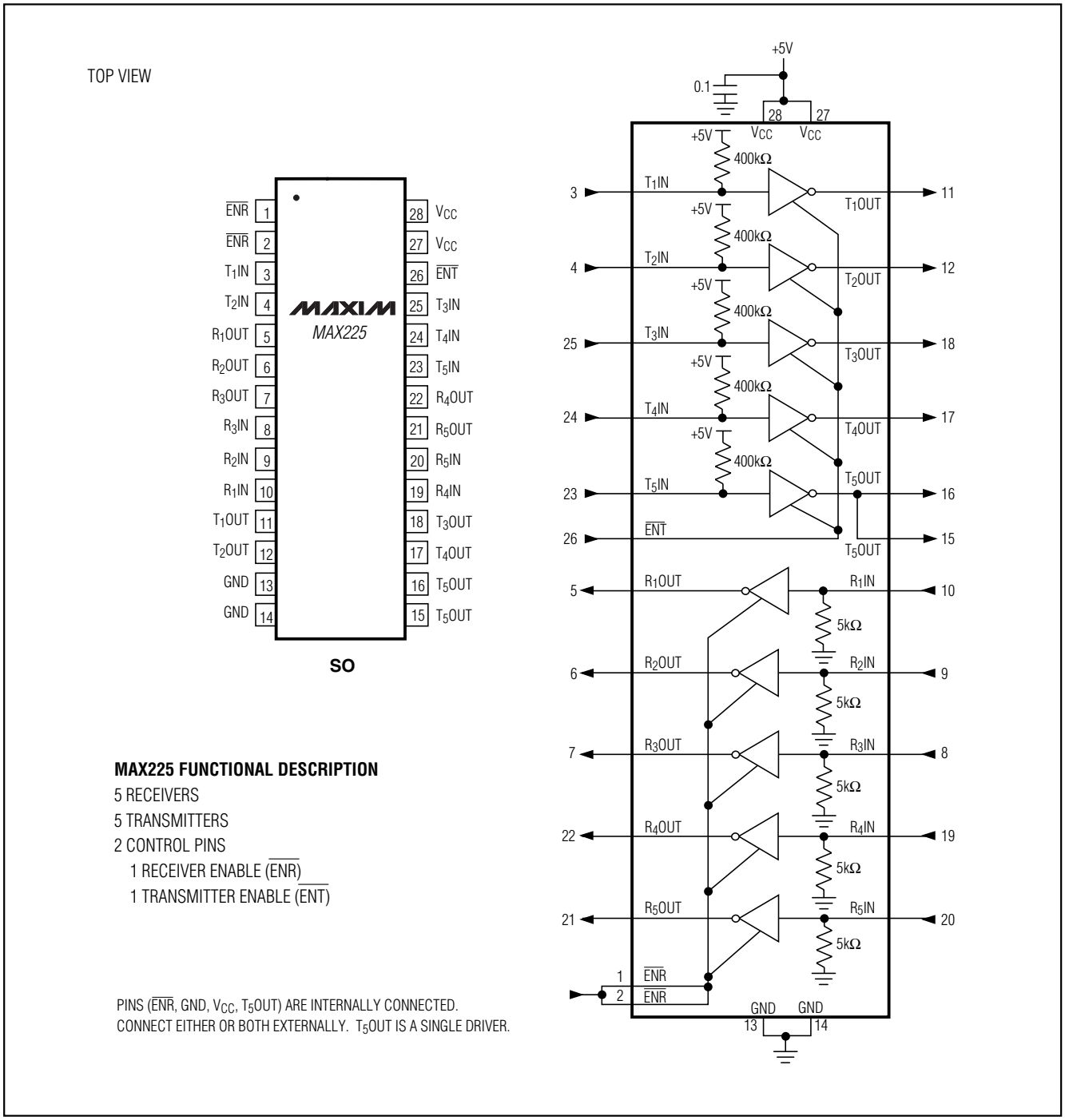


Figure 7. MAX225 Pin Configuration and Typical Operating Circuit

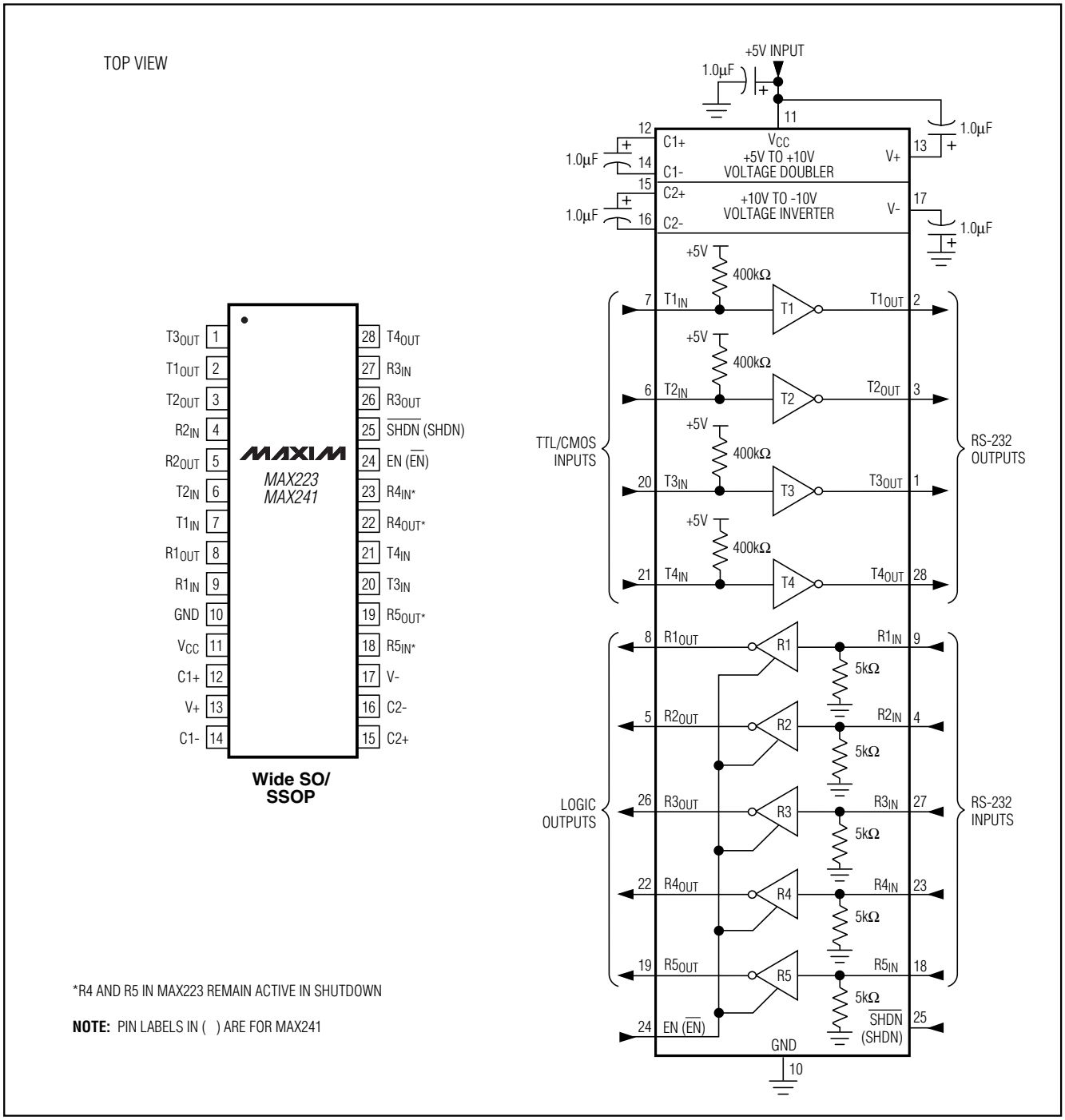


Figure 8. MAX223/MAX241 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

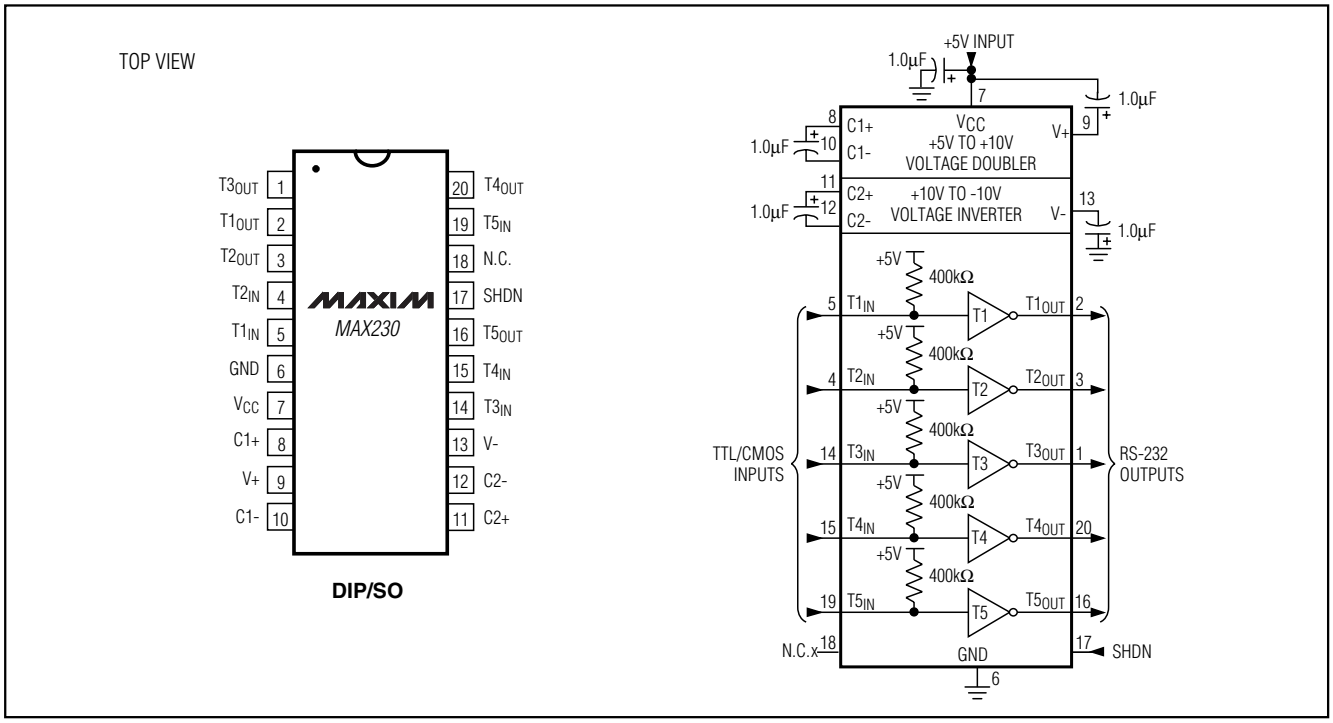


Figure 9. MAX230 Pin Configuration and Typical Operating Circuit

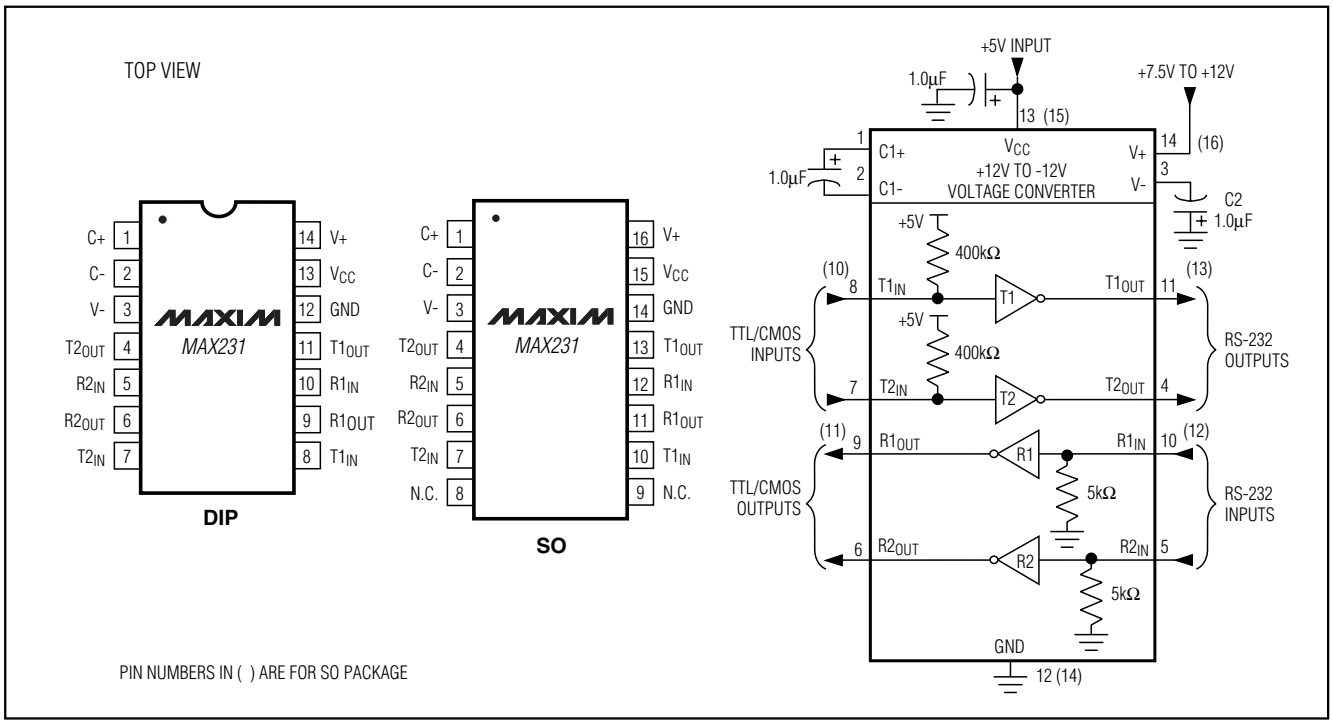


Figure 10. MAX231 Pin Configurations and Typical Operating Circuit

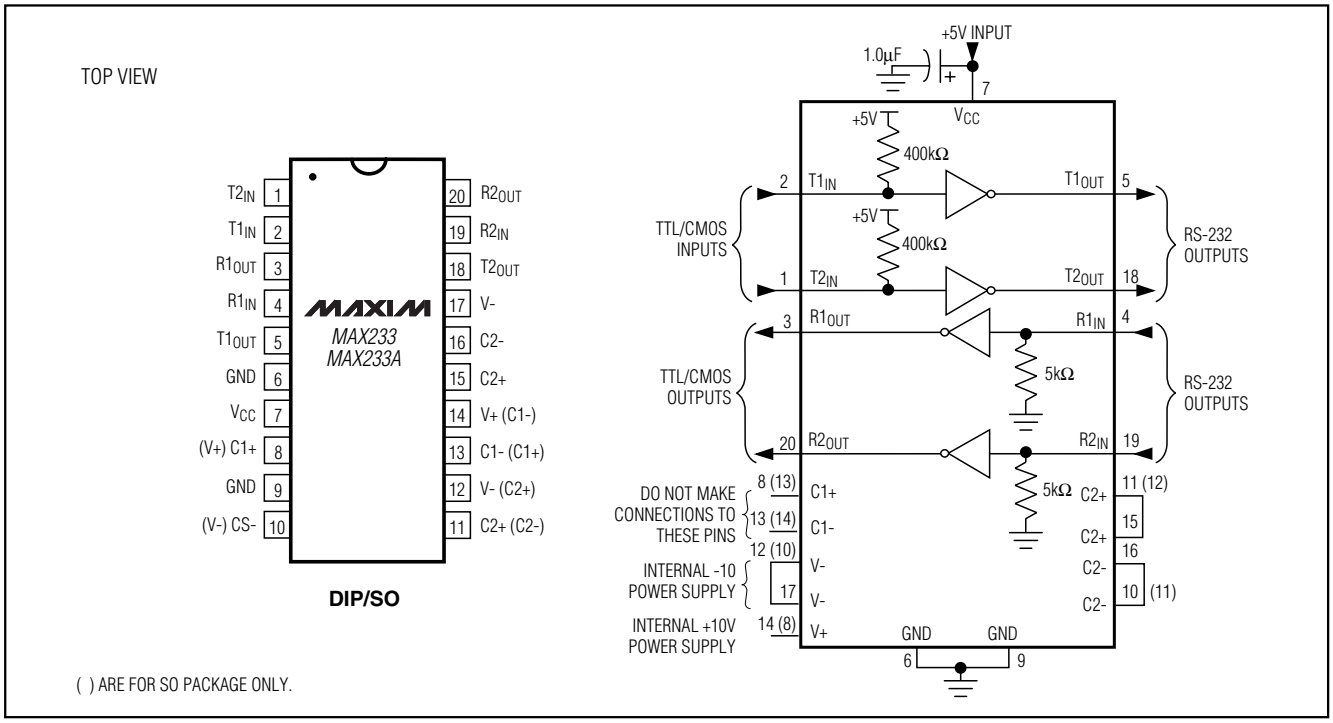


Figure 11. MAX233/MAX233A Pin Configuration and Typical Operating Circuit

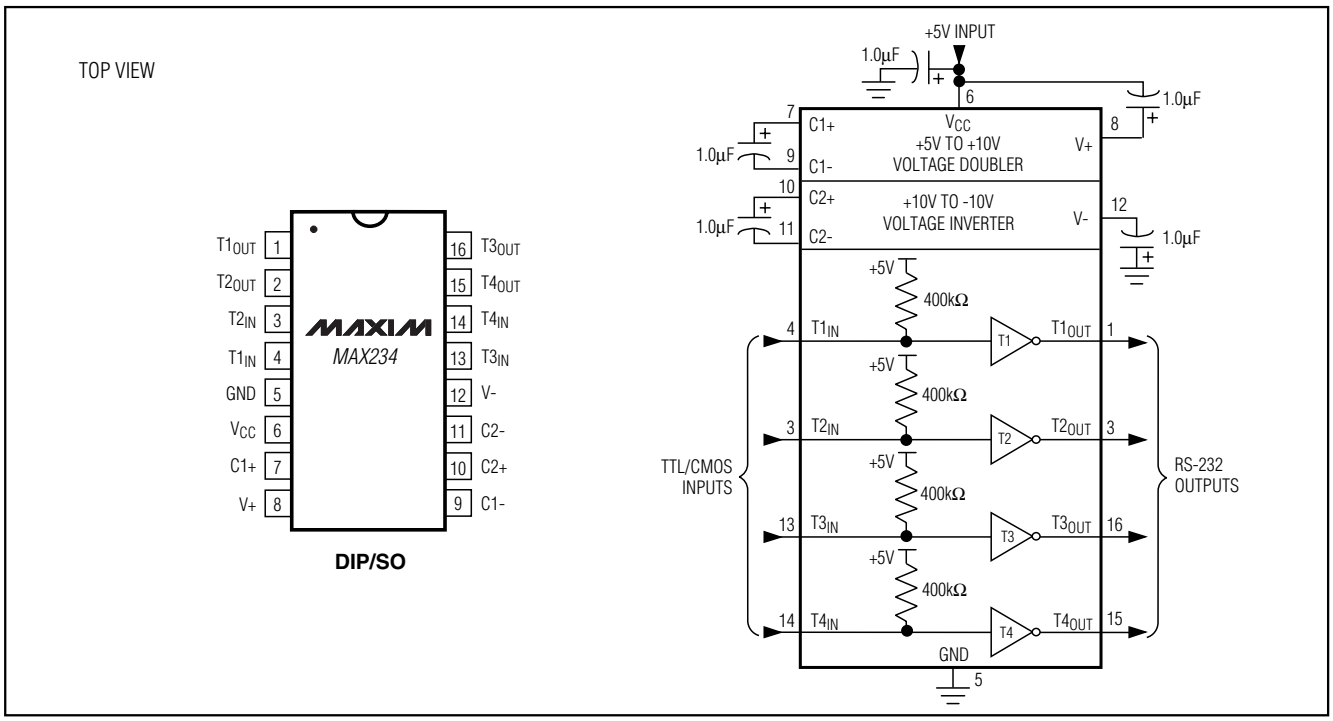


Figure 12. MAX234 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

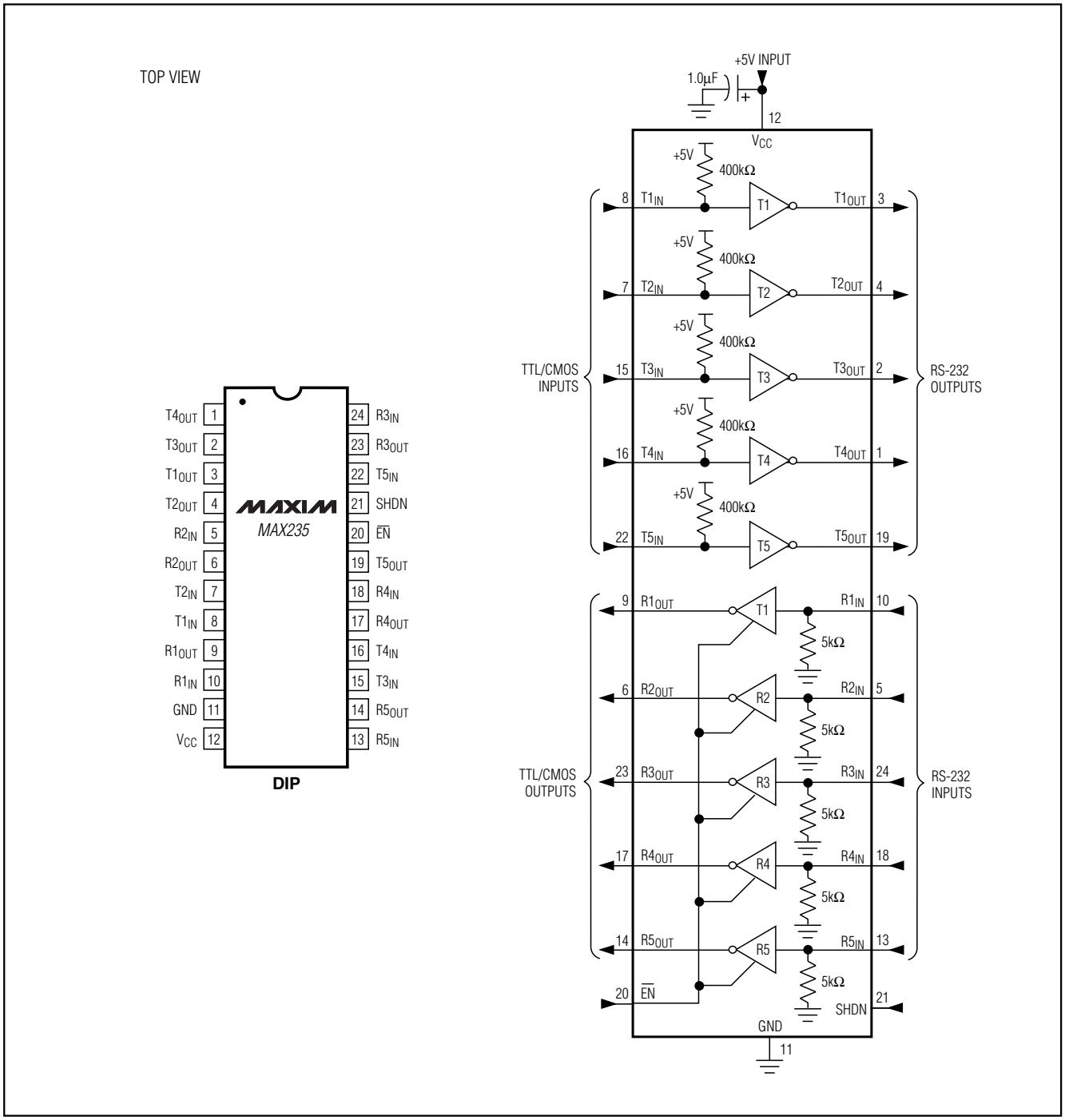


Figure 13. MAX235 Pin Configuration and Typical Operating Circuit

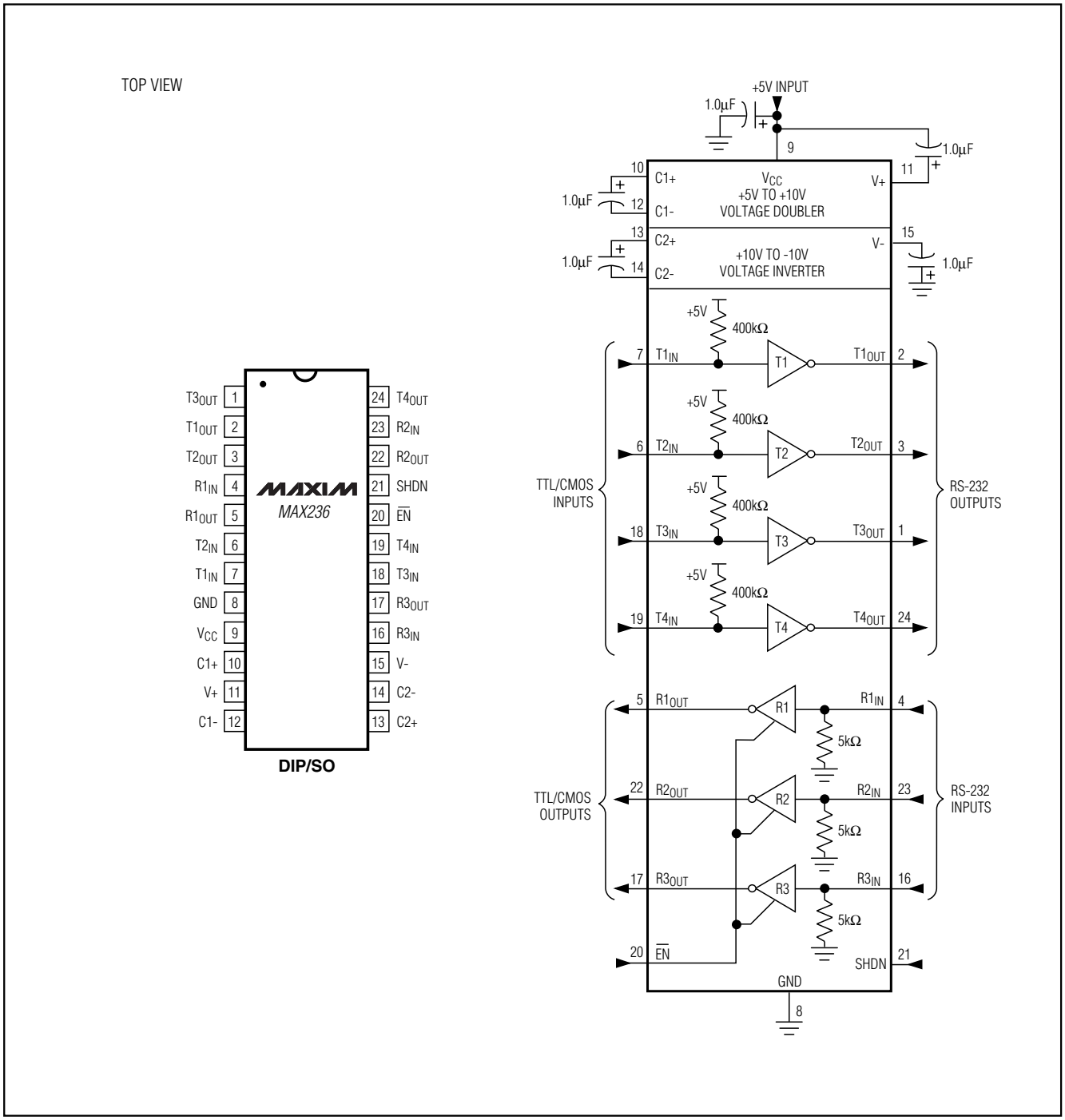


Figure 14. MAX236 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

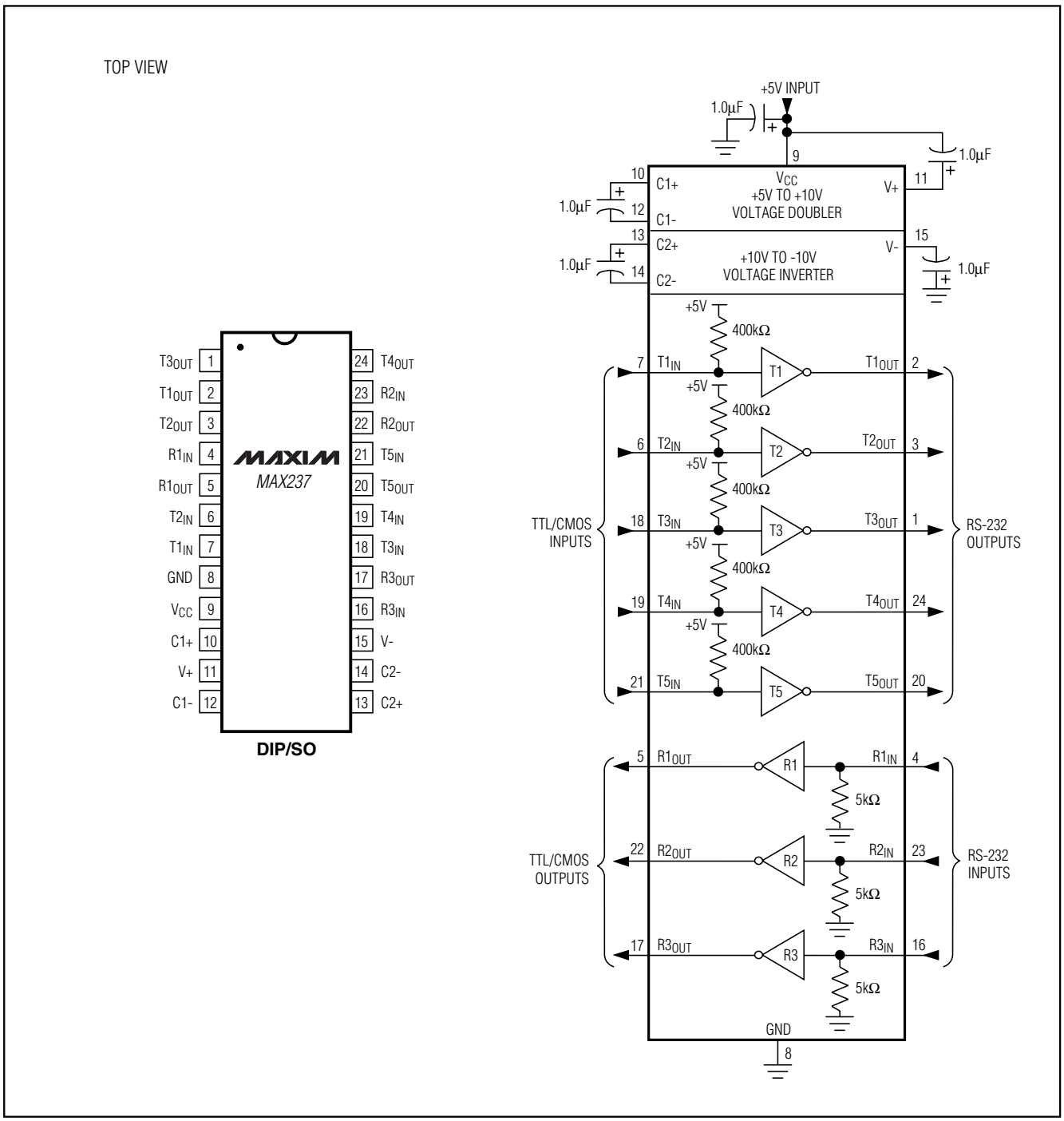


Figure 15. MAX237 Pin Configuration and Typical Operating Circuit

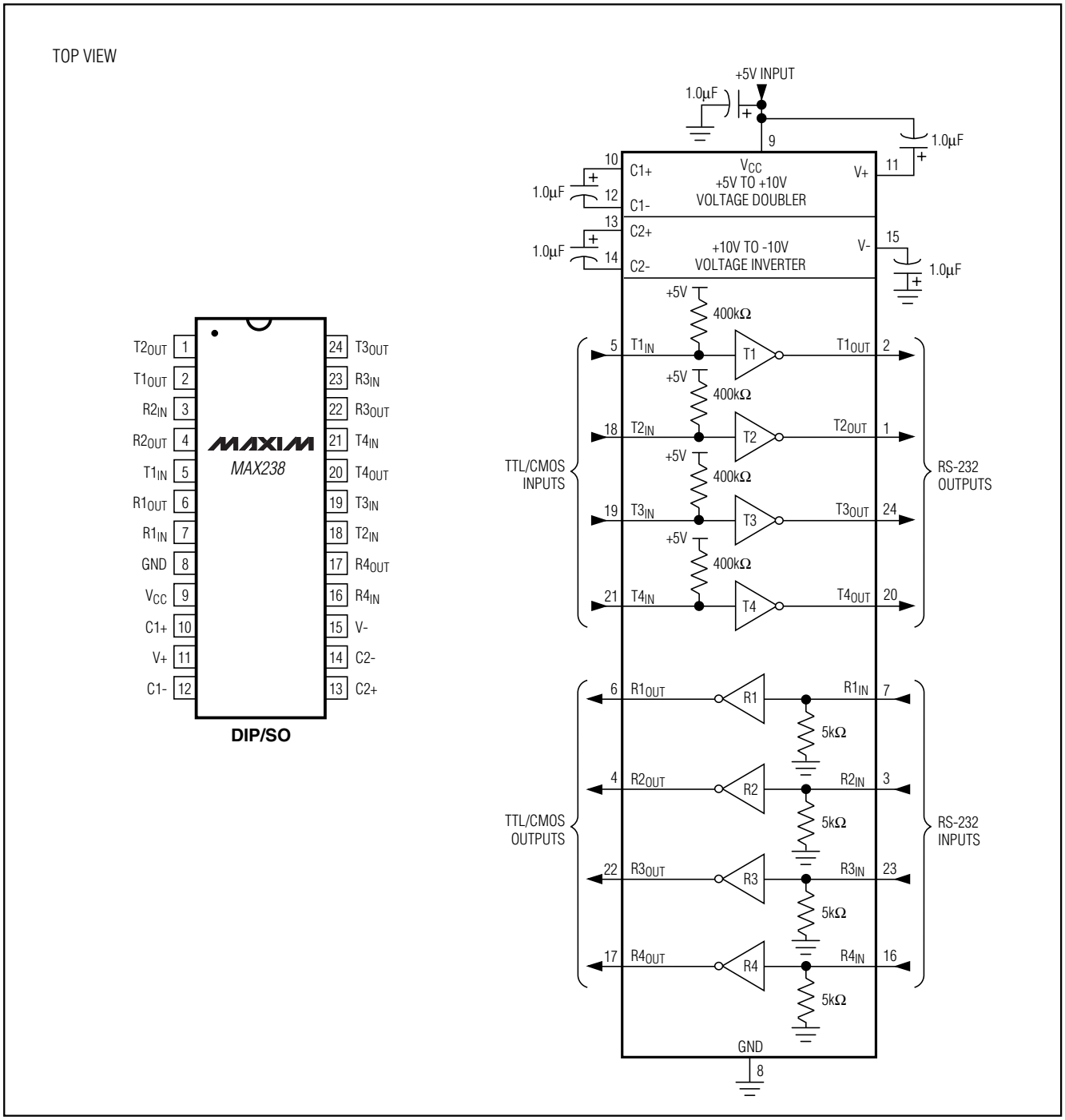


Figure 16. MAX238 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

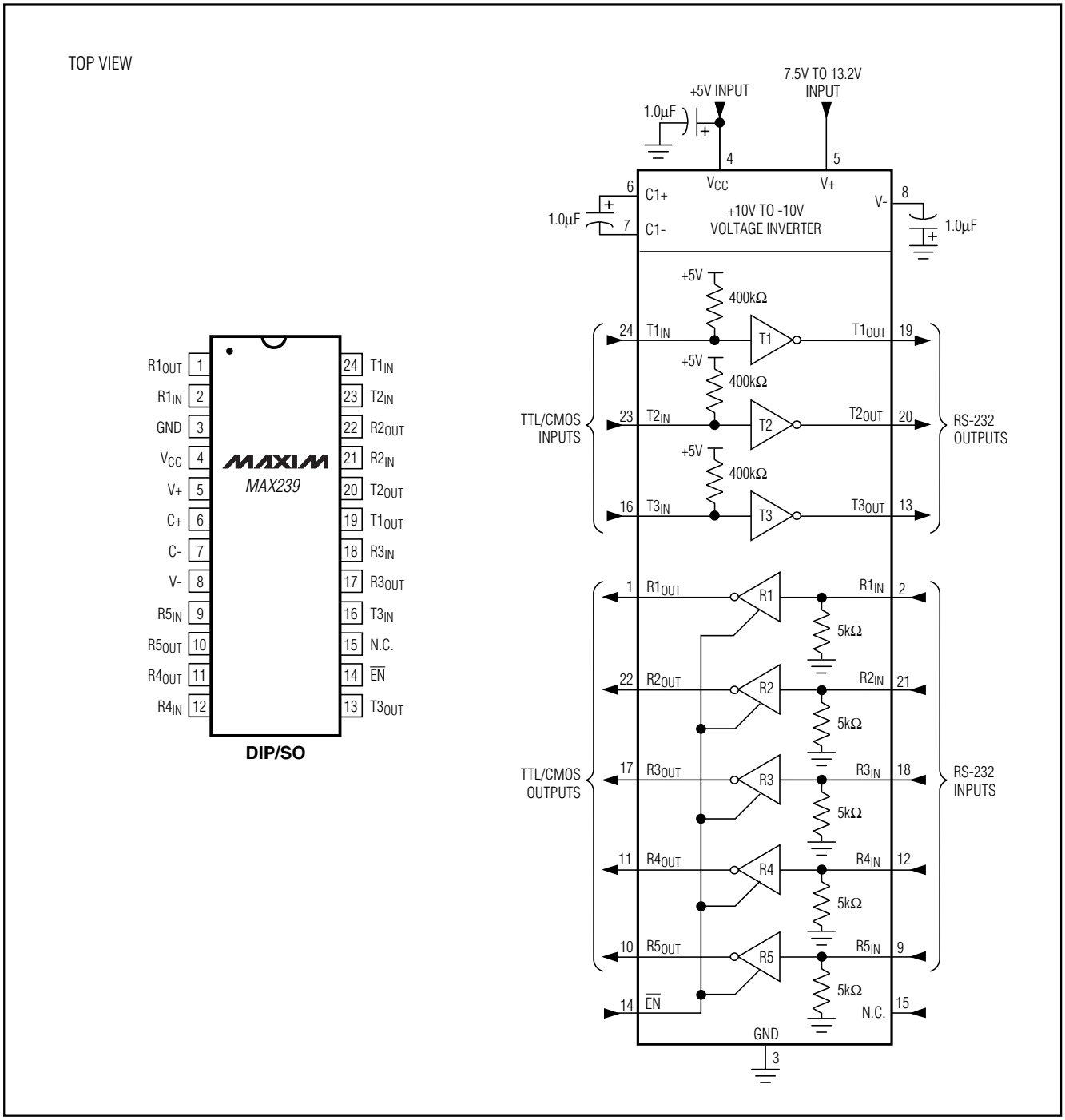


Figure 17. MAX239 Pin Configuration and Typical Operating Circuit

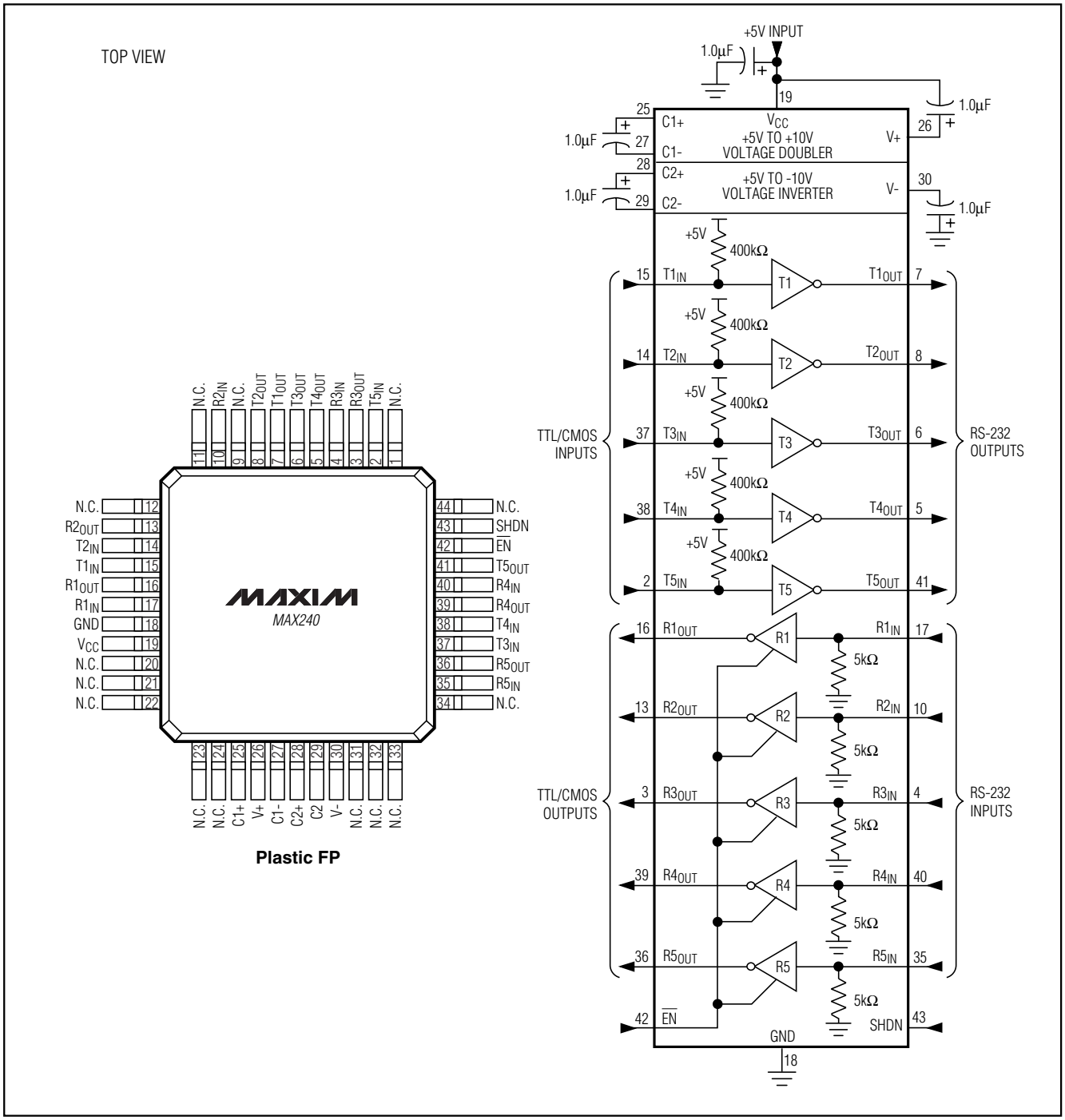
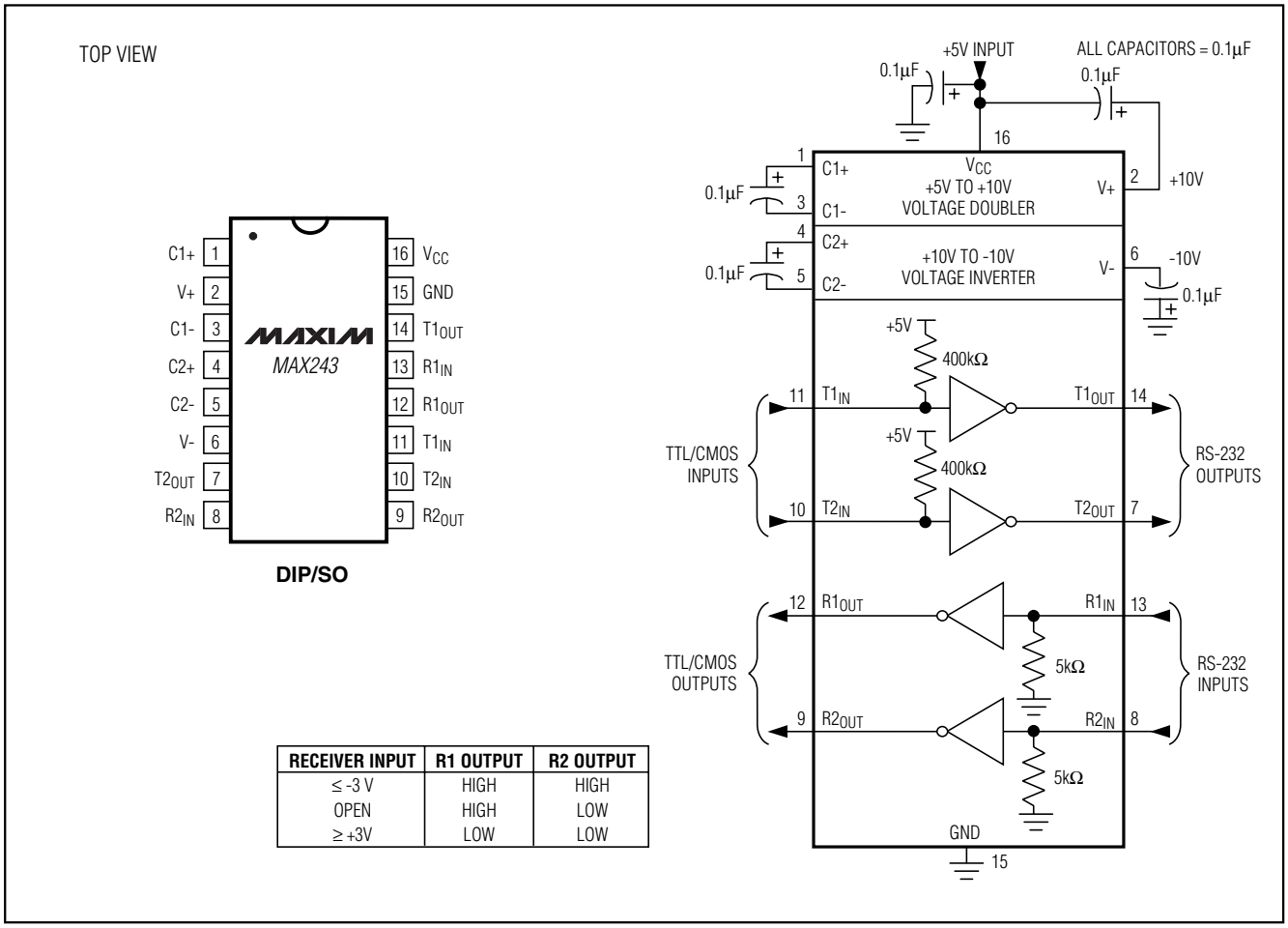


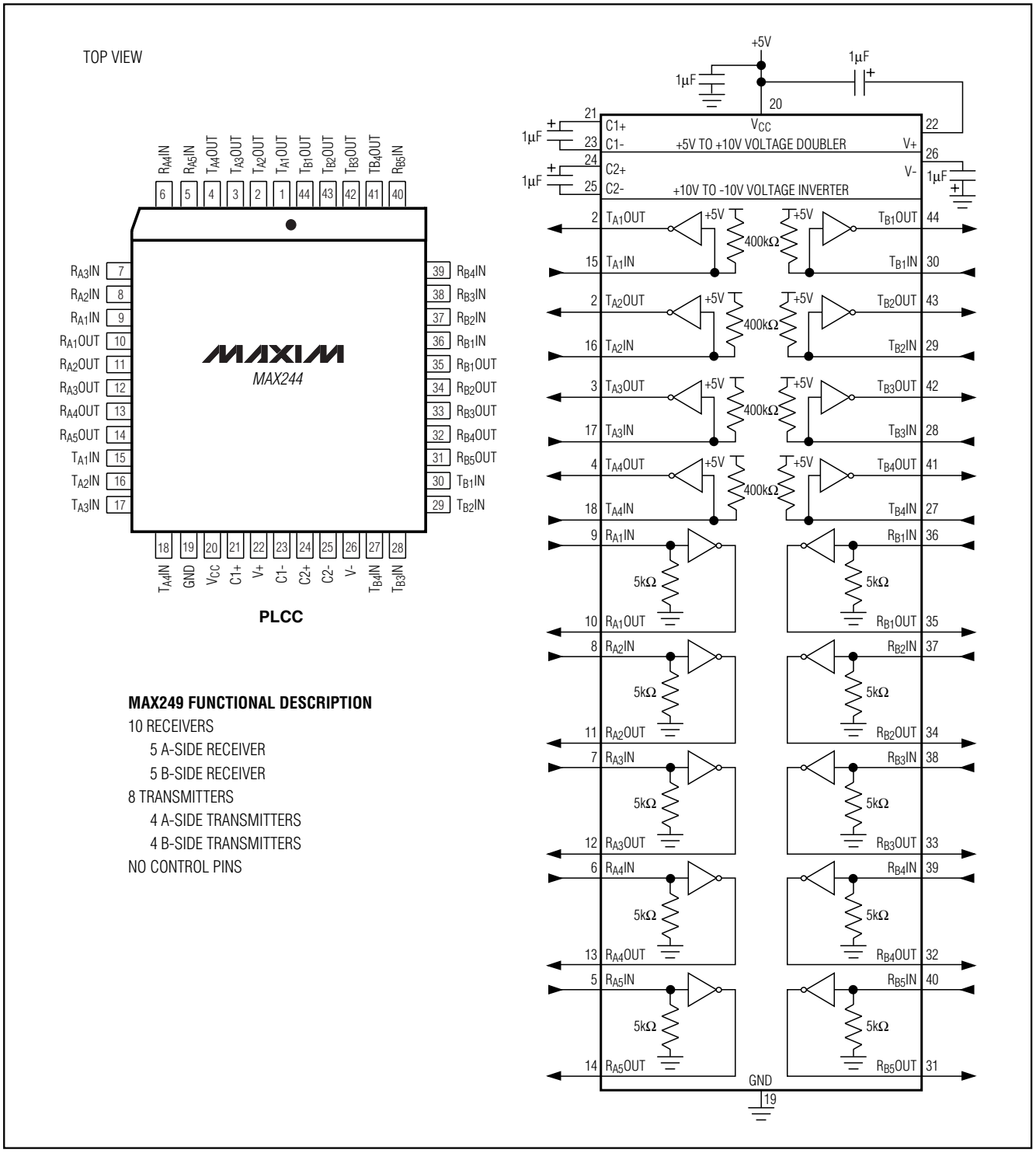
Figure 18. MAX240 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers



RECEIVER INPUT	R1 OUTPUT	R2 OUTPUT
≤ -3 V	HIGH	HIGH
OPEN	HIGH	LOW
≥ +3V	LOW	LOW

Figure 19. MAX243 Pin Configuration and Typical Operating Circuit



MAX249 FUNCTIONAL DESCRIPTION

10 RECEIVERS
- 5 A-SIDE RECEIVER
- 5 B-SIDE RECEIVER

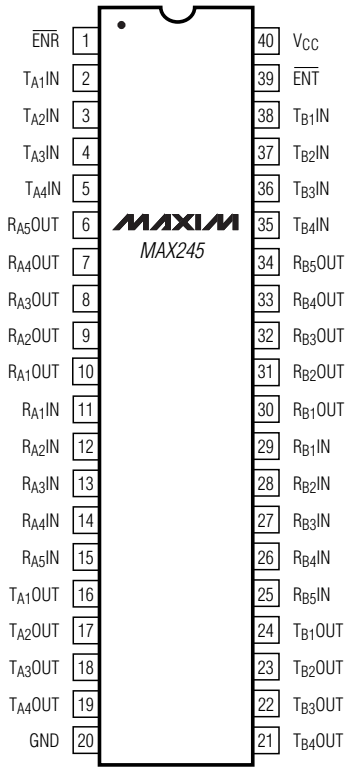
8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

NO CONTROL PINS

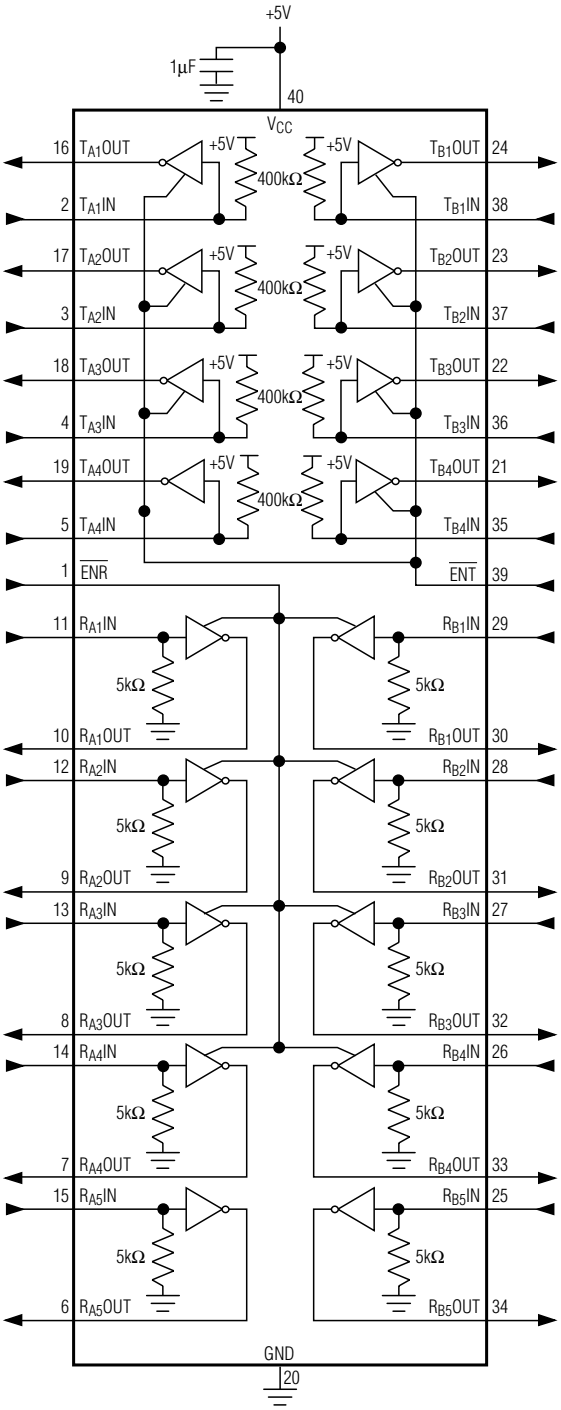
Figure 20. MAX244 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

TOP VIEW



DIP



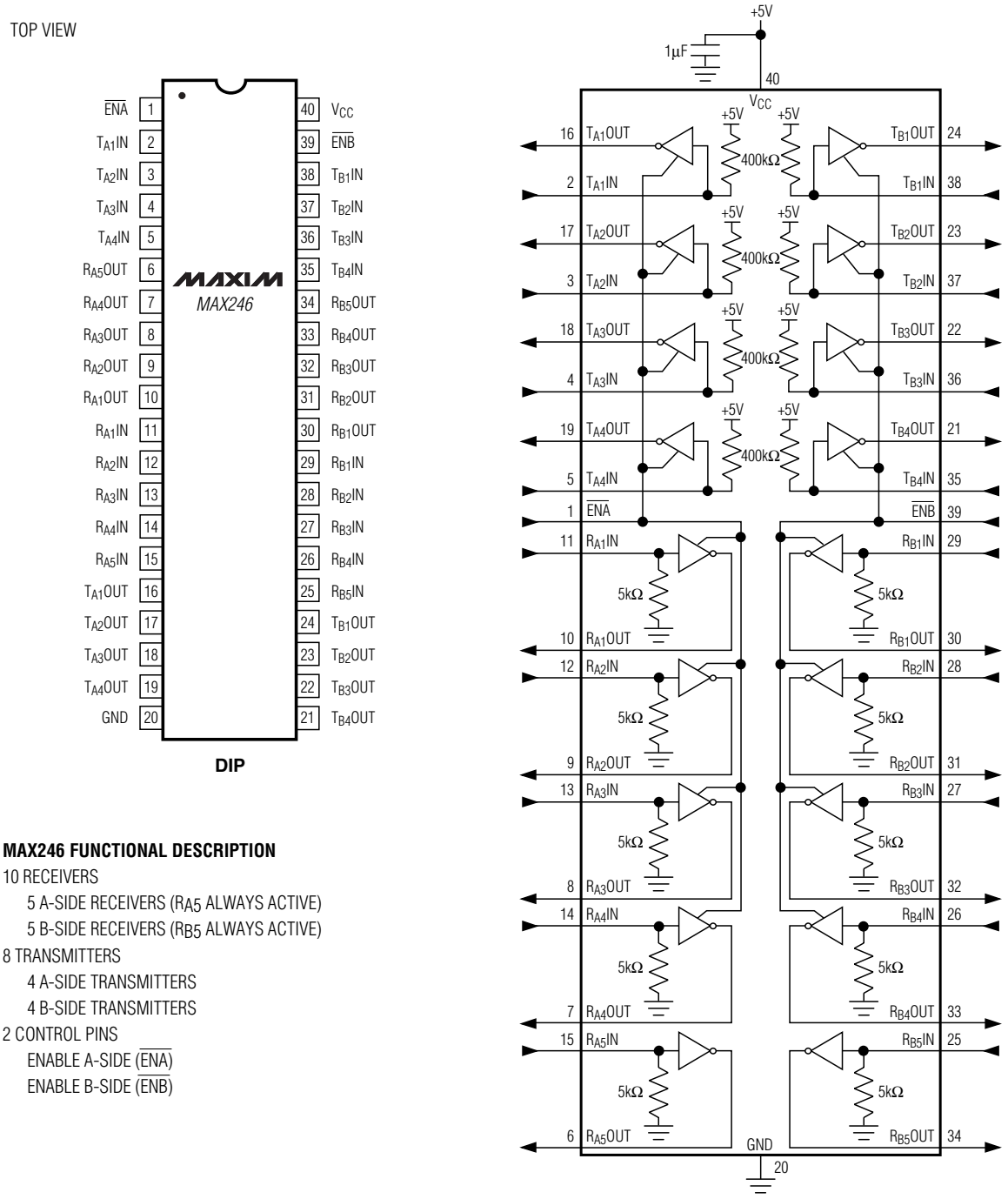
MAX245 FUNCTIONAL DESCRIPTION

10 RECEIVERS
- 5 A-SIDE RECEIVERS (R_{A5} ALWAYS ACTIVE)
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

8 TRANSMITTTERS
- 4 A-SIDE TRANSMITTERS

2 CONTROL PINS
- 1 RECEIVER ENABLE ($\overline{ENR}$)
- 1 TRANSMITTER ENABLE ($\overline{ENT}$)

Figure 21. MAX245 Pin Configuration and Typical Operating Circuit



MAX246 FUNCTIONAL DESCRIPTION

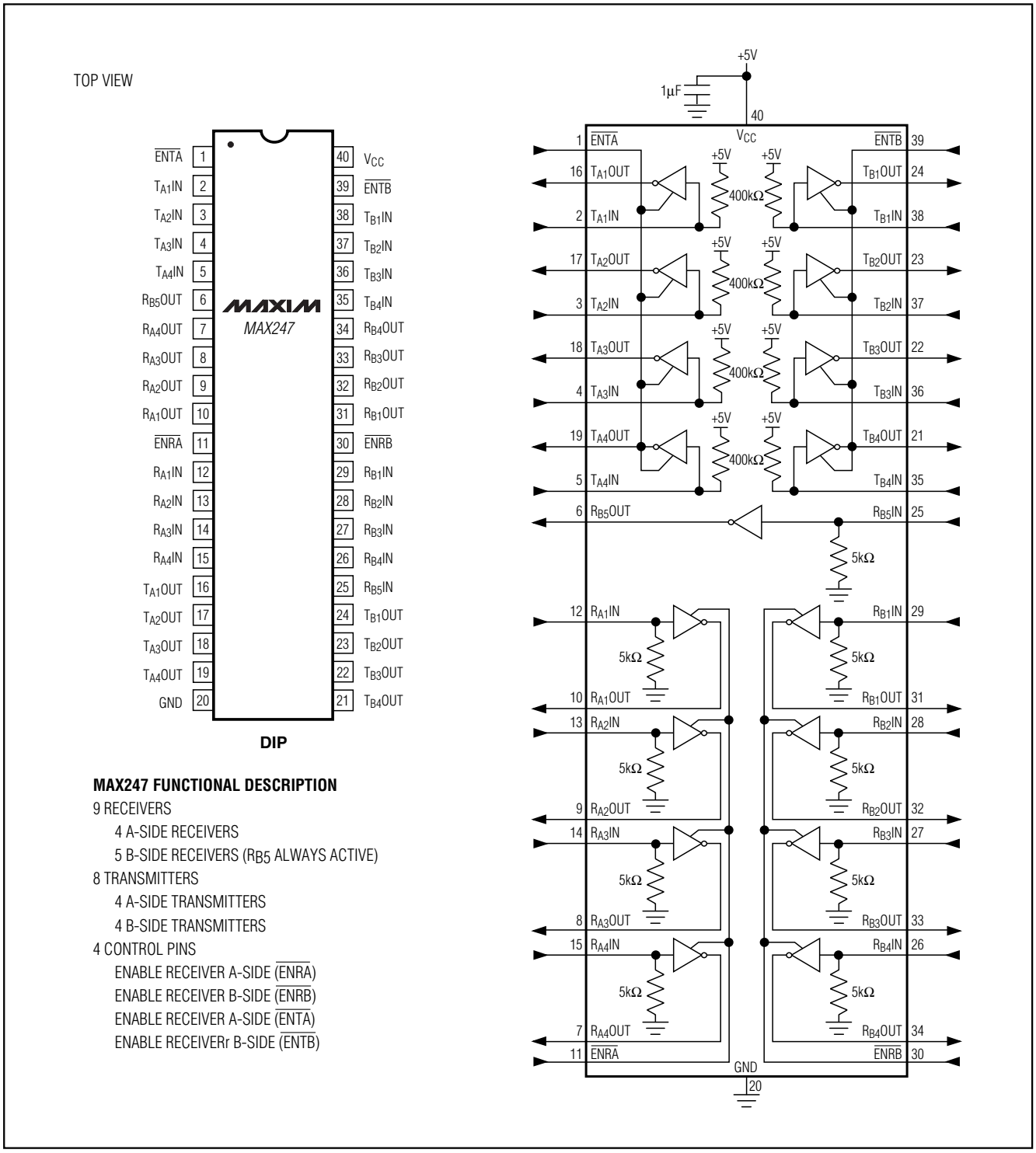
10 RECEIVERS
- 5 A-SIDE RECEIVERS (R_{A5} ALWAYS ACTIVE)
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

2 CONTROL PINS
- ENABLE A-SIDE ($\overline{ENA}$)
- ENABLE B-SIDE ($\overline{ENB}$)

Figure 22. MAX246 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers



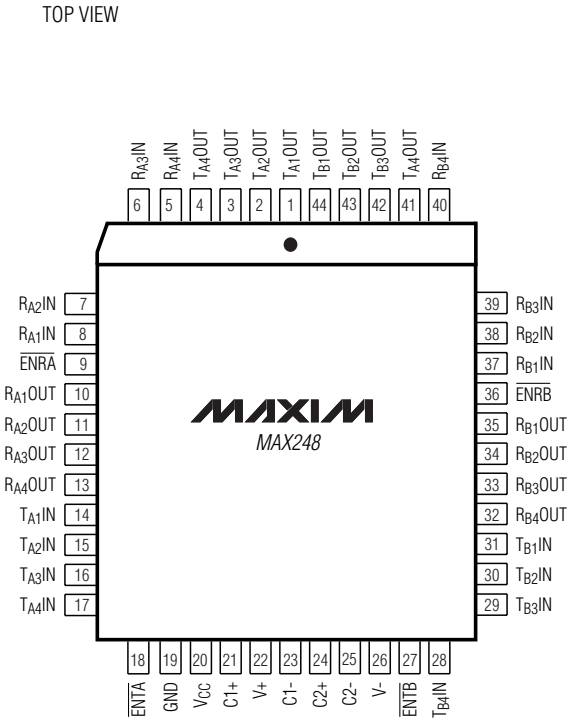
MAX247 FUNCTIONAL DESCRIPTION

9 RECEIVERS
- 4 A-SIDE RECEIVERS
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

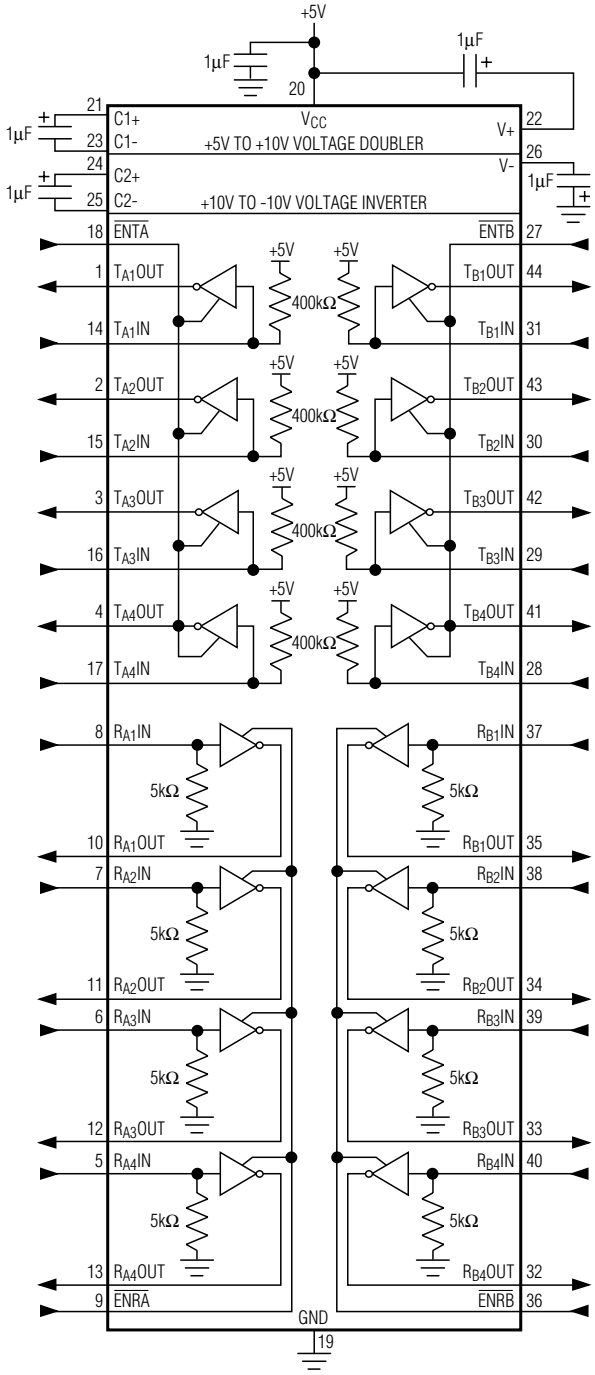
8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVERr B-SIDE ($\overline{ENTB}$)

Figure 23. MAX247 Pin Configuration and Typical Operating Circuit



PLCC



MAX248 FUNCTIONAL DESCRIPTION

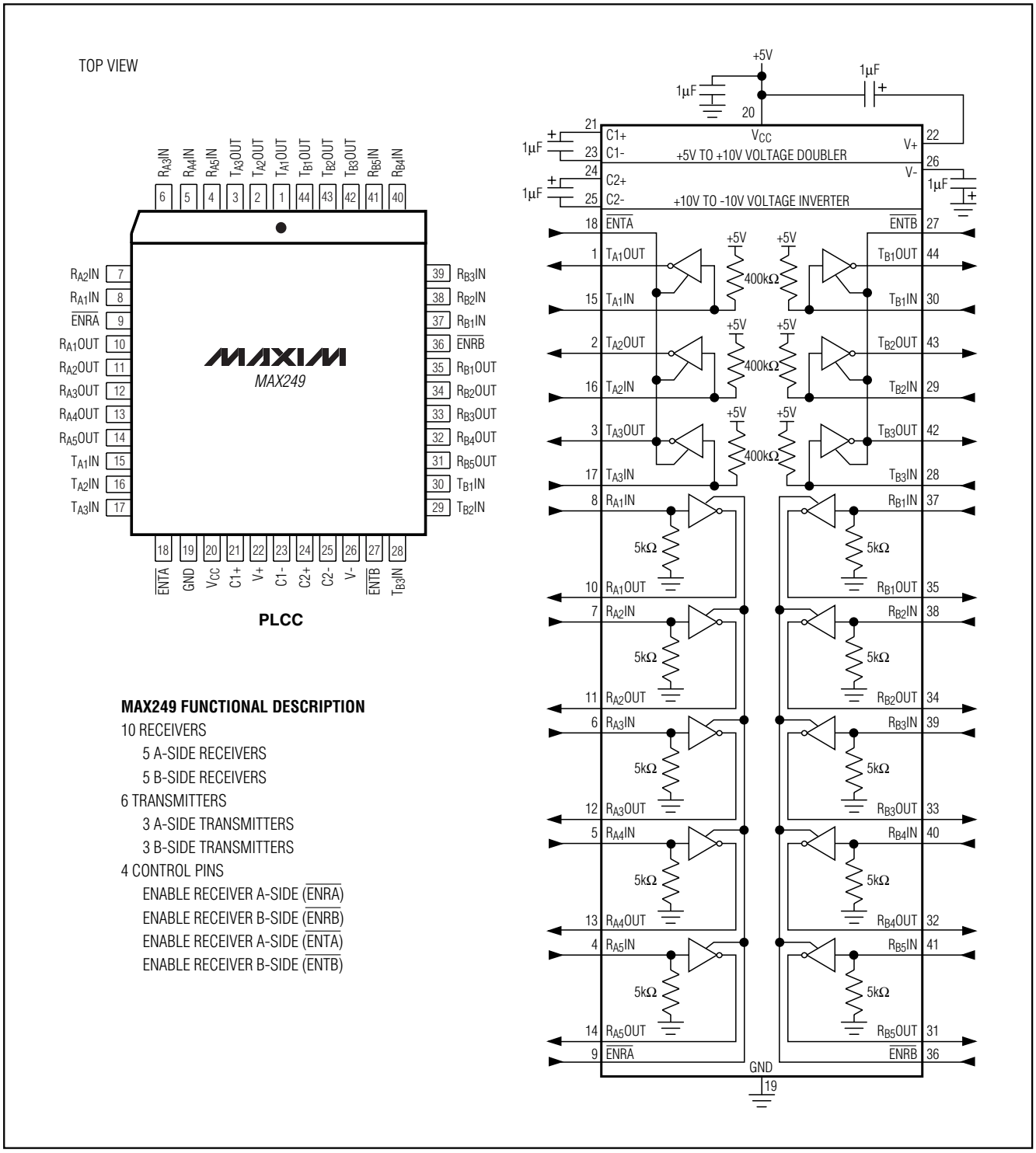
8 RECEIVERS
- 4 A-SIDE RECEIVERS
- 4 B-SIDE RECEIVERS

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENTB}$)

Figure 24. MAX248 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers



MAX249 FUNCTIONAL DESCRIPTION

10 RECEIVERS
- 5 A-SIDE RECEIVERS
- 5 B-SIDE RECEIVERS

6 TRANSMITTERS
- 3 A-SIDE TRANSMITTERS
- 3 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENTB}$)

Figure 25. MAX249 Pin Configuration and Typical Operating Circuit

Ordering Information (continued)

PART	TEMP RANGE	PIN-PACKAGE
MAX222CPN	0°C to +70°C	18 Plastic DIP
MAX222CWN	0°C to +70°C	18 Wide SO
MAX222C/D	0°C to +70°C	Dice*
MAX222EPN	-40°C to +85°C	18 Plastic DIP
MAX222EWN	-40°C to +85°C	18 Wide SO
MAX222EJN	-40°C to +85°C	18 CERDIP
MAX222MJN	-55°C to +125°C	18 CERDIP
MAX223CAI	0°C to +70°C	28 SSOP
MAX223CWI	0°C to +70°C	28 Wide SO
MAX223C/D	0°C to +70°C	Dice*
MAX223EAI	-40°C to +85°C	28 SSOP
MAX223EWI	-40°C to +85°C	28 Wide SO
MAX225CWI	0°C to +70°C	28 Wide SO
MAX225EWI	-40°C to +85°C	28 Wide SO
MAX230CPP	0°C to +70°C	20 Plastic DIP
MAX230CWP	0°C to +70°C	20 Wide SO
MAX230C/D	0°C to +70°C	Dice*
MAX230EPP	-40°C to +85°C	20 Plastic DIP
MAX230EWP	-40°C to +85°C	20 Wide SO
MAX230EJP	-40°C to +85°C	20 CERDIP
MAX230MJP	-55°C to +125°C	20 CERDIP
MAX231CPD	0°C to +70°C	14 Plastic DIP
MAX231CWE	0°C to +70°C	16 Wide SO
MAX231CJD	0°C to +70°C	14 CERDIP
MAX231C/D	0°C to +70°C	Dice*
MAX231EPD	-40°C to +85°C	14 Plastic DIP
MAX231EWE	-40°C to +85°C	16 Wide SO
MAX231EJD	-40°C to +85°C	14 CERDIP
MAX231MJD	-55°C to +125°C	14 CERDIP
MAX232CPE	0°C to +70°C	16 Plastic DIP
MAX232CSE	0°C to +70°C	16 Narrow SO
MAX232CWE	0°C to +70°C	16 Wide SO
MAX232C/D	0°C to +70°C	Dice*
MAX232EPE	-40°C to +85°C	16 Plastic DIP
MAX232ESE	-40°C to +85°C	16 Narrow SO
MAX232EWE	-40°C to +85°C	16 Wide SO
MAX232EJE	-40°C to +85°C	16 CERDIP
MAX232MJE	-55°C to +125°C	16 CERDIP
MAX232MLP	-55°C to +125°C	20 LCC
MAX232ACPE	0°C to +70°C	16 Plastic DIP
MAX232ACSE	0°C to +70°C	16 Narrow SO
MAX232ACWE	0°C to +70°C	16 Wide SO

PART	TEMP RANGE	PIN-PACKAGE
MAX232AC/D	0°C to +70°C	Dice*
MAX232AEPE	-40°C to +85°C	16 Plastic DIP
MAX232AESE	-40°C to +85°C	16 Narrow SO
MAX232AEWE	-40°C to +85°C	16 Wide SO
MAX232AEJE	-40°C to +85°C	16 CERDIP
MAX232AMJE	-55°C to +125°C	16 CERDIP
MAX232AMLP	-55°C to +125°C	20 LCC
MAX233CPP	0°C to +70°C	20 Plastic DIP
MAX233EPP	-40°C to +85°C	20 Plastic DIP
MAX233ACPP	0°C to +70°C	20 Plastic DIP
MAX233ACWP	0°C to +70°C	20 Wide SO
MAX233AEPP	-40°C to +85°C	20 Plastic DIP
MAX233AEWP	-40°C to +85°C	20 Wide SO
MAX234CPE	0°C to +70°C	16 Plastic DIP
MAX234CWE	0°C to +70°C	16 Wide SO
MAX234C/D	0°C to +70°C	Dice*
MAX234EPE	-40°C to +85°C	16 Plastic DIP
MAX234EWE	-40°C to +85°C	16 Wide SO
MAX234EJE	-40°C to +85°C	16 CERDIP
MAX234MJE	-55°C to +125°C	16 CERDIP
MAX235CPG	0°C to +70°C	24 Wide Plastic DIP
MAX235EPG	-40°C to +85°C	24 Wide Plastic DIP
MAX235EDG	-40°C to +85°C	24 Ceramic SB
MAX235MDG	-55°C to +125°C	24 Ceramic SB
MAX236CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX236CWG	0°C to +70°C	24 Wide SO
MAX236C/D	0°C to +70°C	Dice*
MAX236ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX236EWG	-40°C to +85°C	24 Wide SO
MAX236ERG	-40°C to +85°C	24 Narrow CERDIP
MAX236MRG	-55°C to +125°C	24 Narrow CERDIP
MAX237CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX237CWG	0°C to +70°C	24 Wide SO
MAX237C/D	0°C to +70°C	Dice*
MAX237ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX237EWG	-40°C to +85°C	24 Wide SO
MAX237ERG	-40°C to +85°C	24 Narrow CERDIP
MAX237MRG	-55°C to +125°C	24 Narrow CERDIP
MAX238CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX238CWG	0°C to +70°C	24 Wide SO
MAX238C/D	0°C to +70°C	Dice*
MAX238ENG	-40°C to +85°C	24 Narrow Plastic DIP

** Contact factory for dice specifications.*

+5V-Powered, Multichannel RS-232 Drivers/Receivers

Ordering Information (continued)

PART	TEMP RANGE	PIN-PACKAGE
MAX238EWG	-40°C to +85°C	24 Wide SO
MAX238ERG	-40°C to +85°C	24 Narrow CERDIP
MAX238MRG	-55°C to +125°C	24 Narrow CERDIP
MAX239CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX239CWG	0°C to +70°C	24 Wide SO
MAX239C/D	0°C to +70°C	Dice*
MAX239ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX239EWG	-40°C to +85°C	24 Wide SO
MAX239ERG	-40°C to +85°C	24 Narrow CERDIP
MAX239MRG	-55°C to +125°C	24 Narrow CERDIP
MAX240CMH	0°C to +70°C	44 Plastic FP
MAX240C/D	0°C to +70°C	Dice*
MAX241CAI	0°C to +70°C	28 SSOP
MAX241CWI	0°C to +70°C	28 Wide SO
MAX241C/D	0°C to +70°C	Dice*
MAX241EAI	-40°C to +85°C	28 SSOP
MAX241EWI	-40°C to +85°C	28 Wide SO
MAX242CAP	0°C to +70°C	20 SSOP
MAX242CPN	0°C to +70°C	18 Plastic DIP
MAX242CWN	0°C to +70°C	18 Wide SO
MAX242C/D	0°C to +70°C	Dice*
MAX242EPN	-40°C to +85°C	18 Plastic DIP
MAX242EWN	-40°C to +85°C	18 Wide SO
MAX242EJN	-40°C to +85°C	18 CERDIP
MAX242MJN	-55°C to +125°C	18 CERDIP

PART	TEMP RANGE	PIN-PACKAGE
MAX243CPE	0°C to +70°C	16 Plastic DIP
MAX243CSE	0°C to +70°C	16 Narrow SO
MAX243CWE	0°C to +70°C	16 Wide SO
MAX243C/D	0°C to +70°C	Dice*
MAX243EPE	-40°C to +85°C	16 Plastic DIP
MAX243ESE	-40°C to +85°C	16 Narrow SO
MAX243EWE	-40°C to +85°C	16 Wide SO
MAX243EJE	-40°C to +85°C	16 CERDIP
MAX243MJE	-55°C to +125°C	16 CERDIP
MAX244CQH	0°C to +70°C	44 PLCC
MAX244C/D	0°C to +70°C	Dice*
MAX244EQH	-40°C to +85°C	44 PLCC
MAX245CPL	0°C to +70°C	40 Plastic DIP
MAX245C/D	0°C to +70°C	Dice*
MAX245EPL	-40°C to +85°C	40 Plastic DIP
MAX246CPL	0°C to +70°C	40 Plastic DIP
MAX246C/D	0°C to +70°C	Dice*
MAX246EPL	-40°C to +85°C	40 Plastic DIP
MAX247CPL	0°C to +70°C	40 Plastic DIP
MAX247C/D	0°C to +70°C	Dice*
MAX247EPL	-40°C to +85°C	40 Plastic DIP
MAX248CQH	0°C to +70°C	44 PLCC
MAX248C/D	0°C to +70°C	Dice*
MAX248EQH	-40°C to +85°C	44 PLCC
MAX249CQH	0°C to +70°C	44 PLCC
MAX249EQH	-40°C to +85°C	44 PLCC

** Contact factory for dice specifications.*

Package Information

For the latest package outline information, go to **www.maxim-ic.com/packages**.